Empirical evidence that proves a serious game is an educationally effective tool for learning computer programming constructs at the computational thinking level

Cagin Kazimoglu

A dissertation submitted in partial fulfilment of the requirements of the University of Greenwich for the degree of Doctor of Philosophy

June 2013

The University of Greenwich,

School of computing and Mathematical Science,

30 Park Row, Greenwich, SE10, 9LS

DECLARATION

I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

X----------

Cagin Kazimoglu

X----------

Dr. Mary Kiernan

(Supervisor)

X----------

Prof. Liz Bacon

(Supervisor)

X----------

Prof. Lachlan MacKinnon

(Supervisor)

To my beloved friend Bilgin Kanal.

ACKNOWLEDGEMENTS

"Education is the key to unlock the golden door of freedom."

– George Washington Carver

As this thesis is successfully completed, I would like to express my deepest appreciation to all those who provided me support and guidance during my studies.

First of all, I would like to thank my supervisors Dr. Mary Kiernan, Prof. Liz Bacon and Prof. Lachlan Mackinnon at the University of Greenwich for they gave me great support, encouragement and inspiration during my PhD research.

I still remember the very first day I stepped in QM 351 and met Dr. Kiernan and Prof. Bacon for the first time. I even remember the date; it was October 13, 2009. They made an icebreaking comment about my spikey back bag as I believe they had seen in my eyes how I felt insecure and confused – like a scared deer back then.

Dr. Mary Kiernan has always been a friend to me or a colleague perhaps rather than my supervisor. She always gave me friendly and positive advice, encouraged me deeply with indefatigable academic spirit. She thought me that a supervisor – PhD student relationship is a lifelong work partnership. She guided me through my entire journey with invaluable advice and great positive spirit – even at the times when I was feeling desperate and confused. Her friendly attitude, constructive feedback and advice helped me greatly to complete my thesis.

Prof. Liz Bacon is an inspiring figure at the University of Greenwich. She has vast academic knowledge and experience yet she is one of the most down to earth people you can ever meet in your life. In my opinion, her professionalism, modesty and positive attitude towards everyone is a true example of wisdom and how a professor should be. She always responds to her emails timely and she always says "*you are the best and brightest PhD student I've worked with*". I do not know if I am bright or the best; but her invaluable suggestions helped me greatly to come where I am now.

When I met Prof. Laclan Mackinnon for the first time, I was deeply affected by his heavy charisma. He once told me "*you wouldn't get feedback from me that often Cain, but when you do, it will be very detailed and helpful to you*". As he stated, that is exactly what happened in

my entire PhD journey. He did not only help me to find my way when I was lost, but also supported my confidence through constructive feedback and encouragement.

I would like to thank to Dr. Nadarajah Ramesh – a qualified statistician who has considerable experience in data analysis and planning – for his guidance and help on the statistical analysis of this research. Dr. Ramesh provided his support in specifying the necessary statistical tests and his suggestions have been a great help in identifying what statistical methods to use in the analysis of the raw data obtained from the conducted studies.

I would also like to express my very great appreciation to my family especially to my parents and my brothers for their endless love and support. Without them, I could have never completed this thesis. They understood the difficulties of living in London and have given me great financial support as well as in encouragement. "*You can do it Cain, I know you can!*" says my mother. I don't know why, but it always put tears in my eyes.

Finally, thanks to my life time friend *Bilgin Kanal*, for he supported me in this long journey like no other as he has been with me since the very first day I decided to do a PhD. Due to the very stressful and monotonic life of the PhD, I treated him badly from time to time especially when I felt confused or exhausted. I broke his heart, perhaps patronized him due to being stuck between a full time job and a PhD research. Yet, he never said anything – only concerned about my wellbeing. I can never pay back how he supported me in this journey, and therefore, I dedicate this thesis to him.

ABSTRACT

Owing to their easy engagement and motivational nature, games predominantly in young age groups, have been omnipresent in education since ancient times. More recently, computer video games have become widely used, particularly in secondary and tertiary education, as a method of enhancing the understanding of some subject areas (especially in English language education, geography, history and health) and also used as an aid to attracting and retaining students.

Many academics have proposed a number of approaches using video game-based learning (GBL), to impart theoretical and applied knowledge, especially in the Computer Science discipline. Despite several years of considerable effort, the empirical evidence in the GBL literature is still missing, specifically that which identifies what students learn from a serious game regarding programming constructs, and whether or not they acquire additional skills after they have been introduced to a GBL approach. Much of the existing work in this area explores the motivational aspect of video games and does not necessarily focus on what people can learn or which cognitive skills they can acquire that would be beneficial to support their learning in introductory computer programming.

Hence, this research is concerned with the design, and determining the educational effectiveness, of a game model focused on the development of computational thinking (CT) skills through the medium of learning introductory programming constructs. The research is aimed at designing, developing and evaluating a serious game through a series of empirical studies in order to identify whether or not this serious game can be an educationally effective tool for learning computer programming at the CT level.

The game model and its implementation are created to achieve two main purposes. Firstly, to develop a model that would allow students to practise a series of cognitive abilities that characterise CT, regardless of their programming background. Secondly, to support the learning of applied knowledge in introductory programming by demonstrating how a limited number of key introductory computer programming constructs which introductory programming students often find challenging and/or difficult to understand.

In order to measure the impact of the serious game and its underlying game model, a pilot-study and a series of rigorous empirical studies have been designed. The pilot study was conducted as a freeform evaluation to obtain initial feedback on the game's usability. A group of students following Computer Science and related degree programmes with diverse

backgrounds and experience participated in the pilot-study and confirmed that they found the game enjoyable. The feedback obtained also showed that the majority of students believed the game would be beneficial in helping introductory programming students learn computational thinking skills.

Having incorporated the feedback into a revised version of the game, a further series of rigorous studies were conducted, analysed and evaluated. In order to accurately measure the effect of the game, the findings of the studies were statistically analysed using parametric or non-parametric measures depending on the distribution of data gathered. Moreover, the correlations between how well students did in the game, the knowledge gain students felt, and the skills they felt they acquired after their game-play are thoroughly investigated.

It was found that intrinsic motivation, attitude towards learning through game-play, students' perception of their programming knowledge, how well students visualise programming constructs and their problem solving abilities were significantly enhanced after playing the game. The correlations of the studies provided evidence that there is no strong and significant relationship between the progress of students in the game and the computational thinking skills they felt they gained from it. It was concluded that students developed their computational thinking skills regardless of whether or not they reached the higher levels in the game. In addition to this, it was found that there are no strong and significant correlations between the key computer programming constructs and the computational thinking skills, which provides strong evidence that learning how introductory computer programming constructs work and developing computational thinking skills, are not directly connected to each other in the game environment. It was also found that students felt that their conditional logic, algorithmic thinking and simulation abilities had significantly developed after playing the game.

As a result, this research concludes that the designed serious game is an educationally effective tool for a) learning how key introductory computer programming constructs work and b) developing cognitive skills in computational thinking.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

This chapter provides a brief introduction to the problems students have in learning computer programming as well as proposes the use of video games and video game-like environments as one way to potentially overcome these problems. The chapter also states that the empirical evidence which verifies video games are educationally effective tools for learning introductory computer programming is absent from the literature. Having briefly discussed the reasons for this, the chapter reveals the main research question and the objectives of the research along with the methodology and an overview of the thesis.

Section 1.1 describes motivation of the research and section 1.2 reveals the main research question and the research objectives. Section 1.3 discusses the methodology that would be followed to find an answer for the main research question. Section 1.4 lists out the publications related to the thesis and section 1.5 discusses the timeline of the research. Finally, section 1.6 provides a summary and the structure of the thesis.

1.1 Motivation of the research

Computer Science (CS) is now an intrinsic part of our lives and one could argue that a world without computers would be unthinkable. Yet there is a consensus that CS has serious conundrums, particularly in attracting students, low retention rates and low motivation for learning computer programming despite the continuing growth of the IT industry (Beaubouef & Mason, 2005; Kinnunen & Malmi, 2006; Fletcher & Lu, 2009). It is widely accepted that motivation and involvement are imperative in retaining students in CS (Guzdial, 2004; Beaubouef & Mason, 2005) as well as engaging them in learning computer programming by building effective mechanisms for the development of programming skills. However, this is not an easy task, and one of the core aims of learning computer programming should be to constantly highlight that computer programming is not only coding but also about thinking computationally and acquiring cognitive skills to develop effective solutions through understanding of concrete problems.

The widely referenced work of Jeannette Wing (Wing, 2006) highlighted the importance of skill development in programming and defined *computational thinking* (CT) as a problem solving approach concerned with conceptualising, developing abstractions and designing

systems which overlaps with logical thinking and requires fundamental concepts to Computer Science (e.g. abstraction, modelling, algorithmic thinking). Recent studies defend the idea of making CT accessible to everyone and also stress that it is crucial for students to develop skills in CT before they are introduced to formal computer programming (Qualls & Sherrell, 2010; Perkovic *et al.*, 2010).

Further to these, existing research has led to many discussions and ideas on how best to teach introductory computer programming as students suffer from a wide range of difficulties in computer programming courses (Bonar & Soloway, 1983; Lahtinen, Mutka & Jarvinen, 2005; Coull & Duncan, 2011). Numerous studies reported that students view computer programming as a purely technical activity rather than a set of combined problem solving skills (Bennedsen, & Carpersen, 2008; Liu, Cheng & Huang, 2011). Therefore, the majority of students who are learning introductory computer programming tend to develop superficial knowledge and fail to create problem solving strategies through using programming constructs. Recent studies in this field also reported that enrolment in CS programmes has been facing a steady decline despite steps taken to counter this and to bring more students into CS (Ali & Shubra, 2010).

To address this, video games and video game-based tools are proposed as a primary approach for motivating and supporting students in learning introductory programming as well as in developing cognitive skills in computational thinking.

The first of these approaches are interactive syntax-free visual programming environments, such as Scratch (2006), where students often use graphical programming commands to build their programs in order to gain a visual perspective to abstract concepts fundamental to computer programming. In other words, programming is usually done by dragging and dropping blocks from the toolbars onto a stage as forms of scripts which control the behaviour of graphical objects. The use of visual programming environments is often perceived to be ideal because these tools allow students to quickly create solutions without the need for excessive program code.

An increasingly popular approach being followed to support learning introductory programming is via game development classes where although the objective is to design a new game as the product, the rationale is the realisation of basic programming constructs in addition to planning algorithms (Sung, 2009).

The final and certainly the least common way is to facilitate the teaching and learning of introductory computer programming through the use of video game technologies in an educational game context (also referred to as *serious games*) due to several exhibiting features

of games such as learner-centricity, interactivity and immediate feedback. Many studies state that video games are powerful tools for learning purposes as they refer to all groups of people and can provide engagement, personalization and intellectual benefits (i.e. analyse, create, apply and evaluate) (Quinn 2005; Clark, 2009). Additionally, video games are engaging and motivational in nature and it is anticipated that students will be encouraged to learn how programming constructs work in an entertaining and potentially familiar environment, and will then be able to transfer their learning outcomes from that environment into learning introductory computer programming with a programming language. Moreover, curricula that used video games to specialise in learning programming have found positive motivational effects on students (Ater-Kranov *et al.*, 2010).

Despite these efforts, few studies evaluated video games as learning environments specifically in how game-play can be associated to support the education of computer programming (Sung *et al.*, 2011). The empirical evidence that verifies video games are educationally effective tools for learning introductory computer programming is absent from the literature (Hainey *et al.*, 2011; Kazimoglu *et al.*, 2011).

One of the reasons why the evidence is still absent is because the research in this field tends to focus on a drill and practice approach (Graven & MacKinnon, 2008; Yeh, 2009) or are assessments based on early game prototypes that are not available to public play (Barnes *et al.*, 2008; Chang & Chou, 2008; Chaffin *et al.*, 2009). Only a small number of studies review how to learn programming constructs through game-play, without a drill and practice approach, with some statistical analysis (Muratet *et al.*, 2011; Liu, Cheng & Huang, 2011). Moreover, none of the existing work provides sufficient information to be regarded as guidelines that would enable researchers to develop (similar) serious games specifically for learning computer programming constructs and developing computational thinking abilities. The existing work in this field tends to investigate how to adapt and assess serious games in classroom environments rather than proposing concrete methods to improve game-play (Hainey *et al.*, 2011). Therefore, in addition to the missing evidence to prove serious games can be educationally effective tools for learning programming constructs, there is a significant need for clear instructions and analysis on how games can be developed specifically for acquiring problem solving skills to support the education of introductory computer programming.

1.2 Research aim and objectives

To address the above issues, this research investigates the relationship between game-play

and developing skills in computational thinking through learning introductory programming constructs.

The aim of this research is to design a serious game based on the learning experience associated with computational thinking in order to assess whether or not this approach will be supportive to learning how introductory programming constructs work. In other words, this research aims at developing a serious game model for computational thinking and learning (a limited number of) computer programming constructs, which will then be subject to rigorous experimental evaluation and analysis, in order to provide the structured empirical evidence currently missing from the literature.

Hence, the above identified aim has been interpreted into the main research question of this research:

"Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming?"

Reaching the following objectives would create a pathway to answer this main research question:

1) to identify the problems of students with regard to learning introductory computer programming based on the previous work in this area;

2) to investigate the differences and similarities between computational thinking and learning computer programming;

3) to investigate and analyse the current use of serious games to teach programming;

4) to discuss the potential reasons why the statistical evidence regarding serious games and learning is absent from the literature;

5) to design a new game specifically for encouraging users to think computationally and learn how computer programming constructs work;

6) to create an experimental design and conduct a series of rigorous studies to assess whether or not the serious game developed can be an educationally effective approach to support the education of computer programming;

7) to provide a detailed statistical analysis and evaluation of data collected from the structured rigorous studies.

1.3 Methodology of the research

Having identified the research question and the pathway to answer it, it is planned to analyse the literature in game based learning (GBL) and identify serious game models that could be used to develop a serious game specifically for the purpose of learning programming. After analysing the literature, the main research question is revisited and refined. Additionally, the main target group of this research was selected as the first year CS (or a related degree) students. In order to answer the refined main research question, a serious game would be developed specifically for this target group and the impact of the game would be investigated by testing whether or not it is possible to teach a limited number of key programming constructs through game-play. Before moving to the structured empirical stages, the main research question would be divided into several more focused sub research questions that can be investigated individually. These sub research questions would then be investigated through *one group pre – study post – study* experimental design in a series of rigorous structured studies. Finally the data obtained from the studies would be analysed through inferential statics and a conclusion would be drawn by analysing the validity of these statistical findings.

1.4 Publications related to this thesis

The publications related to this thesis are as follows:

1. Kazimoglu, C., Kiernan, M., & Bacon, L. (2010a). Enchanting e-learning through the use of interactive-feedback loop in digital games. 3rd Conference in Human System Interactions (HSI), 502 – 509. IEEE.

2. Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2010b). Developing a game model for computational thinking and learning traditional programming through game-play. In World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Vol. 2010, No. 1, 1378 – 1386.

3. Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2011). Understanding computational thinking before programming: developing guidelines for the design of games to learn introductory programming through game-play. International Journal of Game-Based Learning (IJGBL), 1(3), 30 – 52.

4. Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012a). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. Procedia – Social and Behavioral Sciences, 47, 1991 – 1999.

5. Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012b). Learning Programming at the Computational Thinking Level via Digital Game-Play. Procedia Computer Science, 9, 522 – 531.

6. Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012c). Experimental Evaluation Results of a Game Based Learning Approach for Learning Introductory Programming. In World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Vol. 2012, No. 1, 636 – 647.

1.5 Evolution of the Research

This research has started with reading and analysing the game based learning (GBL) literature in order to learn how to model a serious game to support learning computer programming. Having analysed various video games designed for introductory programming students (e.g. Colobot, Robocode, RoboMind), a survey paper (Kazimoglu, Kiernan & Bacon, 2010a) was generated and published to highlight the fact that all video games are based on the principle of an interaction – feedback loop. This survey paper suggested that the interaction – feedback loop in video games encourages engagement and could be useful to support motivation and learning computer programming. Having established this point, the research focused on investigating computational thinking and recognised the different layers of abstractions in learning computer programming. As a result of this recognition, a second paper (Kazimoglu *et al.*, 2010b) was produced and stated the importance of developing computational thinking abilities through playing video games. This paper also helped to formalise the main research question and the research objectives as they described in Section 1.2. Having completed the background research, the research question was revised to be more clearly focused on the objectives, and a game prototype was undertaken to be developed as a

research vehicle. At this point, a third paper (Kazimoglu *et al.*, 2011) was published in order to introduce the underlying structure of the developing game and the guidelines followed in designing it. During the development stages of the game, the methodology of the research was planned and the experimental design of two rigorous studies was completed. At this point, a free form of evaluation on the game prototype was carried out as a pilot study, and the feedback obtained from this evaluation is published (Kazimoglu *et al*., 2012a; Kazimoglu *et al*., 2012b). It was found that the game reached to the empirical stages where it can be assessed through a structured study and therefore, two rigorous studies were designed to observe the educational effectiveness of the game on the introductory programming students. While the main target group of the studies was selected as the introductory programming students, the research was extended to an additional study in a public girls school in the anticipation to provide some potential support for the results being produced from the other two rigorous studies. After conducting all rigorous studies, part of the findings from one of the main studies were analysed and published (Kazimoglu *et al*., 2012c).

This thesis presents all the data gathered from these studies, how it is analysed, validated and what was found. The results are predominantly focused on the two main rigorous studies and the analytics provided evidence for the contributions described in the conclusions (Chapter 8).

1.6 Summary and structure of the thesis

Chapter 2 – Background Research

This chapter defines the concept of computational thinking and the cognitive abilities that characterise it. The chapter draws together the differences between computational thinking and learning traditional computer programming, and discusses the reasons why there is a dearth of evidence in the literature to support serious games as educationally effective tools for learning how programming constructs work. The chapter also outlines widely referenced serious game models in the literature and discusses their advantages and disadvantages. Finally, the chapter proposes a set of guidelines collected from various resources in the literature in order to support the development of serious games that are designed to foster computational thinking skills and learning introductory programming constructs. The first and the second publications in section 1.4 are explicitly related to this chapter.

Chapter 3 – Developing a Research Testbed

Having analysed the literature, this chapter revisits the main research question and refines it into a structure suitable to be explored with a methodology. The chapter then discusses the current serious game models lack of focus on deep game-play for learning computer programming constructs particularly in relation to skill development in computational thinking. The chapter then demonstrates the design of a new game model based on the body of existing research in serious game modelling and describes a serious game that is built through the proposed guidelines and within the structure of the designed game model. The third publication in section 1.4 is explicitly related to this chapter.

Chapter 4 – Research Methodology

This chapter first outlines the approach followed in this research and then divides the main research question into several different sub research questions so that these could be investigated individually. The chapter then reveals that a *one group pre-study post-study experimental design* was used as the main methodology for collecting data in the structured studies and discusses the reasons why this structure is used. The pre-study and the post-study questionnaires are described and the rationale behind each question asked in the questionnaires is explained.

Chapter 5 – Experimental Design

This chapter discusses the experimental design of the rigorous studies as well as the structure of a pilot study which was conducted as a free form evaluation before these structured studies. The chapter then investigates the ethical issues, experimental variables and the hypotheses generated from the research questions. Finally, the chapter describes potential threats that could bias the outcomes of the studies.

Chapter 6 –Analysis of Experimental Studies

The feedback obtained in the pilot study; the raw data captured from the structured rigorous studies and the statistical analysis of this are explored in this chapter. The chapter first shows the feedback obtained from participants in the pilot study and lists out the modifications that were made on the game before moving to the empirical stages. The chapter then analyses the outcomes of the conducted structured studies in accordance with research questions. The

distribution of data obtained is analysed in detail and all structured studies are investigated in the same way. The results obtained from one study were matched with the results obtained in another study wherever this was possible. Fourth, fifth and the sixth publications in section 1.4 are related to this chapter.

Chapter 7 – Experimental Validation

This chapter investigates the validity of findings obtained in the structured studies and also explores whether or not any confounding variable impacted on the statistical findings before the outcomes of the studies are generalised.

Chapter 8 – Conclusion and Future Work

This chapter draws conclusions and outlines possible direction for future work. A research summary and the limitations of the research findings are also listed in this chapter.

CHAPTER 2
BACKGROUND RESEARCH

The first chapter briefly outlined the problems students have in learning computer programming and provided an introduction to game based learning (GBL) approaches to address these problems. This chapter investigates the reasons why students find learning programming difficult and discusses the importance of computational thinking in more depth.

The chapter further states that the current approaches in GBL do not necessarily consider a deep game-play for learning programming particularly in relation to applied knowledge and skill development. The chapter also highlights that there is a dearth of evidence in the literature to support serious games as an educationally effective approach for learning how programming constructs at the computational thinking level. Finally, the chapter discusses the current game models widely referenced in the literature and proposes guidelines for developing games specifically for learning computer programming based on the body of existing research work in this area.

Section 2.1 reports various difficulties students have in learning traditional computer programming by investigating related research in the literature. Section 2.2 defines what computational thinking is and which cognitive skills and abilities characterise thinking computationally. Additionally, this section provides a detailed discussion on how computational thinking is different from and similar to learning introductory computer programming. Section 2.3 critically reviews current literature in GBL regarding learning computer programming constructs and computational thinking. This section also argues that there is a lack of rigorous empirical evidence to prove that a serious game can be an educationally effective tool for learning programming constructs. Section 2.4 outlines widely referenced serious game models in the literature and reflects on their advantages and disadvantages. Finally, section 2.5 puts forward proposed guidelines derived from the literature in order to develop games specifically for learning programming through game-play.

2.1 Problems of students with regard to learning computer programming

This section discusses various difficulties students have with learning computer programming by analysing seminal work from the literature and then provides a list regarding students' problems when learning introductory computer programming.

The difficulty of learning computer programming is cited as a potential reason for the high attrition rates within the Computer Science (CS) discipline. Numerous studies state that students show a lack of engagement and low motivation in facing the challenges of computer programming (Beaubouef & Mason, 2005; Kinnunen & Malmi, 2006; Fletcher & Lu, 2009). Additionally, the task of learning to program is often recognised as a frustrating and demanding activity by introductory programming students (Bennedsen, Caspersen & Kölling, 2008). Recent studies in this field argue that poor teaching methods, low levels of interaction with students and a lack of interest are the major problems in learning programming (Barker, McDowell & Kalahar, 2009; Coull & Duncan, 2011). Guzdial (2004) refer to this issue by stating "Students want to work on computational artifacts that have meaning for them, e.g., that are interesting and relevant". Previous studies also argue that there might be a link between dropout rates and low motivation for learning computer programming, since often the mechanisms for learning computer programming are seen by students as neither interesting nor relevant (Wilson, 2002; Beaubouef & Mason, 2005).

Students' low motivation and difficulties in learning computer programming are not recent problems reported in the education of Computer Science. In his seminal work, Soloway (1986) clearly stated that the real problems introductory programming students have lie in "*putting the pieces together*" especially in recalling domain specific plans in order to encode the pieces of information into meaningful units. Soloway (1986) also argued that when a program (such as, read 3 integers and output their average) is given as a problem, experts tend to recall meaningful Computer Science concepts as soon as they understand the problem whereas introductory programming students often have a lack of concrete realisation and cannot develop a programming algorithm (even with the programming knowledge). In other words, students lack the skill of developing abstractions and the ability to visualise Computer Science concepts from given problems (McCracken *et al.*, 2001).

Several studies have investigated the reasons why students find computer programming difficult (Gomes & Mendes, 2007; Hawi, 2010; Coull & Duncan, 2011) and presented strong anecdotal evidence that the most valid reason lies within the nature of computer programming. Learning to program requires comprehending abstract concepts about Computer Science and arranging these concepts in a rational order in order to solve real life problems successfully. However, the majority of introductory programming students perceive computer programming as a purely technical activity rather than a series of combined cognitive skills (Bennedsen & Caspersen, 2007). Moreover, students often find the process of learning computer programming difficult because they need to find a solution to a problem by acquiring a new way of thinking in addition to the need to practise a new syntax and grammar in order to communicate their solution to a real life problem (Dalal *et al.*, 2009). Many students are not conscious of this and, despite their training, when they undertake a computing project the reaction of the majority of them is to start coding immediately, skipping the crucial steps of analysis and design and the need to develop abstractions and algorithmic thinking (Rajaravivarma, 2005). Thus, learning computer programming becomes a demanding task and requires abstraction of Computer Science concepts to describe a problem and propose a solution, followed by the need to design and code in order to convert the solution into the syntax of a programming language.

Despite the majority of the work in the literature providing anecdotal evidence; some studies highlight the difficulty of introductory programming courses through experimental evaluation (Alvarez & Scott, 2010). As an example, Bennedsen & Caspersen (2007) ran a survey study among institutions all around the world and reported the failure rates in introductory programming courses. Although participation in their survey was low (80 respondents), it was found that 33% of the students in introductory programming courses are failing and that only 27% of students who are enrolled in Computer Science programmes are graduating on time. Guzdial (2012) also emphasises that failure rates worldwide of 30-50% in the introductory programming courses have been reported for decades. Furthermore, recent research in this field states that even students who have completed introductory programming courses still don't know how to program and/or may not have the ability to use programming codes to solve problems within the Computer Science discipline (Loftus, Thomas & Zander, 2011; Chang *et al.*, 2012).

Previous work states that researchers are aware of the problems that students have in learning to program but there is a lack of knowledge in what causes those problems and how to overcome them (Guzdial, 2004). Boyle, Carter and Clark (2002) concluded that neither prior mathematical knowledge nor computer qualifications had an explicit relationship with students' success in computer programming. Ventura (2005) supported the findings of Boyle, Carter and Clark (2002) and reported that students who are good in mathematical sciences or had some familiarity with computer programming did not always succeed in this area. Lister *et al.* (2004) undertook experimental research involving experts and introductory programming students on their understanding of computer programming. Their findings suggested that experts have a relational level of understanding of programming meaning and that they can envisage and express a syntactic relation between programming concepts and the overall purpose of the program. It is observed that this level of understanding is something that is lacking in most of the introductory programming students. Further to this, de Raadt (2007) reported that majority of introductory programming students tend to give a line by line explanation of the source code rather than describing the overall purpose behind the piece of code. More recently, Bennedsen & Caspersen (2012) conducted an experiment on students who have completed an introductory programming course 3, 15 or 27 months prior to their experiment. The aim of the experiment was to find out the ability of students in recalling programming competences. The study found that syntax and semantic issues in programming blurs when recalling computer programming abilities.

As a result, although there is a lack of rigorous empirical evidence, the problems of students with regard to learning computer programming gathered from several different studies are listed below:

1) Introductory computer programming students tend to:

a) get confused with multiple levels of branching and locations in the programming logic. They gain a good level of understanding when programming segments are explained separately and combined as a whole again (Ali, 2009);

b) have difficulties in composing and coordinating the components of a programs. They are especially confused in managing and understanding error messages in order to debug correctly (Nienaltowski, Pedroni & Meyer, 2008);

c) have difficulties in visualising programming constructs from given problems (McCracken *et al.*, 2001);

d) not have a relational level of understanding of computer programming. (Lister *et al.* 2004);

e) require timely, effective and well-structured feedback in response to their actions (Beaubouef & Mason, 2005).

2) Thinking within the syntax of a programming language is not "*natural*" to many introductory programming students and this creates a problem when they need to learn how to program (Guzdial, 2008). Students need to operate at an operational level of abstraction before producing code in a specific programming language so that they can develop their abilities in solving problems before they start programming (Fletcher & Lu, 2009).

3) The challenges of learning computer programming result in many introductory programming students labelling programming as "too hard". The retention and the positive attitude of students to computer programming, play a key role in learning to program (Kinnunen & Malmi, 2006).

2.2 Computational thinking

This section defines computational thinking as a problem solving approach and puts forward a discussion on how introductory programming students can benefit by developing abilities in computational thinking. The section then categorise skills that encompass computational thinking by analysing various work from the literature. Finally, the section discusses the importance of abstraction ability in computational thinking as well as how thinking computationally is dissimilar to learning introductory programming constructs.

2.2.1 Skills that encompass computational thinking

The problems introductory programming students have in learning programming is relatively connected to the practice of identifying multiple levels of abstractions often referred to as *computational thinking* in the literature (Gomes & Mendes, 2007; McAllister & Alexander, 2008). The concept was first used by Papert (1996) and later deeply investigated

by Wing (2006). In her seminal work, Wing (2006) describes Computational Thinking (CT) as a problem solving approach that combines logical thinking with Computer Science (CS) programming constructs, and that it can be used to solve a problem in any discipline regardless of where the problem lies. In other words, CT is described as a set of intellectual and reasoning skills that state how people interact and learn to think through the language of computation that involves using methods, language and systems of CS. This does not mean that CT proposes problems that need to be solved in the same way a computer tackles them, but rather it encourages the use of critical thinking using concepts fundamental to the CS discipline.

It is widely accepted that introductory programming students need to demonstrate an understanding of the patterns evident in programming rather than focusing only on the syntax and semantics of computer programming (Liu, Cheng & Huang, 2011). To achieve this, CT has been the focal point of recent studies especially within the CS discipline in order to integrate it into the basic curriculum (Qualls & Sherrell, 2010; Perkovic *et al.*, 2010). Despite this, CT is a vaguely defined concept and a clear definition is necessary in order to use this construct to gain insight into problems (Guzdial, 2008; Denning, 2009). Various studies attempted to construct a clear definition for CT, but this resulted in CT having several independent descriptions in the literature rather than a universally agreed clear definition (Perkovic *et al.*, 2010). Moreover, very little of this work has successfully delivered guidance on what cognitive skills demonstrate CT and how these skills can be taught (Sung *et al.*, 2011). In other words, which specific skills comprise CT and how to scaffold these are still controversial, because few studies have empirically evaluated CT (Ater-Kranov *et al.*, 2010).

Wing (2006) identified five core aspects of CT which are conditional logic, distributed processing, debugging, simulation and algorithm building. She argued that CT incorporates all critical skills that involve problem solving with mathematical and engineering thinking and also with systematic and logical thinking. Despite her seminal work, Guzdial (2008) reported that this definition of CT is very abstract and academics need to understand CT better in order to apply it into a curriculum. Denning (2009) supported this idea and argued that CT should not be seen as what CS is all about or a way to decrease the high dropout rates and poor retention of students in CS. He further explains the principles of computing in seven categories referring each of these as particular perspectives or classifications to view CT. These are computing, coordination, communication, recollection, automation, design and evaluation. Perkovic *et al.* (2010) discuss various skills (i.e. executing algorithms, coordination, communication and experimental analysis) according to the fundamental

principles of computing stated by Dennings (2009). Ater-Kranov *et al.* (2010) investigated the magnitude of importance of skills and abilities characterising CT by evaluating the perspective of academics and students. They compared their perspectives and concluded that critical and algorithmic thinking alongside the application of abstractions to solve problems are the top skills that encompass CT. Furthermore, their findings propose that mathematical and engineering thinking is not necessarily a main characteristic of CT because complex CT can also happen spontaneously. In other words, their findings disproved part of the original definition proposed by Wing (2006) as their evidence showed that CT does not incorporate all mathematical and engineering skills.

Recently, Dierbach *et al.* (2011) defined the most common set of CT skills as: identifying and applying problem decomposition, evaluating, building algorithms and developing computation models to problems. Berland & Lee (2011) summarized the categories of CT according to computational activities as they are described in the literature: conditional logic, algorithm building, debugging, simulation and distributed computation. Lee *et al.* (2011) undertook a similar comparable research and examined CT in three aspects: analysis, abstractions and automation. Further to these, Guzdial (2011) argued the research in this field should move away from trying to define what is or what isn't CT and instead focus on the implications of currently identified cognitive skills. Guzdial (2011) further discusses how these skills and abilities can be taught and what ways can be used to measure them.

As can be seen from above, there are various definitions and there is a lack of empirical evidence in defining the explicit boundaries of CT. However, from the analysis of the above listed studies, it can be argued that: "conditional logic", "algorithmic thinking", "debugging", "simulation" and "socialising" are the core five skills that characterise CT within CS discipline.

Four out of five skills are mentioned above (i.e. conditional logic, algorithmic thinking, debugging and simulation) are explicitly mentioned in previous work in computational thinking (Kowalski, 2011; Barr, Harrison & Conery, 2011; Berland & Lee, 2011; Brennan & Resnick, 2012; Basu et al., 2013). However, the socialising aspect of CT is either investigated under distributed computation or distributed processing (Wing, 2006; Berland & Lee, 2011). Both distributed processing and computation involve interaction and communication with other parties to solve a common computational problem. Additionally, Berland & Lee (2011) discuss that distributed computation are considerations and strategy formations that involve multiple parties with different knowledge resources. Hence rather than dividing this concept under different titles, it is simply named as "socialising" in order to refer brainstorming,

cooperation and coordinator towards solving a common problem.

Conditional Logic in CT refers to solving problems with logical thinking through using various computational models. This includes applying problem decomposition to identify problems and/or generating alternative representations of them (Berland & Lee, 2011; Kowalski, 2011). At this level students distinguish between problems and decide whether these problems can or cannot be solved computationally. Furthermore, students are able to evaluate a problem and specify appropriate criteria in order to develop applicable abstractions.

Building algorithms involves the construction of step-by-step procedures for solving a particular problem. Selection of appropriate algorithmic techniques is a crucial part of thinking computationally as this develops abstractions robust enough that they can be reused to solve similar problems (Barr, Harrison & Conery, 2011).

Debugging is analysing problems and errors in logic or in activities. At this stage, students receive feedback on their algorithms and evaluate them accordingly, which also includes reviewing current rules and/or strategies used. Debugging is central to both programming and CT because it involves critical and procedural thinking (Berland & Lee, 2011; Brennan & Resnick, 2012).

Simulation, also called "model building", is the demonstration of algorithms and involves designing and implementing models on the computer, based on the built algorithm(s). In simulation, students design or run models as test beds to make decision about which circumstances to consider when completing their abstraction (Basu *et al.*, 2013).

Socialising refers to the social aspect of CT, which involves coordination, cooperation and/or competition during the stages of problem solving, algorithm building, debugging and simulation. This characteristic of CT allows brainstorming and encourages assessment of incidents as well as strategy development among multiple parties.

2.2.2 Difference between computational thinking and learning introductory programming

Many authors draw the attention to the fact that CT is not a synonym for computer programming (Wing, 2006; Guzdial, 2008; Repenning, Webb & Ioannidou, 2010). However, a prior study revealed that the majority of academics believe that CT is identical to computer programming (Blum & Cortina, 2007). Hence, at this point it is crucial to differentiate computational thinking from computer programming.

One of the core abilities that involve computational thinking is the process of making

abstractions (Sprague & Schahczenski, 2002; Or-Bach & Lavy, 2004). By means of abstraction, Wing (2008) refers to the process where data and various CS concepts (i.e. computer programming constructs) are presented in a similar way to its meanings (as semantics) without clarifying any implementation details. In other words, abstraction refers to the process of generalisation in order to identify the common core of a definition as well as the act of hiding details that are not really necessary to understand how to solve the problem (Kramer, 2007). As an example, the development of programming languages in CS is a general process of abstraction as the development starts from machine language to assembly and then to the high-level languages. Each step hides the details regarding the previous step and can be used as a milestone for the next step. A better example of computational thinking is the contribution of Harry Beck to the renowned London Underground Map (Kramer, 2007). Beck (1931) abstracted the conventional geographic map of London to an intellectual level by hiding unnecessary details and only showed of the train lines as well as the River Thames. He managed to construct an abstract schematic representation by simplifying the curves on the map to horizontal and/or vertical diagonal lines so that the distances between the stations were no longer related to geographical distances. Despite this, the underground map might be confusing if it one attempts to use it as an actual map because like any abstraction the value of the map depends on its actual purpose and can be misleading if used for other purposes. The London Underground map is a perfect example to demonstrate how computational thinking is applied as it is all about the utilisation of abstraction to perform problem solving, conceptualisation, modelling and analysis.

Wing (2010) argues that CT is thinking at the multiple levels of abstraction and it is different from learning traditional computer programming in three different dimensions: a) choosing the right abstraction; b) operating at the operational level of abstraction; c) defining relationship between the layers of abstraction.

Choosing the right abstraction is related to the removal of unnecessary details from an environment as well as identifying critical aspects of the environment in order to address the problem at hand. This requires avoiding unnecessary constraints and analysing the situation critically to create a solution model. Choosing the right abstraction is a core competence in CT as thinking computationally requires the use of symbolic representations or semantics of CS concepts to solve problems. In contrast to this, computer programming emphases solving problems in the same way a computer tackles them. Therefore, abstraction is not the main aim but only taught indirectly in computer programming. Choosing the right abstraction is equally important in computational thinking and computer programming. However, it is crucial to

highlight that computational thinking is the ability to develop high-level conceptual design skills and it is not unique to CS whereas computer programming is specific to the CS discipline. Moreover, students with computational thinking abilities can find computer programming much easier than others because learning computer programming requires the ability to choose the right symbolic and numerical data to produce generic solutions. Therefore, computational thinking is related to conceptualising and modelling solutions whereas computer programming is related to the context rather than concept of a solution and consequently, stands at a more technical level than computational thinking.

Computational thinking operates at the *operational level of abstraction* and the purpose is to produce step-wise refinement approaches whereas computer programming operates at a *procedural level of abstraction* and the main aim is to produce programming code in a specific programming language to solve a problem. The difficulty for students already struggling with abstraction at one level is to be able to distinguish between operational and procedural, how these relate to each other, and how they can be used to help them to develop their programming skills.

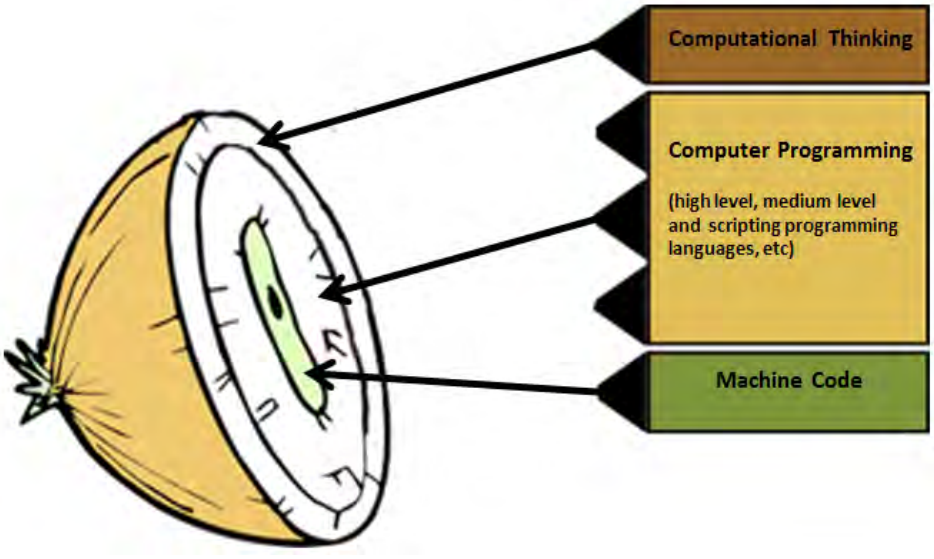

Figure 2.1 – Showing layers of abstraction using Brézillon *et al.* (1997)*'s* onion metaphor.

To explain how/where computational thinking and computer programming operate, the levels of abstraction resemble the layers of an onion – a metaphor frequently used in the cognitive science community (Brézillon *et al.*, 1997; Brézillon, 2003). As shown in Figure 2.1, the executable machine code can be regarded as the heart or innermost layer of the onion (the physical layer). The utilisation of abstraction progresses through the layer of computer programming (the procedural layer) all the way up to the outermost layer which is the CT

layer (the conceptual or operational layer). In moving from the innermost layer to the outermost layer, the patterns become more intellectual, transformed from an informal and complicated real world to a simplified abstract model that is easier to understand. At the computational thinking layer, students design step-wise refinements and functional aggregations to achieve algorithmic improvements whereas at the computer programming layer students contextualise their learning to programming code in a specific programming language to solve the problems at hand.

Finally, *defining relationships between the layers of abstraction* is also an essential part of CT whereas this is not the aim of computer programming. As CT is based on the simplification of reality, its main purpose is to promote understanding and reasoning. To achieve this, students need to exercise all their abstraction skills to constructs models that fit for a specific purpose (in this case to computer programming). In his seminal work, Devlin (2003) clearly reported that "once you realise that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner". Additionally, Kramer (2007) stated that "students should be capable of mapping between reality and abstraction, so that they can appreciate the limitation of abstraction to interpret the implications of model analysis". Mapping between reality and abstraction is a core competence in CT as thinking computationally allows students to develop their knowledge and skills to an extent that they can understand how the layers of abstraction relate to each other, and thus they can develop their introductory programming skills and their CT capabilities (Wing, 2010).

2.3 Game based learning and learning introductory programming constructs

This section discusses the relationship between computational thinking (CT), learning computer programming and game based learning (GBL) approaches. Various studies designed and used GBL approaches to teach CT skills as well as introductory programming constructs to students. An analysis of most widely referenced studies from the literature is discussed in detail in this section. The section also classifies GBL approaches used in the education of introductory programming and discusses why serious game-play has been selected as the approach in this research.

2.3.1 Games as a motivational approach to learning computer programming and the missing evidence in the literature

Wing (2008) argues; "If computational thinking (CT) is added to the repertoire of thinking abilities, then how and when should people learn this kind of concept and how we are going to teach it?" A variety of work has been done recently (Fletcher & Lu, 2009; Orr, 2009; Qualls & Sherrell, 2010; Repenning *et al.*, 2010) to answer these questions by using tools and techniques to reinforce the concepts of Computer Science (CS) and computer programming along with different teaching styles to make CT accessible to students.

Several educational studies proposed computer games (henceforth referred to as games), game-like environments and game programming modules as ways to attract students into computer programming activities (Rajaravivarma, 2005; Robertson & Howells, 2008) and as methods to teach the fundamental concepts of CS (Repenning *et al.*, 2010; Liu, Cheng & Huang, 2011). These approaches are all categorised under game based learning (GBL) in the current literature as games and game-like environments can promote motivation in learning specific content (Garris *et al.*, 2002) and have the potential to allow students to gain abstract programming knowledge in addition to CT skills (Weller, Do & Gross, 2008). Therefore, students would be able to transfer the knowledge and skills acquired from these environments to other problems they encounter when improving their programming skills (Kumar & Sharwood, 2007). Findings from a recent study support the previous work and indicate that most students (81%) have positive attitudes and feel more motivated to learn how computer programming constructs work using a game based model compared to traditional approaches (Ibrahim *et al.*, 2010).

Despite the positive attitudes of students, there is a lack of evidence on whether or not games can engage students in ways of thinking, particularly in CT, that can support them in learning programming logic and prepare them for advanced programming activities (Denner, Werner & Ortiz, 2012). In a recent survey paper, Hainey *et al.* (2011) stated that the empirical evidence in the GBL literature is still missing predominantly in the fields of software engineering, information systems and CS. Early studies in this field demonstrated enthusiasm for games and put forward some evidence that games can enhance motivation to learn computer programming (Kafai, 1995). Despite considerable effort spent over the past few years, to-date, there is a dearth of evidence on what students learn from games regarding programming constructs and whether or not they acquire CT skills after they have been introduced to a GBL approach (Hainey et al., 2011; Denner, Werner & Ortiz, 2012).

The idea of what a game is or how games should be designed can change from one person to another, and this can be argued as a key potential reason why there is a dearth of evidence in games and learning introductory programming. Schell (2008) clearly explained that the definition of a game is perceptual as not all games have the same game characteristics (e.g. fantasy, conflict, outcomes) (Prensky, 2001; Garris *et al.*, 2002). However, he highlights that one game characteristic exists which seems to apply to all games and that is *problem solving*. He suggests that games can be regarded as "a problem solving activity, approached with a playful attitude". Thinking about problem solving, it is known that every game has conflict and players need to solve problems, even sometimes the hidden ones, which emerge as part of the game-play, and need to be solved in order to succeed in the game. Schell (2008) also suggests that a game should generate new problems and offer alternative solutions as this is part of problem solving and the key to retaining player engagement, as they keep coming back to the game environment. Although some of the previous work in this field succeeded in creating enthusiasm for learning computer programming (Graven & MacKinnon, 2008; Eagle & Barnes, 2009; Papastergiou, 2009; Yeh, 2009), it is arguable whether or not they successfully built a constructive problem solving environment where students with little or no programming background can develop skills in CT. Moreover, it is not clear what students learned (or can learn) from these previous studies as very little work provided clear statistical analysis regarding games, or learning how computer programming constructs work (Hainey *et al.*, 2011; Denner, Werner & Ortiz, 2012). Many of the studies provide either anecdotal evidence or initial evaluation results and only a few studies developed games and evaluated them as learning environments using a structured experimental design (Chaffin *et al.*, 2009; Liu, Cheng & Huang, 2011). Therefore, the empirical evidence that verifies games are educationally effective tools for learning how introductory computer programming constructs work is still controversial (Costandi, 2011).

2.3.2 Games and learning how computer programming constructs work

According to the literature available in this area, games and game-based technologies used in the education of introductory programming courses can be classified into three main categories (Sung *et al.*, 2011). These are *individual game development modules* (Sung *et al.*, 2008; Sung *et al.*, 2011), *extensive game development assignments* (Long, 2007) and *learning through game-play* (Liu, Cheng & Huang, 2011; Muratet *et al.*, 2011). All three approaches reported success with a radical increase in students' motivation to learn programming (Leutenegger & Edgington, 2007; Muratet *et al.*, 2011; Sung *et al.*, 2011), hence this provided some evidence that integrating games into the education of introductory computer programming is a promising strategy.

The first of these approaches is the *individual game development modules* which allow students to specifically study a technical aspect or an issue in building games while learning how computer programming constructs work. The aim is not to build an end product from scratch but rather modify a part of one in order to focus on a specific learning outcome (such as loops, event handlers, decision making). Learning outcomes are usually limited with each assignment and can be varied greatly according to the game module. There is consistency in learning with this type of approach as often a game designed by a student is quite similar to a game designed by another student (Sung *et al.*, 2008).

The second approach is the *extensive game development assignment* which aims to develop new games or linear scenarios as an end product. This approach can cover an entire curricula based on custom libraries, different game engines, visual programming tools or new programming languages. Therefore, students can learn introductory programming constructs in an engaging environment alongside developing the fundamental skills necessary to be a computer programmer (such as problem solving and team-working abilities). However, students also need to consider all aspects of producing an end product including, but not limited to, game graphics, sound, game play, physics and narratives. This can sometimes be overwhelming for introductory programming students and therefore, the approach requires game development experience. Chang *et al.* (2012) argues that most of the existing work in this area relies on people who have expertise in game development and programming whereas instructors teaching introductory programming courses are not necessarily well-versed in these concepts and principles.

In addition to these, students might need to learn game programming concepts (such as X and Y axis, gravity and collusion in games) which are not necessarily related to learning

introductory programming constructs. To avoid these difficulties, many studies rely on visual programming tools, such as Scratch (2006) and Alice (2000), simply because these tools allow students to create visual abstractions quickly without the need to write excessive programming code or have a background in games programming (Anewalt, 2008; Maloney *et al.*, 2008). Complex scenarios can be created in these environments by combining character behaviours which inevitably requires an understanding of how to program sequence, conditionals, iteration and objects. Furthermore, visual programming tools remove the syntax rules of genuine programming languages and present programmatic representations as blocks through a simple drag and drop interface. This cleverly separates the programming logic from programming grammar and syntax, allowing students to focus on developing programming strategies with little or no programming background. Despite all these positive traits, research in this field points out that visual programming environments are merely tools and without well-organised teaching methods and learning materials to support them, all they can provide is a "short burst of enthusiasm" (Repenning *et al.*, 2010). A recent study identified that visual programming environments influence not only the learning of introductory programming but also the habits of programing that students develop during their learning process (Meerbaum-Salant *et al.*, 2011). According to this research, when students are asked to perform a programming task they do not approach it by thinking at the algorithmic level but instead, they attempt to solve the problem by using all the blocks that seemed to be relevant for solving the task and randomly combine these blocks into a script in order to try to solve the problem. Additionally, it has been observed that students tend to produce unstructured programming solutions through using various blocks (such as with a repeat-until loop) where the body of blocks are logically coherent and easy to understand, but the outcomes produced by students are no longer coherent and well-organised. One could argue that this is not related to the characteristics of visual programming environments but might be the poor software development skills and weak programming abilities of students. However, scripts and graphical objects are often executed concurrently in visual programming environments. As the scripts are written in the graphical objects, it is difficult for students to develop the skills necessary for building logically coherent solutions as the execution of objects always happen simultaneously. Meerbaum-Salant *et al.*(2011) argue that concurrent programming exists as an integral part of the visual programming environments and although debugging concurrent programs can be seen as a viable concept to support learning, students' tendency to develop unstructured programming solutions (such as an incorrect use of a loop construct), leads to outcomes that contain lots of repetitions in different scripts which are practically impossible

to debug and maintain by introductory programming students. Moreover, there is no mechanism in these environments that might alert students to their mistakes or to the correct use of programming blocks (Meerbaum-Salant *et al.*, 2011).

At this point, it is important to highlight that the intention here is not to alienate visual programming environments from learning introductory programming or to blame these environments in any way, but rather to emphasise that these environments are simply design tools which do not necessarily consider good programming practices as this was not their purpose. They simply lack a mechanism to support students in their quest to understand fundamental ideas in Computer Science (CS) such as to algorithmic thinking, debugging programs and the correct use of programming constructs. Although visual programming environments are very valuable tools and can generate well-structured programs with hundreds of concurrent scripts, one must avoid thinking of these environments as a substitute for pedagogy in learning introductory programming because their characteristics might allow students to incorrectly use programming constructs. More importantly, a student can transfer their bad habits in programming gained from these environments into their further studies in CS. Recently, Lister (2011) suggested that although visual programming environments remove programming syntax problems when learning introductory programming, the need to write algorithms before programming remains essentially a cognitively demanding task. He further indicates that students who have used these environments are still having problems and that the true fault lies with the absence of a pedagogical rethink of what should happen before and after the use of these tools.

The final approach is *learning through game-play* where students can learn specific content or gain skills by playing games. This approach is often referred to as *Serious Games* in the literature (Bergeron, 2006). Michael & Chen (2005) define serious games as "a game in which education (in its various forms) is the primary goal, rather than entertainment". Charsky (2010) slightly expands this definition by referring to serious games as a combination of instructional and video game elements that aim to provide relevant learning experiences rather than focussing on entertainment. Charsky (2010) also reports that serious games should not be confused with 1990s *edutainment* approaches as edutainment provided one of the lowest forms of education (i.e. drill and practice) with less than entertaining game-play whereas serious games facilitate learning higher order thinking skills (i.e. analysing, modelling, testing, evaluating) and does not exclusively use drill and practice activities.

This research is solely focuses on this third approach* mainly for the following three reasons: a) serious games is a conventional way to develop computational thinking skills

* From here on the term *learning through game-play* and *serious games* used interchangeably to refer the same concept.

because all games are fundamentally abstractions from real or fantasy situations; b) when players play a game, they understand constructs through usage which is often referred to as *discovery learning* ; c) Game theory states that games have standard, and well-understood, patterns, which enable new players to quickly understand and partake in a game on the basis of previous knowledge and familiarity with the pattern (Osborne, 2004). This overcomes any issues of unfamiliarity with a programming language or paradigm, reifies the operational abstraction of the programming constructs into a sequence of operational steps in the game environment, enabling students to complete the game and potentially gain understanding of the constructs at a level that makes sense to them.

In addition to these, Sung (2008) reports that *learning* introductory programming *through game-play* is independent of game programming and instead of doing programming; students are expected to understand CS concepts and develop their abilities in problem solving. CT patterns (i.e. decomposition, abstraction, pattern generalisation, algorithm design) are context and application independent and therefore, can easily be reflected and developed through game-play (Basawapatna *et al.*, 2011). They further suggest that once students understand conceptually how to present a pattern, they should be able to transfer and use it in the context they choose. Moreover, the majority of the studies using the previous two approaches (i.e. *individual game development models*, *extensive game development assignment*) follow an instructivist style rather than a constructivist one. In the previous two approaches, students are often given instructions by an expert tutor and knowledge acquisition is governed by that tutor in a module based teaching model. However, in a previous study McKenna & Laycock (2004) provided evidence that whilst an instructivist approach appears to work for short-term knowledge transfer; constructivist approaches provides deeper transferable understanding and the longer-term retention of knowledge.

To date, a limited amount of work has been undertaken to scaffold the development of CT skills and *learning how computer programming constructs work through game-play*. Long (2007) investigated the factors that kept students playing an open source tank fight game called Robocode (2001). Robocode (2001) was originally designed as an environment to support Java programming by allowing its players to develop artificial intelligence for their tank. The findings of this study demonstrated that students were more interested in discovering winning strategies in the game rather than either programming or debugging their programs. The study concluded with some evidence that an educational game can explicitly be designed to be predominantly motivational, but this does not necessarily mean that students would focus on the learning material in the game. Further to this, Barnes *et al.* (2007;

2008) and Chaffin *et al.* (2009) developed and evaluated their own games for the premise of supporting a deep understanding of loops and arrays in undergraduate programming courses. Although the intention behind these studies was to construct a serious games approach, they presented drill and practice activities during the game-play and the learning material was designed to overlay the game mechanism particularly for the reason of using the same game with different learning content. Regrettably, none of these studies reported a well-structured evaluation or demonstrated inferential statistical analysis to support their assertion that their games are educationally effective tools. Among the games developed for the purpose of learning programming Colobot (2007) is known to be the only complete commercial game that mixes interactivity, game-play, learning content and narrative elements (Muratet *et al.*, 2009). Players command different vehicles by writing pseudo codes in an in-game specific programming language (which is similar to C++) in order to complete various tasks. In contrast to this, Colobot (2007) does not support a multiplayer game environment, it is not free, and cannot be modified according to a specific curriculum.

In recent years, studies investigating the relationship between digital game-play and learning programming have increased. Papastergiou (2009) evaluated the effectiveness and motivational appeal of a game she developed for high school students mainly for learning computer memory and CS concepts. Her findings demonstrated that learning through game-play can promote abstract knowledge while encouraging motivation in learning computer memory concepts. Muratet *et al.* (2009; 2011) demonstrated their own framework called "Prog & Play", an open source real time strategy game built for the purpose of strengthening skills of students in programming. They designed a series of studies and asked various teachers and students to provide evaluative feedback regarding their game. However, participation in the experiment was quite low (15 students in the first experiment) and students chosen to participate in the study were deliberately selected for their motivation to play games. Although they observed encouraging results (such as an increase in students' interest in learning programming), their experimental design had serious flaws due to the fact that all participants were intentionally selected to have a good gaming background rather than being a random selection of the population. Li & Watson (2011) designed a Java based prototype with a car racing game theme in order to teach variables, methods, event handling and decision making. Wang & Hue (2011) also presented a similar system using a soccer game in the anticipation that this will attract students to learn computer programming. However, both of these studies did not present a structured experimental evaluation and arguably they could be considered to be male-oriented which might cause a gender-bias

problem in education. While Coelho *et al.* (2011) presented their work in progress to create a serious game for introductory programming, Liu, Cheng & Huang (2011) presented statistical evidence that students apply extensively different problem solving strategies in their game which has a direct correlation with their understanding of programming concepts. Their results showed that students who solve problems at a superficial level in their game are in fact the same students who are not motivated to learn programming. They concluded that the critical thinking and problem solving abilities of students can be fostered through game-play.

Despite the fact that experimental research in learning through game-play is advancing, the majority of current approaches do not evaluate whether or not knowledge has been gained or CT skills acquisition has occurred after game-play. Only a limited number of studies presented well-structured experimental research in this area and most of these predominantly focus on increasing the motivation in students (Hainey *et al.*, 2011). Therefore, it is not clear what students learn from playing serious games specifically designed for learning programming or how this might impact on their problem solving abilities. In addition to this, the majority of work in this area focuses on the reinforcement of conceptual programming knowledge rather than contextual and applied knowledge. Supporting the learning of conceptual programming knowledge is an effective method, but it is arguable whether or not it provides opportunities for students to develop their skills in CT.

In conclusion, while a number of approaches to the development of learning through game-play have been proposed, there is a dearth of evidence on what students learn from these environments and whether or not they develop the practical skills necessary to become effective programmers. More importantly, many studies do not provide access to their game framework and therefore it is not possible to a) design an experimental study to test their framework and relate the results back to their original work b) observe how the features described in their paper can be applied.

For the reasons described above, a new game model and the implementation of this was decided to be developed to a) allow students to practise their skills and abilities in CT, even if they have little or no programming background b) support students through the process of learning computer programming by demonstrating how a limited number of introductory programming constructs work in practice.

2.4 Serious game models

This section discusses the most widely referenced serious game development and evaluation models from the literature by analysing their advantages and disadvantages.

Learning in serious games is a multi-dimensional construction of learning skills and cognitive learning outcomes (Pivec, Dziabenko, & Schinnerl, 2003). To achieve this, it is necessary to provide a deep level of interactivity that stimulates players to be engaged in the learning environment. It is also crucial to allow players to design a development plan or make decisions at certain points and test how the outcomes of the game are generated based on their decisions and actions.

Garris, Ahlers, & Driskell (2002) developed a game based learning (GBL) model that illustrated how players can be engaged when they play a serious game. Although many different frameworks and models are proposed after this (O'Neil, Wainess & Baker, 2005; de Freitas & Oliver, 2006; Robertson & Howell, 2008), Garris *et al.* (2002)'s game based learning model remains as the most widely referenced and accepted work in the literature.

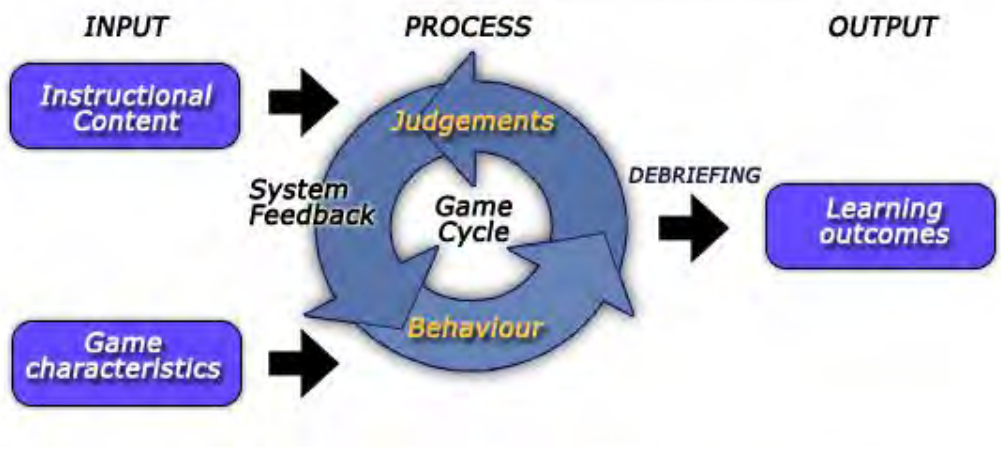

Figure 2.2 – Input – process – output game model Garris et al. (2002).

As shown in Figure 2.2, Garris *et al.* (2002) proposed an *input – process – output game model* that primarily aims to enhance the intrinsic motivation of players towards achieving learning outcomes. Through this model they integrated a repeated *judgments – behaviour – feedback* cycle that would engage players into playing a serious game as well as increasing their enjoyment and confidence. In other words, the model was explicitly developed to show pathways on how to design serious games to be intrinsically motivational. They also reported that learning outcomes from their model can be categorised as *skill based*, *knowledge based* and *affective*. They argue that in a game environment, learning happens when players evaluate

a situation and consider every possible option they perceive. Further to this, players make judgments based on evaluations and modification of their behaviour within the game cycle which results in the game-play continuing within the repeated *judgments – behaviour – feedback* cycle as players intrinsically monitor the situation and manipulate it.

Based on this model, Garris *et al.* (2002) and later Ma *et al.* (2007) argued that learning in games concerns repetition of cyclic contents. They suggested that cyclic learning contents in games can be achieved by separating instructional content from the game characteristics. Therefore, a serious game can explicitly be designed to be intrinsically motivational. Additionally, this type of approach ensures that a game can be used for multiple learning contents and purposes within a domain.

A number of studies (Robertson & Howell, 2008) also provided models and evidence on how serious games can be designed and evaluated. Gee (2003) described a cyclic content of events very similar to *input – process – output* game model where players can probe, hypothesize, reprobe and rethink. O'Neil, Wainess & Baker (2005) proposed an educational framework and argued that isolating instructional content from the game structure is not an effective approach. They also defended the premise that serious games are not effective in isolation and should be combined with other instructional support. De Freitas & Oliver (2006) proposed a four-dimensional framework (i.e. pedagogic considerations, learner specifications, context and model of representation) for helping tutors to evaluate the potential of games within their practice. It addressed a gap in the literature by focusing on context, learning theory, practice and learner groups in using serious games and simulation. Their framework is an extended methodology that could be used for evaluating serious games in addition to designing them for learning purposes. Despite this, Robertson & Howell (2008) put forward the argument that their approach heavily relied on having a good background in computer games and that it does not sufficiently support tutors in identifying which games would be applicable for given learning outcomes. More recently, Suttie *et al.* (2012) and Abeele *et al.* (2012) proposed their own player-centred frameworks that are specifically focused on the design and development of serious games. Despite their respected attempts to ease the work of the serious game designers, both approaches focus on theoretical underpinnings and lack of a clear demonstration of how to produce motivational and pedagogically effective games. As a result, although there are many other serious game models in the literature, research in game based learning (GBL) often reference the Garris *et al.* (2002) model as the ideal method to show how learners engage in educational games and as a way to illustrate how learning take place in games.

When the work of Garris *et al.* (2002) is investigated; there is no doubt that the model encourages players to understand or remember concepts at an abstract level as the learning material is covered in the game-play. However, it is arguable whether or not this model can achieve a form of constructivist learning because a) the presentation of learning material is not an integral part of game characteristics or game-play (i.e. what is being learned is independent from the learning platform); b) what makes a game motivational does not have an explicit relationship with the learning material.

Many studies (Prensky, 2004; Gee, 2005; Arnseth, 2006) stated the importance of constructivist learning and identified that the contextualization of gaming in regard to learning and the quality of discourse surrounding the game-play is more important when game-play is explicitly designed to support learning. Gee (2005) states that the dilemma between "knowledge as information" and "knowledge as activity and experience" triggers another conflict which comes from research on cognition. This is the dilemma between general, abstract understandings, and situated understandings (the ability to understand in ways that are customizable to different specific situations).

As an example of this, Graven and MacKinnon (2008) successfully integrated an introductory programming curriculum for Java into their games using a quest-based approach that facilitated learning through repetition – a vital aspect of the work of Garris *et al.* (2002). Additionally, the learning material in their game has no explicit relationship to the game-play and the way it was presented is epitomised as a *drill and practice approach* in educational theory. They offered the players a far greater level of control over their interactions, introducing a level of constructivist design in the environment, arguing that games are inherently constructivist in nature. However, they concluded that their approach needed further development as players are not essentially engaged in learning content despite the fundamental constructive structure of their game. They determined that the failure of many games for learning programming comes about as a result of failing to build a relationship between the learning materials and the game-play aspects of the game, concluding that games need to be developed in a way that learning materials should be an integral part of the game-play. According to their research, the learning materials can be integrated into a game environment similar to a traditional instructional design model which is an effective and a reasonable strategy but not constructivist in nature. They further argue that instructivist artefacts offer clear instructions, structure and familiarity but deep and long lasting knowledge is more likely to arise from constructivist learning environments. When the learning content overlays on the top of the game-play, the knowledge is usually delivered through a series of statements via

text, graphic or audio and more often than not, the GBL approach ends up being a game-like version of an educational website containing the same thematic units and hence, lacks a constructivist structure.

In addition to these, Savery and Duffy (1996) state, "cognition is not just within [an] individual but rather it is a part of [the] entire context, i.e., cognition is distributed." In other words, learning cannot be separated from how it is learned and it is an individual construction of both content and context. Thus the learning environment and the learning material should engage with each other and both of them needs to be constructivist in structure. When learning material overlays on the top of the game-play, the game-play can be enjoyable and an effective way of supporting the learning of conceptual knowledge but it does not provide opportunities for students to develop their skills in computational thinking. At this point, a clear definition is required to explain the difference between games that support the learning and reinforcement of conceptual knowledge, and games that support the learning of procedural and applied knowledge, and through this skills acquisition and development. In the first case the contextual relationship between the focus of the game and the knowledge being acquired is less important and may be completely abstract, whereas in the latter case the contextual relationship between the game and the knowledge is paramount, hence the main concern to see the utilisation of game-play.

2.5 Guidelines specifically designed to develop games for learning how programming constructs work through game-play

This section highlights that there is no clear structure or set of guidelines for developing a game specifically for learning introductory programming. Therefore, a series of guidelines were derived from the literature and presented in this section in order to draw a pathway to answer the research question of this research (i.e. *Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming?*).

The current available guidelines in game based learning (GBL) discuss lowering the technological requirements and also put forward suggestions on motivational driven game design, pedagogic attributes, adaptation and assessment mechanisms (Moreno-Ger *et al.*, 2008). However, little work has been done to provide a set of guidelines on how to develop serious games particularly for learning programming purposes. Sung (2009) discusses the integration of game development and proposes guidelines to be considered when game

development is taken as an approach to engage students in learning programming. Furthermore, he suggests using his guidelines when game development or game content is integrated in computer programming classes. Although his guidelines are useful in this field, he does not propose that learning traditional programming can also be done through game-play. Moreover, it is arguable whether or not the work of Sung (2009) sufficiently considers computational thinking and learning traditional programming together as he did not explicitly mention the development of computational thinking skills.

With respect to previous work, a series of suggestions were gathered from separate resources in order to draw a map in developing a serious game specifically for learning how programming constructs work through game-play. These suggestions were selected based on the listed problems students have in learning computer programming (see Section 2.1). Although there is no evidence to defend that these guidelines would support the successful development of a serious game, each part of the guidelines stands as strong suggestions in previous work (Guzdial, 2008; Lu & Fletcher; Sung, 2009;). As an example, Guzdial (2008) defended the idea that an approach to teach computational thinking (CT) should be available to all students and must be accessible at all times. He reported the results of experimental research on programming control flows in order to highlight how students perform better when programming code is presented at a level that makes sense to them.

Hence, it is planned to use these guidelines for building a serious game that would allow the development of CT skills and learning computer programming constructs.

Various suggestions taken from the literature are categorised into 8 different parts:

1) Institutional insight
2) Academic support and scaffolding strategies
3) Gender and expertise neutrality
4) Settings for serious games
5) Conceptual integrity
6) Learning as part of the game-play
7) Collaboration, coordination and competition
8) Constructivist learning.

Each part of the guidelines is described below.

2.5.1 Institutional insight

The most important consideration when designing game content is to avoid making significant changes to the order and core structure of a computer programming course. In other words, the game-play should be relevant to, and consistent with, the learning material of a traditional programming course. There needs to be a strong link between the game-play and the curriculum of a programming course so that students can transfer their learning outcomes from the game into learning programming with a programming language. To achieve this, recent research proposes that developing CT needs to be addressed separately from a programming curriculum as these two set of skills have different learning goals (Wing, 2008; Guzdial, 2008). To overcome this conflict, many institutions have already started revising the fundamental nature of programming courses and are introducing new first year modules particularly to develop student skills in CT. Because GBL should not attempt to change the nature of the programming courses, current approaches try to simultaneously address both the curricula and the development of student skills. Lu & Fletcher (2009) argue that setting CT in separate courses, or as separate sections within a course, could positively affect the efficiency of GBL within this domain. On one hand, there can be games specifically designed for improving computational and algorithmic thinking; on the other hand games could explicitly be designed for learning programming code to solve problems. Nonetheless, a GBL approach for learning computer programming at the CT level must work within the bounds of an institutional oversight and should not change the learning objectives prepared by an institution. As suggest by Sung (2009), the main strategy to apply this is the use of a partial curriculum scope that aims to achieve limited number of learning outcomes each time a GBL approach is designed.

2.5.2 Academic support and scaffolding strategies

The background of academics might play a critical role in learning how programming constructs work through a GBL approach. Regrettably, not all traditional programming instructors are familiar with games and game development technologies. More importantly, academics may not be aware of the differences between teaching traditional programming and how this might be reflected in GBL. As an example, the *forever* block in Scratch (2010) may result in novice programmers assuming that an infinite loop is good programming practice and has the same purpose in traditional programming. Through using this block students can create working scenarios, and might assume that an infinite loop epitomises the flow of time in

programming. Thus, they might accept that they need to create an infinite loop and all programming commands should exist within it. However, in traditional programming an infinite loop is often created mistakenly, and with negative consequences. It is crucial to realise that there are two problems here. First, students need to re-learn consistent concepts (such as objects, loops) after playing the game because the game may hide details about these concepts. Second, students need meaningful feedback to drive them to the correct use of computer programming constructs. Henceforth, a serious game should prepare the base for scaffolding strategies through delivering feedback that would encourage students on the correct use of computer programming constructs. Additionally, scaffolding tasks may drive students to practice CT skills from game-playing experience but applying these skills in programming with a programming language requires help from the instructors. The key role of instructors is when skills and tactics gained from game playing are transferred into learning programming syntax and techniques. Consequently it is crucial to ask: Which tasks are more appropriate for game-play and which tasks are more appropriate for instructors?

2.5.3 Gender and expertise neutrality

It is important to design a GBL approach that is both gender and expertise neutral. This problem is a core pedagogical problem in all science, technology, engineering, and mathematics (STEM) fields (Hill, Corbett, & St Rose, 2010). However, when designing serious games this problem needs extra attention because there is a common assumption that most games are male oriented and that most women are not good game players despite the fact that research results show otherwise (Pratchett, 2005). An approach for learning CT should avoid male or female oriented settings and should not trigger a gender bias issue. Currently students come with a wide variety of backgrounds, prior knowledge and abilities and this will have an impact on how they learn computer programming. If the learning environment is aimed at the pace of the slowest student, the high achieving students might become bored and frustrated. Conversely, if it is aimed at the high achieving students it is likely that those who are finding the topic difficult will drop out. However, taking the middle ground in the hope of delivering to the widest student audience will still not meet the needs of all students. Thus, the solution to this problem lies in supporting students in gaining the required underpinning skills and knowledge at their own pace, regardless of their background, while letting those who already have the skills and knowledge skip the preliminary stages and move to a more advanced level (Cooper, Dann & Puasch, 2000).

2.5.4 Settings for serious games

The content used in a serious game to encourage CT should be free, simple and available for use by students at all times (Guzdial, 2008). Many visual programming tools are available free of charge and include tutorials but it is very hard to find free serious online games specifically designed for developing CT skills. One of the core aims of this research is to design a game that is free and available to the public so that people can use it. Furthermore, some research defends the use of serious games in a classroom environment (Ketelhut *et al.*, 2005), while others take the view that serious games are more appropriate in students' home than in the classroom environment because it is difficult to restrain game-play within a limited period of time (Egenfeldt-Nielsen, 2005). It is important to recognise that achieving both settings in one game is difficult as different settings have different dynamics and thus it needs to consider different game rules and play time. Additionally, it is not possible to say one setting is better than another as both settings have their own strong points. What needs to be done is to clarify an approach on how to implement the game in a chosen setting, as well as how students would be supported throughout the game.

2.5.5 Conceptual Integrity

Conceptual integrity refers to maintaining a central theme on computer programming constructs. The ultimate goal of a GBL approach designed for developing CT is to aid students in learning and using core programming constructs to solve various problems at an abstract level. Therefore, a game should present problems at a level that everyone can understand and be able to develop a solution. This way, students can develop their skills in problem solving through using scientific concepts and transfer them to learning to write program code. This will make learning programming more manageable and sensible than a traditional teaching approach. It is essential to focus on the development of CT and learning from experience rather than simply presenting conceptual and abstract knowledge. Through this, the solution to a problem should be conceptually traceable back to the origins of the problem, and should not focus on how to write the program code as this is at a different level (Repenning *et al.*, 2010). It is crucial to recognise that CT is at an operational level of abstraction and stands on a conceptual layer whereas computer programming is more related to context rather than concept and thus stands on a procedural level. The conceptual integrity of a GBL that aims to develop skills in computational thinking should be closer to the level of CT rather than learning

programming at a coding level.

2.5.6 Learning as part of the game-play

Jenkins (2002) states that "programming is not a body of knowledge, it is a skill" and the most effective type of learning in this field is *learning by doing*. A GBL approach for learning programming should provide opportunities for an active learning, trial and error paradigm rather than simply supporting students through conceptual knowledge. Choosing a suitable genre for the development of a serious game is the first step in designing the game-play. By doing so, it is possible to make the computational model behind the game explicit, so it can be planned to include the different pedagogical requirements identified. In particular, the actions of the player should focus on triggering state transitions and drive them to learning the instructional content (such as using loops) while the sequence of these actions should lead them to one or more outcomes (such as using loops to deal with iterations).

It is anticipated that CT requires combining features of visual programming tools and commercial off the shelf (COTS) games and therefore, using strategy or puzzle solving games where solving problems can have meaningful outcomes, can be an ideal solution. This way, students can incorporate different features which would enable them to improve their skills in CT. Through the game-play such as dragging and dropping commands into a specific area, players can develop their own strategies within a problem based learning environment which has learning outcomes related to programming. For example, in Sid Meier's Railroads (2006) computer programming constructs can easily be adapted to make learning a core part of the game-play. Players can manage a railroad business where building stations and signal towers can control railroad switches and thus they can indirectly control the movements of trains on the railways. By controlling these switches players can direct a train inside a circle railway causing it to loop until a specific condition is met. When the end condition of the loop is satisfied the train can leave the circle railway and continue on its way. Moreover, the signal towers can be managed by using different functions and a manager class to control the state of switches and finally the cars of the train can easily be associated with arrays. The crucial factor here is to relate learning closely to how players play the game and what they have to do in order to demonstrate good game-play.

Finally, neither visual programming tools nor the currently available games specifically designed for learning computer programming sufficiently support learning as a core structure of a game. When learning becomes an integral part of the game-play, it means it is a part of

game *dynamics* and *aesthetics* and therefore the entire game *mechanics*. Regrettably, there is very little work in the literature that investigates the relation between game mechanics and the integration of learning outcomes as the majority of studies in the literature trust in the constructivist and motivational nature of the games (Kazimoglu *et al.*, 2010b). Whitton (2007) states that any engagement in a game does not always lead to an engagement in learning and that the sole reason of using games for learning purposes should not be because they are perceived to be motivational. Hence, in order to create an engaging game-play, learning needs to be evaluated as a part of the game mechanics which eventually means games should be designed for high level learning goals (i.e. analyse, create, apply and evaluate) rather than targeting low level learning goals (i.e. understand, remember).

2.5.7 Collaboration, coordination and competition

Sancho *et al.* (2008) underpinned the importance of collaboration, coordination and competition in learning programming within a GBL environment. These guidelines also support this and stress that one or more of these aspects of socialising (i.e. collaboration, coordination and competition) are necessary in order to create an effective environment for learning programming. In collaboration, the players in the same team have to collaborate to reach the best solution they can and they usually play a predefined role with concrete duties and responsibilities to help each other and perform activities within the game-play. This way, slow paced students can observe behaviours and judgments of fast paced students which allow the GBL environment to serve as a platform for effective learning. Additionally, fast paced students can observe and develop different strategies and thus benefit from each other (Sancho *et al.*, 2008). Recent studies in this field report that student-to-student collaboration within a learning context is the most powerful predictor that learning is taking place (Barker *et al.*, 2009). Ladd and Harcourt (2005) argue that if competition is used successfully in games for learning programming, it offers fun and engagement for students while providing a challenge between players that continually encourages them to develop efficient solutions to problems. However, unnecessary competition that does not consider gender or expertise neutrality may result in players dropping out or feeling alienated from a game environment. Thus, competition needs to be integrated into a game environment carefully as it should not pressure students to compete with other students but rather encourage them to compete with themselves. Therefore, competition among students is best reserved as optional, particularly for those students who desire additional challenges so that this may allow advanced students additional learning

opportunities beyond the curriculum. As a result, appropriate and effective use of competition and/or collaboration should be offered within the GBL environment in order to engage students in supplementary activities.

2.5.8 Constructivist learning

The final category for these proposed guidelines considers the use of *constructivist learning* in a GBL model in order to ground the gameplay in pedagogical theory. In their seminal work, Savery and Duffy (1996) state that constructivism is a philosophical explanation of how people understand and know, and that one of the best examples of constructivist learning is problem based learning. They categorised constructivist learning environments in three propositions: a) Understanding is the core concept of constructivism and cognition is in our interaction with the learning environment not just within an individual; b) puzzlement is the stimulus for learning and it determines the body of what is being learned; c) knowledge evolves through social negotiation and evaluation of individual understanding and thus requires individual support. Savery and Duffy (1996) present problem based learning as an instructional model and demonstrate how it is consistent with the principles of instruction and constructivism. Furthermore, they state that the work of Lebow (1993) is significant in interpretation of instructional strategies within a constructivist learning environment. The instructional principles taken from constructivism in the work of Lebow (1994) are listed below:

1) "Anchor all learning activities to a large task or problem.
2) Support the learner in developing ownership for the overall problem or task.
3) Design an authentic task.
4) Design the task to reflect the complexity of the environment.
5) Give the learner an ownership of the process used to develop a solution.
6) Design the learning environment to support and challenge the learner's thinking.
7) Encourage testing ideas against alternative views and alternative context.
8) Provide opportunity and support on both the content learned and the learning process."

As shown from the propositions of Savery and Duffy (1996) and the instructional principles of constructivism proposed by Lebow (1994) clearly guide how game-play should be grounded within a pedagogical perspective for learning programming at the level of CT. It is crucial to highlight that students who actively engage with the learning material are more likely to recall information, and thus it is essential that the learning content should be an integral part of interaction and feedback in a game environment. This way, players can learn from a game by using in-game elements in a constructivist manner rather than any less efficient instructivist model (such as a drill and practice approach).

2.6 Summary

This chapter first explained the problems students have in learning computer programming and then discussed what computational thinking (CT) is and which cognitive skills encompass computational thinking. Having performed a detailed analysis of the current literature, the chapter clarified that conditional logic, algorithmic thinking, debugging, simulation and socialising are the core five cognitive skills that characterise CT. Further to this, the chapter explained that computational thinking is different from learning computer programming in three different dimensions: a) choosing the right abstraction; b) operating at the operational level of abstraction; c) defining relationship between the layers of abstraction.

After defining the differences between computational thinking and learning computer programming, the chapter explored how several studies investigated games and game-like environments to implement game based learning for teaching computational thinking strategies as well as introductory computer programming constructs. The chapter also discusses that although considerable efforts have been made to develop game based approaches to teach CT and introductory computer programming, very few results provided structured rigorous tests based on statistical evidence to show whether or not a serious game can be an educationally effective tool for learning how introductory programming constructs work at the computational thinking level. Therefore the empirical evidence regarding this is still missing in the GBL literature.

Finally, the chapter discussed the different serious game models, particularly the *input – process – output game model* of Garris *et*.al (2002). In order to draw a pathway to answer the research question (i.e. *Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming?*), strong suggestions were derived from the literature to create a series of guidelines.

The next chapter revisits the research questions and refines it into a structure that can be explored through a methodology. It will then present a new innovative game model and implementation to demonstrate how to develop a game specifically for learning computer programming constructs through game-play.

CHAPTER 3

DEVELOPING A RESEARCH TESTBED

Chapter 2 argued that the current research and models in game based learning (GBL) do not necessarily focus on deep game-play for learning how computer programming constructs work particularly in relation to applied knowledge and skill development. Having analysed the literature, this chapter refines the main research question of this research and explores a direct contextual relationship to the application of computational thinking in the process of learning how introductory computer programming constructs work through game-play. Further to this, the chapter introduces a new game model specifically designed to learn computer programming constructs while also developing computational thinking skills. The chapter then presents *Program Your Robot*, a serious game that was developed as an implementation of the game model through applying the proposed guidelines outlined in Chapter 2 Section 2.5.

Section 3.1 refines the main research question of this research as a consequence of the reviewed literature. Section 3.2 presents a game model called the *interaction – feedback loop* which is specifically developed for learning computer programming constructs at the computational thinking level based on the body of work in this area. Section 3.3 presents *Program Your Robot*, a serious game that is the research testbed developed through following the proposed guidelines and within the structure of *interaction – feedback loop* game model.

3.1 Refining the main research question

Having defined the cognitive skills that characterise computational thinking (i.e. conditional logic, algorithmic thinking, debugging, simulation and socialising) in the previous chapter (Chapter 2 Section 2.2), this section revisits the main research question of this research and refines it in order to make it more focused.

As discussed in Chapter 1, the main research question of this research is:

"Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming?"

This research question was complex and structured but it was too broad and needed a limitation in order to ground the question into a research study. Therefore, the concept of "*computational thinking*" is replaced with "*computational thinking skills*" which would refer to the cognitive abilities that characterise computational thinking (i.e. conditional logic, algorithmic thinking, debugging, simulation and socialising). A similar modification was performed on "*introductory programming*", as it is changed to "*how key introductory programming constructs work*" to narrow down the main research question with a limited number of key computer programming constructs particularly the first four programming constructs introduced in the computer programming course at the University of Greenwich (i.e. programming sequence, functions, decision making and loops). Hence, the main research question was redefined with a limited number of skills and constructs so that it could be explored through a modelling structure. The main research question is refined as:

"Can a serious game be designed to support the development of computational thinking skills through the medium of learning how key introductory programming constructs work?"

3.2 Interaction–feedback loop: a new model for learning how programming constructs work through game-play

Having refined the main research question, this section introduces a new game model that would be used to answer the question. The game model was specifically developed for this research and a discussion was put forward to explain why a new model was developed rather using an existing serious game model.

From among the mentioned game models in Chapter 2 Section 2.4, the *input – process – output* game model of Garis *et al.* (2002) was the one which most closely aligned to the structure of this research because the model a) encourages learners to intrinsically monitor problems, manipulate them and come up with a solution; b) is based on judgements and behaviours of learners which is an ideal way of developing skills in computational thinking.

Despite its distinct advantages, this research identifies three important drawbacks of *input – process – output game model*: firstly, the model focuses on motivating players intrinsically but it does not necessarily emphasis whether or not players are engaged with the learning material. Secondly, the definition of what a game characteristic is can change from one game to the other and there is no consensus in the literature about what game characteristics are. Finally, it

is debatable whether or not the model follows a constructivist approach as the learning content does not evolve through game-play because it is not a part of the game characteristics. Each of these problems is discussed in detail below.

The first problem with the *input – process – output game model* is that although it is exceptional in intrinsically motivating players in a game environment, it does not ensure achieving higher level learning goals (i.e. analyse, create, apply and evaluate) are learned. The model is more concerned about motivating players and engaging them with lower level learning goals (e.g. remember or understand). In other words, the integration of learning content may lack a direct relationship with the playing experience simply because what makes a game motivational has no relationship with the learning content. Therefore, when learning content is not part of the *game characteristics*, players are likely to focus on game-play and ignore the learning content.

The second problem is that it is arguable whether or not Garris *et al.* (2002) categorised *game characteristics* accurately for all serious games as there are intense debates about what *game characteristics* are in the literature (Pivec, Dziabenko & Schinnerl, 2003). As an instance, Prensky (2001) defined the characteristics of games as "rules, goals and objectives, outcomes and feedback, conflict (and/or competition, challenge, opposition), interaction, and representation of story" while Garris *et al.* (2002) categorised game characteristics as "fantasy, rules/goals, sensory stimuli, challenge, mystery and control". As there is no consensus on what game characteristics really are, it is easy to misinterpret what is essential to design a game according to an *input – process – output game model*. Despite the fact that the model proposed by Garris *et al.* (2002) is extraordinary and exceptional in showing how engagement can be achieved in serious games, it does not explicitly focus on designing games for a specific purpose or underpin sufficiently what game characteristics are essential for learning a specific piece of learning content.

Finally, as discussed in Chapter 2 Section 2.4, this model tends to focus more on an instructivist approach rather than a constructivist one because the learning outcomes are separated from the game mechanics and do not explicitly improve through the game-play. Although an instructivist approach can be very informative, lacking the constructivist structure cause the model to have is no essential relationship between the game-play and the learning objectives.

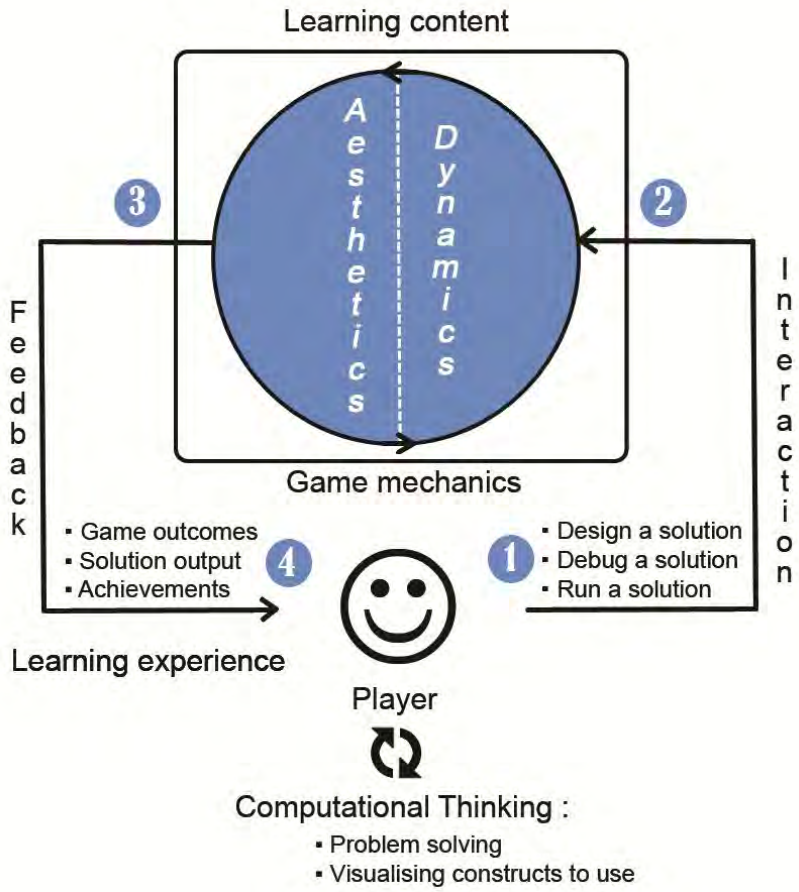

Figure 3.1 – Interaction – feedback loop game model.

As shown from Figure 3.1, in order to overcome these limitations the *interaction–feedback loop* game model was developed specifically for designing games to teach how computer programming constructs work at the computational thinking level. This new model is built on the top of the work of Garris *et al.* (2002) and similarly, it is perceived that players continuously need to increase their problem visualisation and solution-development as they progress through a serious game. Despite this, the learning content is not separated from the game characteristics in this model. The crucial difference between this model and the *input – process – output game model* is that the learning material is designed to be an integral part of aesthetics, dynamics and game mechanism rather than overlaying on the top of the game-play. The rationale behind this is to present the learning material as in-game elements that players eventually use to play the game.

The learning material in the *interaction–feedback loop* is designed to be an integral part of the game-play in the game mechanics. In other words, the model was created to develop computational thinking skills for learning computer programming constructs rather than being a generic solution to overlay the game with different learning content. The learning material

can be delivered through two units: *interaction* and *feedback*. While the interaction part consists of designing, debugging and running solutions, the feedback part provides outcomes, solution outputs and achievements. Therefore, players can analyse, create and apply their solutions which would drive them to focus on high level learning goals rather than low level learning goals. In other words, when players are intrinsically motivated in a game environment, they would also be engaged in the learning material as they would be using the learning material as in-game elements.

The *interaction – feedback* loop covers the first flaw of *input – process – output game model* by recombining learning content with the game-cycle. Rather than overlaying on the top of the game-cycle, the learning content is an integral part of the game-cycle. The model divides the game cycle and game characteristics into three categories as *mechanics*, *dynamics* and *aesthetics* (MDA) (Salen & Zimmerman, 2003; Hunicke et al., 2004; Schell, 2008). According to Schell (2008), *mechanics* describe procedures and rules, particularly the components of a game. Hunicke *et al.* (2004) discuss that any action, behaviour and/or control method afforded to player is related to game mechanics. Together with the game content, mechanics support overall game-play, how players can or cannot achieve their goals. *Dynamics* describe the run-time behaviour of mechanics and how it acts on player inputs (Hunicke *et al.*, 2004). Dynamics also allow players to leave their marks in the game because when players interact with the game environment, they change the dynamic structure of game-play, which ultimately creates *aesthetics* experiences. Finally *aesthetics* describe game response and outcomes evoked by player interaction. In other words, as players play a game, the game mechanics generate feedback according to the player's action. Aesthetics also describe emotional content, which refers to all the kinds of fun players get from playing it (Salen & Zimmerman, 2003). Thus, aesthetics are various game components (such as sensation, narratives and challenges) that define how a game looks and feels. Each game pursues multiple aesthetic goals in varying degrees according to the game genre. Schell (2008) states that aesthetics are the most important part of game design. This is because they have the most direct relationship to player's experience. Together with the dynamics, aesthetics create an infinite game cycle within the mechanics as dynamic actions generate aesthetic experiences, and in the same way aesthetics can allow new dynamics to be available in the game-play.

The *interaction – feedback loop* addresses the second flaw of the *input – process – output game model* by replacing *game characteristics* and *game cycle* with *game mechanics*, *dynamics* and *aesthetics*. As mentioned previously, there is no ubiquitous agreement in the literature as to what really game characteristics are and additionally these may change from

one game to the other. However, game mechanism, dynamics and aesthetics are precise and well defined concepts (Hunicke *et al.*, 2004; Schell, 2008) that exist in all games without exception as all games have a) rules, goals and limitations (*game mechanics*); b) interaction and structure of game-play (*dynamics*); c) response and outcome evoked by player actions (*aesthetics*).

Finally, the *interaction-feedback loop* is an iterative cycle of learning where players can learn from experience as the game mechanics guide them to discover how programming constructs work. Therefore, as players interact within the game mechanics and try to demonstrate a good game-play they develop their skill using a constructivist approach rather than an instructivist one.

The *interaction – feedback loop* is a development on the top of the Garris *et al.* (2002)'s work but it is not intended to replace the *input – process – output game model* as the loop is specifically created to develop computational thinking skills rather than gaining of pedagogic knowledge and thus, it has its own limitations. Despite the fact that the *interaction – feedback loop* addresses three important flaws of Garris' *et al.* (2002)'s work, there is a need to develop this model further through investigating the pedagogic foundations in order to establish learning theories and instructional strategies into it. The *interaction – feedback loop* model explicitly supports experimental, discovery/inquiry and constructivist approaches to teaching and learning and it defends the idea of using only the basic learning attributes of drill and practice to teach a subject at hand in an educational game is not an efficient instructional strategy. Despite the fact that the model is based on pedagogical approaches that promote questioning and active experimentation by learners, this need to be explored further in order to ground the model into a pedagogic context especially in discovery learning, constructivism and experimental learning. Arguably, this is a limitation of the model as this research merely focuses on skill acquisition and development in computational thinking rather than being a generic approach for pedagogic knowledge gain. Considering this, the *interaction – feedback loop* can be developed further to be grounded on learning and instructional theories and thus the instructional events and experiences integrated with the game-play can be clearly reported to adapt the model for other learning purposes. Additionally, this would allow researchers to manipulate key variables in the model and determine what factors have effect on learner motivation and achievement and thus, they can have a clear solid foundation for informing future designs.

In conclusion, the *interaction – feedback loop* can be an ideal way for developing computational thinking skills to learn how computer programming constructs work because the

model a) clearly illustrates that learning (skill, knowledge based and affective) takes place within the game as part of mechanics, aesthetics and dynamics; b) does not overlay learning content on the top of a game but rather learning is an integral part of the game characteristics and game experience; c) shows how computational thinking is reflected to the game environment in a constructivist approach using the iterative cycle of interaction and feedback.

In order to accurately apply the *interaction – feedback loop* model, an implementation was developed through the proposed series of guidelines that was presented earlier in Chapter 2 Section 2.5. The implementation is based on based on constructivist theories, accounts for learning objectives, academic support, scaffolding strategies, gender and expertise neutrality, as well as activities based on an optional competition.

3.3 Implementation

This section presents an innovative serious game for practising and developing skills in computational thinking (CT) for the purpose of learning introductory programming constructs through digital game-play. A description of how a limited number of key introductory computer programming concepts have been mapped onto the game-play is provided and also and how an equivalent set of skills characterising CT can be acquired through playing the game. Further to this, the section explains how this serious game is grounded on the *interaction – feedback loop* model and how the game applies the proposed guidelines listed in Chapter 2 Section 2.5. Finally, the potential benefits of this game as a support tool to foster student motivation and abilities in problem solving are discussed.

In order to address the main research question (i.e. *Can a serious game be designed to support the development of computational thinking skills through the medium of learning how key introductory programming constructs work?*), a puzzle-solving serious game, *Program your robot* (http://www.programyourrobot.com), was developed and grounded on the *interaction – feedback loop* model.

Program your robot was designed to achieve two important goals: firstly, to develop a framework that would allow players to practise their skills and abilities in CT, even if they have little or no programming background. Secondly, to support the learning of procedural and applied knowledge for a limited number of key introductory computer programming constructs (i.e. programming sequence, functions, decision making and loops).

The game was particularly designed to practice four out of five CT skills (conditional logic, algorithmic thinking, simulation, debugging). The socialising aspect of CT was not included

because the main aim of the research was to encourage the development of individual cognitive skills that would support students to learn how computer programming constructs work.

The game was developed in Adobe Flash (2013) using actionscript 3 as the default programming language and supported by javascript, extensible markup language (XML), PHP: Hypertext Preprocessor and mySQL database. The Adobe Flash environment was chosen because it has an object oriented language similar to Java and a very good video compression technology suitable to create online indie-type games. Despite this, Adobe Flash has limitations such as it uses CPU power intensively and does not support mobile devices (Engadget, 2011).

3.3.1 Design and development

The aim of *Program Your Robot* is to steer a character to its target via the most viable route through using a series of commands that plays a key aspect in constructing efficient solutions. The game is designed to be a puzzle solving action game where players control a robot and help it to reach specific destination(s) by giving commands.

In many ways, this game-play is similar to Karel The Robot (1981) as in both games players need to control a character by using different set of commands. However, it is crucial to understand that Karel was not designed through a game based learning (GBL) model nor does it follow guidelines to deliver timely and effective feedback specifically to support computational thinking (CT) and learning how programming constructs work. Additionally, *Program Your Robot* was also inspired from other games particularly from Light-Bot (2008), Microsoft's Tinker (2008) and Robozzle (2010). However, there are considerable differences between *Program Your Robot* and the other similar games listed here.

Firstly, the learning material in *Program Your Robot* is represented in game elements and mapped onto part of the computer programming curriculum, more specifically the first four key areas (i.e. programming sequence, functions, decision making and loops) taught within the Computer Science department of University of Greenwich. Secondly, four out of five main categories of CT skills (i.e. *conditional logic*, *algorithmic thinking*, *debugging and simulation*) are explicitly integrated as patterns into the game mechanics of *Program Your Robot*. In other words, the game is built on the top of the cognitive structure of CT rather than for fun only whereas the games listed above are created for fun and not for learning purposes. Additionally, none of the above mentioned games sufficiently focus on the accurate use of programming constructs or that map to an introductory programming curriculum as this was not their aim.

Therefore, although the game-play of *Program Your Robot* was inspired from other games, crucial differences guided the development, such as the necessity to consider accurate use of programming constructs and the intention to practise cognitive CT skills during game-play.

Figure 3.2 shows the early prototype of the game where players help a character to collect flags at randomized locations. As the players proceed through the levels, the number of flags increased and thus problems presented in higher levels also increased in complexity. The distribution of flags was completely random and when players successfully collected all of the flags in one level, they progressed to the next level of the game. Additionally, a text area was designed and named as *equivalent programming logic* to show the java code equivalent to the logic created by the players in their game play.

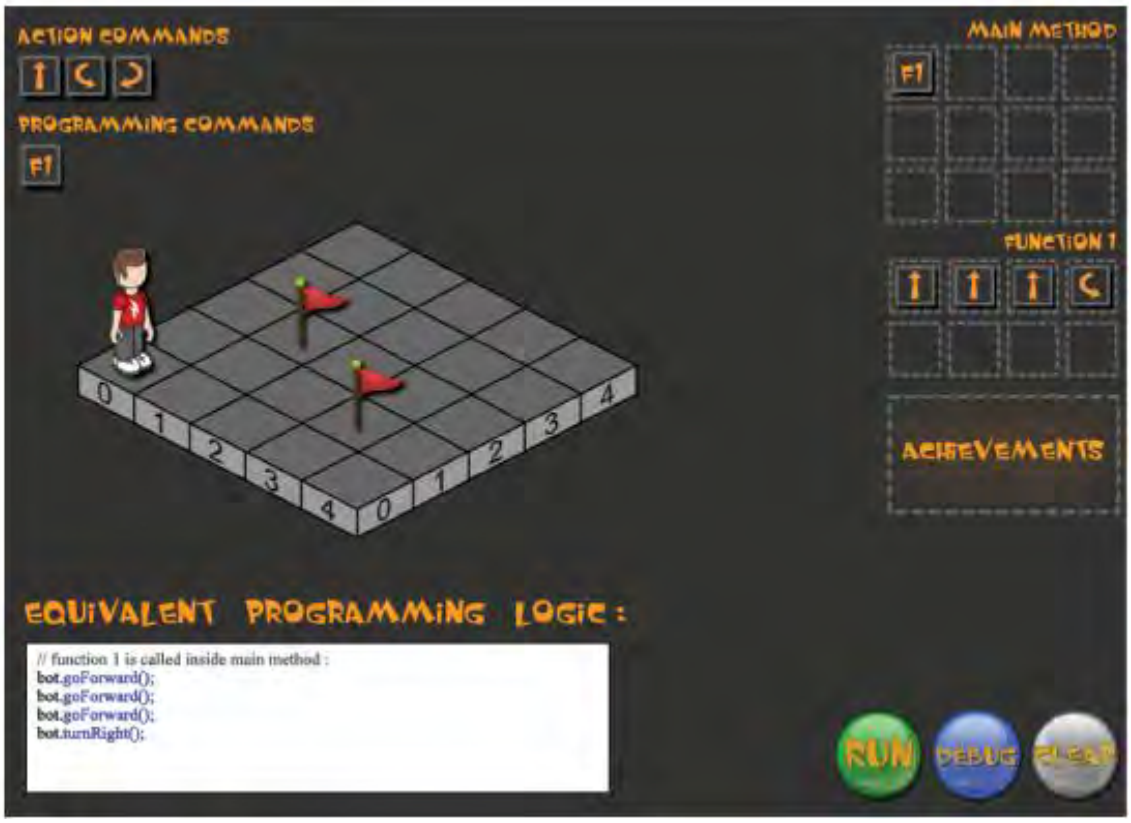

Figure 3.2 – Early prototype version of *Program Your Robot*.

Although the general design of the game was kept similar, the game continually evolved during the development period. The theme of the game was changed to helping a robot to escape from a grid platform by reaching the teleport square (each level contains only one teleport square), which will take players to the next level in the game. The game character was changed to a robot because it was aimed to achieve a game-play that does not trigger a gender bias problem. There are six levels in the current version of the game, each one having different

challenges to overcome and each one more difficult than the previous level. During the game-play, players need to construct solutions through using programming and symbolic representations in order to find pathways to help the robot reach the teleport square.

As shown in Figure 3.3, the current version of the game was visually enhanced with a better interface and game dynamics in order to provide an advanced game experience. Similar to its prototype, the current version of the game offers a series of commands to player in order to control the robot. The commands players can give to the robot are divided into two types: *action commands* and *programming commands*. While action commands are used to move the robot on the grid platform (such as go forward one space, turn to right), programming commands indirectly impact on these actions and facilitate constructing solutions. Both types of commands can be executed by the robot by dragging them from their associated toolbars on the left of the screen, and dropping them into specific areas called *instruction slots*. Players can give instructions to the robot by dragging and dropping any number of commands, of either type, into these instruction slots in any sequence they choose, for as long as empty slots are available. To complete a level, players need to instruct their robot to walk to the teleport square and they then light the robot's lights, which will then allow them to proceed to the next level. As players progress through the levels, the grid platform expands and new challenges (such as enemy robots and walls) are introduced. Additionally, each level contains bonus items that can be collected by the robot. These *collectible items* are randomly scattered every time players start to play a level, and thus this ensures that a problem presented to a player at one level is different from a problem presented to another player, or indeed the same player repeating the level in order to consolidate their learning. The randomness of the collectible items is also limited in the current version in order to guarantee the complexity of levels stays broadly consistent for each player.

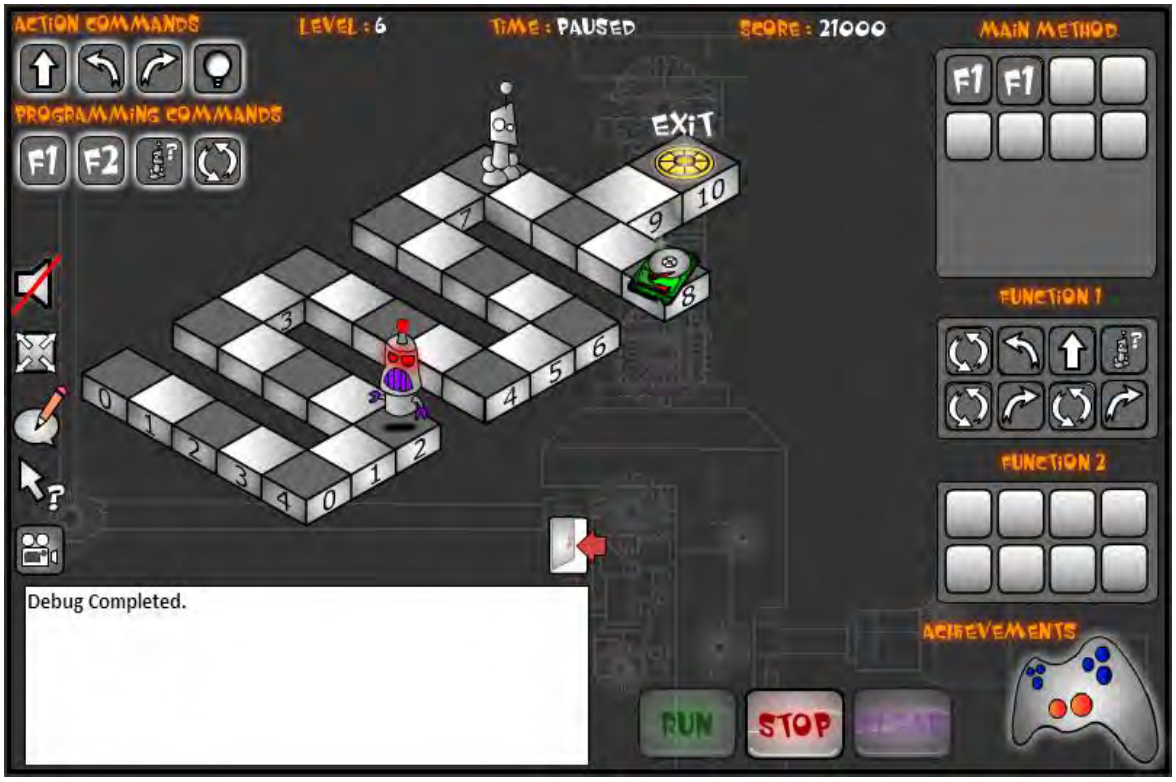

Figure 3.3 – The current version of *Program Your Robot*.

In addition to this, the current literature was examined in order to decide how to reward players in *Program Your Robot*. In their work, Bayliss and Schwartz (2009) discuss how game design can inform instructional design particularly on introductory programming courses. They state that there are two main forms of feedback available to evaluate students' learning progress, *summative* and *formative* feedback. While *formative feedback* provides suggestions based on student actions allowing them to try different solutions and understand a problem at a deeper level, *summative feedback* rewards students for achieving their goals through positive feedback. They further argue that delivering the two types of feedback is an important part of both good game design and instructional design. Additionally, the idea of using an achievement-based system is utilized in their work as a strategy to increase student motivation through behavioural conditioning. Having analysed this work, two different reward systems were integrated into *Program Your Robot* in order to motivate the player to construct efficient CT solutions as well as to provide motivation to learn how programming constructs work. These rewards are termed as *game score* and *achievements* respectively. Both reward systems are built on the top of the rules presented in the game which can be categorised into two as *operational* and *consecutive* rules.

Operational rules are the principles that players need to know in order to play the game.

These are delivered in two forms, video and text based tutorials (through dialogue boxes) at the beginning of each level. Each level starts with a video tutorial offering hints to players regarding what they need to do in that level. A text based tutorial is also available at every level for players who might require more support or who need to learn more on how programming constructs work. Alternatively, players can skip these tutorials and learn to play the game through a trial and error approach. Both tutorial screens are optional and predominantly explain the features of the game in addition to how challenges can be overcome using symbolic representations of computer programming constructs (henceforth called programming constructs).

Consecutive rules are designed to be the underlying logical structure of the game. These are simply the unwritten procedures regarding developing efficient algorithms to win the game. Similar to learning computer programming, an efficient algorithmic solution in this game can only be discovered by practising and combining different programming constructs. The goal here is to establish a well-structured framework that will continuously drive players to find underlying consecutive rules in the game. The current literature in CT also supports this idea and reports that abstracting game rules is an effective way to practise algorithmic and critical thinking, both core CT concepts (Berland & Lee, 2011; Lee *et al.*, 2011)

The first reward system is the *game score* that is measured in the game by tracking down the solution constructed by players and matching this solution to a set of pre-defined structured solutions, during the game-play. Furthermore, the game evaluates the efficiency of a player's solution based on how well they understand how programming constructs work. In other words, the overall score achieved by the players depends on how well they construct their solution. For example, those using functions, score more points than, duplicating the same commands in the main method. Additionally, the fewer number of slots a player uses to construct a solution, the higher the points score they receive. Thus, the desired solution lies in creating repeatable patterns with as few slots as possible, which can be accomplished by using various CT skills. Therefore, we ensure that players achieve a high score when they demonstrate a deep understanding of how programming constructs work in the game, such as when they combine loops with functions.

The second reward system is the *achievements* which are gold medals that players can unlock when they successfully complete additional challenges in the game. *Achievements* are meta-goals defined outside of the game's main purpose and are a feature of modern games. Achievements are mostly used in the commercial off the shelf (COTS) games to drive players to complete a specific task in a game however; this task often does not have a direct effect on

further gameplay (Hamari & Eranti, 2011). This modern feature is then integrated into *Program Your Robot* in order to drive players to complete additional challenges to further demonstrate their understanding of how programming constructs work. These challenges are evidence of programming *kudos* and are associated with using programming commands in the game coherently. For example, players do not need to use loops in level 5, they can use recursive functions to construct a solution to complete this level. Should the player use recursive functions, they can unlock the recursive achievement and similarly, should the player use loops they can unlock the loops achievement in the game. Therefore, the game rewards players who use computational thinking strategies to develop elegant solutions, and more importantly the reward offered to a player is related to their solution.

Players can perform different actions when they have finished constructing their solution. The first of these is simulating the solution by pressing the RUN button. When the RUN button is pressed, the commands inside the *Main Method* are performed in the sequence requested by the player. Thus, the game demonstrates how different programming commands are executed visually.

The *equivalent programming logic area* which existed in the prototype version is removed from the game and replaced with an *information area* to provide clear and comprehensible description to players when they have any problems regarding their game-play. As an example, should the solution a player has generated not work for any reason, they can debug the solution which allows players to step through the algorithm one command at a time, to try and work out why the robot doesn't behave in the way they expected. After a successful debugging process, the errors/warnings are shown on the *information area*, and are specifically written without any technical terms or jargon words to ensure players with little or no programming background can understand how to deal with their errors. Additionally, the debug mode works slower than normal run-time in order to show the player the sequence of commands, executing them one at a time on the screen. This allows player to find the bugs and work out how to fix them more easily. Hence, the main purpose of the debug facility is to encourage players to adopt the habit of debugging when they have problems in their solutions.

3.3.2 Associating game-play with computational thinking

As discussed previously in Chapter 2 Section 2.2.1, the core five cognitive skills characterising computational thinking (CT) are defined as *conditional logic*, *algorithmic thinking*, *debugging*, *simulation* and *socialising*. Conditional logic is a problem solving ability

related to problem identification and decomposition as well as to critical thinking (Wing, 2006). Due to the puzzle solving structure of the game, players are required to use their conditional logic in order to find the most effective pathway for their robot to escape. They need to think critically through a series of steps when generating their algorithms and ask themselves questions such as: Should I collect the collectible item? Is there a better or more efficient solution that I could have designed? Further to this, when players construct their solutions, they also need to use *algorithmic thinking* in order to complete all levels using as few slots as possible; hence this encourages players to create repeatable patterns using programming constructs. *Debugging* solutions in the game allows monitoring solution algorithms and detecting potential errors which is an integral component of both CT and programming (Wing, 2006). Correspondingly, *simulation* of solutions is also available in the run-time mode both for observing the behaviour of the robot and analysing whether or not a sufficient winning strategy is created in the game.

Despite four out of the five CT skills can be practised during the game-play, *Program Your Robot* was not explicitly designed to encourage *socialising* aspect of CT because it was primarily aimed to encourage the development of individual cognitive abilities to support learning of computer programming. Nonetheless, a limited level of *socialising* can happen indirectly through the reward systems integrated into the game. For those players who want to have additional challenges a high score list has been designed where advanced players can submit their scores and share them with other players. Although this does not address the full concept of socialising, participating in a score system encourages limited interaction among players. Additionally, the participation in the high score list is optional, hence players are not excluded from the game because of not doing very well but rather it is aimed to encourage them to perform better each time they play the game so that they can release their high score for others to see when they really do well. It is also anticipated that this approach will continually drive players to compete with themselves to improve their performance and achieve the best available outcome. As mentioned in the proposed guidelines, using competition without considering cultural issues, expertise neutrality, and/or gender bias problems may result many of the players dropping out from the game environment because of constantly feeling under pressure. Therefore, to eliminate a potential competitive element, scores are only released with the permission of the player.

Table 3.1 shows a set of game activities that describes how a student can develop their skills in CT through game-play. These game activities are associated with the previously defined skills that characterise CT and they illustrate how cognitive skills can be developed in *Program*

Your Robot. Additionally, Table 3.1 validates the rationale of identified CT skills as outlined in the literature. It is anticipated that this type of game-play allows players to visualise how programming constructs work as each programming construct has a corresponding action in the game.

Task	Associated CT skill category	Game activity	Rationale of the skill category
Defining problems and decomposing them into different units	**Conditional Logic**	Help the robot to reach the teleporter. Activate robot's light when robot stands on the teleporter.	CT is described as a problem solving approach in various studies (Wing, 2006; Guzdial 2008). In conjunct to this, Schell (2008) explains the idea of what a game is as *"a problem solving activity, approached with a playful attitude."*
Creating efficient and repeatable patterns	**Building Algorithms**	Create a solution algorithm to complete all levels with as few slots as possible. Use functions to create repeatable patterns.	Perkovic *et al.* (2010) describe computation as *"the execution of algorithms that go through a series of stages until a final state is reached."*
Practising the debug-mode	**Debugging**	Press the debug button to monitor your solution algorithm to detect any potential errors in your logic.	Wing (2006) describes *"debugging"* as an essential component of both CT and programming.
Practising the run-time mode	**Simulation**	Observe the movements of your robot during the run-time. Can you follow your solution algorithm? Do you observe the expected behaviours?	Moursund (2009) reports that *"the underlying idea in computational thinking is developing models and simulations of problems."*
Brainstorming, cooperation and/or competition	**Socialising**	As an optional challenge, try to complete with a friend of yours in the game. Which one of you scored better? What advice would you give yourself and to them for scoring better in the game? Discuss.	Berland & Lee (2011) refers social perspective of CT as *"distributed computation in which different pieces of information or logic are contributed by different players during the process of debugging, simulation or algorithm building."*

Table 3.1 - Examples of game activities associated with various categories of CT.

3.3.3 An implementation of Interaction – feedback - loop model

The development of *Program Your Robot* was based on the *interaction – feedback loop* model that was presented in Section 3.1. As explained earlier, the players are expected to design, run and debug solutions in *Program Your Robot* which forms the dynamic parts of the game mechanics. The aesthetics responses to these actions are reflected as the movements of the robot, game score and achievements in the game. Moreover, the learning content (i.e. programming constructs and skills that encompass computational thinking) is offered as in-game elements in a constructivist approach rather than an instructivist one. In other words, the learning content in the game is an inseparable part of playing the game and not laying on the top of the game-play – thus, learning originates from the gaming-experience. While dynamics actions of players produce aesthetics responses in the game, players use computational thinking (CT) and problem solving abilities to discover how programming constructs work. Additionally, players use their *conditional logic* to decide whether or not to complete levels using the shortest path to the teleport square or to capture all collectible objects before doing so. Players visualise which programming constructs to use by using *algorithmic thinking* and they can *simulate* and *debug* their solutions at any point during their game-play. The learning is delivered through learning experience by designing and testing solutions which is described in the literature as the ideal way of learning computer programming (Mayer 1981; Jenkins, 2002; Feldgen & Clua, 2004; Kinnunen & Simon, 2012). As a result, the game demonstrates a well-grounded implementation of the *interaction – feedback loop* model.

As argued earlier, *Program Your Robot* is also developed through the proposed guidelines presented in Chapter 2 Section 2.5. How the specifications in the guidelines are followed during the development of the game is discussed below.

The *institutional insight* in *Program Your Robot* is ensured by mapping the learning content in the game onto part of the computer programming curriculum taught within the Computer Science department of University of Greenwich. The programming constructs introduced in the first four weeks of the computer programming curriculum (i.e. programming sequence, methods, decision making and loops) are accessible as in-game elements. The game does not attempt to change the learning objectives set for the computer programming course and it only uses a part of the current curriculum to achieve a limited number of learning outcomes.

Program Your Robot is intended to drive players to analyse, visualise and practise the correct use of computer programming constructs and thus aims for higher level learning goals (i.e. analyse, create, apply and evaluate) through a *constructivist learning structure*. The

conceptual integrity of the game is based on practising CT skills from the game experience and not on teaching any language-specific programming code. In other words, the game is not designed as an operational refinement approach that describes actions in terms of pseudo-code. The skills and tactics acquired from the game-experience can be transferred to programming code but currently this needs the help of an instructor. The game is also concerned about *scaffolding strategies* as the whole idea behind constructing solutions is to think computationally.

Further to this, the game does not focus on a specific gender, and players do not need to have any programming knowledge to play the game. The *setting of the game* is designed to be *gender and expertise neutral* and specifically supports first year programming students because a) the theme of the game (i.e. a robot trying to escape from a maze) is not male or female oriented; b) players do not need to have prior computer programming knowledge to play the game. More importantly, the game is free and online – therefore accessible at all times (http://www.programyourrobot.com). The score system in the game encourages some level of competition in the game as players can submit their score to a high score list when they really do well in the game. This type of approach leads to a non-compulsory *competition* among those players who want to compete with each other and thus provides a limited interaction among players. As the competition in the game recognises a gender and expertise neutrality, players are not obligated to submit their scores to the high score list.

As a result, *Program Your Robot* is a solid implementation of the *interaction – feedback loop* game model as well as the proposed guidelines that is derived from the existing research in this area.

3.4 Summary

This chapter refined the original main research question of this research in order to provide more focus and ground it into a modelling structure. An innovative model (i.e. *interaction – feedback loop* game model) for learning how Computer Science programming constructs work through game-play, based on the body of work existing in this area, was then presented. Further to this, the chapter establishes the premise that games designed to encourage players to think computationally through puzzle-solving game-play, in an environment contextually based on Computer Science programming concepts, are conducive to learning programming. By way of illustration a serious game named *Program Your Robot* was developed which does not only focus on the operational level of abstraction and skill acquisition in computational thinking, but also contextualises four introductory programming constructs (i.e. programming

sequence, functions, decision making, loops) into the game-play.

Having explained the rationale behind *Program Your Robot* and the reasons why the game was designed as it is, a research approach and a research design is needed in order to answer the main research question of this research. In other words, it is required to assess whether or not this game is an educationally effective solution for learning how a limited number of computer programming constructs work at the computational thinking level. The next chapter describes this research approach and design as well as divides the main research question into several sub questions that could be investigated separately in an experimental structure.

CHAPTER 4

RESEARCH METHODOLOGY

Having refined the main research question, developed a game model and an implementation of this, it is required to structure the research approach and design in order to answer the main research question. This chapter discusses the research approach, and divides the main research question into several sub research questions in order to ground it into an experimental research structure. The chapter first explains the approach followed in this research and discusses how the main research question is divided into several parts. Additionally, the chapter discusses why survey questionnaires were used as the main form of collecting demographic data from target subjects. The rationale behind each question asked in the questionnaires and what is expected to be learned from these is also specified. Moreover, the outline regarding how each question in the questionnaires would be analysed and how the data would be used in the statistical analysis of studies are also covered in this chapter.

Section 4.1 describes the approach of this research as a combination of a qualitative methodological framework (i.e. phenomenography) and a quantitative case study. This section argues how and why a qualitative research approach was blended into a quantitative case study design. It also explains how it aims to provide empirical evidence through a blended qualitative research approach whereas this is often provided through quantitative methods. Section 4.2 revisits the main research question of this research and how this question is divided into eight different sub research questions. The rationale behind each sub research question and their relationship to the main research question is also discussed in this section. Section 4.3 explains the pre-study questionnaire design as well as the rationale behind each question. Additionally, the plan regarding how pre-study questions would be used in the statistical analysis of studies and the pathway used to validate the findings is also revealed in this section. The final section of this chapter which is section 4.4 follows a similar structure to Section 4.3 but rather than the pre-study questions, the rationale behind the post-study questions is investigated.

4.1 Blending phenomenography to a quantitative research approach

The main goal of this research is to evaluate *Program Your Robot* and its underlying game model (i.e. *interactive – feedback loop*) as a potential learning approach for learning

computer programming constructs as well as developing abilities in computational thinking (CT). To achieve this, it is crucial to guide this study through the lens of a theoretical research approach in order to accurately assess students' perception of learning outcomes regarding how computer programming constructs work. This section explains the research approach used in this study and discusses how a quantitative case study is combined with a qualitative research approach in order to measure whether or not *Program Your Robot* can be an educationally effective tool for introductory programming students.

This research is blended in the view of learning within an empirical research framework that was first introduced in Marton's seminal work (1981) as "Phenomenography". Phenomenography is "an approach to educational research that seeks to describe a phenomenon in the world as others sees them, the object of the research being variation in ways of experiencing the phenomenon of interest" (Marton & Booth, 1997). This qualitative research approach describes that an empirical study is never separated from the object of perception or content of thought (Richardson, 1999). According to Bruce *et al.* (2004) "a fundamental assumption underlying phenomenographic research is that there is finite number of qualitatively different understandings of a particular phenomenon". In other words, Phenomenography aims at analysing experiential descriptions, people's conception of reality that can be considered as "true" and therefore, the definition in this type of research falls in between the natural science domain and traditional social sciences (Richardson, 1999). In his seminal work, Marton (1981) discussed that the realness of "reality" is not independent of people's perception and in the same way the realness of an experience is a part of "reality" because it is not possible to separate what is experienced from the experience itself. He labels making statements about reality as the "first-order perspective"; and people's description, analysis and understanding of experiences as the "second-order perspective" which is where the phenomenographic research is defined. In other words, Phenomenography does not make statements about the world but it makes statements about people's understanding of the world. In his later publications, Marton (1986) defended the formulation of the second-order perspective as different ways in which people experience, understands or conceptualise various aspect of reality in an autonomous sense that is simply not possible to be derived from reports from the first order perspective especially when "research is directed towards experiential descriptions and learning" (Marton & Booth, 1997). More importantly, phenomenographic research results refer to a certain aspect of reality or inter-subjective (commonly agreed) meaning of that aspect. This is to say that a phenomenographic research can detect a phenomenon, an aspect of reality that is experienced or conceptualised repeatedly from the

perception of participants* (Marton, 1994). To characterise how a phenomenon is understood and/or perceived is by definition a qualitative question. Thus, phenomenograpic research is defined as an empirical approach as it aims to discover the qualitatively different ways people experience, conceptualise, analyse and understand various aspect of a phenomenon (Masters, Ramsden & Stephanou, 1992). Moreover, Marton & Booth (1997) stated that phenomenographic research describes a phenomenon from "the report of inferences of subjects" and suggest that this approach is strongly recommended to be used in educational research.

Understanding students' perception regarding a serious game (i.e. *Program Your Robot*) and what learning outcomes can be obtained from their game-play experience is the main goals of this research. Considering this, phenomenography has been chosen as a methodological framework for this research mainly because of three important reasons:

Firstly, within the structure of this research, students' perception and experience cannot be separated from what they are learning. As learning through playing games involves an immersive interaction and feedback, it is not possible to separate what is being learned from the students' game experience as these two concepts are not autonomous. Consistent with the definition of phenomenography, the subject (student) and the object (learning experience) in this study are not independent from each other (Marton 1981; Marton 1986). Further to this, Ornek (2008) argued that the framework of phenomenography is one of the best methods that can be used when investigating "learning experience" and highlighted that phenomenographic research allows students to express their thoughts and feelings as they experience or understand a selected concept.

Secondly, although being qualitative, phenomenographic research is empirical and suitable to provide evidence on whether or not a game playing activity can develop students' understanding, perception and experience on various introductory programming constucts. Orgill (2002) clearly indicated that phenomenography is an empirical study because researchers using this methodological framework are not studying their awareness or understanding regarding a phenomenon but rather they examine the awareness and reflection of their subjects in an open-minded way. This is to say that phenomenographic research is always designed independent of researchers' own perspective and thus the results can clarify the different ways the same phenomenon has been experienced by a group of people regardless of the perception of the researcher. In her seminal work Trigwell (2000) reported that phenomenography is different from all the other research approaches in terms of being experimental, qualitative, focused on second-order perspective and internally related (subject

* The words participants and students are used interchangeably in this research to refer to the target group of the research.

and object are not independent) and thus this makes it an appropriate research approach for this particular research.

Finally, Marton (1986) highlighted three crucial points in phenomenographic research: the aspect of learning (the qualitative differences between learning outcomes), learning of concepts (the phenomenon that is being learned) and people's conception on various aspects of life. The first two lines of Marton's work are related to this research as students' perception of learning experience regarding a game based learning activity, as well as the effect of this on their skills in computational thinking, is the main focus of this research. It aims to see the multiple different conceptions students would have regarding learning programming concepts through playing a game. It is aimed to investigate whether or not a student's experience would encourage them "to develop conceptual understandings" which is one of the biggest benefits of phenomenographic research (Entwistle, 1997). Thus, it is anticipated that the different conceptions students would develop during their game-play could be beneficial to understand whether or not *Program Your Robot* is an educationally effective tool to support the learning of programming constructs at the computational thinking level.

Despite the fact that this research is guided through phenomengraphy, there are considerable differences that separate this study from regular phenomenographic research. The most distinct difference is the data collection methods used in this study. The primary way of data collection in a standard phenomenographic research is semi-structured interviews or open-ended questions (Marton 1986; Marton 1994). This data collection method allows a great advantage to phenomenography as researchers never need to bracket their own theories and preconceptions, and thus exclusively focus on the experiences of the participants. Although Marton & Booth (1997) argued that there are other ways of analysing how people conceive different aspects regarding a phenomenon (i.e. a combination of close and open-ended questions), using interviews for data collection has been carried out as the predominant way to reveal participants' way of experiencing a concept (Mann, 2010). Richardson (1999) stated that Marton's idea of using interviews for data collection is a straightforward method developed out of common sense considerations regarding learning and teaching. In a regular phenomenographic research, researchers ask various questions to participants in order to elaborate their experiences and what they mean by certain concepts. The interviews follow a protocol where participants are encouraged to reflect their experiences regarding a concept through a series of open-ended initial and follow up questions to stimulate discussion (Mann, 2010). Most phenomenographic research also involves pilot interviews in order to decide whether or not the initial questions reveal the sorts of experiences necessary to address the

focus of the research (Bowden, 2005). However, an *interview only* structure raised a lot of criticism especially when the researcher has a position within participants' own academic institution (Richardson, 1999; Orgill, 2002). As an example, Richardson (1999) argued various ethical issues (i.e. participant's disclosure, meta-awareness of being interviewed) about how much pressure is put upon to both the interviewer (researcher) and the interviewees (participants). More recently, Bowden (2005) pointed out more criticisms to this and argued that "when data collection has relied only on interviews, no other evidence exists beyond the transcripts to inform the analysis process".

To overcome the criticism raised by the previous studies, this research is blended with phenomenography and a quantitative data analysis. Although, this research collects participants' perceptions and follows phenomenography (i.e. this research is non-dualist, structured qualitatively, focused on second perspective and internally related), the entire research focuses on quantitative hypothesis-testing rather than qualitative hypothesis-generating. On one hand, it was aimed to evaluate the second perspective – participant's perception of learning – and the outcome of this (i.e. learning programming constructs through playing a game) whereas on the other hand close-ended questionnaires were used to obtain data from participants rather than the classic data collection methods of phenomenographic research (i.e. the semi-structured interviews and hypothesis gathering process). As a result, this research differs from a regular phenomenographic research in terms of methods used to obtain and analyse data. As Bowden (2005) indicated a phenomenographic analysis involves discovery and construction, and it is an inductive way of working from the data to the results (bottom up logic) rather than a way of constructing and testing a hypothesis (top down logic). This is a crucial point where this study is separated from a regular phenomengraphic research as it is aimed to construct and test a series of hypothesis similar to a quantitative experimental research. In other words, this research uses quantitative data analysis for hypothesis testing to demonstrate an outcome. Yet, the research follows a qualitative approach because it is not intended to show the richness of data (Bowden, 2005) but rather aimed at demonstrating the variation of which there is clear evidence that students learn how programming constructs work from *Program your Robot*. Hence, this perceptual research falls in between being a phenomenographic research and a quantitative case study.

4.2 Main research question and sub research questions

This section outlines the eight sub questions generated from the main research question in

order to investigate intrinsic motivation, perception of knowledge, visualisation of programming constructs and problem solving abilities of students.

As discussed in Chapter 3 Section 3.1, the (refined) main research question of this research is:

"Can a serious game be designed to support the development of computational thinking skills through the medium of learning how key introductory programming constructs work?"

There are two key aspects of the main research question. The first one is *learning how key introductory programming constructs work* and the second one is the *development of computational thinking skills*.

The first aspect of the main research question i.e. *learning key introductory programming constructs work* is investigated under two important subtitles. These are the *motivation to learn introductory programming constructs* and the *actual learning of the introductory programming constructs*.

As discussed in Chapter 2 Section 2.1, one of the biggest problems in the education of introductory computer programming is the low motivation of students. Despite the efforts to improve the education of computer programming over the years, the literature in this field still reports that novice students have low motivation (Jenkins, 2001; Bennedsen & Caspersen, 2007; Tsukamoto *et al.*, 2012) and difficulties in learning computer programming (Lister et al, 2004; Hwang, 2012). Various studies in game based learning provided evidence that games and game-like environments are successful in motivating students in learning computer programming constructs (Leutenegger & Edgington, 2007; Muratet *et al.* 2011; Sung *et al.* 2011). Further to this, *Program Your Robot* was designed in a way that the learning material in the game is an integral part of the game-play. However, it is uncertain whether or not the game would encourage students to learn more about computer programming constructs and increase their motivation towards this. In order to assess this statement, it is necessary to investigate whether or not *Program your Robot* would a) motivate students in learning computer programming constructs and b) change students' attitude towards learning computer programming after their game-play.

In addition to the motivational aspect of computer programming, it is required to measure whether or not students can learn how programing constructs work from playing *Program Your Robot*. As discussed in Chapter 3 Section 3.3, the first four computer programming

constructs introduced in the computer programming course at the University of Greenwich (i.e. programming sequence, functions, decision making and loops) was integrated to *Program Your Robot* as in-game elements. Hence, it is required to assess students' perception of knowledge in these programming constructs in order to answer the first aspect of the main research question (i.e. *learning how key introductory programming constructs work*).

As a result, the following sub research questions were created:

Research Question 1:

Is there a difference between students' attitude to learn computer programming through playing games before and after they play *Program Your Robot*?

Research Question 2:

Is there a difference between students' intrinsic motivation to learn computer programming before and after they play *Program Your Robot*?

Research Question 3:

Is there a difference in students' understanding of "programming sequence" in computer programming before and after they play *Program Your Robot*?

Research Question 4:

Is there a difference in students' understanding of "functions" (methods) in computer programming before and after they play *Program Your Robot*?

Research Question 5:

Is there a difference in students' understanding of "decision making" in computer programming before and after they play *Program Your Robot*?

Research Question 6:

Is there a difference in students' understanding of "loops" in computer programming before and after they play *Program Your Robot*?

The second aspect of the main research question is the *development of computational thinking skills.* As stated in Chapter 2 Section 2.2*,* computational thinking is a topic that has arisen from the field of Computer Science as a problem solving approach concerned with

conceptualizing, developing abstractions and designing systems which overlaps with logical thinking and requires concepts fundamental to computing. As discussed in Chapter 2, Section 2.2.1, the current literature divides computational thinking into five cognitive skills: conditional logic, algorithmic thinking, simulation, debugging and socialising (Wing, 2008; Ater-Kranov *et al*, 2010; Perkovic *et al.*,2010; Dierbach *et al.*, 2011; Berland & Lee, 2011;). Currently, introductory programming students are expected to develop these skills during their lectures whilst also trying to learn the syntax of a new programming language. Considering this, these skills cannot be accurately measured in the pre-study of this research as the above cognitive abilities are not explicitly taught to students. To resolve this issue, the literature regarding computational thinking was investigated in order to identify the nature of computational thinking so that this can be blended into the pre-post study structure of this research.

Ater-Kranov *et al.* (2010) argues that problem solving and critical thinking are the two most ubiquitously agreed computational thinking skills in the literature. Lee *et al.* (2011) undertook a similar research and examined computational thinking in three aspects: analysis, abstractions and automation. They concluded that computational thinking is a problem solving approach as the abstraction, automation and analysis steps occur on a continuum through the use-modify-create progression. Additionally, Dierbach *et al.* (2011) reported that the most common set of computational thinking skills are identifying and applying problem decomposition as well as developing computation models to solve problems.

Based on the conclusion of these recent studies, this research divided computational thinking into two important areas: *problem solving skills* and *the ability to visualise constructs from given problems*. *Problem solving skills* refers to the analysis and decomposition of problems into individual segments as well as the ability to develop models to deal with these problem segments. *Visualising constructs from given problems* is related to making analysis, abstraction and automation as when students understand which programming constructs to use in order to solve a specific problem they perform an analysis, subordinate concepts and connects any related concepts as a group and finally they perform an automated deduction because a possible solution to a specific problem often needs to be iterated and generalised.

As a result, in order to answer the second aspect of the main research question (i.e. *development of computational thinking skills*) the following sub research questions were created:

Research Question 7:

Is there a difference between students' perception of their problem solving abilities before and after they play *Program Your Robot*?

Research Question 8:

Is there a difference between students' perception of their ability to visualise programming constructs from given problems before and after they play *Program Your Robot*?

Despite the fact that the above research questions can investigate a considerable part of computational thinking, it does not cover previously defined computational thinking skills (i.e. conditional logic, algorithmic thinking, simulating solutions, debugging and socialising). These skills were explicitly investigated in the post-study questionnaire of this research. However, a research question for each category was not created because students were not taught any of these skills explicitly and therefore, it was not possible to ask them to rate their own computational thinking skills before they played the game.

In addition to these research questions, an extra research question (i.e. ninth research question) was added to investigate whether or not the participants' perception regarding how difficult computer programming was would change before and after they play *Program Your Robot*. The research question is as follows:

Research Question 9:

Is there a difference between students' perception of difficulty of computer programming before and after they play *Program Your Robot*?

Although the experimental structure of this research is explained in the next chapter (Chapter 5 Section 5.1), it is necessary to explain the initial structure of this research at this point in order to discuss how the above research questions would be merged into the experimental design of the research. Originally three different studies were designed to be conducted in this research where two of these were conducted in higher education and one of them was conducted in a public girls' school. Although questions asked to students were slightly different, the structure of all studies was identical, that is, answer a pre-study questionnaire, students then play *Program Your Robot*, followed by a post-study questionnaire.

The reason why a pre and post study structure was followed is because it was designed to observe students' perception of their knowledge differences in programming constructs as well as their perception regarding learning programming constructs through game-play before and

after they played *Program Your Robot*. In addition to these, it was designed to investigate the impact of *Program Your Robot* on the computational thinking skills of students (i.e. conditional logic, algorithmic thinking, simulating solutions, debugging and socialising). As this study follows a phenomenographic approach, participants were surveyed both in the pre and the post study and were expected to reflect their experiences regarding the game environment.

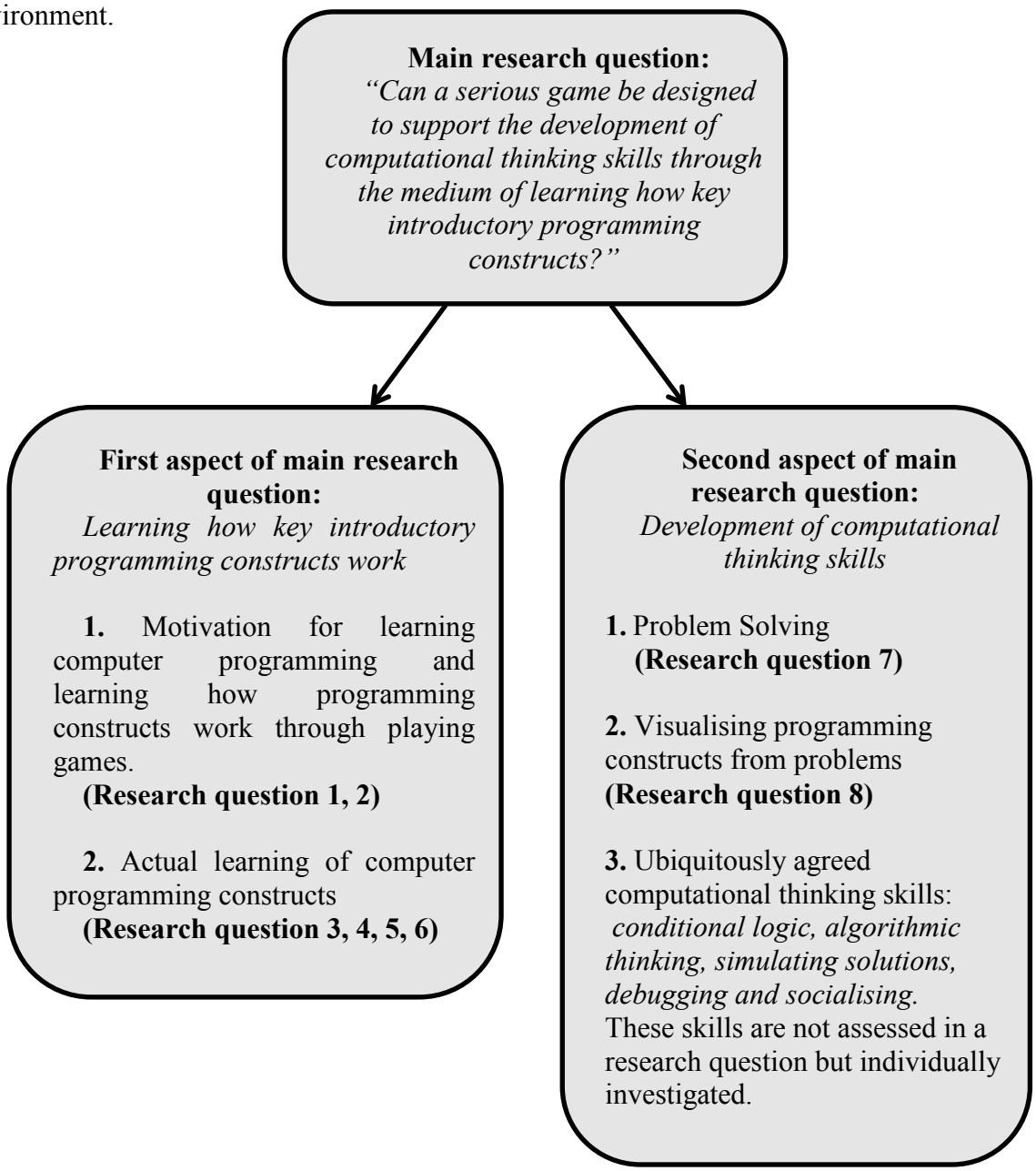

Figure 4.1 – Showing how the main research question divided into two different parts and which sub questions belong to which part of the main research question.

Figure 4.1 illustrates the two main key areas of the main research question as well as which

of these areas the listed eight sub research questions belong. From this point on, the sub research questions listed above are simply referred to as research questions of this study as they are merged to a pre – post study structure. How these research questions would be analysed and what questions were asked to students in this pre – post study is described in the following sections.

4.3 Pre-study questionnaire

As indicated in Section 4.2, the experimental structure of this research fits into a pre-study post-study design. This section focuses on the pre-study part of the research and identifies the rationale of each question used in the pre-study questionnaire. Additionally, the outline of how the pre-study questions would be used in statistical analysis and validation of results is also discussed in this section.

There are four sections in the pre-study questionnaire: *personal information*, *institutional information*, *background in computer programming* and finally *video games and learning*. The *personal information* part collects data from participants to classify them according to their age groups, gender, ethnicity and mathematical qualifications. The *intuitional information* part explores whether or not participants were considering giving up their degree programmes. The *background in computer programming* part investigates participants' knowledge and background in computer programming constructs. Finally, *video games and learning* measures the attitude of participants to learning computer programming constructs through playing games. Each of these sections is discussed below.

4.3.1 Personal information

The first part of the pre-study questionnaire includes collecting personal information from the participants. This section of the pre-study questionnaire was designed to collect data from participants about their a) University ID number (or a randomly distributed number given to participants at the beginning of the study); b) gender; c) age range; d) ethnicity (according to UK government standards); e) degree programme and f) highest mathematical qualifications they achieved. As mentioned earlier however, three different studies were designed to be conducted as part of this research. These studies were conducted in Kyrenia, Cyprus; Greenwich, UK and Dartford, UK respectively where the first two were completed in higher education and the last one in a public girls school. Each of these studies targeted different subjects in different geographical locations at different times and therefore, it was not always

possible to collect all of this data consistently in all three studies. In order to explain the differences and similarities among these studies, they are referred to as the Cyprus, the Greenwich and the PGS studies from this point onwards.

The target group of the Cyprus and the Greenwich studies were first year Computer Science (or a related degree) students who were registered on an introductory programming course. The PGS study was an extension of the other two studies and conducted in a public girls school in order to observe the reflection of the school girls to *Program Your Robot* to identify if the game could be used below degree level without modification. As the target groups were completely different from one another, it was not possible to ask the same set of personal information questions in all three studies. Additionally, there were considerable differences between the students in the Cyprus and the Greenwich studies (i.e. ethnicity, mathematical qualification and age-range) as these studies were conducted in different countries.

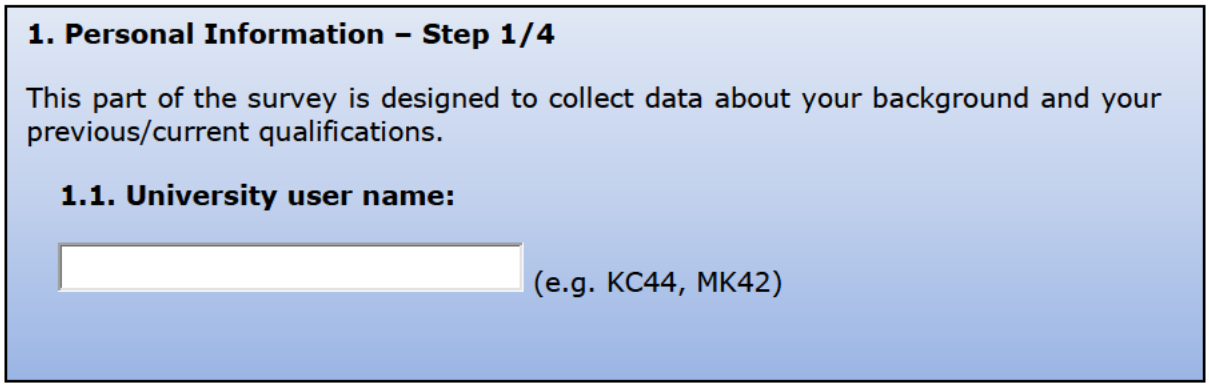

Figure 4.2 – username question in the pre-study questionnaire.

As shown from Figure 4.2, the first question asked to participants in the studies was their university user name or a randomly distributed number given to participants at the beginning of the studies. In the Cyprus and in the PGS studies, the identity of participants was kept confidential by ensuring that they received a random unique number from their teaching staff at the beginning of the study. Students used this number for their pre and post questionnaires so that it would be possible to match the responses given to the pre-study with the responses given to the post-study without identifying individuals. The teaching staff involved in the studies distributed and collected these unique numbers and, the author of this thesis was never aware which random number was distributed to which participant. Although this is a very efficient way of keeping the identity of participants anonymous, this method was not used in the Greenwich study mainly for two reasons a) participation in this study was considerably higher compared to the other studies; b) it was essential to have control as the study was

conducted in ten different computer labs simultaneously. To work out this problem, participants in the Greenwich study were asked to use their username rather than a random unique number. Students were well informed that their usernames and their answers would never be shared with anyone else.

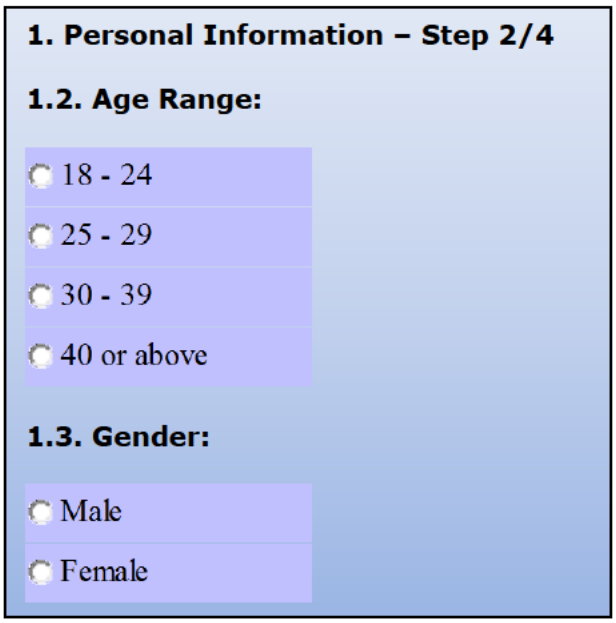

Figure 4.3 – Age-range and gender questions in the pre-study questionnaire.

Having entered their University username/random unique number, participants were asked to select their age-range and gender. The age-range and gender questions were asked of participants for two main reasons a) to categorise participants according to their gender and age-range b) to investigate whether or not age-range and gender would have an influence on the findings of these studies. In other words, it is intended to investigate the impact of age and gender on participants' responses and the statistical outcomes of the studies. This is related to the internal validity of the Cyprus and the Greenwich studies particularly to regression threat which is further discussed in Chapter 5 Section 5.4.

Finally, the age range and gender of participants were collected in the Cyprus and the Greenwich studies only and were not collected in the PGS study as all participants in the PGS study were 15 years old school girls. As the age and gender of participants were constant, a regression threat analysis for the PGS study was not performed. The details regarding the internal validity of the PGS study is further discussed in Chapter 7 Section 7.3.

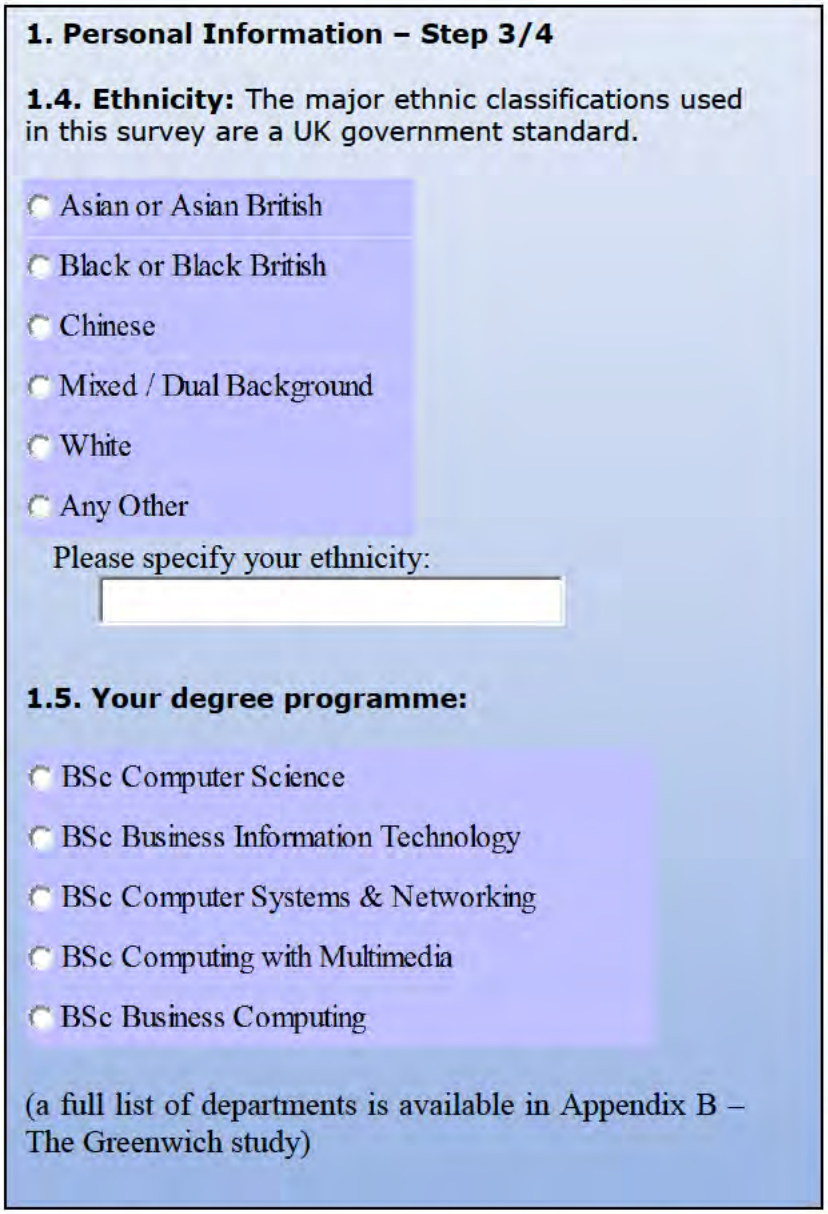

Figure 4.4 – Ethnicity and degree programme questions in the pre-study questionnaire.

Figure 4.4 illustrates how participants' ethnicity and degree programmes were collected in the pre-questionnaire. The ethnicity of participants were collected in the Greenwich and the PGS studies but ignored in the Cyprus study due to the reason that a UK government standard ethnic classification was used and this does not apply to Cyprus because the ethnicity of students in Cyprus is different than the ethnicity of students in UK. Further to this, participants were asked to select their degree programme in the Greenwich studies whereas this question was removed from the pre-questionnaire in the PGS and the Cyprus study because a) all participants were registered to the "BSc in Information Technology" in the Cyprus study and b) all participants were studying for level 2 Information Communication Technology (ICT) in

the PGS study.

It was planned to investigate the effect of ethnicity and degree programme of participants on the findings of the studies in a similar way to the age-range and gender. However, this decision was eventually abandoned as the Greenwich was the only study that the degree programme students were studying for could be captured. Hence, even if an impact of degree programme would be identified in the Greenwich study, this could not be accurately generalised as the same type of analysis could not be performed on the results of the other studies. Another vital point is that the target group of the PGS and the Greenwich study is not the same and therefore, the impact of ethnicity of target groups cannot be measured accurately due to the vast differences in participants (i.e. age range and gender). In this case the ethnicity and degree programme of students were collected for observation purposes and would not be used in the statistical analysis of the studies or validity of findings.

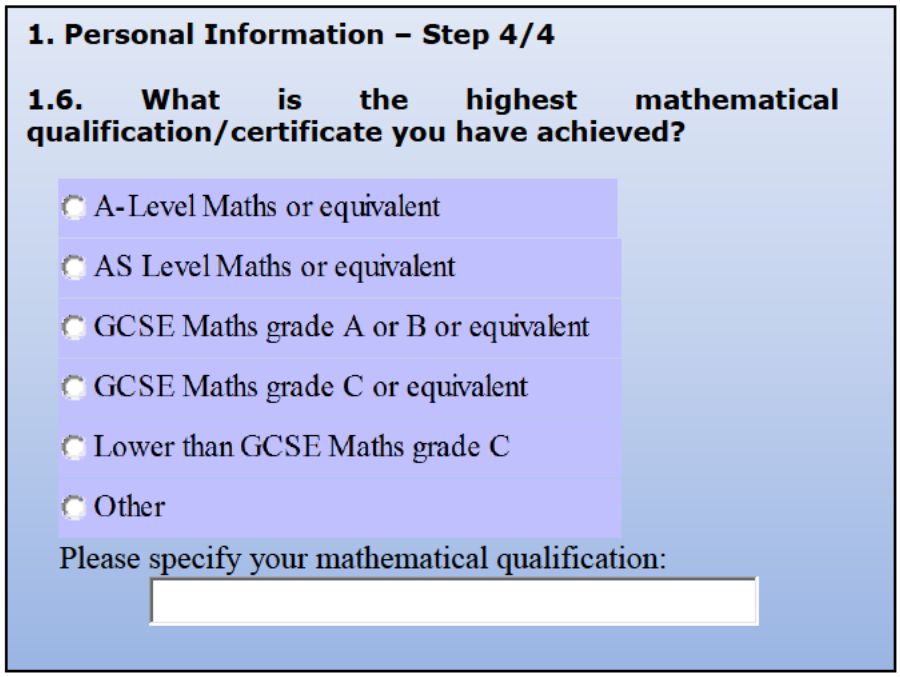

Figure 4.5 – Mathematical qualifications question in the pre-study questionnaire.

The final section of the personal information part included obtaining the highest mathematical qualification participants achieved. Similar to the participants' degree programme, this question was only asked in the Greenwich study and ignored in the other two studies because a) a General Certificate of Secondary Education (GCSE) level grade is not a requirement to study a Computer Science or a similar degree in Cyprus. A wide variety of different mathematical degrees are recognised in Cyprus and it was simply not possible to list

all of these in the questionnaire of the pre-study; b) all participants in the PGS study were 15 years old school girls and it was simply inappropriate to ask them about their mathematical qualifications.

Figure 4.5 illustrates how the highest mathematical qualification of participants was collected in the Greenwich study. This data was obtained because it was designed to measure whether or not mathematical qualifications have an effect on how much participants would learn from *Program Your Robot*. In other words, it was intended to identify whether or not those with higher mathematical qualifications learn more about how programming constructs work from the game environment compared to others. The results obtained from this analysis would provide an indication of whether or not mathematical qualifications of students have an effect on the outcome of the Greenwich study. Hence, similar to age-range and gender, mathematical qualifications of participants are related to the internal validity of studies particularly to the regression threat which will be explored in Chapter 7 Section 7.1.4.

4.3.2 Institutional information

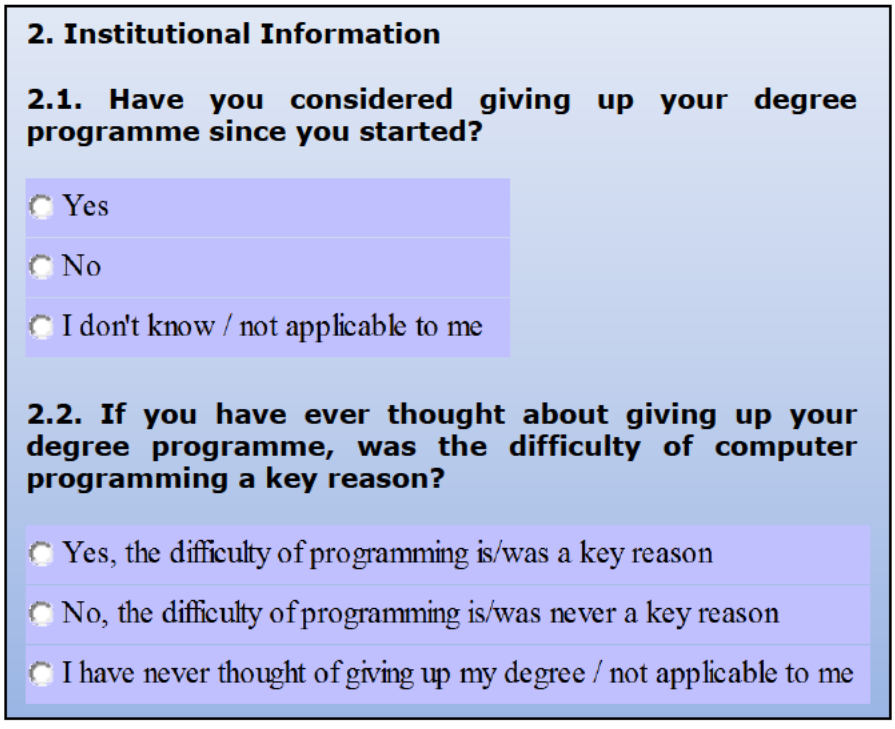

Figure 4.6 – Institutional information part of the pre-study questionnaire.

The intuitional information part of the pre-study questionnaire investigates two important points in this research a) to examine whether or not participants are considering giving up their

degree programme b) to identify whether or not the difficulty of computer programming is a key reason if participants were considering giving up their degree programme. Figure 4.6 shows how the intuitional part of the questions was asked to participants in the pre-study questionnaire.

Both questions in the institutional part of the pre-study questionnaire were asked in the Greenwich and the Cyprus studies but not included in the PGS study due to the reason that the target group in the PGS was not registered to a Computer Science or a similar degree programme. In addition to this, the Cyprus and the Greenwich studies were scheduled to be conducted five weeks after participants started to their computer programming courses. Therefore, the questions listed in this section were asked to participants soon after they started to their degrees.

4.3.3 Background in computer programming

This section is arguably the most important section of the pre-study questionnaire as it is aimed to collect data about participants' past experiences in computer programming, their knowledge level in computer programming according to their own perception, how difficult they find learning computer programming and finally their intrinsic motivation to learn computer programming. Additionally, participants' current perception of their problem solving abilities and their ability to visualise programming constructs is also collected in this part of the pre-study questionnaire.

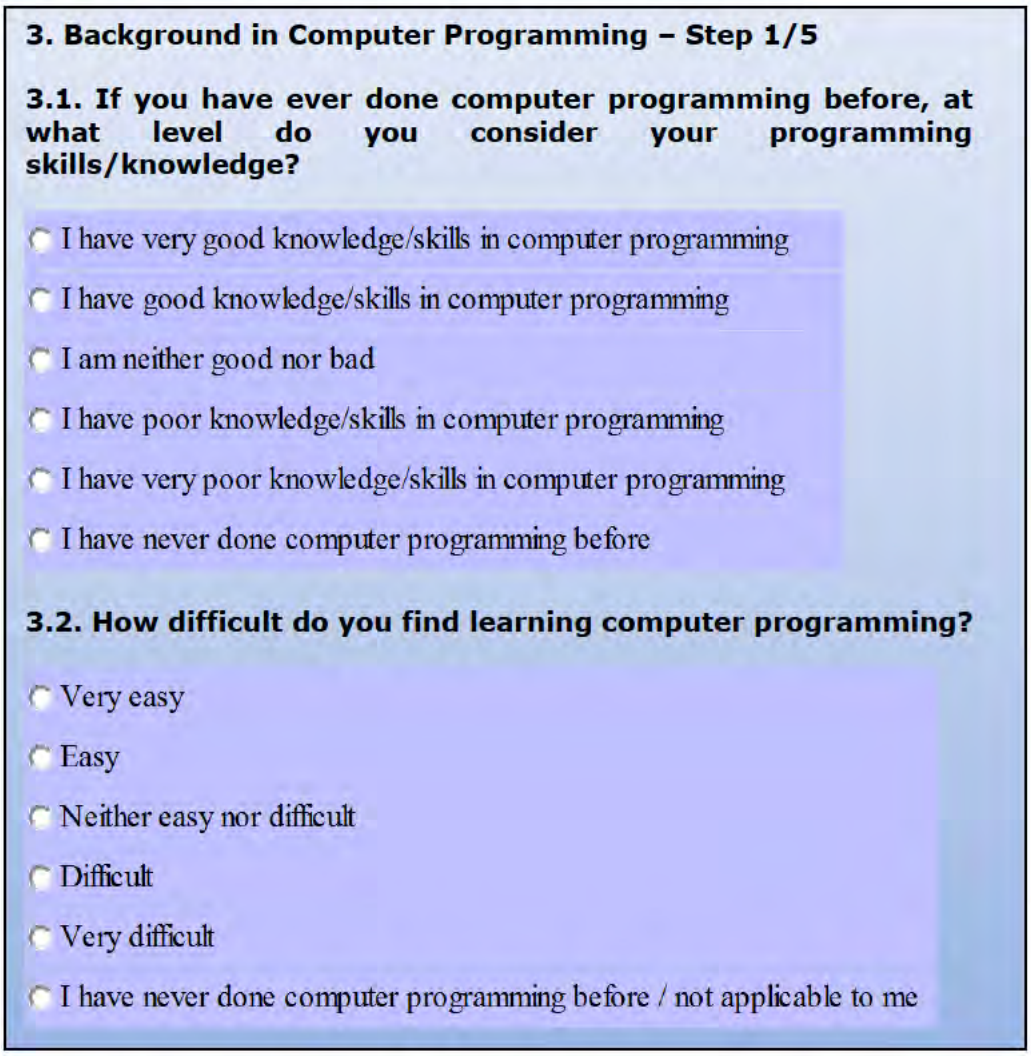

Figure 4.7 – Collecting data on participants' computer programming knowledge and the difficulty of learning computer programming in the pre-study questionnaire.

As illustrated in Figure 4.7, participants were first asked to rate their current knowledge in computer programming (if they have any) according to their own perception. Having done this, participants were asked to rate how difficult they find learning computer programming. Question 3.1 was asked to observe how good participants think their knowledge is in computer programming. Although this question is not explicitly related to any research question stated in Section 4.2, it would enable one to identify the big picture on whether or not participants think *Program Your Robot* improved their general knowledge in computer programming. Question 3.2 is explicitly related to the ninth research question that would be examined in the Cyprus and the Greenwich study (i.e. is there a difference between students' perception of difficulty of computer programming between pre and post study?). Both questions 3.1 and 3.2 were asked

in the pre and post-study questionnaires of the Cyprus and the Greenwich studies. The answers given to these questions would then be matched to observe whether or not there is a significant increase during these studies regarding participants' perception of their computer programming knowledge as well as the difficulty of computer programming. Question 3.2 does not exist in the pre-study questionnaire of the PGS study simple because they were not learning computer programming.

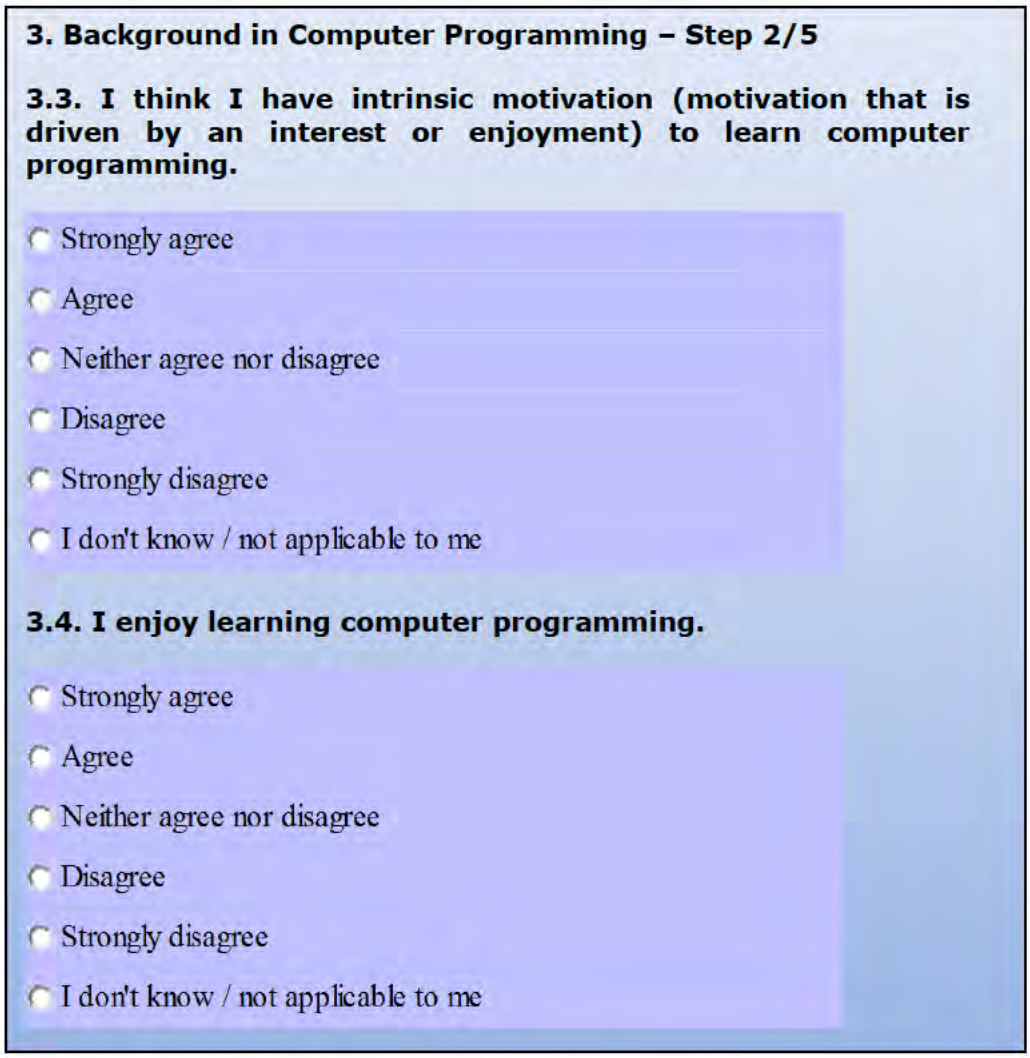

Figure 4.8 – Collecting data on participants' intrinsic motivation and enjoyment in computer programming in the pre-study questionnaire.

As shown in Figure 4.8, questions 3.3 and question 3.4 explore the intrinsic motivation of students to learn computer programming which is directly related to the second research question of this research (i.e. Is there a difference between students' intrinsic motivation to learn computer programming before and after they play *Program Your Robot*?). The intrinsic

motivation and the enjoyment in learning are similar concepts and there are two important reasons why this concept was designed to be analysed in two different questions in the pre-study questionnaire of the Cyprus and the Greenwich studies: a) to observe whether or not participants would answer these related questions consistently b) if the results obtained from the first question (question 3.3) in one study is significantly different than the results of the same question in the other study, then the observations obtained from the second question (question 3.3) would be matched to debate the potential reasons behind the differences. Similar to question 3.2, question 3.4 does not exist in the PGS pre-study questionnaire due to the reason that participants were not registered to a computer programing course.

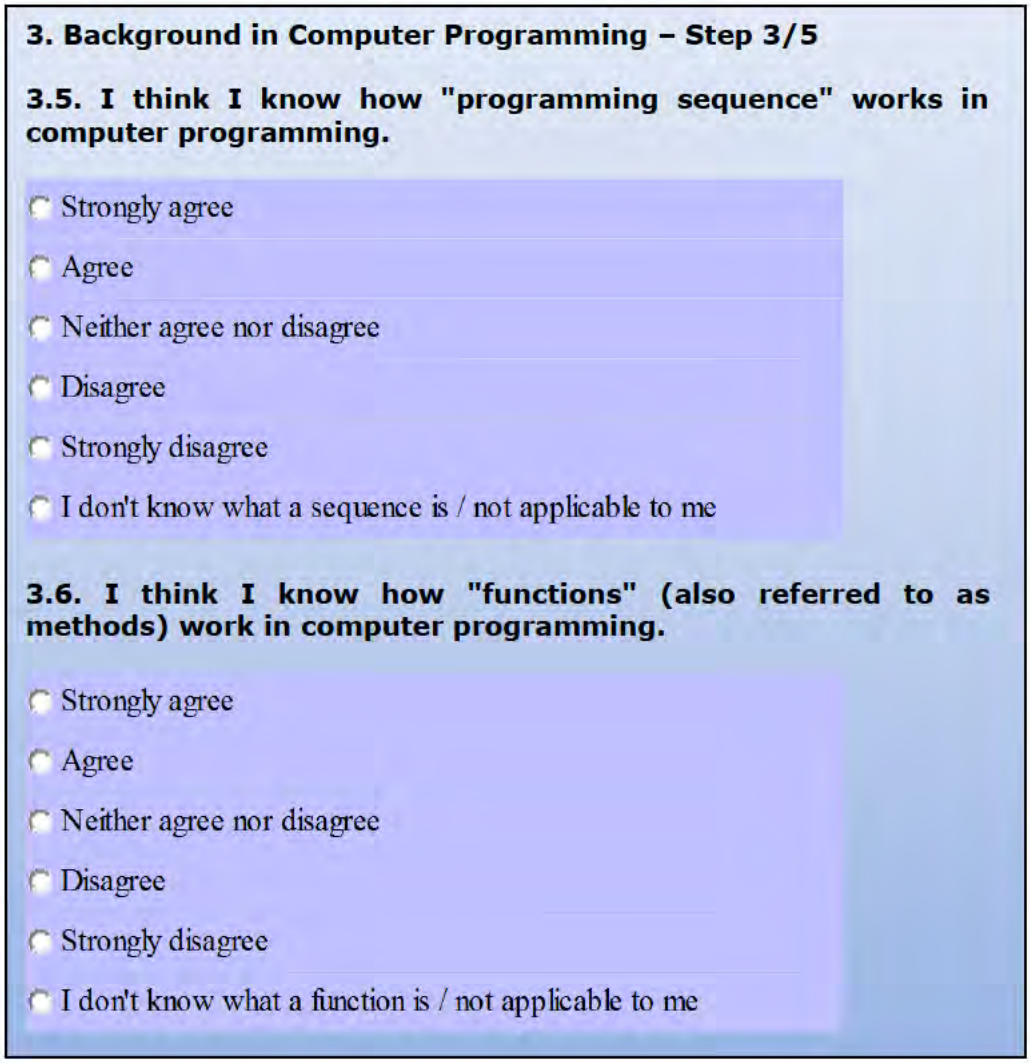

Figure 4.9 – Collecting data on participants' knowledge in programming sequence and functions in the pre-study questionnaire.

Having asked their intrinsic motivation to learn computer programming, participants were requested to rate their knowledge in four different programming constructs (i.e. programming sequence, functions, decision making and loops) that are an integral part of the game-play in *Program Your Robot*. Figure 4.9 and Figure 4.10 illustrate how these programming constructs were asked to participants in the pre-study questionnaire. The same set of questions listed in these figures would also be asked in the post-study questionnaire after participants played *Program Your Robot*. The answers of participants would then be matched and the third, fourth, fifth and sixth research questions would be investigated according to the data captured (i.e. is there a difference in students' understanding of programming sequence, functions, decision making and loops in computer programming between pre and post study?). The questions listed in Figures 4.9 and 4.10 were asked in the pre-study questionnaire of all studies.

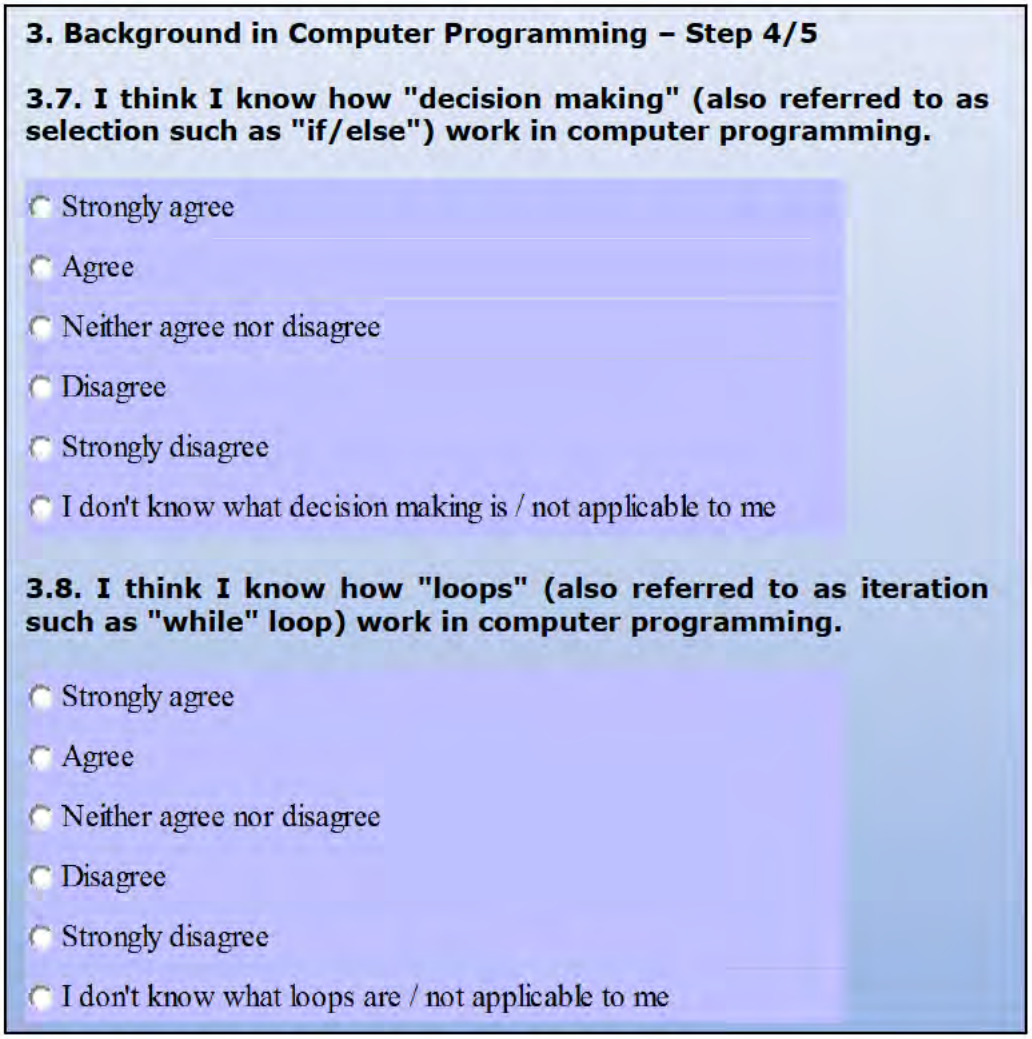

Figure 4.10 – Collecting data on participants' knowledge in programming sequence and functions in the pre-study questionnaire.

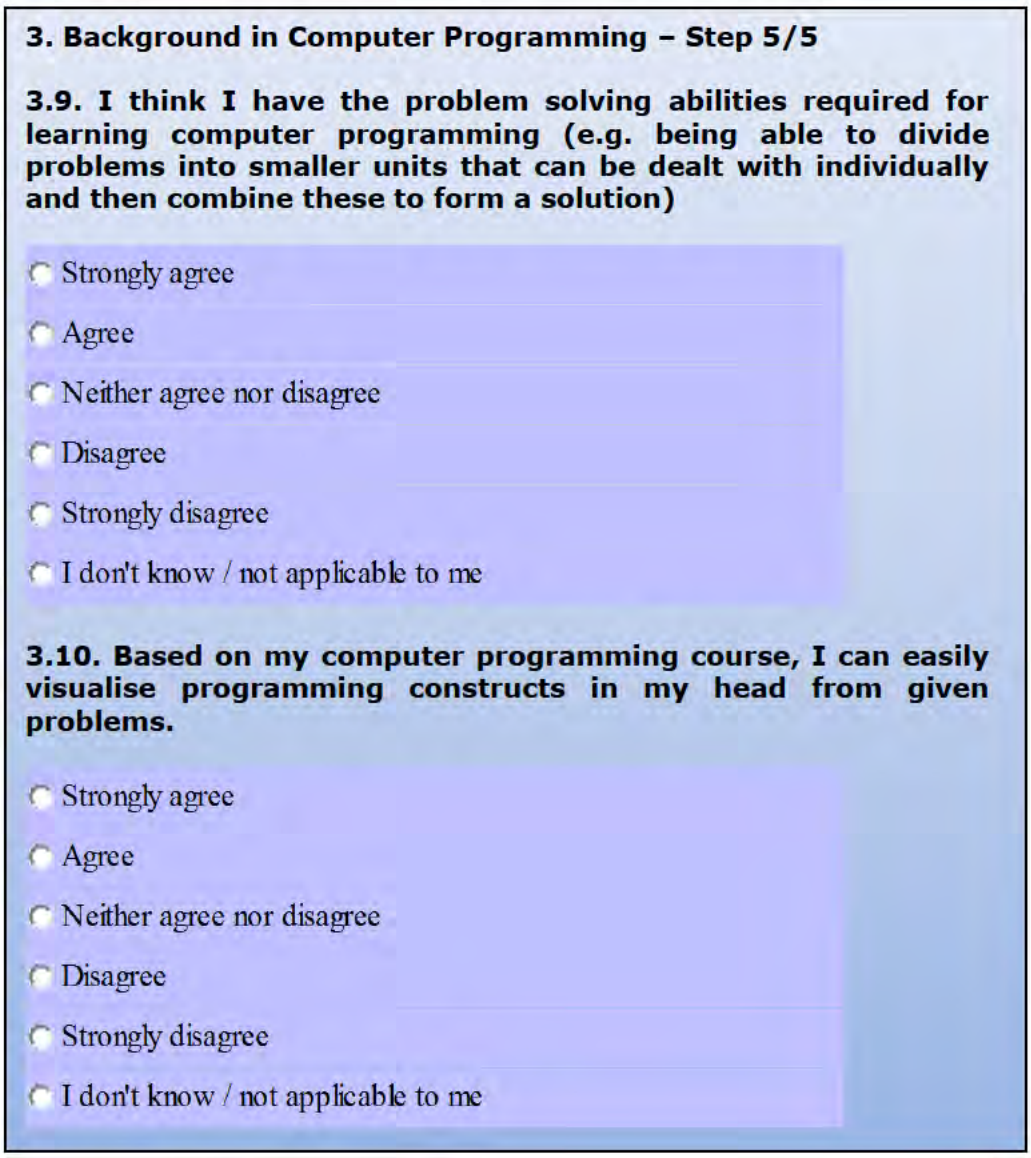

Figure 4.11 – Collecting data on participants' problem solving abilities and their ability to visualise programming constructs in the pre-study questionnaire.

As shown in Figure 4.11, the final set of questions asked in the "Background in computer programming" section were participants' problem solving abilities and their ability to visualise programming constructs from given problems. These questions are directly linked to the seventh (i.e. Is there a difference between students' perception of their problem solving abilities before and after they play *Program Your Robot*?) and the eighth research questions (i.e. Is there a difference between students' perception of their ability to visualise programming constructs from given problems before and after they play *Program Your Robot*?). Additionally, both of these questions were asked in the pre-study questionnaire and also in the

post-study questionnaire of all studies. The answers of the participants would be matched to analyse whether or not the studies make a significant difference in participants' perception of problem solving abilities as well as their ability to visualise programming constructs which are both related to their computational thinking abilities.

4.3.4 Games and learning

The final section of the pre-study questionnaire is designed to collect data on participants' gaming background as well as their perception regarding learning computer programming through playing games. Each of the questions listed in this section is asked in all three studies.

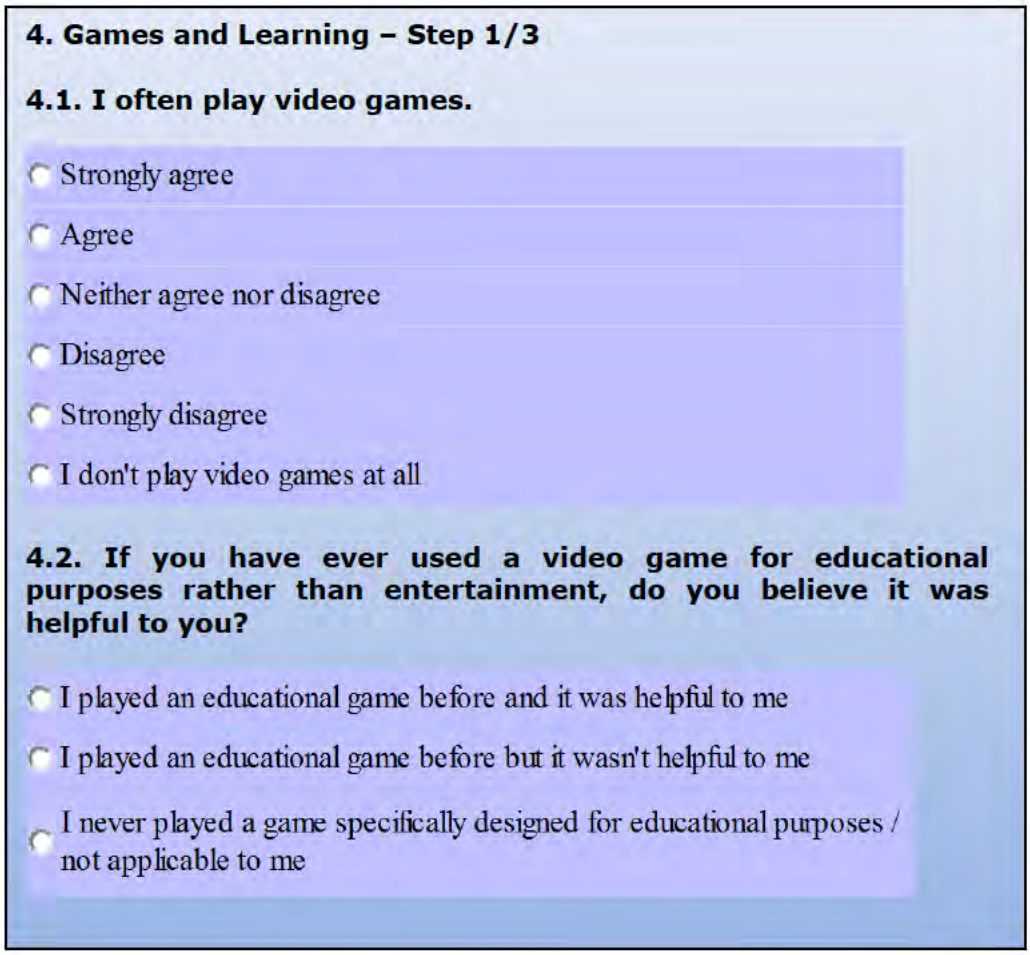

Figure 4.12 – Collecting data on participants' gaming experience in the pre-study questionnaire.

As shown in Figure 4.12, participants were first asked to rate whether or not they often played video games. Having answered this question, participants were requested to answer if

they have previous experiences regarding serious games. The intention behind these questions was to identify those participants who have a strong gaming background as well as those participants who don't have a gaming background. Should a positive significant outcome be obtained from the studies; it is intended to measure the impact of playing games in order to verify the validity of the findings. In other words, it is planned to compare the responses of those participants who have a good gaming background against the responses of those participants who do not have a good gaming background. In this way, it is anticipated to identify whether or not playing games often has an impact on participants' perception of programming knowledge learned from *Program Your Robot*.

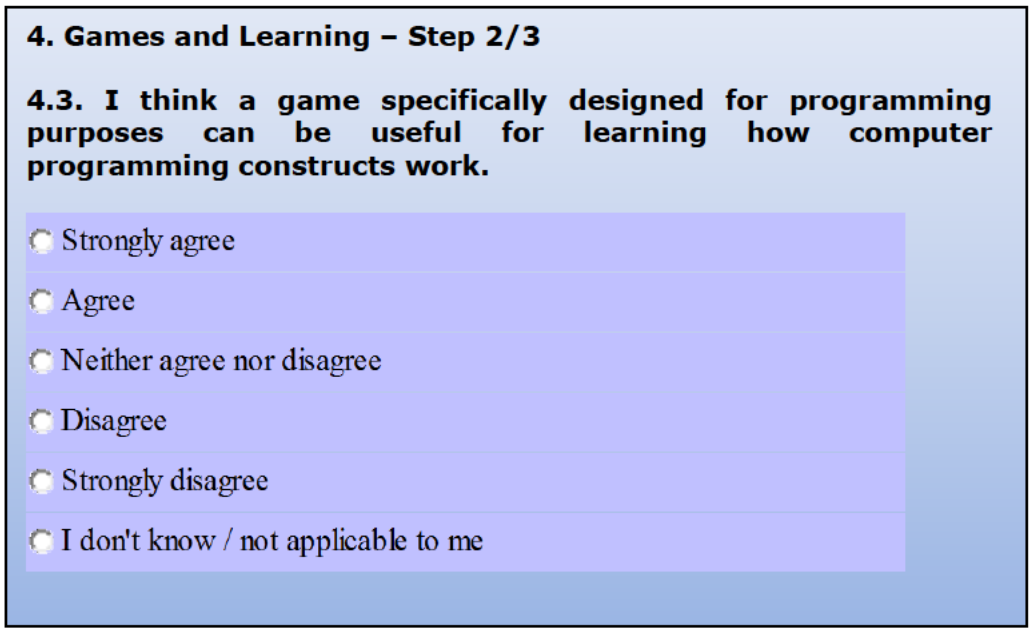

Figure 4.13 – Collection data on participants' attitude to learning programming constructs through playing games in the pre-study questionnaire.

The last closed-ended question asked in the pre-study questionnaire is directly related to the first research question of this study (i.e. Is there a difference between students' attitude to learn computer programming through playing games before and after they play *Program Your Robot*?). The rationale behind this is to observe students' attitude regarding learning computer programming constructs through playing games before they played *Program Your Robot*. The same question is asked in the post-study questionnaire and the results would be matched to decide whether or not a significant change in participants' attitude happens during the studies.

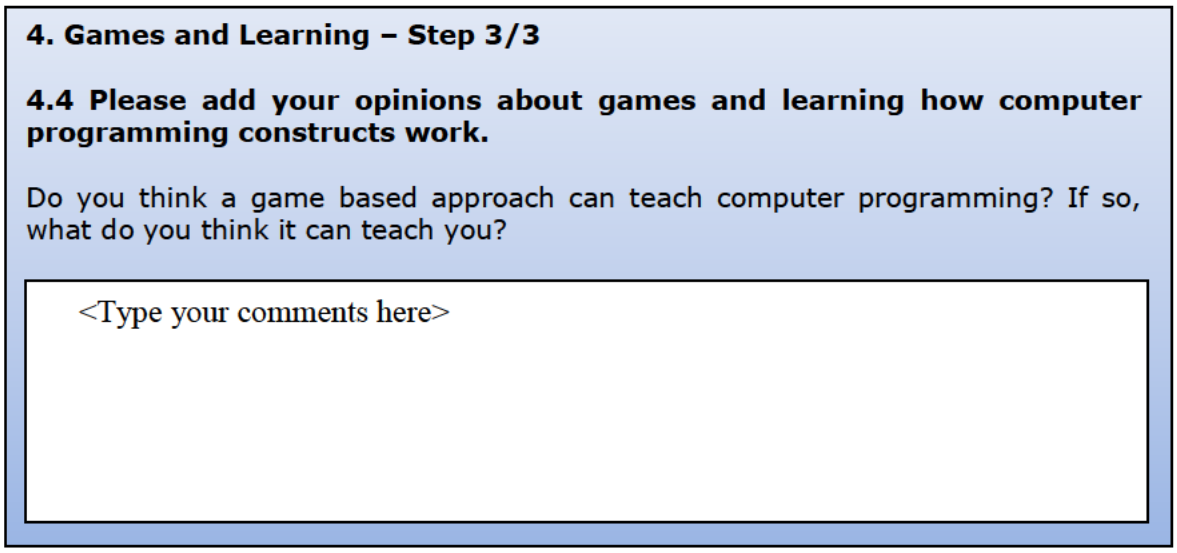

Figure 4.14 – Collecting data on participants' opinions regarding games and learning how computer programming constructs work in the pre-study questionnaire.

The final question of the pre-study questionnaire was a semi-structured open-ended question aimed at exploring participants' viewpoints on a potential game based learning approach for learning how computer programming constructs work before they played *Program Your Robot*. This open-ended question was asked in all studies and as shown in Figure 4.14, participants were provided a series of questions to guide them – thus making the question semi-structured. The intention behind this open-ended question was to obtain the preliminary viewpoints of participants regarding games and learning programming constructs before they played the game. Although a similar question would be asked in the post-study, it is not possible to compare students' responses to this semi-structured question in the same way that the closed-ended questions would be matched. There are two main reasons for this: a) answering the semi-structured question was completely optional and many students might not want to provide their viewpoint and b) it is not possible to analyse the viewpoints of students statistically through using quantitative statistical methods as the responses are merely comments. As a result, the viewpoints of students would be used to report preliminary viewpoints of participants as supportive evidence rather than as a statistical outcome.

This section described the purpose of each question asked in the pre-study questionnaire. Although the virtual structure of the pre-study questionnaire was identical for all studies, it was simply not possible to use the same pre-study questionnaire in all studies due to the differences in the target groups. The post-study questionnaire is described in the next section of the chapter.

4.4 Post-study questionnaire

This section focuses on the post-study part of the research and discusses the rationale behind each question asked in the post-study questionnaire. The section outlines how the data obtained from the post-study would be used and which question refers to which research question identified at the beginning of the study. There are five different sections in the post-questionnaire and these are listed as follows: *Username (or random unique number)*, *game experience*, *computer programming*, *computational thinking skills* and finally *attitude to learning through playing games*.

The *username/random unique number* section collects university usernames or the random number given to participants so that their answers given in the post-study could be matched with their answers given in the pre-study. The *game experience* section explores the gaming experience of participants, whether or not they liked the game and/or found it useful for learning computer programming purposes. The *computer programming* section measures participants' perception of their knowledge in computer programming after playing the game. The *computational thinking* section assesses participants' perception of computational thinking abilities and finally, *attitude to learning through playing games* explores participants' perception regarding learning computer programming constructs through game-play after they played *Program Your Robot*. Each of these sections is discussed below.

4.4.1 Username / random unique number

As shown in Figure 4.15, the username/unique number in the post-study was collected exactly in the same way as in the pre-study. The username/unique number of participants was required to match the responses participants gave to the post-study with the responses they had given to the pre-study. Similar to the structure of the pre-study, participants in the Greenwich study were asked to enter their username whereas participants in the Cyprus and the PGS study were asked to use their random unique number for this part of the questionnaire.

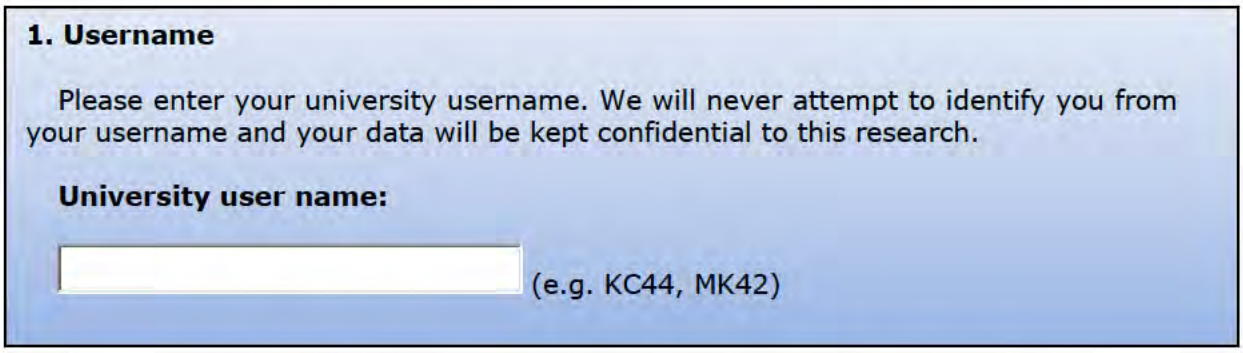

Figure 4.15 – Username question in the pre-study questionnaire.

4.4.2 Game experience

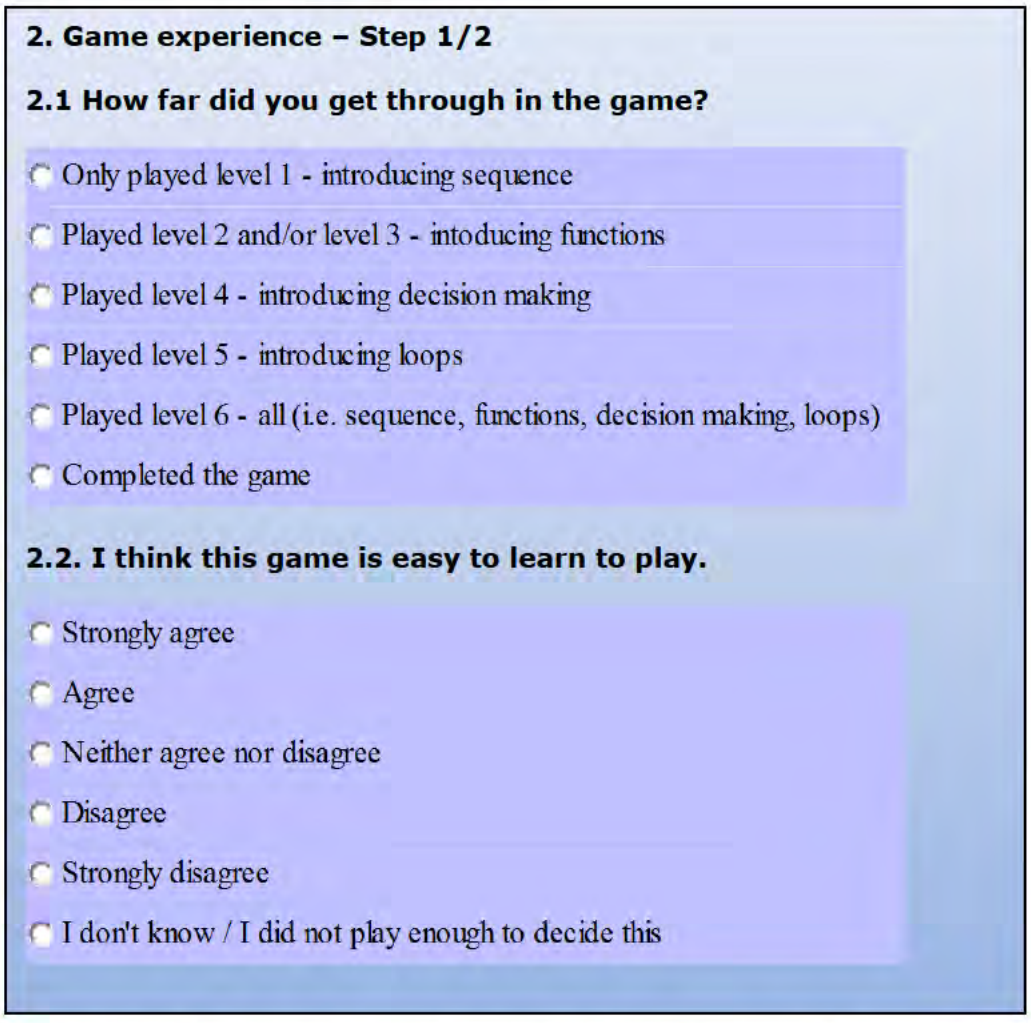

Figure 4.16 – Collecting participants' progress in *Program Your Robot* in the post-study questionnaire.

The game experience section is created for gathering data on the participants' gaming experience of *Program Your Robot*. As shown in Figures 4.16 and 4.17, it was aimed to identify whether or not participants a) achieved high levels in the game b) found the game easy to play c) think the game presents programming constructs effectively and d) think they were introduced to this game at the right time (i.e. when they are learning introductory computer programming).

In question 2.1 participants were asked to answer the highest level they achieved in the game and in question 2.2 they were asked whether or not they think the game is easy to play. Whilst question 2.3 explores whether or not the game stands as a good example for presenting computer programming constructs, question 2.4 examines whether or not the game was introduced at an appropriate time to them.

All five questions in this section were asked in the Cyprus and the Greenwich studies. However, question 2.4 was not included in the PGS study as participants were not registered on a computer programming course.

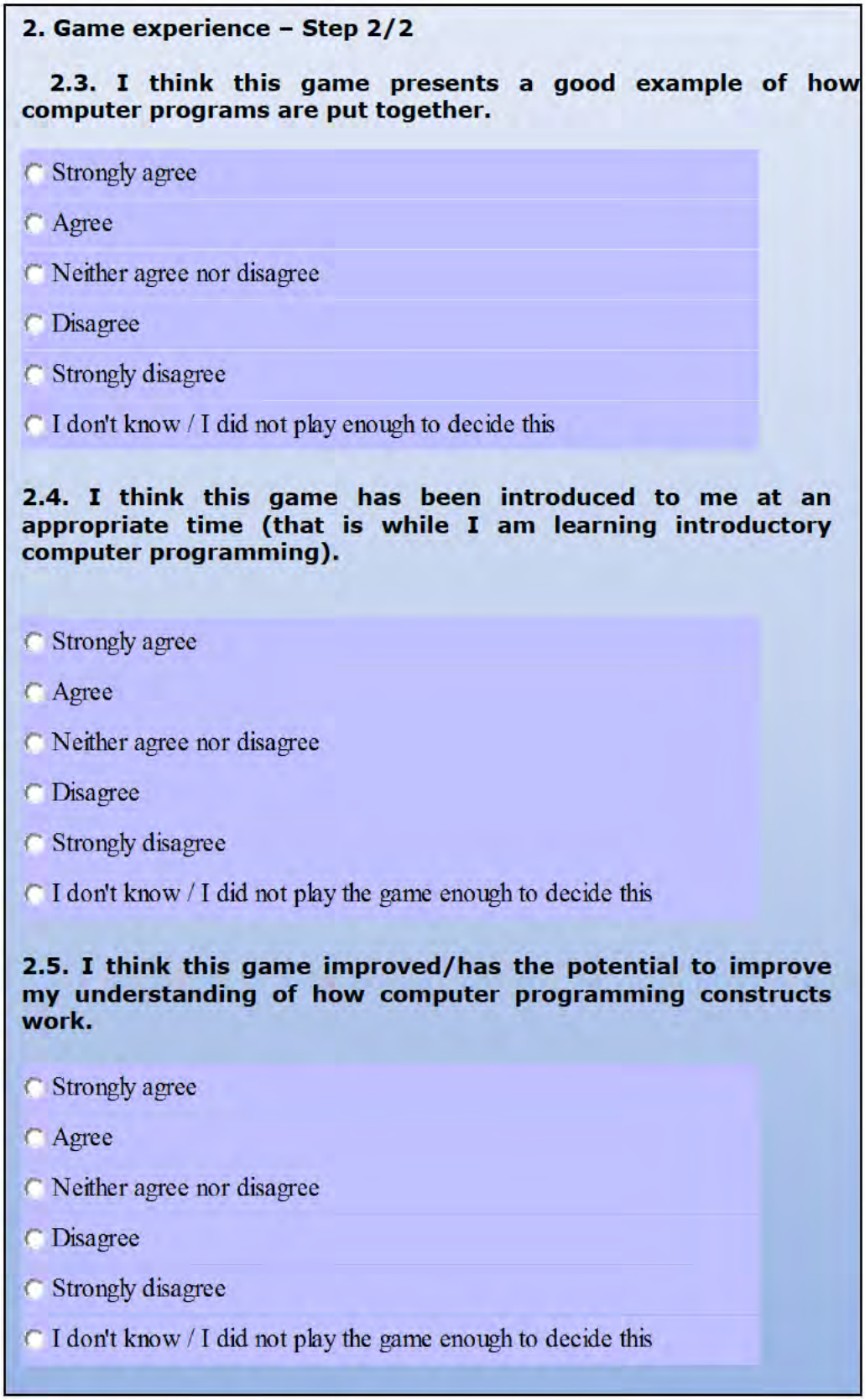

Figure 4.17 – Collecting gaming experience of participants regarding *Program Your Robot* in the post-study questionnaire.

The majority of the questions in the "game experience" section of the pre-study questionnaire were asked for observation purposes and the answers given to these questions would be used as supportive data should an issue be raised when verifying the findings of the studies. Only the answers given to question 2.1 (i.e. students' progress in the game) would be used in statistical analysis of the studies. The responses collected from this question would measure whether or not there is a correlation between achieving higher levels in the game and computational thinking skills. In other words, it is aimed at investigating whether or not those students who achieved higher levels in the game used computational thinking skills (i.e. conditional logic, algorithmic thinking, simulation, debugging and socialising) more often than those students who did not achieve high levels in the game.

4.4.3 Computer programming

This section is the crux of the post-study questionnaire as the data obtained from this section will be used to analyse and answer the research questions listed in Section 4.2. Additionally, all questions asked to participants in this section of the post-study exist in the pre-study questionnaire. Therefore, it is designed to match the answers given to these questions, to the answers obtained from the pre-study questions. The questions asked in this section were slightly modified from their pre-study versions and re-written in order to emphasise that the post-study questionnaire was designed to gather participants' perception and experiences after they played *Program Your Robot*.

The computer programming section of the post-study questionnaire follows a parallel structure to the pre-study questionnaire. In question 3.1, participants are asked to rate their computer programming knowledge and skills after they play *Program Your Robot*. Further to this, question 3.2 examines how difficult participants find learning computer programming after their game-play. Identical to their equivalent questions in the pre-study, question 3.1 is written to identify whether or not participants think the game improved their general knowledge in computer programming and question 3.2 is explicitly written to interpret the ninth research question of the study (i.e. Is there a difference between students' perception of difficulty of learning computer programming before and after they play *Program Your Robot*?). The answers given to question 3.1 would be used when verifying the validity of the experimental findings and accordingly, the responses given to question 3.2 would be used in the statistical analysis of the ninth research question.

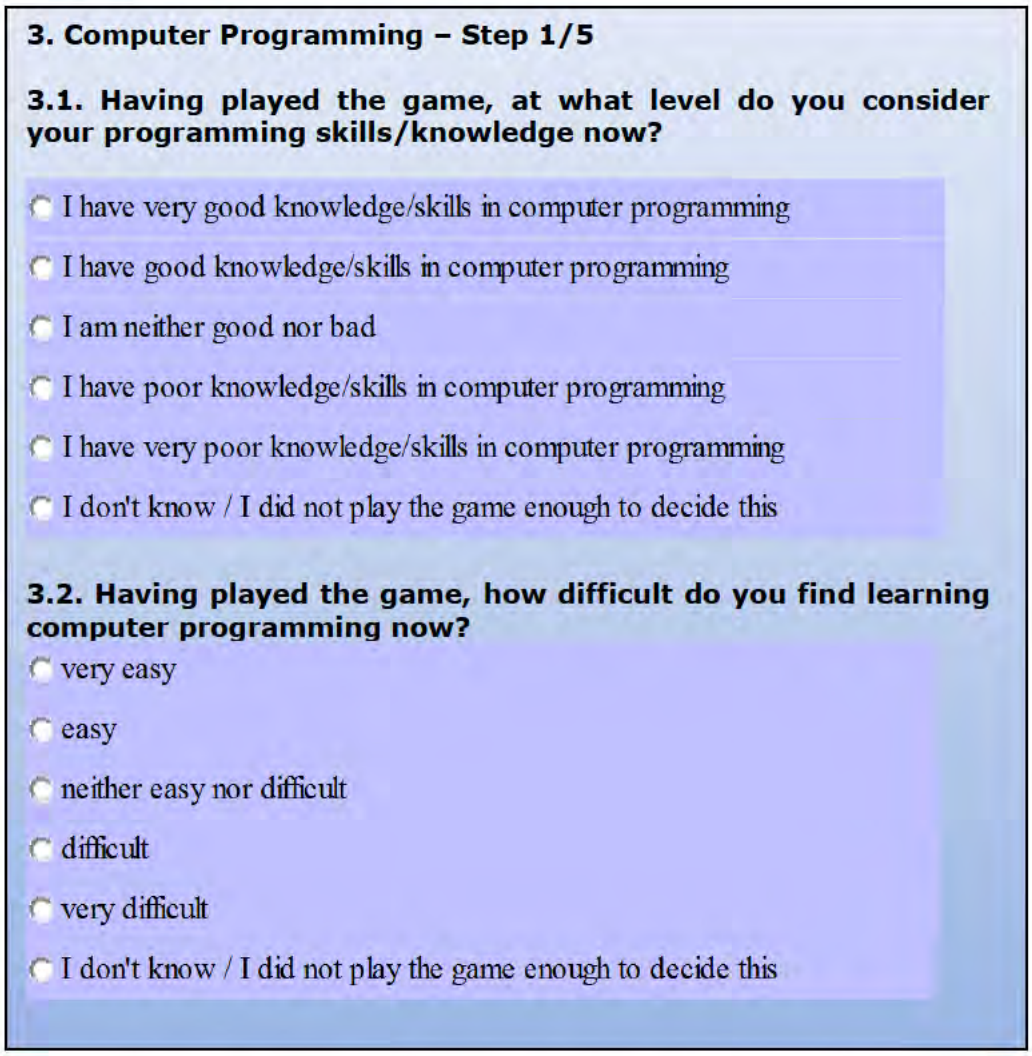

Figure 4.18 – Collection of data on participants' computer programming knowledge and the difficulty of learning computer programming in the post-study questionnaire.

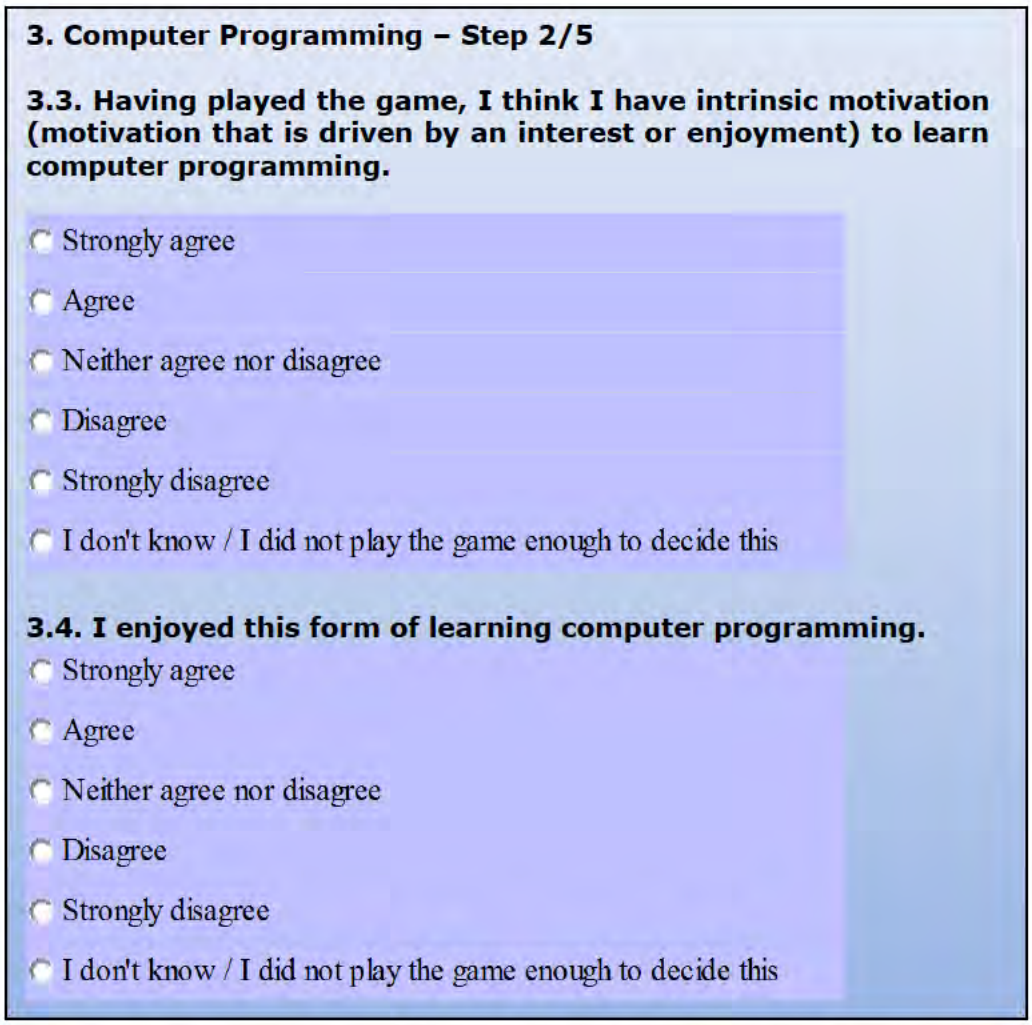

Figure 4.19 – Collection of data on participants' intrinsic motivation and enjoyment in learning computer programming in post-study questionnaire.

Whilst question 3.3 explores participants' intrinsic motivation to learn computer programming after they played *Program Your Robot*, question 3.4 is intended to define whether or not participants enjoyed learning computer programming through playing the game. Similar to their pre-study equivalents, these questions are closely related to each other and directly linked to the second research question of this study (i.e. Is there a difference between students' intrinsic motivation to learn computer programming before and after they play *Program Your Robot*?). The responses given to these questions would be matched with their pre-study counterparts. It is important to highlight that the intention behind these questions was not to compare *Program Your Robot* against a traditional learning and teaching of computer programming but rather to investigate how much students enjoyed learning computer programming constructs before and after their game-play.

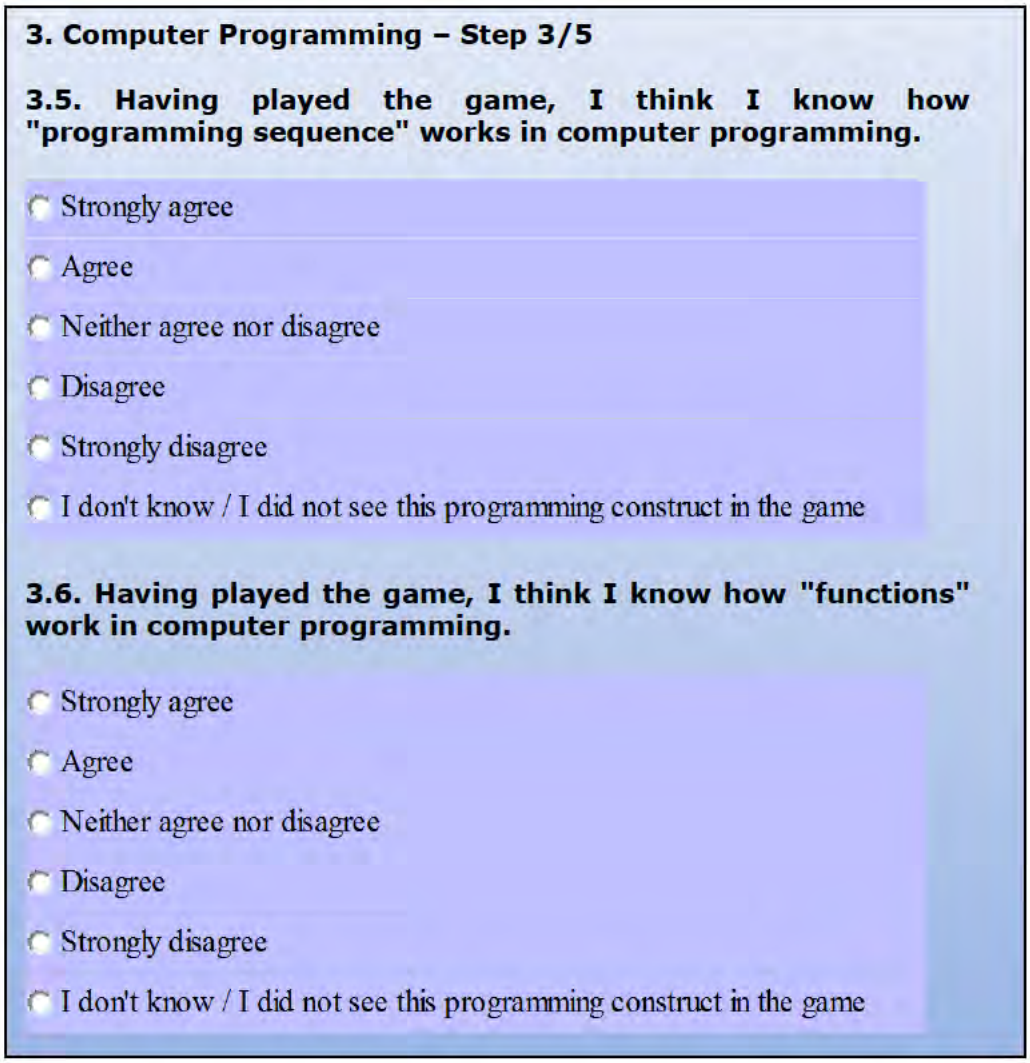

Figure 4.20 – Collection of data on participants' knowledge of programming sequence and functions in the post-study questionnaire.

Figure 4.20 shows how participants were requested to rate their knowledge in programming sequences and in functions after playing *Program Your Robot*. The answers provided to these questions would be matched with the answers participants provided to the pre-study questionnaire equivalents of these questions. The data obtained would be used in the analysis of third (i.e. Is there a difference in students' understanding of "programming sequence" in computer programming before and after they play *Program Your Robot*?) and fourth research questions (i.e. is there a difference in students' understanding of "functions" in computer programming before and after they play *Program Your Robot*?).

Figure 4.21 illustrates how decision making and loop programming constructs were investigated in the post-study questionnaire of the studies. The answers given to these questions would be matched with the corresponding pre-study questions to decide whether or not playing the game made a difference in the participants' perception of their knowledge regarding decision making and loop programming constructs. The data obtained will be used in the analysis of the fifth (i.e. Is there a difference in students' understanding of "decision making" in computer programming before and after they play *Program Your Robot*?) and the sixth research questions (i.e. Is there a difference in students' understanding of "loops" in computer programming before and after they play *Program Your Robot*?).

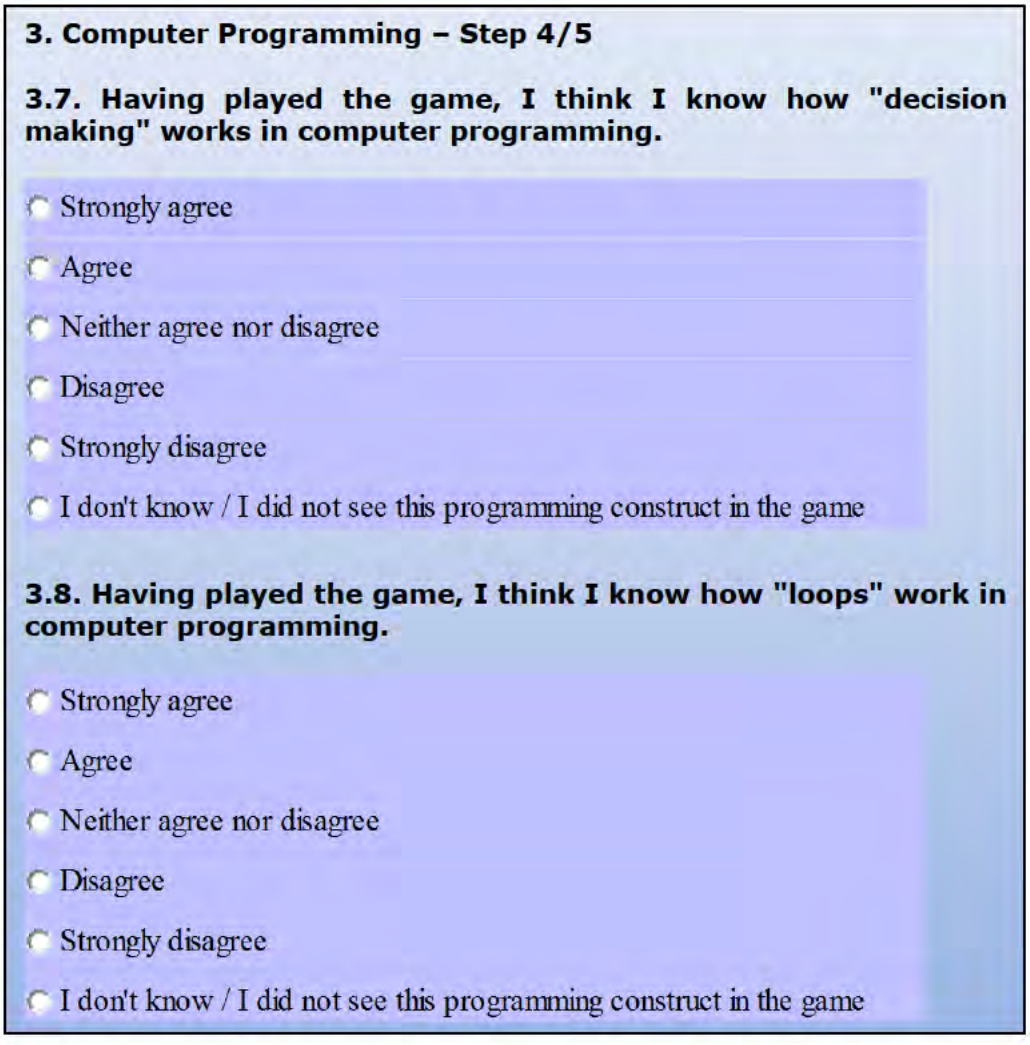

Figure 4.21 – Collection of data on participants' knowledge in decision making and loops in the post-study questionnaire.

The final part of the computer programming section of the post-study questionnaire investigates participants' problem solving abilities and their ability to visualise programming constructs after their game-play. The answers gathered from these questions would first be matched with their pre-study equivalents and then be used in the analysis of seventh (i.e. Is there a difference between students' perception of their problem solving abilities before and after they play *Program Your Robot*?) and the eight research questions (i.e. Is there a difference between students' perception of their ability to visualise programming constructs from given problems before and after they play *Program Your Robot*?).

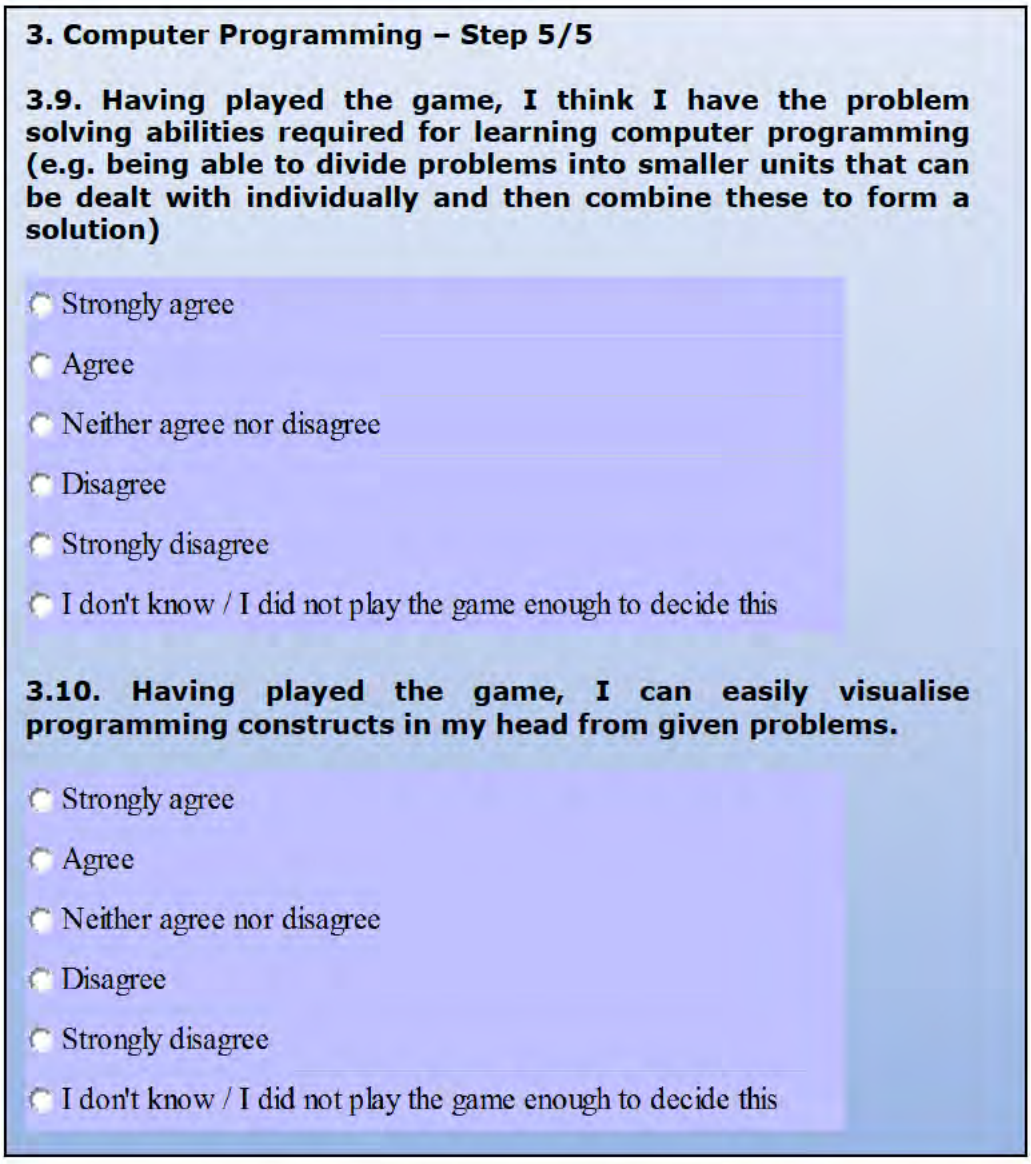

Figure 4.22 – Collection of data on participants' problem solving abilities and their ability to visualise programming constructs in the post-study questionnaire.

4.4.4 Computational thinking skills

The fourth section of the post-study predominantly investigates the computational thinking skills of participants after they played the game. The answers given to this section will be used to assess whether or not there are statistical correlations between participants' perception of computational thinking skills and how far they progressed in the game. Additionally, it was designed to identify whether or not strong correlations exist between computational thinking skills and the programming constructs presented in the game.

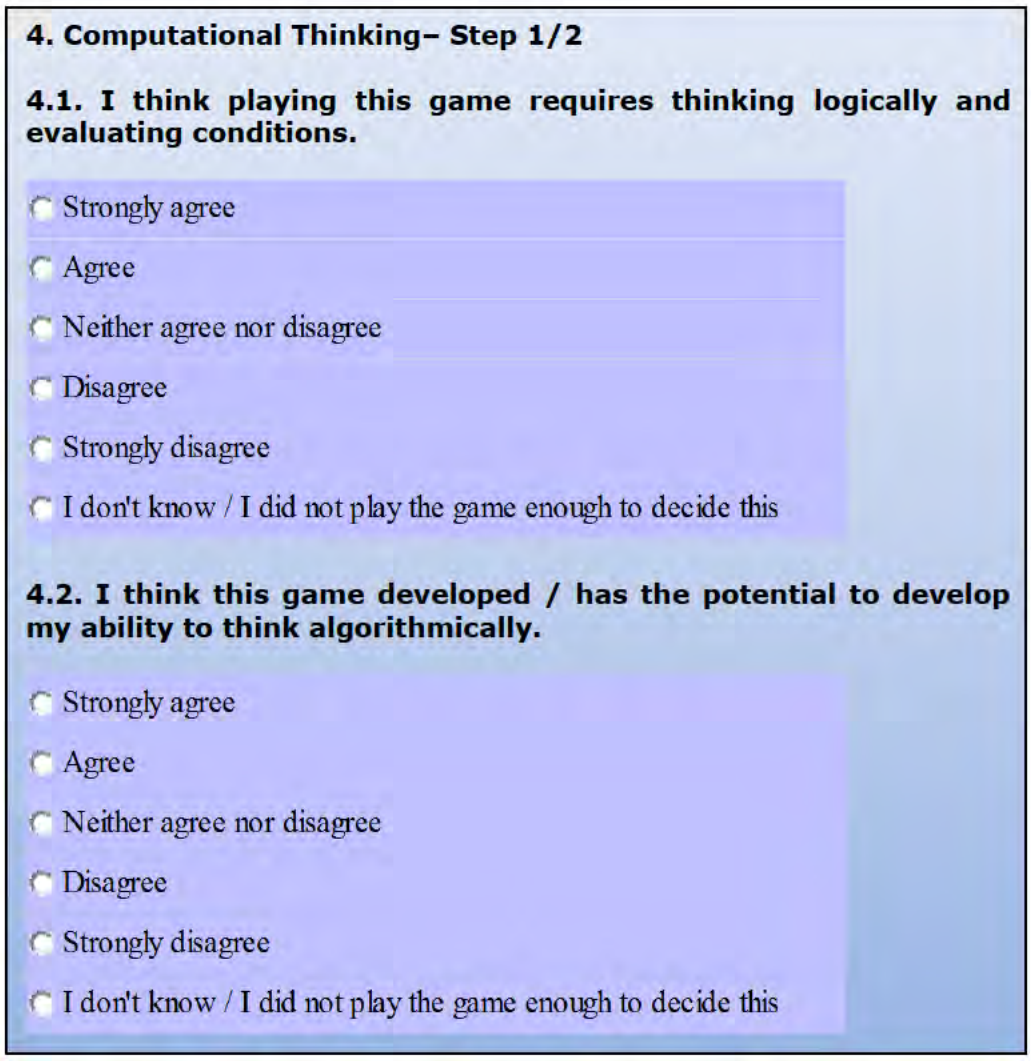

Figure 4.23 – Collection of data on participants' conditional logic and algorithmic thinking skills in the post-study questionnaire.

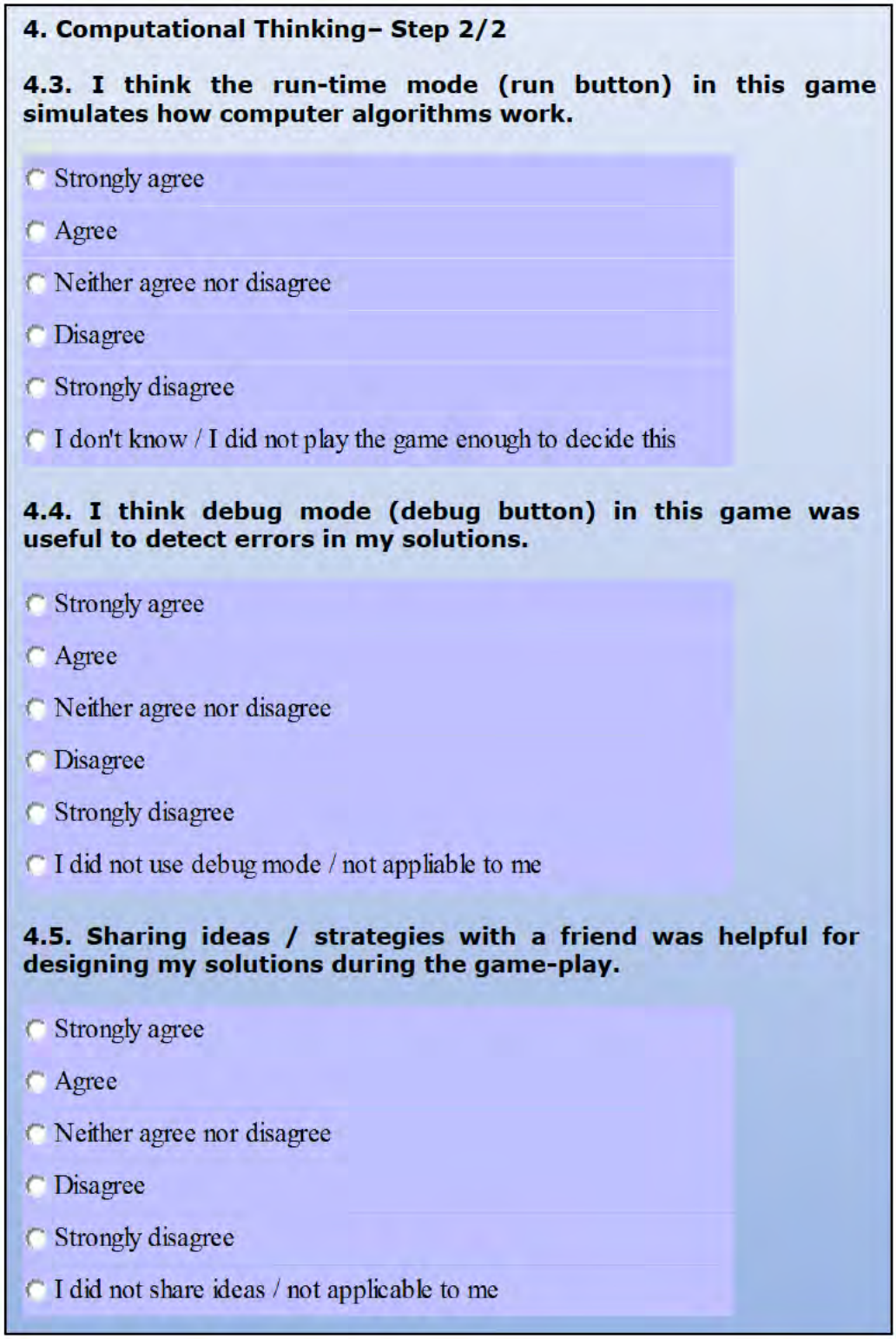

Figure 4.24 – Collection of data on participants' conditional logic and algorithmic thinking skills in the post-study questionnaire.

As shown in Figures 4.23 and 4.24, each computational thinking category integrated into *Program Your Robot* was explored in a different question in the post-study questionnaire. The conditional logic, algorithmic thinking aspects of computational thinking is investigated in the

first two questions whereas simulation, debugging and socialisation are explored in the rest of the questions.

At this point, it is important to highlight that only the *cooperation* aspect of socialising is investigated in this research as *competition* in the game was optional and it was not intended to compare players' progress with each other. Additionally, *Program Your Robot* was not explicitly designed to support cooperation during the game-play. Therefore, this research only investigates whether or not participants share ideas during their game-play and whether they would find this useful for constructing their solutions.

The answers given to computational thinking questions would be matched with the answers given to programming constructs in the game (i.e. programming sequence, functions, decision making and loops) as well as with how far students progressed in the game in order to accurately determine whether or not there are strong correlations between computational thinking skills and a) achieving high levels in the game and b) programming constructs specifically integrated into the game environment as part of the game-play.

4.4.5 Attitude to learning computer programming through playing games

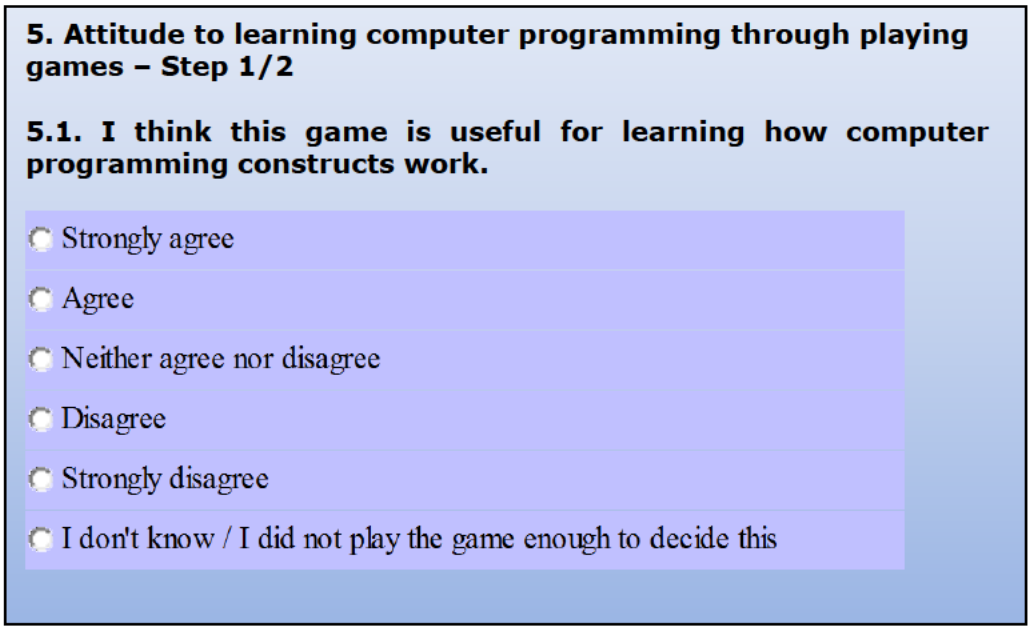

Figure 4.25 – Collection of data on participants' attitude to learning programming constructs through playing games in the post-study questionnaire.

The last section of the post-study questionnarie contains one closed ended and one open-ended question. Figure 4.25 illustrates the closed ended question asked to participants which is directly related to the question 4.3 previously asked in the pre-study (i.e. I think a game specifically designed for programming purposes can be used for learning how computer programming constructs work). The answers given to these questions would be matched with each other and used for analysing the first research question of this study (i.e. Is there a difference between students' attitude to learn computer programming through playing games before and after they play *Program Your Robot*?).

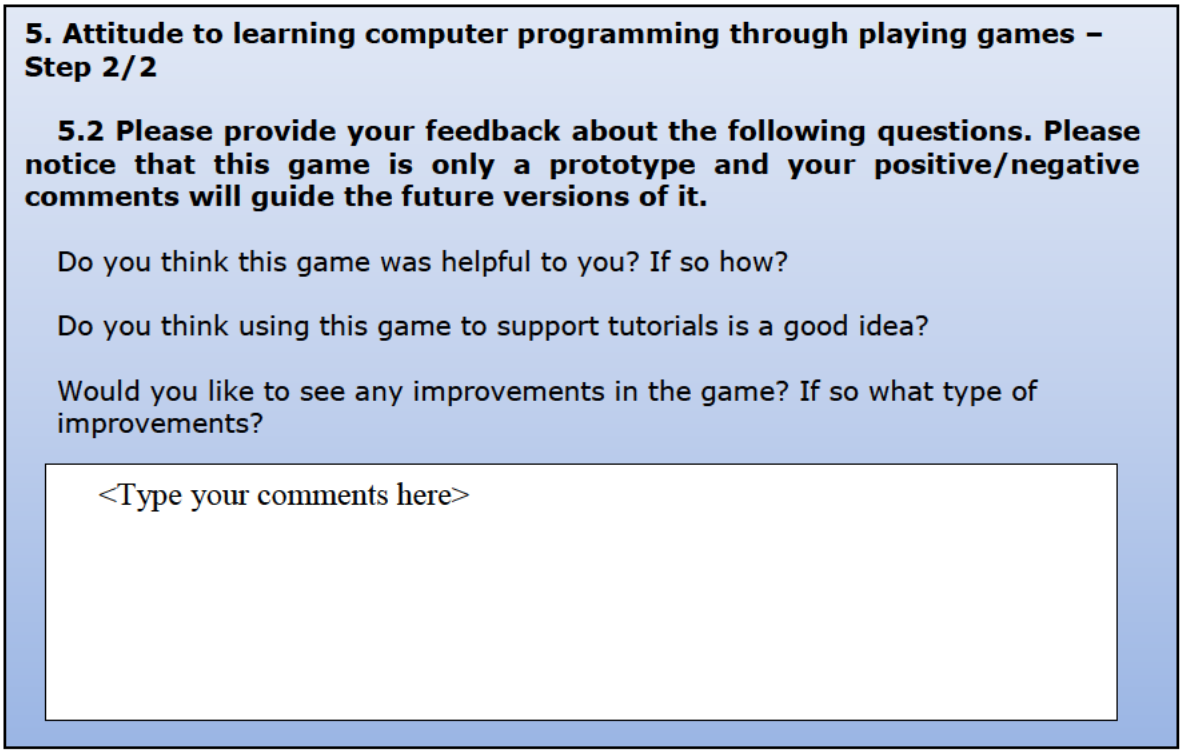

Figure 4.26 – Collection of data on participants' opinions regarding Program Your Robot and learning programming in the post-study questionnaire.

Figure 4.26 shows the very last question of the post-study which is an open-ended question asked to participants to explore their viewpoints and personal experience about *Program Your Robot*. The question was asked to participants at the end of the post-study questionnaire to investigate whether or not a) *Program Your Robot* was useful to them b) using *Program your Robot* is a good idea to support their tutorial hours and c) they would like to see any improvements in the game. The rationale behind this question is to obtain qualitative data that could be supportive for the findings of the studies. As this research falls between a phenomenographic approach and a quantitative case study, it was designed to investigate

participants' perception regarding *Program Your Robot* and learning programming in order to provide qualitative evidence in addition to statistical evidence that would come from the analysis of the closed-ended question.

4.5 Summary

This chapter identified the research approach of this experimental research as a mixed methodology that is the combination of phenomenography and a quantitative case study. It was explained that phenomenography is an inductive method that is mixed with a case study to observe participants' second order perspective in using a serious game (i.e. *Program Your Robot*) as this research is a perceptual study that observes students' understanding of computational thinking skills and introductory programming constructs. The chapter also explained why a phenomengraphic research is blended into a quantitative case study and what benefits are obtained from applying this structure. The main research question of this research is revisited and divided into several sub research questions in order to explore different aspects of the research question individually. Having overlaid the research approach and clarified the research question, the chapter further explained the pre-study and the post-study questionnaires used in the studies. The rationale behind each question asked to participants and how these questions are related to the research questions are also explained in this chapter. Additionally, the differences between the studies and the questions therein studies are also provided. Finally, a brief plan is outlined regarding how the data, collected from pre and post study questionnaires, would be analysed.

The next chapter discusses the experimental design of three studies as well as the structure of a pilot study designed to obtain initial feedback from students before the studies were conducted. The next chapter also describes the rationale for participant selection, the setting for each study as well as the limitations and potential threats of these studies.

CHAPTER 5
EXPERIMENTAL DESIGN

This chapter explains one of the key aspects of this research which is the experimental structure of the studies (i.e. the Cyprus, the Greenwich and the PGS studies). The chapter first explains the experimental design of a pilot study that was conducted to obtain initial feedback regarding the game from the students who were studying a Computer Science degree at the University of Greenwich. After clarifying the purpose and the participant selection of the pilot study, the chapter explains the structure of the three studies as well as their similarities and differences. The ethical issues in the studies are also explained. Further to these, the chapter outlines the hypotheses of this research and classifies the experimental variables (i.e. independent and dependent variables) in the studies. Finally, the limitations and the threats in the structured studies are also discussed in this chapter.

Section 5.1 discusses the semi-structured pilot study as well as the experimental design of all three structured studies. The section discusses the main motive behind using a quasi-experimental design in this research and also outlines the reasons why a gold standard intervention evaluation (such as a cluster randomised controlled trial) was not as an experimental design. Further to this, the section explains participant selection, setting for each study, the differences and the similarities between the conducted studies. Section 5.2 investigates ethical issues regarding the research and the studies. Whilst section 5.3 outlines experimental variables and the hypotheses generated from the research questions, section 5.4 describes the limitations and potential threats of the three structured studies.

5.1 Experimental design

This section first describes a pilot study and then outlines the structure of the three experimental studies. The experimental design of the conducted rigorous studies fits into *the one group pre and post study design without a control group* among quantitative research methodologies. This experimental design was chosen because it was not possible to create a control group for this research. Additionally, it was not possible to use a more reliable experimental research structure such as Randomised Controlled Trials (RCTs) due to the ethical constraints of the research. The experimental design, the similarities and the differences between these experimental studies are all discussed in this section.

5.1.1 Experimental design of the pilot study

A pilot study was designed to obtain perception and feedback from students who were all studying a degree within the Computer Science discipline at University of Greenwich in order to identify the positive and negative issues of the game to determine if it was suitable for use in the structured studies. The pilot study was not designed as a structured study; participants were offered a series of open-ended questions that would guide them to express their perception and experiences regarding the game. Open-ended questions were used to encourage participants to express their opinion towards the questions offered to them and it was not intended to limit them within a rigorous structured approach (i.e. closed-ended questions). The open questions involved: a) specifying personal experiences as they occur – good and bad; b) the number of levels participants achieved in the game and c) reporting crashes/errors along with the circumstances if participants experienced any errors. It is also crucial to underline that the structure of the pilot study was coherent with the semi-structured interview schedule discussed earlier in phenomengoraphic research approach as obtaining participants' perception regarding an interference (i.e. the game) was the main purpose of the pilot study (Orgill, 2002; Bowden, 2005; Ornek, 2008; Mann, 2010).

The pilot study was designed as a voluntary exercise and the feedback of participants was confidential as it was not aimed to expose the identity of participants. Additionally, the confidentiality of the pilot study allowed minimising one of the criticisms of the semi-structured interview data collection method of phenomengorahpic research namely, the meta-awareness of the participants who are being interviewed. As argued previously, when a participants identity is exposed they might provide "expected answers" rather than what they really think and/or feel about a phenomenon especially when the interviewees are students and the interviewer is an academic member of staff in the same institution (Richardson, 1999). To minimise this threat, participants submitted their viewpoints without providing any personal details. Students were well-versed in this procedure before they participated in the pilot study and they were assured by their teaching staff that their anonymity was guaranteed. Additionally, they were informed about how their answers would be used, and thus they were under no obligation to answer any of the open-ended questions suggested at the beginning of the study.

The participants in the pilot study were from a wide range of programmes within the Computer Science discipline at the University of Greenwich. The majority of participants were either second year or a third year undergraduate students. More importantly, as participants

were studying different degree programmes within Computer Science, their programming knowledge and skills were considerably different. This proved beneficial in terms of evaluation as the feedback in the pilot study came from participants with diverse knowledge, background and experience. At the end of the study, twenty-five students provided feedback and some of these provided reports in remarkable detail.

The results of the pilot study were not analysed through a statistical measure as this was an initial freeform evaluation of the game with a group of volunteer students who had already studied a computer programming course. The aim of the pilot study was not to measure whether or not participants are learning from the game but rather it was aimed to a) find out if *Program Your Robot* was ready to run the structured studies and b) investigate if the game indeed provides an abstraction to programming constructs to facilitate learning introductory programming as well as whether or not it genuinely encompass skills in computational thinking.

Despite this fact, the data gathered from the pilot study shows that the majority of participants (18 out of 25) found the game useful in helping them to understand introductory programming constructs and develop their problem solving skills. Some participants reported that the game actually developed their knowledge regarding the programming constructs introduced in the game. The evaluation of participants regarding this pilot study together with how their feedback was incorporated to enhance the game prototype is discussed in Chapter 6 Section 6.1.

5.1.2 Experimental design of the studies

As discussed in Chapter 4 Section 4.3, three different rigorous studies were designed to provide a systematic and structured evaluation of *Program Your Robot* and the underlying game model (i.e. *interaction – feedback loop* game model). The results of these studies will provide analytic data to determine whether or not the game model is successful in encouraging the development of computational thinking (CT) skills and as a result, whether or not the game helps students to learn and use key concepts in introductory programming. Both of these aspects will be analysed separately and in combination, to ensure the accuracy of this approach and any benefits that can be derived from it. To achieve this, all studies are designed to investigate the experiences of students before and after playing *Program Your Robot*, and whether or not they perceive:

a) their knowledge of key introductory programming constructs (i.e. sequence, methods, decision making and loops) introduced in the game increased;
b) their skills in computational thinking (i.e. conditional logic, algorithmic thinking, simulating solutions, debugging and cooperation) were enhanced;
c) an increase in intrinsic motivation to learn programming;
d) an increase in intrinsic motivation to learn programming through playing games;
e) an increase in the ability to visualise programming constructs from given problems;
f) an increase in their problem solving abilities.

In order to select an experimental design for this research, a deep investigation of the game based learning (GBL) literature was performed. It was found that recent work in GBL especially those concerned with healthcare argues that the gold standard in experimental research is Randomised Controlled Trials (RCTs) and Clustered Randomised Controlled Trials (cRCTs) (Kato *et al.*, 2008; Knight *et al.*, 2010; Arnab *et al.*, 2013). Both in RCTs and cRCTs, the effectiveness of an intervention (in this case a serious game) was measured by dividing the participants into two equally but randomly disturbed groups which are often referred to as the *control* and the *experimental group*. Through conducting a double blind study, this type of scientific approach ensures that any improvement by participants is due to the intervention applied and not biased by confounding variables (i.e. outside factors). (c)RCTs are often used in medical intervention with patient population as clinical trials. However, recent studies in GBL stated that this type of approach also provides an efficient evaluation standard to measure the effect of serious games on participants (Arnab *et al.*, 2013).

The difference between cRCTs and RCTs is that RCTs are individually randomised controlled trials whereas cRCTs study interventions directed towards a deliberately selected population. Additionally, cRCTs have control over the interventions directed towards individuals and also has the ability to control the contaminations across the whole population (such as control over individual's changing behaviours influencing other individuals).

Despite the awareness of cRCTs and RCTs, this research did not use randomised trials as experimental design because a) it was not possible to create a control group for the selected target group and the reasons for these are discussed below; b) when the ethical approval was obtained for the research, the university ethics committee (UREC) insisted on ensuring that students receive the same experience throughout the studies. Despite the fact that this research aims to measure the effect of an educational vehicle (i.e. *Program Your Robot*), if some students were advantaged from it, this would be considered unfair and thus, it was not possible

to treat students differently. Therefore, the randomised control trial experimental design was not applied in this research because there were ethical restrictions regarding dividing students into two random groups and applying different educational interventions.

Having identified that (c)RCTs were not suitable, the experimental design of this research decided to be *the one group pre – post test design without a control group.* This approach is sometimes referred to as *Quasi – Experimental Study* among the quantitative research methodologies due to lack of a control group (NCTI, 2012). This experimental design is based on looking at one group of individuals who receive an intervention and the effects of this by checking the difference in pre-test and post-test results.

At the beginning of the studies, participants were invited to complete the pre-study questionnaire previously presented in Chapter 4 Section 4.3. Having filled in the pre-study questionnaire, participants were then invited to play *Program Your Robot* for about 30-40 minutes depending on their motivation. After they played the game, participants were asked to complete the post-study questionnaire which was discussed earlier in Chapter 4 Section 4.4. The total time allocated for the studies was an hour. However, it was pleasing to note that, all three studies took longer than the allocated time as many participants insisted on playing longer or saved the link of the game to play later. Both pre and post study questionnaires were completed online.

As discussed earlier in Chapter 4 (Sections 4.3 and 4.4), all questions asked to participants were closed-ended questions with the exception of an open-ended question at the end of the each questionnaire (i.e. pre and post). A Likert scale was used in all comparable closed-ended questions, which is a rating measure used extensively when assessing attitude and perception. The scale ranged from "*strongly disagree*" (1) to "*strongly agree*" (5) with a "*not applicable to me / I do not know* (0)" choice. The Likert scale is chosen because it offered a number of qualitatively different viewpoints and this is an integral part of phenomenographic approach. Additionally, an open-ended question is added at the end of questionnaires to allow students to express their thoughts before and after their game experience.

In all three studies, teaching staff were involved with the participant cohort and a random selection of students was implemented. This means that the author did not know about participants' background in computer programming or in games until they completed the studies. In other words, participants were randomly selected without looking whether or not they have a good gaming or computer programming background.

Although this study fits into *the one group pre – post test design*, there are significant differences that serve the purpose of the research.

Firstly, this research is not aimed at measuring participants' knowledge or skills in terms of a test as the term *test* refers to a systematic approach for assessing knowledge or skills in the academy. The questions designed in the pre and post studies do not assess participants' knowledge or skills and none of the questions in the pre and post questionnaires has a correct answer. The main aim of this experimental research is to measure the perspective of participants regarding an experience and define whether or not a change happened in their attitude, learning and motivation. This is also the crux of this research where the phenomenographic approach was blended into the experimental design as participants themselves decide whether or not the game improves their understanding of different computer programming constructs. Hence, it is more appropriate to define this experimental research as *one group pre – post study design* rather than *one group pre – post test design* because this is not a knowledge measurement study but it is a perspective study.

Secondly, in a traditional *one pre – post test study*, the questions asked to the participants are always identical. However, in this experimental research, the questions asked to participants in the post-study were not always the same as the questions asked to participants in the pre-study (see Chapter 4 Sections 4.3 and 4.4). Whilst most questions (e.g. participants' motivation for learning introductory programming) are asked both in the pre and the post study questionnaires, some questions (e.g. How far participants achieved in the game) simply do not exist in both studies. The reason for this is because it would be inappropriate to ask some questions in the pre-study before participants actually played the game. Additionally, not all questions were designed to be assessed through a statistical measure because a) not all questions exist in both studies (i.e. pre and post); b) some questions in studies are simply not comparable due to the reason that these were aimed at collecting personal data or qualification of participants (e.g. What is your highest mathematical qualification?).

It is often advised that one group pre – post study design should be viewed with caution since the differences in the pre and post-study might not be related to the intervention as there are always potential factors that might affect the outcome of a study. These potential factors are often referred to as *confounding variables or threats* in experimental research (SRM, 2006). To measure the effect of confounding variables, a control group is often suggested to a pre – post study design (SRM, 2006; NCTI, 2012). Despite the fact that a control group does not remove confounding variables, a truly comparable group assures that any confounding variable that applies to experimental group would also be reflected in the control group (SRM, 2006). Therefore a confounding variable affects both groups at the same level and any beneficial result drawn from the study can strongly be linked directly to the intervention by

comparing the two groups.

Despite these advantages, the experimental studies in this research do not have a control group for the reasons described below:

Firstly, in the Cyprus and in the Greenwich studies, participants had only just registered for their computer programming course and therefore, they either had little or no knowledge regarding computer programming constructs introduced in the game (i.e. sequence, functions, decision making and loops). Participants in the PGS study were studying for level 2 Information Communication Technology (ICT) and were not registered to a computer programming course. Hence, it was simply not possible to divide students into two random groups and conduct the studies in an hour as a considerable number of participants had no prior knowledge regarding introductory programming constructs presented in the game.

Secondly, there is no universally agreed way of teaching computational thinking skills to students as this is a relatively new abstract concept and how to teach computational thinking is an active research area (Repenning, Webb & Ioannidou, 2010; Berland & Lee, 2011; Lee, 2011). Therefore, it is not possible to teach computational thinking skills (i.e. conditional logic, algorithmic thinking, simulation, debugging and socialising) to students in a control group within the time allocated for the studies.

Finally, *Program Your Robot* is not intended to replace or compete against any conventional approach for learning computer programming constructs. A control group was not created because there was no alternative model sought to compare the game against. As a result, only one group was formed in each of the experimental studies.

Despite the fact that all three studies are conducted in the same way; there were substantial differences in participants due to population selection, age range, gender, ethnicity and educational background.

	The Cyprus study	**The Greenwich study**	**The PGS study**
Total invited participants	75	189	82
Total responses collected	68	145	52
Participants' age range	Between 18 – 40 or above	Between 18 – 40 or above	All participants were 15
Participants' gender	Both male and female	Both male and female	Only female
Ethnic classification used	No	UK Government Standard	UK Government Standard
Highest mathematical qualification asked?	No	Yes	No
Institution who conducted the study	University / Cyprus	University / UK	Public School / UK
Participants were studying	A Computer Science degree	A Computer Science degree	Level 2 ICT
Consent form used?	Yes	Yes	Yes
Parental Contest form Used?	No	No	Yes
Pre – Post Study is matched through using	Unique numbers	University Username	Unique numbers

Table 5.1 – Differences and similarities between the conducted studies.

The differences and similarities between the participants in all conducted studies are summarised in Table 5.1. As explained in Chapter 4 Section 4.3, two of these studies were taken at university level while one of them was conducted in a public girl school. The Cyprus and the Greenwich studies were very similar in terms of how they were conducted and what type of questions were asked to participants. Participants in both studies were studying a

degree programme related to Computer Science and also were registered to a computer programming course. All participants were randomly selected and invited to participate regardless of their background in games and computer programming. The only notable difference between these two studies (i.e. the Cyprus and the Greenwich) was the ethnic classification and the mathematical qualification of students due to the reason that the studies were conducted in different countries (each with their own exam qualifications and ethnic background).

More importantly, the main target audience of this research was first year introductory programming students as *Program Your Robot* was designed according to the computer programming curriculum at University of Greenwich. However, it is intended to extend this research to a public school in order to provide data as to whether or not the game could be beneficial for school pupils.

Studies in computer programming highlighted that gender does not affect computer programming performance as when boys and girls have similar experience; they are equally interested and effective in learning computer programming (Bruckman, Jenson & DeBonte, 2002; Joiner *et al.*, 2010). These studies also report that learning computer programming is correlated with the amount of time students spent programming rather than gender factors. In contrast to this, other research states that girls are less interested in Computer Science than boys (Shashaani, 1997; Zimmermann & Sprung, 2008; Mason, Cooper & Comber, 2011). Kelleher, Pausch & Kiesler (2007) argued that visual programming tools (such as Scratch, Alice) and games are ideal ways to engage girls into the field of Computer Science. However, to date very little work has provided statistical evidence regarding what girls can learn from a game or game-like environment that is designed to teach computer programming constructs (Denner, Werner & Ortiz, 2012).

Previous research in computer programming and games clearly stated that a) boys are more motivated to study Computer Science than girls (Kelleher, Pausch & Kiesler, 2007; Zimmermann & Sprung, 2008) and b) boys spent more time for playing computer games than girls (Chou & Tsaib, 2007). Considering this fact, a public girls school (PGS) was selected as a target school in this research in order to investigate whether or not *Program Your Robot* would increase school girls' intrinsic motivation and interest in learning computer programming.

The PGS study was conducted in the same way as the other two studies with one distinct difference that is the whole study was conducted by the ICT teacher of participants rather than the author. Additionally, all participants were younger than 18 and their ICT teacher had to get

the permission of participants' parents/legal guardians through parental/legal guardian consent forms before the study was conducted. Further to this, each participant received a consent form individually before the actual study conducted as the participation was voluntary. All forms (i.e. parental/guardian forms and participant consent forms) were distributed and collected by the ICT teacher.

Due to ethical issues, the author was not allowed to be present during PGS study. Because of this reason, there was no opportunity to deal with potential problems or observe how the pupils' game progressed during the study. This situation raised many problems and these are discussed in Chapter 6 Section 6.4.

5.2 Ethical issues

This section discusses the ethical issues that were considered when the experimental studies were designed. Although an ethical guideline is not used for the research, each study implements appropriate ethical standards by getting approval from the University of Greenwich University Research Ethics Committee (UREC) before the studies took place.

The ethical issues involved contacting participants from two different universities and a public girls' school to participate in *Program Your Robot* that involves a pre and a post questionnaire. As indicated above, all participants in the studies were notified that they were under no obligation to answer any of the questions asked to them. Each participant received a consent form at the beginning of the studies so that they were able to decide whether or not they wanted to participate. Additionally, the identity of all participants was kept confidential. The participants in the PGS study also received a legal parental/legal guardian forms as all of them were under 18. These forms were distributed and collected by their ICT teacher and therefore, their identity was also kept confidential. Both parental/legal guardian and participant consent forms were approved by UREC before the studies were conducted.

The consent forms used in the studies informed all participants that:

a) their answers will never be shared with anyone;
b) the data collected from them will be kept for research purposes only;
c) if at any time they wished to withdraw from the study, they were free to do so without providing a reason;
d) their anonymity is guaranteed and that their choice to participate or not in this research

to have no effect on their studies;

e) statistical information would be generated from their answers and some of the information gathered may be published as part of the project report; however individuals will not be identified and information will be kept confidential;

f) all data gathered will be held securely in accordance with the Data Protection Act until the research project is completed.

In the Cyprus and the PGS studies, participants were asked to use a unique number they selected before participating whereas participants in the Greenwich study used their university usernames. None of the participants were asked to write their names on the consent forms. Participants in the Cyprus and in the PGS studies used their unique number in the pre and the post questionnaires so that it was possible to match their responses. A similar procedure was followed in the Greenwich study but, as already explained, rather than a random unique number, participants used their own username as this provided more control over the study.

Having signed the consent forms, participants were asked to complete two online questionnaires both before and after they played *Program Your Robot*. The questionnaires were available on SurveyMonkey – a popular online survey service that has international network security certificates (SurveyMonkey, 1999). The answers captured from all studies were kept online on SurveyMonkey and were not copied to elsewhere (such as to a hard-disk).

5.3 Experimental variables, research questions and hypotheses

As per quantitative experimental research, this study involves independent and dependent variables as well as research questions and hypothesis. This section first describes independent and dependent variables collected from all studies and then structures research questions previously discussed in Chapter 4 Section 4.2 into the experimental design of the research. Finally, a null and an alternative hypothesis are generated for each research question in order to analyse these questions statistically.

Variables	The Cyprus study	The Greenwich study	The PGS study
Independent Variables	Age, gender	Age, gender, ethnicity, mathematical qualifications	Ethnicity
Dependent Variables	1. Attitude to learning through playing games. 2. Intrinsic motivation for learning programming. 3. Problem solving abilities 4. Skills that encompass Computational Thinking 5. Ability to visualise programming constructs from given problems 6. Knowledge on programming constructs introduced in the game 7. The difficulty of programming	1. Attitude to learning through playing games. 2. Intrinsic motivation for learning programming. 3. Problem solving abilities 4. Skills that encompass Computational Thinking 5. Ability to visualise programming constructs from given problems 6. Knowledge on programming constructs introduced in the game 7. The difficulty of programming	1. Attitude to learning through playing games. 2. Intrinsic motivation for learning programming. 3. Problem solving abilities 4. Skills that encompass Computational Thinking 5. Ability to visualise programming constructs from given problems 6. Knowledge on programming constructs introduced in the game

Table 5.2 – Dependent and Independent variables in the studies.

As shown from Table 5.2 there was two types of variables in all three studies: *independent* and *dependent* variables. These variables radically changed from one study to the other as participant population varied extensively. The *independent variables* were collected to categorise different data sets and the *dependent variable*s were captured to test the effect of obtained data to these categories. The independent variables in the Greenwich study were age, gender, mathematical qualifications and ethnicity. The Cyprus study involved only age and gender as ethnicity and mathematical qualifications of participants were not collected. Correspondingly, age and gender variables were kept constant in the PGS study because all participants were 15 years old girls and therefore, the only independent variable was ethnicity.

The dependent variables in the studies were the answers given to research questions by

participants: attitude to learning programming through playing games, intrinsic motivation for learning programming, key computer programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops), problem solving abilities and finally the ability to visualise programming constructs from given problems. Consistent with the research questions, each key computer programming construct (i.e. sequence, functions, decisions and loops) is observed separately in all three studies. Correspondingly, the difficulty of computer programming is collected in the Cyprus and the Greenwich studies; however this was not considered in the PGS due to the reason that none of the participants were enrolled to a computer programming course.

The data types of independent variables are extensively different from one another. Gender and ethnicity independent variables are nominal because these values could be assigned as numbers but these numbers cannot be ordered or measured meaning that they are just labels (for example, male can be coded as 0; female as 1). Age range is continuous data because it can be counted, ordered and measured on a continuum or scale. Moreover, a mathematical qualification is discreet data as it can be counted, ordered and unlike the age range it is precisely measurable. Further to this, all dependent variables used in this study are ordinal data as they can be ordered and counted, but not measurable. The Likert scale used in the studies ranged from 0 to 5 and each rating indicates more satisfaction than the rating before it (such as a strongly agree indicates more satisfaction than an agree response). The rating scale used provided the ability to order and count the data obtained from the participants but the distinction between the rating points is not measurable. As an example, the difference between a strongly agree and an agree response can be less than the difference between a strongly disagree and disagree response. Therefore, all observations gathered from the participants exist on an ordinal scale.

#	Research Question	Null Hypothesis (Ho1)	Alternative Hypothesis (Ha1)
1	Is there a difference in students' attitude to learn computer programming through playing games between the pre and the post study?	There is no significant difference in students' attitude to learn computer programming through playing games between the pre and the post study.	Students' attitude to learning computer programming through playing games is significantly increased between the pre and the post study.
2	Is there a difference in students' intrinsic motivation to learn computer programming between the pre and the post study?	There is no significant difference in students' intrinsic motivation to learn computer programming between the pre and the post study.	Students' intrinsic motivation to learn computer programming is significantly increased between the pre and the post study.
3	Is there a difference in students' understanding of "programming sequence" in computer programming between the pre and the post study?	Students felt no significant difference in students' understanding of "programming sequence" between the pre and the post study.	Students felt that their understanding of "programming sequence" significantly increased between the pre and the post study.
4	Is there a difference in students' understanding of "functions" (methods) in computer programming between the pre and the post study?	Students felt no significant difference in students' understanding of "functions" (methods) between the pre and the post study.	Students felt that their understanding of "functions" (methods) significantly increased between the pre and the post study.
5	Is there a difference in students' understanding of "decision making" in computer programming between the pre and the post study?	Students felt no significant difference in students' understanding of "decision making" between the pre and the post study.	Students felt that their understanding of "decision making" significantly increased between the pre and the post study.
6	Is there a difference in students' understanding of "loops" in computer programming between the pre and the post study?	Students felt no significant difference in students' understanding of "decision making" between the pre and the post study.	Students felt that their understanding of "decision making" significantly increased between the pre and the post study.
7	Is there a difference in students' perception of their problem solving abilities between the pre and the post study?	There is no significant difference in students' perception of their problem solving abilities between the pre and the post study.	Students felt that their problem solving abilities significantly increased between the pre and the post study.
8	Is there a difference inn students' perception of their ability to visualise programming constructs from given problems between the pre and the post study?	There is no significant difference in students' perception of their ability to visualise programming constructs from given problems between the pre and the post study.	Students' perception of their ability to visualise programming constructs from given problems is significantly increased between the pre and the post study.
9*	Is there a difference in students' perception of difficulty of computer programming between the pre and the post study?	There is no significant difference in students' perception of difficulty of computer programming between the pre and the post study	Students' perception of the difficulty of computer programming significantly decreased between the pre and the post study

9* this research question was not asked in the PGS study.

Table 5.3 – Showing research questions, null and alternative hypothesis used in the studies.

As shown from Table 5.3, the research questions previously described in Chapter 4 section 4.2 were reorganised according to the experimental structure of the studies. Additionally, for each, a null and an alternative hypothesis were added to each research question in order to statistically analyse the data captured for these research questions.

Although the same set of research questions are used in the studies, it was not possible to investigate each research question for all three studies at once through inferential statistics because of the diversity in the participant population. As mentioned previously in section 5.1, participants in the studies were considerably different from each other in terms of age, education level and their background in computer programming. Therefore, it was not possible to investigate a research question for all three studies at once simply because the independent variables in the studies are not the same.

5.4 Threats to the Validity of Findings

There are various factors that might affect the cause-effect relationship or outcome of an experimental research. These factors often jeopardize the validity of research findings and for this specific reason they are often referred to as "*threats*" (SRM, 2006). This section discusses how these threats can have a potential effect on the outcomes of the studies as well as what precautions were taken to minimise them.

There are two main categories of validity: *internal validity* and *external validity*. It is crucial to clarify that threats that apply to internal/external validity are not all or none, black or white, present or absent in this study (or arguably in any experimental research). The validity of an experimental research varies along a continuum in various degrees and often threats cannot be eliminated completely. Additionally, the findings of a study can be affected through threats in a variety of ways, which are not always predictable. Henceforth, the aim here is to demonstrate a well-structured plan to achieve the best cause-effect relationship possible between the independent and dependent variables as well as to minimise the effect of confounding variables to a certain degree that would not bias the outcome of the studies.

Assessment Step	Process	Decision
Statistical analysis	Find statistical significance (i.e. statistical results are valid i.e. if P value is less than 0.05)	If statistical results show there is a difference due to game intervention, move to internal validity assessment. If not, stop here.
Internal Validity	Evaluate internal validity based on research design and procedures followed during the studies.	Does the difference between the groups depend on effects of confounding factors or bias? Validate history, maturity, mortality and regression threats. If participant quotes reflect the same outcome as closed-ended questions, move to external validity assessment. If not, stop here.
External Validity	Generalise the findings obtained in the studies	Do the people, time and location factors have a major impact on the findings of research? Decide whether or not a strong external validity exists.

Table 5.4 – Steps for assessing the validity of experimental findings.

Table 5.4 shows the assessment plan for the validity of findings. As shown from the table, it is first intended to analyse the results of each study through applied inferential statistics. Having analysed the data using inferential statistics, it is aimed to use internal validity on the results of the studies to ensure the impact of the game and any benefits that can be derived from it. This is planned to be done through investigating the statistical data based on different internal validity categories.

Should the results demonstrate strong reasons to believe internal validity, an external validity evaluation will be carried out to investigate whether or not the findings of the studies can be generalised. Additionally, it is planned to report participant responses to open-ended questions in order to decide whether or not these responses are consistent with the answers given to closed-ended questions. The following sections describe the internal and external threats that are recognised in this research.

5.4.1 Threats to internal validity

Threats to internal validity refer to any factor that can be ruled out as a rival explanation to an association of a cause-effect relationship in an experimental study. These are usually the confounding variables (such as people's historical background or previous experiences) behind the cause-effect relationships. Threats to internal validity are divided into six main categories: history threat, maturation threat, testing threat, instrumentation threat, mortality threat and regression threat (Slack & Draugalis, 2001; SRM, 2006). However, only four of these categories (history, maturation, mortality and regression) are considered in this research as the other two (testing, instrumentation) are not major threats in this research due to the structure of the studies (described below).

The *history threat* is related to participants' background knowledge and past experiences. This threat might play a crucial role in the outcome of the studies as a specific historical event or a chain of events could cause a result that is not directly related to *Program Your Robot*. As an example, some participants might have played a similar (serious) game to *Program Your Robot* previously and as a result of this, they might overrate the game-play. Additionally, participants' background in computer programming might have a huge impact on how well they can perform in the game.

In order to measure the impact of the history threat, participants' previous experiences in computer programming were captured in all studies. It is aimed to compare participants with good computer programming knowledge with those participants with little or no programming knowledge in order to assess whether or not a history threat will impact on the findings of the studies. It is anticipated that, in a study where the history threat does not impact on the outcome, the participants with little or no computer programming knowledge will learn more from the game, i.e. more than the participants with a good knowledge of computer programming. In addition to this, all participants provided information whether they have ever used a video game for educational purposes rather than entertainment, as well as their attitude to learning programming through playing games. All of these questions will be analysed for internal validity reasons in order to decide whether or not an historical threat would affect the outcome of the studies.

The *maturity threat* is similar to the history threat with one distinct difference, that is, rather than a historical event, *the growing up effect* biases the outcome of the study. Maturity threat involves the events that transpire in participants' lives over a period of time rather than a specific or chain of historical events (SRM, 2006).

In this research, the maturity threat is related to the participants' habit of playing games. As participants' maturity differs, their attitude to learning through games can also change and thus, this might affect the outcome of the studies. In other words, the participants who often play games can perceive *Program Your Robot* different than others. As an example, 15 years old participants in PGS study are more likely to play games than university students because teenagers tend to have more spare time to manage than young adult/adult students. Correspondingly, the maturity of a university student may not be the same as the maturity of a 15 years old pupil and this might affect how they perceive a game based learning environment.

As participants are extensively different in terms of age, ethnicity and their cultural background, their maturity to accept learning through playing games can vary widely. In order to measure the maturity threat, it is proposed to compare the participants responses regarding their computer programming knowledge among those participants who play games often and those do not play games often.

The *testing* and the *instrumentation* threats operate in a *pre-test and post-test design* where the test scores of participants are matched to observe an improvement. The *testing threat* occurs when a test score is improved because participants repeated the tests rather than the effect of an intervention. This threat is particularly concerned when researchers apply exactly the same test both in pre-test and post-test, in a study. The *instrumentation threat* is the opposite of this and operates when researchers use alternative forms rather than the identical tests in pre and post-test study design. The instrumentation threat often impacts on the outcome of a study when the alternative test (post-test) used is not at the same level of difficulty as the main test (pre-test).

Both the testing and instrumentation threats are not major threats in this study mainly because: a) this research does not use a *pre-test post-test design* in this study. As argued in section 5.1.2, the experimental design of this research is a *pre-study post-study design* where participants themselves decide whether or not their knowledge or attitude is improved; b) this research does not use score points from participants to match questions in the pre and post study. Each question in the questionnaires is evaluated independently and there is no right or wrong answer to choose.

The testing threat does not apply to this study because none of the studies uses exactly the same questionnaire in the pre and the post study. Correspondingly, the instrumentation threat operates at a minimal level because there is no right or wrong answer for questions in the studies. Therefore, testing and instrumentation threats are minor confounding variables in this study.

The *mortality threat* endangers this study when/if participants drop out in the middle of a study because people who drop out are often tend to provide negative feedback. When drop rates are equal or higher than non-drop rates then the validity of findings become debatable. The degree of mortality can be measured by comparing the dropout rates against non-dropouts. If there are no major differences between the two groups then it can be assumed the mortality threat does not bias the results.

The *regression threat* (also called regression to the mean) is arguably the most difficult internal validity threat to control in this research. The threat occurs when non-random participants are selected and when two measures (a pre-study and a post-study) are poorly correlated in a research (Slack & Draugalis, 2001). In other words, a regression threat occurs when the independent variables bias the results of the study. This is sometimes referred to as *"you can only go up from here"* phenomenon as it refers to an increase in post-study results relative to the population even if no intervention is given to participants (SRM, 2006). The corollary to this might be that the people who scored worst in the post-study might not be the same people who scored badly in the pre-study which would indicate that the intervention used actually made them worse relative to the population.

In order to measure the regression threat, a multiple linear regression (MLR) analysis is proposed to be used in this research. MLR is a statistical analysis method that measures the effect of multiple independent variables on a mean score of a dependent variable so that it can be detected whether or not the selected population bias the outcomes of a study. In this research, it is designed to look at the effect of age, gender and mathematical qualifications of participants on their perception of their computer programming knowledge gained from playing *Program Your Robot*.

Finally, it is crucial to make clear that the primary consideration in internal validity is to observe whether or not the changes between the pre and post study can be truly linked to the game environment and not to other possible causes. The main aim in internal validity is not to remove these threats but to ensure that they do not bias the outcome of the research at a critical level so that the validity of the work can be related to its natural environment. The internal validity of the studies is further discussed in Chapter 7 Section 7.1.

5.4.2 Threats to external validity

Threats to external validity refer to the estimated truth of a conclusion derived from our studies and can be categorised into three: people, place and time. It involves generalisation of

the conclusion of an experimental study based on population selection and ecological background (Shuttleworth, 2009). In other words, it is the degree to which the conclusion obtained from a study will hold for other people in other places at other times (Calder, Phillips & Tybout, 1982).

External validity is accessed after a successful establishment of internal validity and it is often reported that studies which involves "a measure of attitude and interest" are very susceptible to this threat (Slack & Draugalis, 2001). Despite this fact, this study was designed to have a strong external validity mainly because of two reasons: a) the experimental structure of this study is replicated three times in different places on different people at different times without any major differences; b) participants in all studies were randomly selected regardless of their background in programming or in games.

External validity threats are most likely to occur when the groups are not randomly selected (Shuttleworth, 2009). Additionally, if the targeted population of a study is a small subpopulation within a larger population then the results may not establish generalizability and thus it is often suggested to replicate the study to observe whether or not results are adequate (Slack & Draugalis, 2001). As the same study is repeated at multiple times with minor changes on randomly selected participants, it is anticipated that the outcomes obtained from this research will have strong external validity.

5.4.3 Other threats

There are many other threats that can be considered when conducting an experimental study. Some of these are concurrent, construct, content and social interaction threats (Slack & Draugalis, 2001). Despite the fact that each of these threats has their own individual standpoints; most of these are correlated either to internal or external validity (or to both) in varying degrees (SRM, 2006). As an instance, construct validity refers to the assessment on how well the ideas and theories are applied into an intervention and whether or not different measures are used to examine these constructs. Threats to construct validity involves, but not limited to, poor thinking of concepts applied, poor definition of constructs measured and mono-operation bias (a single measurement of an intervention) all of which are also related to external validity. Correspondingly, social interaction threats are related to internal validity as these threats occurs because key people in the study (e.g. participants, researcher, teaching staff) are aware of each other's presence and the role they play in the research (SRM, 2006).

It is important to underline that it is not possible to entirely eliminate the possibility of these

threats simply because human interaction always has an impact on cause-effect relationships in varying degrees. Examining each of these threats individually would certainly create too many variables to consider in terms of this research and thus this might divert the study from its main purpose. Therefore, this research considers all of these threats as part of the internal and external validity of the findings. As argued previously, the validity of a piece of research is by investigating the degree to which a study accurately answers the questions it was intended to answers. Considering too many threats would certainly create a very specific study structure where the result could no longer be generalised. Therefore, none of the above mentioned threats are deliberately measured in this research.

5.5 Summary

This chapter outlined the structure of all three experimental studies conducted in addition to the pilot study designed as a precursor to these studies and it also explained the ethical issues regarding the studies and how these were addressed. The chapter further described in what ways studies differ from each other and what principles are considered when conducting these (i.e. population selection, research questions, independent and dependent variables). Finally, the chapter discussed the potential threats that may apply to the findings of the studies as well as outlining a plan on how to measure the impact of these threats.

The next chapter investigates empirical analysis of all the studies conducted. It first provides the raw data obtained from the pilot study as well as explaining the modifications made to the game as a result of the findings of the pilot study. Having performed this, the chapter presents an analysis of each experimental study and applies appropriate measurements to the responses obtained in order to evaluate the obtained data accurately through inferential statistics.

CHAPTER 6
ANALYSIS OF EXPERIMENTAL STUDIES

This chapter focuses on the analysis and evaluation of a pilot study as well as three empirical studies: the Cyprus, the Greenwich and the PGS studies.

Section 6.1 reports the observations collected from 25 participants in a pilot study which was specifically conducted to measure whether or not *Program Your Robot* had reached the stage where empirical studies could be carried out. Additionally this section shows participants' quotes to demonstrate a flow of game activities related to the computational thinking stages given in the game description. This section closes by describing the enhancements and modifications made to the game before conducting the empirical studies.

Section 6.2 discusses and analyses the first empirical study, that is the Cyprus study. This section first investigates the methods used to analyse the normality of data as well as describes why multiple methods are selected for identifying whether or not the data came from a normally distributed population. Having identified the normal distribution of data, each dataset is then interpreted in accordance with the related research question through using a parametric statistical measure (i.e. a paired samples t-test). The section then discusses the correlations among the computational thinking skills (i.e. conditional logic, algorithmic thinking, simulating, debugging and cooperation), and their associations with how far students progressed in the game, as well as how these skills are related to the computer programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops). Further to these, a series of scatterplots were generated to identify whether or not there are linear relationships between variables where strong correlations are identified.

Section 6.3 and section 6.4 follow a very similar structure to Section 6.2, but focusses on the Greenwich and the PGS studies respectively. As the distribution of data was found to be non-normally distributed in both the Greenwich and the PGS studies, a non-parametric measure (i.e. Wilcoxon signed ranks test) was used to analyse the datasets rather than a parametric measure (paired samples t-test). Section 6.3 also describes what technical difficulties were encountered in the PGS study and how these were overcome to conduct the PGS study.

The data obtained in the studies were analysed through the research question and their hypotheses, which are listed earlier in Chapter 5 Section 5.3, by either accepting or rejecting

the null hypotheses. The output of the stats regarding the conducted rigorous studies (i.e. presented in Sections 6.2, 6.3 and 6.4) are generated from the IBM software package used for statistical analysis (SPSS). The raw data gathered from the conducted studies were entered into SPSS by numbering the Likert scale answers collected from participants into relevant numbers. This included a conversion process that followed from "*strongly disagree*" (1) to "*strongly agree*" (5). Despite the fact that it is showen on bar charts, the "*not applicable to me / I do not know* (0)" choice was ignored when calculating the statistical outcomes of the studies. Having entered the independent variables as numbers to SPSS, the procedure for checking the normal distribution was undertaken in order to decide which statistical method to use for the analysis of the data.

Finally, each section closes with a summary of the list of findings obtained from the relevant empirical study.

6.1 Pilot Study

A pilot study of *Program Your Robot* was conducted to measure whether or not the game had actually reached the stage where a detailed structured evaluation could be carried out. The achievements and the high score system in the game were absent when the pilot study was conducted as these were under development. This section reports the feedback obtained from this pilot study and lists a series of changes made to the game, based on this feedback.

6.1.1 Pilot Study evaluation

Three different questions were asked to participants in the pilot study and all of these questions were non-obligatory as the intention behind them was to guide participants in providing their feedback. These questions were: a) specifying personal experiences about the game as they occur – both good and bad; b) reporting crushes and errors if they encountered any; and, c) stating the number of levels achieved in the game.

Twenty five students successfully completed the pilot study and the feedback obtained from the participants demonstrated that the overwhelming majority of them (18 out of 25) found the game well-suited for helping introductory computer programming students understand how introductory computer programming constructs work. Additionally, the reports from participants put forward some evidence that the game is beneficial in enhancing introductory computer programming students' problem solving skills.

One important aspect of the pilot study was that the participants involved were studying on

different degree programmes and therefore, their programming knowledge and skills varied considerably. It was anticipated that this would be beneficial in terms of evaluation as the feedback was obtained from participants with diverse backgrounds, knowledge and experiences. The following quotes from participants demonstrate that they found the game interesting and helpful for developing problem solving abilities specifically for learning introductory computer programming:

Student 1: *"As I am a programmer, I didn't find the game complicated or hard. The game felt straightforward and I found that the game puts across the idea of structuring a program. The functions could be considered as classes and the decision making is a Boolean value. Those are the basics of programming, a way to show how to simplify the way of coding a program."*

Student 2: *"In my point of view, this game was really good to introduce the fun of programming to students who want to study programming."*

Student 3: *"I have completed all levels in the game. I didn't have any problems as I found the commands easy to understand. As the game went on it became quite complex but I managed to understand the concept behind it."*

Student 4: *"In the robot game, I managed to play up to level 5 with a score of 38000. I found the game interesting to play as it was easy to follow the instructions. I think the interface is quite simple and not overly done. I had no major issues with the game."*

None of the participants stated that they experienced an error or a crash in the game. However, some participants reported bugs (degrading quality and performance problems of the game) and that almost all of them provided their suggestions regarding the game mechanics and user interface. Some of these suggestions are cited below:

Student 5: *"Having no achievements in the game was quite a let-down as games like this require some sort of reward for how you coded the robot."*

Student 6: *"It isn't clear that you need to activate the lights at the end of the run, if you run debug mode it doesn't find an error or tell you that you have missed the lights."*

Student 7: *"There is no option to return back to the main screen of the game if the players decide to stop playing half way through the game. The game has an auto save system which is*

impressive but it doesn't notify users of [sic] *such a system exist."*

Student 8: *"I had no major issues in the game but at times when the application is running the game/robot seems to slow down."*

Student 9: *"I felt that some dialogue boxes were unnecessary as it gave information players didn't need in order to complete a level. The dialogue boxes need to be simplified."*

Student 10: *"The game needs a high score page to reward the people who use guile and don't rush through completing the levels."*

These suggestions were used to support further development of the game in order to deliver an improved game-play experience.

A series of observations clearly indicate that participants evaluated the game at a critical level in considerable detail. A particular participant reported that he assessed the performance of the game on various machines and discovered that the game requires 2.00 ghz CPU processor or above in order to avoid any degrading quality or performance. Another participant noted that the game is developed as a Flash application (Adobe Flash, 2013) and thus is not accessible through various mobile devices due to the technical restrictions of the underlying technology.

Further to these, an in-depth analysis of the reports provided additional evidence that the game actually fostered the type of computational skills intended to be encompassed by the game-play. Although the pilot study was not intended to assess self-rated perception of learning or educational effectiveness of the game, some excerpts from participants clearly put forward evidence that the game encouraged them to use conditional logic, algorithm building, simulating solutions, debugging and socialising during their game-play. Particular student quotes are cited below to demonstrate a flow of game activities relating to the computational thinking stages from the game description:

Associated computational thinking skill: conditional logic

Student 11: *"I tried all sort of tricks using decision making instruction but I failed going any further than level 4 probably because of my poor problem solving skills☺. Nonetheless, it was good fun crossing the first 3 levels. I liked the fact that the further I was going the more sense it was making."*

Student 12: *"I enjoyed playing the game and it enhanced my knowledge towards methods and how to call declared functions. Overall, I thought the game encourages you to think logically and was really entertaining at the same time."*

Associated computational thinking skill: building algorithms

Student 13: *"The game is very well designed and it is one of the games which need a lot of thinking. I got total score of 30750. I didn't experience any errors while finishing this game and it was very easy. In my point of view this game was really good to introduce the fun of programming to students who want to study programming."*

Associated computational thinking skill: debugging

Student 14: *"I found debug button useful because it provides messages when I forgot to call a function. However, when I ran the debug mode it didn't find an error or tell me that I have missed the lights or I could not progress until I have done it."*

Associated computational thinking skill: simulation

Student 15: *"The game is very well thought out, for example, the demonstration of decision making logic through if statement was a well thought out example, and the graphical demonstration of this concept is quite creative."*

Student 16: *"The game is not difficult as you have to pre-plan what steps and where to turn in order to collect key items and land on a teleporter to complete the stages. However, whilst playing on level 4, I planned my predicted movements and as I began to run simulation I was confronted with a confusing message about degraded performance. Overall, the game has some issues that need to be addressed but I believe it is a fun way in order to beginners to understand the concept of programming."*

Student 17: *"I thought that the whole idea behind the game is a good one and I found that using it was quite enjoyable because it included one of the very fundamental premises for teaching programming which is motivating students to continue through regular reward for accomplishment."*

Associated computational thinking skill: socialising

Student 18: *"The game needs a high score page to reward people who use guile and don't*

rush through the screen. Nonetheless, I enjoyed playing it because I competed against a friend of mine."

6.1.2 Modifications incorporated from Pilot Study

As seen from above, the results of the pilot study were positive and beneficial. The feedback obtained from participants was used to improve *Program Your Robot* before carrying on to the empirical evaluation stage. The changes incorporated from the feedback obtained from the pilot study are:

a) an achievements section was developed to reward players after they discover five good practices in programming;

b) a high score chart was designed where players can submit their scores and share it with other players. The participation in the high score chart is left as optional so that players can submit their scores when they really do well;

c) the interface used for a decision making command originally offered three different options (i.e. if robot faces an enemy robot, if robot stands on the edge of the platform, if robot faces a wall). This was reprogrammed and reduced to only one option (i.e. if robot faces an enemy robot) due to the reason that the decision making command was found misleading;

d) a dialogue message box was designed to prompt players who forgot to use the light command when the robot stands on the teleporter;

e) debug mode was re-designed to show currently executing commands;

f) text tutorial instructions were simplified and made optional;

g) video tutorials were added at the beginning of each level;

h) the screen size, text size and colours, graphical glitches (i.e. obstructing objects in the game), buttons, tutorial screens and pop up messages were enhanced. The screen size was expanded, a full screen option was added and the colours used in the game were made more vivid than before;

i) the time based animations were converted to tween based animations in order to

provide a smooth gaming experience.

The reports gathered from the pilot study provide evidence that *Program Your Robot* has the potential to enhance the problem solving abilities of students who are learning introductory computer programming. As a result, the game was improved by incorporating all of the suggestions raised in the pilot study.

Having successfully enhanced the game-experience, a set of rigorous studies were arranged in order to provide a systematic and structured evaluation of the game.

6.2 The Cyprus study evaluation and statistical analysis

This section provides a detailed analysis of the results obtained from the first structured study, which is the Cyprus study. The section first analyses the distribution of data collected and then divides the data into different subsets so that each dataset can be interpreted in accordance with the related research question. The section then discusses the correlations among the computational thinking skills (i.e. conditional logic, algorithmic thinking, simulating, debugging and cooperation), whether or not these skills are related to achieving high levels in the game, and how these skills are related to computer programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops). A series of scatterplots were also created to show the direction of relationships where strong correlations are identified.

A total of 75 students participated in the Cyprus study, 7 of which had completed the pre-study but not the post-study. As a result, 68 valid comparable responses were gathered at the end of the study. Among the valid responses (N=68), 44 (67.7%) came from male students and 24 (35.3%) from female students. The age of the students ranged between 18 and 29, where 50 (73.5%) of them were between 18 – 24 years old; and 18 (26.5%) of them were between 25 – 29 years old. All students were studying towards an Information Systems undergraduate degree and the study was conducted five weeks after they started their computer programming course.

As shown in Figure 6.1, one of the first questions directed to participants in the pre-study was whether or not the difficulty of programming was a key reason they thought about giving up their degree programme. There are two aspects to this question: a) to define how many students have thought giving up their degree programmes since they started, and b) how many of these thought that the difficulties of learning computer programming was a key reason for

this.

Whilst a total of 46 (67.6%) students have never thought of giving up their degree programme, 22 (32.3%) students had considered giving up their degrees since they started and 13 (19.1%) of these students thought that the difficulty of programming was a key reason why they considered giving up their degree programme. In other words, among the total number of students who considered giving up their degree programme 13 out of 22 (59%) reflected that they could give up their degree simply because of the difficulty of computer programming. These findings support the previous studies undertaken in this area (Bennedsen & Caspersen, 2007; Gomes & Mendes, 2007; Coull & Duncan, 2011) and provide additional evidence that current approaches to teaching computer programming requires support and improvement.

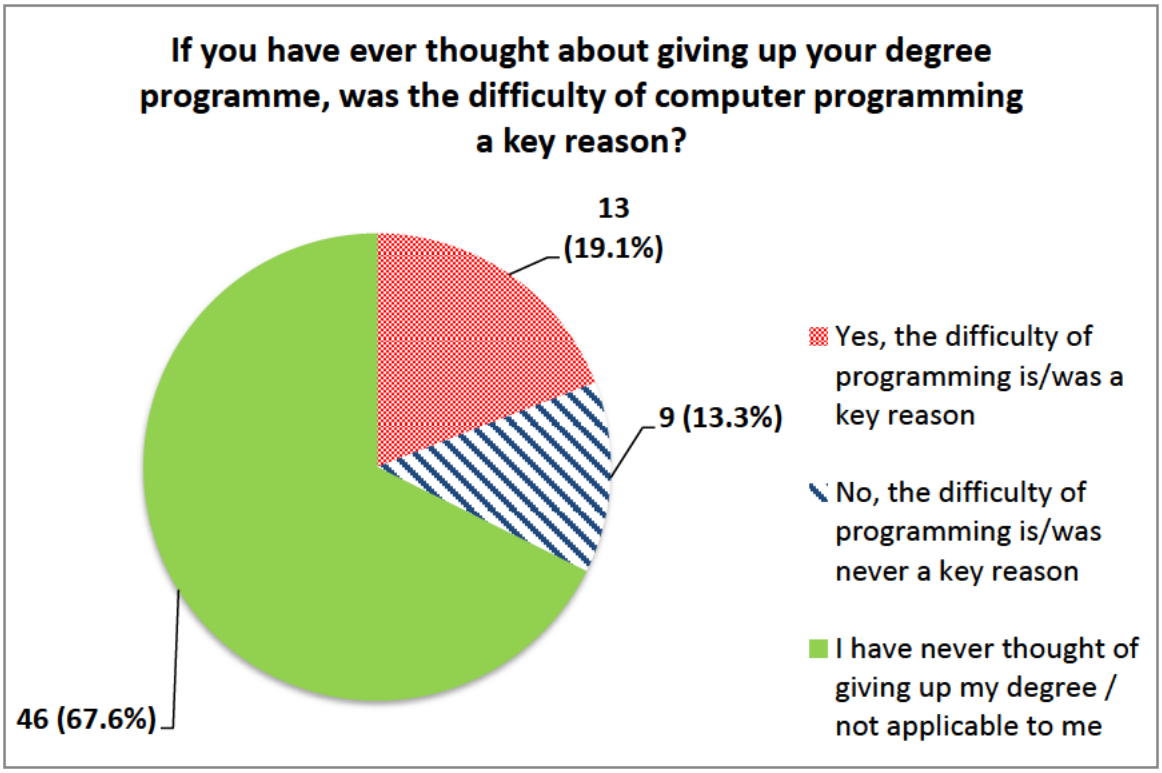

Figure 6.1 – Number of students in the Cyprus study who agreed that the difficulty of programming was a key reason to give up their degree programme in Information Systems.

In order to examine the rest of the results accurately in the context of the research questions, it was crucial to identify the correct method for an inferential statistical analysis. As the experimental structure is based on comparing a sample group's responses before and after they play the game, a statistical hypothesis test was needed to evaluate the before-and-after observations on the same subjects. Therefore, a procedure for carrying out either a paired t-test or a Wilcoxon signed ranks test (non-parametric equivalent of paired t-test) was

performed.

The paired t-test measurement was to be selected should the data captured fit a normal distribution and similarly the Wilcoxon signed ranks test would be available if the data captured did not fit a normal distribution. In other words, a paired t-test within a group or the non-parametric equivalent of this (i.e. Wilcoxon signed ranks test) was appropriate because there is just one observation for each combination of the ordinal values (i.e. a measurement in the pre-study and the same measurement in the post-study).

The distribution of data was defined by using four different methods: *Histogram*, *Quantile – Quantile plots*, *Skewness and Kurtosis normality check* and *Shapiro-Wilk test*. One-sample *Kolmogorov Smirnov* test was also applied but only provided a historical perspective rather than an accurate outcome.

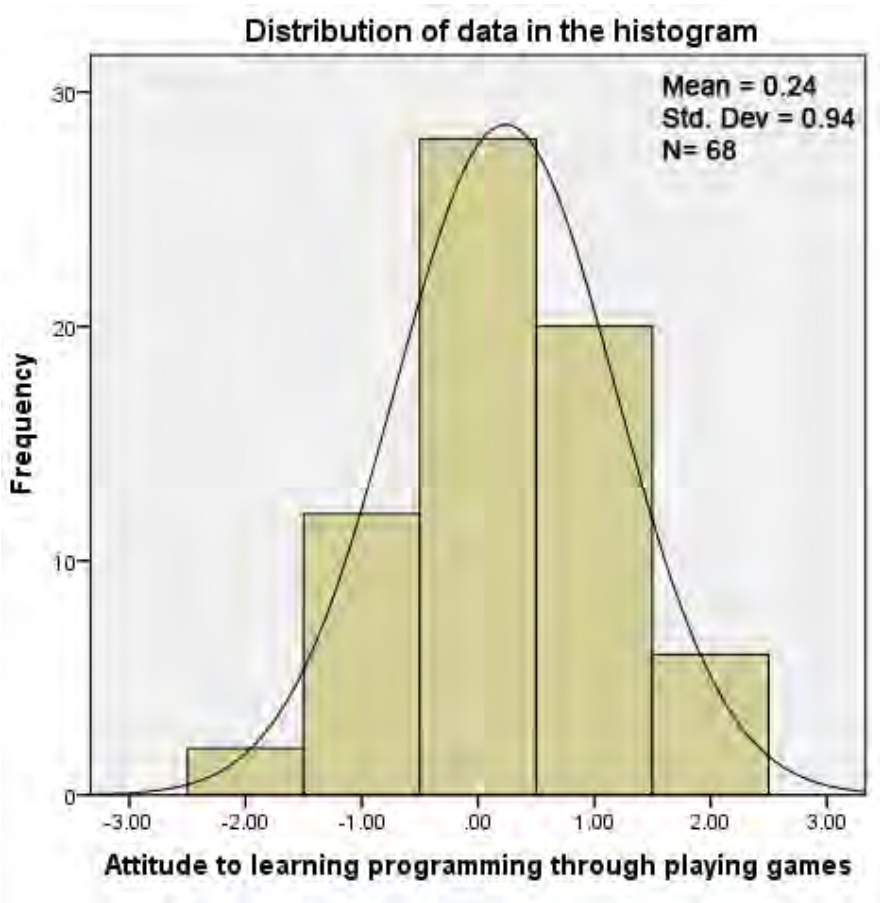

Figure 6.2 – Histogram showing distribution of data captured on the difference between attitudes to learn computer programming through playing games in the Cyprus study (Research question 1).

A histogram showing the distribution of data along with normal quantile – quantile (Q-Q) pilots was used as the main method to observe the distribution of data as these methods are

predominantly used for observing how close the distribution of data is to a normal distribution. A Skewness and Kurtosis normality check was also performed in order to identify skews and peak points. Figure 6.2 demonstrates the generated histogram by interpreting the data obtained for the first research question (i.e. difference in students' attitude to learn computer programming through playing games between the pre and the post study). As shown in the figure, the histogram does not incline at an angel to any horizontal or vertical position. In a perfect normal distribution, the distribution of data draws exactly a bell shaped curve and that half of the values obtained are located on the negative side (less than the mean value) where the other half is located on the positive side (greater than the mean value) of the curve. Additionally, in a perfect normal distribution, the population mean value is exactly 0 ($\mu = 0$) and the population standard deviation value is exactly 1 ($\sigma = 1$). The histogram generated from the data obtained from the Cyprus study demonstrates a close relationship to this as the distribution is neither too flat nor too peaked. The population mean value is very close to 0 ($\mu = 0.24$) and the standard deviation is very close to 1 ($\sigma = 0.94$). Moreover, it is often assumed that in a perfect normal distribution over 60% of the data is distributed within 1 standard deviation of the mean (between -1 and 1) and 95% within 2 standard deviation (between -2 and 2) (Moore, MacCabe & Craig, 2009). The histogram shown in Figure 6.2 supports these claims as all observations exist within the range of -2 and 2. Despite these findings, a histogram is not a very sensitive tool to recognise a normal distribution by itself and therefore, normal quantile-quantile (Q-Q) plots are used to investigate the distribution of data further.

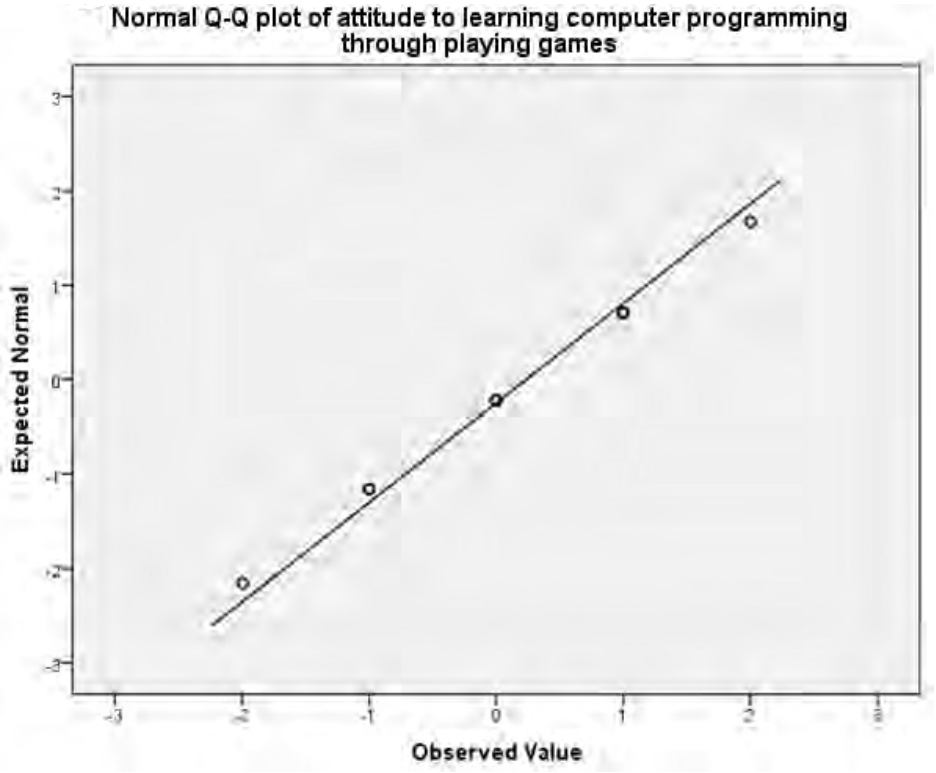

Figure 6.3 – Normal quantile-quantile (Q-Q) plots showing distribution of observations captured on the difference between attitudes to learn computer programming through playing games in the Cyprus study (Research question 1).

A quantile-quantile plot (often referred to as Q-Q plot) is an exploratory graphical device used to examine the validity of a distributional assumption for a data set (Christensen, 2011). It is often used with a histogram to accurately identify whether or not the shape generated in the histogram demonstrates a normal distribution as the histogram can only show skews and peak points visually regarding the distribution of a data set. When the data distribution approximately follows a normal distribution, the Q-Q plot roughly becomes a straight line with a slight positive slope. In a perfect normal distribution, the observations embrace the linear line of Q-Q plots (Christensen, 2011).

The Q-Q plots generated from the observations of the first research question in the Cyprus study is presented in Figure 6.2. As shown from this figure, the observations hug the linear line and no multiple clusters of observations are visible which means that the data obtained does not concentrate on specific points. In other words, the Q-Q plot provides strong reason to believe that the data came from a normally distributed population.

Despite the strong evidence obtained from the Q-Q plots, it was necessary to define the degree of Skewness and Kurtosis issues in terms of statistics, in order to ensure the

distribution of data does not have a heavier tail or higher peak than normal. To perform this, a Skewness and Kurtosis normality check was undertaken.

Descriptive Statistics

Skewness & Kurtosis Issues	N	Mean	Std. Deviation	Skewness		Kurtosis	
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Std. Error
Attitude to learning computer programming through playing games	68	.2353	.94817	**.061**	.291	**.251**	.574
Valid N (listwise)	68						

Table 6.1 – Skewness and Kurtosis normality check results on the difference between attitudes to learn computer programming through playing games in the Cyprus study (Research question 1).

Table 6.1 demonstrates the Skewness and Kurtosis of the attitude of the participants to learning computer programming through playing games between the pre and post study (Research Question 1). In a normal distributed data set, the Skewness value falls between -1 and 1, usually a value very close to 0. A Skewness issue can also be measured by multiplying the Skewness standard error with three and therefore, if the absolute value of the Skewness is less than three times of standard error then the distribution is accepted to have no major issues. Equally, the Kurtosis value is calculated by following the same rule (i.e. must be between -1 and 1). If both Skewness and Kurtosis conditions are satisfied, then the data is assumed to be in a normal distribution.

When Table 6.1 is analysed, it can be observed that both Skewness (0.06) and Kurtosis (0.25) values are very close to 0. Moreover, the standard error calculation for Skewness (0.291*3 = .87 > 0.06) and Kurtosis (0.574*3 = 1.7 > 0.25) satisfy that the data came from a normally distributed population.

Finally, a *Shapiro – Wilk* test was undertaken to support the previous claims on the normal distribution of the data set. The *Shapiro-Wilk* measure included one-sample *Kolmogorov Smirnov* test as well but this provided a historical perspective to the results rather than an accurate outcome because: a) recent studies show that the *Kolmogorov Smirnov* test is powerful when the sample size and significance level is large (Wilcox, 2011) and, b) various studies reference the *Shapiro-Wilk* test as the most powerful normality test and a better alternative to the *Kolmogorov-Smirnov* test (Mendes & Pala, 2003; Keskin, 2006; Farrel &

Stewart, 2006). Recent research in statistics also supported these findings and concluded that the *Shapiro-Wilk* test is the most powerful test for all types of distribution and sample sizes whereas the *Kolmogorov-Smirnov* is the least powerful test (Razali & Wah, 2011). As a result, the *Shapiro-Wilk* test was chosen as the primary test in this research.

Tests of Normality

Normality Check	Kolmogorov-Smirnov		Shapiro-Wilk	
	Statistic	Sig.	Statistic	Sig.
Attitude to learning computer programming through playing games	.216	.040	.903	.080

Table 6.2 – The Shapiro Wilk and the Kolmogorov Smirnov test results on the difference between attitudes to learn computer programming through playing games in the Cyprus study (Research question 1).

Before the *Shapiro-Wilk* test was carried out, a null hypothesis (i.e. H0 – the sample population is normally distributed) and an alternative to disprove this (i.e. H1 – the sample population is not normally distributed) were created. If the data comes from a normally distributed population, the significant value generated from the results would be greater than 0.05 ($p > 0.05$) and the result of this is the null hypothesis will be accepted. As shown from the results in Table 6.2, the significant value for the *Shapiro-Wilk* test is greater than 0.05 ($p = 0.08$), therefore the null hypothesis is accepted that is to say the data set came from a normally distributed population. The *Kolmogorov Smirnov* results was not used as the sample size of the Cyprus study (N=68) is not large enough to generate an accurate outcome for this test.

The normality tests discussed above have been undertaken for each research question and followed the same procedure in the same order. In all the research questions, it was found that the Skewness and Kurtosis values were always between -1 and 1 with minor Skewness and Kurtosis issues (sometimes the distribution skewed to right or slightly peak within the distribution). Additionally, a histogram was generated from the data obtained for each research question and the significant value for these was measured through the *Shapiro-Wilk* test. In all cases, it was found that the data set showed a distribution close to a normal distribution. Because the data set obtained for each research question has gone through the same normality tests, only the procedure for the first research question (i.e. difference in students' attitude to learn computer programming through playing games between the pre and

the post study) is described here as the rest of the research questions were analysed in the same way.

All four methods applied for identifying a normal distribution (i.e. *Histogram*, *Quantile – Quantile plots*, *Skewness and Kurtosis normality check* and *Shapiro-Wilk test*) provided strong evidence that the captured data in the Cyprus study fit a normal distribution. Therefore, two-tailed paired t-tests were chosen as the method to analyse the raw data gathered from this study. The t-tests were based on different pairs of sample data as laid out in the research questions which are accompanied by the hypothesis previously described in Chapter 5 Section 5.3.

6.2.1 Research Question 1 – Is there a difference in students' attitude to learn computer programming through playing games between the pre and the post study?

Figure 6.4 shows the responses given by the students regarding their attitudes to learning computer programming through playing games and Table 6.3 demonstrate the two tailed paired t-test results analysis for the same data set.

Prior to their game-play, 52 out of 68 students (76.4%) strongly agreed and agreed that learning programming through playing games could be useful, in the pre-study. Having played the game, this number increased to 60 (88.2%). These results show that the majority of students were very positive about learning how computer programming constructs work even before the game was introduced to them, and the study only slightly increased their attitudes. The number of undecided students decreased from 11 (16.1%) to 6 (8.8%) and the number of students who disagreed that learning programming through playing games was useful, decreased from 5 (7.4%) to 2 (2.9%) during the study.

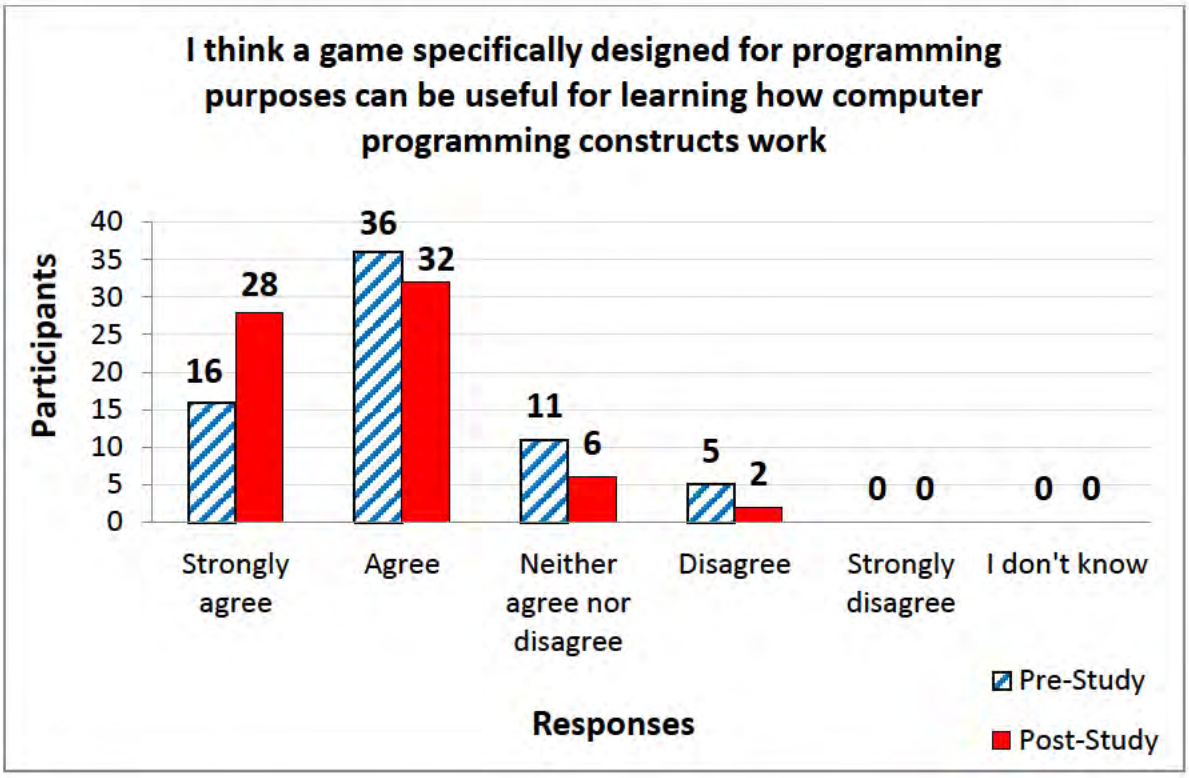

Figure 6.4 – Students' attitude to learning computer programming through playing games between the pre and post study in the Cyprus Study.

Paired Samples Statistics		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Pre video game attitude to learning programming through game-play	**3.94**	68	.808	.098
	Post video game attitude to learning programming through game-play	**4.22**	68	.750	.091

Paired Samples Test		Mean	t	df	Sig. (2-tailed)
Pair 1	Pre video game attitude to learning programming through game-play - Post video game attitude to learning programming through game-play	**-.279**	**-2.259**	67	**.027**

Table 6.3 – Paired t-test results of students' attitude to learning programming through playing games between the pre and post study in the Cyprus study.

As shown from Table 6.3, the difference between the pre and the post study results regarding the attitude of students to learning programming through game-play is not major (*M*

= 3.04 in the pre-study; M = 4.22 in the post-study; *mean difference* = 0.28). Although, the mean difference was not large, it was needed to determine whether or not this difference is significant. The results of the paired t-test shows that there was a significant difference in the attitude of students for learning programming between the pre-study (M=3.9, SD=0.80) and the post study (M=4.2, SD=.075) conditions; t (67) = 2.25, p = 0.02. As the 2-tailed significant value calculated from the difference between the pre and the post study is less than 0.05 (p = 0.02), the pre and post study do in fact differ. In this case, the null hypothesis for the groups does not differ in attitudes to learning programming through playing games is rejected. The sample-paired statistics suggest the alternative hypothesis that is to say the game slightly increased the attitude of students regarding learning programming through playing games during the study.

6.2.2 Research Question 2 – Is there a difference in students' intrinsic motivation to learn computer programming between the pre and the post study?

Figure 6.5 illustrates students' perception about their intrinsic motivation (motivation that is driven by interest and enjoyment) to learn computer programming both in the pre and post study. Out of 68 students, 21 (30.9%) students strongly agreed and 15 more (22.1%) agreed that they have intrinsic motivation to learn computer programming before they participated in our study. On the other hand, 3 (4.4%) students strongly disagreed and 13 (19.1%) more disagreed that they have intrinsic motivation to learn computer programming. It is crucial to notice that these findings support the previous work done in this area (Bennedsen & Caspersen, 2007) and that 47% of students (32 out of 68) who participated in this study either did not have intrinsic motivation to learn programming or were neutral about it.

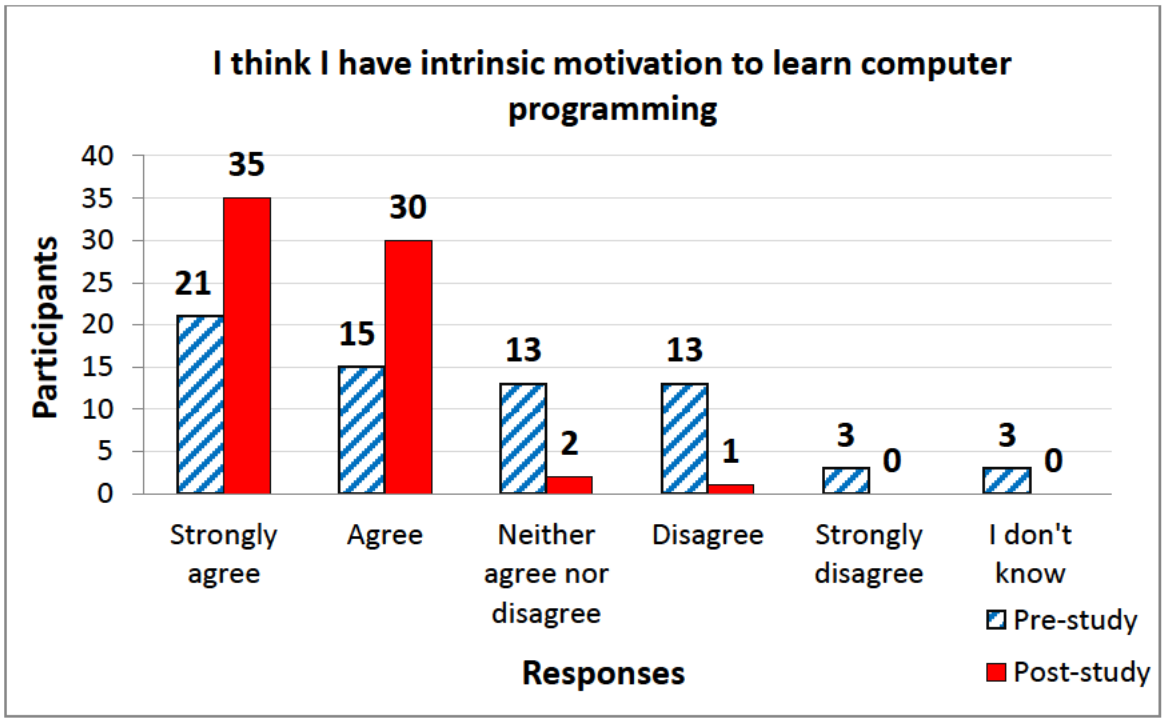

Figure 6.5 – Students' perception about their intrinsic motivation to learn computer programming between the pre and post study in the Cyprus study.

After participating in the game-play activity, students were asked the same question to understand whether or not the game impacted on their perception of intrinsic motivation. The results show that a total of 35 (51.5%) students strongly agreed and 30 (44.1%) students agreed that the game increased their intrinsic motivation to learn computer programming. In other words, 65 out of 68 students felt intrinsically motivated to learn computer programming after they played the game. Moreover, a total of 3 (4.4%) students didn't know whether or not they had intrinsic motivation to learn computer programming in the pre-study whereas this number decreased to zero in the post-study.

A paired sample t-test was conducted to identify whether or not the difference in the intrinsic motivation for learning programming between the pre and the post study is significant. As can be observed in Table 6.4, there was a significant difference in students' perception of their intrinsic motivation to learn computer programming between pre (M=3.40; SD = 1.37) and post (M=4.56; SD = 0.55) study, t (67) = 6.79, p = 0.000. The mean difference was found to be 1.15, which indicates that a large difference existed between the pre and the post study responses. Moreover, the p value is very close to zero (less than 0.000), and thus the findings provide evidence to support the alternative hypothesis which indicates that groups indeed differ in their intrinsic motivation to learn computer programming. Specifically, the results of the two tailed paired sample t-test suggest that when participants

played the game, their perception of intrinsic motivation to learn computer programming increased.

Paired Samples Statistics		Mean	N	Std. Deviation	Std. Error Mean
Pair 2	Pre computer programming motivation	**3.40**	68	1.373	.166
	Post computer programming motivation	**4.56**	68	.557	.068

Paired Samples Test		Mean	t	df	Sig. (2-tailed)
Pair 2	Pre computer programming motivation – Post computer programming motivation	**-1.162**	**-6.794**	67	**.000**

Table 6.4 – Paired t-test results of the difference between students' perception of their intrinsic motivation to learn computer programming between the pre and post study in the Cyprus study.

6.2.3 Research Question 3,4,5,6 – Is there a difference in students' perception of their knowledge in programming sequence, methods, decision making and loops between the pre and the post study?

Figure 6.6 illustrates the results of student responses based on key introductory programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops). Students were first asked to rate their current knowledge on the programming constructs according to their own perspective in the pre-study. The same students then played the game and afterwards rated their knowledge again in the post study. The results show that students felt their knowledge has been enhanced in all programming constructs, particularly in programming sequence and functions. The smallest improvement happened in loops which were predicted because this construct is introduced in the later stages of the game whereas programming sequence and functions are introduced during the early levels of the game. Hence, it was expected that not all students would be able to finish all levels.

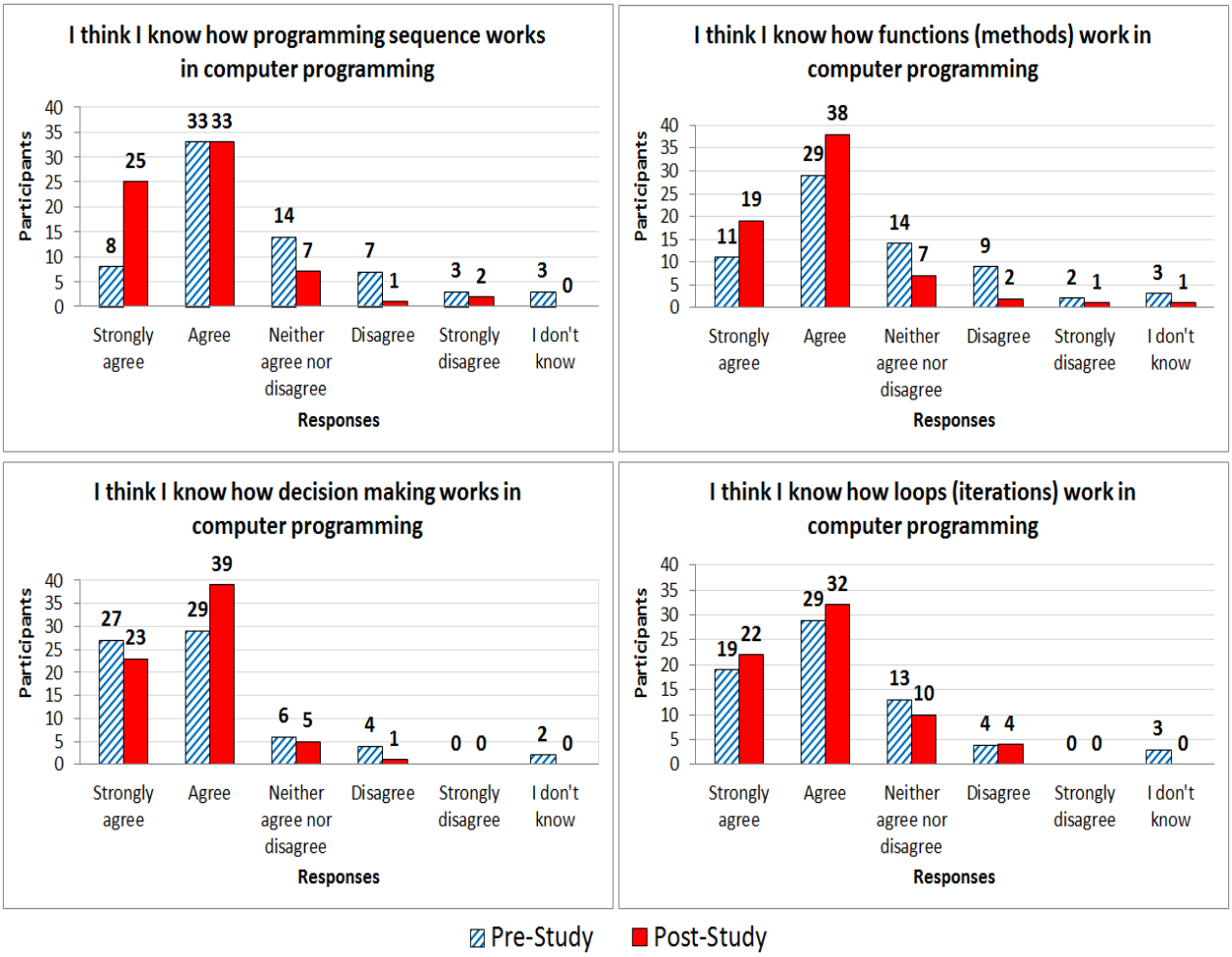

Figure 6.6 – Students' perception of their knowledge on programming constructs between the pre and post study in the Cyprus study.

In the pre-study, a total of 41 (60.3%) students strongly agreed and agreed that they knew the concept of programming sequence in computer programming. In the post-study, this number increased to 58 (85.3%) and the number of students who were undecided decreased from 14 (20.5%) to 7 (10.2%). Although the number of agreed students stayed the same, a large difference (25%) was identified in the number of students who strongly agreed that they know programming sequence. Additionally, a similar positive difference was observed in functions. Whilst 40 (58.8%) out of 68 students claimed they knew how functions work in the pre-study, this number increased to 57 (81.8%) after they played the game. Prior to the game-play experience, 14 (20.5%) out of 68 students strongly disagreed and disagreed that they knew how functions work in computer programming. Having played the game, this number decreased to 4 (5.8%) which is a considerable difference (14.6%). Further to this, an improvement in knowledge regarding loops and selection was also observed. A total of 56

(82.3%) students strongly agreed, or agreed that they knew decision making constructs in programming in the pre-study. After the game-play, this number increased to 62 (91.1%).

Decision making is also the only construct in which a decrease was observed in the number of strongly agreed students. To identify the reason for this, the responses from the pre-study were matched to the responses given in the post-study. The findings show that 4 (5.8%) students changed their opinion from strongly agree to agree. Although the exact reason for this is not known, one may conjecture that students realised they did not have as much knowledge on this construct as they thought, after the game play. The number of students who disagreed that they know decision making was reduced from 4 (5.8%) to 0 during the study. Moreover, 48 out of 68 (70.5%) students stated that they knew loops in computer programming in the pre-study, a total of 54 (79.4%) students either strongly agreed or agreed that their knowledge on loops has been enhanced after they played the game.

In addition to these, 3 (4.4%) out of 68 students stated that they had no prior knowledge regarding sequence, loops and functions before they played the game. 2 (2.9%) students also stated that they did not know about decision making in computer programming. Having played the game, only one of these students selected the "I don't know" choice. An investigation in students' responses revealed that 2 (2.9%) out of 3 agreed that their knowledge has been improved in sequence, decision making and loops after the game-play. The other student selected "neither agree nor disagree" choice for loops and decision making. When the gathered data was examined, it was found that this student never made it to the higher levels in the game where decision making and loops are introduced. Therefore, the student was unable to develop an understanding on decision making and loops because this student never saw them in the game.

A two tailed sample-paired t-test was carried out in order to identify whether or not the knowledge gain felt by the participants was significant. Table 6.5 demonstrates the results of the sample-paired t-test regarding the four key programming constructs presented in the game.

Paired Samples Statistics		Mean	N	Std. Deviation	Std. Error Mean
Pair 3	Pre Sequence Knowledge/Skills	**3.04**	68	1.387	.168
	Post Sequence Knowledge/Skills	**4.21**	68	.802	.097
Pair 4	Pre Functions Knowledge/Skills	**2.96**	68	1.491	.181
	Post Functions Knowledge/Skills	**4.03**	68	.897	.109
Pair 5	Pre Decision Knowledge/Skills	**3.41**	68	1.528	.185
	Post Decision Knowledge/Skills	**4.19**	68	.675	.082
Pair 6	Pre Loop Knowledge/Skills	**3.29**	68	1.477	.179
	Post Loop Knowledge/Skills	**4.04**	68	.818	.099

Paired Samples Test		Mean	t	df	Sig. (2-tailed)
Pair 3	Pre Sequence Knowledge/Skills - Post Sequence Knowledge/Skills	**-1.162**	**-5.825**	67	**.000**
Pair 4	Pre Functions Knowledge/Skills - Post Functions Knowledge/Skills	**-1.074**	**-5.194**	67	**.000**
Pair 5	Pre Decision Knowledge/Skills - Post Decision Knowledge/Skills	**-.779**	**-3.882**	67	**.000**
Pair 6	Pre Loop Knowledge/Skills - Post Loop Knowledge/Skills	**-.750**	**-3.724**	67	**.000**

Table 6.5 – Paired t-test results of the difference between students' perception of their intrinsic motivation to learn computer programming between the pre and post study in the Cyprus study.

The sample-paired t-tests indicated that the difference between the pre and the post study regarding programming sequence t (67) = 5.82; $p < 0.000$; functions t (67) = 5.19; $p< 0.000$; decision making t (67) = 3.88; $p< 0.000$; and loops t (67) = 3.72; $p<0.000$ is significant in all cases. The mean value regarding all programming constructs presented in the game increased considerably in the post-study and the highest increase happened in programming sequence (M = 3.04; SD = 1.38 in pre study, M = 4.21; SD = 0.80 in post study) and the lowest increase

happened in loops (M = 3.29; SD = 1.47 in pre study, M = 4.04; SD = .81 in post study). As illustrated in Table 6.5, the 2-tailed significant value calculated from the difference between the pre and post study regarding each programming construct is found to be less than 0.05 ($p = 0.000$). This provides strong evidence to reject the null hypothesis and accept the alternative hypothesis for research questions 3, 4, 5 and 6 that were stated in Chapter 5 Section 5.3. In other words, the paired-samples t-tests results provided strong evidence to accept that the difference in participants' perception of their knowledge regarding how programming constructs (programming sequence, functions, decision making and loops) work between the pre and the post study is indeed significant.

6.2.4 Research Question 7, 8 – Is there a difference in students' problem solving abilities and the ability to visualise programming constructs from given problems between the pre and the post study?

The final research questions were measuring the difference in students' problem solving abilities and their ability to visualise programming constructs from a given problem between pre and post study. In their seminal work, McCracken *et al.* (2001) provided evidence that many novice programming students need skill development in abstracting a problem from given definitions. They reported that students must identify relevant aspects of a problem statement before modelling those elements into an appropriate abstraction. Grounded within the previous work, research question 7 (participants' problem solving abilities) and 8 (participants' ability to visualise programming constructs from given problems) were asked to investigate whether or not a level of abstraction for developing problem solving abilities can be supported through *Program Your Robot*.

Students were first asked to rate their problem solving abilities in the pre-study and correspondingly the same question was asked in the post-study and the responses were matched. The results of the pre-study showed that a total of 55 (80.8%) students strongly agreed and agreed that they have problem solving abilities required for learning programming. Whilst 10 (14.7%) students had no opinion either way, 2 (2.9%) students disagreed and 1 student (1.4%) did not provide an answer. In total, 13 (19.1%) students either did not have an opinion or disagreed that they have these abilities. After their game-play, a total of 59 (86.7%) students strongly agreed or agreed that the game improved or has the potential to improve their problem solving abilities. The numbers of undecided students decreased from 10 (14.7%) to 7 (10.2%) and the number of students who disagreed decreased from 2 (2.9%) to

0. Only 1 student (1.4%) strongly disagreed that the game improves problem solving abilities. As a result, the raw data captured demonstrated that overwhelming majority of students thought the game increased their problem solving abilities.

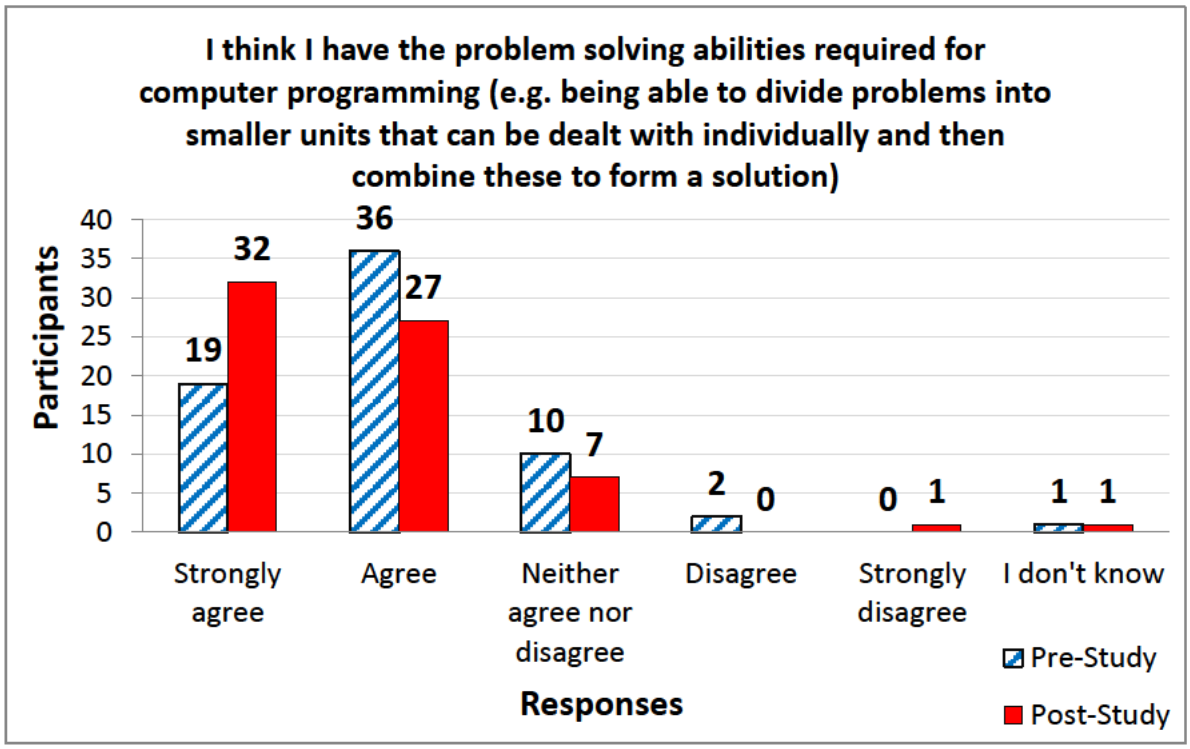

Figure 6.7 – Students' perception of their problem solving abilities between the pre and post study in the Cyprus study.

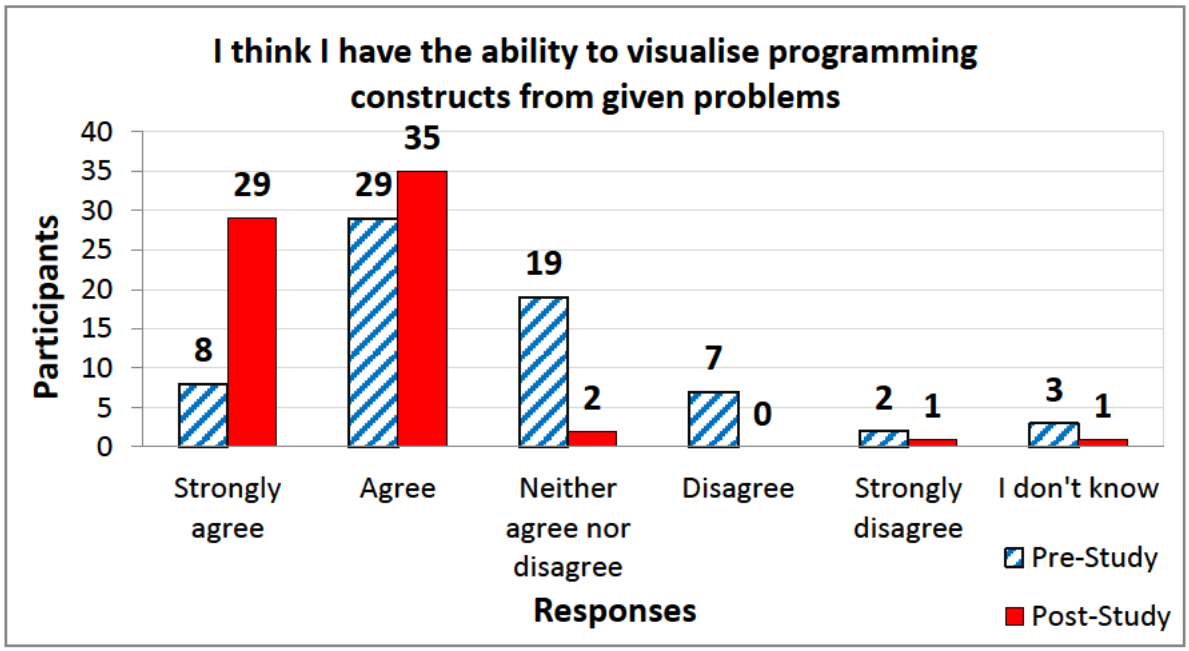

Figure 6.8 – Student's perception of their ability to visualise programming constructs from given problems between the pre and post study in the Cyprus study.

Further to their problem solving abilities, students' perception regarding their ability to visualise programming constructs increased during the study. As shown in Figure 6.8, the data obtained from the pre-study supported the arguments of McCracken *et al.* (2001) and showed that a considerable percentage (45.5%) of students were unable to visualise programming constructs from given problems. Before the study, 37 (54.4%) students strongly agreed or agreed that they have the ability to visualise programming constructs and this number is increased to 64 (94.1%) in the post-study. The results showed that the ability of students to visualise programming constructs extensively increased during the study as the difference between the two groups is vast (39.7%). Moreover, only 2 (2.9%) students were undecided after playing the game whereas this was 19 (27.9%) prior to the study. Finally only one (1.47%) student disagreed that the game does not help developing the ability to visualise constructs whereas this number was 9 (13.2%) in the pre-study.

As a result, the findings indicate that the students' perception of their problem solving abilities (5.9% difference between groups) and the ability to visualise programming constructs (39.7% difference between groups) greatly increased during the study. To measure whether or not these differences are significant, two tailed paired-samples t-tests were performed and the results are discussed below.

A paired-samples t-test was conducted to compare students' perception of their problem solving abilities in pre and post study conditions. The results show that there was a significant difference between the pre-study (M=3.24; SD = 1.14) and post-study (M=4.38; SD = 0.69) conditions t (67) = 6.93, p =0.000. Additionally, another paired-samples t-test was performed to examine students' perception of their ability to visualise programming constructs from given problems during the study. The findings provide evidence that there was a significant difference between the pre-study (M=3.16; SD = 1.26) and post-study (M=4.38; SD =.57) conditions t (67) = 7.46, p= 0.000. These results suggest that students' perception of their problem solving abilities and the ability to visualise programming constructs were improved during the study. In both cases, the mean value is increased considerably (Mean difference in problem solving is 1.14; Mean difference in visualising constructs is 1.22) and the 2-tailed significant value calculated from the difference between the pre and the post study is found to be less than 0.05 (p. = 0.000). In this case, the null hypothesis in research questions 7 and 8 (stated in Chapter 5 Section 5.3) which defends that there is no difference in this group's problem solving abilities and the ability to visualise constructs is rejected. Specifically, these results provide strong evidence to support the alternative hypothesis for research questions 7 and 8 which is that the difference happened in problem solving abilities and the ability of

visualising constructs is significant.

Paired Samples Statistics		Mean	N	Std. Deviation	Std. Error Mean
Pair 7	Pre problem solving abilities	**3.24**	68	1.148	.139
	Post problem solving abilities	**4.38**	68	.692	.084
Pair 8	Pre visualising prog. constructs	**3.16**	68	1.265	.153
	Post visualising prog. constructs	**4.38**	68	.574	.070

Paired Samples Test		Mean	t	df	Sig. (2-tailed)
Pair 7	Pre problem solving abilities – Post problem solving abilities	-1.147	**-6.938**	67	**.000**
Pair 8	Pre visualising prog. constructs – Post visualising prog. constructs	-1.221	**-7.468**	67	**.000**

Table 6.6 – Paired t-test results of the difference in students' perception of their problem solving abilities and the ability to visualise programming constructs between the pre and the post study in the Cyprus study.

6.2.5 Research Question 9 – Is there a difference in students' perception of the difficulty of computer programming between the pre and the post study?

As an additional research questions, students' perception on the difficulty of programming was also investigated. Initially, students were asked to rate the difficulty of programming according to their perception in the pre-study and similarly, a comparable question was asked in the post-study. The responses were then matched and the results are presented in Figure 6.8.

Prior to their game-play, none of the students who participated in the study rated the difficulty of computer programming as very easy and only 8 out of 68 (11.7%) students rated the difficulty of computer programming as easy. Whilst 28 out of 68 (41.1%) students were neutral, 26 out of 68 (38.2%) students found computer programming difficult. Additionally 4 out of 68 (5.8%) students indicated that they found computer programming very difficult whereas 2 (2.9%) students did not answer this question. Having played the game, 23 out of 68 (33.8%) students rated learning computer programming as very easy and 37 (54.4%) more

rated it as easy. The number of neutral students decreased from 28 (41.1%) to 6 (8.8%) and the number of students who found learning computer programming difficult decreased from 26 (38.2%) to 2 (2.9%). More interestingly, none of the students in the post-study rated learning computer programming very difficult after their game-play.

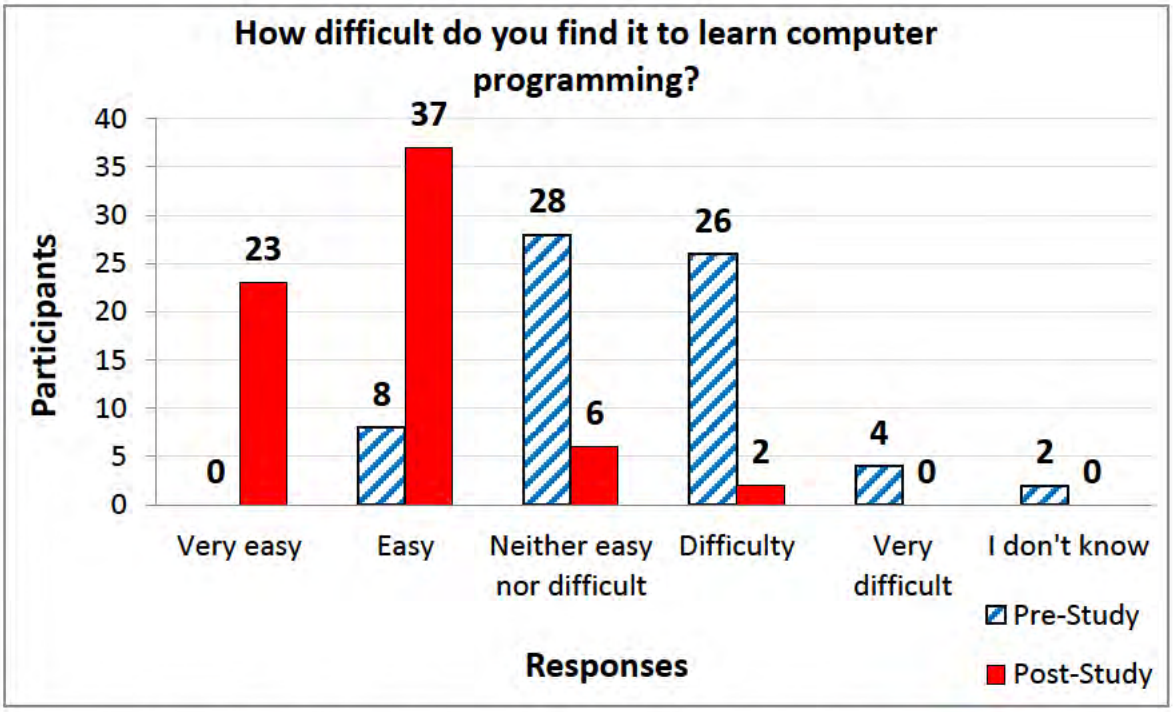

Figure 6.9 – Students' perception of the difficulty of computer programming between the pre and post study in the Cyprus study.

Paired Samples Statistics		Mean	N	Std. Deviation	Std. Error Mean
Pair 9	Pre computer programming difficulty	**2.41**	68	.966	.117
	Post computer programming difficulty	**3.60**	68	.626	.076

Paired Samples Test		Mean	t	df	Sig. (2-tailed)
Pair 9	Pre computer programming difficulty – Post computer programming difficulty	-1.191	**-8.359**	67	**.000**

Table 6.7 – Paired t-test results of the difference in students' perception of learning computer programming between pre and post study.

A paired-samples t-test was conducted to measure whether or not the difference identified in students' perception of difficulty of learning programming between the pre and post study is significant. The paired-samples t-test result indicate that the difficulty of programming was significantly higher in the pre-study (*M*=2.41; *SD* =0.96) than in the post-study (*M*=3.60; *SD* = 1.14), *t* (67) = 8.35, p=0.000. According to the ordinal scale used (i.e. 5 = very easy, 1 = very difficult), the data obtained from the study show that students rated the difficulty of programming easier in the post-study than in the pre-study. As the two-tailed significant value is lower than 0.05 (p=0.000), the null hypothesis in research question 9 which defends that there is no difference in group's perception of difficulty of computer programming is rejected.

The result of the paired-samples t-test provided strong evidence to support the alternative hypothesis, that is to say the group's perception of difficulty of computer programming is significantly different. The game intervention created a positive effect on students as the mean difference between the pre and the post study (*Mean difference* = 1.19) clearly indicate that students rated learning computer programming easier after they played the game than prior to their game-play.

6.2.6 Summary of findings regarding research questions

Research Question#	Pairs	Mean Difference (*M*)	Two tailed significant value (p)
1	Pre video game attitude to learning programming through game-play – Post video game attitude to learning programming through game-play	**0.28**	**0.027**
2	Pre computer programming motivation – Post computer programming motivation	**1.16**	**0.000**
3	Pre sequence knowledge – Post Sequence Knowledge	**1.16**	**0.000**
4	Pre functions knowledge – Post functions knowledge	**1.07**	**0.000**
5	Pre decision making knowledge – Post decision knowledge	**0.78**	**0.000**
6	Pre loop knowledge – Post loop knowledge	**0.75**	**0.000**
7	Pre problem solving abilities – Post problem solving abilities	**1.14**	**0.000**
8	Pre visualising constructs – Post visualising constructs	**1.22**	**0.000**
9	Pre difficulty of computer programming – Post difficulty of programming	**1.19**	**0.000**

Table 6.8 – Summary of samples paired t-test results of research questions used in the Cyprus study.

Table 6.8 demonstrates a summary of samples-paired t-test results of all the research questions used in the Cyprus study. The results show that the two-tailed significant value was lower than 0.05 (p=0.027) in the first research question and was lower than 0.001 (p=.000) for the rest of the research questions. This indicates that the findings of the samples-paired t-test provided strong evidence to reject the null hypothesis and accept the alternative hypothesis for each of the research questions evaluated above. The most significant difference between the groups happened in visualising programming constructs from given problems (*mean difference* = 1.22) and the least significant difference happened in the attitude of students' to learning computer programming through playing games (*mean difference* = .27). The findings suggest that students already had a good attitude to learning computer programming through playing games before they participated in the study (M = 3.94) and that the game interference had only slightly affected their attitude (M = 4.22). On the other hand, the samples-paired t-test results indicate that the majority of students felt their ability to visualise programming constructs from given problems noticeably improved after their game- play (M = 4.38).

As argued in Chapter 2 Section 2.2, various studies in Computer Science state that the ability to use abstraction is a core competence and an indicator of success for learning computer programming (Hazzan, 2003; Bennedsen & Caspersen, 2006; Kramer, 2007). The same studies also argue that those students who struggle to understand programming often lack the ability to use abstraction and cannot distinguish between conceptual and operational levels and how these two really relate to each other.

The results of the Cyprus study provide strong evidence that students perceived that they were able to visualise programming constructs better after playing the game than prior to playing the game. In other words, the game intervention provided a concrete representation regarding the four programming constructs (i.e. programming sequence, functions, decision making and loops) introduced in the game that allowed students explore how these work in a way that actually made sense to them. The validity of sample-paired t-test results and how this is related to the concept of abstraction is further investigated in Chapter 7 Section 7.1.

6.2.7 Statistical correlations

In addition to the samples-paired t-test, a Pearson product-moment correlation coefficient (also called Pearson's r) was used as a measure to look into the strength and direction of associations among computational thinking skills as well as between these skills and how well players played the game. As described in Chapter 2 Section 2.2, cognitive skills that

encompass computational thinking were defined as conditional logic, algorithmic thinking, debugging, simulation and socialising according to the literature available in this area (Wing, 2006; Berland & Lee, 2011). It is crucial to highlight that only the *cooperation* aspect of *socialising* is investigated in this research and as argued in Chapter 3 Section 3.3, the competition in the game was optional and therefore, was not investigated in the studies.

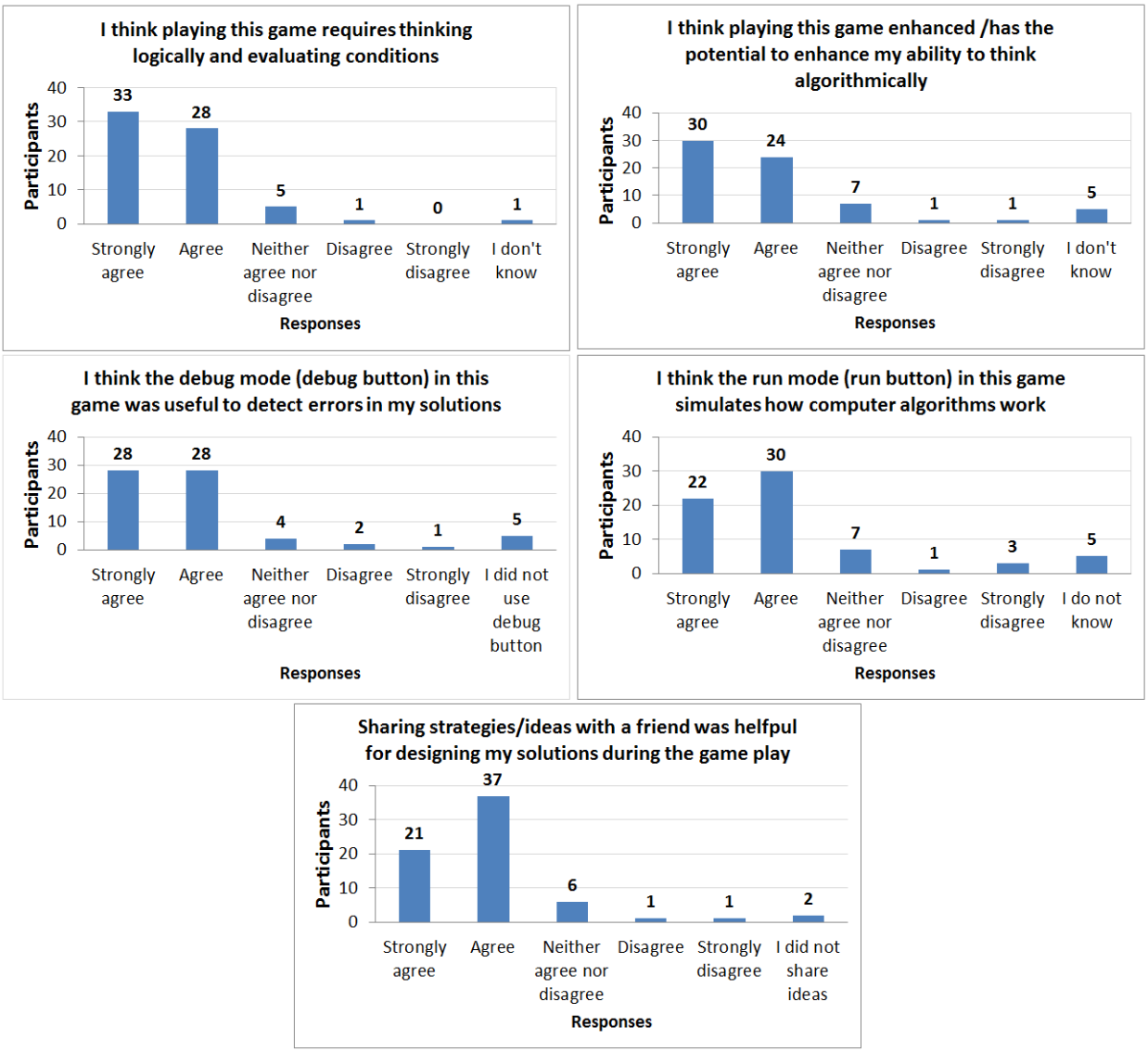

Figure 6.10 – Students' perception of how well computational thinking skills were presented in the game in the Cyprus study.

As there is no universally agreed way to teach computational thinking skills, it was simply not possible to ask students to rate them in the pre-study and thus all skills were asked to be evaluated after students played the game. Each of the five variables is examined through tests for normality similar to previous research questions and in all cases it was found that the responses came from a fairly normally distributed population with minor Skewness and

Kurtosis issues which did not impact on the outcome.

Figure 6.10 illustrates how well students think the game encompasses computational thinking skills. By asking questions about each of these skills autonomously, it was designed to identify whether or not students felt that their skills in computational thinking were enhanced during their game-play. 58 (85.2%) out of 68 students strongly agreed and agreed that the game requires conditional logic. While 5 (7.3%) students were neutral about this, only 1 (1.4%) student disagreed. Further to this, 54 (79.4%) out of 68 students strongly agreed and agreed that the game enhanced (or has the potential to enhance) their algorithmic thinking ability. Among these responses, 7 (10.2%) students were neutral and 2 (2.9%) students strongly disagreed and disagreed that the game enhanced (or has the potential to enhance) their ability to think algorithmically. Whilst 5 (7.3%) students indicated that they never used the debug button in the game, 56 (88.8%) out of 63 students strongly agreed and agreed that the debug mode (or debug button) was useful to them in detecting and handling errors in their solutions. Moreover, 52 (76.4%) out of 68 students strongly agreed and agreed that running their solutions in the game simulates how computer algorithms work in practice. 7 (10.2%) students were neutral and 4 (5.8%) strongly disagreed or disagreed that the run button does not simulate how computer algorithms work. Further to these, a total of 66 (97%) students shared ideas and strategies during their game-play. Among these students, 58 (87.8%) out of 66 strongly agreed and agreed that sharing ideas and strategies was useful for designing their solutions in the game.

A Pearson product-moment correlation coefficient was computed to assess the relationships among computational thinking skills as well as the relationship between these skills and the maximum level players reached in the game. The findings show that there is a positive correlation among all skills where some of these are significant and strong; others are not. Despite the fact that all correlations are in the positive direction, the associations between the maximum level students achieved and five categories of computational skills were always either weak or not significant.

When working with regional data that comes from a normally distributed population, a Pearson's product-moment correlation coefficient (often referred to as Pearson's r) is used to identify the strength and direction of correlations between two variables. A strong positive correlation is identified when Pearson's r value closes to positive 1, and similarly a strong negative correlation is defined when Pearson's r closes to negative 1. Although there are only crude estimates available for interpreting the strength of a correlation, it is generally accepted that there is a strong positive correlation between two or more variables when Pearson's r is

equal or greater than +0.7 (Song, 2007). Correspondingly, a modest strong correlation ranges from +0.49 to +0.69 and a weak positive correlation is accepted between + 0.2 and +0.39. Any correlation that ranges between +0.01 and +0.019 is often accepted as negligible or does not exist at all. Finally, the negative correlations also follow the same guidelines but with a negative value rather than a positive value.

Pearson product-moment correlation coefficient		Maximum game level participants reached	Conditional Logic and Evaluating Conditions	Algorithmic Thinking	Running and Simulating Solutions	Debugging and Handling Errors	Cooperation (Sharing ideas and strategies)
Maximum game level participants reached	Correlation Coefficient	1.000	.331*	.337**	.362**	.209	.189
	Sig. (2-tailed)	.	.010	.005	.002	.087	.122
	N	68	68	68	68	68	68
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.331**	1.000	**.867****	**.791****	.415**	.574**
	Sig. (2-tailed)	.010	.	.000	.000	.000	.000
	N	68	68	68	68	68	68
Algorithmic Thinking	Correlation Coefficient	.337**	**.867****	1.000	**.817****	.441**	**.631****
	Sig. (2-tailed)	.005	.000	.	.000	.000	.000
	N	68	68	68	68	68	68
Running and Simulating Solutions	Correlation Coefficient	.362**	**.791****	**.817****	1	.483**	**.670****
	Sig. (2-tailed)	.002	.000	.000	.	.000	.000
	N	68	68	68	68	68	68
Debugging and Handling Errors	Correlation Coefficient	.209	.415**	.441**	.483**	1	**.601****
	Sig. (2-tailed)	.087	.000	.000	.000	.	.000
	N	68	68	68	68	68	68
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.189	.574**	**.631****	**.670****	**.601****	1
	Sig. (2-tailed)	.122	.000	.000	.000	.000	.
	N	68	68	68	68	68	68

**. Correleation is significant at the 0.01 level (2-tailed).

Table 6.9 - Pearson product-moment correlation coefficient showing relationships among computational thinking skills and also between these skills and the maximum game level students reached in the Cyprus study.

Based on the above statistical knowledge, it was observed that all correlations are in the positive direction where some of these are significant; others are not. Table 6.9 illustrates that there is a positive relation between the maximum level students achieved and a) conditional logic (r=0.311, n=68, p = 0.01), b) algorithmic thinking (r = 0.337, n = 68, p = 0.005) and c) simulating solutions (r = 0.362, n = 68, p = 0.002) respectively. Despite being significant the correlation coefficient identified in all three cases was not strong (Pearson's r <= +0.39). This means that, a number of players who reached high levels in the game agreed that the game enhanced their conditional logic, algorithmic thinking and the ability to simulate how

computer algorithms work. However, as the positive correlation was weak, this also put forward evidence despite not achieving the high levels in the game students felt that their conditional logic, algorithmic thinking and simulating solution abilities were enhanced. This outcome is linked to the pace of learners, a crucial concern regarding game based learning that is highlighted in previous work (Prensky, 2006). Kazimoglu *et al.* (2011) argued that a game approach should not force students to test their skills against other students. This was on the basis that it determines expertise or capability, and a game should encourage them to develop skills at their own pace. Pearson's correlation results demonstrate that students who agreed their abilities in conditional logic, algorithmic thinking and simulation were enhanced are not essentially the same students who did well in the game. In other words, slow paced students also felt that they used conditional logic, algorithmic thinking and observed simulation of computer algorithms during their game-play. This provides strong evidence that *Program Your Robot* successfully supported students in gaining the required underpinning skills at their own pace, while letting those who already have the skills skip the preliminary stages and move to a more advanced level.

In addition to these, no strong or significant correlation was identified between the highest level students achieved and debugging or cooperation during the study. The correlation in between the two pairs is positive ($r= 0.2$ for debugging and $r = 0.18$ for cooperation) but there is no evidence to prove that the relationship is either strong or significant. It was expected to observe a negligible or no relationship in the social aspect of computational thinking as this was merely investigated through sharing strategies/ideas and only supported by the high score system in the game. However, the findings indicate that there was also no relationship between debugging and how far students progressed in the game providing strong evidence that the success of doing well in the game was not related to how much they used the debugging feature.

As demonstrated with bold text in Table 6.9, the correlations among the five computational thinking skills were investigated in order to identify whether or not these affect each other in the game-play. The results show that there was a strong positive correlation between a) conditional logic and algorithmic thinking ($r = 0.867$, $n = 68$, $p = 0.000$); b) conditional logic and simulating solutions ($r= 0.791$, $n= 68$, $p= 0.000$) and c) algorithmic thinking and simulating solutions ($r=0.817$, $n=68$, $p= 0.000$). The Pearson's coefficient was positive, strong ($r>=0.7$) and significant ($p<0.01$) in all three cases. This means that an increase in conditional logic was correlated with an increase in algorithmic thinking and also with the ability to simulate how computer algorithms work in the game-play. As a result of this, an increase in

algorithmic thinking also caused an increase in simulating solutions. The Pearson's coefficient values provide strong evidence that when players used conditional logic in the game (such as when they try to find the most coherent pathway for their robot) they also developed abilities in algorithmic thinking and simulating solutions.

The findings also indicate that there is a positive, modest strong and significant correlation between a) algorithmic thinking and cooperation (r= 0.631, n=68, p=0.000); b) simulating solutions and cooperation (r= 0.67, n=68, p=0.000) and c) debugging and cooperation (r=0.601, n=68, p=0.000). The Pearson's coefficient (r = 0.6) provided evidence that the associations between these pairs were mediocre in all cases (r^2=0.36, 36% correlated). Therefore, it provides evidence that the more players cooperated and shared strategies, the more they thought algorithmically, used debug and simulated their solutions. However, as the association is not very strong between the pairs (r < = 0.7), this means that those players who did not cooperate in the game also developed abilities in algorithmic thinking, debugging and simulation.

As a result, the Pearson's r calculated from the associations of computational thinking skills showed that there are strong positive and significant correlations among algorithmic thinking, conditional logic and simulating solutions. To observe to what extent these are associated with each other a series of scatterplots were created. Figure 6.11 illustrates these scatterplots where strong, positive and significant correlations are identified.

As can be observed from the figure, rates are spread around the least squares regression line in a linear association where the majority of rates are noticeably high values (4 or 5). The first scatterplot shows that an increase in algorithmic thinking is 75% associated with an increase in conditional logic (r^2=.75). The second scatterplot demonstrate that higher rate of simulating solutions tends to result in substantially higher algorithmic thinking (r^2=.67, 67% correlated) and finally the third scatterplot illustrates that simulating solutions is extensively associated with conditional logic (r^2=.63, 63% correlated).

These scatterplots provide additional evidence that three out of the five computational thinking skills were successfully integrated and strongly related to each other according to the data obtained from the participants. The scatterplots for debugging and socialisation are not presented as the associations of these with other computational thinking skills were not significant.

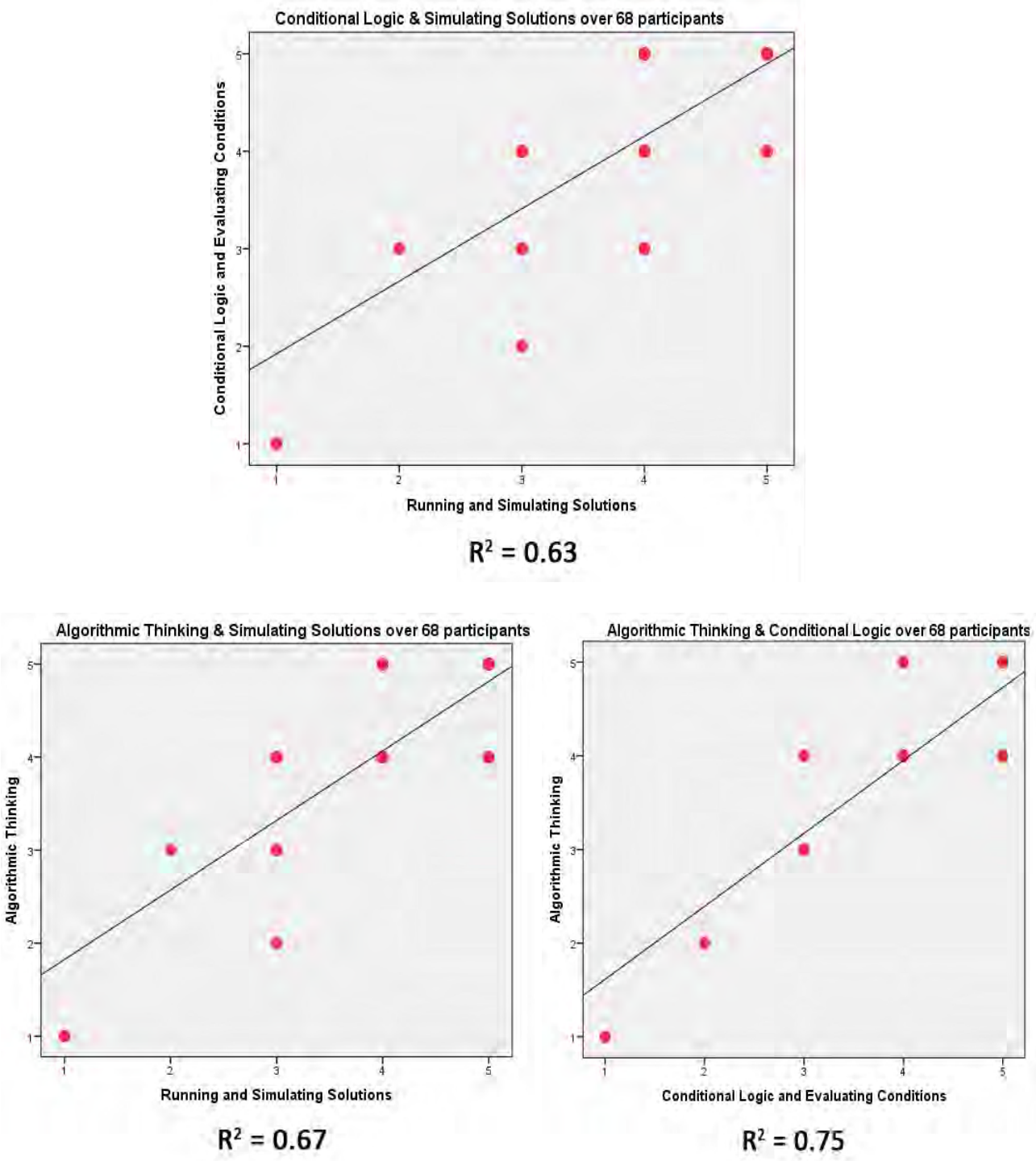

Figure 6.11 – Scatterplots showing strong correlations among algorithmic thinking, conditional logic and simulating solutions.

Pearson product-moment correlation coefficient		Post Programming Sequence Knowledge	Post Functions Knowledge	Post Decision Making Knowledge	Post Loops Knowledge
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.322**	335**	.512**	.477**
	Sig. (2-tailed)	.007	.004	.000	.000
	N	68	68	68	68
Algorithmic Thinking	Correlation Coefficient	.364**	.544*	**.618****	.402**
	Sig. (2-tailed)	.002	.045	.000	.001
	N	68	68	68	68
Running and Simulating Solutions	Correlation Coefficient	.414**	.301*	**.643****	.495**
	Sig. (2-tailed)	.000	.013	.000	.000
	N	68	68	68	68
Debugging and Handling Errors	Correlation Coefficient	.254*	**.628****	.337**	.432**
	Sig. (2-tailed)	.037	.000	.005	.000
	N	68	68	68	68
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.286	.178	.360**	.424**
	Sig. (2-tailed)	.180	.146	.003	.000
	N	68	68	68	68

** Correleation is significant at the 0.01 level (2-tailed).
* Correleation is significant at the 0.05 level (2-tailed).

Table 6.10 – Pearson product-moment correlation coefficient showing relationships between computational thinking skills and students' perception of their programming knowledge.

A Pearson's correlation was conducted between computational thinking skills and programming constructs introduced in the game in order to identify to what extent these are related to each other according to participants' feedback. The findings show that all programming constructs introduced in the game (i.e. sequence, functions, decision making and loops) were associated with all computational thinking skills except socialisation (i.e. conditional logic, algorithmic thinking, debugging and simulation) at a significant level. However it was identified that none of these associations were strong enough to assume that there is a direct correlation between learning programming constructs and developing computational thinking skills.

Although it has been discussed in Chapter 4 Section 4.4, it is important to highlight hereafter again that only the *cooperation* aspect of *socialising* is investigated in all three studies. As discussed in Chapter 3 Section 3.3, *Program your Robot* was not explicitly designed to support socialising during the game-play and only clearly supports the development of four out of five computational thinking skills (i.e. conditional logic,

algorithmic thinking, simulation, debugging). Hence to investigate the social aspect of computational thinking, participants were asked whether or not they shared ideas and strategies during their game-play as well as whether or not they found this useful (see Chapter 4 Section 4.4.4). As *sharing ideas and strategies* only cover the *cooperation* aspect of socialising, the terms *cooperation* and *socialising* interchangeably used during the analysis of the study correlations.

As illustrated in Table 6.10, the Pearson's correlation measure shows that there was a positive, modest strong and significant association between algorithmic thinking and decision making (r= 0.618, n=68, p=0.000), between simulating solutions and decision making (r= 0.643, n=68, p=0.000) and finally between debugging solutions and functions (r= 0.628, n=68, p=0.000). These findings indicate that participants who had higher knowledge in decision making tended to think algorithmically more frequently (r^2= 0.38, 38% correlated) and similarly simulated their solutions more often than usual (r^2= 0.41, 41% correlated). Additionally, those participants with higher knowledge in functions used the debug feature considerably (r^2= 0.39, 39% correlated). However, as the correlations among these pairs are modest strong (r <= 0.7 in all cases), it is not possible to assume that computational thinking skills and programming knowledge are directly related to each other, meaning that a substantial number of participants who did not have good knowledge in decision making or in functions also used algorithmic thinking, simulated and debugged their solutions at a high rate.

Additionally, the above findings provide statistical evidence to support the statement that computational thinking is not programming. In her seminal work, Wing (2006, 2008) identified that computational thinking was raised from the field of Computer Science. However, she underpinned that computational thinking is not a synonym for programming, but is a set of concepts that can be related to programming constructs and can thereby help in learning computer programming. Although Wing (2010) provided a detailed overview to support this statement, to date there is little or no statistical evidence to support her arguments in a game based learning environment. The above Pearson's correlation results delivered statistical evidence that there are no strong correlations between the five categories of computational thinking skills (conditional logic, algorithmic thinking, simulation, debugging and cooperation) and the four programming constructs (i.e. programming sequence, functions, decision making and loops) introduced in *Program Your Robot*.

There are two important indications that must be underlined when interpreting the findings of the Pearson's coefficient: On one hand, there was no strong association between learning

programming and computational thinking skills meaning that none of these skills are unique to computer programming and that even without good knowledge or background in computer programming, participants felt that their skills in computational thinking were developed. On the other hand, the association between computational thinking skills and the four programming constructs were always significant and positive (except socialisation) and in three cases it was modestly strong (algorithmic thinking – decision making, simulating solutions – decision making, debugging solutions – functions). Therefore, there is strong evidence to suggest that students with strong computational thinking abilities can perform better in computer programming than others.

As illustrated in Table 6.11, the final Pearson correlation was computed to assess the relationship among visualising constructs, programming knowledge and problem solving abilities between the pre and the post study. It was aimed at defining the degree of associations among these based on participant responses in order to understand whether or not there is a significant relationship between them. Initially, the difference between the responses given in pre and post study is calculated for visualising constructs, programming knowledge and problem solving abilities. These categories were then paired with one another and a Pearson product-moment correlation computed to assess whether or not a change that happened in one category affected the other(s). The results indicate that there was a very strong, positive correlation between participants' ability to visualise programming constructs and their perception of programming knowledge, $r = 0.89$, $n = 68$, $p = 0.000$. Additionally, a strong positive correlation between participants' ability to visualise programming constructs and their problem solving abilities was identified, $r = 0.80$, $n = 68$, $p = 0.000$. Finally, the Pearson's r also defined that there was a strong positive correlation between participants' problem solving abilities and participants' perception of their programming knowledge, $r = 0.73$, $n = 68$, $p=0.000$.

These findings provide evidence that the more participants visualised programming constructs from given problems, the more they felt their programming knowledge enhanced ($r^2= 0.8$, 80% correlated). Correspondingly, the more participants visualised programming constructs from given problems, the more they felt their problem solving abilities developed ($r^2= 0.64$, 64% correlated). As a consequence of this, those participants who felt that they developed problem solving abilities also felt that their programming knowledge was enhanced ($r^2= 0.53$, 53% correlated). Overall, the associations among the differences in programming knowledge, visualisation of constructs and problem solving abilities were positive, strong and more importantly significant. This provides strong evidence that as participants visualised

programming constructs from given problems during their game-play, they felt that their problem solving abilities were developed. As a result of this, their perception of programming knowledge was enhanced significantly. In other words, according to the data analysis an increase in visualising programming constructs from given problems resulted in the development of problem solving abilities as well as a significant increase in programming knowledge.

Pearson product-moment correlation coefficient		Difference in the ability to visualise constructs	Difference in programming knowledge	Difference in problem solving abilities
Difference in the ability to visualise constructs	Pearson Correlation	1	**.896****	**.803****
	Sig. (2-tailed)		.000	.000
	N	68	68	68
Difference in programming knowledge	Pearson Correlation	**.896****	1	**.736****
	Sig. (2-tailed)	.000		.000
	N	68	68	68
Difference in problem solving abilities	Pearson Correlation	**.803****	**.736****	1
	Sig. (2-tailed)	.000	.000	
	N	68	68	68

** Correlation is significant at the 0.01 level (2-tailed).

Table 6.11 – Pearson product-moment correlation coefficient showing associations among visualising constructs, programming knowledge and problem solving abilities.

6.2.8 Summary of findings regarding correlations

The results of the Cyprus study are further investigated in Chapter 7 Section 7.1 in order to ascertain whether or not the statistical findings are internally and externally valid. A summary of results obtained from Pearson's correlation coefficient assessment in the Cyprus study is listed below:

There was a strong, positive and significant correlation between:

a) conditional logic and algorithmic thinking, r = 0.86, n = 68, p = 0.000;
b) conditional logic and simulating solutions, r= 0.79, n= 68, p= 0.000;
c) algorithmic thinking and simulating solutions, r=0.81, n=68, p= 0.000;

d) ability to visualise programming constructs and programming knowledge gained, $r = 0.89$, $n = 68$, $p = 0.000$;

e) ability to visualise programming constructs and problem solving, $r = 0.80$, $n = 68$, $p = 0.000$;

f) problem solving abilities and programming knowledge gained, $r = 0.73$, $n = 68$, $p=0.000$.

There was a strong, positive and significant correlation between:

a) algorithmic thinking and cooperation, $r= 0.63$, $n=68$, $p=0.000$;

b) simulating solutions and cooperation, $r= 0.67$, $n=68$, $p=0.000$;

c) debugging and cooperation, $r=0.6$, $n=68$, $p=0.000$;

d) algorithmic thinking and knowledge in decision making, $r= 0.61$, $n=68$, $p=0.000$;

e) simulating solutions and knowledge in decision making, $r= 0.64$, $n=68$, $p=0.000$;

f) debugging solutions and knowledge in functions ($r= 0.62$, $n=68$, $p=0.000$).

6.3 The Greenwich study evaluation and statistical analysis

This section analyses the distribution of data collected in the Greenwich study using the same structure as the Cyprus study was analysed. Rather than a parametric measure (i.e. paired samples t-test), a non-parametric measure (i.e. the Wilcoxon signed ranks test) was used to analyse the results due to the non-normal distribution of data.

The Greenwich study was conducted with 189 participants where 44 (23%) of these either dropped out or completed the pre-study but did not complete the post-study during the study. Overall 145 out of 189 (77%) valid responses were successfully collected and interpreted by matching pre-study responses to post-study responses. It was observed that the dropout rates of the Greenwich study (23%) were considerably higher than the Cyprus study (9.4%). Although the exact reason(s) for this is not measurable, this may be due to changes in students' background characteristics, educational performance and their attitude to the body of research.

Among the valid responses obtained, 125 out of 145 (86.2%) were from male students and 20 out of 145 (13.8%) were from female students. Whilst 126 (86.8%) out of 145 students were in between 18 – 24, 11 (7.5%) more were in the 25 – 29 age range. Moreover, 6 (4.1%) students were in the 30 – 39 age range and 2 (1.3%) students were above 40. Although ethnic

classification of students was not used when analysing the data, this was obtained from students for future analysis of the data set. In terms of ethnicity, 66 (45.5%) out of 145 participants were white, 33 (22.7%) were Asian or Asian British and 25 (17.2%) more were Black or Black British. While 7 (4.8%) out of 145 had a mixed background, 3 (2%) participants labelled themselves as Chinese. Finally, 11 (7.8%) participants indicated that they have other ethnic background that did not fit into the UK government standard ethnic classification.

Similar to the Cyprus study, the Greenwich study was conducted five weeks after students started their computer programming course. However, students who participated in the Greenwich study were from various programmes including, but not limited to, Computer Science, Software Engineering, Business Computing, Information Systems and Computer Systems & Network whereas in the Cyprus studies all students were on an Information Systems degree programme.

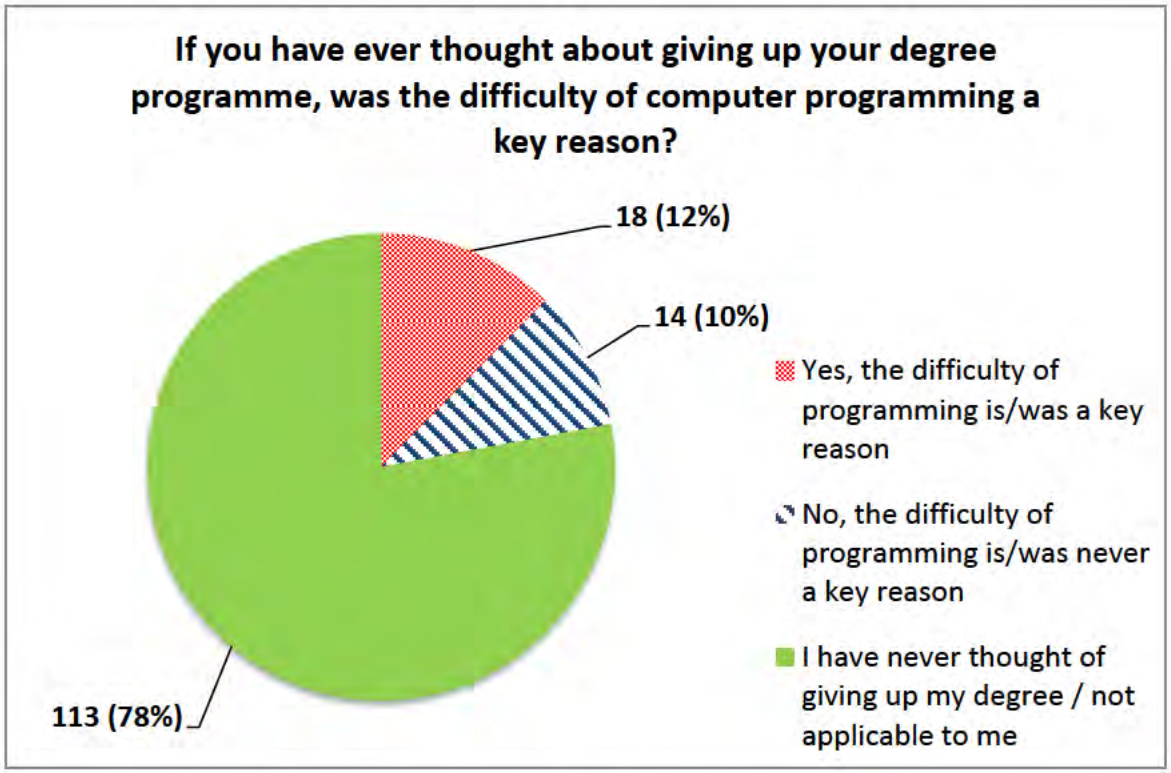

Figure 6.12 – The number of students in the Greenwich study who agrees that the difficulty of programming was a key reason to give up their degree programme.

The first question students answered in the pre-study of the Greenwich study was whether or not they thought about giving up their degree programme due to the difficulty of computer

programming. As shown from the above figure, 32 out of 145 (22%) students considered giving up their degree programmes five weeks after they had started to their degree programme. The results show that out of 32 students, 18 (56.2%) labelled the difficulty of computer programming as a key reason to give up their degree programme. This outcome is consistent with the previous findings obtained in the Cyprus study as in the Cyprus study 32.3% of students considered giving up their degree programmes and 59% of this believed that the difficulty of computer programming was a key reason for this. Overall, the results of both the Cyprus and the Greenwich studies indicated that an important number of students (between 22% - 32%) considered giving up their degree programmes shortly after they started their degree programme.

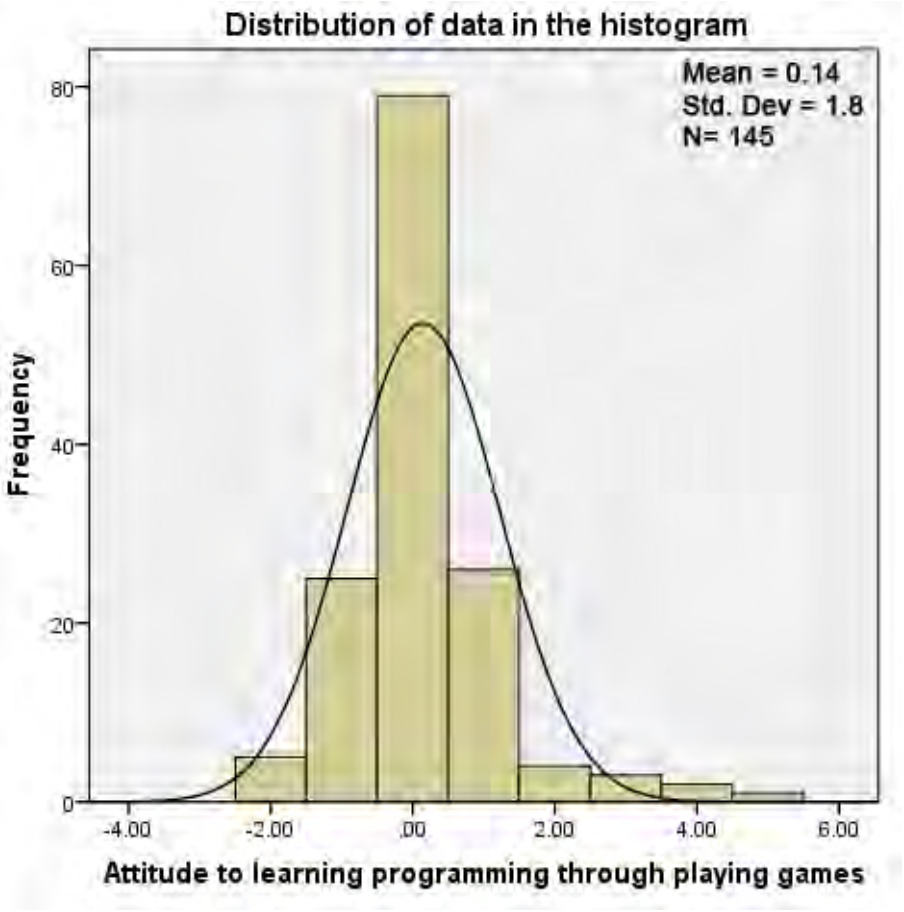

Figure 6.13 – Histogram showing distribution of data captured on the difference between attitudes to learn computer programming through playing games in the Greenwich study (Research question 1).

In order to define the appropriate statistical method(s) suitable for investigating the raw data collected from the Greenwich study a procedure for defining whether or not the data came from a normally distribution population is applied. This procedure involved generating a

histogram with normal *quantile – quantile (Q-Q) plots* in addition to a *Skewness and Kurtosis normality check* and a *Shapiro Wilk test*.

Figure 6.13 shows the distribution of data gathered for the first research question (i.e. difference in students' attitude to learn computer programming through playing games between the pre and the post study) in the Greenwich study. As shown from the figure, the histogram has Skewness issues as it is skewed to the right that causes an asymmetry in the distribution of data. This means that there is more data in the right tail in the dataset than would be expected in a normal distribution. More importantly, the histogram has major Kurtosis issues as the peak Kurtosis point is way over than what would be expected in a normal curve. Although the population mean value is very close to 0 ($\mu = 0.14$), the population standard deviation value is above 1 ($\sigma = 1.8$).

In their seminal work, Hyvärinen & Oja (2000) states that random variables with a negative Kurtosis are called *subgaussian*, and those variables with positive Kurtosis are referred as *supergaussian*. A *supergaussian* distribution is referred to as *leptokurtic distribution* (narrow-arched) when the distribution has higher peaks and fatter tails compared to a normal distribution (*mesokurtic distribution*) (Investopedia, 2013a).

Figure 6.13 demonstrates precisely a *leptokurtic distribution* and shows that the majority of observations are concentrated around the mean value whereas the rest of the observations have very low variations. This means that there are clusters in the distribution of data which proves the data came from a non-normally distributed population.

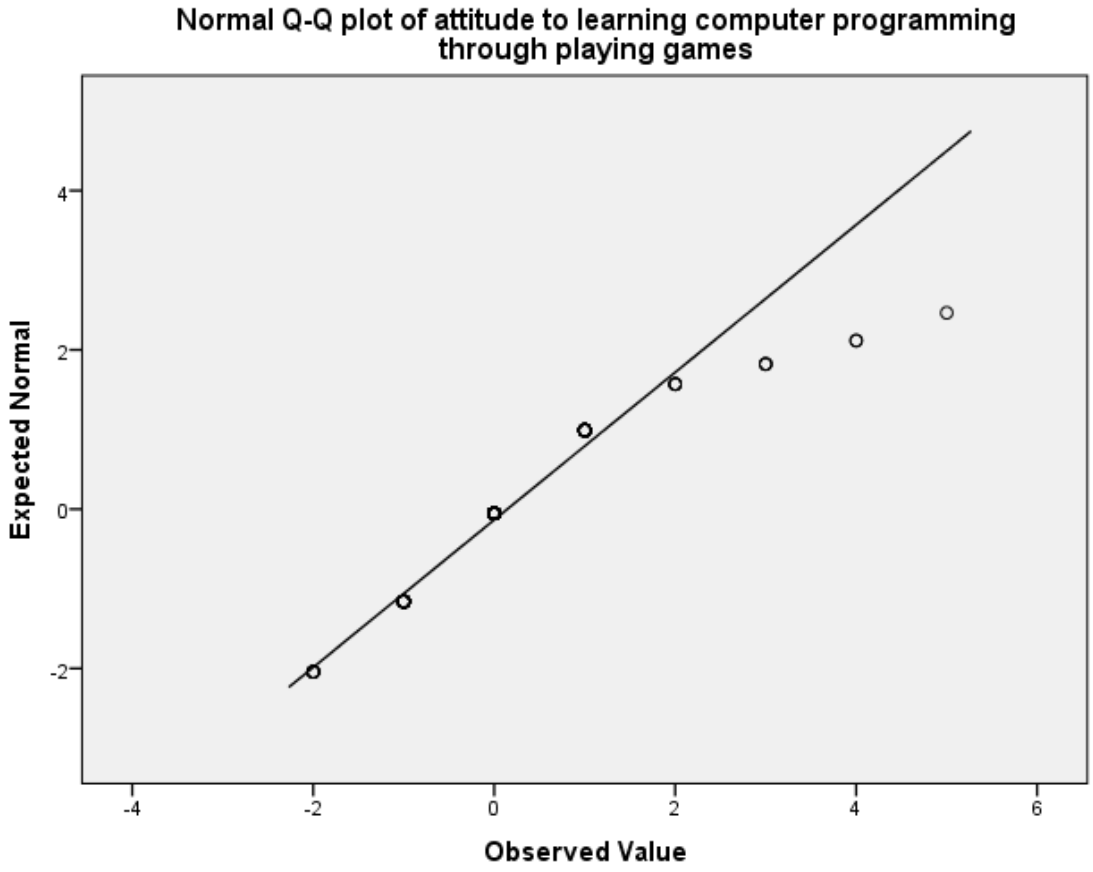

Figure 6.14 – Normal quantile – quantile (Q-Q) plots showing distribution of observations captured on the difference between attitudes to learn computer programming through playing games in the Greenwich study (Research question 1).

In order to investigate the histogram further and define where exactly the observations cluster, a quantile – quantile (Q-Q) plot was undertaken. As shown from Figure 6.14, the Q-Q plot generated from the data obtained for the first research question (i.e. difference in students' attitude to learn computer programming through playing games between the pre and the post study) shows a narrow arched shape as the heavy tailed population have higher peak than the benchmark normal population. Additionally, the straight line on the figure shows a perfect normal distribution.

The plot shows that observations start coherent with their normal counterparts initially but then these soon depart from the normal curve. In other words, the observations show a normal distribution at first, specifically in the lower values, as all of them hug the linear line. However, the observed values move from the median into the right hand tail and concentrate around the mean in order to balance the extreme members in the population. As a result, the

outcome resembles a bow shape normal Q-Q plot that starts coherent with the normal curve, arches across the target line and finishes below the line. Because not all of the observations hug the linear line, the Q-Q plot provides strong evidence that the data came from a non-normally distributed population.

Descriptive Statistics

Skewness & Kurtosis Issues	N	Mean	Std. Deviation	Skewness		Kurtosis	
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Std. Error
Attitude to learning computer programming through playing games	145	**.1448**	**1.7999**	**1.418**	.201	**4.424**	.400
Valid N (listwise)	145						

Table 6.12 – Skewness and Kurtosis normality check on the difference between attitudes to learn computer programming through playing games in the Greenwich study (Research question 1).

Table 6.12 demonstrates the Skewness and Kurtosis issues in the data obtained from the attitude of participants to learning computer programming through playing games between the pre and the post study in the Greenwich study. As shown from the table, both Skewness (1.4) and Kurtosis (4.4) values are very far from being 0 and the absolute value of Skewness (0.201 * 3 = 0.6 < 1.4) and Kurtosis (0.4 * 3 = 1.2 < 4.4) does not satisfy three times of their standard error rule. Hence, the data obtained from the difference in attitude to learning computer programming through playing games has major Skewness and Kurtosis issues as the distribution of the data is too peak (4.4) as well as asymmetric (1.4). As a result, the Skewness and Kurtosis normality check support the results of histogram and normal Q-Q plots and thus puts forward more evidence that the data obtained did not come from a normally distributed population.

A *Shapiro-Wilk* and a *Kolmogorov-Smirnov* test was performed as a final assessment to measure the distribution of data obtained from the difference in attitude to learning computer programming through playing games, between the pre and the post study (first research question). A null hypothesis (i.e. H0 – the sample population is normally distributed) and an alternative to this (i.e. H1 – the sample population is not normally distributed) was created in order to interpret the test results accurately. As shown from Table 6.13, the significant value (p) generated from the *Shapiro-Wilk* was less than 0.05 (p=0.000), therefore the null hypothesis which indicates the data came from a normally distributed population is rejected.

Hence, the *Shapiro-Wilk* normality test provides evidence that the data obtained for the first research question in the Greenwich study did not come from a normally distributed population.

Tests of Normality

Normality Check	Kolmogorov-Smirnov		Shapiro-Wilk	
	Statistic	Sig.	Statistic	Sig.
Attitude to learning programming through playing games	.305	.000	.812	.000

Table 6.13 – The Shapiro Wilk and the Kolmogorov Smirnov test results on the difference between attitudes to learn computer programming through playing games in the Greenwich study (Research question 1).

The normality check methods described above (*Histogram*, *Quantile – Quantile Plots*, *Skewness and Kurtosis normality check* and the *Shapiro-Wilk* test) were undertaken for all nine research questions in the Greenwich study and the results show that the distribution of data was non-normal in each of the datasets collected for these research questions. In other words, each research question is analysed in the same way as the first research question (*Histogram*, *Quantile – Quantile Plots*, *Skewness and Kurtosis normality check* and the *Shapiro-Wilk* test) and it was found that the data sets came from a non-normally distributed population.

As the data sets obtained for the research questions did not fit a normal distribution, it was simply not possible to perform two tailed samples-paired t-tests within one group to analyse the data captured in the Greenwich study. One strategy that could be applied was to make the non-normal data distribution resembles a normal distribution by using a statistical transformation so that samples-paired t-tests would be available to analyse the data statistically. However, there are many different transformations in the statistical literature and it is not always obvious which of these fits best to the situation at hand (Osborne, 2010). Further to this, all statistical transformations have their own limitations and choosing to apply one of these may result losing some of the observed values in the data sets (Sherman, 2010).

As an example, a Box-Cox Transformation could have been applied to make the distribution of data sets resemble a normal distribution but would also mean risking two important drawbacks: a) In their seminal work, Box and Cox (1964) clearly described that this

transformation is designed for non-negative responses and it cannot transform negative observations**.** Therefore applying this transformation to the Greenwich dataset would mean risking to lose all negative values collected from participants; b) Box-Cox does not change the distribution of data considerably when the distribution of data has a very heavy tail (Hossain, 2011). As mentioned previously, the data sets collected for research questions have strong Skewness and Kurtosis issues that result in heavy tails and peakedness in the distribution. Therefore, it was not estimated whether or not a Box-Cox transformation could change the distribution to such extent that it would overcome all heavy tails and peakedness issues. Because of these reasons, a statistical transformation was avoided and the distribution of data was left as non-normally distributed.

As a result, samples-paired t-tests were not available to analyse the Greenwich study datasets and therefore, the non-parametric equivalent of this which was the Wilcoxon signed rank test was undertaken. As indicated by Laerd Statistics (2012a), the Wilcoxon signed-rank test is the non-parametric equivalent to the dependent t-tests as the Wilcoxon signed-ranks test does not assume normality in data whereas parametric dependent-test does. In other words, the Wilcoxon signed rank test is the alternative of samples-paired t-tests when the normal distribution assumption is violated and the use of paired t-test is simply not appropriate. Similar to its parametric equivalent, all Wilcoxon signed rank tests in this study were based on different pairs of sample data and interpreted through research questions which are accompanied by the hypothesis as described in Chapter 5 Section 5.3. The results are analysed by comparing the responses given in the Greenwich study with the responses given in the Cyprus study because these are in fact the same study conducted in different locations at different times on different people.

6.3.1 Research Question 1 – Is there a difference in students' attitude to learn computer programming through playing games between the pre and the post study?

Figure 6.15 demonstrates the responses obtained in the Greenwich study regarding students' attitude to learning computer programming through playing games (first research question) both in the pre and the post study in the Greenwich study. Initially 31 (21.3%) out 145 students strongly agreed and 71 (48.9%) more students agreed that a game specifically designed for programming purposes can be useful for learning how computer programming constructs work. While 33 (22.7%) out of 145 students remained neutral, 3 (2%) students

disagreed and 7 (4.8%) students did not provide their viewpoints. Having played the game, the number of students who strongly agreed increased from 31 (21.3%) to 47 (32.4%) whereas the number of students who agreed decreased from 71 (48.9%) to 68 (46.8). To investigate the reasons why this decline happened, the responses of those students who changed their opinion were investigated. It was found that 3 (2%) students who originally agreed changed their opinions to strongly agree after their game-play. Further to this, the number of students who were neutral decreased from 33 (22.7%) to 23 (15.8%) and correspondingly the number of disagreed students decreased from 3 (2%) to 1 (0.6%) between pre and post study. While 5 (3.4%) students did not provide their opinions, only 1 (0.6) student strongly disagreed that a game can be used for learning how computer programming constructs work in the post-study.

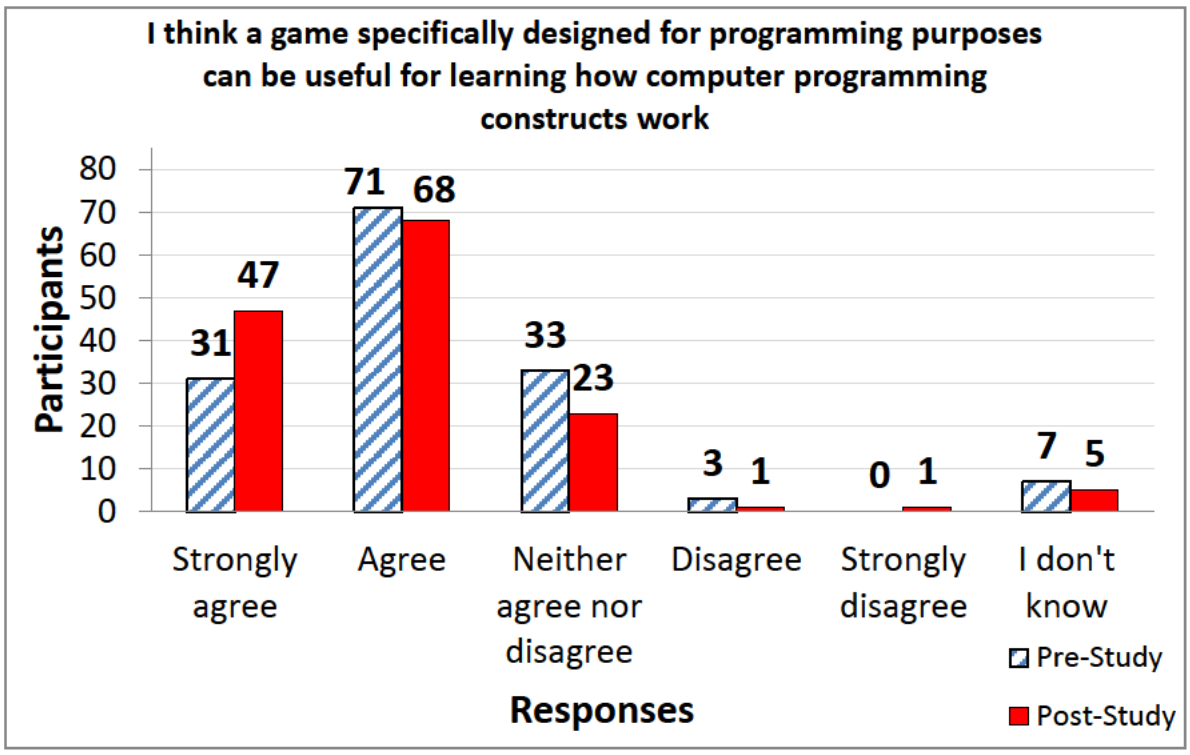

Figure 6.15 – Students' attitude to learning computer programming through playing games between the pre and the post study in the Greenwich study.

The raw data gathered for the first research question in the Greenwich study was consistent with the results obtained in the Cyprus study as in both studies a considerable percentage of students (76.4% in the Cyprus study; 70.3% in the Greenwich study) were motivated to learn how computer programming constructs work through game-play before they participated in the studies. Previously in the Cyprus study, it was found that the number of students who

strongly agreed, and agreed, that a game can be useful for learning how computer programming constructs work, increased from 52 (76.4%) to 60 (88.2%). The Greenwich study results are also consistent with this as the number of strongly agreed and agreed students increased from 102 (70.3%) to 115 (79.3%).

The results of both studies suggest that over 70% of the students were already motivated to learn computer programming through game-play before they participated in the studies. Having played the game, the number of participants who strongly agreed and agreed that a game can be used to learn computer programming constructs was slightly increased (11% increase in the Cyprus; 9% increase in the Greenwich).

A Wilcoxon signed rank test was carried out to measure whether or not there is a difference in students' attitude to learning programming through playing games in the Greenwich study. The results indicate that there was a slight increase in students' attitude to learn programming through playing games between pre-study (M=3.63; SD=1.0) and post-study (M=3.99; SD=1.0) conditions; average rank of 43.40 vs. average rank of 43.54. The Wilcoxon signed ranks test shows that the difference in students' attitude to learning computer programming through playing games between the pre and the post study is significant ($z=3.309$, $p<0.05$). Thus, the null hypothesis that specifies the groups does not differ in attitudes to learn computer programming through playing games is rejected. Therefore, Wilcoxon signed ranks test results support the alternative hypothesis that is to say students' attitude regarding learning programming through playing games significantly increased during the study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre video game attitude to learning programming through game-play	145	**3.63**	1.013	0	5
Post video game attitude to learning programming through game-play	145	**3.99**	1.064	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post video game attitude to learning programming through game-play – Pre video game attitude to learning programming through game-play	Negative Ranks	**26[a]**	**43.40**	1128.50
	Positive Ranks	**60[b]**	**43.54**	2612.50
	Ties	**59[c]**		
	Total	145		

a. Post video game attitude < Pre Video game attitude

b. Post video game attitude > Pre video game attitude

c. Post video game attitude = Pre video game attitude

Wilcoxon Signed Ranks Test	Post video game attitude to learning programming through game-play – Pre video game attitude to learning programming through game-play
Z	**-3.309[d]**
Asymp. Sig. (2-tailed)	**.001**

d. Based on negative ranks.

Table 6.14 – Descriptive statistics and Wilcoxon Signed Ranks Test results of students' attitude to learning computer programming through playing games between the pre and the post study in the Greenwich study.

6.3.2 Research Question 2 – Is there a difference in students' intrinsic motivation to learn computer programming between the pre and the post study?

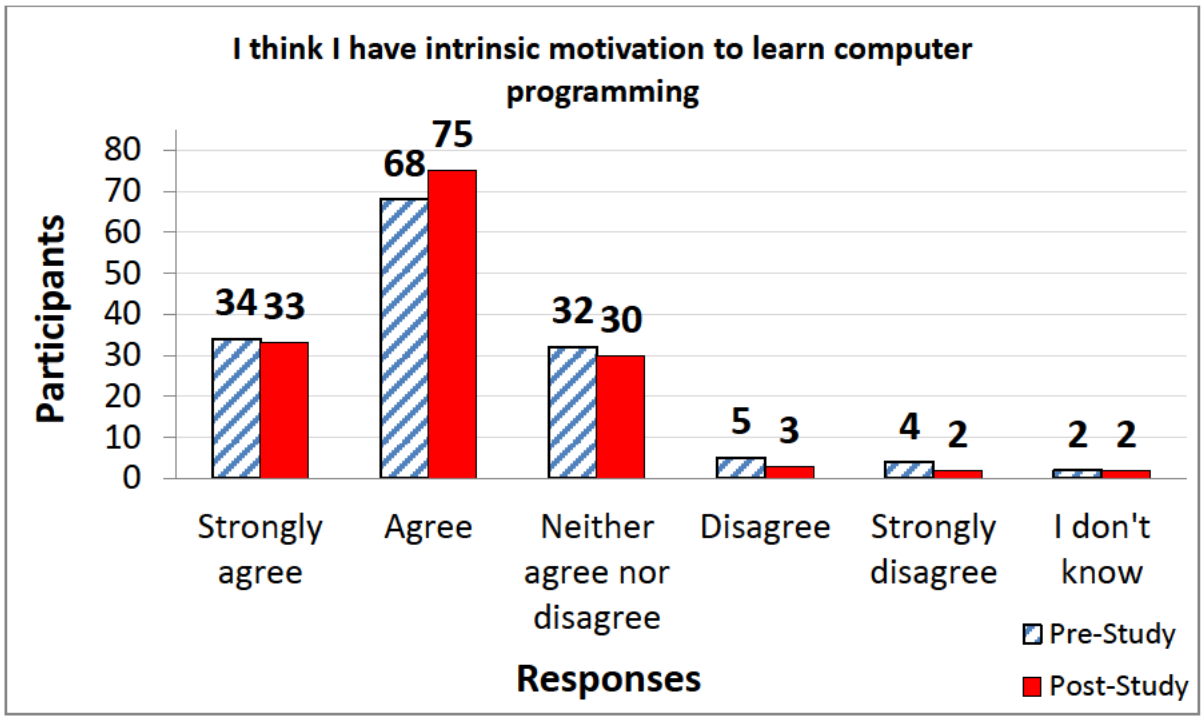

Figure 6.16 – Students' perception about their intrinsic motivation to learn computer programming between the pre and the post study in the Greenwich study.

Figure 6.16 illustrates students' perception about their intrinsic motivation (motivation that is driven by interest or enjoyment) to learn computer programming between pre and post study in the Greenwich study. Prior to the study, 34 (23.4%) out of 145 students strongly agreed and 68 (46.8%) more agreed that they have intrinsic motivation to learn computer programming. In other words, a total of 102 (70.2%) students felt that they were intrinsically motivated to learn computer programming before the study was conducted. Additionally, while 32 (22%) students were neutral, 9 (6.2%) students strongly disagreed and disagreed that they have intrinsic motivation to learn computer programming in the pre-study. Having played the game, the number of strongly agreed students decreased from 34 (23.4%) to 33 (22.8%) as it was found that 1(0.6%) student who strongly agreed in the pre-study switched to agree in the post-study. Further to this, the number of agreed students increased from 68 (46.8%) to 75 (51.7%) and the number of neutral students decreased from 32 (22%) to 30 (20.6%). The total number of strongly disagreed and disagreed students decreased from 9

(6.2%) to 5 (3.4%) and 2 (1.3%) students who did not know the answer in the pre-study did not change their opinion after their game-play. The results of the second research question in the Greenwich study support the findings of the Cyprus study and show that there has been an increase in the students' perception of their intrinsic motivation to learn computer programming after their game-play. However, this increase (4.2% increase happened in strongly agreed and agreed students) is much less than the increase that happened in the Cyprus study (36.5% increase happened in strongly agreed and agreed students). Before investigating the reasons of this, a Wilcoxon signed ranks test was performed to assess whether or not the findings were significant.

The Wilcoxon signed ranks test revealed that there is an increase in students' perception of their intrinsic motivation for learning programming between pre (M=3.56, SD=0.9) and post (M=3.88, SD=0.9) study conditions in the Greenwich study; average rank of 40.85 vs. average rank of 46.63. As illustrated in Table 6.15, the Wilcoxon signed ranks test results show that the increase in students' perception of their intrinsic motivation for learning programming between the pre and the post study is significant (z=3.095, $p<0.05$). As the 2-tailed significant value is less than 0.05 (p=0.02), the null hypothesis that indicates the groups' perception of their intrinsic motivation to learn computer programming does not differ is rejected. In this case, the Wilcoxon signed ranks test results support the alternative hypothesis that is the students' perception of their intrinsic motivation to learn computer programming is significantly increased between the pre and the post study during the Greenwich study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre computer programming intrinsic motivation	145	**3.56**	.942	0	5
Post computer programming intrinsic motivation	145	**3.88**	.924	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post computer programming intrinsic motivation – Pre computer programming intrinsic motivation	Negative Ranks	**24[a]**	**46.63**	1119.00
	Positive Ranks	**60[b]**	**40.85**	2451.00
	Ties	**61[c]**		
	Total	145		

a. Post computer programming intrinsic motivation < Pre computer programming intrinsic motivation

b. Post computer programming intrinsic motivation > Pre computer programming intrinsic motivation

c. Post computer programming intrinsic motivation = Pre computer programming intrinsic motivation

Wilcoxon Signed Ranks Test	Post computer programming intrinsic motivation – Pre computer programming intrinsic motivation
Z	**-3.095[d]**
Asymp. Sig. (2-tailed)	**.002**

d. Based on negative ranks

Table 6.15 – Descriptive statistics and the Wilcoxon Signed Ranks Test results of students' perception about their intrinsic motivation to learn computer programming between the pre and the post study in the Greenwich study.

As the statistical results of the Wilcoxon signed rank test demonstrated, the increase that happened in the Greenwich study is significant, it was decided to investigate why this increase is considerably less than the increase that happened in the Cyprus study. Although it is not possible to define the precise reasons for this difference, a detailed analysis was performed on the other questions students responded to in both studies. The results revealed that participants in the Cyprus study found learning computer programming constructs through *Program Your Robot* much more enjoyable than participants in the Greenwich study and thus one may conjecture this impacted their perception of intrinsic motivation to learn computer programming. Figure 6.17 and Figure 6.18 illustrate the differences in the students'

enjoyment of learning computer programming between the pre and post study both in the Cyprus and the Greenwich studies respectively. In the pre-study students were asked to rate their current level of enjoyment for learning computer programming. In the post-study, students were asked the same question as "this form of learning computer programming" which referred to *Program Your Robot*. It is crucial to be clear that the intention behind this question is by no means to compare traditional learning of computer programming against *Program Your Robot*. The objective was merely to explore how much students enjoyed learning computer programming constructs before and after their game-play.

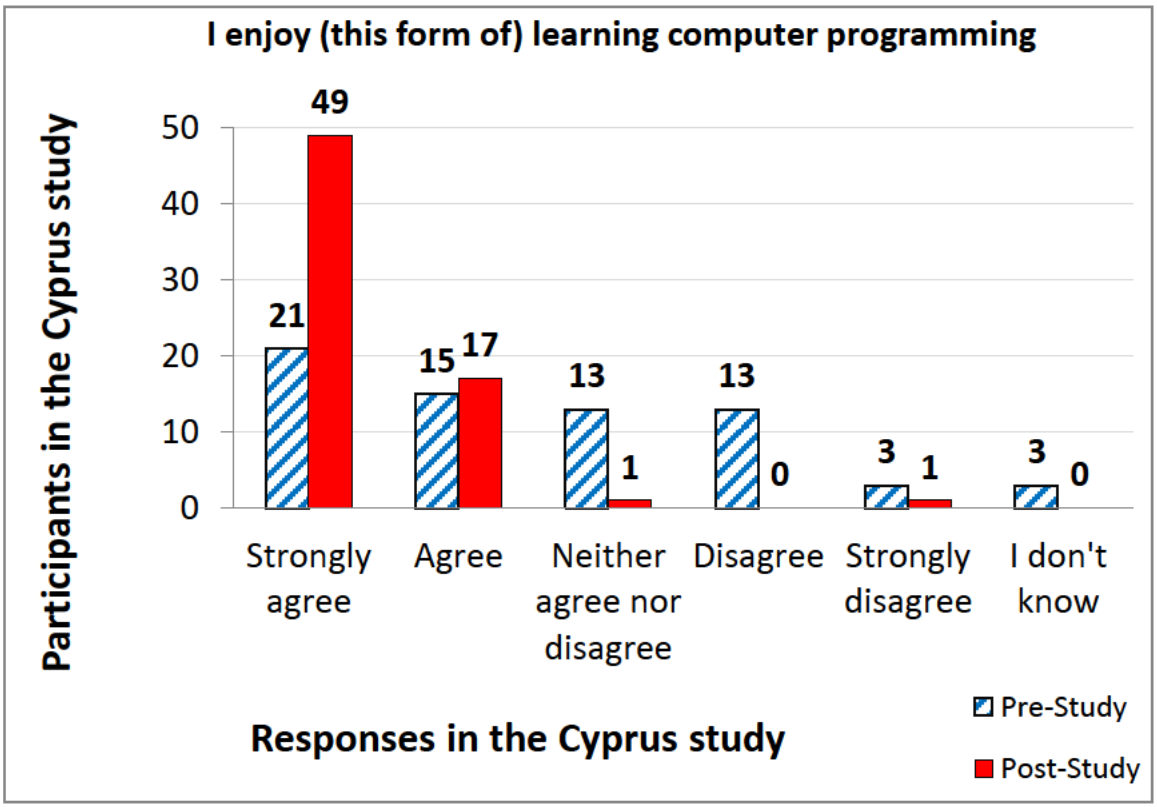

Figure 6.17 – Students' enjoyment of learning computer programming between the pre and the post study in the Cyprus study.

Prior to the Cyprus study, 21 (30.8%) out of 68 participants strongly agreed and 15 (22%) more agreed that they enjoy learning computer programming. Having played the game, the number of strongly agreed students increased from 21 (30.8%) to 49 (72%) and correspondingly the number of agreed students increased from 15 (22%) to 17 (25%). More interestingly, the number of students who disagreed that they enjoyed computer programming was decreased from 13 (19.1%) to 1 (1.4%). Similarly, the number of neutral students decreased from 13 (19.1%) to 1 (1.4%) as well. It is pleasing to note that only 1 (1.4%)

student strongly disagreed in the post-study.

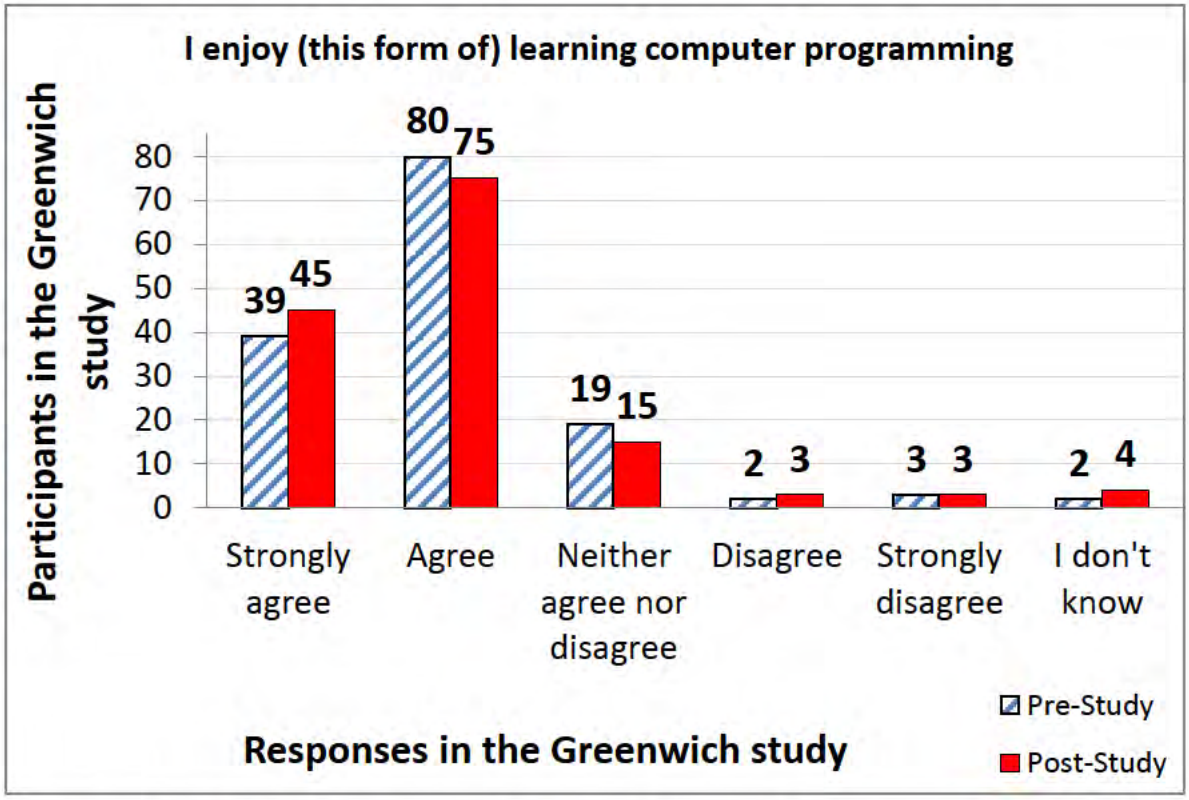

Figure 6.18 – Students' enjoyment of learning computer programming between the pre and the post study in the Greenwich study.

In the pre-study of the Greenwich study, 39 (26.8%) participants strongly agreed and 80 (55.1%) more agreed that they enjoy learning computer programming. After their game-play, the number of strongly agreed students is increased from 39 (26.8%) to 45 (31%) while the number of agreed students decreased from 80 (55.1%) to 75 (51.7%). Additionally, those students who were neutral decreased from 19 (13.1%) to 15 (10.3%) in the post-study. Whilst the number of strongly disagreed students remained at 3 (2%), the number of disagreed students increased from 2 (1.3%) to 3 (2%) during the study. Finally, those who did not want to the answer increased from 2 (1.3%) to 4 (2.75%) in the post-study.

Pearson's correlation in Cyprus Experiment		Intrinsic motivation to learn computer programming	Enjoyment in learning computer programming
Intrinsic motivation to learn computer programming	Pearson Correlation	1	**.893****
	Sig. (2-tailed)		.000
	N	68	68
Enjoyment in learning computer programming	Pearson Correlation	**.893****	1
	Sig. (2-tailed)	.000	
	N	68	68

Spearman's rank correlation in Greenwich Experiment		Intrinsic motivation to learn computer programming	Enjoyment in learning computer programming
Intrinsic motivation to learn computer programming	Correlation Coefficient	1.000	**.834****
	Sig. (2-tailed)	.	.000
	N	145	145
Enjoyment in learning computer programming	Correlation Coefficient	**.834****	1.000
	Sig. (2-tailed)	.000	.
	N	145	145

**. Correlation is significant at the 0.01 level (2-tailed).

Table 6.16 – Correlations between intrinsic motivation to learn computer programming and enjoyment in learning programming during the Cyprus and the Greenwich studies.

The results obtained from the students' enjoyment to learn computer programming clearly suggest that in both studies participants rated that they enjoyed learning computer programming more in the post-study than in the pre-study. The findings clearly indicate that participants in the Cyprus study are inspired more than participants in the Greenwich study as the number of strongly agreed and agreed students increased from 36 (52.9%) to 65 (95.5%) in the Cyprus study whereas in the Greenwich study this was only increased from 119 (82%) to 120 (82.7%). As revealed in Table 6.16, the enjoyment students got from learning computer programming is very strongly correlated to their perception of intrinsic motivation to learn computer programming both in the Cyprus ($r = 0.89$, $n = 68$, $p = 0.000$) and in the Greenwich ($r = 0.83$, $n = 68$, $p = 0.000$) studies. This means that as students enjoyed learning computer programming their intrinsic motivation increased consistently during the studies. More importantly, because students in the Cyprus study enjoyed learning computer programming constructs with *Program Your Robot* more than students in the Greenwich study; their intrinsic motivation to learning computer programming was also increased more. In other words, because students in the Cyprus study engaged with *Program Your Robot* more

than students in the Greenwich study, one may conjecture that this impacted on their perception of intrinsic motivation to learn computer programming.

It is crucial to highlight that both study results were statistically significant and correlated to how much students enjoyed from learning the game environment. Therefore, it is possible to conclude that the study intervention always increases intrinsic motivation to learn computer programming but the degree of this change may depend on how much students enjoyed playing with *Program Your Robot*.

6.3.3 Research Question 3,4,5,6 – Is there a difference in students' perception of their knowledge in programming sequence, methods, decision making and loops between the pre and the post study?

Figure 6.19 shows the responses of students based on the key introductory programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops) in the Greenwich study. The results are very consistent with the previous finding in the Cyprus study and provide evidence that students felt their knowledge regarding all programming constructs, introduced in the game, has been enhanced, particularly in programming sequence and in loops.

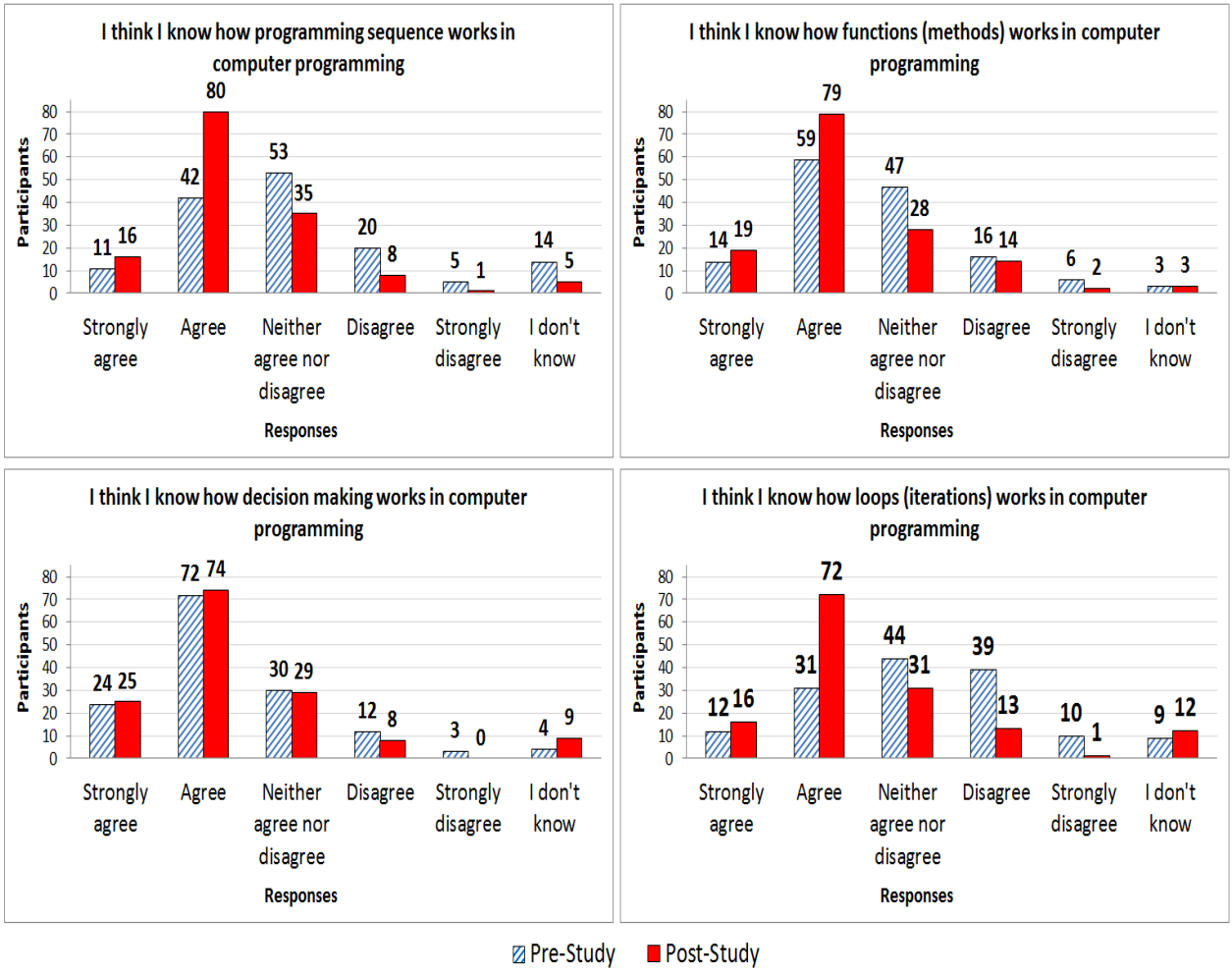

Figure 6.19 – Students' perception of their knowledge on programming constructs between the pre and post study of the Greenwich study.

In the pre-study, 53 (36.5%) out of 145 students strongly agreed and agreed that they know programming sequence. Having played the game the number of strongly agreed students increased from 11 (7.5%) to 16 (11%) and the number of agreed students increased from 42 (28.9%) to 80 (55.1%). Consequently, the number of neutral students decreased from 53 (36.5%) to 35 (24.1%). Prior to the game-play, 20 (13.7%) students disagreed and 5 (3.4%) more strongly disagreed that they know programming sequences. After the game-play, the number of disagreed students decreased from 20 (13.7%) to 8 (5.5%) and the number of strongly disagreed students decreased from 5 (3.4%) to 1 (0.6%). It is also pleasing to note that the number of students who didn't want to rate their programming sequence knowledge was also decreased from 14 (9.6%) to 5 (3.4%) during the study. Moreover, the same positive difference was observed in the functions, decision making and particularly in loops. Before

the study was conducted, a total of 73 (50.3%) students strongly agreed and agreed that they knew functions (methods) in computer programming. After playing the game, this number increased to 98 (67.5%) and the number of neutral students decreased from 47 (32.4%) to 28 (19.3%). Whilst 16 (11%) students originally disagreed and 6 (4.1%) more disagreed that they knew functions (methods), these were decreased to 14 (9.6%) and 2 (1.3%) respectively during the study. Further to this, 24 (16.5%) students strongly agreed and 72 (49.5%) more agreed that they know decision making in the pre-study. The number of strongly agreed and agreed students then increased to 25 (17.2%) strongly agree and 74 (51%) agree in the post-study. Accordingly, a total of 15 (10.3%) students strongly disagreed and disagreed that they knew decision making in the pre-study whereas this was decreased to 8 (5.5%) in the post-study. Finally, 12 (8.2%) students strongly agreed and 31 (21.3%) more agreed that they knew loops before the study was conducted. Having played the game, the number of strongly agreed students increased to 16 (11%) and the number of agreed students increased to 72 (49.6%). Further to these, those students who were neutral about how loops work was reduced 44 (30.3%) to 31 (21.3%) during the game-play. It is also pleasing to report that while 49 (33.7%) students strongly disagreed and disagreed that they knew loops in the pre-study, this was reduced to only 14 (9.6%) in the post-study. As a result, the raw data presented in Figure 6.19 shows that students felt that their knowledge was increased regarding all programming constructs during the study.

An interesting outcome of the Greenwich study was the noticeable decrease in the total number of students who did not know how programming sequence works whereas in other programming constructs this was either stable or increased. The reason for the noticeable decrease was because programming sequence was introduced in the first level of the game and therefore, it was accessible to everyone in the game. On the other hand, students needed to complete early levels in the game in order to access the functions, decision making and loop elements of the game. As some students did not reach the higher levels during the allocated time for the study, these students were unable to assess whether or not other programing constructs (i.e. functions, decision making, loops) introduced in the game impacted their knowledge. Thus, the number of students who did not know how other programming constructs work either stayed the same (functions) or increased (decision making, loops). This is because those students never saw these constructs in the game and therefore, they were unable to answer the same question in the post-study as they were asked whether or not they felt an improvement in their knowledge. To analyse whether or not the increase students felt in their knowledge is significant, descriptive statistics and a Wilcoxon

Signed Ranks test was performed.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre Sequence Knowledge	145	**2.88**	1.235	0	5
Pre Functions Knowledge	145	**3.05**	1.145	0	5
Pre Decision Knowledge	145	**3.40**	1.089	0	5
Pre Loop Knowledge	145	**2.53**	1.354	0	5
Post Sequence Knowledge	145	**3.60**	1.017	0	5
Post Functions Knowledge	145	**3.62**	1.014	0	5
Post Decision Knowledge	145	**3.61**	1.203	0	5
Post Loop Knowledge	145	**3.37**	1.295	0	5

Table 6.17 – Descriptive statistics of students' perception of their knowledge on programming constructs in the pre and post study of the Greenwich study.

Table 6.17 shows the descriptive statistics and population mean differences between the pre and post study for all programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops). For each programming construct assessed above, the post study population mean was higher than the pre study population mean value which provides evidence that students felt that their knowledge regarding all four programming constructs were increased during the study. Furthermore, Table 6.17 demonstrate that the highest knowledge difference happened in loops (*mean difference* =0.84) whereas the lowest difference happened in decision making (*mean difference* = 0.21) in the Greenwich study.

Ranks		N	Mean Rank	Sum of Ranks
Post Sequence Knowledge – Pre Sequence Knowledge	Negative Ranks	18[a]	52.28	941.00
	Positive Ranks	81[b]	49.49	4009.00
	Ties	46[c]		
	Total	145		
Post Functions Knowledge – Pre Functions Knowledge	Negative Ranks	17[d]	40.82	694.00
	Positive Ranks	67[e]	42.93	2876.00
	Ties	61[f]		
	Total	145		
Post Decision Knowledge – Pre Decision Knowledge	Negative Ranks	22[g]	33.25	731.50
	Positive Ranks	44[h]	33.63	1479.50
	Ties	79[i]		
	Total	145		
Post Loop Knowledge – Pre Loop Knowledge	Negative Ranks	10[j]	23.10	231.00
	Positive Ranks	68[k]	41.91	2850.00
	Ties	67[l]		
	Total	145		

a. Post Sequence Knowledge < Pre Sequence Knowledge
b. Post Sequence Knowledge > Pre Sequence Knowledge
c. Post Sequence Knowledge = Pre Sequence Knowledge
d. Post Functions Knowledge < Pre Functions Knowledge
e. Post Functions Knowledge > Pre Functions Knowledge
f. Post Functions Knowledge = Pre Functions Knowledge
g. Post Decision Knowledge < Pre Decision Knowledge
h. Post Decision Knowledge > Pre Decision Knowledge
i. Post Decision Knowledge = Pre Decision Knowledge
j. Post Loop Knowledge < Pre Loop Knowledge
k. Post Loop Knowledge > Pre Loop Knowledge
l. Post Loop Knowledge = Pre Loop Knowledge

Wilcoxon Signed Ranks Test	Post Sequence Knowledge – Pre Sequence Knowledge	Post Functions Knowledge – Pre Functions Knowledge	Post Decision Knowledge – Pre Decision Knowledge	Post Loop Knowledge – Pre Loop Knowledge
Z	-5.500[m]	-5.079[m]	-2.472[m]	-6.648[m]
Asymp. Sig. (2-tailed)	.000	.000	.013	.000

m. Based on negative ranks

Table 6.18 – Wilcoxon signed Ranks test results of students' perception of their knowledge on programming constructs in the pre and post study of the Greenwich study.

A Wilcoxon signed ranks test was performed in order to evaluate whether or not the difference students felt in their knowledge on programming constructs was significant. The test results revealed that there was an increase in students' perception of their knowledge in

programming sequence (*M*=2.88, *SD*=1.23 in the pre-study; *M*=3.60, *SD*=1.01 in the post-study), functions (*M*=3.05, *SD*=1.14 in the pre-study; *M*=3.62, *SD*=1.01 in the post-study), decision making (*M*=3.4, *SD*=1.08 in the pre-study; *M*=3.61, *SD*=1.2 in the post-study) and loops (*M*=2.53, *SD*=1.35 in the pre-study; *M*=3.37, *SD*=1.29 in the post-study) between pre-study and post-study conditions.

As illustrated in Table 6.18, the Wilcoxon signed ranks test results show that the increase happened in students' perception of their knowledge regarding programming sequence ($z=5.5$, $p<0.05$), functions ($z=5.079$, $p<0.05$), decision making ($z=2.472$, $p<0.05$) and loops ($z=6.648$, $p<0.05$) is significant. As the 2-tailed significant value was less than 0.05 in all cases, the null hypotheses that indicate the groups' perception of their knowledge regarding programming sequence, functions, decision making and loops are all rejected. In this case, the Wilcoxon signed ranks test results support the alternative hypotheses that students' perception of their knowledge regarding all programming constructs is significantly increased between the pre and the post study during Greenwich study.

In the Cyprus study students rated the lowest improvement in loops whereas in the Greenwich study this was in decision making. Equally, students in the Cyprus study rated the highest improvement in programming sequence and this was in loops in the Greenwich study. One reason for this change may be because of the difference in computer programming curriculums being taught to them. Nevertheless, the findings of the Greenwich study support the findings of the Cyprus study and provide strong evidence that the students' perception of their knowledge had indeed improved after they played the game.

6.3.4 Research Question 7, 8 – Is there a difference in students' problem solving abilities and the ability to visualise programming constructs from given problems between the pre and the post study?

The seventh research question was designed to assess whether or not students felt their problem solving abilities were developed during the study and the eighth research question investigated whether or not they thought the game supported visualising how programming constructs work. Similar to the Cyprus study, students were first asked to rate their problem solving abilities in the pre-study and a similar question was asked in the post-study after they played the game.

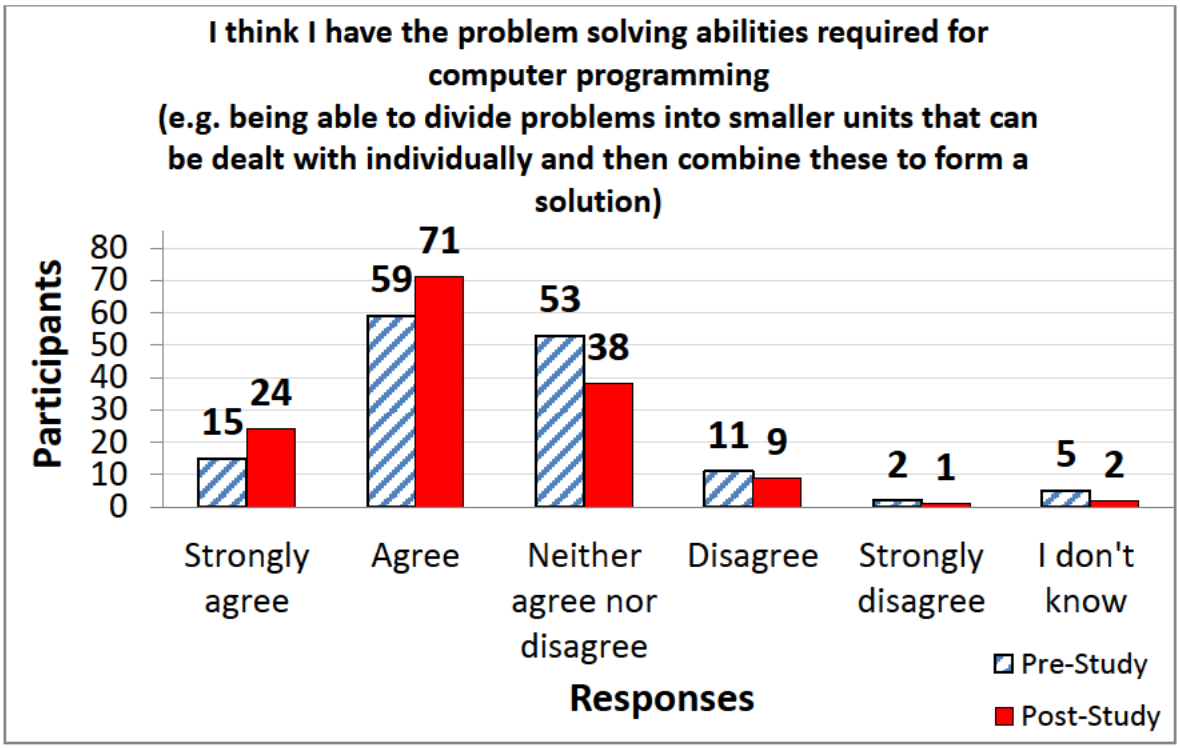

Figure 6.20 – Students' perception of their problem solving abilities between the pre and the post study in the Greenwich study

In the pre-study, 15 (10.3%) students strongly agreed and 59 (40.6%) more agreed that they have the problem solving abilities required learning for computer programming. While 53 (36.5%) students were neutral, a total of 13 (8.9%) students strongly disagreed and disagreed that they have the problem solving abilities required for learning computer programming. Having played the game, the number of strongly agreed students increased from 15 (10.3%) to 24 (16.5%) and the number of agreed students increased from 59 (40.6%) to 71 (48.9%). Correspondingly, the number of neutral students decreased from 53 (36.5%) to 38 (26.2%). Those students who strongly disagreed and disagreed that they have the problem solving abilities required for learning computer programming also decreased from 13 (8.9%) to 10 (6.8%). Finally, the study had a minor impact on students who did not want to answer this question as these were decreased from 5 (3.4%) to 2 (1.3%) during the study. Overall, the raw data captured shows that prior to the study, only 74 students (51%) strongly agreed and agreed that they have problem solving abilities to learn computer programming whereas this number rose to 95 (65.5%) after students played the game.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre Problem Solving Abilities	145	**3.25**	.939	0	5
Post Problem Solving Abilities	145	**3.67**	.850	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post Problem Solving Abilities – Pre Problem Solving Abilities	Negative Ranks	**30[a]**	**49.70**	1491.00
	Positive Ranks	**67[b]**	**48.69**	3262.00
	Ties	**48[c]**		
	Total	145		

a. Post Problem Solving Abilities < Pre Problem Solving Abilities

b. Post Problem Solving Abilities > Pre Problem Solving Abilities

c. Post Problem Solving Abilities = Pre Problem Solving Abilities

Wilcoxon Signed Ranks Test	Post Problem Solving Abilities – Pre Problem Solving Abilities
Z	**-3.305[d]**
Asymp. Sig. (2-tailed)	**.001**

d. Based on negative ranks

Table 6.19 – Descriptive statistics and Wilcoxon signed ranks test results of students' perception of their problem solving abilities in the pre and post study of Greenwich study.

To measure whether or not the increase in the students' perception of their problem solving abilities is statistically significant, a Wilcoxon signed ranks test was performed and descriptive statistics analysed. As shown from Table 6.19, descriptive statistics revealed that there is an increase in students' perception of their problem solving abilities for learning computer programming between pre (M=3.25, SD=0.93) and post (M=3.67, SD=0.85) study conditions in the Greenwich study; average rank of 48.69 vs. average rank of 49.7. Additionally, the Wilcoxon signed ranks test results show that the increase in students' perception of their problem solving abilities for learning programming between the pre and the post study is significant (z=3.305, p<0.05). As the 2-tailed significant value is less than 0.05 (p=0.01), the null hypothesis which defends the group's perception of their problem solving abilities to learn computer programming does not differ is rejected. Therefore, the findings support the alternative hypothesis that is to say the students' perception of their

problem solving abilities to learn computer programming is significantly increased between the pre and the post study during the Greenwich study.

Figure 6.21 clearly shows that only a limited number of students were able to visualise programming constructs from given problems prior to the Greenwich study whereas this increased considerably after students played the game. In the pre-study, a total of 52 (35.5%) strongly agreed and agreed that they have the ability to visualise programming constructs from given problems. In the post-study, this number was increased to 95 (65.5%). Whilst those students who were neutral decreased from 53 (43.4%) to 35 (24.1%), the number of strongly disagreed students declined from 9 (6.2%) to only 1 (0.6%) and the number of disagreed students decreased from 26 (17.9%) to 11 (7.5%). Finally, those who did not answer reduced from 5 (3.4%) to 3 (2%).

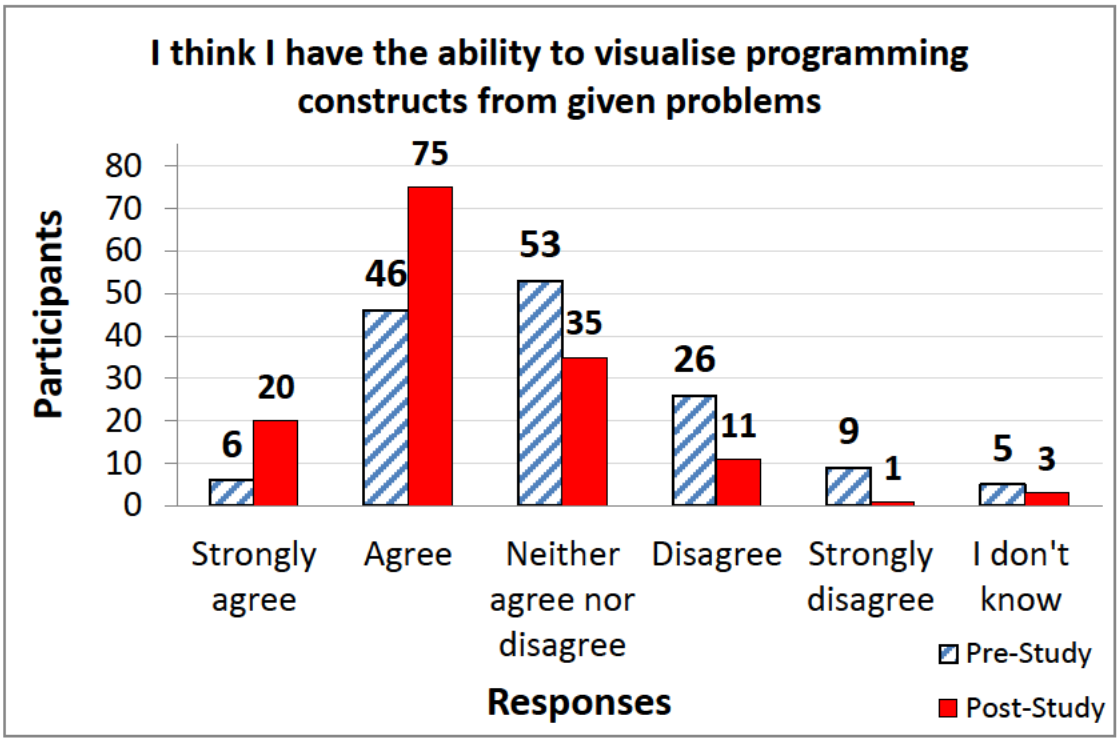

Figure 6.21 – Students' perception of their ability to visualise programming constructs from given problems between the pre and the post study in the Greenwich study

Table 6.20 presents the Wilcoxon signed ranks test results along with descriptive statistics of students' perception of their ability to visualise programming constructs from given problems between the pre and the post study in the Greenwich study. As shown from the findings, there was an increase in the students' perception of their ability to visualise programming constructs from given problems between pre (M=3.01, SD=1.06) and post

(*M*=3.65, *SD*=0.82) study conditions in the Greenwich study; average rank of 42.3 vs. average rank of 53.2. Further to this, Wilcoxon signed ranks test results show that the increase happened in students' perception of their ability to visualise programming constructs from given problems is significant ($z=5.242$, $p<0.05$). As 2-tailed significant value is very close to 0 ($p=0.000$), the null hypothesis which specifies that the groups' perception of their ability to visualise programming constructs from given problems does not differ is rejected. The Wilcoxon signed ranks test results support the alternative hypothesis that is to say the students' perception of their ability to visualise programming constructs from given problems is significantly increased between the pre and the post study during the Greenwich study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre visualising prog. constructs	145	**3.01**	1.068	0	5
Post visualising prog. constructs	145	**3.65**	.829	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post visualising prog. constructs – Pre visualising prog. constructs	Negative Ranks	**25**[a]	**42.30**	1057.50
	Positive Ranks	**75**[b]	**53.23**	3992.50
	Ties	**45**[c]		
	Total	145		

a. Post visualising prog. constructs < Pre visualising prog. constructs

b. Post visualising prog. constructs > Pre visualising prog. constructs

c. Post visualising prog. constructs = Pre visualising prog. constructs

Wilcoxon Signed Ranks Test	Post visualising prog. constructs – Pre visualising prog. constructs
Z	**-5.242**[d]
Asymp. Sig. (2-tailed)	**.000**

d. Based on negative ranks

Table 6.20 – Descriptive statistics and Wilcoxon signed ranks test results of students' perception of their ability to visualise programming constructs from given problems in the pre and post study of the Greenwich study.

The results obtained from the seventh (i.e. difference in students' perception of their problem solving abilities) and eighth (i.e. difference in students' perception of their ability to

visualise constructs) research questions in the Greenwich study are consistent with the findings obtained in the Cyprus study. In the Cyprus study the number of students who strongly agreed and agreed that they have problem solving abilities required for learning programming is increased from 55 (80.8%) to 59 (86.7%) whereas this increased from 74 (51%) to 95 (65.5%) in the Greenwich study. Correspondingly, the number of students who strongly agreed and agreed that they have the ability to visualise programming constructs from given problems increased from 37 (54.4%) to 64 (94.1%) in the Cyprus study and a similar increase was observed in the Greenwich study from 52 (35.8%) to 95 (65.5%). Although the extent to which the studies impacted on students' perception varied extensively, a significant positive increase was observed in both studies regarding students' perception of their problem solving abilities and the ability to visualise programming constructs from given problems.

6.3.5 Research Question 9 – Is there a difference in students' perception of the difficulty of computer programming between the pre and the post study?

As in the Cyprus study, an additional question was asked to students to observe the difference between their perception regarding the difficulty of learning programming before and after their game-play in the Greenwich study. In the pre-study, only 6 (4.1%) out of 145 students rated the difficulty of computer programing as very easy and 19 (13.1%) more rated the difficulty of computer programming as easy. While 70 (48.2%) students were neutral, a total of 48 (33.1%) students rated learning computer programming as either difficult or very difficult. Having played the game, the number of very easy ratings increased from 6 (4.1%) to 18 (12.4%) and the number of easy ratings increased from 19 (13.1%) to 50 (34.4%). Neutral students decreased from 70 (48.2%) to 65 (44.8%) and those students who rated learning computer programming difficult or very difficult decreased from 48 (33.1%) to 10 (6.8%). As a result, the findings indicate that these students were influenced by the study and rated learning computer programming easier than before their participation in the study.

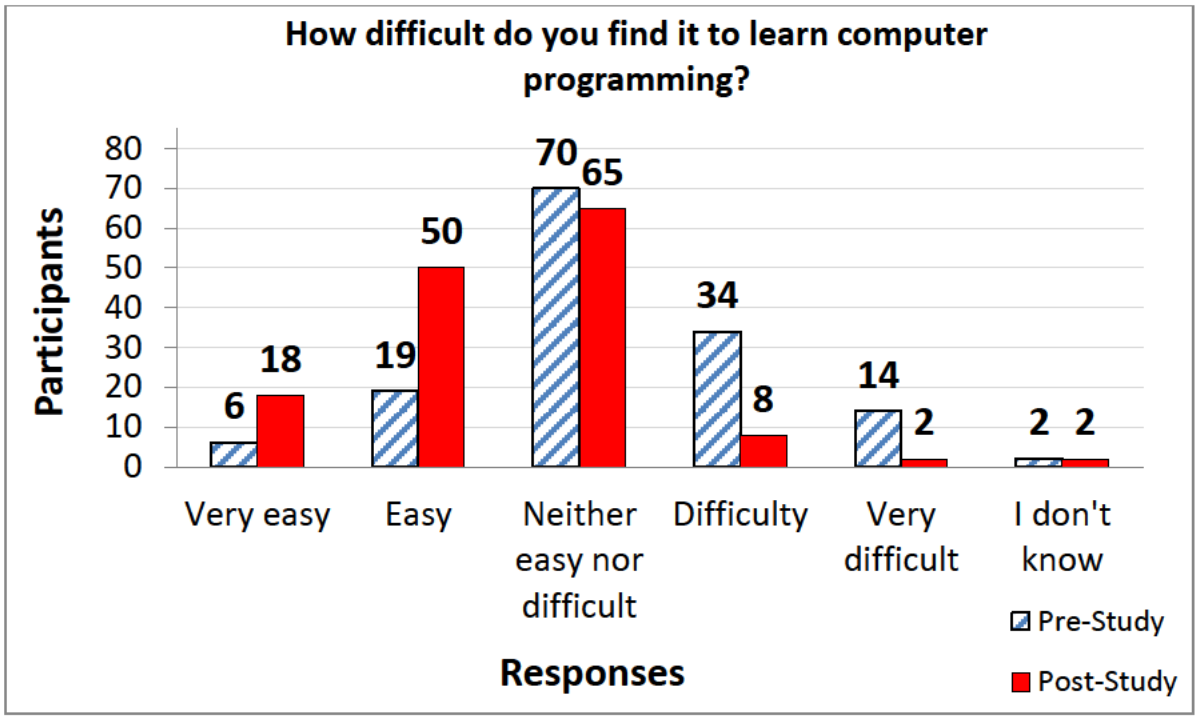

Figure 6.22 – Students' perception of the difficulty of learning computer programming between the pre and post study in the Greenwich study.

A Wilcoxon signed ranks test and descriptive statistics were undertaken to assess whether or not the difference in students' perception regarding the difficulty of learning computer programming between the pre and the post study in the Greenwich study is significant. The findings provide evidence that students found learning computer programming easier in the post-study (M=2.65; SD = 0.96) than in the pre-study (M=3.42; SD = 0.81) conditions (average rank of 31.77 vs. average rank of 47.41). It is crucial to highlight here that M=1 represents very difficult and M=5 represents very easy. Further to these, the Wilcoxon signed ranks test results show that the difference in students' perception regarding the difficulty of learning computer programming between the pre and post study is significant (z=7.06, p<0.05). As 2-tailed significant value is very close to 0 (p=0.000), the null hypothesis which specifies that the group's perception of difficulty of learning computer programming does not change is rejected. Therefore, the findings from the Wilcoxon signed ranks test support the alternative hypothesis that is to say students found learning computer programming easier after playing the game than before they participated in the Greenwich study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre computer programming difficulty	145	**2.65**	.968	0	5
Post computer programming difficulty	145	**3.42**	.814	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post computer programming difficulty – Pre computer programming difficulty	Negative Ranks	**11[a]**	**31.77**	349.50
	Positive Ranks	**79[b]**	**47.41**	3745.50
	Ties	**55[c]**		
	Total	145		

a. Post computer programming difficulty < Pre computer programming difficulty

b. Post computer programming difficulty > Pre computer programming difficulty

c. Post computer programming difficulty = Pre computer programming difficulty

Wilcoxon Signed Ranks Test	Post computer programming difficulty – Pre computer programming difficulty
Z	**-7.060[d]**
Asymp. Sig. (2-tailed)	**.000**

d. Based on negative ranks

Table 6.21 – Descriptive statistics and Wilcoxon signed ranks test results of students' perception regarding the difficulty of learning computer programming between the pre and post study in the Greenwich study.

6.3.6 Summary of findings regarding research questions

Research Question#	Pairs	Mean Difference (*M*)	Two tailed significant value (p)
1	Pre video game attitude to learning programming through game-play – Post video game attitude to learning programming through game-play	**0.36**	**0.001**
2	Pre computer programming motivation – Post computer programming motivation	**0.32**	**0.002**
3	Pre sequence knowledge – Post Sequence Knowledge	**0.72**	**0.000**
4	Pre functions knowledge – Post functions knowledge	**0.57**	**0.000**
5	Pre decision making knowledge – Post decision knowledge	**0.21**	**0.000**
6	Pre loop knowledge – Post loop knowledge	**0.84**	**0.000**
7	Pre problem solving abilities – Post problem solving abilities	**0.42**	**0.000**
8	Pre visualising constructs – Post visualising constructs	**0.64**	**0.000**
9	Pre difficulty of computer programming – Post difficulty of programming	**0.77**	**0.000**

Table 6.22 – Summary of Wilcoxon signed ranks test results of research questions used in the Greenwich study.

Table 6.22 shows a summary of all the research questions used in the Greenwich study. The results show that the two two-tailed significant value (p) was lower than 0.05 for all the research questions which indicates that the findings from the Wilcoxon signed ranks tests provided strong evidence to reject the null hypothesis and accept the alternative hypothesis for each research question evaluated in the Greenwich study. The most significant difference between the groups happened in the students' perception of their knowledge in loops (*mean difference* = 0.84) and the least significant difference happened in students' perception of their knowledge in decision making (*mean difference* = 0.21). These results did not exactly match with the findings of the Cyprus study as it was found that the most significant difference happened in visualising programing constructs from given problems (*mean difference* = 1.22) and the least significant difference happened in the attitude of students' to learning computer programming through games (*mean difference* = 0.27).

At this point, it is important to state that data in the Cyprus study was captured from a relatively normal distributed population while data in the Greenwich study came from a non-normal distribution population. This means that the data were spread roughly symmetric in

the Cyprus study where half of the observations were less than the population mean value, and the other half were greater than the mean value. However, in the Greenwich study the distribution of data was skewed to the right and had major Kurtosis issues. In other words, majority of data were heavily clustered around the mean value whereas the rest of the data had very low variations. Therefore, it is anticipated that the difference between the two studies (i.e. the Cyprus and the Greenwich) was because of the distribution of data.

6.3.7 Statistical Correlations

As in the Cyprus study, the associations among the five cognitive skills fundamental to computational thinking (i.e. conditional logic, algorithmic thinking, debugging, simulation and socialising) are investigated in the post-study of the Greenwich study. In addition to analysing the association among these skills, the correlations between these skills and how far students progressed through the game, were also investigated. As data did not come from a normally distributed population in the Greenwich study, a Pearson product-moment correlation was not suitable to the data captured. In this case, the Spearman's rank-order correlation which is the non-parametric equivalent of the Pearson correlation (Laerd Statistics, 2012b) was used to examine the association among these skills and their correlations with how far students progressed in the game. The Spearman's rank-order correlation has fewer assumptions when compared to Pearson's product-moment correlation as a) it can be applied to non-normal data b) it converts any ordinal data (e.g. Likert scale questionnaires) into ranked data and c) it is suitable for any data set (ordinal, interval/ratio) regardless of whether or not it is normally distributed. At this point, one could argue that if the Spearman's rank-order correlation is suitable for both normally and non-normal data, why it is not used all the time on behalf of a Pearson product-moment correlation. The reason for this is because Spearman's rank-order correlation presents a limited statistical analysis and therefore it is less informative when compared to a Pearson product-moment correlation **(**Hauke & Kossowski, 2011). As an example, in a Pearson product-moment correlation, the percentage of variance (also called the coefficient of determination) can be calculated by taking the square of the correlation whereas in a Spearman's rank-order correlation this is considered inappropriate for justifying the effect size of a relationship (How2stats, 2011). Hence what Spearman's rank-order correlation provides as an outcome is quite limited for further analysis when compared to the outcome of Pearson product-moment correlation. Nonetheless, a Pearson product-moment correlation was simply not available for analysing the data in the Greenwich study

and thus, the non-parametric equivalent of this that is the Spearman's rank-order correlation was used.

Figure 6.23 shows the perception of students on how well they think skills that encompass computational thinking were blended into *Program Your Robot*. For the first two skills, students provided their perception on whether or not playing the game required evaluating conditions and thinking algorithmically. For the rest of the skills, students were asked to rate whether or not debugging, run-time and sharing ideas were helpful to them when they designed their solutions in the game. The data obtained shows that 120 (82.7%) out of 145 students strongly agreed, and agreed, that the game requires thinking logically and evaluating conditions. Moreover, 98 (67.5%) students strongly agreed, and agreed, that the game enhanced (or has the potential to enhance) their ability in algorithmic thinking. Further to these, 89 (61.3%) out of 145 students used the debug mode in the game and 71 (79.7%) out of 89 students strongly agreed and agreed that debug mode was useful to detect errors in their solutions. Correspondingly, 103 (71%) out of 145 students strongly agreed and agreed that run-time mode in the game successfully simulates how computer algorithms work. Finally 90 (62%) out of 145 students shared strategies and ideas during their game-play. From among these students 63 (70%) of them strongly agreed and agreed that sharing ideas during their game-play was helpful to them.

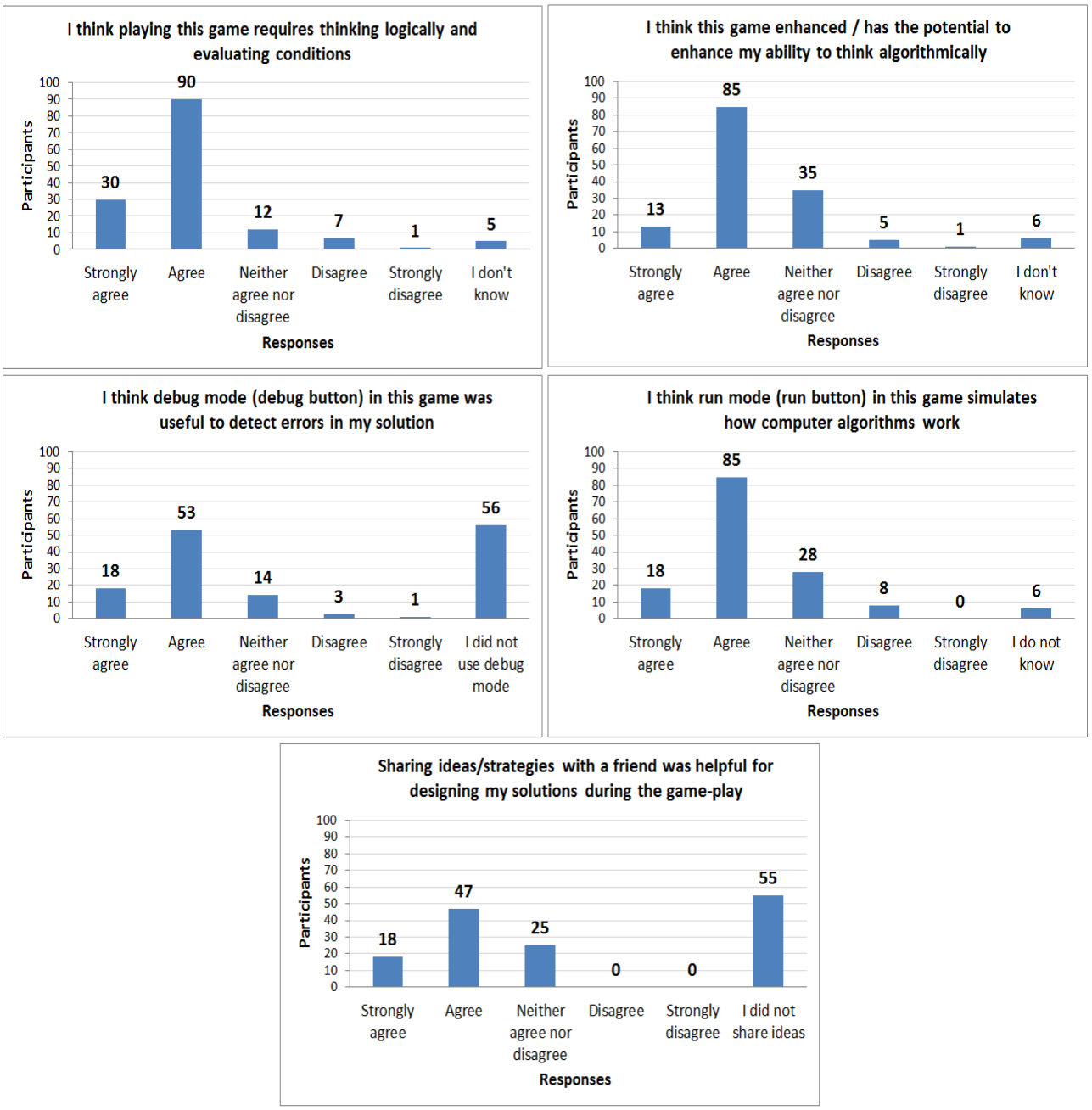

Figure 6.23 –Students' perception of how well computational thinking skills were presented in the game.

Although the results obtained in the Greenwich study were generally consistent with the results obtained from the Cyprus study, there were some differences between the two studies regarding the computational thinking skills due to the students' background, previous experiences regarding programming and programming curriculum being taught. While there was no large difference between the groups regarding conditional logic, algorithmic thinking and simulating solutions, participants provided different viewpoints in debugging and sharing

ideas. The findings regarding conditional logic and simulating solutions between the two studies were very consistent as in the Greenwich study it was found that 82.7% students strongly agreed, and agreed, the game requires thinking logically and evaluating condition while this was 85.2% in the Cyprus study. Correspondingly, in the Greenwich study 71% of students strongly agreed, and agreed, that run-time in the game, simulates how computer algorithms work and this was found to be 76.4% in the Cyprus study. Additionally, the majority of participants in both studies (71% in the Greenwich; 79.3% in the Cyprus) felt that the game enhanced (or has the potential to enhance) their algorithmic thinking ability after they played the game. Despite the fact that students' feedback was positive in both studies; there was a distinct difference between the two studies regarding the usage of debugging and sharing ideas and strategies in the game. While 63 (92.6%) out of 68 students used the debug mode in the Cyprus study, only 89 (61.3%) out of 145 students used the same feature in the Greenwich study. Similarly, 66 out of 68 (97%) students shared ideas and strategies during the Cyprus study whereas in the Greenwich study this was only 90 (62%).

The difference between the groups regarding debugging solutions is very interesting and it postulated that this was because students were being taught a different programming curriculum. In the Cyprus study students learned how to debug programs and what benefits can be obtained from this in their lectures before their participation in the study whereas in the Greenwich study the same concept had not yet been introduced to students in their tutorial hours. Therefore, the majority of students in the Cyprus study already knew what the debugging was before they participated whereas in the Greenwich study students had not been introduced to this concept. The results show that only a small percentage of students in the Cyprus study (7.4%) ignored the debug feature in the game whereas nearly the half of the students (38.7%) did not use the same feature in the Greenwich study. This is a solid demonstration of learning behaviour as it provides evidence that students tried to apply concepts they learned in their lectures into the game environment.

It is known from the literature that computational thinking skills are transferable skills (Wing 2008; Wing 2010) and that a game environment can be used as a framework to utilise and develop a wide range of skills and knowledge that might be transferable across a wide section of industry (Connolly, Stansfield & Hainey, 2007). The difference between the behaviour of students regarding the usage of the debug mode provides evidence that the opposite of this is also possible as students who already knew about *debugging* used the debug-mode in the game more often than the other students. In other words, the behaviour of students and how they played *Program Your Robot* was considerably different based on what

they knew about computer programming.

When the Cyprus and the Greenwich study results are matched regarding the data obtained for computational thinking skills (see Figures 6.10 and 6.23), it can be observed that there is a considerable difference in *cooperation* (*sharing ideas and strategies)* between the studies. The results show that only a very minor percentage of the population (3%) did not share ideas and strategies during the Cyprus study whereas this was a considerable percentage of population (38%) in the Greenwich studies. In other words, the students in the Cyprus study cooperated and communicated more often than the students in the Greenwich study. Although it is not possible to detect exact reasons for this as there were simply too many variables to consider, it is anticipated that this is related to *situated learning*.

In their seminal work, Lave and Wenger (1991) described that learning takes place in the same context it is applied and it is not a simple transmission of abstract knowledge from one individual to another, but rather a social process where knowledge is co-constructed. Hung (2002) took this work further and investigated how important it is to be social when learning. He argued that students who learn in communities with shared interests tend to benefit more than students who learn in isolated environments. He also argued that social learning environments provide consistent experiences and therefore people can benefit from the knowledge of others who are more knowledgeable than they are.

According to the data presented in Figures 6.10 and 6.23, the students in the Cyprus study provided more positive feedback than the students in the Greenwich study. One can conjecture that one of the reasons why this happened was because the Cyprus study had a better *situated learning* environment than the Greenwich study as almost all students shared their ideas and experiences with others. In other words, because the students in the Cyprus study coordinated and communicated more compared to the Greenwich study, they also felt that they benefited more from the game in terms of learning.

Spearman's rank correlation coefficient		Maximum game level participants reached	Conditional Logic and Evaluating Conditions	Algorithmic Thinking	Running and Simulating Solutions	Debugging and Handling Errors	Cooperation (Sharing ideas and strategies)
Maximum game level participants reached	Correlation Coefficient	1.000	.307**	.277**	.332**	.122	.148
	Sig. (2-tailed)	.	.000	.005	.000	.145	.075
	N	145	145	145	145	145	145
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.307**	1.000	**.766****	**.810****	.137	.069
	Sig. (2-tailed)	.000	.	.000	.000	.101	.408
	N	145	145	145	145	145	145
Algorithmic Thinking	Correlation Coefficient	.227**	**.766****	1.000	**.754****	.163*	.119
	Sig. (2-tailed)	.006	.000	.	.000	.049	.156
	N	145	145	145	145	145	145
Running and Simulating Solutions	Correlation Coefficient	.332**	**.810****	**.754****	1.000	.152	.031
	Sig. (2-tailed)	.000	.000	.000	.	.067	.709
	N	145	145	145	145	145	145
Debugging and Handling Errors	Correlation Coefficient	.122	.137	.163*	.152	1.000	**.525****
	Sig. (2-tailed)	.145	.101	.049	.067	.	.000
	N	145	145	145	145	145	145
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.148	.069	.119	.031	**.525****	1.000
	Sig. (2-tailed)	.075	.408	.156	.709	.000	.
	N	145	145	145	145	145	145

**. Correleation is significant at the 0.01 level (2-tailed).
*. Correleation is significant at the 0.05 level (2-tailed).

Table 6.23 – Spearman's rank-order correlation coefficient showing relationships among computational thinking skills and also between these skills and the maximum game level students achieved.

A Spearman rank-order correlation was performed in order to measure the correlations among computational thinking skills as well as how these skills are related to the maximum level players reached in the game. As shown from Table 6.23, the correlations among computational thinking skills are positive in all cases where some of these relations are strong and significant, others are not. Similar to the Cyprus study results, the correlations between the maximum level students achieved and the five categories of computational thinking skills are always either weak (Spearman's rho <= +0.39) or insignificant ($p > 0.05$). It is crucial to highlight that the Spearman's rank-order correlations in the Greenwich study support the findings of Pearson's correlations in the Cyprus study as no strong correlation was identified between achieving high levels and computational thinking skills in both studies. In other words, despite not achieving high levels in the game a number of students felt that their abilities in *conditional logic*, *algorithmic thinking* and *simulating solutions* were enhanced after playing the game. Further to this, the correlation between the maximum level students

achieved and *debugging* and/or *cooperation* was found to be neither strong nor significant. This means that there is no evidence to prove achieving high levels in the game would result in more *debugging* of solutions or more *cooperation* among students.

The correlations among five computational thinking skills were investigated to identify whether or not these skills are related to each other in the game. The results show that there was a strong positive and significant correlation between a) conditional logic and algorithmic thinking ($r = 0.766$; $n=145$; $p=0.000$) b) conditional logic and simulating solutions ($r=0.81$; $n=145$; $p=0.000$) and c) algorithmic thinking and simulating solutions ($r=0.754$; $n=145$; $p=0.000$). The Spearman's rank-order coefficient proves that the associations are positive, strong ($r>=0.7$) and significant ($p<0.05$) in all three cases. More importantly, the findings obtained from Spearman's rank-order coefficient regarding the associations are very consistent with the correlations identified previously in the Cyprus study. The results of Greenwich study showed that an increase in conditional logic causes an increase in algorithmic thinking and also an increase in the ability to simulate how computer algorithms work. Correspondingly, an increase in algorithmic thinking results an increase in simulating solutions. This means that when players use conditional logic in the game they also develop abilities in algorithmic thinking and simulating solutions.

Despite the fact that Spearman's rank-order correlation shows how strong the correlations are among computational thinking skills, it was simply not possible to investigate the percentage of variance as a coefficient of determination cannot be calculated in a Spearman's rank-order correlation (Laerd statistics, 2012b). As argued previously, this is a limitation of the Spearman's rank-order correlation and one of the vital points that separates it from a Pearson's coefficient correlation. Therefore, although three scatterplots were generated to provide more evidence on correlations, these were merely used to show whether or not relationships are linear. It was found that three out of five computational thinking skills (i.e. conditional logic, algorithmic thinking and simulating solutions) were successfully integrated in to the game environment and their relationships are rationally linear.

Figure 6.24 shows scatterplots where strong, positive and significant relationships are identified among computational thinking skills. In the Cyprus study, these scatterplots were used to calculate associations between two computational thinking skills due to the fact that the distribution of data was adequately normal. As this was not possible in the Greenwich study, the scatterplots were only used to look whether or not the strongly identified relationships are linear. In statistics, scatterplots are always used when calculating a Pearson's correlation coefficient or a Spearman's rank-order correlation coefficient as this serves as a

double-check method to support these.

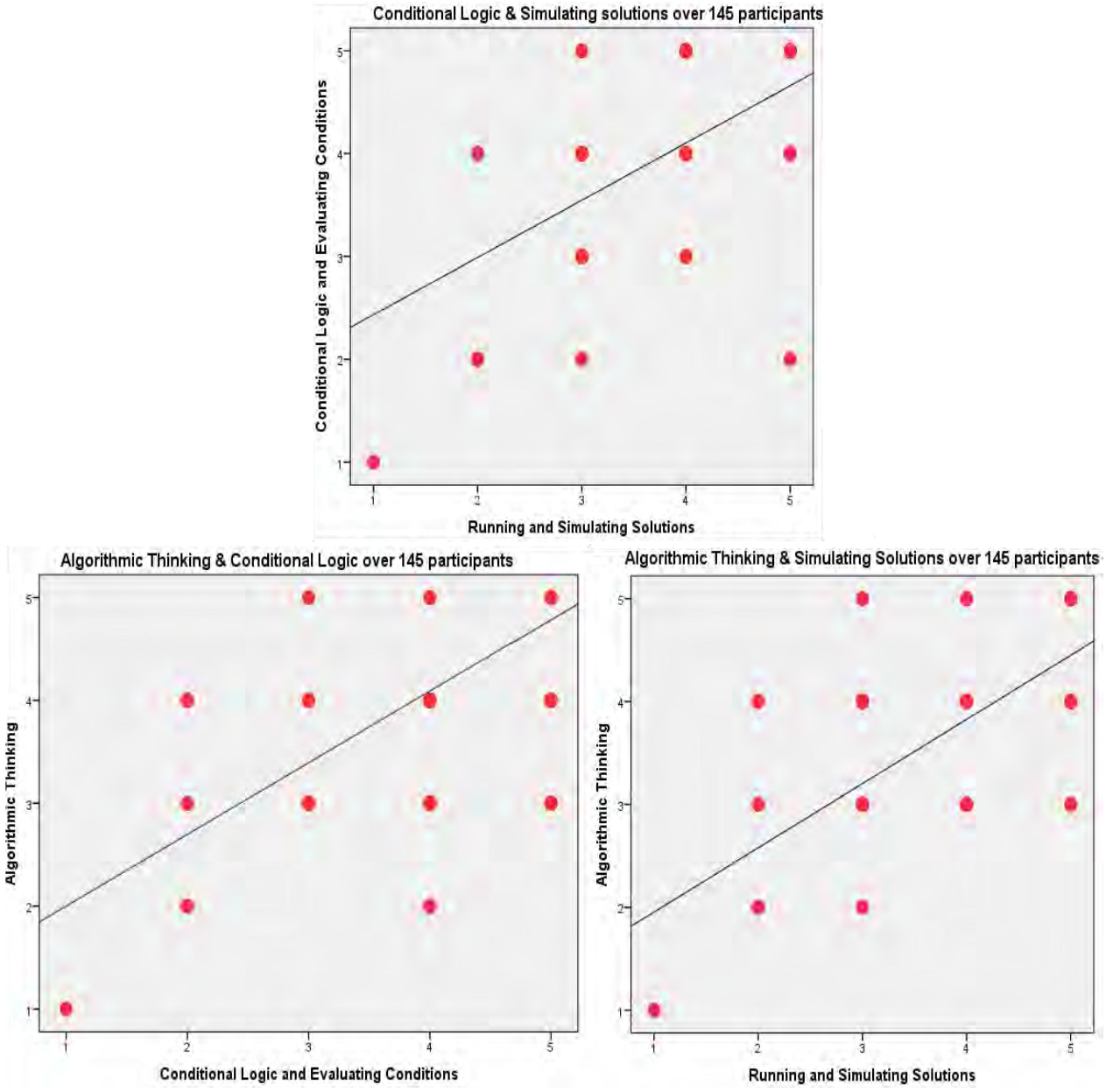

Figure 6.24 – Scatterplots showing strong correlations among algorithmic thinking, conditional logic and simulating solutions according to data collected in the Greenwich study

The scatterplots presented in Figure 6.24 clearly demonstrate that the strong and significant relationships identified above (i.e. condition logic and simulating solutions; algorithmic thinking and conditional logic; algorithmic thinking and simulating solutions) are reasonably linear. The red dots represent the observations gathered from participants and the linear line represents the associations among the computational thinking skills. For all three cases the

relationship is rationally linear which means that as the value of one variable increases, so does the value of the other variable. The scatterplots for debugging and socialisation are not presented in here as the association between these and other computational thinking skills were neither strong nor significant.

A Spearman's rank-order correlation was conducted between computational thinking skills and programming constructs presented in the game in order to investigate to what extent these are related to each other according to the dataset captured from the Greenwich study. As shown from the results in Table 6.24, all programming constructs introduced in the game (i.e. sequence, functions, decision making and loops) are associated to *conditional logic*, *algorithmic thinking* and *simulating solutions* at a significant level. Despite this, none of these associations are strong enough to conclude that there is a strong relationship between programming constructs and these skills. Additionally, no significant relationship was found between programming constructs and *debugging* and in between programming constructs and *cooperation*. The Spearman's rank-order correlation shows that there was a positive, modest strong and significant relationship between algorithmic thinking and decision making ($r=0.53$; $n=145$; $p=0.000$), and in between simulating solutions and decision making ($r=0.616$; $n=145$; $p=0.000$). Additionally, it was identified that there is a positive, significant but a weak relationship between conditional logic and all programming constructs introduced in the game.

Spearman's rank correlation coefficient		Post Programming Sequence Knowledge	Post Functions Knowledge	Post Decision Making Knowledge	Post Loops Knowledge
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.341**	.361**	.420**	.361**
	Sig. (2-tailed)	.000	.000	.008	.000
	N	145	145	145	145
Algorithmic Thinking	Correlation Coefficient	.491**	.450**	**.530****	.345**
	Sig. (2-tailed)	.000	.000	.000	.000
	N	145	145	145	145
Running and Simulating Solutions	Correlation Coefficient	.414**	.452**	**.616****	.380**
	Sig. (2-tailed)	.000	.000	.000	.000
	N	145	145	145	145
Debugging and Handling Errors	Correlation Coefficient	.252	.161	.096	.119
	Sig. (2-tailed)	.020	.053	.249	.152
	N	145	145	145	145
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.272	.117	.076	.040
	Sig. (2-tailed)	.010	.160	.364	.636
	N	145	145	145	145

**. Correleation is significant at the 0.01 level (2-tailed).

Table 6.24 – Spearman's rank-order correlations between computational thinking skills and students' perception of their programming knowledge in the Greenwich study.

These findings match with the earlier findings in the Cyprus study expect that in the Cyprus study a modest strong significant relationship between debugging solutions and functions (r=0.628; n=68; p=0.000) was identified whereas in the Greenwich study this simply does not exist. It is anticipated that this is because almost all students in the Cyprus study (97%) used the debug feature in the game whereas this was ignored by a considerable percentage of students (38%) in the Greenwich study. The Spearman's rank-order correlations also show that there was no significant relationship between cooperation and programming constructs in the Greenwich study whereas in the Cyprus study a weak but significant relationship was identified between cooperation and decision making (r=0.360; n= 68; p= 0.000) and in between cooperation and loops (r=0.424; n= 68; p= 0.000). This might be due to the fact that majority of students cooperated when playing the game in the Cyprus study whereas this did not happen in the Greenwich study.

These associations show that a number of students with a higher level of knowledge in decision making used algorithmic thinking and simulated solutions more often than the others.

However, as the correlation among these pairs are modest strong (r<=0.7 in all cases), it was simply not possible to conclude that *algorithmic thinking* or *simulating solutions* is directly related to decision making in computer programming. This is to say that neither algorithmic thinking nor simulating solutions are unique to computer programming and that those students who did not have knowledge about decision making in programming also felt that their skills in conditional logic, algorithmic thinking and simulating solutions were developed after playing the game. Complementary to the Cyprus study findings, these results support the arguments of Wing (2010) and more importantly they provide statistical evidence that there are no strong correlations between five categories of computational thinking skills and the four programming constructs presented in *Program Your Robot*.

Spearman's rank correlation coefficient		Difference in the ability to visualise constructs	Difference in programming knowledge	Difference in problem solving abilities
Difference in the ability to visualise constructs	Correlation Coefficient	1.000	**.785****	**.781****
	Sig. (2-tailed)	.	.000	.000
	N	145	145	145
Difference in programming knowledge	Correlation Coefficient	**.785****	1.000	**.767****
	Sig. (2-tailed)	.000	.	.000
	N	145	145	145
Difference in problem solving abilities	Correlation Coefficient	**.781****	**.767****	1.000
	Sig. (2-tailed)	.000	.000	.
	N	145	145	145

** Correleation is significant at the 0.01 level (2-tailed).

Table 6.25 – Spearman's rank-order correlation coefficient showing relationships among visualising constructs, programming knowledge and problem solving abilities between the pre and post study of Greenwich study.

The last Spearman's rank-order correlation was conducted to assess relationships among visualising constructs, programming knowledge and problem solving abilities between the pre and the post study of the Greenwich study. The results show that there was a strong, positive correlation between students' perception of their ability to visualise programming constructs and their perception of programming knowledge, r= 0.785, n=145; p =0.000. A strong positive correlation between participants' perception of their ability to visualise programming constructs and their perception of problem solving abilities was also identified, r=0.781; n=145; p=0.000. Finally, The Spearman's rank-order correlation shows that there was a

strong correlation between participants' perception of their programming knowledge and their perception of problem solving abilities, r=0.767; n=145; p=0.000. These provide evidence that the more students visualised programming constructs from given problems, the more they felt their programming knowledge developed. Additionally, the more students who visualised programming constructs from given problems, the more they felt their problem solving abilities were developed. Hence, the correlations results show that as participants felt that their programming knowledge was improved, they used problem solving abilities and visualised constructs from problems. As a result, students felt that their programming knowledge was increased significantly after playing the game.

6.3.8 Summary of findings regarding correlations

The results of the Greenwich study was further analysed in Chapter 7 section 7.1 along with the Cyprus study results in order to identify whether or not the statistical findings have internal and/or external validity issues. A summary of the Greenwich study results obtained from Spearman's rank-order correlation coefficient assessments is presented below:

There was a strong, positive and significant correlation between:

a) conditional logic and algorithmic thinking, r=0.76; n= 145; p=0.000;
b) conditional logic and simulating solutions, r=0.81; n=145; p=0.000;
c) algorithmic thinking and simulating solutions, r= 0.75; n=145; p=0.000;
d) ability to visualise programming constructs from given problems and problem solving abilities, r=0.78; n=145; p=0.000;
e) ability to visualise programming constructs from given problems and programming knowledge gained, r=0.78; n=145; p=0.000;
f) problem solving abilities and programming knowledge gained, r=0.76; n=145; p=0.000.

There was a modest-strong, positive and significant correlation between:

a) debugging and cooperation, r=0.52; n=145; p=0.000;
b) algorithmic thinking and knowledge in programming sequence, r=0.5; n=145; p=0.000;

c) algorithmic thinking and knowledge in decision making, r=0.53; n=145; p=0.000;

d) simulating solutions and decision making, r=0.61; n=145; p=0.000.

6.4 The PGS study evaluation and statistical analysis

This section analyses the results of the PGS study in the same structure that the Greenwich study results were analysed.

A series of technical difficulties were encountered in the PGS study that impacted on the participants' experience and perception regarding *Program Your Robot*. The most important of these was a network traffic issue that prevented participants playing the game smoothly and therefore caused a degraded performance experience of the game. When participants were invited to play the game in the PGS study, many of them experienced a reduced speed of play and some even reported that they were unable to play the game at all. The Information and Communications Technology (ICT) teacher stopped the study and told participants that the study would be conducted on another day. The network administrator of school later identified that this problem was caused because of an intense network traffic communication between the game and the hosting server where the game is located. After an in-depth investigation, two main reasons were identified as to why a major network traffic issue was raised in the PGS study whereas this was not encountered in previous studies.

Firstly, because the previous studies were conducted in Universities, their network was several orders of magnitude better than a public girls school. Therefore, even if there was high network traffic between the game application and its hosting server, this did not impact on the game performance.

Secondly, there was no control over the PGS study meaning that the author of this study was not present during the study so for ethical reasons the study was conducted via the ICT teacher. In other words, the author was unable to test the game in the school before the study was conducted.

Having recognised that the game was using 5-6% of the bandwidth available in the PGS study, a packet analyser was installed to identify the cause of the traffic being generated. Through using the packet sniffer, it was identified that the network traffic was bound to the hosting server where the game is stored. The network administrator highlighted that the game continuously attempted to work with files in the host directory and when a class of pupils in the same year group attempted to play the game, they overwhelmed the server on their school network. The network administrator also informed that they cannot host the game at their

school as they did not have an available server. To overcome these problems, the game was re-programmed and a standalone version was created by removing the features that potentially caused frequent communication between the game and the hosting server. The standalone version was created through removing the high score submission system and partial features of the save/continue system that uses a series of host files. Consequently, the standalone version was tested on various machines that have no access to the Internet at the University of Greenwich before continuing to the PGS study. A series of viewpoints were captured from those participants who did not experience a major problem in the first attempt of the study and these were removed from the system before participants were re-invited to participate.

Despite the problem clearly being described as a network bandwidth issue to pupils, it was felt that many participants may have returned to the game environment with tarnished views. A total of 85 pupils were invited to participate in the PGS study whereas only 52 (61%) of these completed both the pre and the post study. Unfortunately, a considerable number of pupils dropped out during the second attempt of the PGS study as their first attempt to play may have left a negative experience for them. A total of 33 (38.9%) participants left before completing the post-study and therefore their viewpoints were excluded from the evaluation.

From among those who completed the study, 40 (77%) out of 52 participants were White, 4 (7.6%) participants were Black or Black British, 5 (9.6%) participants were Asian or Asian British and 3 (5.7%) participants had a dual background. All participants in the PGS study were female and 15 years old. As none of the participants were enrolled on a computer programming course, it was not possible to investigate whether or not the difficulty of programming was a major issue for them. Additionally, the results of the PGS study were investigated independently from the Cyprus and the Greenwich studies as the target group was not the same and this study was merely an investigation of whether or not it is possible to use the same game without any modifications for a different target group.

In order to identify which statistical method is appropriate to analyse the results of PGS study, a procedure was carried out to define whether or not the data came from a normally distributed population. This procedure followed a consistent structure with the previous studies that is a *histogram*, *normal quantile – quantile* (Q-Q) *plots*, a *Skewness and Kurtosis normality check* and finally *Shapiro Wilk test*.

Figure 6.25 shows the distribution of data captured for the first research question (the difference in pupils' attitude to learn programming through playing games between the pre and the post study). As shown from the figure, the histogram has major Skewness and Kurtosis issues as the distribution is both asymmetric and peaks over the normal curve. There

is more data on the right tail than would be expected in a normal distribution (Skewness issue) and the peak point of the distribution is way over the peak point of the normal curve (Kurtosis issue). The population mean value ($\mu = 0.64$) is close to 0 but the standard deviation is way above 1 ($\sigma = 1.6$). Additionally, the dataset is not ranged within 2 standard deviation of the mean (between -2 and 2) and expanded much more than this (between -5 and 5). Hence, the histogram provides evidence that the dataset came from a non-normal distribution because observations are heavily concentrated around the mean value.

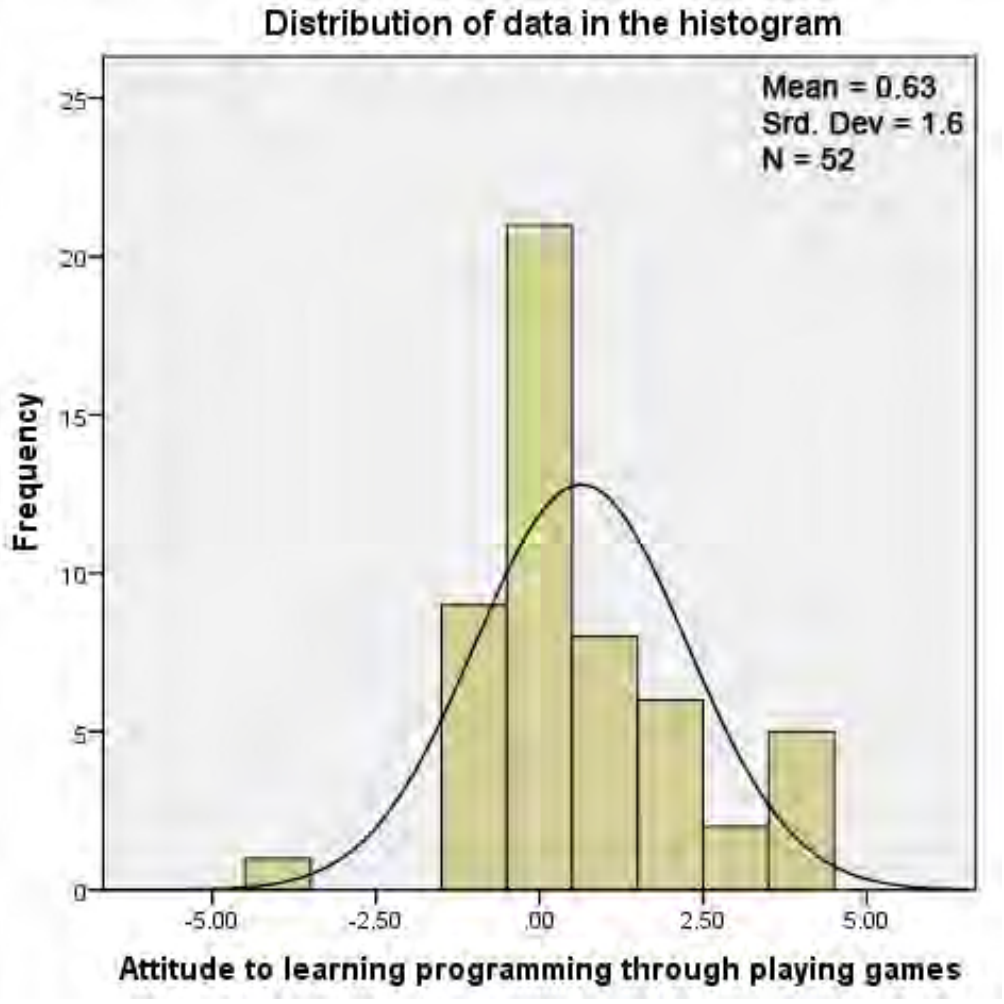

Figure 6.25 – Histogram showing distribution of data captured on the difference between attitudes to learn computer programming through playing games in the PGS study (Research question 1).

A quantile – quantile (Q-Q) plot was undertaken from the data obtained for the first research question (i.e. difference in students' attitude to learning computer programming through playing games between the pre and the post study) in the PGS study in order to observe how the observations are scattered. As shown from Figure 6.26, the Q-Q plot shows a

narrow arched shape where the majority of observations do not embrace the normal Q-Q linear line. The horizontal axis represents observed values gathered from the difference of pre and post study regarding the attitude of participants to learning programming through playing games whereas the vertical axis shows the expected observations in the normal probability plot. The linear line represents a perfect normal distribution and when data comes from a normally distributed population observations hug this linear line. Because the majority of observations do not hug the linear line in Figure 6.26, the Q-Q plot provides evidence that the data came from a non-normally distributed population.

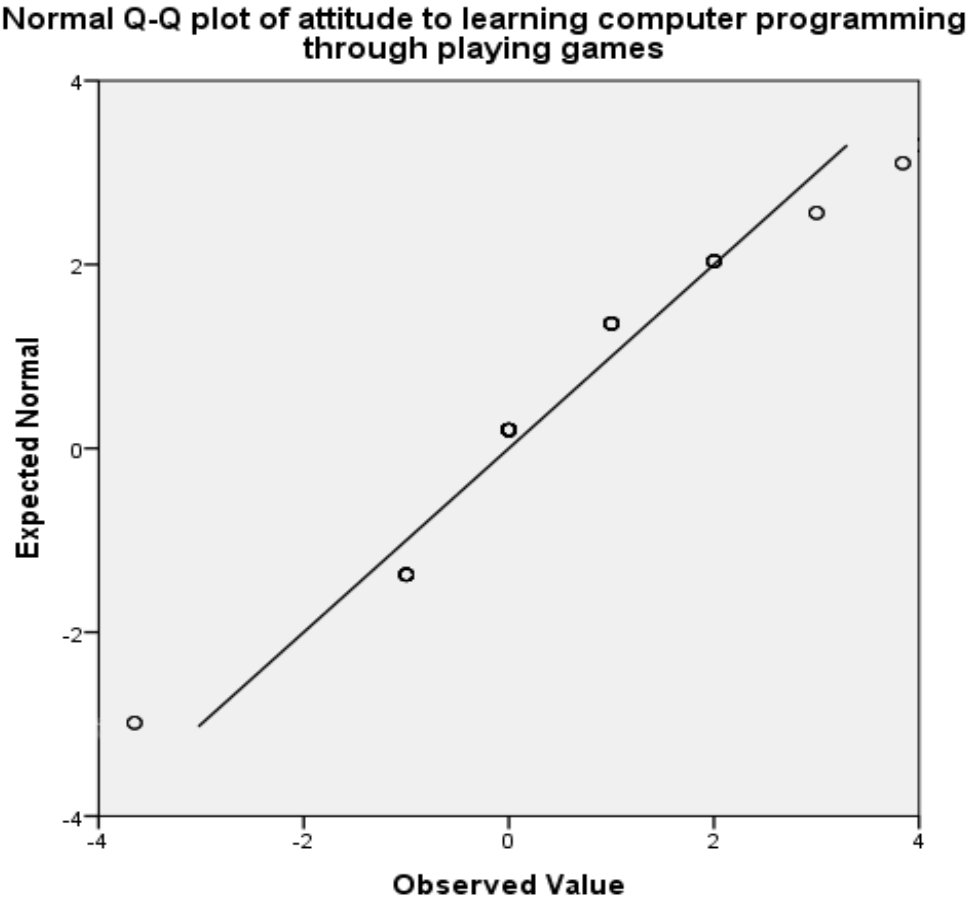

Figure 6.26 – Normal quantile – quantile (Q-Q) plot showing distribution of observations captured on the difference between attitudes to learn computer programming through playing games in the PGS study (Research question 1).

Descriptive Statistics

Skewness & Kurtosis Issues	N	Mean	Std. Deviation	Skewness		Kurtosis	
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Std. Error
Attitude to learning computer programming through playing games	52	**.6346**	**1.6212**	**1.481**	.330	**1.162**	.650
Valid N (listwise)	52						

Table 6.26 – Skewness and Kurtosis normality check on the difference between attitudes to learn computer programming through playing games in the PGS study (Research question 1).

Table 6.26 shows Skewness and Kurtosis normality check regarding the attitude of participants to learning computer programming through playing games between the pre and post study in the PGS study. The absolute values of Skewness (1.48) and Kurtosis (1.16) do not fall in between -1 and 1 range and that they are not close to 0. Additionally, it is expected that in a normal distribution the absolute value of Skewness and Kurtosis should be less than three times their standard error. While the value of Skewness does not satisfy this rule (0.33*3 = 0.99 < 1.48), the value of Kurtosis satisfies (0.65*3 = 1.95 > 1.26). In order to accept that the data came from a normally distributed population, both Skewness and Kurtosis issues must satisfy the conditions. Therefore, it is not possible to accept that data came from a normally distributed population.

Tests of Normality

Normality Check	Kolmogorov-Smirnov		Shapiro-Wilk	
	Statistic	Sig.	Statistic	Sig.
Attitude to learning programming through playing games	.238	.000	.812	**.003**

Table 6.27 – The Shapiro Wilk and the Kolmogorov Smirnov test results on the difference between attitudes to learn computer programming through playing games in the PGS study (Research question 1).

A *Shapiro – Wilk* test was undertaken to measure whether or not this would support the previous findings (i.e. histogram, Q-Q plots and Skewness and Kurtosis check) regarding the normality of data. Before the *Shapiro – Wilk* test was conducted a null hypothesis (i.e. H0 –

the sample population is normally distributed) and an alternative hypothesis to disprove this (i.e. H1 – the sample population is not normally distributed) was created. Should the data be normally distributed; the test value would be greater than 0.05 and the null hypothesis would be accepted. On the other hand, should the data be non-normally distributed, the test value would be less than 0.05 and this time the null hypothesis would be rejected. As shown from Table 6.27, the significant value for the *Shapro – Wilk* test is less than 0.05 (p= 0.03), therefore, the null hypothesis is rejected and the results support the alternative hypothesis. In other words, the *Shapiro – Wilk* test provided strong evidence that the data came from a non-normally distributed population.

The Kolmogorov Smirnov test results were not used as it primarily provides a historical perspective to results and the sample size in the PGS study (N=52) is not large enough to obtain an accurate result from this test. Further to this, the rest of the research questions were analysed in the same way as the first research question (i.e. difference in students' attitude to learn computer programming through playing games between the pre and the post study). All four methods used for identifying the normal distribution (i.e. histogram, Q-Q plots, Skewness and Kurtosis normality check and Shapiro – Wilk test) provided predominantly similar results for all research questions evaluated in the PGS study and in all cases the results show that data captured in the PGS study did not fit a normal distribution. In this case, it was not possible to perform two tailed samples-paired t-test within one group to analyse the datasets. As a result, the Wilcoxon signed rank test, the non-parametric equivalent of two tailed samples-paired t-test, was performed to analyse the results.

6.4.1 Research Question 1 – Is there a difference in pupils' attitude to learn computer programming through playing games between the pre and the post study?

Figure 6.27 demonstrates participants' attitude to learning computer programming through playing games before and after they played *Program Your Robot* in the PGS study. Prior to study, 1 (1.9%) out of 52 pupils strongly agreed and 15 (28.8%) more agreed that a game can be useful for learning how computer programming constructs work through game-play. While 20 (38.4%) pupils were neutral, 11 (21.1%) of them disagreed and 1 (1.9%) more strongly disagreed with same the statement. A total of 9 (17.3%) pupils indicated that they do not know the answer before their game-play. Having played the game, the number of strongly agreed pupils increased from 1 (1.9%) to 3 (5.76%) and the number of agreed pupils increased

from 15 (28.8%) to 16 (30.7%). Despite these improvements, the number of neutral pupils increased from 16 (30.7%) to 20 (38.4%) and the number of disagreed pupils increased from 8 (15.3%) to 11 (21.1%). Additionally, those who strongly disagreed decreased from 3 (5.7%) to 1 (1.9%) and 1 (1.9%) pupil did not provide an answer.

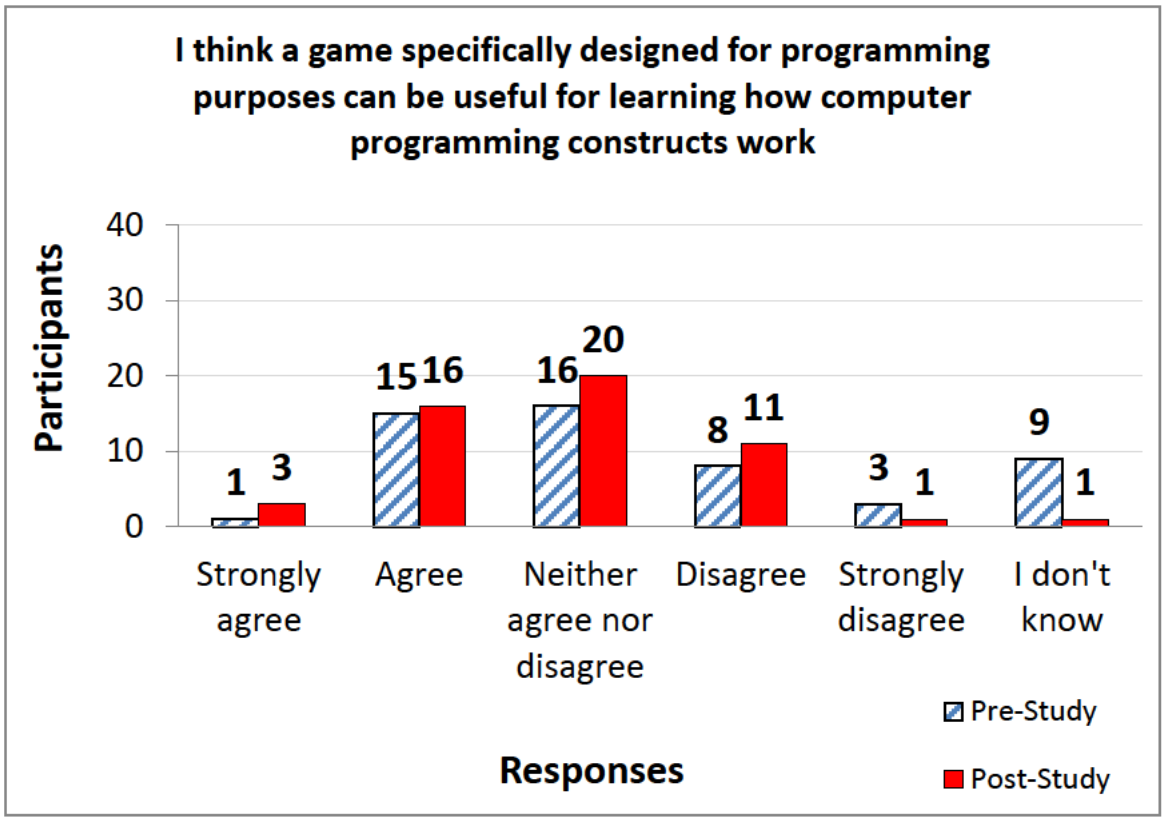

Figure 6.27 – Pupils' attitude to learning computer programming through playing games between the pre and the post study in the PGS study.

The demographic data gathered for the first research question did not provide an accurate outcome whether or not an increase in pupils' attitude towards learning computer programming through game-play happened after they played the game. Therefore, a Wilcoxon signed ranks test and descriptive statistics were performed in order to understand whether or not a significant change happened between the pre and the post study regarding the attitude of participants to learning computer programming through game-play.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre video game attitude to learning programming through game-play	52	**2.54**	1.461	0	5
Post video game attitude to learning programming through game-play	52	**3.12**	1.003	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post video game attitude to learning programming through game-play – Pre video game attitude to learning programming through game-play	Negative Ranks	**13[a]**	**12.42**	161.50
	Positive Ranks	**20[b]**	**19.98**	399.50
	Ties	**19[c]**		
	Total	52		

a. Post video game attitude < Pre Video game attitude

b. Post video game attitude > Pre video game attitude

c. Post video game attitude = Pre video game attitude

Wilcoxon Signed Ranks Test	Post video game attitude to learning programming through game-play – Pre video game attitude to learning programming through game-play
Z	**-1.044[d]**
Asymp. Sig. (2-tailed)	**.296**

d. Based on negative ranks.

Table 6.28 – Descriptive statistics and Wilcoxon signed ranks test results of pupils' attitude to learning computer programming through playing games between the pre and the post study in the PGS study.

As shown in Table 6.28, a Wilcoxon signed ranks test and descriptive statistics were performed to assess whether or not there is a difference in pupils' attitude to learning programming through playing games. The descriptive statistics indicate that there was an increase in pupils' attitude to learning programming through playing games between pre-study (M=2.54; SD=1.4) and post study (M=3.12; SD=1.0) conditions; average rank of 12.42 vs. average rank of 19.98. However, the Wilcoxon signed ranks test results show that the

difference happened in pupils' attitude to learn computer programming through game-play between the pre and the post study is not significant (z=1.044; p=0.29). As the two-tailed significant value is greater than 0.05, the null hypothesis is accepted that is to say there is no significant difference between the attitude of pupils regarding learning computer programming through playing games between the pre and post study. In other words, *Program Your Robot* did not significantly impacted on pupils' perception regarding learning programming through playing games.

6.4.2 Research Question 2 – Is there a difference in pupils' intrinsic motivation to learn computer programming between the pre and the post study?

Figure 6.28 shows pupils' perception about their enjoyment in learning computer programming both in the pre and post study in the PGS study. In the pre-study, none of the participants strongly agreed, or agreed, that they would enjoy learning computer programming. While 8 (15.3%) out of 52 pupils were neutral, a total of 20 (38.4%) pupils strongly disagreed, and disagreed, that they would enjoy learning computer programming. Moreover, 24 (46.1%) pupils did not know what to answer. After playing the game, 1 (1.9%) pupil strongly agreed and 16 (30.7%) more agreed that they would enjoy learning programming. Whist, the neutral pupils were raised from 8 (15.3%) to 21 (40.3%), the number of strongly disagreed, and disagreed, pupils reduced from 20 (37.7%) to 13 (25%). More interestingly, those who did not know the answer in the pre-study decreased from 24 (46.1%) to only 1 (1.9%) in the post-study.

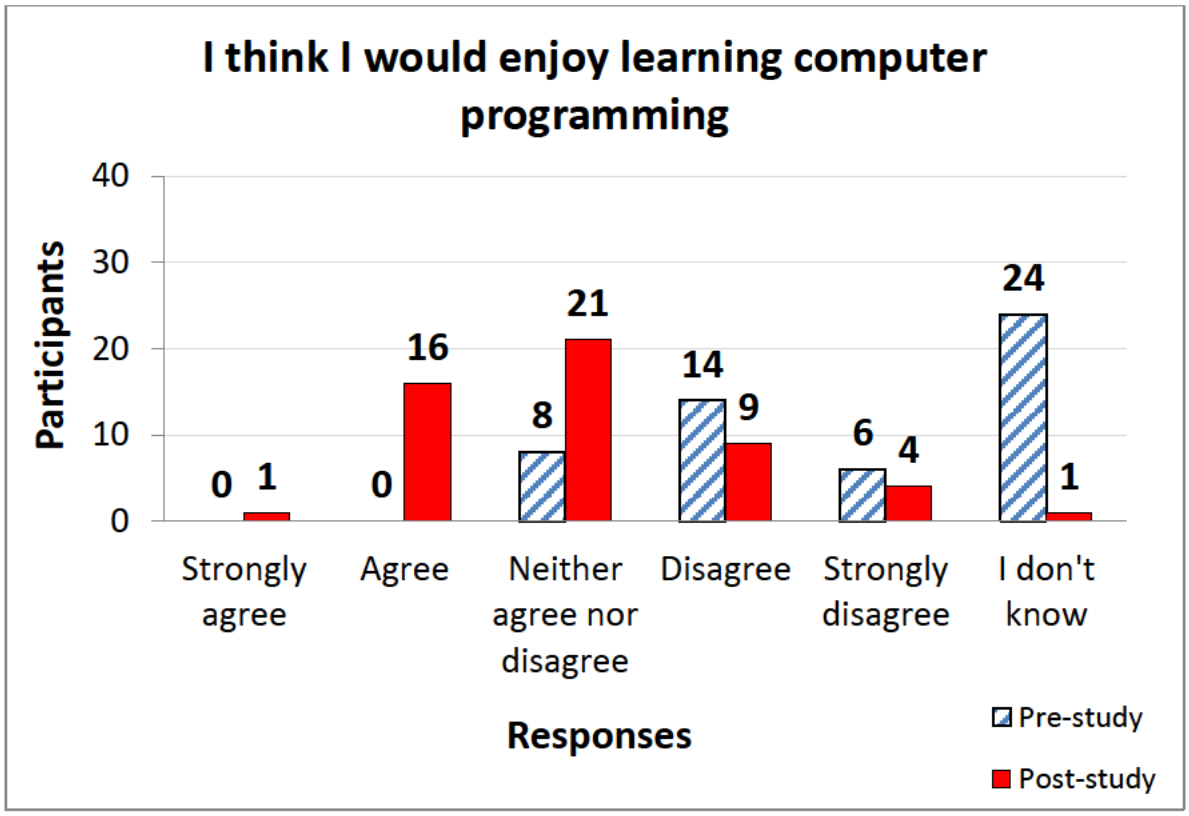

Figure 6.28 – Pupils' perception about their enjoyment in learning computer programming between the pre and the post study in the PGS study.

A Wilcoxon signed ranks test and descriptive statistics were undertaken to investigate whether or not a significant increase happened in the pupils' perception of their enjoyment in learning computer programming between pre and post study. As illustrated in Table 6.29, there is an increase in the pupils' perception of their enjoyment in learning computer programming between pre (M= 1.12; SD=1.16) and post (M=2.96; SD=1.02) conditions in the PGS study; average rank 8.50 vs average rank 23.83. Moreover, the Wilcoxon signed ranks test results show that this increase is significant (z= 5.871; $p<0.05$). As the two tailed significant value is less than 0.05 (p=0.000), the null hypothesis that indicates the groups' perception of their enjoyment in learning computer does not differ is rejected. In this case, the Wilcoxon signed ranks test results support the alternative hypothesis that is to say pupils' perception of their enjoyment in learning computer programming is significantly increased between the pre and the post study in the PGS study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre Enjoy Programming	52	**1.12**	1.166	0	3
Post Enjoy Programming	52	**2.96**	1.028	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post Enjoy Programming – Pre Enjoy Programming	Negative Ranks	**1[a]**	**8.50**	8.50
	Positive Ranks	**45[b]**	**23.83**	1072.50
	Ties	**6[c]**		
	Total	52		

a. Post Enjoy Programming < Pre Enjoy Programming

b. Post Enjoy Programming > Pre Enjoy Programming

c. Post Enjoy Programming = Pre Enjoy Programming

Wilcoxon Signed Ranks Test	Post Enjoy Programming - Pre Enjoy Programming
Z	**-5.871[d]**
Asymp. Sig. (2-tailed)	**.000**

d. Based on negative ranks.

Table 6.29 – Descriptive statistics and Wilcoxon signed ranks test of pupils' perception about their enjoyment in learning computer programming between the pre and the post study in the PGS study.

The result obtained from the second research question of the PGS study is very interesting especially when this is compared to the results of the first research question. According to the statistical outcomes obtained, pupils' enjoyment towards learning computer programming is increased after they played the game but their attitude to learning computer programming through game-play did not change. This means that pupils felt that they would enjoy more learning of computer programming after they played *Program Your Robot* which proves that their intrinsic motivation to learn computer programming in increased. However, their attitude towards learning computer programming through game-play did not change which means that they felt learning computer programming could have been presented in a better game-play than *Program Your Robot*. It is suggested that participants' attitude regarding learning through game-play did not change because of the major technical difficulties encountered in their first trial and their first impression of the game affected their perception of it. Nevertheless, it is important to highlight that none of the pupils strongly agreed or agreed that they would enjoy learning computer programming before they

participated in the study whereas this increased to 17 (32.6%) after their participation. More importantly, the Wilcoxon signed rank test results provide statistical evidence that pupils' enjoyment of learning computer programming is significantly increased after they played the game.

6.4.3 Research Question 3,4,5,6 – Is there a difference in pupils' perception of their knowledge in programming sequence, methods, decision making and loops between the pre and the post study?

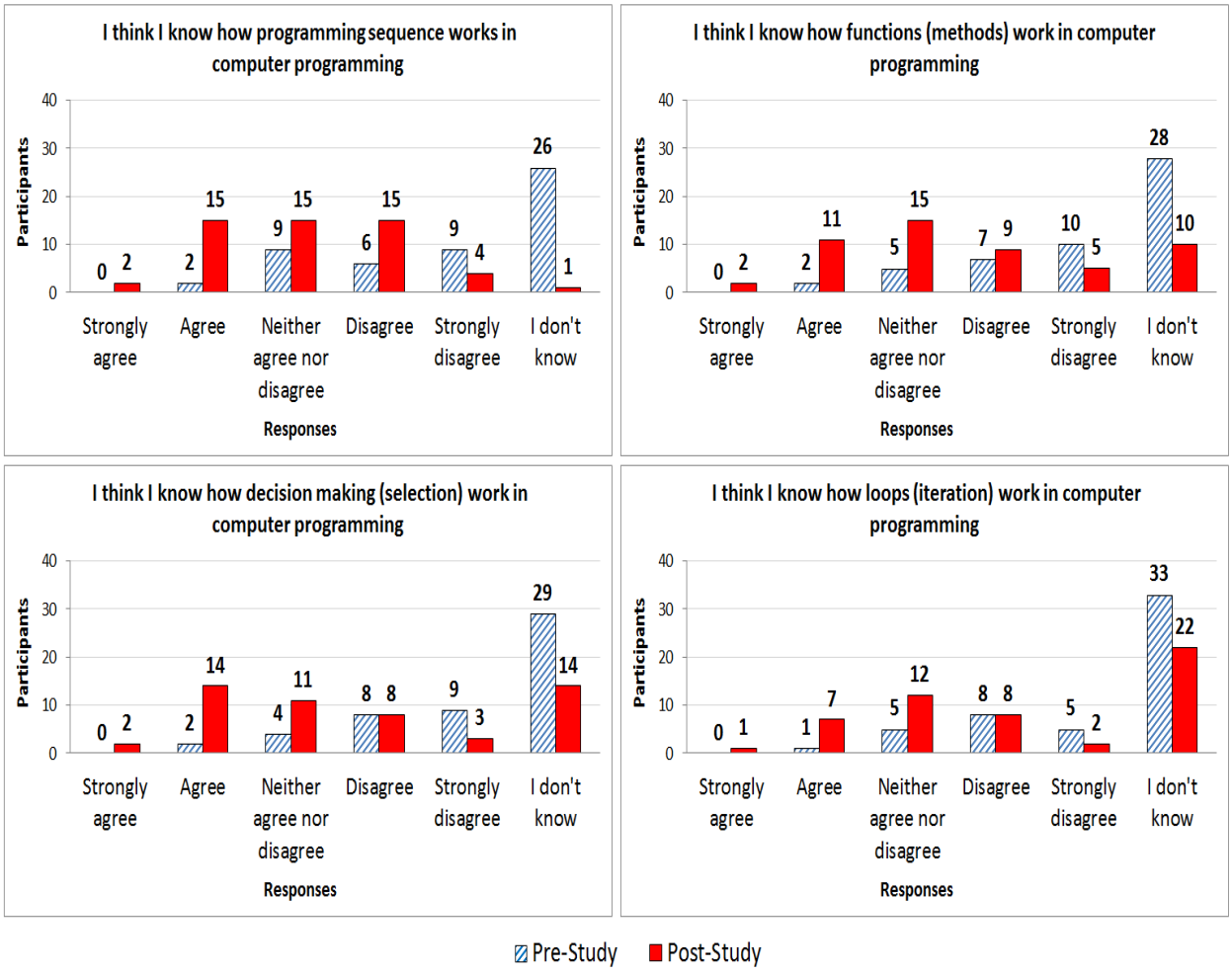

Figure 6.29 – Pupils' perception of their knowledge on programming constructs between the pre and the post study in the PGS study

Figure 6.29 shows the responses of pupils given to four programming constructs introduced in the game (i.e. programming sequence, functions, decision making and loops) during the PGS study. In the pre-study, none of the pupils strongly agreed and only 2 (3.8%)

out of 52 pupils agreed that they knew programming sequence. While 9 (17.3%) pupils were neutral, a total of 15 (28.8%) people strongly disagreed and disagreed that they knew how programming sequence works. In addition to these, 29 (55.7%) pupils did not provide an answer on whether or not they knew how programming sequence works. In the post-study, the number of strongly agreed and agreed pupils increased from 2 (3.8%) to 17 (32.6%). Despite this improvement, the number of neutral pupils increased from 9 (17.3%) to 15 (28.8%) and consequently, those pupils who strongly disagreed, and disagreed, also increased from 15 (28.8%) to 19 (36.5%). Additionally, those who did not answer decreased from 26 (50%) to 1 (1.9%) after participants played the game.

A similar difference was observed in functions as the number of pupils who strongly agreed that they know functions increased from none to 2 (3.8%) and similarly those who agreed increased from 2 (3.8%) to 11 (21.1%). Whilst, the number of neutral pupils increased from 5 (9.6%) to 15 (28.8%), those who strongly disagreed, and disagreed, decreased from 17 (32.6%) to 14 (26.9%). Moreover, those who did not answer the question decreased from 28 (53.8%) to 10 (1.9%) after the game-play.

Prior to the study, only 2 (3.8%) pupils agreed, and none of them strongly agreed, that they knew decision making. Having played the game, those who strongly agreed and agreed increased to 16 (30.7%). While neutral pupils increased from 4 (7.69%) to 11 (21.1%) during the study, those who strongly disagreed and disagreed decreased from 17 (32.6%) to 11 (21.1%). Additionally, those who did not answer the question also decreased from 29 (55.7) to 14 (26.9%).

In the pre-study, none of the pupils strongly agreed and only one (1.9%) pupil agreed that she knew loops before participating in the study. Having played the game, the number of strongly agreed and agreed pupils increased from 1 (1.9%) to 8 (15.3%). Those who were neutral also increased from 5 (9.6%) to 12 (23%). The number of disagreed pupils stayed the same and those who strongly disagreed decreased from 5 (9.6%) to 2 (3.8%). Finally, pupils who did not answer whether or not they know how loops work decreased from 33 (63.4%) to 22 (42.3%).

The demographic data collected regarding the all programming constructs (i.e. programming sequence, functions, decision making and loops) presented in the game shows that participants felt an increase in their knowledge after they played the game. To investigate whether or not this increase is significant, a Wilcoxon signed ranks test and descriptive statistics were undertaken. Table 6.30 shows descriptive statistics and mean differences between the pre and the post study regarding all programming constructs introduced in the

game (i.e. programming sequence, functions, decision making and loops). The findings show that, in each case the post population mean was higher than the pre population mean which provides evidence that pupils felt their knowledge was improved during the study regarding all programming constructs.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre Sequence Knowledge	52	**1.08**	1.296	0	4
Pre Functions Knowledge	52	**.90**	1.192	0	4
Pre Decision Knowledge	52	**.87**	1.172	0	4
Pre Loop Knowledge	52	**.77**	1.148	0	4
Post Sequence Knowledge	52	**2.87**	1.103	0	5
Post Functions Knowledge	52	**2.35**	1.507	0	5
Post Decision Knowledge	52	**2.27**	1.658	0	5
Post Loop Knowledge	52	**1.67**	1.618	0	5

Table 6.30 – Descriptive statistics of pupils' perception of their knowledge on programming constructs in the pre and post study of PGS study.

A Wilcoxon signed ranks test was undertaken to evaluate whether or not the increase observed in descriptive statistics regarding pupils' perception of their knowledge on programming constructs is significant. The findings show that there was an increase in pupils' perception of their knowledge in programming sequence (*M*=1.08; *SD*=1.29 in the pre-study; *M*=2.87; *SD*=1.10), functions (*M*=0.90; *SD*=1.19 in the pre-study; *M*=2.35; *SD*=1.50 in the post-study), decision making (*M*=0.87; *SD*=1.17 in the pre-study; *M*=2.27; *SD*=1.65 in the post-study) and loops (M=0.77; SD=1.14 in the pre-study; M=1.67; SD=1.61 in the post-study) between the pre-study and post-study conditions. As can be observed from Table 6.31, the Wilcoxon signed ranks test show that the increase happened in pupils' perception of their knowledge regarding programming sequence (z=5.766; p=0.000), functions (z=4.890; p=0.000), decision making (z=4.963; p=0.000) and loops (z=3.974; p=0.000) is significant. As the 2-tailed significant value was less than 0.05 (p=0.000) in all cases, the null hypotheses that indicate pupils' perception of their knowledge regarding programming sequence, functions, decision making and loops does not change between the pre and the post study is rejected. In other words, the Wilcoxon signed ranks test results provide strong evidence to support the alternative hypotheses which are pupils' perception of their knowledge regarding all programming constructs is significantly increased between the pre and the post study during the Wilcoxon study.

Ranks		N	Mean Rank	Sum of Ranks
Post Sequence Knowledge – Pre Sequence Knowledge	Negative Ranks	4[a]	8.00	32.00
	Positive Ranks	44[b]	26.00	1144.00
	Ties	4[c]		
	Total	52		
Post Functions Knowledge – Pre Functions Knowledge	Negative Ranks	6[d]	10.75	64.50
	Positive Ranks	36[e]	23.29	838.50
	Ties	10[f]		
	Total	52		
Post Decision Knowledge – Pre Decision Knowledge	Negative Ranks	4[g]	8.00	32.00
	Positive Ranks	34[h]	20.85	709.00
	Ties	14[i]		
	Total	52		
Post Loop Knowledge – Pre Loop Knowledge	Negative Ranks	3[j]	7.00	21.00
	Positive Ranks	23[k]	14.35	330.00
	Ties	26[l]		
	Total	52		

a. Post Sequence Knowledge < Pre Sequence Knowledge

b. Post Sequence Knowledge > Pre Sequence Knowledge

c. Post Sequence Knowledge = Pre Sequence Knowledge

d. Post Functions Knowledge < Pre Functions Knowledge

e. Post Functions Knowledge > Pre Functions Knowledge

f. Post Functions Knowledge = Pre Functions Knowledge

g. Post Decision Knowledge < Pre Decision Knowledge

h. Post Decision Knowledge > Pre Decision Knowledge

i. Post Decision Knowledge = Pre Decision Knowledge

j. Post Loop Knowledge < Pre Loop Knowledge

k. Post Loop Knowledge > Pre Loop Knowledge

l. Post Loop Knowledge = Pre Loop Knowledge

Wilcoxon Signed Ranks Test	Post Sequence Knowledge – Pre Sequence Knowledge	Post Functions Knowledge – Pre Functions Knowledge	Post Decision Knowledge – Pre Decision Knowledge	Post Loop Knowledge – Pre Loop Knowledge
Z	-5.766[m]	-4.890[m]	-4.963[m]	-3.974[m]
Asymp. Sig. (2-tailed)	.000	.000	.000	.000

m. Based on negative ranks

Table 6.31 – Wilcoxon signed ranks test results of pupils' perception of their knowledge on programming constructs in the pre and post study of PGS study.

The most exciting outcome of the PGS study was while more than the half of participants indicated that they did not have previous knowledge regarding any programming constructs in the pre-study, this number decreased considerably in the post-study. It is crucial to highlight that the PGS study encouraged their participants to make up their own minds and decide whether or not they learned how computer programming constructs work from the game environment as initially the majority of them did not know what to answer to questions whereas after playing the game they did. Further to this, none of the participants strongly agreed that they knew any of the programming constructs before the study was conducted whereas this increased in between 1.9% – 3.8% (1 to 2 participants) in the post-study.

The findings of the PGS study regarding participants' perception of computer programming constructs are encouraging and it is anticipated that these results would have been better than this if the technical difficulties explained earlier had never happened. The raw data collected from the study and the descriptive statistical analysis of this provide evidence that there was an increase in pupils' perception of their knowledge regarding all programming constructs after they played the game. More importantly, the Wilcoxon signed ranks test delivers strong evidence that this increase is statistically significant which means that should the study be repeated under the same circumstances with the same experimental structure a very similar outcome would be obtained.

6.4.4 Research Question 7, 8 – Is there a difference in pupils' problem solving abilities and the ability to visualise programming constructs from given problems between the pre and the post study?

As illustrated in Figure 6.30, only 4 (7.69%) out of 52 participants strongly agreed and agreed that they have problem solving abilities required to learn computer programming in the pre-study. Having played the game, this number is increased to a total number of 23 (44.2%). In addition to this, those who were neutral increased from 7 (13.4%) to 20 (38.4%) during the study and the total number of pupils who strongly disagreed, and disagreed, that they have problem solving abilities, decreased from 19 (36.5%) to 8 (15.3%) in the post-study. Finally, those who did not know the answer decreased from 22 (42.3%) to 1 (1.92%) after the study was conducted.

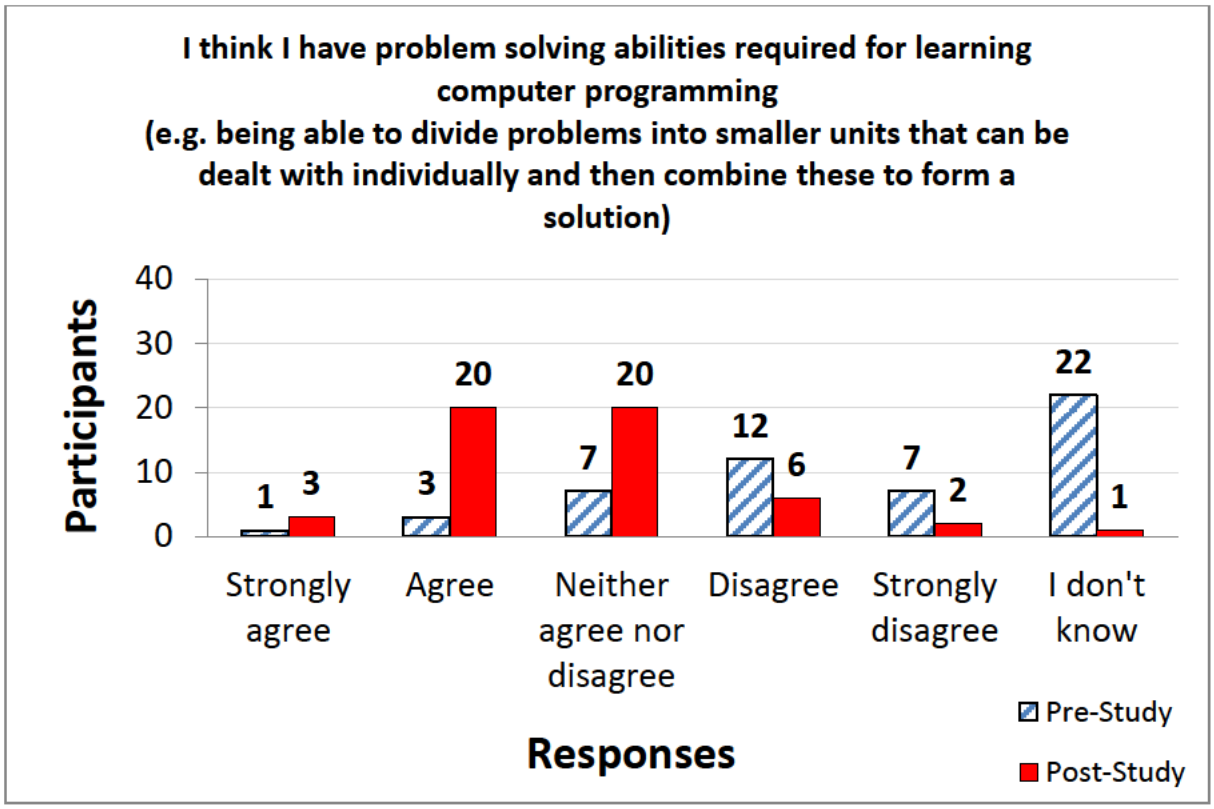

Figure 6.30 – Pupils' perception of their problem solving abilities between the pre and the post study in the PGS study

A Wilcoxon signed ranks test was performed with descriptive statistics in order to assess whether or not there was a significant increase in pupils' perception of their problem solving abilities during the study. As shown from Table 6.32, descriptive statistics revealed that there is an increase in pupils' perception of their problem solving abilities for learning computer programming between pre (M=1.33; SD=1.39) and post (M=3.25; SD=1.00) study conditions in the PGS study; average rank of 7.50 vs. average rank of 22.85. More importantly, Wilcoxon signed ranks test results show that the increase happened in pupils' perception of their problem solving abilities between the pre and post study is significant (z=5.751; p <0.05). As the 2-tailed significant value is less than 0.05 (p=0.000), the null hypothesis that indicates pupils' perception of their problem solving abilities to learn computer programming does not differ is rejected. In this case, Wilcoxon signed ranks test result support the alternative hypothesis that is to say pupils' perception of their problem solving abilities to learn computer programming is significantly increased between the pre and the post study during the PGS study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre Problem Solving Abilities	52	**1.33**	1.396	0	5
Post Problem Solving Abilities	52	**3.25**	1.007	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post Problem Solving Abilities – Pre Problem Solving Abilities	Negative Ranks	**1**[a]	**7.50**	7.50
	Positive Ranks	**43**[b]	**22.85**	982.50
	Ties	**8**[c]		
	Total	52		

a. Post Problem Solving Abilities < Pre Problem Solving Abilities

b. Post Problem Solving Abilities > Pre Problem Solving Abilities

c. Post Problem Solving Abilities = Pre Problem Solving Abilities

Wilcoxon Signed Ranks Test	Post Problem Solving Abilities – Pre Problem Solving Abilities
Z	**-5.751**[d]
Asymp. Sig. (2-tailed)	**.000**

d. Based on negative ranks

Table 6.32 – Descriptive statistics and Wilcoxon signed ranks test results of pupils' perception of their problem solving abilities in the pre and the post study of the PGS study

Prior to the PGS study, the majority of pupils indicated that they were unable to visualise programming constructs from given problems despite the fact that the programming constructs in the game (i.e. programming sequence, functions, decision making and loops) were introduced to them by their ICT teacher before they participated in the study. After playing the game, the number of pupils who strongly agreed and agreed that they can visualise programming constructs from given problems are increased from 3 (5.7%) to 20 (3.8%). While pupils who were neutral increased from 7 (13.4%) to 12 (23%) during the study, those who strongly disagreed, and disagreed, decreased from 17 (32.6%) to 9 (17.3%). Finally, the number of participants who did not know the answer decreased from 25 (48%) to 11 (21.1%) between the pre and the post study conditions.

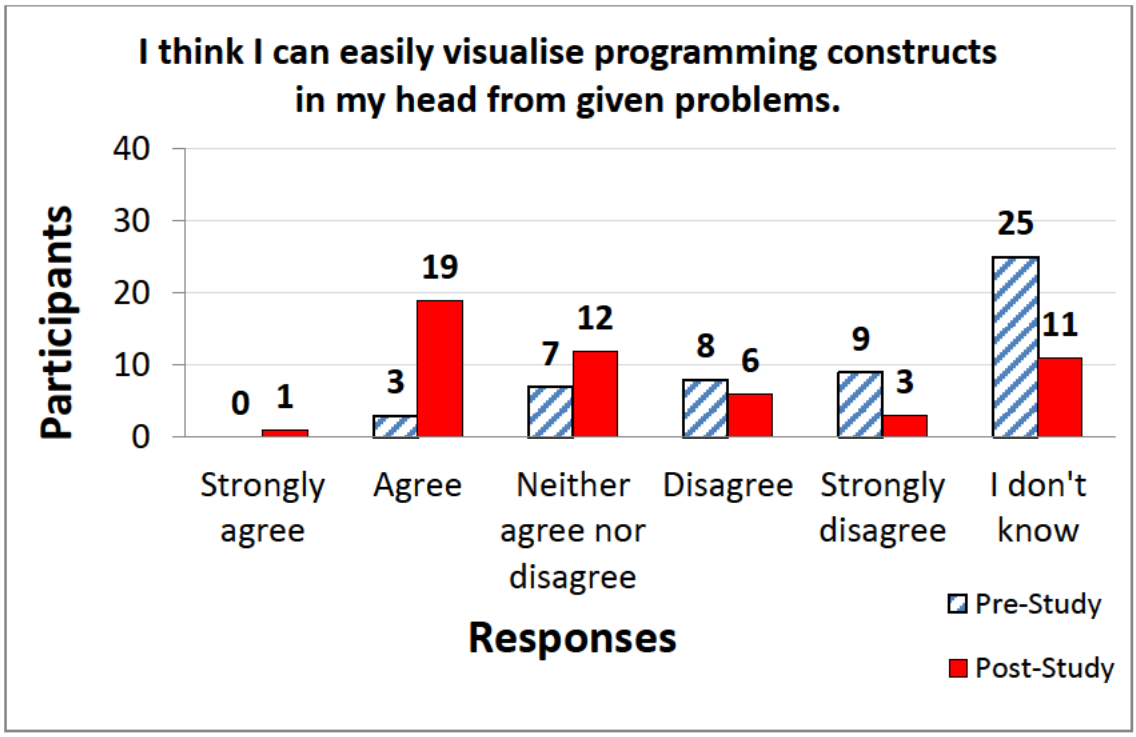

Figure 6.31 – Students' perception of their ability to visualise programming constructs from given problems between the pre and the post study in the PGS study

Table 6.33 illustrates descriptive statistics and Wilcoxon signed rank test results of pupils' perception of their ability to visualise programming constructs from given problems between the pre and the post study in the PGS study. As shown from the results, there was an increase in pupils' perception of their ability to visualise programming constructs from given problems between pre (M=1.12, SD=1.30) and post (M=2.54, SD=1.59) study conditions in the PGS study; average rank of 13.5 vs. average rank of 22.5. To investigate whether or not this increase is significant a Wilcoxon signed ranks test was conducted. The results of this test show that the increase happened in the pupils' perception of their ability to visualise programming constructs from given problems is significant ($z=4.391$; $p<0.05$). As the 2-tailed significant value is less than 0.05 ($p=0.000$), the null hypothesis which defends pupils' perception of their ability to visualise programming constructs from given problems does not differ, is rejected. Therefore, the Wilcoxon signed ranks test results support the alternative hypothesis that is to say pupils' perception of their ability to visualise programming constructs from given problems is significantly increased between the pre and the post study during the PGS study.

Descriptive Statistics	N	Mean	Std. Deviation	Minimum	Maximum
Pre visualising prog. constructs	52	**1.12**	1.308	0	4
Post visualising prog. constructs	52	**2.54**	1.590	0	5

Ranks		N	Mean Rank	Sum of Ranks
Post visualising prog. constructs – Pre visualising prog. constructs	Negative Ranks	**7[a]**	**13.50**	94.50
	Positive Ranks	**34[b]**	**22.54**	766.50
	Ties	**11[c]**		
	Total	52		

a. Post visualising prog. constructs < Pre visualising prog. constructs

b. Post visualising prog. constructs > Pre visualising prog. constructs

c. Post visualising prog. constructs = Pre visualising prog. constructs

Wilcoxon Signed Ranks Test	Post visualising prog. constructs – Pre visualising prog. constructs
Z	**-4.391[d]**
Asymp. Sig. (2-tailed)	**.000**

d. Based on negative ranks

Table 6.33 – Descriptive statistics and Wilcoxon signed ranks test results of pupils' perception of their ability to visualise programming constructs from given problems in pre and post study of the PGS study.

6.4.5 Summary of findings regarding research questions

A summary of the statistical outcome of all research question evaluated in the PGS study is presented in Table 6.34. As illustrated in the table, the most significant difference between the pre and the post study conditions happened in pupils' perception of their problem solving abilities (*mean difference* = 1.92) whereas the least significant difference happened in pupils' perception of their knowledge in loops (*mean difference* = 0.9).

Research Question#	Pairs	Mean Difference (*M*)	Two tailed significant value (p)
1	Pre video game attitude to learning programming through game-play – Post video game attitude to learning programming through game-play	**0.58**	**0.296**
2	Pre computer programming motivation – Post computer programming motivation	**1.84**	**0.000**
3	Pre sequence knowledge – Post Sequence Knowledge	**1.79**	**0.000**
4	Pre functions knowledge – Post functions knowledge	**1.45**	**0.000**
5	Pre decision making knowledge – Post decision knowledge	**1.4**	**0.000**
6	Pre loop knowledge – Post loop knowledge	**0.9**	**0.000**
7	Pre problem solving abilities – Post problem solving abilities	**1.92**	**0.000**
8	Pre visualising constructs – Post visualising constructs	**1.42**	**0.000**

Table 6.34 – Summary of Wilcoxon signed ranks test results of research questions evaluated in the PGS study

Although it is not possible to match the results of the PGS study with the results of Cyprus or Greenwich study due to the differences in the target groups, the statistical analysis of all three studies provided positive and similar outcomes. One could argue that participants' positive feedback in the PGS study was not because of an enhancement of skills/knowledge but simply because participants were asked to play a game rather than their regular lessons. However, there are two important pieces of evidence that firmly proves this is not the case.

Firstly, the PGS study was not conducted during pupils' ICT lessons. The whole study was conducted as an after school activity. Secondly, the result of the first research question (i.e. difference in pupils' attitude to learn computer programming through playing games between the pre and the post study) proves that pupils did not find *Program Your Robot* that enjoyable as the game did not statistically increase the pupils' attitude towards learning programming through playing games. In other words, pupils did not like *Program Your Robot* to an extent that it would change their attitude towards learning computer programming through playing games. This may be an effect of the technical difficulties encountered in managing the study. Nevertheless, this outcome provides strong reasons to believe that participants did not enjoy the game sufficiently enough that would encourage them to learn

more from the game environment. Despite this, the statistical results clearly show that pupils felt that their knowledge regarding four programming constructs, their problem solving abilities and the ability to visualise constructs improved after their game-play.

6.4.6 Statistical Correlations

Figure 6.32 shows the perception of pupils on how well they think the fundamental skills of computational thinking (i.e. conditional logic, algorithmic thinking, simulation, debugging and cooperation) were integrated into or encouraged by *Program Your Robot*. According to the responses collected, the majority of pupils strongly agreed and agreed that three out of five computational thinking skills (i.e. conditional logic, algorithmic thinking and simulating solutions) are well grounded in and encouraged by the game. The data obtained shows that 30 (57.6%) out of 52 pupils strongly agreed and agreed that the game requires thinking logically and evaluating conditions. Additionally, 20 (38.4%) out of 52 pupils strongly agreed and agreed that the game enhanced (or has the potential to enhance) their ability to think algorithmically. Further to these, 24 (46.1%) out of 52 pupils strongly agreed and agreed that the run-time in game simulates how computer algorithms work. In contrast to this, none of the pupils strongly agreed that the debug mode in the game was useful to detect errors or that sharing ideas or strategies during their game-play was useful to them in order to develop their solutions better. While, only 8 (15.3%) pupils agreed that the debug mode was useful to detect errors, 14 (27%) pupils agreed that sharing ideas or strategies helped them develop their solutions during the game-play.

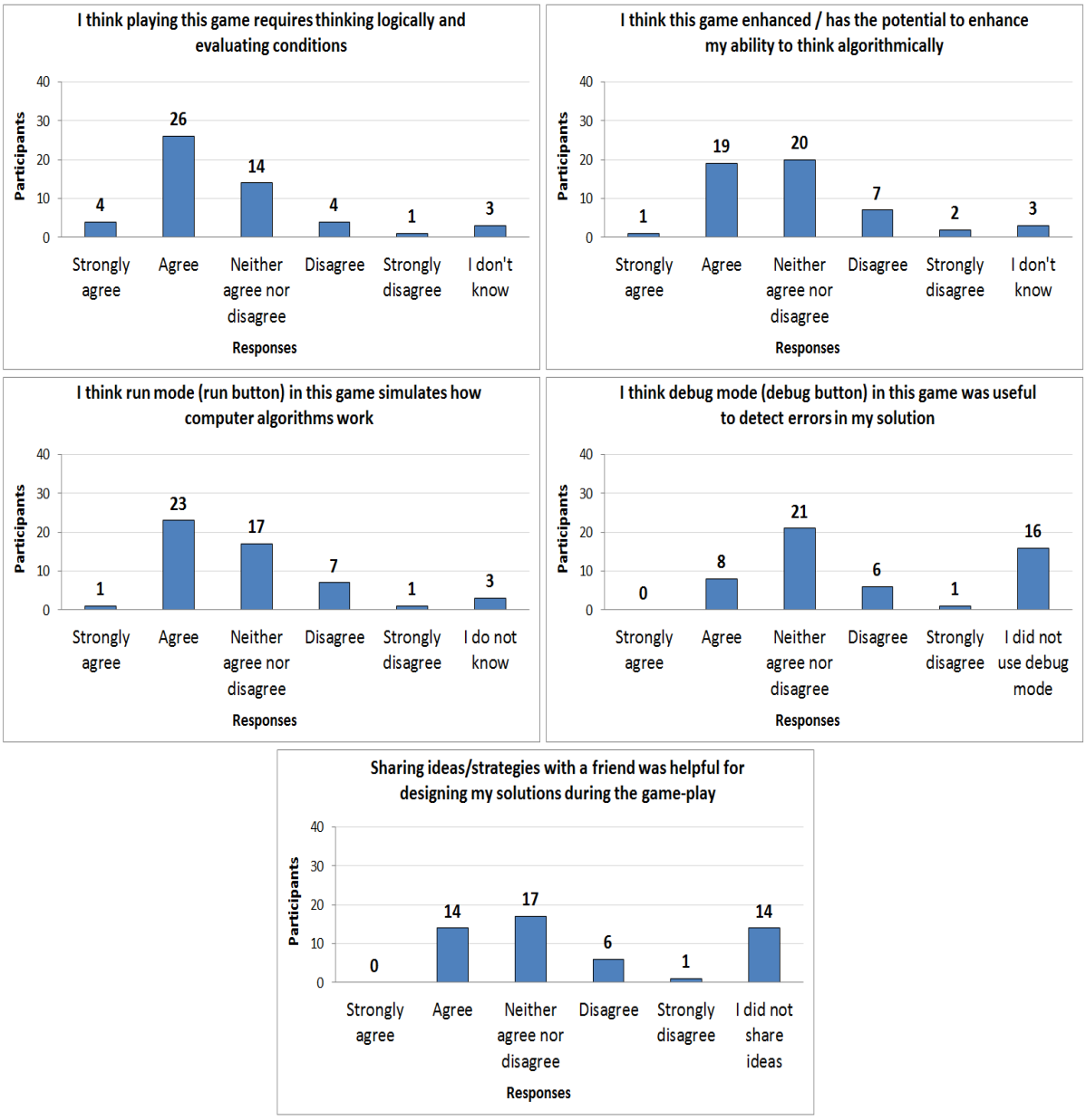

Figure 6.32 – Students' perception of how well computational skills were presented in the game

The raw data collected from the PGS study shows that 14 (27%) out of 52 pupils did not share ideas and 16 (30.7%) out of 52 pupils did not use the debug mode at all in the game. Although these results cannot be directly linked or compared to earlier studies due to the differences in target groups, it is clear that pupils in the PGS study demonstrated a similar behaviour to students who participated in the Greenwich study. In both studies, considerable percentage of participants did not share ideas (between 27% - 38%) and did not use the debug mode (30% - 38%) in the game. These results supports the previous arguments raised in the

Greenwich study that is Cyprus study had a better *situated learning* environment than both the PGS and the Greenwich studies. Further to this, it is anticipated that pupils in the PGS study did not fully understand the debug mode in the game as this concept was not introduced to them before they played the game. Although it is not possible to identify the exact reasons why this had happened, similarities between the Greenwich and the PGS studies lead to the assumption that participants did not use debugging because they were unaware of this concept prior to their game-play.

As illustrated in Table 6.35, a Spearman's rank correlation was undertaken to assess the correlations among computational thinking skills as well as how these skills are correlated to the maximum level players reached in the game in the PGS study. The results are consistent with previous findings particularly with the outcomes of the Greenwich study as the statistical method used for analysis was identical in both studies. The correlations among all computational thinking skills are positive where some of these are significant ($p<0.05$), modestly strong ($0.49 < r <= 0.69$) and strong ($r>=0.7$), others are not.

No strong and significant relationship was identified between achieving high levels in the game and any of the computational thinking skills. This means that a number of participants reached higher levels in the game and felt that they developed abilities in computational thinking. However, those who did not achieve high levels in the game also felt that they developed computational thinking abilities particularly in *conditional logic*, *algorithmic thinking* and *simulating solutions*. As a result, developing computational thinking skills has no strong correlation with achieving high levels in the game. In other words, participants felt that they developed their skills in computational thinking even in early levels of the game.

Spearman's rank correlation coefficient		Maximum game level participants reached	Conditional Logic and Evaluating Conditions	Algorithmic Thinking	Running and Simulating Solutions	Debugging and Handling Errors	Cooperation (Sharing ideas and strategies)
Maximum game level participants reached	Correlation Coefficient	1.000	.331*	.361**	.397**	.190	.293*
	Sig. (2-tailed)	.	.017	.001	.004	.177	.035
	N	52	52	52	52	52	52
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.331*	1.000	**.644****	**.667****	.044	.279*
	Sig. (2-tailed)	.017	.	.000	.000	.756	.045
	N	52	52	52	52	52	52
Algorithmic Thinking	Correlation Coefficient	.361**	**.644****	1.000	**.730****	.108	.355**
	Sig. (2-tailed)	.001	.000	.	.000	.446	.001
	N	52	52	52	52	52	52
Running and Simulating Solutions	Correlation Coefficient	.397**	**.667****	**.730****	1.000	.300*	.370**
	Sig. (2-tailed)	.004	.000	.000	.	.031	.007
	N	52	52	52	52	52	52
Debugging and Handling Errors	Correlation Coefficient	.190	.044	.108	.300*	1.000	**.495****
	Sig. (2-tailed)	.177	.756	.446	.031	.	.005
	N	52	52	52	52	52	52
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.293**	.279*	.355**	.370**	**.495****	1.000
	Sig. (2-tailed)	0.35	0.45	.001	.007	.005	.
	N	52	52	52	52	52	52

Table 6.35 – Spearman's rank correlation coefficient showing relationships among computational thinking skills and also between these skills and the maximum game level students achieved

According to the responses collected in the PGS study, a significant, positive and strong correlation was identified between algorithmic thinking and simulating solutions in the game (r=0.73; n=52; p=0.000). Additionally, a significant, positive and modestly strong correlation was identified between a) conditional logic and algorithmic thinking (r=0.644; n=52; p=0.000) b) conditional logic and simulating solutions (r=0.667; n=52; p=0.000) b) cooperation and debugging (r=0.495; n=52; p=0.005). These associations are consistent with the correlations previously identified in the Cyprus and the Greenwich studies. In this case, it is possible to conclude that an increase in algorithmic thinking also causes an increase in simulating solutions. Correspondingly, an increase in conditional logic also causes an increase in algorithmic thinking and as a result of this an increase in simulating solutions. Therefore, when players use conditional logic in the game, they also develop abilities in algorithmic thinking and simulating solutions.

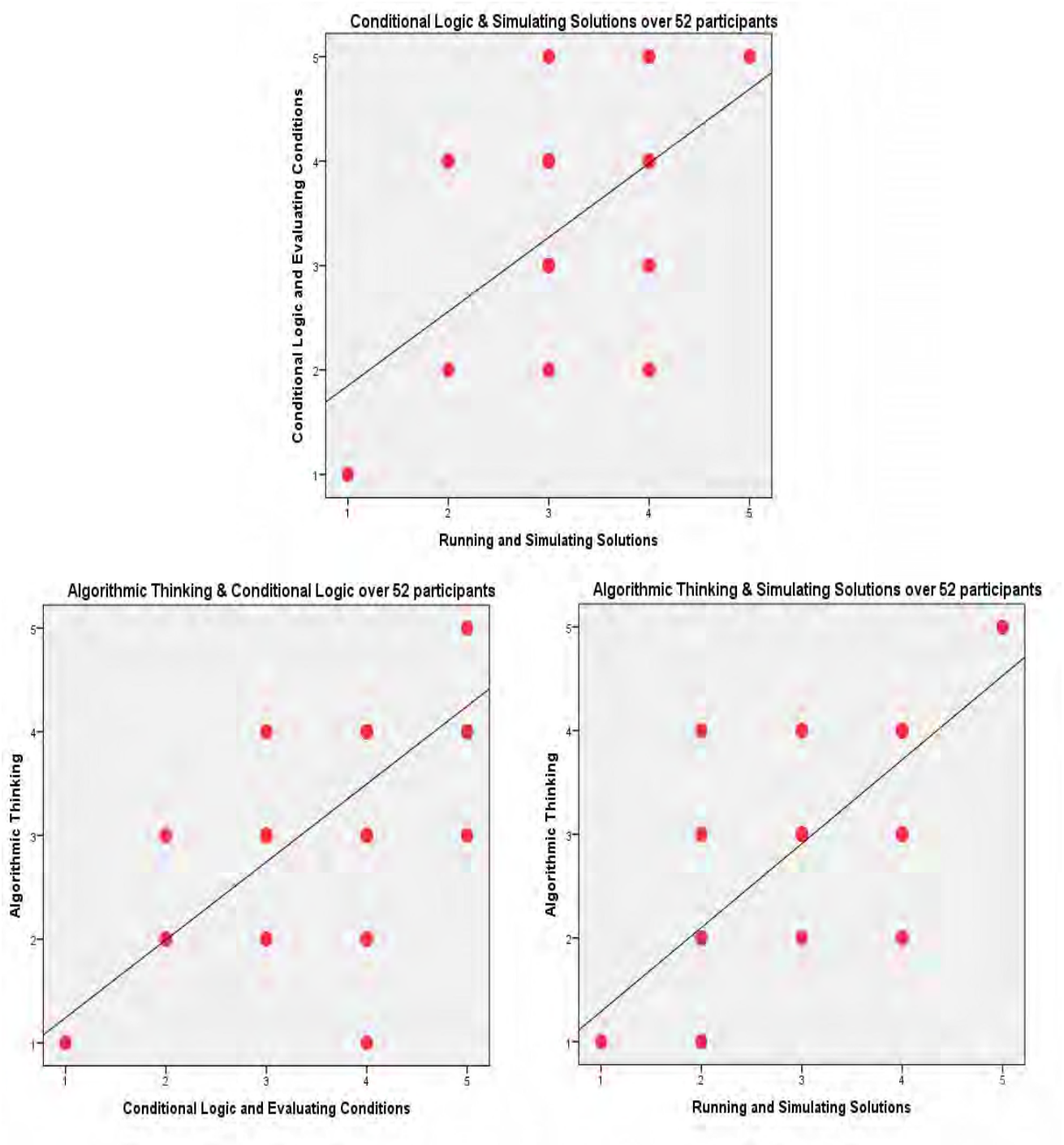

Figure 6.33 – Scatterplots showing strong correlation between algorithmic thinking and simulating solutions and modestly strong correlations between conditional logic and algorithmic thinking and between conditional logic and simulating solutions.

Figure 6.33 shows scatterplots where strong and modestly strong significant relationships are identified among computational thinking skills. The Spearman's rank correlation results show that the only strong, positive and significant relationship was in between algorithmic thinking and simulating solutions. The scatterplots illustrated in Figure 6.33 shows that the

strong association between algorithmic thinking and simulating solution is linear meaning the relationship between them is non-monotonic (there is a direct relation between the variables). Additionally, the scatterplots revealed that the modestly strong relationship between algorithmic thinking and simulating solutions, and the modestly strong relationship between conditional logic and simulation solutions is also linear. This means that as the observation values in one skill increase, the observed values in the other skill also increase.

As the scatterplot generated a non-linear shape for the distribution of observations, it is possibly to assume that the strong and the modestly strong relationships identified in Table 6.35 (with the exception of debugging and cooperation) are relatively linear. Despite this, it is important to highlight that the only strong, linear and significant correlation is between algorithmic thinking and simulating solutions (r=0.73; n=52; p=0.000) as the other correlations are modestly-strong at best.

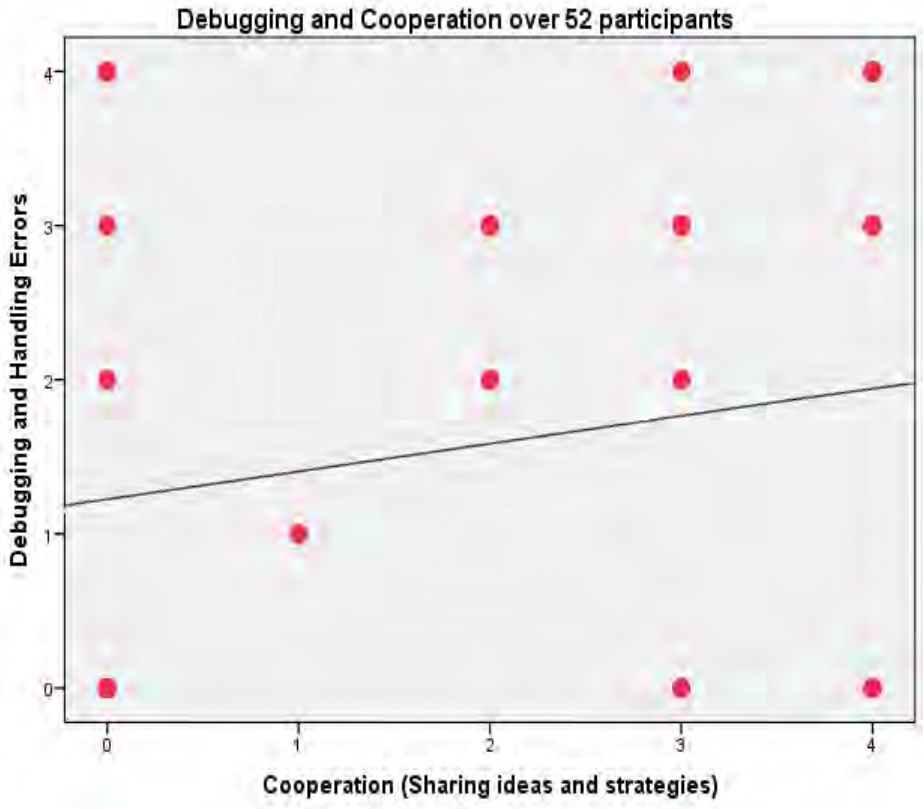

Figure 6.34 – Scatterplots showing modestly strong correlation between debugging and cooperation (sharing ideas and strategies).

Figure 6.34 shows the scatterplot generated from the association between debugging and cooperation. As shown in the figure the relation between these two skills are non-linear because the generated scatterplot line is almost a straight line. This means that the two skills

(i.e. debugging and cooperation) are related to some extent with each other but their association is monotonic meaning that it is not possible to assume that an increase in the usage of debugging would directly result a significant increase in cooperation. At this point, it is important to highlight that the outcome of the Spearman's rank test result for this particular association (r=4.95; n=52; p=0.05) cannot be accepted simply because the relation between debugging and cooperation is non-linear. The scatterplots for the association of these two skills (i.e. debugging and cooperation) with other computational thinking skills are not displayed here as these were identified to be insignificant.

Spearman's rank correlation coefficient		Post Programming Sequence Knowledge	Post Functions Knowledge	Post Decision Making Knowledge	Post Loops Knowledge
Conditional Logic and Evaluating Conditions	Correlation Coefficient	.444**	.434**	.493**	.308**
	Sig. (2-tailed)	.000	.001	.000	.002
	N	52	52	52	52
Algorithmic Thinking	Correlation Coefficient	.414**	**.588****	**.646****	.393**
	Sig. (2-tailed)	.002	.000	.000	.004
	N	52	52	52	52
Running and Simulating Solutions	Correlation Coefficient	.343*	.431**	**.608****	.366**
	Sig. (2-tailed)	.013	.001	.000	.008
	N	52	52	52	52
Debugging and Handling Errors	Correlation Coefficient	.092	.181	.237	.386
	Sig. (2-tailed)	.518	.200	.091	.206
	N	52	52	52	52
Cooperation (Sharing ideas and strategies)	Correlation Coefficient	.174	.184	.228	.229
	Sig. (2-tailed)	.116	.150	.202	.117
	N	52	52	52	52

*. Correleation is significant at the 0.05 level (2-tailed).
**. Correleation is significant at the 0.01 level (2-tailed).

Table 6.36 – Spearman's rank correlations between computational thinking skills and pupils' perception of their programming knowledge in the PGS study.

Table 6.36 illustrates the Spearman's rank correlations between computational thinking skills and pupils' perception of their programming knowledge in the PGS study. As shown from the table, all programming constructs are associated to *conditional logic*, *algorithmic thinking* and *simulating solutions* at a significant level. However, the degree of correlations varies differently as some of these are modestly strong, others are weak. Further to this, no

significant correlation was defined between programming constructs and debugging, and in between programming constructs and cooperation. The Spearman's rank correlation's show that there are modestly strong and significant relationships between a) algorithmic thinking and functions (r=0.588; n=52; p=0.000) b) algorithmic thinking and decision making (r=0.646; n=52; p=0.000) c) simulating solutions and decision making (r=0.608; n=52; p=0.000). It was also identified that there are positive, significant but weak relationships between conditional logic and all programming constructs introduced in the game. As all correlations identified above are either modestly strong or weak, it is not possible to conclude that there are direct and non-monotonic relationships between computational thinking skills and learning programming constructs. This means that those pupils who developed algorithmic thinking during their game-play felt that their perception of knowledge in decision making and functions enhanced more than the other pupils. Additionally, those who felt they simulated solutions in the game also felt that they learned decision making more compared to the others.

The correlations obtained in the PGS study cannot be matched with the correlations obtained from the previous studies as a) the target groups were complete different and b) the participants in the PGS study were not learning computer programming. Nevertheless, the findings gathered from the correlations between computational thinking and programming constructs in the PGS study are consistent with the correlations obtained from the previous studies and therefore, these can support the previous argument raised that is there are no direct relationship between computational thinking skills and learning programming constructs through *Program Your Robot*.

A final Spearman's rank correlation was performed to investigate the associations among pupils' perception of their ability to visualise constructs from given problems, programming knowledge gained and problem solving abilities between the pre and the post study of the PGS study. The findings of the rank correlations show that there was a positive, modestly strong and significant correlation between pupils' perception of their ability to visualise programming constructs and their perception of programming knowledge gained, r=0.522; n=52; p=0.000. Additionally a positive, modestly strong and significant correlation was identified between pupils' perception of their ability to visualise programming constructs and their perception of problem solving abilities, r=0.579; n =52; p=0.000. Finally, the Spearman's rank correlation shows that there was a positive, significant and almost strong correlation between pupils' perception of their programming knowledge gained and their perception of problem solving abilities, r=0.610; n=52; p=0.000. Based on these results, it is

possible to report that the more pupils visualised programming constructs from given problems, the more they felt their problem solving abilities developed. Correspondingly, the more they visualised programming constructs, the more they felt their programming knowledge improved. As a result of this, it was found that pupils' perception of how much they learned from the game was modestly related to how much they felt their problem solving abilities developed.

Spearman's rank correlation coefficient		Difference in the ability to visualise constructs	Difference in programming knowledge	Difference in problem solving abilities
Difference in the ability to visualise constructs	Correlation Coefficient	1.000	**.522****	**.579****
	Sig. (2-tailed)	.	.000	.006
	N	52	52	52
Difference in programming knowledge	Correlation Coefficient	**.522****	1.000	**.610****
	Sig. (2-tailed)	.000	.	.000
	N	52	52	52
Difference in problem solving abilities	Correlation Coefficient	**.579****	**.610****	1.000
	Sig. (2-tailed)	.006	.000	.
	N	52	52	52

** Correleation is significant at the 0.01 level (2-tailed).

Table 6.37 – Spearman's rank correlations among pupils' perception of visualising constructs, programming knowledge gained and problem solving abilities between the pre and post study of PGS study.

6.4.7 Summary of findings regarding correlations

Unlike the other two studies, the results of PGS study cannot be investigated any further and the reasons for this are explained in Chapter 7 Section 7.2. To summarise the findings from Spearman's rank correlation coefficient assessments, a list is created below:

Only one strong, positive and significant correlation was identified in the PGS study and this was in between algorithmic thinking and simulating solutions, r=0.73; n=52; p=0.000.

There was a modestly strong, positive and significant correlation between:

a) conditional logic and algorithmic thinking, r=0.644; n=52; p=0.000;
b) conditional logic and simulating solutions, r=0.667; n=52; p=0.000;

c) algorithmic thinking and functions in programming, r=0.588; n=52; p=0.000;
d) algorithmic thinking and decision making in programming, r=0.646; n=52; p=0.000;
e) simulating solutions and decision making in programming, r=0.608; n=52; p=0.000;
f) visualising constructs and programming knowledge, r=0.522; n=52; p=0.000;
g) visualising constructs and problem solving abilities, r=0.529; n=52; p=0.000;
h) problem solving abilities and programming knowledge, r=0.61; n=52; p=0.000.

6.5 Summary

This Chapter first presented the feedback collected from a pilot study specifically designed to measure whether or not *Program Your Robot* has reached to the stage where a detailed evaluation could be carried out. Having analysed the feedback, the chapter explained the enhancements made to *Program Your Robot* before empirical studies were conducted. The chapter then presented a very detailed analysis of data collected from the Cyprus, the Greenwich and the PGS studies respectively. The demographic datasets were analysed in great detail to identify whether or not the datasets in each study fit to a normal distribution. The chapter then discussed what statistical methods were used for analysing each study and why these were selected rather than any other statistical method. A parametric measure (paired samples t-test) was used to analyse the data in the Cyprus study as the distribution of data was found to be relatively close to a normal distribution. Contrary to this, a non-parametric measure (Wilcoxon signed ranks test) was used to analyse the data in Greenwich and PGS studies where the distribution of data was found to be non-normally distributed.

Datasets were interpreted through the research questions in each study in considerable detail. Additionally, when reporting the results, the findings obtained of the Greenwich study was compared to the Cyprus study as the target group of these two studies was the same. The chapter also provided a statistically analysis on the correlations among five computational thinking skills (i.e. conditional logic, algorithmic thinking, simulating, debugging and cooperation), their associations with how far participants achieved in the game. Additionally, the correlations between these skills and the programming constructs introduced in the game (i.e. programming sequence, functions, decision making, loops) were investigated to observe whether or not there were strong associations between them. The difficulties faced in conducting the PGS study and how these were anticipated to affect the outcome of the study were also reported and discussed.

It was found that the statistical analysis of the Cyprus and the Greenwich studies supported

the alternative hypotheses (listed in Chapter 5 Section 5.3) and provided strong evidence to reject null hypothesis in all cases. The correlations in these studies showed that there are strong and linear associations between:

a) conditional logic and algorithmic thinking;

b) conditional logic and simulating solutions;

c) algorithmic thinking and simulating solutions;

d) ability to visualise programming constructs and problem solving abilities;

e) ability to visualise programming constructs and programming knowledge gained from the game;

d) problem solving abilities and programming knowledge gained from the game.

No strong correlations were identified between programming constructs and computational thinking skills or between how far students achieve in the game and computational thinking skills. The PGS study also provided a similar outcome to support the findings of the Cyprus and the Greenwich studies.

In the next chapter, the internal and external validity of the Cyprus and the Greenwich studies are analysed to conclude whether or not the outcome of these studies can be generalised. The chapter also explains the reasons why it was not possible to investigate the PGS study in terms of internal and external validity. Finally, the next chapter reports participants' quotes from all the studies and evaluate whether or not these quotes reflect a similar outcome to the responses given to closed-ended questions in the studies.

CHAPTER 7
EXPERIMENTAL VALIDATION

This chapter investigates *internal* and *external validity* of findings obtained in the Cyprus and Greenwich studies and examines whether or not any confounding variable impacted on the outcome of these studies. The inferential statistical analysis of the Cyprus and the Greenwich studies revealed that after playing *Program Your Robot* there was a positive significant reinforcement in students' perception of their a) attitude regarding learning computer programming through playing games; b) motivation in learning computer programming; c) knowledge in programming constructs introduced in the game (i.e. programming sequence, functions, decision making, loops); d) ability to visualise constructs and finally; e) problem solving abilities. However, it is not certain whether or not these statistical findings was a result of playing *Program Your Robot* or any confounding variable biasing the studies. In this chapter, the effect of confounding variables to statistical findings of the Cyprus and the Greenwich studies is investigated in considerable detail. The internal and external validity of the PGS study is not investigated as this study was merely conducted as an extension of the other two studies (i.e. Cyprus and Greenwich) and its main purpose was to observe whether or not school girls can benefit from *Program your Robot*. The reasons why the PGS study was not validated is also discussed in this chapter.

As described in Chapter 5 Section 5.4, the first step of the assessment plan of this research was the statistical analysis of raw data gathered from studies which was investigated in Chapter 7. The second and the third step is the internal and external validity of statistical findings and these are explored in this chapter. Following this, the internal validity of the Cyprus and the Greenwich studies are investigated in Section 7.1 under four categories: *history threat*, *maturity threat*, *mortality threat* and finally *regression threat*. Each of these threats was investigated and discussed based on raw data and/or inferential statistics obtained from the studies. The purpose was to conclude whether or not any of these threats can be ruled out as a rival explanation to the significant findings obtained in the Cyprus and the Greenwich studies. Section 7.2 discusses the final step of the assessment of findings that is the external validity of the Cyprus and Greenwich studies. This involves generalisation of the findings of experimental studies based on population selection and ecological background. Section 7.3 discusses the reasons why the findings of the PGS study were not investigated in terms of *internal* and

external validity. This section also describes why the positive outcomes of the PGS study were primarily used to support the findings in the Cyprus and the Greenwich studies rather than being treated as an individual study. Finally, Section 7.4 reports several quotes from students which provide qualitative evidence that the majority of them found that the game well suited to supporting the education of introductory computer programming.

7.1 Internal validity of the Cyprus and the Greenwich studies

This section discusses the internal validity of the findings obtained from the Cyprus and the Greenwich studies. *Internal validity* refers to any rival explanation that can be ruled out as an alternative reason to game interference for the cause-effect associations of findings in the experimental studies (i.e. Cyprus and Greenwich study). As previously described in Chapter 5 Section 5.4.1, threats to internal validity are divided into six main categories i.e. *history*, *maturity*, *testing*, *instrumentation*, *mortality* and *regression threat*. Four out of six of these categories are explored in this section in order to define whether or not they biased the outcome of the studies. Two of these threats (i.e. *instrumentation* and *testing* threats) do not have a major potential to impact the findings of the studies. Although the reasons for this are stated previously in Chapter 5 section 5.4.1, hereafter discussed again in order to emphasize the importance of this.

The *instrumentation threat* particularly endangers a study when a post-test is designed easier than a pre-test because under such circumstances participants can improve their score even though there would be no intervention. In other words, participants can score better in the post-test than the pre-test simply because the post-test is easier than pre-test. This threat does not apply to the Cyprus and the Greenwich studies mainly for two reasons a) the experimental structure of these studies is not a pre – post knowledge test but rather a pre – post study questionnaire about perceptions. It was not designed to collect or compare scores of participants in the studies and each research question is evaluated separately; b) the questions asked of participants do not have a correct answer. Therefore, it is not possible to set a "difficulty" for the pre and the post studies. Participants themselves decide which answer is the correct according to their own perception. Hence, the instrumentation threat does not apply to the outcome of the studies.

The *testing threat* endangers a study in a similar way as the instrumentation threat in that it is mainly related to "learning from experience" rather than the difficulty of the tests. A testing threat applies to a pre – post test study when the pre and post tests are exactly the same. As the

test in the pre-study is repeated in the post-study, participants can improve their score simply because they repeated the same test and discovered their own mistakes. In other words, the testing threat impacts the outcome of a study when participants can learn from their experiences and improve their scores even without an intervention. Similar to the instrumentation threat, this threat does not apply to the Cyprus and the Greenwich studies because a) the pre and the post study in these studies are not exactly the same. There are various questions in pre-study that were not repeated in the post-study (e.g. if you have ever thought about giving up your degree programme, was the difficulty of computer programming a key reason?) and in the same way there are questions in the post study that were simply not possible to be asked in the pre-study (e.g. how far did you achieve in the game?); b) There are no right or wrong answers in the questions and therefore, it is not possible for participants to learn from their mistakes. As argued on various occasions, the studies measure participants' motivation and attitude in learning computer programming, in addition to whether or not the game interference has an impact on their knowledge regarding how introductory programming constructs work (i.e. programming sequence, functions, decision making and loops).

As a result, the way that the pre and the post studies designed eliminated instrumental and testing threats from being major threats that could impact the outcome of these studies. The other internal threats (i.e. *history*, *maturity*, *mortality* and *regression*) are examined and discussed below:

7.1.1 History threat

The history threat impacts the outcome of a study when a specific past event or a chain of events impacts participants' behaviour during the experimental research. A history threat is generally a concern of longitudinal studies (research studies that are repeated over long periods of time); however it can also impact studies that are conducted in short periods of time. The history threat in this research is regarded as any significant advantage participants could have gained in answering one or more questions in the questionnaires. In this research the history threat was identified to be as: a) participants' background knowledge and experiences in computer programming; b) participants' previous experiences in educational games particularly with *learning through game-play*. If the majority of participants have sufficiently good programming knowledge background (regarding programming sequence, functions, decision making and loops), then it is less likely that an improvement in their perception of computer programming knowledge between the pre and the post study, will be observed.

Additionally, if participants had overly negative or positive experiences regarding *learning through game-play* this could impact how they would perceive *Program Your Robot* and thus could affect the outcomes of the studies. The statistical outcomes of the studies were investigated in depth in order to identify whether or not the history threat had a major impact on the results.

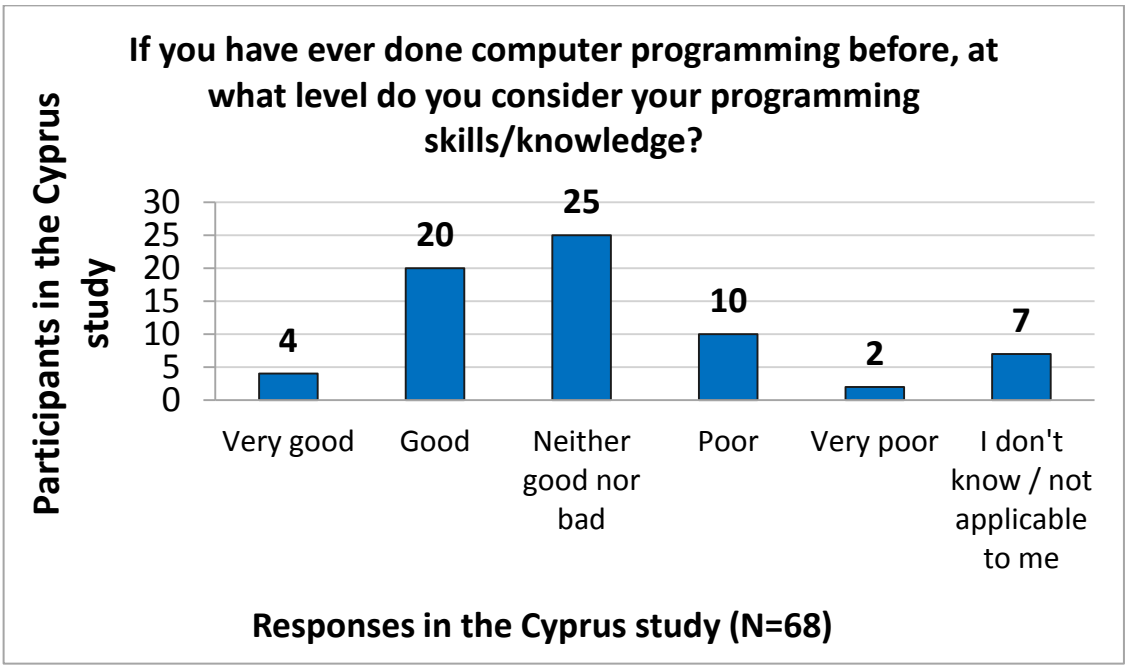

Figure 7.1 – Students' perception of their computer programming skills/knowledge before they participated in the Cyprus study.

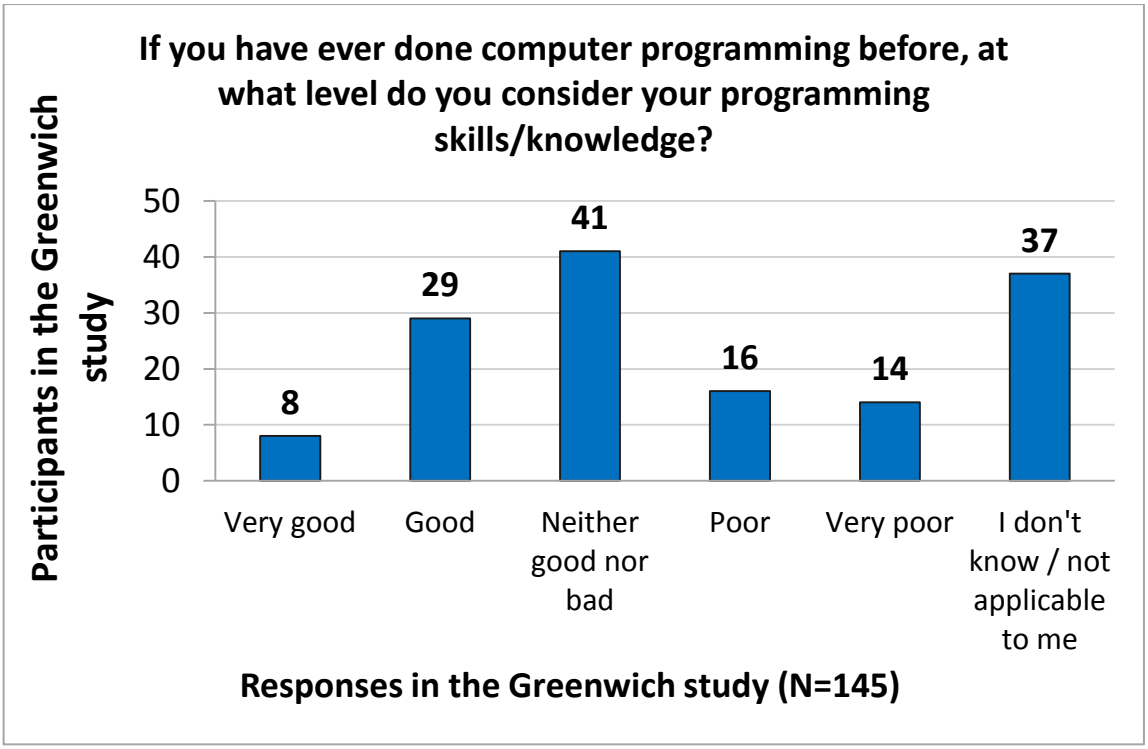

Figure 7.2 – Students' perception of their computer programming skills/knowledge before they participated in the Greenwich study.

As shown in Figures 7.1 and 7.2, 24 (35.2%) out of 68 participants believed that they have very good or good programming knowledge before they played the game in the Cyprus study. Whilst 25 (36.7%) participants were neutral, a total of 12 (17.6%) participants rated their knowledge and skills in computer programming as poor or very poor. Additionally, 7 (10.2%) participants claimed that they have no previous knowledge or experience regarding computer programming prior to their game-play in the Cyprus study. The responses collected in the Greenwich study are fairly comparable to the responses given in the Cyprus study. A total of 37 (25%) students rated their programming knowledge and skills as very good or good before the Greenwich study was conducted. While 41 (28.2%) out of 145 students were neutral, 30 (20.6%) more rated their programming knowledge and skills as either poor or very poor before they played the game. Finally 37 (25.5%) out of 145 students indicated that they had no prior knowledge or skills regarding computer programming prior to their game-play in the Greenwich study.

Although the demographic data obtained from the Cyprus and the Greenwich studies were not exactly the same, it is important to highlight that a very small percentage of students (5.5% - 5.8%) rated their programming knowledge/skills as very good before they played *Program*

Your Robot. This provides some evidence that the majority of students who participated in the Cyprus and the Greenwich studies did not have programming knowledge/skills to such extend that would prevent them learning how computer programming constructs work from the *Program Your Robot*. In other words, the target group was selected accurately as the majority of students did not rate their programming knowledge/skills very good or good before they participated in the studies.

In addition to participants' previous knowledge and skills in computer programming, all participants in the Cyprus and the Greenwich studies provided their feedback regarding whether or not they used a video game for educational purposes before they played *Program Your Robot*. As illustrated in Figures 7.3 and 7.4, the majority of the participants (63.2% in Cyprus; 58.6% in Greenwich) indicated that they had played educational games before they participated in the Cyprus and the Greenwich studies. In the Cyprus study, 36 (53%) out of 68 students claimed that they had played an educational game before and it was helpful to them. Whilst, 7 (10.2%) students claimed that they had played an educational game before but it was not helpful to them, a total of 25 (36.7%) students indicated that they had never played a game specifically designed for educational purposes in the Cyprus study. The responses given to the same question in the Greenwich study were also similar to this. 60 (41.3%) out of 145 students stated that they had played an educational game and it was helpful to them. While 25 (17.4%) students indicated that they had also played an educational game but it was not helpful to them, 60 (41.3%) more specified that they had never played an educational game prior to their participation in the Greenwich study.

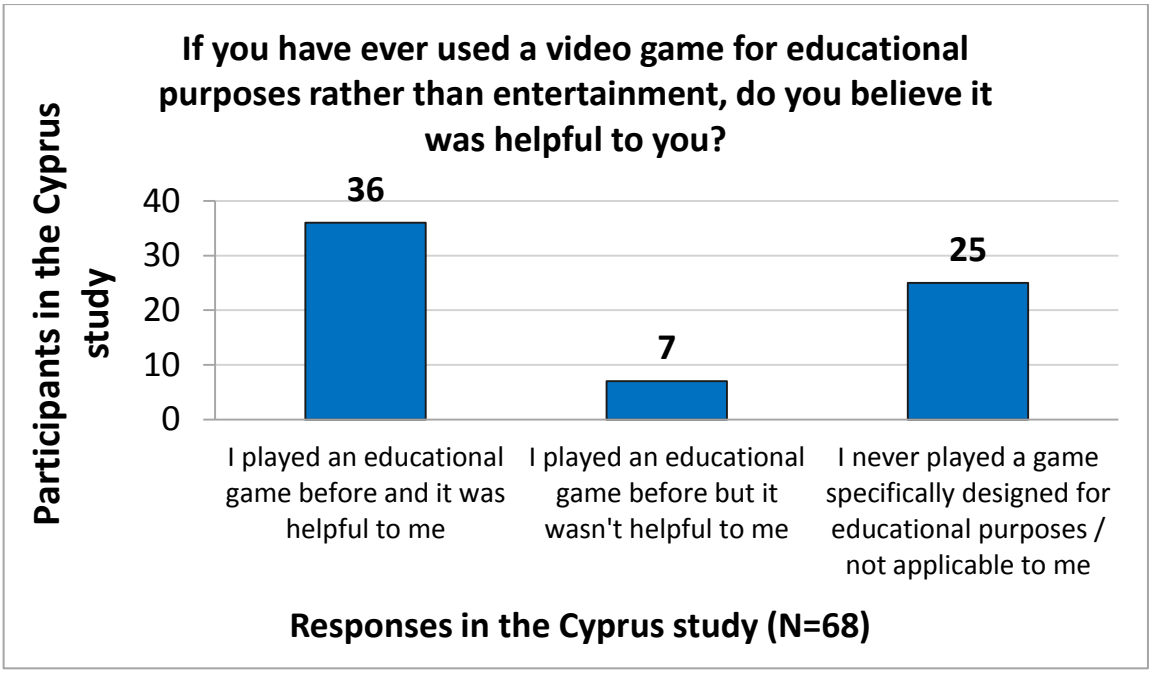

Figure 7.3 – Students' previous experiences regarding video games used for educational purposes rather than entertainment before they participated in the Cyprus study.

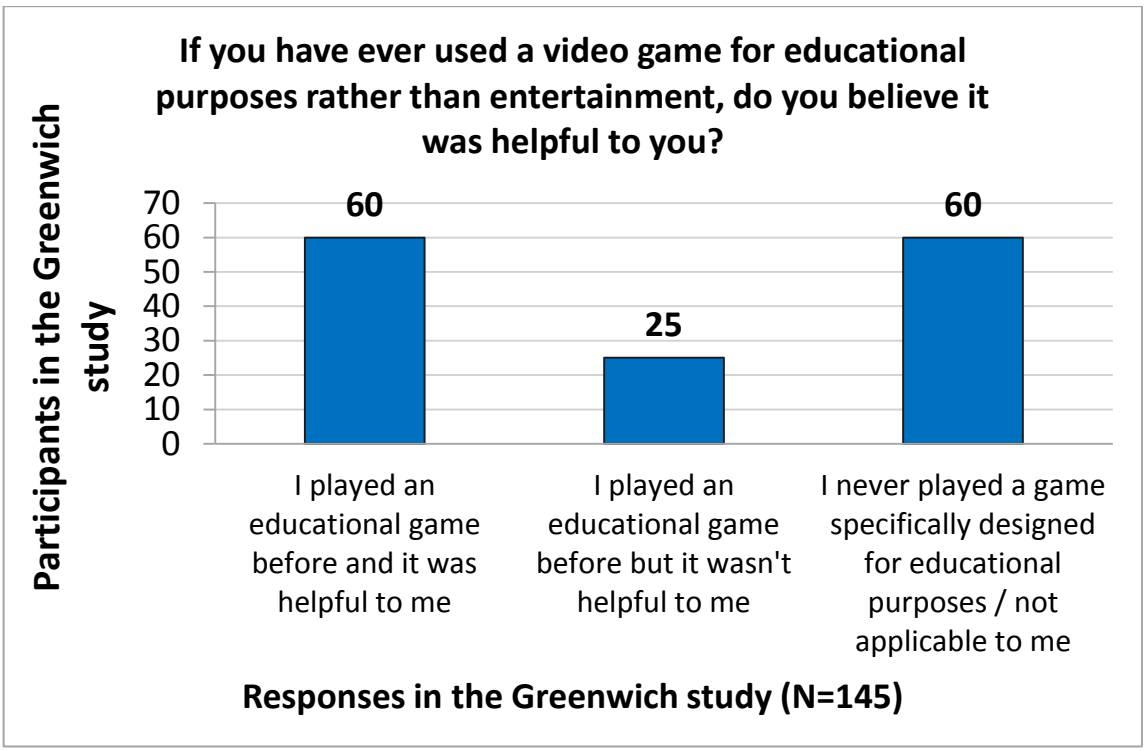

Figure 7.4 – Students' previous experiences regarding video games used for educational purposes rather than entertainment before they participated in the Greenwich study.

The data obtained in the Cyprus and the Greenwich studies show that the majority of participants did not have excessively negative experiences regarding educational games that

could have impacted their perception negatively to *Program Your Robot*. Further to this, 36 (53%) participants in the Cyprus study and 60 (41.3%) participants in the Greenwich study provided positive feedback that they played an education game before and it was useful to them. This provides evidence that the majority of participants did not have negative attitudes to *learning through game-play* and more importantly most participants were aware of educational games. As the participants' previous experiences regarding educational games were neither excessively positive nor extremely negative, it is possible to conclude that the target groups came from a randomly selected population. This provides some preliminary evidence to support the premise that the history threat regarding participants' previous experiences of educational games did not bias the outcome of the studies.

Descriptive statistics of students with little / no programming background in the Cyprus experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**.50**	10	.850	.269
Post Computer Programming Knowledge	**4.20**	10	.632	.200

Descriptive statistics of students with good programming background in the Cyprus experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**3.90**	10	.738	.233
Post Computer Programming Knowledge	**4.50**	10	.527	.167

Table 7.1 – Difference of programming knowledge of those students who have little or no programming knowledge and of those students who have fairly good or good programming knowledge between the pre and thepost study in the Cyprus study.

A descriptive statistical analysis was undertaken to assess whether or not those participants with no or little computer programming knowledge learned how computer programming constructs (i.e. programming sequence, functions, decision making and loops) work more than those participants who had a fair or good programming knowledge background. It is anticipated that, in a study where a history threat does not bias the outcome, participants with little or no programming knowledge would learn more from the game when compared to those who had fairly good or good computer programming knowledge. In order to evaluate this assumption, 10 participants who had poor or no programming knowledge were randomly selected from among those who rated their knowledge neutral, good or very good and 10 random selection was done from among those who rated their knowledge as none, poor or very poor. The reason why 10 participants were selected for each group is because only a limited

number of students rated their computer programming knowledge as very good and good before participating in the study.

As illustrated in Table 7.1, the descriptive statistics of those students who had little or no programming knowledge was compared against to those students who had fairly good or good programming knowledge in the Cyprus study. The mean difference regarding students' perception of their programming knowledge between the pre and post study in the first group (students with little or no programming knowledge) was found to be 3.7. In the second group (students with fairly good or good programming knowledge), the mean difference of students' perception of their programming knowledge between the pre and post study was found 0.6. These results show that during the study, those students who had little or no programming knowledge felt that they learned how programming constructs work more than those students who had a fairly good or good programming knowledge. As the descriptive statistical analysis was undertaken at the significant level ($p<0.05$), it is possible to conclude that this provides evidence to support that the premise that the history threat did not impact on the outcome of the Cyprus study.

Descriptive statistics of students with little / no programming background in the Greenwich experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**.85**	20	.745	.167
Post Computer Programming Knowledge	**3.65**	20	.745	.167

Descriptive statistics of students with good programming background in the Greenwich experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**3.65**	20	.616	.138
Post Computer Programming Knowledge	**3.80**	20	.745	.167

Table 7.2 – Difference of programming knowledge of those students who have little or no programming knowledge and of those students who have fairly good or good programming knowledge between the pre and the post study in the Greenwich study.

The same descriptive statistics was undertaken on the Greenwich data to measure whether or not the history threat impacted on the findings of the Greenwich study. As the sample size was higher, a total of 20 participants were evaluated in each group (little or no programming knowledge vs good programming knowledge) in the Greenwich study.

Table 7.2 illustrates descriptive statistics of the difference regarding programming

knowledge of those students who had little or no programming knowledge against those students who had fairly good or good programming knowledge between the pre and post study in the Greenwich study. To investigate whether or not the history threat impacted on the outcome of the Greenwich study, 20 participants who had poor or no programming knowledge were randomly selected from among those who rated their knowledge neutral, good or very good and similarly 20 participants were selected from among those who rated their knowledge as none, poor or very poor. The reason why 20 participants were selected for each group is because more students rated their computer programming knowledge as very good and good compared to the Cyprus study.

The findings are similar to those of the Cyprus study results and show that students who had little or no programming knowledge felt that they learned how programming constructs work from the game more than those students who had fairly good or good programming knowledge. The mean difference in the first group (those who rated their knowledge as none, poor or very poor) was found to be 2.8 whereas the mean difference in the second group (those who rated their knowledge as neutral, good or very good) was found 0.15. The descriptive statistics provide evidence that those who already have programming knowledge learned less from the game about how programming constructs work compared to those who had little or no programming knowledge prior to their game-play. Similar to the Cyprus studies these tests were undertaken at a significant level ($p<0.05$).

The descriptive statistical analysis of programming knowledge of students in the pre and post study of the Cyprus and the Greenwich studies clearly provide evidence that the history threat did not have a major impact on the outcome of these studies as those students with little or no programming background learned more regarding how computer programming constructs work from the game than those with good programming background. Additionally, all participants were randomly selected and it was identified that participants' background knowledge in computer programming and their previous experiences regarding educational games were broadly different. Therefore, it is possible to conclude that the history threat did not bias the outcome of these studies.

7.1.2 Maturity threat

History and maturity threats are very similar concepts with the exception of one distinct difference: while *history threat* is related to a specific event or a chain of events in participants' lives, the *maturity threat* is related to changes in participants (both physically and

psychologically) during a study. Similar to the *history threat*, the *maturity threat* is a major threat in longitudinal studies where observations take long periods of time sometimes even decades. So when a longitudinal study is completed, it is difficult to determine the cause of the discrepancy as it can be due to time factor rather than the study. In other words, subjects can change during the course of a study or even in between assessments and thus the cause-effect relationship might be caused by the maturity threat rather than an intervention applied.

As both the Cyprus and the Greenwich studies are short timed studies, maturity threat is more related to participants' behaviour in playing games rather than the changes happening in them during the studies. As the study duration is only one hour, it is not possible for participants to become more mature or the time factor to impact the outcome of studies. However, there can be considerable differences in participant responses that depend on how mature they are in terms of playing video games. As the maturity of participants differs, the amount of time they can spare for playing games can be different. Additionally, participants' attitude to learning through playing games can be affected by their maturity. Henceforth, in this study maturity threat is linked to how often participants play video games. In order to assess whether or not maturity threat has a major impact on the findings of the Cyprus and the Greenwich studies, the responses of those students who play games often were compared against to the responses of those students who do not play games often.

Figures 7.4 and 7.5 show how much students agreed that they play video games often in the Cyprus and the Greenwich studies. According to the results obtained in the Cyprus study, a total of 38 (55.8%) students strongly agreed and agreed that they often play video games. While 11 (16.1%) students were neutral, 16 (23.5%) out of 68 students strongly disagreed and disagreed that they play video games often. Additionally, 3 (4.4%) students indicated that they do not play video games at all in the Cyprus study. When the Greenwich study results are investigated, it was observed that the number of students who play games often is even higher than in the Cyprus study. 107 (73.7%) out of 145 students strongly agreed and agreed that they play video games often. Whilst 15 (10.3%) students remained neutral, 20 (13.7%) students strongly disagreed, and disagreed, that they play video games often. Finally, 3 (2%) students indicated that they do not play video games at all.

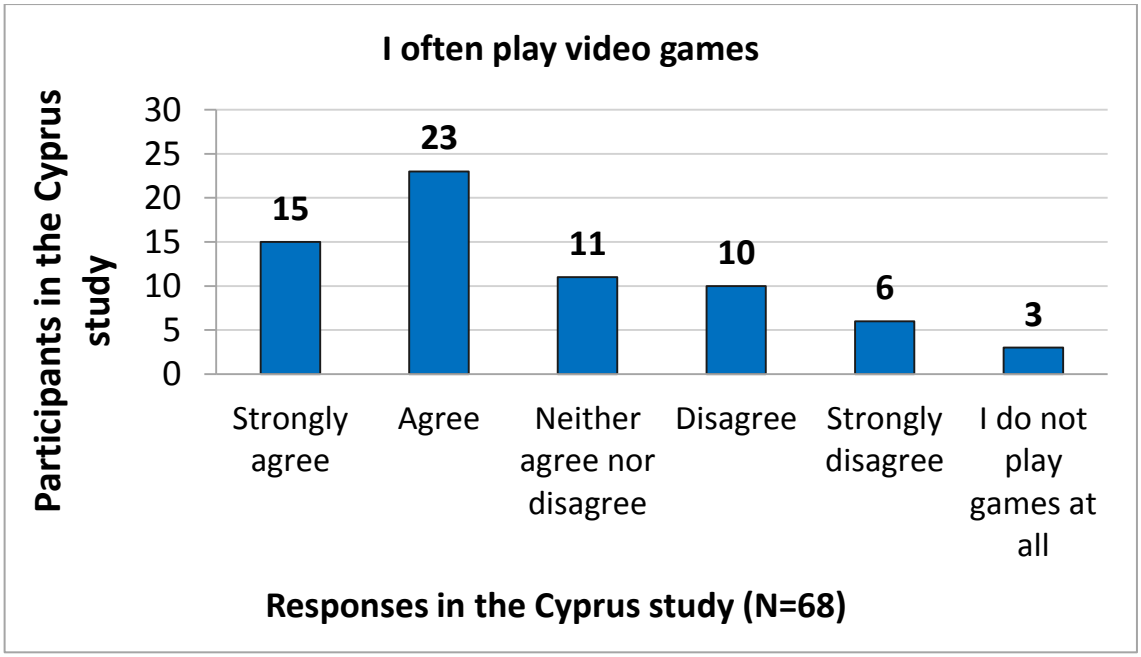

Figure 7.5 – How much students agree that they play video games often in the Cyprus studies.

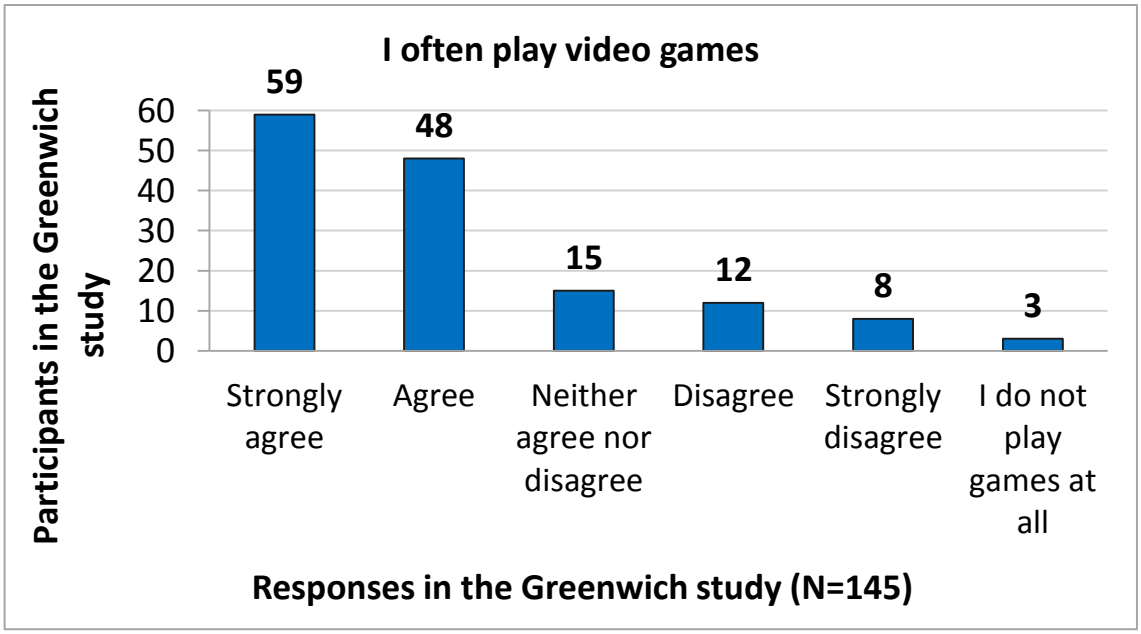

Figure 7.6 – How much students agree that they play video games often in the Greenwich studies.

The differences in participants' computer programming knowledge were investigated among those participants who play games often and those who do not play games often between the pre and the post studies of the studies. It is anticipated that, in a study where the maturity threat does not bias the outcome, the difference in participants' programming

knowledge would not depend on whether or not participants play video games often. In order to evaluate this assumption, 10 participants who strongly agreed and agreed that they often play video games were randomly selected and compared against 10 other randomly selected participants who either do not play games at all or strongly disagreed and disagreed that they often play video games. The reason why only 10 participants were selected is because there weren't many non-gamers among participants as many of the participants indicated that they play video games. Therefore, 10 gamer responses (those who often play video games) were compared against 10 other non-gamer (those who do not often play video games) responses to measure the maturity threat. The mean difference between these groups in the Cyprus study was assessed and presented below.

Descriptive statistics of students who do not often play video games in the Cyprus experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Programming Knowledge	**3.00**	10	1.287	.407
Post Programming Knowledge	**4.20**	10	.632	.200

Descriptive statistics of students who often play video games in the Cyprus experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Programming Knowledge	**3.30**	10	.949	.300
Post Programming Knowledge	**4.40**	10	.699	.221

Table 7.3 – Difference in programming knowledge of those students who play video games often and of those students who do not play video games often between the pre and post study in the Cyprus study.

Table 7.3 illustrates the difference in programming knowledge between those students who play video games often and those students who do not play video games often in the Cyprus study. The mean difference regarding students' perception of their programming knowledge between the pre and post study in the first group (10 randomly selected students who do not often play video games) was found to be 1.2. In the second group (10 randomly selected students who often play video games), the mean difference of students' perception of their programming knowledge between the pre and post study was found 1.1. Both descriptive statistical analyses were undertaken at the significant level ($p<0.05$). As the mean difference between the two groups is found to be very small (*mean difference* = 0.1), it is possible to

conclude that whether students were gamers or not, did not have a major impact on students' perception of gained programming knowledge in the Cyprus study.

The difference in programming knowledge between those students who play video games often and those students who do not often play video games in the Greenwich study is presented in Table 7.4. According to the table, the mean the difference in the first group (10 randomly selected students who do not often play video games) was 0.5 and the mean difference in the second group (10 randomly selected students who often play video games) was 0.3. The descriptive statistical analyses of both first and second group were undertaken at the significant level ($p<0.05$). The results show that the mean difference between the two groups is very small (*mean difference* = 0.2) which shows that students' perception of gained programming knowledge were not affected by how much they play video games in the Greenwich study.

Descriptive statistics of students who do not often play video games in the Greenwich experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Programming Knowledge	**3.20**	10	1.350	.427
Post Programming Knowledge	**3.70**	10	.949	.300

Descriptive statistics of students who often play video games in the Greenwich experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Programming Knowledge	**3.00**	10	1.476	.467
Post Programming Knowledge	**3.30**	10	.983	.353

Table 7.4 - Difference in programming knowledge of those students who play video games often and of those students who do not play video games often between the pre and post study in the Greenwich study.

The above descriptive statistical analyses revealed that there is a small mean difference in students' perception of their gained programming knowledge between non-gamers (those who do not play video games often) and gamers (those who play video games often) both in the Cyprus (*mean difference* = 0.1) and in the Greenwich studies (*mean difference* = 0.2). Although it is not possible to directly relate these findings to how often students play video games as students' age range, educational background and gender factors are not considered here, the above results clearly provide evidence that those participants who do not play games

often learned how programming constructs work as much as those participants who often play games. Therefore, it is possible to say that when age, gender and educational background factors are kept constant, how often students play games did not have a major impact on the outcome of how much they felt they learned from the game environment. In other words, there is evidence that suggest the maturity threat (participants' behaviour in playing games) did not bias the outcome of the studies.

7.1.3 Mortality threat

Mortality threat endangers a study when too many participants drop out of an experimental study. The main reason why the mortality threat is regarded as dangerous is because those participants who drop out of the study often tend to provide negative feedback and hence, if too many participants drop out from a study, this often means losing a considerable number of negative responses. Therefore, the results of a study might seem to be more positive than it really is. To estimate the degree of mortality threat, the dropout group is often compared against the non-dropout (participation) group (SRM, 2006). If there are no major differences between the groups, it is assumed that mortality was happening across the entire sample and is not biasing the outcome of the study. However, if the difference between the non-dropout and drop-out group is large, then the potential biasing effect of mortality needs to be considered carefully.

Experiment	Invited participant number	Total number of participants who completed the study	Drop-out rate	Participation rate
The Cyprus experiment	75	68	9.2%	90.8%
The Greenwich experiment	189	145	23.2%	76.8%

Table 7.5 – Participation and drop-out rates in Cyprus and Greenwich studies.

Table 7.5 illustrates the drop-out and non-dropout rates in both studies. While only 7 (9.2%) participants dropped out in the Cyprus study, the total number of participants who dropped out in the Greenwich study was 44 (23.2%). The exact reasons why the dropout rates of the Greenwich study is higher than the Cyprus study is not known and cannot be calculated precisely. However, the fact that the results of both studies are similar provides evidence that the modestly high dropout rates in the Greenwich study did not bias the outcome of the study.

More importantly, the drop-out rate of the studies is nowhere close to participation rates.

There is a considerable gap between the dropout and non-dropout rates of the Cyprus (81.6% difference) and the Greenwich (53.6% difference) studies. Therefore, it is possible to accept that mortality happened across the entire population of studies and the mortality threat did not bias the outcome of the studies.

7.1.4 Regression threat

A *regression threat* is a statistical phenomenon that occurs whenever a randomly selected population for a study is discovered to be a non-random sample with extreme scores. In other words, a regression threat endangers a study when subjects are selected on the basis of extreme scores (either high or low) that might impact the outcome of a study. As an example, if participants were selected based on their extremely low knowledge in computer programming in this study, the improvements at the end of the study might be due to regression toward the mean rather than the game's effectiveness as in reality participants *cannot know any lower than they already know* in computer programming. This is to say when a sample is selected just because it is "low-performing," any corrective measures applied will very likely to get the scores up simply because of regression toward the mean and not because of any real improvement due to game intervention. The most efficient solution to control regression toward the mean problem is to add a control group that does not receive the intervention. Should the control group shows the same change as the experimental group, then it can be assumed that the issue happened across the population. Despite this, it was simply not possible in this study to add a control group due to the reasons explained earlier in Chapter 5 Section 5.1.2. This makes the regression threat arguably the most dangerous internal threat that could impact the outcome of this study.

As a control group was not established into the experimental structure of this study, alternative ways were sought to assess whether or not regression threat had a major impact on the outcome of this study. Firstly, the average mean value of participants' knowledge in computer programming was investigated to observe whether or not there is considerable difference between the mean scores in the pre and the post study of the studies. Having identified the difference between the mean values in participants' perception of their computer programming knowledge, it was necessary to investigate whether or not *regression threat* has an effect on this.

To achieve this, a *multiple linear regression* is performed to predict the effect of explanatory variables to the outcome of the studies. *Multiple linear regression* (MLR) is a

statistical technique that models the mathematical relationship between two or more explanatory (independent) variables and a response (dependent) variable in an experimental study (Investopedia, 2013b). The model measures how the population mean response changes according to explanatory variables and therefore, it estimates the parameters of the population regression line (Yale, 1998). In other words, MLR can detect the effect of independent variables on a mean score of a dependent variable so that it can be identified whether or not the selected population has a major role in obtaining the outcome of a study. In the Cyprus study, MLR is used to measure whether or not age and gender (independent variables) have an effect on the participants' perception of their programming knowledge (dependent variables) between the pre and post study. In the Greenwich study, in addition to age and gender, mathematical qualifications of participants was also considered when MLR was undertaken. The ethnic classification of participants was simply ignored in the Greenwich study as this would mean categorising people's knowledge levels in computer programming according to their races. By performing a MLR, it is aimed to measure whether or not the selected population (in terms of their age, gender and mathematical qualifications) can be a major cause to obtain participants' perception of gained knowledge in computer programming. In other words, it was aimed at identifying the degree of correlations between participants' age, gender, mathematical qualifications and their perception of programming knowledge gained between pre and post study of studies.

Descriptive statistics of students' perception of their programming knowledge in the Cyprus experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**2.88**	68	1.322	.160
Post Computer Programming Knowledge	**4.24**	68	.576	.070

Descriptive statistics of students' perception of their programming knowledge in the Greenwich experiment

	Mean	N	Std. Deviation	Std. Error Mean
Pre Computer Programming Knowledge	**2.17**	145	1.556	.129
Post Computer Programming Knowledge	**3.63**	145	.824	.068

Table 7.6 – Descriptive statistics of students' perception of their programming knowledge in the Cyprus and the Greenwich studies.

As illustrated in Table 7.6, students scored their perception of programming knowledge

considerably lower before they participated in the Cyprus (M = 2.88) and the Greenwich (M=2.17) studies. After playing the game, the average score regarding students' perception of their programming knowledge raised from 2.88 to 4.24 (mean difference = 1.36) in the Cyprus study and from 2.17 to 3.63 (mean difference =1.46) in the Greenwich study. The large gap between the pre and the post study in both studies indicates that a regression to the mean values could have happened during the studies. In other words, it is possible that this large difference could have been caused by the regression threat (the extreme scores in the target group) rather than the game intervention. Therefore, to estimate the risk of regression and to investigate the effect of population to the mean scores, a multiple linear regression was performed in the studies.

Multiple linear regression (MLR) analysis consists of three main statistical stages which are an Analysis of Variance (ANOVA) test, a model summary and correlation coefficients. The first stage shows ANOVA test results regarding the overall impact of independent variables (also called predictors) on the dependent variable. To interpret the ANOVA test results correctly, a null and an alternative hypothesis was created. The null hypothesis (Ho1) indicates that there is no significant linear relationship between predictors and dependent variables. The alternative hypothesis (Ha1) indicates that there is a strong and significant linear relationship between predictors and dependent variables. When the significant value of the ANOVA test (p value) is greater than 0.05 ($p>0.05$), the null hypothesis is accepted and in the same way when the significant value is less than 0.05 ($p<0.05$), the null hypothesis is rejected thus the alternative hypothesis is supported. Accepting the null hypothesis means that the outcome of dependent variable has no significant correlation with the independent variables which provide evidence that regression threat does not have a major impact on the outcome of the study. The opposite of this is rejecting the null hypothesis which means that the outcome of dependent variable is somehow correlated with the independent variables and therefore, a regression threat impacts on the outcome of the study.

The second stage is a model summary that shows how strong the correlations are in between the predictors and the dependent variable. The R value demonstrates the correlation coefficient and R^2 indicates how strong the correlation is. The crude estimate available for interpreting the strength of correlations in multiple linear regression is exactly the same as Pearson's correlation. This is to say that a strong positive correlation is equal or greater than +0.7; a modest strong correlation ranges from +0.49 to +0.69 and a weak correlation is accepted between +0.2 and +0.39. Any correlation that ranges between +0.01 and +.019 is often accepted as negligible or does not exist at all. The negative correlations also follow the

same structure but with a negative value rather than a positive value.

The final stage is the correlation coefficients which provide evidence on whether or not the correlation between each independent variable and the dependent variable is significant and strong. Each independent variable is matched with the dependent variable individually in order to identify whether or not the predictors have a significant impact on observations.

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	1.481	2	.740	.376	**.688[b]**
	Residual	128.049	65	1.970		
	Total	129.529	67			

a. Dependent Variable: Difference in programming knowledge between pre and post study

b. Predictors: (Constant), Gender, Age Range

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	**.107[c]**	**.011**	-.019	1.40356

c. Predictors: (Constant), Gender, Age range

Coefficients[d]

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
Age Range	.164	.375	.054	.438	**.663**
Gender	.278	.354	.097	.785	**.435**

d. Dependent Variable: Difference in programming knowledge between pre and post study

Table 7.7 – Multiple regression analyses of the relationship between students' perception of their programming knowledge and various potential predictors (i.e. gender, age range) in the Cyprus study.

As illustrated in Table 7.7, the ANOVA test results of the Cyprus study show that the significant number is greater than 0.05 (F= Mean Square Regression (MSReg) / Mean Square Residual (MSE) = 0.376; p=0.688). Further to this, the multiple regression model summary with two predictors (i.e. age range, gender) produced $R^2 = 0.011$ (1%); which is a very small and negligible number. The coefficient results for students' age range and gender also have no

significant correlation with their perception of gained computer programming knowledge. The coefficient significant value for age range and gender variables is higher than 0.05 (p=0.66 for age range; p=0.43 for gender) which indicates that students' age range or gender has no individual linear relationship with students' perception of programming knowledge. As the ANOVA test results generated an insignificant value, the null hypothesis which indicates that there is no significant linear relationship between predictors (students' age range and gender) and students' difference in programming knowledge is accepted.

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	37.677	3	12.559	4.154	**.077[b]**
	Residual	426.281	141	3.023		
	Total	463.959	144			

a. Dependent Variable: Difference in programming knowledge between pre and post study

b. Predictors: (Constant), Mathematical Qualifications, Gender, Age Range

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	**.285[c]**	**.081**	.062	1.73876

c. Predictors: (Constant), Mathematical Qualifications, Gender, Age range

Coefficients[d]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	Gender	1.247	.421	.240	2.962	**.004**
	Age Range	.661	.303	.181	2.182	**.031**
	Mathematical Qualifications	-.013	.109	-.010	-.123	**.902**

d. Dependent Variable: Difference in programming knowledge between pre and post study

Table 7.8 – Multiple regression analyses of the relationship between students' perception of their programming knowledge and various potential predictors (i.e. mathematical qualifications, gender, age range) in the Greenwich study.

In addition to the analysis of the Cyprus study, a correlation and multiple linear regression analysis was conducted to examine the relationship between students' perception of their

programming knowledge and potential predictors (i.e. gender, age range, mathematical qualifications) in the Greenwich study. Table 7.8 provides a summary of descriptive statistics and analysis of results. The ANOVA test results show that the overall relationship between the independent variables and the dependent variable is neither strong nor significant (F= Mean Square Regression (MSReg) / Mean Square Residual (MSE) = 4.154; p=0.077). The multiple regression model with three predictors (i.e. age range, gender, mathematical qualifications) produced R^2 = .081 (8%) which is a negligible relationship. Despite these, the coefficient results show that the students' age range and their gender have a significant correlation with students' perception of gained computer programming knowledge ($p<0.05$) in the Greenwich study. As the significant value is lower than 0.05 both for age range and gender variables (p=0.04 for gender; p=0.031 for age range), it is possible to report that students' age range and gender has a significant correlation with students' perception of programming knowledge. Despite this, the mathematical qualification of students (the third predictor) has no significant correlation (p= 0.902). Furthermore, as the significant value in the ANOVA test is greater than 0.05 (p= 0.077), the null hypothesis is accepted. This is to say that there is no significant positive linear correlation between the predictors (i.e. students' age range, gender and mathematical qualifications) and students' perception of gained programming knowledge in Greenwich study.

In conclusion, the findings of multiple linear regression analysis of the Cyprus and Greenwich studies show that the independent variables in both studies do not have a major impact on the students' perception of their gained programming knowledge between the pre and post study. Although gender and age range variables in the Greenwich study have a significant correlation with the students' perception of gained programming knowledge, the relationship between these are ignorable (R^2=0.004; p=0.04 in gender; R^2=0.024; p=0.031 in age range). The ANOVA test results of both studies supported the null hypotheses that is to say the gender, age range (and mathematical qualification in the Greenwich study) do not have an impact on the outcome of the study (i.e. programming knowledge of students).

As a result, it is possible to conclude that the regression threat did not have a major impact on the findings of studies as it is identified that participants' age range, gender and mathematical qualifications do not have a significant impact on the outcomes.

7.1.5 Summary of internal validity of the Cyprus and the Greenwich studies

The evaluation of the internal validity of the Cyprus and the Greenwich studies show that:

a) *History threat* did not bias the outcomes of the studies as it was found that those students with little or no computer programming background learned significantly more than those students who have a good computer programming background during the studies.

b) *Maturity threat* did not endanger the outcome of the studies because those students who do not play games often learned how programming constructs work as much as those students who often play games during the studies.

c) *Mortality threat* did not impact the outcome of the studies because the results of Cyprus and Greenwich studies are similar and consistent despite the fact that the dropout rates of studies are different.

d) *Regression threat* did not endanger the outcome of the studies as it was identified that the independent variables collected in the studies (i.e. age range, gender in Cyprus; age range, gender and mathematical qualifications in Greenwich) did not have a significant and strong effect on how much students learned about programming constructs in the game.

These results provide strong evidence that the internal threats listed above did not bias the outcome of the Greenwich and the Cyprus studies at a critical level. In this case, the internal validity of the research supports the premise that the cause-effect relationship of studies is drawn from the game interference rather than any other confounding factor.

7.2 External validity of the Cyprus and the Greenwich studies

External validity is the extent to which the outcome of a study can be generalised to other people at other times in different locations. The external validity of a research is generally investigated under two titles which are *population validity* and *ecological validity*. This section discusses the population and the ecological validity of the Cyprus and the Greenwich studies.

Population validity is the type of external validity that evaluates whether or not the selected sample population represents a real world population. Strong population validity can be obtained by applying a random selection sampling method rather than convenience sampling (Explorable, 2011). Ecological validity is related to the testing environment and evaluates whether or not the testing environment affects the behaviour of participants. Strong ecological validity can be obtained by repeating a study in various different locations at different times in a testing environment where participants would not feel uneasy or nervous. As a result, the external validity of a research is weak when a) the target population of a study is selected from a single geographic location and the study is not repeated with different participants at different times (ecological validity) b) a small size of data is collected from a certain population without applying a randomised selection (population validity).

Having ensured the cause-effect relationship of the studies is drawn from the game interference, the external validity of this research (both population and ecological) needs to be discussed.

The Cyprus and the Greenwich studies have exceptionally strong population validity because students who participated in these studies have wide variations in terms of their age group, gender, ethnicity and culture. Moreover, participants participated in the studies without any consideration as to whether or not they have a good gaming or computer programming background. All participants were first year computer programming students and the sample size of the study is large enough to be able to generalise the outcome of the studies. Further to this, meaningful statistical methods were used when analysing the datasets. This is to say a parametric measure is used to analyse data when a normal distribution is identified and similarly, a non-parametric measure is used when the data came from a non-normally distributed population. In this case, it is possible to conclude that sample groups in the studies were as representative as possible and can be extrapolated to a population as a whole.

Additionally, both studies have strong ecological validity because the studies were a) conducted in the tutorial hours of students in the same computer lab where their regular sessions take place; b) repeated in different geographical locations on different participants (Cyprus and UK); c) anonymous and voluntary and therefore, there was no critical reason for students to feel nervous or ill at ease when participating.

Repeating the same study in different locations at different times with different groups of students studying introductory programming ensured that the experimental structure of this study has a strong external validity. The only downside of the experimental study was the lack of a control group which is a limitation of this research and is further discussed in Chapter 8.

As both the population and ecological validity conditions are satisfied, it can be concluded that the external validity of this study applies to all first year programming students who are in between 18 – 40 age range (regardless of their gender, ethnicity and mathematical qualification).

7.3 Internal and external validity of the PGS study

As indicated in Chapter 6 Section 6.4, the internal and external validity of the PGS study is not investigated as this study was merely conducted as an extension of the other two studies (i.e. Cyprus and Greenwich). The aim of PGS study was to observe whether or not school girls can benefit from *Program your Robot* as well as to measure whether or not this game would develop their abilities in computational thinking. To achieve this, the main experimental structure of the study was modified and made available to school pupils. Despite this, it is not possible to regard the PGS study as an individual study and generalise its findings in its own right. The reasons for these are discussed below:

Firstly, *Program Your Robot* was designed to practice four sets of programming constructs (i.e. programming sequence, functions, decision making and loops) at the computational thinking level for students who are learning introductory computer programming. The school pupils who participated in PGS study were not registered on a computer programming course and they were not enrolled on a Computer Science or a similar degree programme. Hence, as the majority of participants in the PGS study had no basic motivation to learn any of the computer programming constructs introduced in the game, they cannot be regarded as a target group for this research. Additionally, the main goal in the PGS study was to observe changes in participants' perception of their a) problem solving abilities; b) ability in visualising how computer programming work; and c) computational thinking skills (conditional logic, algorithmic thinking, simulating, debugging and social learning) rather than measuring their knowledge in computer programming constructs. Therefore, as the PGS students did not really represent a target group of the study, it is not possible to validate any of the findings of this study.

Secondly, the findings obtained from the evaluation of the PGS study were not always positive and significant. As argued in Chapter 6 section 6.3.1, the difference in pupils' attitude to learn computer programming through playing games was found to be insignificant (first research question in the PGS study). In order to evaluate the internal validity of a study, the integrity of the outcomes must be consistent and need to support each other. In other words, if

a specific outcome of a study is insignificant (i.e. pupil's attitude to learn computer programming through game-play in the PGS study), the internal validity of that study is endangered and often not investigated further as an insignificant outcome breaks the integrity of the work.

Thirdly, the majority of internal threats in the PGS study cannot be evaluated accurately because of the participants' background and population selection. As an instance, it is not possible to assess whether or not the history threat impacted on the results of the study as almost all participants had very little or no computer programming background. Therefore, it is not possible to observe how well participants who had good programming background did compared to those participants who had little or no programming background. Additionally, 30 out of 82 (36.5%) participants dropped out from the study and because the study was not repeated in another girl's school it is not possible to assess whether or not the mortality threat impacted the outcome of the PGS study. Finally, two out of three independent variables (age and gender) collected in the PGS study were constant which is a major obstacle that prevents a multiple linear regression analysis to be carried out. As discussed in Section 7.1.4, a multiple linear regression is performed to measure the effect of two or more independent variables on a dependent variable observed in a study. As all participants were 15 years old girls, a multiple linear regression cannot be performed simply because age and gender independent variables are constant for all participants. The only non-constant independent variable was ethnicity and as argues previously using ethnic classification to categorise people according to their race might raise some ethical issues. Additionally, having only one independent variable simply does not satisfy the prerequisite for using a multiple linear regression. Hence, it is not possible to identify whether or not a regression threat endangered the outcome of the PGS study through using a multiple linear regression. As a result, the internal validity of the PGS study cannot be evaluated accurately in the way that the results of the previous studies were evaluated.

Finally, repeating a study is vital in order to validate the population and the ecological validity of a research and, it is important for observing whether or not the results would be duplicated on different participants. As the PGS study was not repeated and the results are not internally validated, it is not possible to perform an external validity to generalise the outcomes of the study.

In conclusion, the findings of the PGS study cannot be validated internally or generalised because a) not all findings of the study were significant (i.e. the difference in pupils' attitude to learn computer programming through playing games); b) an internal validity of the statistical

findings cannot be accurately assessed c) an external validity cannot be achieved as statistical findings were not assessed internally.

7.4 Open-ended question answers obtained from the studies

This section presents the feedback obtained from the participants at the end of the post-study in each study. Although an open-ended question was asked to participants at the end of the pre-study questionnaire in each study, very few participants answered this question. The insufficient responses obtained from participants indicated that students did not have an idea on how to learn computer programming constructs through playing games. Therefore, the feedback provided in the pre-study part of the studies is not reflected in this section.

Despite the fact that very few players provided their viewpoints in the pre-study, many participants provided their viewpoints regarding their game experience after playing *Program Your Robot*. It is postulated that participants did not provide their viewpoints prior to their game-play because they did not have an idea about how a game can teach how computer programming constructs work effectively.

A number of qualitative student comments were collected at the end of each study and the majority of these were predominantly positive. More importantly, many participants evaluated the game at a critical level and their comments provided qualitative evidence to reinforce the findings and the validity of the Cyprus and the Greenwich study.

In the post-study part of the studies, participants were asked to answer whether or not: a) the game was helpful to them; b) using *Program Your Robot* is a good idea to support their tutorials; c) if they would like to see improvements in the game and if so what type of improvements. The majority of participants provided constructive and positive feedback regarding the game. The optimistic attitude and excitement of students underpins the findings of studies and provided qualitative evidence that the game was indeed supportive to them. Several quotes from students are cited below to provide qualitative evidence that they found the game well suited to help them to understand introductory programming constructs. The quotes from students are listed in the order that the studies were evaluated that is the Cyprus, the Greenwich and the PGS study respectively.

7.4.1 The Cyprus study quotes

Student 1:

"I think the game is really good and should be used in the education of computer programming."

Student 2:

"I think the game emotionally drives you to complete it and I personally did not get satisfied until I completed it. I simply ended up playing it until the very end!"

Student 3:

"I think if this game is improved further it can be very useful for learning how computer programming constructs work."

Student 4:

"I think playing the game is difficult and it really pushes you to think logically."

Student 5:

"I think the functions designed at one level should be available in the next level as well so that we would not end up designing them continuously. In its current state, playing the game feels like work and very tiring. For example, dragging and dropping commands are a bit annoying. Another improvement can be on the rotations (i.e. turn left/right) as currently this is confusing. Despite that I am sure these changes will provide a smooth gaming experience, I am not sure how this will contribute to players' learning process of computer programming."

Student 6:

"The idea behind the game is really good. It was also fun to play."

Student 7:

"I felt like I am playing chess because I had to think out every move! The game really has the potential for learning programming."

Student 8:

"I think this game is only useful to those people who know nothing about computer programming but wish to learn it."

Student 9:

"I think this game is useful for practicing computer algorithms and learning introductory programming. I wish there was more levels to play :)."

Student 10:

"I think this game is useful to improve logical thinking and abilities. However, I am not sure if the game really helps understanding how programming concepts work."

Student 11:

"I think this game promotes logical thinking. I also think we should be given homework or tutorials to play similar games so that we would have ideas on how to design interactive software for educational purposes."

7.4.2 The Greenwich study quotes

Student 1:

"Yes, it was a good idea to use this game but improving the graphics can help me understand more."

Student 2:

"Simplifies understanding"

Student 3:

"I will use this learning tool at home and with my studies"

Student 4:

"I think this game is going in a good path. I would like it to have more levels that would develop my ideas of computer programming even better. You could also include on the side

how the code would look like if it was shown in Java (or any other language) this would give insight to the player of how that "program" would look like as the real thing. Another idea would be a save button so that users could track their progress in learning, maybe even have user accounts."

Student 5:

"Great game, hope it does come into action into the future."

Student 6:

"Making the conditional statement customisable would have allowed for more advanced functions for example testing if the edge is in front of you, if it is, turn and test again, if the path is in front of you then move etc."

Student 7:

"I like the idea of this game but the execution was a little confusing at times, the decision making was a bit confusing to use and would have benefited from a more robust tutorial."

Student 8:

"Bloody good work and a great idea!"

Student 9:

"When a specific function is executed it could be highlighted, this would help to better understand which function does what. Also at the end of each level there could be an example of a real java code showing up, showing how it would work if typed."

Student 10:

"The game was entertaining an also very original. Excellent work."

Student 11:

"The game should have more levels, different enemies that require certain conditions to overcome. Maybe adding weapons so the robot can take out the enemy robots? Graphics are

not an issue the cartoony look seems friendly."

Student 12:

"During the run time, I would like to see which action was executed and which did not, i.e. when I did a wrong turn it was quite hard to find out on which turn did it stop at! Thanks."

Student 13:

"For improvements on the game I think more levels could be added in future versions of it. Harder levels could also be included to help the user really tackle any issues they are having with computer programming so they can use the game for practicing!"

7.4.3 The PGS study quotes

Student 1:

"I thought it was a useful method of teaching us how sequences work."

Student 2:

"It wasn't good when all of class first attempted to play the game as kept freezing and would not allow us to play. Level 4 was very hard and I did not understand it. I felt like I needed better and more interesting instructions."

Student 3:

"I understood most of the game after trying scenarios and didn't tend to read the tutorial as they were a bit long winded. I found the game interesting but I feel that more simplified instructions would be more beneficial."

Student 4:

"The graphic should be better so that the game can be more engaging to play."

7.5 Summary

This chapter evaluated the internal validity of the Cyprus and the Greenwich studies particularly the *history*, *maturity*, *mortality* and *regression* threats. The potential effect of these threats and whether or not they bias the outcome of the Cyprus and the Greenwich studies were

investigated and supported with statistical analysis. It was found that none of these threats has a major impact on the findings of these studies. Having verified the internal validity of the studies, the external validity is assessed in terms of the population and the ecological validity in order to generalise the results. The internal validity of the Cyprus and the Greenwich studies supported the premise that the significant results obtained from the studies was as a result of the game intervention rather than a confounding variable. Moreover, it was found that these studies were externally valid simply because the findings of studies were similar despite the fact that they were conducted in different locations at different times with randomly selected first year computer programming students. Further to this, a detailed discussion was provided on why the results of the PGS study cannot be validated internally and externally. Finally, quotes were reported from various students from all studies in order to provide qualitative evidence that the game was indeed supportive for introductory programming students.

In the next chapter, the limitations of this research and conclusions drawn from the findings of studies and potential future work are discussed.

CHAPTER 8
CONCLUSION & FUTURE WORK

This chapter outlines the main contributions and limitations of this research and the possible future developments and research. Section 8.1 describes a summary of the research and covers the aims of the research and how this was carried out. Section 8.2 describes whether or not the research objectives are met. This section also describes how the main research question was fluctuated during the research. Section 8.3 specifies the main contributions of the research under two titles as modelling and statistical contributions. Section 8.4 lists the limitations of the research, and subsequently, section 8.5 discusses some of the possible directions for future research and analysis. Finally, section 8.6 briefly discusses the conclusion achieved at the end of the research.

8.1 Summary of the research

This research primarily focused on developing a game model and the implementation of this (i.e. *Program Your Robot*) in order to construct a direct relationship to the application of computational thinking in the process of learning how a limited number of key introductory programming constructs work (i.e. programming sequence, functions, decision making and loops). The statistically valid evidence that proves learning by playing games is an educationally effective solution has long been absent from the literature in learning how programming constructs work (Hainey *et al.*, 2011; Mitamura, Suzuki & Oohori, 2012). This research was aimed at providing the missing statistical evidence that a serious game can be an educationally effective tool for developing skills in computational thinking (i.e. conditional logic, building algorithms, simulation, debugging and socialising) and learning how a defined range of introductory programming constructs work (i.e. programming sequence, functions, decision making and loops).

The research first identified what is computational thinking and how it can help to develop a student's abilities to support the learning of computer programming. The current problems in *learning through playing games (or serious games)* are also identified specifically in the field of learning computer programming constructs. Having identified these problems, the research introduced an improved game model called the *interaction – feedback loop* based on the body of the existing work in this area. This game model was used in conjunction with a series of

published guidelines (Kazimoglu *et al.*, 2011) to develop *Program Your Robot*, a serious game specifically designed for a) learning how the defined range of introductory computer programming constructs work at the level of computational thinking; b) developing cognitive skills that encompass computational thinking

In order to evaluate the effectiveness of *Program Your Robot*, a semi-structured (i.e. the pilot study) and three structured studies (i.e. the Cyprus, the Greenwich and the PGS) were designed and conducted.

The semi-structured study was conducted as a pilot study at the University of Greenwich and involved students with diverse background and knowledge in computer programming. This was a free form initial evaluation of the game before moving to the empirical stages of the research. The feedback obtained from the semi-structured study was overwhelmingly positive and the participants provided constructive suggestions and guidance for improving the game experience of *Program Your Robot*.

Having improved the game based on the feedback obtained in the pilot study, three rigorous studies were conducted. The experimental design of these studies was virtually the same and fitted into *one group pre – post study design* among quantitative research methodologies. The studies followed the structure of a pre-study questionnaire, student then play the game, followed by a post-study questionnaire and they were conducted in two different Universities and in a public girls school.

The responses of participants are matched in all studies and the obtained data was analysed using inferential statistics for the premise of providing the missing statistical evidence in the literature. A parametric measure (i.e. samples paired t-test) was used when the distribution of data was found to be relatively close to a normal distribution, and a non-parametric measure (i.e. Wilcoxon signed ranks test) was used when the distribution of data was found to be non-normally distributed. The statistical correlations among the five computational thinking skills, their associations with how far participants progressed in the game, and with the programming constructs introduced in the game were also investigated in considerable detail. Finally the validity of the statistical findings was explored in order to generalise the outcome of the studies. It was found that the results of the two studies that were conducted in the Universities (i.e. the Cyprus and the Greenwich studies) are internally and externally valid which means that the significant outcomes of these studies can be generalised for introductory programming students. The findings of PGS study provided support for the finding produced from the other two main studies (i.e. the Cyprus and the Greenwich studies), but otherwise not usable because a) the participants in the PGS study did not represent the target group of the research and b) the

findings of PGS study cannot be internally or externally validated.

8.2 Meeting research aims and objectives

This section revisits the main research question, aims and objectives of the research outlined in Chapter 1 Section 1.2, and discusses whether or not the objectives were met and how the main research question fluctuated during the research.

The original research question of this research was:

"Can a serious game be designed to support the development of computational thinking through the medium of learning computer programming?"

Seven different objectives were set in order to answer this research question (see Chapter 1 Section 1.2).

The first objective was to *identify the most common problems students experience in learning introductory computer programming*. This objective was achieved by reviewing the literature and defining the major problems/obstacles students experience when learning computer programming (see Chapter 2 Section 2.1). Achieving this objective provided an insight for recognizing the need to support students at an operational level of abstraction so that they can challenge their abilities in solving problems.

The second objective was to *investigate the differences and similarities between computational thinking and inherently learning introductory programming*. This objective was achieved by identifying and discussing three different dimensions between learning computer programming and computational thinking (see Chapter 2 Section 2.2.2). Understanding the outcome of this objective was a crucial step in the research as it highlighted the decision to design a game that encouraged computational thinking abilities rather than teaching programming code to students.

The third objective was *an analysis of the current use of serious games to support learning computer programming*. In order to achieve this objective, various studies that use a game based learning approach to teach computer programming was critically reviewed and discussed. Moreover, the most widely referenced serious game development and evaluation models were investigated in order to acknowledge advantages and disadvantages of the current models used to design serious games (see Chapter 2 Sections 2.3 and 2.4). This objective was

also successfully achieved and it was decided to develop a new game model to focus on the development of computational thinking abilities.

The fourth objective was to *identify the reason why the statistical evidence regarding serious games and learning is absent from the literature*. This objective was too broad and not really related to the research question and therefore, it was refined: *to identify the reason why the statistical evidence regarding serious games and learning computer programming is absent from the literature*. Despite this modification, the objective was not met as there are different viewpoints in the literature about why the statistical evidence is missing (Vogel *et al.*, 2006; Means *et al.*, 2010). Although a very brief discussion was provided (see Chapter 2 Section 2.3.1), the exact reasons why the statistical evidence is missing in games and learning computer programming is perceptual, and hence cannot be listed.

Having achieved the first three objectives, it was decided that the research question is not necessarily focused on the computational thinking skills identified in the literature (i.e. conditional logic, building algorithms, simulation, debugging and socialising). Additionally, it was essential to narrow down the term "learning computer programming" in order to limit the research question with a defined range of introductory programming constructs work (i.e. programming sequence, functions, decision making and loops). Therefore, the research question was refined:

"Can a serious game be designed to support the development of computational thinking skills through the medium of learning how key introductory programming constructs work?"

Having improved the research question, *Program Your Robot* was developed in order to achieve the fifth objective that is to *design a new game specifically for encouraging users to think computationally and learn how computer programming constructs work* (see Chapter 3 Section 3.3).

The sixth objective was *to create an experimental design and conduct a series of rigorous studies to assess the educational impact of Program Your Robot on students*. At this stage, the main research question was revisited again and divided into eight different sub research questions in order to ground these into the experimental design of the rigorous studies. This objective was achieved by conducting a pilot study and three rigorous studies on different target groups (see Chapter 5 Sections 5.1 and 5.2).

The final objective was *to provide a detailed statistical analysis and evaluation of data collected from the structured rigorous studies*. The feedback obtained from the pilot study was

incorporated into *Program Your Robot* and the data gathered from the rigorous studies is analysed in considerable detail. Therefore, this objective was also successfully completed.

Overall, six out of seven objectives were met in this research. During the research, the main research question was refined after a critical review of the literature and it is divided into eight different components in order to merge it with the experimental design of the rigorous studies.

8.3 Main Contributions

There are two types of contributions in this research. The first one is the *modelling contributions* which are related to the approach of developing a structured game model specifically for learning how programming constructs work at the computational thinking level. The second one is the *statistical contributions* which provide the missing rigorous statistical evidence in the literature. These contributions are listed below.

8.3.1 Modelling contributions

1) A new game model called the *interaction – feedback loop* is developed based on the previous research in game based learning particularly on building on the work of Garris, Ahlers & Driskell (2002). The model proposed the rationale that learning material should be presented as in-game elements and must be an integral part of the game-play in a serious game. The *interaction – feedback loop* model was specifically created for developing computational thinking skills for the purpose of learning computer programming constructs rather than being a generic solution onto which were built different learning content. As a very recent example, Hong *et al.* (2013) used the model of Garris *et al.* (2002) to design an archaeology video game for high school students in order to attract them to science. They conducted a study with 80 students from different high schools in order to measure the impact of their game. Although they reported statistical results about the increased interest and motivation of students, it is not clear if/what students learned from the game environment.

2) A serious game named *Program Your Robot* was developed based on the *interaction – feedback loop* model. The game was designed as a step-wise refinement approach to practise and learn how key introductory programming constructs (i.e. programming sequence, functions, decisions and loops) work at the operational level of abstraction

rather than at a procedural level. Additionally, the game is freely available online for anyone to play (http://www.programyourrobot.com). A series of game activities that players can experience in *Program Your Robot* are also discussed to show how students can develop their skills in computational thinking through playing the game (Kazimoglu *et al.*, 2012a).

3) A series of guidelines were gathered from various resources in the literature, and were used to inform the design of *Program Your Robot* (see Chapter 2 Section 2.5 for the guidelines). The guidelines identify the important points that should be considered when designing serious games for the purpose of learning how programming constructs work through game-play. The guidelines were originally individual viewpoints in the literature and the contribution here was combining, categorising and supporting these viewpoints in order to provide a guide for designing games specifically for learning computer programming constructs. The guidelines are published and part of the game based learning literature (Kazimoglu *et al.*, 2011).

8.3.2 Statistical contributions

Based on the statistical analysis of the data collected from the two studies conducted in higher education (i.e. the Cyprus and the Greenwich studies), the following findings were found. These findings are validated both internally and externally and therefore, there is evidence to support that the list below applies to introductory programming students.

1) Playing *Program Your Robot* significantly increased students' perception of their

a) intrinsic motivation to learn computer programming;
b) attitude to learn computer programming through playing games;
c) knowledge regarding how key computer programming constructs (i.e. programming sequence, functions, decisions and loops) work;
d) problem solving abilities;
e) ability to visualise programming constructs from given problems.

2) Playing *Program Your Robot* significantly decreased students' perception of the difficulty of learning computer programming.

3) There are strong, significant and linear correlations in *Program Your Robot* between

a) conditional logic and algorithmic thinking;
b) conditional logic and simulating solutions;
c) algorithmic thinking and simulating solutions;
d) ability to visualise programming constructs and problem solving abilities;
e) ability to visualise programming constructs and programming knowledge gained from the game environment;
f) problem solving abilities and programming knowledge gained from the game environment;

4) It was found that *Program Your Robot* significantly encompasses 3 (conditional logic, algorithmic thinking, simulation) out of 5 core computational thinking skills (i.e. programming sequence, functions, decisions and loops).

5) No strong and significant or linear correlation was identified between key computer programming constructs (i.e. programming sequence, functions, decisions and loops) and computational thinking skills (i.e. conditional logic, algorithmic thinking, simulation, debugging and socialising).

6) No strong and significant or linear correlation was identified between how far students progressed in the game and their computational thinking skills (i.e. conditional logic, algorithmic thinking, simulation, debugging and socialising).

8.4 Limitations of the research

Although this research has achieved its main aims, there were some unavoidable limitations. These limitations are listed below.

1) There was no control group in the conducted studies because

a) the majority of students who participated in the Cyprus and the Greenwich studies had only just registered for their computer programming courses and therefore, it

simply was not possible to separate them into two equal random groups and conduct the studies in proportionately divided environments as the computer programming and computer gaming background of students was unknown.

b) there is no universally agreed way of teaching computational thinking skills to students as this is a relatively new abstract concept and currently there is no conventional way for teaching computational thinking skills;

c) there was no alternative model sought to compare *Program Your Robot* against.

2) As there was no control group in the conducted studies, the experimental structure of the studies was vulnerable to internal threats particularly to regression to the mean threat. The best way to measure the impact of the regression threat is to create a control group and to observe the regression to the mean between the responses. As this was not possible in this research, a multiple linear regression was used to measure the effect of regression in the studies.

3) The experimental design of this research was selected as *the one group pre and post study design without a control group* as it was not possible to select a more reliable experimental design (such as randomised controlled trials) due to the ethical restrictions of the research. The University ethics committee (UREC) insisted on conducting studies where students receive the same experience and none of them was advanced through a specific type of intervention as this would be considered unfair. Therefore, it was practically not possible to divide students into two as the control (i.e. traditional learning group) and the experimental group (the game group) to measure the effectiveness of the game (i.e. *Program Your Robot*). Hence, the ethical restrictions were a limitation for the research as the experimental design of the studies was shaped around the ethical approval from the UREC.

4) *Program Your Robot* was not designed to explicitly support the socialising aspect of computational thinking. The game is designed to allow those players who seek additional challenges to participate in a high score list and show their score in the game. However, this only provides a very limited level of socialising among players. Additionally, only the cooperation aspect of socialising was investigated in all three studies by asking them whether or not they shared ideas and strategies during their game-play. As a result, *Program Your Robot* does not clearly encourage any form of socialising which is a limitation of this study.

5) The ethnicity and the degree programme of participants were collected but not used when verifying the internal validity of findings. Originally, it was aimed to investigate the effect of ethnicity and degree programme of students on the findings of the studies in a similar way to the age-range and gender. However, this decision was abandoned as the Greenwich study was the only study that these were collected. Additionally, it was not aimed to categorise participants according to their races. All three structured studies were based on clarifying whether or not there is a skill acquisition in computational thinking and knowledge gain in computer programming constructs after students played *Program Your Robot*. Although research based on ethnicity or degree programmes is essential in Computer Science, this was not the aim of the studies in this research. Therefore, the effect of ethnicity and the degree programme on the outcome of the studies is not known because they are not considered as an identifying factor in multiple linear regression analysis.

8.5 Future work

There are a couple of routes available as future work in this research.

Firstly, the conducted experimental studies did not have a control group due to the reasons explained in Section 8.3. However, should the investigation of computational thinking skills be removed from the experimental design; a double blind study can be conducted to compare the responses from a control group with the responses obtained from an experimental group. Therefore, a very strong experimental structure could be established to provide even more strong statistical evidence.

Secondly, the effectiveness of the *interaction – feedback loop* was only tested through using *Program Your Robot.* The game model needs to be verified further by designing and testing it with other games. More importantly, the game model can be developed substantially in order to measure whether or not it can be adapted into other areas. As discussed in Chapter 3 Section 3.3, the *interaction – feedback loop* model supports experimental, discovery/inquiry and constructivist approaches to teaching and learning but these are very briefly discussed in this research as the main focus of the conducted studies were skill development and acquisition towards learning introductory programming constructs. Despite the fact that the model is based on promoting questioning and active learning through experimentation, these needs to be explored further in order to establish it into learning theories and instructional strategies. Exploring the *interaction – feedback loop* in a pedagogic context would certainly extract a

more generic model that can be adapted to other disciplines and would also allow researchers to manipulate key variables in the model in order to determine what factors have effect on learner motivation and achievement. Having performed this, the model then can be tested with different user groups to measure its' educational effectiveness in order to provide evidence that it can be used in other areas.

Thirdly, gender, ethnicity and age group factors were not used in the statistical analysis of the studies despite the fact that these were collected from participants. It is crucial to highlight that the aim of this research was to identify whether or not a knowledge gain and skill acquisition happened during the experiment and therefore, these factors were not considered. The age group and gender factors were used in multiple linear regression to measure whether or not these impact the outcome of the studies. However, age groups, gender, degree programmes or ethnicity were not considered when assessing knowledge and skill acquisition from the game environment. Therefore, within a control-experimental structure, an Analysis of Variance (ANOVA) test could be performed to determine whether or not there is any significance in knowledge gain and skill acquisition between male and female students who use *Program Your Robot*. The mean of ethnic, degree programme and age test scores could also be presented through ANOVA test results.

Fourthly, *Program Your Robot* was designed to operate at an operational level of abstraction to practise how programming constructs work and therefore, the game does not produce code in a specific programming language. It is important to indicate that the game's operational stepwise refinement approach could be described in pseudo-code, which could then be utilised with a code generator to produce programming language-specific code. This was not within the current scope of this research, but could be a future development in the game.

Finally, the social aspect of learning was briefly explored in the experimental studies as the main purpose of the research was focused on knowledge gain and skill acquisition before and after students play *Program Your Robot*. Hence, the game was not explicitly designed to encourage cooperation (or competition) during the game-play and therefore, the socialising aspect of computational thinking was only briefly examined in the studies (i.e. Sharing ideas / strategies with a friend was helpful for designing my solutions during the game-play). A possible future work could be exploring how an explicit socialised game-experience could impact students' learning progress. One strategy is to develop *Program Your Robot* further by integrating it into one of the social networks (such as Facebook, Google+). By doing this, *Program Your Robot* can allow players to design solution patterns together, overcome certain challenges in a multi-player mode and/or share their strategies with others. Thus, the social

aspect of learning can be fully investigated.

8.6 Final words

This research sought to answer the main research question: *"Can a serious game be designed to support the development of computational thinking skills through the medium of learning how key introductory programming constructs work?"*. The findings of the conducted studies provided strong evidence that *Program Your Robot* indeed supported students in practising computer programming constructs as well as in developing their computational thinking skills.

There are two major contributions in this research : a) a new game model (i.e. *interaction – feedback loop*) and the serious game implementation of this (i.e. *Program Your Robot*) was developed through the guidelines derived from the previous work in the literature; b) the statistical evidence regarding games as educationally effective solutions for learning introductory computer programming was long absent from the literature and this research provided solid empirical evidence that a serious game (i.e. *Program Your Robot*) is an educationally effective solution for learning how computer programming constructs work at the computational thinking level.

REFERENCES

Abeele, V., Schutter, B., Geurts, L., Desmet, S., Wauters, J., Husson, J., Audenaeren L.V., Broeckhoven F. V., Anneman, J.H. & Geerts, D. (2012). P-III: A Player-Centered, Iterative, Interdisciplinary and Integrated Framework for Serious Game Design and Development. In S. Wannemacker, S. Vandercruysse & G. Clarebout (Eds.), *Serious Games: The Challenge* (280, 82 – 86): Springer Berlin Heidelberg.

Adobe Flash, (2013). Industry-leading authoring environment for producing expressive interactive content, Available at: http://www.adobe.com/products/flash.html (last access: May, 2013).

Ali, A., (2009). A Conceptual Model for Learning to Program in Introductory Programming Courses. *Issues in Informing Science and Information Technology*, 6, 517 – 529.

Ali, A., & Shubra, C. (2010). Efforts to Reverse the Trend of Enrollment Decline in Computer Science Programs. *Issues in Informing Science and Information Technology*, 7, 16.

Alice, (1999) Carnegie Mellon University, Available at: http://www.alice.org/ (last access: May, 2013).

Anewalt, K. (2008). Making CS0 fun: an active learning approach using toys, games and Alice. J. Comput. Small Coll., 23(3), 98 – 105.

Arnab, S., Brown, K., Clarke, S., Dunwell, I., Lim, T., Suttie, N., Louchart, S., Hendrix, M., & de Freitas, S. (2013). The Development Approach of a Pedagogically-Driven Serious Game to support Relationship and Sex Education (RSE) within a classroom setting. *Computers & Education*.

Arnseth, H. C. (2006). Learning to play or playing to learn – A critical account of the models of communication informing educational research on computer gameplay. *Game Studies*, 6(1).

Ater-Kranov, A., Bryant, R., Orr, G., Wallace, S., & Zhang, M. (2010). Developing a community definition and teaching modules for computational thinking: accomplishments and challenges. *In Proceedings of the 2010 ACM conference on Information technology education*, 143 – 148.

Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence Computer

Science introductory course students to persist in the major. *SIGCSE Bulletin*, 41(1), 153–157.

Barnes, T., Powell, E., Chaffin, A., & Lipford, H. (2008). Game2Learn: improving the motivation of CS1 students. *In Proceedings of the 3rd international conference on Game development in Computer Science education* ,1 – 5.

Barnes, T., Richter, H., Powell, E., Chaffin, A., & Godwin, A. (2007). Game2Learn: building CS1 learning games for retention. *In ACM SIGCSE Bulletin*, 39 (3), 121 – 125. ACM.

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. Learning & Leading with Technology, 38(6), 20 – 23.

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. *Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer Science education*.

Basu, S., Dickes, A., Kinnebrew, J. S., Sengupta, P., & Biswas, G. (2013). CTSiM: A Computational Thinking Environment for Learning Science through Simulation and Modeling. *In The 5th International Conference on Computer Supported Education*.

Bayliss, J. D., & Schwartz, D. I. (2009). Instructional design as game design. *In Proceedings of the 4th International Conference on Foundations of Digital Games*.

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for Computer Science students: some thoughts and observations. *SIGCSE Bull.*, 37(2), 103 – 106.

Beck, H. C. (1931). Mr Beck's Underground Map. Capital Transport.

Bennedsen, J., & Caspersen, M. E. (2012). Persistence of elementary programming skills. *Computer Science Education*, 22(2), 81 – 107.

Bennedsen, J., Caspersen, M. E., & Kölling, M. (Eds.). (2008). Reflections on the Teaching of Programming: Methods and Implementations (Vol. 4821). *Springer*.

Bennedsen J. & Caspersen M.E., (2007), Failure rates in introductory programming. *SIGCSE Bull.*, 39, 121 – 127.

Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bull.*, 38(2), 39 – 43.

Bergeron, B. (2006). Developing Serious Games (Game Development Series).

Berland, M & Lee, V. R., (2011), Collaborative Strategic Board Games as a Site for Distributed Computational Thinking, *International Journal of Game-Based Learning* (IJGBL), 1(2), 65 – 81.

Blum, L., & Cortina, T. J. (2007). CS4HS: an outreach program for high school CS teachers. *In ACM SIGCSE Bulletin*, 39 (1), 19 – 23. ACM.

Bonar, J., & Soloway, E. (1983). Uncovering the principles of novice programming. Paper presented at the *Tenth ACM SIGACTSIGPLAN Symposium on Principles of Programming Languages*.

Bowden, J. (2005). Reflections on the phenomenographic research process. In Doing Developmental Phenomenography, J. Bowden & P. Green (Eds). Qualitative Research Methods Series. *RMIT University Press*.

Box, G E. P. & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society*, (2), 211 – 252.

Boyle, R., Carter, J., & Clark, M. (2002). What makes them succeed? Entry, progression and graduation in Computer Science. *Journal of Further and Higher Education*, 26(1), 3-18.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *In Proceedings of the 2012 annual meeting of the American Educational Research Association*.

Brézillon, P. (2003). Using context for supporting users efficiently. In System Sciences, 2003. *Proceedings of the 36th Annual Hawaii International Conference* on, 1-9. IEEE.

Brézillon, P., Gentile, C., Saker, I., & Secron, M. (1997). SART: A system for supporting operators with contextual knowledge.

Bruce, C., McMahon C., Buckingham L., Hynd J., Roggenkamp M., & Stoodly I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, 3, 143–160.

Bruckman, A., Jenson, C., and DeBonte, A. (2002). Gender and Programming Achievement in a CSCL Environment. *In Proc. CSCL 2002*, 119 - 227.

Calder, B.J., Phillips L.W. & Tybout A.M. (1982). The concept of External Validity. *Journal of Consumer Research*, 9, 240 – 244.

Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental evaluation of teaching recursion in a video game. *Paper presented at the Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*.

Chang, W. C., & Chou, Y. M. (2008). Introductory c programming language learning with game-based digital learning. *In Advances in Web Based Learning-ICWL 2008* , 221 – 231. Springer Berlin Heidelberg.

Chang, J. K. W., Dang, L. H., Pavleas, J., McCarthy, J. F., Sung, K., & Bay, J. (2012). Experience with Dream Coders: developing a 2D RPG for teaching introductory programming concepts. *Journal of Computing Sciences in Colleges*, 28(1), 227 – 236.

Charsky, D. (2010). From edutainment to serious games: A change in the use of game characteristics. Games and Culture, 5(2), 177 – 198.

Choua, C. & Tsaib, M. (2007) Gender differences in Taiwan high school students' computer game playing. *Computers in Human Behavior*, 23(1), 812 – 824.

Christensen, R. (2011). Plane Answers to Complex Questions: The Theory of Linear Models, Springer; 4th ed.

Clark, D. (2009). Games and eLearning. Caspian Learning. Available at: http://www.caspianlearning.com/Whtp_caspian_games_1.1.pdf (last access: May, 2013).

Coelho, A., Kato, E., Xavier, J., & Gonçalves, R. (2011). Serious game for introductory programming. In Serious Games Development and Applications, 61 – 71. Springer Berlin Heidelberg.

Colobot (2007). A new 3D real time game of strategy and adventure. Available at: http://www.ceebot.com/colobot/index-e.php (last access: May, 2013).

Connolly, T. M., Stansfield, M. & Hainey T. (2007). An application of games-based learning within software engineering, *British Journal of Educational Technology*, 38(3), 416 – 428.

Cooper, S., Dann, W., & Puasch, R. (2000). Alice: A 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15(5), 107 – 116.

Costandi, Mo. (2011). Video-game studies have serious flaws. Nature.com. Available at: http://www.nature.com/news/2011/110916/full/news.2011.543.html (last accessed: May, 2012).

Coull, N.J. & Duncan, I.M.M. (2011). Emergent requirements for supporting introductory programming. *ITALICS*, 10(1), 78 – 85.

Dalal, N., Dalal, P., Kak, S., & Antonenko, P. (2009). Rapid digital game creation for broadening participation in computing and fostering crucial thinking skills. *International Journal of Social and Humanistic Computing*, 1(2), 123 – 137.

Dallal, G. E. (2000). The Regression Effect / The Regression Fallacy. Retrieved from: http://www.jerrydallal.com/LHSP/regeff.htm (last access: January, 2013).

de Freitas, S. & Oliver, M. (2006). How can exploratory learning with games and simulations within the curriculum be most effectively evaluated. *Computers and Education* 46 (3), 249–264.

de Raadt, M., (2007). A review of Australasian investigations into problem solving and the novice programmers. *Computer Science Education*, 17(3), 201 – 213.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of Computer Science concepts? *Computers & Education*, 58(1), 240 – 249.

Denning, P. J. (2009). The profession of IT Beyond computational thinking, *Commun. ACM*, 52, 28 – 30.

Devlin, K. (2003). Why universities require Computer Science students to take math, *Commun. ACM*, 46, 9, 37 – 39.

Dierbach, C., Hochheiser, H., Collins, S., Jerome, G., Ariza, C., Kelleher, T., ... & Kaza, S. (2011, March). A model for piloting pathways for computational thinking in a general education curriculum. *In Proceedings of the 42nd ACM technical symposium on Computer science education*, 257 – 262.

Eagle, M., & Barnes, T., (2009). Experimental evaluation of an educational game for improved learning in introductory computing. SIGCSE Bull., 41(1), 321 – 325.

Egenfeldt-Nielsen, S. (2005). Beyond edutainment: Exploring the educational potential of computer games. *Copenhagen, Denmark: University of Copenhagen*.

Engadget, (2011). Available at : http://www.engadget.com/2011/11/09/adobe-confirms-flash-player-is-dead-for-mobile-devices/

Entwistle, N. (1997). Introduction: phenomenography in higher education. *Higher Education Research & Development*, 16, 127 – 134.

Explorable, 2011 External Validity, Available at: http://explorable.com/external-validity (last access: March, 2013).

Farrel, P.J. & Stewart, K.R. (2006). Comprehensive Study of Tests For Normality And Symmetry: Extending The Spiegelhater Test. *Journal of Statistical Computation and Simulation*, 76 (9), 803 – 816.

Feldgen, M., & Clua, O. (2004). Games as a motivation for freshman students learn programming. *Paper presented at the Frontiers in Education*, FIE 2004.

Fletcher, G. H. L., & Lu, J. J., (2009). Education Human computing skills: rethinking the K-12 experience. *Commun. ACM*, 52(2), 23 – 25.

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & Gaming*, 33(4), 441–467.

Gee, J. P. (2005). Game-like learning: An example of situated learning and implications for opportunity to learn. Available at : http://www.academiccolab.org/resources/documents/Game-Like%20Learning.rev.pdf (last access: April, 2013)

Gee, J. P. (2003). What video games have to teach us about learning and literacy. *Comput. Entertain.*, 1(1), 20-20.

Gomes, A. & Mendes, A. J. (2007). Learning to Program - Difficulties and Solutions, *International Conference on Engineering Education – ICEE 2007*, 283 – 287.

Graven, O. H. & MacKinnon, L. M., (2008). Prototyping a Games-Based Environment for

Learning. *E-learn 2008 World Conference on E-learning*, 2661 – 2668.

Guzdial, M., (2012). Research Questions in Computing Education. Computing Education Blog. Available at: http://computinged.wordpress.com/2012/05/03/blog-post-999-research-questions-in-computing-education/ (last access: May, 2013).

Guzdial, M. (2011). "A Definition of Computational Thinking from Jeannette Wing." Retrieved from: http://computinged.wordpress.com/2011/03/22/a-definition-of-computational-thinking-from-jeanette-wing/ (last access: January, 2013).

Guzdial, M., (2008). Paying the way for computational thinking. *Commun. ACM*, 51(8), 25-27.

Guzdial, M., (2004). Programming environments for novices. In S. Fincher and M. Petre (Eds.), *Computer Science Education Research*, 127-154. Lisse, The Netherlands: Taylor & Francis.

Hainey, T., Connolly, T., Stansfield, M., & Boyle, L. (2011). The use of computer games in education: A review of the literature. Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches, *IGI Global*.

Hamari, J. & Eranti, V. (2011). Framework for Designing and Evaluating Game Achievements., *Proceedings of Digra 2011 Conference: Think Design Play, Hilversum, Netherlands*, 14-17, 2011.

Hauke, J. & Kossowski, T. (2011). Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data, *Quaestiones Geographicae*, 30(2), 87 – 93.

Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. Computers & Education, 54(4), 1127-1136.

Hazzan, O. (2003). How Students Attempt to Reduce Abstraction in the Learning of Mathematics and in the Learning of Computer Science, *Computer Science Education*, 13 (2), 95 – 122.

Hill, C., Corbett, C., & St Rose, A. (2010). Why So Few? Women in Science, Technology, Engineering, and Mathematics. American Association of University Women. 1111 Sixteenth

Street NW, Washington, DC 20036.

Hong, J. C., Hwang, M. Y., Chen, Y. J., Lin, P. H., Huang, Y. T., Cheng, H. Y., & Lee, C. C. (2013). Using the saliency-based model to design a digital archaeological game to motivate players' intention to visit the digital archives of Taiwan's natural science museum. *Computers & Education*.

Hossain, M, Z., (2011) The Use of Box-Cox Transformation Technique in Economic and Statistical Analyses, *Journal of Emerging Trends in Economics and Management Sciences (JETEMS)*, 2(1), 32 – 39.

How2stats, 2011. Spearman Rank Correlation. Retrieved from: http://how2stats.blogspot.co.uk/2011/09/spearman-rank-correlation.html (last access: January 2013)

Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. *Paper presented at the Game Design and Tuning Workshop at the Game Developers Conference*.

Hwang, W. Y., Shadiev, R., Wang, C. Y., & Huang, Z. H. (2012). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. Computers & Education, 58(4), 1267-1281.

Hyvärinen, A. and Oja, E., (2000). Independent Component Analysis: A Tutorial, *Neural Networks*, 13(4-5):411-430. Retrieved from: http://cis.legacy.ics.tkk.fi/aapo/papers/IJCNN99_tutorialweb/ (last access: February 2013)

Ibrahim, R., Yusoff, R. C. M., Mohamed, H., & Jaafar, A. (2010). Students Perceptions of Using Educational Games to Learn Introductory Programming. *Computer and Information Science*, 4(1).

Investopedia, (2013a). Leptokurtic definition in Statistics, Available at: http://www.investopedia.com/terms/l/leptokurtic.asp (last access: February 2013).

Investopedia, (2013b), Multiple linear regression, Available at: http://www.investopedia.com/terms/m/mlr.asp (last access: March, 2013)

Jenkins, T. (2001), The motivation of students of programming, *SIGCSE Bull.*, 33 (3), 53 – 56.

Jenkins, T. (2002). On the difficulty of learning to program. *In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 53 – 57.

Jenson, J., Castell, S. d., & Fisher, S. (2007). Girls playing games: rethinking stereotypes, *Paper presented at the Proceedings of the 2007 conference on Future Play*.

Joiner, R., Iacovides, J., Owen, M., Gavin, C., Clibbery, S., Darling, J., & Drew, B. (2011). Digital Games, Gender and Learning in engineering: Do females benefit as much as males?. Journal of Science Education and Technology, 20(2), 178-185.

Kafai, Y. B. (1995). Minds in play: Computer game design as a context for children's learning. *Routledge*.

Karel The Robot. (1981). Robot simulation that affords a gentle introduction to computer programming. Available at: http://karel.sourceforge.net/ (last access : April, 2013).

Kato, P. M., Cole, S. W., Bradlyn, A. S., & Pollock, B. H. (2008). A video game improves behavioral outcomes in adolescents and young adults with cancer: a randomized trial. *Pediatrics*, 122(2), e 305- e 317.

Kazimoglu, C., Kiernan, M., Bacon, L. & MacKinnon, L. (2012a) Learning Programming at the Computational Thinking Level via Digital Game-Play, *Procedia Computer Science*, 9, 522 – 531.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012b). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia-Social and Behavioral Sciences*, 47, 1991 – 1999.

Kazimoglu, C., Kiernan, M., Bacon, & MacKinnon L. (2011) Understanding Computational Thinking before Programming: Developing Guidelines for the Design of Games to Learn Introductory Programming through Game-Play, *International Journal of Game Based Learning (IJGBL)*, *IGI Global*, 30 – 52.

Kazimoglu, C., Kiernan, M., Bacon, L. & Mackinnon, L. (2010). Developing a game model for computational thinking and learning traditional programming through game-play, J. Sanchez and K. Zhang, (eds.), *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, 1378 – 1386.

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling Alice motivates middle school girls to learn computer programming. *Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems*.

Keskin, S. (2006). Comparison of Several Univariate Normality Tests Regarding Type I Error Rate and Power of the Test in Simulation Based Small Samples, *Journal of Applied Science Research*, 2(5), 296 – 300.

Ketelhut, D. J., Clarke, J., Dede, C., Nelson, B., & Bowman, C. (2005). Inquiry teaching for depth and coverage via multi-user virtual environments. *Paper presented at the National Association of Research in Science Teaching.*

Kinnunen, P., & Malmi, L., (2006). Why students drop out CS1 course? Paper presented at the Proceedings of the second international workshop on Computing education research, *ICER'06*, 97-108.

Kinnunen, P., & Simon, B. (2012). My program is ok – am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1), 1 – 28.

Knight, J. F., Carley, S., Tregunna, B., Jarvis, S., Smithies, R., de Freitas, S., Dunwell, I., & Mackway-Jones, K. (2010). Serious gaming technology in major incident triage training: A pragmatic controlled trial. *Resuscitation*, 81(9), 1175 – 1179.

Masters, G., Ramsden, P., & Stephanou, A. (1992). Displacement, velocity, and frames of reference: Phenomenographlc studies of students' understanding and some implications for teaching and assessment. *Am. J. Phys*, 60(3).

Kowalski, R. (2011). Computational logic and human thinking: how to be artificially intelligent. Cambridge University Press.

Kramer, J. (2007). Is abstraction the key to computing? *Commun. ACM*, 50(4), 36 – 42.

Kumar, D. D., & Sherwood, R. D. (2007). Effect of a problem based simulation on the conceptual understanding of undergraduate science education students. *Journal of Science Education and Technology*, 16(3), 239 – 246.

Ladd, B., & Harcourt, E. (2005). Student competitions and bots in an introductory programming course. *Journal of Computing in Small Colleges*, 20(5), 274–284.

Laerd Statistics (2012a), Wilcoxon Signed-Rank Test using SPSS, Available at: https://statistics.laerd.com/spss-tutorials/wilcoxon-signed-rank-test-using-spss-statistics.php (last access: May, 2013).

Laerd Statistics (2012b), Spearman's Rank-Order Correlation, Available at: https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php (last access: May, 2013).

Lahtinen, E., Mutka, K. A. & Jarvinen, H. M. (2005). A study of the difficulties of novice programmers. In Proceedings of the 10th Annual SIGSCE Conference on Innovation and Technology in Computer Science Education (*ITICSE 2005*), 14 – 18.

Lave, J. & Wenger, E. (1991). Situated Learning. Legitimate peripheral participation, Cambridge: University of Cambridge Press.

Lebow, D. (1993). Constructivist values for instructional systems design: Five principles toward a new mindset. Educational Technology Research and Development, 41(3).

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2(1), 32-37.

Leutenegger, S. & Edgington, J. (2007), A games first approach to teaching introductory programming, *SIGCSE Bull.*, 39 (1), 115 – 118.

Li, F. W., & Watson, C. (2011). Game-based concept visualization for learning programming. *In Proceedings of the third international ACM workshop on Multimedia technologies for distance learning*, 37 – 42. ACM.

Light-Bot. (2008). Control a robot by giving commands to it. Available at: http://armorgames.com/play/2205/light-bot (last access: April, 2013).

Lister, R. (2011). Programming, Syntax and Cognitive Load. ACM Inroads, 2(2), 21 – 22.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., ... & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4), 119-150.

Liu, C. C., Cheng, Y. B., & Huang, C. W. (2011). The effect of simulation games on the

learning of computational problem solving. *Computers & Education*, 57(3), 1907-1918.

Loftus, C., Thomas, L., & Zander, C. (2011). Can graduating students design: revisited. *Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer Science education.*

Long, J. (2007). Just for fun: Using programming games in software programming training and education-A field study of IBM robocode community. *Journal of Information Technology Education*, 6, 279 – 290.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking about computational thinking. Paper presented at the Proceedings of the 40th ACM technical symposium on Computer Science education.

Ma, Y., Williams, D., Prejean, L., & Richard, C. (2007). A research agenda for developing and implementing educational computer games. *British Journal of Educational Technology*, 38, 513–518.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. *SIGCSE Bull.*, 40(1), 367 – 371.

Mann, L. (2010). Critical Features of Phenomenography, Retrieved from: http://aaee-scholar.pbworks.com/w/page/1177079/Research%20Method%20-%20Phenomenography (last access: January, 2013).

Marton F., & Booth S. (1997). Learning and Awareness. *New Jersey: Lawerence Erlbaum Associaates*.

Marton, F. (1994). Phenomenography. In T. Husen & T. N. Postlethwaite (Eds.), *The International Encyclopaedia of Education*, 2 (8), 4424 - 4429.

Marton, F. (1986). Phenomenography – A research approach investigating different understandings of reality. *Journal of Thought*, 21(2), 28-49.

Marton, F. (1981). Phenomenography – describing conceptions of the world around us, *Instructional Science*, 10 (19821), 177 -200.

Mason, R., Cooper, G., & Comber, T. (2011). Girls get it. *ACM Inroads*, 2(3), 71-77.

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM*

Comput. Surv., 13(1), 121-141.

McAllister, G. & Alexander, S., (2008). Key aspects of teaching and learning in Computer Science. In H. Fry, S. Ketteridge & S. Marshall (Eds.), A handbook for teaching and learning in higher education, 282-295. Third Edition, New York: Taylor & Francis.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4), 125-180.

McKenna, P., & Laycock, B. (2004). Constructivist or instructivist: pedagogical concepts practically applied to a computer learning environment. *In ACM SIGCSE Bulletin*, 36(3), 166 – 170. ACM.

Means, B., Toyama, Y., Murphy, R., Bakia, M., & Jones, K. (2010). Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in scratch. In Proceedings of the 16th annual joint conference on Innovation and technology in Computer Science education, 168 – 172. ACM.

Mendes, M. & Pala, A. (2003). Type I Error Rate and Power Of Three Normality Tests. Pakistan Journal of Information and Technology, 2(2), 135 – 139.

Michael, D. R., & Chen, S. L. (2005). Serious games: Games that educate, train, and inform. *Muska & Lipman/Premier-Trade*.

Mitamura, T., Suzuki, Y., & Oohori, T. (2012). Serious games for learning programming languages. *In Systems, Man, and Cybernetics (SMC)*, 2012 IEEE International Conference on, 1812 – 1817. IEEE.

Moore, D.S., MacCabe, G.P. & Craig, B.A. (2009). Introduction to the practice of statistics, New York, *NY: Freeman.*

Moreno-Ger, P., Burgos, D., Martínez-Ortiz, I., Sierra, J. L., & Fernández-Manjón, B. (2008). Serious game design for online education. *Computers in Human Behavior*, 24, 2530–2540.

Moursund, D. (2009). Computational Thinking. IAE-pedia. Available at

http://iaepedia.org/Computational_Thinking (last access: May 2013).

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. P. (2011). Experimental feedback on Prog&Play: a serious game for programming practice. *In Computer Graphics Forum*, 30 (1), 61 – 73. Blackwell Publishing Ltd.

Muratet, M., Torguet, P., Jessel, J. P., & Viallet, F. (2009). Towards a serious game to help students learn computer programming. *International Journal of Computer Games Technology*.

NCTI, (2012). National Center for Technology Innovation, Retrieved from: http://www.nationaltechcenter.org/index.php/products/at-research-matters/quasi-experimental-study/ (last access: January, 2013).

Nienaltowski, M.H., Pedroni, M., Meyer, B., (2008) Compiler error messages : what can help novices? *Proceedings of the 39th SIGCSE technical symposium on Computer Science education*, 168 – 172.

O'Neil, H., Wainess, R., & Baker, E. (2005). Classification of learning outcomes evidence from the computer games literature. *Curriculum Journal.*, 16(4), 455–474.

Or-Bach, R., & Lavy, I. (2004). Cognitive activities of abstraction in object orientation: an empirical study. ACM SIGCSE Bulletin, 36(2), 82 – 86.

Orgill, M. K. (2002). Phenomenograpy. Retrieved from: http://www.minds.may.ie/~dez/phenom.html (last access: January, 2013).

Ornek, F. (2008). An overview of a theoretical framework of phenomenography in qualitative education research: An example from physics education research. *Asia-Pacific Forum on Science Learning and Teaching*, 9 (2), Retrieved from: http://www.ied.edu.hk/apfslt/v9_issue2/ornek/index.htm (last access: January, 2013).

Orr, G. (2009). Computational thinking through programming and algorithmic art. *In Proceedings of the SIGGRAPH Talks* (p. 31).

Osborne, J, W. (2010). Improving your data transformations: Applying the Box-Cox transformation, Practical Assessment, Research & Evaluation, 15(12), Retrieved from: http://pareonline.net/pdf/v15n12.pdf (last access: February, 2013).

Osborne, M. J. (2004). An introduction to game theory (Vol. 3, No. 3). *New York: Oxford University Press.*

Quinn, C. N. (2005). Engaging Learning: Designing e-Learning Simulation Games. Pfeiffer.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations, *International Journal of Computers for Mathematical Learning,* Vol. 1, No. 1, 95 – 123.

Papastergiou, M. (2009). Digital Game-Based Learning in high school Computer Science education: Impact on educational effectiveness and student motivation, *Computers & Education*, 52, 1 – 12.

Perkovic, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational thinking across the curriculum. *Paper presented at the Proceedings of the fifteenth annual conference on Innovation and technology in Computer Science education*, 123 – 127. ACM.

Pivec, M., Dziabenko, O., & Schinnerl, I. (2003). Aspects of game-based learning. I-Know, 3, 217–224.

Pivec, M., Koubek, A., & Dondi, C. (Eds.). (2004). Guidelines for game-based learning. Lengerich, Germany: Pabst Verlag.

Pratchett, R. (2005). Gamers in the UK: Digital play, digital lifestyles. Retrieved from http://open.bbc.co.uk/newmediaresearch/files/BBC_UK_Games_Research_2005.pdf (last access: December 2012).

Prensky M. (2006). Computer games and learning: digital game-based learning. In: Raessens J, Goldstein J, editors. *Handbook of computer games studies*. *MIT Press*, 59 – 79.

Prensky, M. (2004). The motivation of gameplay. *Horizon*, 10(1).

Prensky, M. (2001). Digital game based learning. New York, NY: McGraw-Hill.

Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Journal of Computing Sciences in Colleges*, 25(5), 66-71.

Rajaravivarma, R., (2005). A games-based approach for teaching the introductory programming course. SIGCSE Bull., 37(4), 98-102.

Rask, K. (2010). Attrition in STEM fields at a liberal arts college: The importance of grades and pre-collegiate preferences, Economics of Educaion, 29(6), 892 – 900.

Razali, N.M. & Wah, Y.B. (2011). Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests, *Journal of Statistical Modeling and Analytics*, 1(1), 21 – 33.

Repenning, A., Webb, D. & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools, *Proceedings of the 41st ACM technical symposium on Computer Science education*. 265 – 269. ACM.

Richardson, J, T, E. (1999). The Concepts and Methods of Phenomenographic Research, *Review of Educational Research*, 69(1), 53-82.

Robertson, J. & Howells, C. (2008). Computer game design: Opportunities for successful learning. *Computers & Education*, 50 (2), 559 – 578.

Robocode. (2001). Build the best - destroy the rest! Available at: http://robocode.sourceforge.net (last accessed May, 2013).

RoboMind. (2005). Availiable at: http://www.robomind.net/en/index.html (last accessed: April, 2013).

Robozzle. (2010). An addictive robot-programming puzzle game. Available at: http://www.robozzle.com (last access: April, 2013).

Salen, K., & Zimmerman, E. (2003). Rules of play: Game design fundamentals. Cambridge, MA: MIT Press.

Sancho, P., Gomez-Martin, P. P., & Baltasar, F. M. (2008). Multiplayer role games applied to problem based learning. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, 69 – 70.

Savery, J. R., & Duffy, T. M. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, 35(5), 31–38.

Schell, J. (2008). The art of game design: A book of lenses. San Francisco, CA: Morgan Kauffman.

Scratch. (2010). A programming language for everyone: Create interactive stories, games, music and art – and share them online. Available at: http://scratch.mit.edu (last access: April 2013).

Shashaani, L. (1997). Gender Differences in Computer Attitudes and Use Among College Students, *Journal of Educational Computing Research*, 16(1), 37 – 52.

Sherman, P.J. (2010). Tips for Recognizing and Transforming Non-normal Data. Retrieved from: http://www.isixsigma.com/tools-templates/normality/tips-recognizing-and-transforming-non-normal-data/ (last access: February 2013).

Shuttleworth, M. (2009). External Validity. Retrieved from: http://explorable.com/external-validity.html (last access: January, 2013).

Sid Meier's Railroads (2006). Available at: http://www.2kgames.com/railroads/railroads.html (last access: January, 2013).

Slack, M.K. & Draugalis, J. R. (2001). Establishing the Internal and External Validity of Experimental Studies, *American Journal of Health-System Pharmacy*, 58(22). Retrieved from : http://www.medscape.com/viewarticle/414875 (last access : January, 2013).

Soloway, E. (1993). Should we teach students to program?, Communication of the ACM, vol. 36 (19), 21 – 24.

Song, P. X. K. (2007). Correlated Data Analysis: Modelling, Analytics, and Applications, *Springer Series in Statistics.*

Sprague, P., & Schahczenski, C. (2002). Abstraction the key to CS1. Journal of Computing Sciences in Colleges, 17(3), 211-218.

SRM (Social Research Methods). (2006). Available at: http://www.socialresearchmethods.net/kb/intsing.php (last access: January, 2013).

Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. W., Goldstein, D. S., & Nordlinger, J. (2011). Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses. Education, *IEEE Transactions on*, 54(3), 416 – 427.

Sung, K. (2009). Computer games and traditional CS courses. *Commun. ACM*, 52(12).

Sung, K., Panitz, M., Wallace, S., Anderson, R., & Nordlinger, J. (2008). Game-themed programming assignments: the faculty perspective. *ACM SIGCSE Bulletin*, 40(1), 300 – 304.

SurveyMonkey. (1999). Web survey development company. Available at: http://www.surveymonkey.com (last access: January, 2013).

Suttie, N., Louchart, S., Lim, T., Macvean, A., Westera, W., Brown, D., & Djaouti, D. (2012). Introducing the "Serious Games Mechanics" A Theoretical Framework to Analyse Relationships Between "Game" and "Pedagogical Aspects" of Serious Games. *Procedia Computer Science,* 15, 314 – 315.

Tinker. (2008). Microsoft's puzzle video game in which the player controls a robot through various mazes and obstacle courses. Available at: http://www.microsoft.com/games/en-gb/Games/Pages/tinker.aspx (last access: May 2013).

Trigwell, K. (2000). A phenomenographic interview on phenomenography. In: J. A. Bowden & E. Walsh (Eds.). Qualitative Research Methods Series. *RMIT University Press*, 62-82.

Tsukamoto, H., Nagumo, H., Takemura, Y. & Nitta N., (2012), Change of Students' Motivation in an Introductory Programming Course for Non-computing Majors, *Advanced Learning Technologies (ICALT), 2012 IEEE 12th International Conference*.

Ventura, P. R. J., (2005). On the origins of programmers: identifying predictors of success for an objects first cs1, *Computer Science Education*, 15(3), 223 – 243.

Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: A meta-analysis. *Journal of Educational Computing Research*, 34(3), 229-243.

Wang, M., & Hu, X. (2011). SoccerCode: A Game System for Introductory Programming Courses in Computer Science. *In Proceedings of the World Congress on Engineering and Computer Science* (Vol. 1).

Weller, M. P., Do, E. Y.-L., & Gross, M. D. (2008). Escape machine: teaching computational thinking with a tangible state machine game. *Paper presented at the Proceedings of the 7th international conference on Interaction design and children.*

Whitton, N. (2007). Motivation and computer game based learning. *In Proceedings of the 27th Australasian Society for Computers in Learning in Tertiary Education*, 1063 – 1067.

Wilcox, R.R. (2011). Some practical reasons for reconsidering the Kolmogorov–Smirnov test. *British Journal of Mathematical and Statistical Psychology*, 50 (1).

Wilson, B. C. (2002). A study of factors promoting success in Computer Science including gender differences. Computer Science Education, 12(1-2), 141-164.

Wing, J. M. (2010). Computational Thinking: What and Why. Retrieved from: http://www.cs.cmu.edu/ourcs/presentations/ct.pdf (last access: May, 2013).

Wing, J. M. (2008). Computational thinking and thinking about computing, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366, 3717 – 3725.

Wing, J.M. (2006). Computational thinking. *Commun. ACM,* 49 (3), 33 – 35.

Yale, 1998, Multiple linear regression, available at: http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm (last access: March, 2013).

Yeh, K.C., (2009). Using an Educational Computer Game as a Motivational Tool for Supplemental Instruction Delivery for Novice Programmers in Learning Computer Programming. Paper presented at the Proceedings of the 20th Annual Society for Information Technology and Teacher Education International Conference.

Zimmermann, L., & Sprung G. (2008) Technology is Female: How Girls Can Be Motivated to Learn Programming and Take up Technical Studies through Adaptations of the Curriculum, Changes in Didactics, and Optimized Interface Design, *International Conference on Engineering Education (ICEE)*, Retrieved from : http://iceehungary.net/download/fullp/full_papers/full_paper454.pdf (last access : January, 2013).

The Cyprus study participant consent form

(a game based learning research study conducted by Cagin Kazimoglu, a PhD candidate at the University of Greenwich, as part of his PhD)

What is this study? What do you want me to do?

The majority of students find computer programming difficult and whilst this is not something everyone will do, computer programming can be beneficial to learn as the use of technology increasingly affects our everyday lives. In recent years, the interest in computer programming has decreased and students are losing their motivation to learn it. For this reason, new ways of teaching computer programming are being researched. One of these new ways is through the use of video games and video game-based technologies (henceforth referred to as games). Following this trend, we have developed a game which we hope will increase the motivation of students to learn programming as well as support their learning process in how computer programming concepts work. We would therefore like to ask you to play this game and provide us with your opinion on whether this works for you or not.

In this study, you will be asked to fill out two online questionnaires by the researcher, first before you play a video game and the second after you played it. The purpose of the first questionnaire is to gather information about your prior experience in computer programming, if any. We also ask you questions about your motivation in learning programming and your attitude (either positive or negative) to the use of games for learning purposes, particularly for computer programming. After filling in the first questionnaire, we will kindly ask you to play a game which is online and available at: http://www.programyourrobot.com. You will have the chance to play the game for up to about 25 - 30 minutes if you wish to. Having played the game, you will be asked to fill out the second online questionnaire which basically involves questions about your game experience. The second questionnaire is aimed at gathering your viewpoints on the game including whether you think it has helped your skills in problem solving to develop. Additionally, we want to seek your views on whether you think this game provokes intrinsic motivation for learning programming and supports how introductory computer programming constructs (particularly in programming sequence, decisions, loops and functions) work. Based on your responses, we plan to undertake an evaluation that will enable us to reflect on how we can improve the delivery of computing programming courses in future as well as games specifically developed for learning programming.

While you are under no obligation to answer any of the questions we would appreciate it if you would provide an answer to all of the questions. We would most likely remove your answers if you skip too many questions; therefore, we would appreciate if you could choose the best answer that suits you in each question. Please be assured that this research is completely confidential and no attempts will be done to identify you. Although you will be asked to enter personal information (such as your age, gender), your answers will only be viewed by the researcher (Cagin Kazimoglu) and will never be disclosed to others (including University staff, your tutors and course instructors/coordinators). You will be given a random unique number at the beginning of the study and this is only asked in the study because it is important that there is the facility to link your responses from the first questionnaire to your responses in the second questionnaire. We will never attempt to identify you from your unique number and your data will be kept for research purposes only. The researcher will be available during the entire study and please do not hesitate to ask him questions should you want to.

Both questionnaires should take about 5-10 minutes to complete and the entire study will be no more than an hour. If at any time you wish to withdraw from the study, please inform the researcher/personal tutor and you will be free to leave, no reasons need to be provided. Please be assured that your choice to participate or not in this study will have no effect on your studies.

What are you going to do with my answers?

The answers you provide will be analysed and the data will be held securely by University of Greenwich in accordance with data protection laws until the research project is completed. We will keep the data for research purposes only and we intend to generate statistical information from this data.

Some of the data we have gathered may be published as part of the project report; however individuals will not be identified and will be kept confidential. All data gathered will be destroyed on completion of the research.

For further information about this study please contact:

Cagin Kazimoglu
PhD candidate
Office : QM 165
Smart Systems Technologies Department,
School of Computing and Mathematical Sciences
University of Greenwich
Old Royal Naval College, London SE10 9LS
Tel: 020 8331 8550
Email: c.kazimoglu@greenwich.ac.uk

Eur Ing Dr Mary Kiernan
Office : QM 330
School of Computing and Mathematical Sciences
University of Greenwich,
Old Royal Naval College, London SE10 9LS
Tel: 020 8331 7974
Email: m.kiernan@greenwich.ac.uk

So, what do I need to do?

STEP 1: Go to pre-study link: https://www.surveymonkey.com/s/pre-game-study, select the choice in each question that fits best to you. (5-10 minutes)

STEP 2: Go to game link: https://www.programyourrobot.com, you can play the game up to about 25-30 minutes.

STEP 3: Go to post-study link: https://www.surveymonkey.com/s/post-game-survey, select the choice in each question that fits you the best. (5-10 minutes)

That's it, you are done!

This research is completely confidential and the IP address of your computer will not be tracked during the study.

I have read the consent form and agree to participate in the study and understand that I am free to withdraw at any time.

Your unique number: .....

Signature: .....

The Greenwich study participant consent form

(a game based learning research study conducted by Cagin Kazimoglu, a PhD candidate at the University of Greenwich, as part of his PhD)

What is this study? What do you want me to do?

The majority of students find computer programming difficult and whilst this is not something everyone will do, computer programming can be beneficial to learn as the use of technology increasingly affects our everyday lives. In recent years, the interest in computer programming has decreased and students are losing their motivation to learn it. For this reason, new ways of teaching computer programming are being researched. One of these new ways is through the use of video games and video game-based technologies (henceforth referred to as games). Following this trend, we have developed a game which we hope will increase the motivation of students to learn programming as well as support their learning process in how computer programming concepts work. We would therefore like to ask you to play this game and provide us with your opinion on whether this works for you or not.

In this study, you will be asked to fill out two online questionnaires by the researcher, first before you play a video game and the second after you played it. The purpose of the first questionnaire is to gather information about your prior experience in computer programming, if any. We also ask you questions about your motivation in learning programming and your attitude (either positive or negative) to the use of games for learning purposes, particularly for computer programming. After filling in the first questionnaire, we will kindly ask you to play a game which is online and available at: http://www.programyourrobot.com. You will have the chance to play the game for up to about 25 - 30 minutes if you wish to. Having played the game, you will be asked to fill out the second online questionnaire which basically involves questions about your game experience. The second questionnaire is aimed at gathering your viewpoints on the game including whether you think it has helped your skills in problem solving to develop. Additionally, we want to seek your views on whether you think this game provokes intrinsic motivation for learning programming and supports how introductory computer programming constructs (particularly in programming sequence, decisions, loops and functions) work. Based on your responses, we plan to undertake an evaluation that will enable us to reflect on how we can improve the delivery of computing programming courses in future as well as games specifically developed for learning programming.

While you are under no obligation to answer any of the questions we would appreciate it if you would provide an answer to all of the questions. We would most likely remove your answers if you skip too many questions; therefore, we would appreciate if you could choose the best answer that suits you in each question. Please be assured that this research is completely confidential and no attempts will be done to identify you. Although you will be asked to enter personal information (such as your university username, age, gender), your answers will only be viewed by the researcher (Cagin Kazimoglu) and will never be disclosed to others (including University staff, your tutors and course instructors/coordinators). Your university username is only asked because it is important that there is the facility to link your responses from the first questionnaire to your responses in the second questionnaire. We will never attempt to identify you from your username and your data will be kept for research purposes only. The researcher will be available during the entire study and please do not hesitate to ask him questions should you want to.

Both questionnaires should take about 5-10 minutes to complete and the entire study will be no more than an hour. If at any time you wish to withdraw from the study, please inform the researcher/personal tutor and you will be free to leave, no reasons need to be provided. Please be assured that your choice to participate or not in this study will have no effect on your studies.

What are you going to do with my answers?

The answers you provide will be analysed and the data will be held securely by University of Greenwich in accordance with data protection laws until the research project is completed. We will keep the data for research purposes only and we intend to generate statistical information from this data.

Some of the data we have gathered may be published as part of the project report; however individuals will not be identified and will be kept confidential. All data gathered will be destroyed on completion of the research.

For further information about this study please contact:

Cagin Kazimoglu
PhD candidate
Office : QM 165
Smart Systems Technologies Department,
School of Computing and Mathematical Sciences
University of Greenwich
Old Royal Naval College, London SE10 9LS
Tel: 020 8331 8550
Email: c.kazimoglu@greenwich.ac.uk

Eur Ing Dr Mary Kiernan
Office : QM 330
School of Computing and Mathematical Sciences
University of Greenwich,
Old Royal Naval College, London SE10 9LS
Tel: 020 8331 7974
Email: m.kiernan@greenwich.ac.uk

So, what do I need to do?

STEP 1: Go to pre-study link: https://www.surveymonkey.com/s/pre-game-study, select the choice in each question that fits best to you. (5-10 minutes)

STEP 2: Go to game link: https://www.programyourrobot.com, you can play the game up to about 25-30 minutes.

STEP 3: Go to post-study link: https://www.surveymonkey.com/s/post-game-survey, select the choice in each question that fits you the best. (5-10 minutes)

That's it, you are done!

This research is completely confidential and the IP address of your computer or your username will not be tracked during the study.

I have read the consent form and agree to participate in the study and understand that I am free to withdraw at any time.

Your university username: .....
(e.g. KC44, MK42)

Signature: .....

The PGS study participant consent form

(a game based learning research study conducted by Cagin Kazimoglu, a PhD candidate at the University of Greenwich, as part of his PhD)

What is this study? What do you want me to do?

The majority of people find computer programming difficult and whilst this is not something everyone will do, computer programming can be beneficial for many people to learn as the use of technology increasingly affects our everyday lives. In recent years, the interest in computer programming has decreased and people are losing their motivation to learn it. For this reason, new ways of teaching computer programming are being researched. One of these new ways is through the use of video games and video game-based technologies (henceforth referred to as games). Following this trend, we have developed a game which we hope will increase the motivation of pupils to learn programming as well as support their learning process in how computer programming concepts work. We would therefore like to ask you to play this game and provide us with your opinion on whether it works for you or not.

In this study, you will be asked to fill out two on-line questionnaires, the first before you play the game, and the second after you have played it. The purpose of the first questionnaire is to gather information about your prior experience in developing computer algorithms and computer programming, if any. After filling in the first questionnaire, we will ask you to play a game which is on-line and available at: http://www.programyourrobot.com. You will have the chance to play the game for up to about 25 - 30 minutes if you wish to. Having played the game, you will be asked to fill out the second online questionnaire which basically involves questions about your experience of playing the game. Based on your responses, we plan to undertake an evaluation that will hopefully enable us to help people to learn programming in future.

While you are under no obligation to answer any of the questions we would appreciate it if you would provide an answer to all of the questions. In this consent form and in the questionnaire you will notice that you are asked for a unique number. At the beginning of this study, your ICT Teacher will give you a unique number and you will be asked to use this number in both questionnaires so that we can link your responses from the first questionnaire to your responses in the second questionnaire. Your ICT teacher will be available during the entire study and please do not hesitate to ask him questions if you have any.

Please be assured that your anonymity is guaranteed as your unique number has no link with your name so we cannot possibly identify you. Your identity will be kept confidential and only the researcher (Cagin Kazimoglu) will access your answers. More importantly, your answers will never be shared with others (including your school's staff, your parent or legal guardian) and no attempt will be made to identify you. If at any time you wish to withdraw from the study, please inform your ICT teacher and you will be free to leave, no reasons need to be provided.

What are you going to do with my answers?

The answers you provide will be analysed and the data will be held securely by University of Greenwich in accordance with Data Protection laws until the research project is completed. We will keep the data for research purposes only and we intend to generate statistical information from this data. Some of the data we have gathered may be published as part of the project report; however individuals will not be identified and information will be kept confidential. All data gathered will be held securely in the University of Greenwich server in accordance according to university guidelines including the Data Protection Act until the research project is completed which is expected to be June 2013.

I have read the consent form and agree to participate in the study and understand that I am free to withdraw at any time.

Your unique number : .....

Signature: .....

The PGS study parental consent form

To be completed and returned by the parent or legal guardians of all pupils taking part in this educational research study.

Parental Information Sheet

This research study is investigating the use of a dedicated computer game to teach computational thinking, through the medium of helping the game players to develop skills in basic computer programming. Computational thinking is essentially focussed on problem solving, and helping pupils to develop their skills in this area will help them in their use of computers at all levels. This research is being conducted by Cagin Kazimoglu a PhD candidate at the University of Greenwich.

The research study involves three main stages and will be completed in approximately one hour in total. Three stages are involved: a pre-game questionnaire, actual playing of the game and a post-game questionnaire respectively. In the pre-game questionnaire will ask some questions about previous experience and knowledge in the area of computer programming and the general use of computers, and some questions about their view of computer games. The questionnaire should take between 5 -10 minutes to complete. Having completed this, your daughter will be asked to play our game for around 30 minutes. After playing the game, we will ask your daughter to answer a series of questions in a post-game questionnaire, which will be related to her experiences and viewpoints about the game. The post-game questionnaire should take between 5-10 minutes to complete.

Please be assured that both questionnaires are confidential and that the answers from the questionnaires will be anonymised, so there will be no information about your daughter and her responses provided outside of the class situation. The research (Cagin Kazimoglu) will have access to the answers in the anonymised form, and information the pupil participants, but not by name, nor will he have any other information to link back to your daughter. To ensure that this information is properly controlled, all elements of the study, the questionnaires, the game, and the data collected, have been approved by the Research Ethics Committee of the University of Greenwich. Data will be held securely by University of Greenwich in accordance with university guidelines including the Data Protection Act until the research project is completed which is expected to be July 2013 after which it will be destroyed.

I agree to my daughter, **Name, Surname ……** taking part in all the activities described above.

SIGNATURE OF PARENT or LEGAL GUARDIAN

Date: ……

Signature: ……

This form will be collected by your daughter's ICT Teacher and (s)he will be available at all times during the study.

For further information about this study please contact your daughter's ICT teacher.

The Cyprus study – pre study questionnaire

Hi!

Welcome to our pre-game questionnaire. This questionnaire intends to get view your points on computer programming, and your attitude to a potential game to support learning of computer programming constructs and skills.

The questionnaire consists of four different parts. These are

1. Personal Information
2. Institutional Information
3. Background in computer programming
4. Attitude to games and learning

The questionnaire will take 5 - 10 minutes to complete.

Please be assured that this questionnaire is completely confidential and no attempts will be done to identify you. You will be given a unique number and asked to enter this when filling in the questionnaire. We will not ask for your name or any information that would allow us to identify you as an individual. The number you have been given is only asked for because we will want you to complete one further questionnaire after you have played our game and we need to compare your responses from that to the results from this questionnaire. **We will never attempt to identify you from your number and your data will be kept for research purposes only.**

Thank you for participating in our study. We really appreciate your contribution to our research!

1. Personal Information – Step 1/4

Personal Information is the first part of the study. Please provide us some details about yourself that will enable us to evaluate your results statistically.

1.1. Please enter the unique number you have been given to use for this questionnaire:

We will never attempt to identify you from your number and your data will be kept confidential to this research.

1.2. Gender

☐ Male

☐ Female

1.3. Age Range:

☐ 18 – 24

☐ 25 – 29

☐ 30 – 39

☐ Above 40

2. Institutional Information – Step 2/4

This part of the questionnaire is designed to collect data on whether you are considering giving up your degree programme and identifying if programming is a key reason why this may happen.

2.1. Have you considered giving up your degree programme since you started?

☐ Yes

☐ No

☐ I do not know / not applicable to me

2.2. If you have ever thought about giving up your degree programme, was the difficulty of programming a key reason?

☐ Yes, the difficulty of programming is/was a key reason

☐ No, the difficulty of programming is/was never a key reason

☐ I have never thought of giving up my degree / not applicable to me

3. Background in programming – Step 3/4

This part is designed to collect data about your background as well as your current/previous experiences about computer programming (if any).

3.1 If you have ever done computer programming before, at what level do you consider your programming skills/knowledge?

☐ I have very good knowledge/skills in computer programming

☐ I have good knowledge/skills in computer programming

☐ I am neither good nor bad

☐ I have poor knowledge/skills in computer programming

☐ I have very poor knowledge/skills in computer programming

☐ I have never done computer programming before

3.2. How difficult do you find learning computer programming?

☐ Very easy

☐ Easy

☐ Neither easy nor difficult

☐ Difficult

☐ Very difficult

☐ Not Applicable / I don't know

3.3. I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I do not know/ not applicable to me

3.4. I enjoy learning computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I do not know/ not applicable to me

3.5. I think I know how "programming sequence" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a programming sequence is/ not applicable to me

3.6. I think I know how "functions" (also referred as methods) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a function is/ not applicable to me

3.7. I think I know how "decisions" (also referred as selection or decision making such as "if else") works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a decision is/ not applicable to me

3.8. I think I know how "loops" (also referred to as iteration such as "while" loop) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a loop is/ not applicable to me

3.9. I think I have problem solving abilities required for learning computer programming? (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution)

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad in problem solving

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

3.10. Based on my computer programming course(s), I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4. Games and Learning – Step 4/4

This part of the questionnaire is designed to collect data on your current/past experience with games and their potential use for learning purposes particularly for computer programming.

4.1. I often play video games.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ not applicable to me / I cannot play video games

4.2. If you have ever used a video game for educational purposes rather than entertainment, do you believe it was helpful to you?

☐ I played an educational game before and it was helpful to me

☐ I played an educational game before but it wasn't helpful to me

☐ I never played a game specifically designed for educational purposes / not applicable to me

4.3. I think a video game specifically designed for computer programming purposes can be useful for learning how computer programming constructs work..

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4.4. Please add your opinions about games and learning how computer programming constructs work.

Do you think a game based approach can teach computer programming? If so, what do you think it can teach you?

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

- THANK YOU FOR YOUR PARTICIPATION –

The Greenwich study – pre study questionnaire

Hi!

Welcome to our pre-game questionnaire. This questionnaire intends to get view your points on computer programming, and your attitude to a potential game to support learning of computer programming constructs and skills.

The questionnaire consists of four different parts. These are

1. Personal Information
2. Institutional Information
3. Background in computer programming
4. Attitude to games and learning

The questionnaire will take 5 - 10 minutes to complete.

Please be assured that this questionnaire is completely confidential and no attempts will be done to identify you. You will be asked to enter your university username when filling in the questionnaire; however this will not be used to identify you as an individual. Your university username is only asked for because we will want you to complete one further questionnaire after you have played our game and we need to compare your responses from that to the results from this questionnaire. **We will never attempt to identify you from your university username and your data will be kept for research purposes only.**

1. Personal Information – Step 1/4

Personal Information is the first part of the study. Please provide us some details about yourself that will enable us to evaluate your results statistically.

1.1. Please enter your university username (e.g. KC44, KM42):

We will never attempt to identify you from your username and your data will be kept confidential to this research.

1.2. Gender

☐ Male

☐ Female

1.3. Age Range:

☐ 18 – 24

☐ 25 – 29

☐ 30 – 39

☐ Above 40

1.4. Ethnicity:

The major ethnic classifications used in this questionnaire are a UK government standard.

☐ Asian or Asian British

☐ Black or Black British

☐ Chinese

☐ Mixed / Dual Background

☐ White

☐ Any other ethnic group:

1.5. Your degree Programme :

☐ BEng Software Engineering

☐ BEng Computer Systems & Networking

☐ BEng Embedded Computer Systems

☐ BSc Business Computing

☐ BSc Computer Science

☐ BSc Computing with Games Development

☐ BSc Computing with Multimedia

☐ BSc Internet Computing

☐ BSc Software Engineering

☐ BSc Computer Security & Forensics

☐ BSc Computer Systems & Networking

☐ BSc Computing with Embedded Systems

☐ BSc Mobile Computing & Communications

☐ BSc Business Information Systems

☐ BSc Business Information Technology

☐ BSc IT with Digital Media / Networking / Security

☐ BSc Web Business Systems

☐ BSc Digital Animation & Production

☐ BSc Digital Media Technologies

☐ BSc Games & Multimedia Technologies

☐ BSc Multimedia Technology

☐ BSc Web Technologies

☐ Any other (please specify):

1.6 What is the highest mathematical qualification/certificate you have achieved?

☐ A-Level Maths or equivalent

☐ AS Level Maths or equivalent

☐ GCSE Maths grade A or B or equivalent

☐ GCSE Maths grade C or equivalent

☐ Lower than GCSE Maths grade C

☐ Any other (please specify):

2. Institutional Information – Step 2/4

This part of the questionnaire is designed to collect data on whether you are considering giving up your degree programme and identifying if programming is a key reason why this may happen.

2.1. Have you considered giving up your degree programme since you started?

☐ Yes

☐ No

☐ I do not know / not applicable to me

2.2. If you have ever thought about giving up your degree programme, was the difficulty of programming a key reason?

☐ Yes, the difficulty of programming is/was a key reason

☐ No, the difficulty of programming is/was never a key reason

☐ I have never thought of giving up my degree / not applicable to me

3. Background in programming – Step 3/4

This part is designed to collect data about your background as well as your current/previous experiences about computer programming (if any).

3.1 If you have ever done computer programming before, at what level do you consider your programming skills/knowledge?

☐ I have very good knowledge/skills in computer programming

☐ I have good knowledge/skills in computer programming

☐ I am neither good nor bad

☐ I have poor knowledge/skills in computer programming

☐ I have very poor knowledge/skills in computer programming

☐ I have never done computer programming before

3.2. How difficult do you find learning computer programming?

☐ Very easy

☐ Easy

☐ Neither easy nor difficult

☐ Difficult

☐ Very difficult

☐ Not Applicable / I don't know

3.3. I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I do not know/ not applicable to me

3.4. I enjoy learning computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I do not know/ not applicable to me

3.5. I think I know how "programming sequence" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a programming sequence is/ not applicable to me

3.6. I think I know how "functions" (also referred as methods) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a function is/ not applicable to me

3.7. I think I know how "decisions" (also referred as selection or decision making such as "if else") works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a decision is/ not applicable to me

3.8. I think I know how "loops" (also referred to as iteration such as "while" loop) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a loop is/ not applicable to me

3.9. I think I have problem solving abilities required for learning computer programming? (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution)

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad in problem solving

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

3.10. Based on my computer programming course(s), I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4. Games and Learning – Step 4/4

This part of the questionnaire is designed to collect data on your current/past experience with games and their potential use for learning purposes particularly for computer programming.

4.1. I often play video games.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ not applicable to me / I cannot play video games

4.2. If you have ever used a video game for educational purposes rather than entertainment, do you believe it was helpful to you?

☐ I played an educational game before and it was helpful to me

☐ I played an educational game before but it wasn't helpful to me

☐ I never played a game specifically designed for educational purposes / not applicable to me

4.3. I think a video game specifically designed for computer programming purposes can be useful for learning how computer programming constructs work..

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4.4. Please add your opinions about games and learning how computer programming constructs work.

Do you think a game based approach can teach computer programming? If so, what do you think it can teach you?

.....
.....
.....
.....
.....
.....

.....
.....
.....
.....
.....

- THANK YOU FOR YOUR PARTICIPATION -

The PGS study – pre study questionnaire

Hi!

Welcome to our pre-game questionnaire. This questionnaire intends to get view your points on computer programming and your attitude to a potential game to support learning of computer programming constructs and skills.

The questionnaire consists of four different parts. These are

1. Personal Information
2. Background in computer programming
3. Attitude to games and learning

The questionnaire will take 5 - 10 minutes to complete.

Please be assured that this questionnaire is completely confidential and no attempts will be done to identify you. You will be given a unique number and asked to enter this when filling in the questionnaire. We will not ask for your name or any information that would allow us to identify you as an individual. The number you have been given is only asked for because we will want you to complete one further questionnaire after you have played our game and we need to compare your responses from that to the results from this questionnaire. **We will never attempt to identify you from your number and your data will be kept for research purposes only.**

Thank you for participating in our study. We really appreciate your contribution to this research!

1. Personal Information – Step 1/3

Personal Information is the first part of the study. Please provide us some details about yourself that will enable us to evaluate your results statistically.

1.1. Please enter the unique number you have been given to use for this questionnaire:

We will never attempt to identify you from your number and your data will be kept confidential to this research.

1.2. Ethnicity:

The major ethnic classifications used in this questionnaire are a UK government standard.

- ☐ Asian or Asian British
- ☐ Black or Black British
- ☐ Chinese
- ☐ Mixed / Dual Background
- ☐ White
- ☐ Any other ethnic group:

2. Background in programming – Step 2/3

This part is designed to collect data about your background as well as your current/previous experiences about computer programming (if any).

3.1 If you have ever done computer programming before, at what level do you consider your programming skills/knowledge?

- ☐ I have very good knowledge/skills in computer programming
- ☐ I have good knowledge/skills in computer programming
- ☐ I am neither good nor bad
- ☐ I have poor knowledge/skills in computer programming
- ☐ I have very poor knowledge/skills in computer programming
- ☐ I have never done computer programming before

3.3. I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

- ☐ Strongly agree
- ☐ Agree
- ☐ Neither agree nor disagree
- ☐ Disagree
- ☐ Strongly disagree
- ☐ I do not know/ not applicable to me

3.4. I think I know how "programming sequence" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a programming sequence is/ not applicable to me

3.5. I think I know how "functions" (also referred as methods) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a function is/ not applicable to me

3.6. I think I know how "decisions" (also referred as selection or decision making such as "if else") works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a decision is/ not applicable to me

3.7. I think I know how "loops" (also referred to as iteration such as "while loop) work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know what a loop is/ not applicable to me

3.8. I think I have problem solving abilities required for learning computer programming? (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution)

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad in problem solving

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

3.9. Based on my computer programming course(s), I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I am neither good nor bad

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4. Games and Learning – Step 3/3

This part of the questionnaire is designed to collect data on your current/past experience with games and their potential use for learning purposes particularly for computer programming.

4.1. I often play video games.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ not applicable to me / I cannot play video games

4.2. If you have ever used a video game for educational purposes rather than entertainment, do you believe it was helpful to you?

☐ I played an educational game before and it was helpful to me

☐ I played an educational game before but it wasn't helpful to me

☐ I never played a game specifically designed for educational purposes / not applicable to me

4.3. I think a video game specifically designed for computer programming purposes can be useful for learning how computer programming constructs work..

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree / I have no opinion either way

☐ Disagree

☐ Strongly disagree

☐ I do not know / I am not sure

4.4. Please add your opinions about games and learning how computer programming constructs work.

Do you think a game based approach can teach computer programming? If so, what do you think it can teach you?

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

- THANK YOU FOR YOUR PARTICIPATION -

The Cyprus study – post study questionnaire

Hi!

Welcome to the second part of our research - the post-game questionnaire. Now that you played the game we need your feedback in order to decide whether or not this game works for you.

Similar to the pre-game questionnaire, this questionnaire consist of several parts which are:

1. Participation number
2. Game experience
3. Computer programming
4. Computational thinking
4. Attitude to learning computer programming through game-play

We need your unique number in order to match your answers from this questionnaire with the answers you have given us in the previous questionnaire.

Once again, thank you for participating. We really appreciate your contribution!

1. Participant number – Step 1/5

1.1. Please enter the unique number you have been given to use for this questionnaire:

We will never attempt to identify you from your unique number and your data will be kept confidential to this research.

2. Game Experience – Step 2/5

Please rate each of the followings according to your game-play experience.

2.1. How far have you been able to go through in the game?

☐ Only played level 1 - introducing sequence
☐ Played level 2 and/or 3 – introducing functions (methods)
☐ Played level 4 – introducing decision making (selection)
☐ Played level 5 – introducing loops (iteration)
☐ Played level 6 – all
☐ Completed the game

2.2. I believe this game is easy to learn to play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.3. I think this game presents a good example of how computer programs are put together.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.4. I think this game introduced to me at an appropriate time (that is while I am learning introductory computer programming).

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ Not applicable to me

☐ I don't know / I did not play enough to decide this

2.5. I think this game improved/ has the potential to improve my understanding of how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3. Computer programming – Step 3/5

Please rate each of the followings according to your game-play experience.

3.1. Having played the game, at what level do you consider your programming/skills now?

☐ I have very good knowledge/skills in computer programming

☐ I have good knowledge/skills in computer programming

☐ I am neither good nor bad

☐ I have poor knowledge/skills in computer programming

☐ I have very poor knowledge/skills in computer programming

☐ I don't know / I did not play enough to decide this

3.2. Having played the game, how difficult do you find learning computer programming now?

☐ Very easy

☐ Easy

☐ Neither easy nor difficult

☐ Difficult

☐ Very difficult

☐ I don't know / I did not play the game enough to decide this

3.3. Having played the game, I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.4. I enjoyed this form of learning computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.5. Having played the game, I think I know how "programming sequence" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.6. Having played the game, I think I know how "functions" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.7. Having played the game, I think I know how "decision making" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.8. Having played the game, I think I know how "loops" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.9. Having played the game, I think I have the problem solving abilities required for learning computer programming (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution).

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.10. Having played the game, I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4. Computational thinking – Step 4/5

Please rate the followings according to your game experience and usage in the game.

4.1. I think playing this game requires thinking logically and evaluating conditions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.2. I think this game developed/ has the potential to develop my ability to think algorithmically.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.3. I think the run-time mode (run button) in this game simulates how computer algorithms work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.4. I think the debug mode (debug button) in this game was useful to detect errors in my solutions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.5. Sharing ideas / strategies with a friend was helpful for designing my solutions during the game-play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I did not share ideas / not applicable to me

5. Attitude to learning computer programming through game-play – Step 4/5

5.1. I think this game is useful for learning how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play the game enough to decide this

Please provide your feedback about the following questions. Please notice that this game is only a prototype and your positive/negative comments will guide the future versions of it.

Do you think this game was helpful to you? If so how?

Do you think using this game to support tutorials is a good idea?

Would you like to see any improvements in the game? If so what type of improvements?

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

THANK YOU FOR YOUR PARTICIPATION

The Greenwich study – post study questionnaire

Hi!

Welcome to the second part of our research - the post-game questionnaire. Now that you played the game we need your feedback in order to decide whether or not this game works for you.

Similar to the pre-game questionnaire, this questionnaire consist of several parts which are:

1. Username
2. Game experience
3. Computer programming
4. Computational thinking
4. Attitude to learning computer programming through game-play

We need your university username in order to match your answers from this questionnaire with the answers you have given us in the previous questionnaire.

Once again, thank you for participating. We really appreciate your contribution!

1. Username – Step 1/5

1.1. Please enter your university username:

We will never attempt to identify you from your username and your data will be kept confidential to this research.

_____ **(**e.g. KC44, MK42)

2. Game Experience – Step 2/5

Please rate each of the followings according to your game-play experience.

2.1. How far have you been able to go through in the game?

☐ Only played level 1 - introducing sequence

☐ Played level 2 and/or 3 – introducing functions (methods)

☐ Played level 4 – introducing decision making (selection)

☐ Played level 5 – introducing loops (iteration)

☐ Played level 6 – all

☐ Completed the game

2.2. I believe this game is easy to learn to play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.3. I think this game presents a good example of how computer programs are put together.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.4. I think this game introduced to me at an appropriate time (that is while I am learning introductory computer programming).

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ Not applicable to me

☐ I don't know / I did not play enough to decide this

2.5. I think this game improved/ has the potential to improve my understanding of how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3. Computer programming – Step 3/5

Please rate each of the followings according to your game-play experience.

3.1. Having played the game, at what level do you consider your programming/skills now?

☐ I have very good knowledge/skills in computer programming

☐ I have good knowledge/skills in computer programming

☐ I am neither good nor bad

☐ I have poor knowledge/skills in computer programming

☐ I have very poor knowledge/skills in computer programming

☐ I don't know / I did not play enough to decide this

3.2. Having played the game, how difficult do you find learning computer programming now?

☐ Very easy

☐ Easy

☐ Neither easy nor difficult

☐ Difficult

☐ Very difficult

☐ I don't know / I did not play the game enough to decide this

3.3. Having played the game, I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.4. I enjoyed this form of learning computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.5. Having played the game, I think I know how "programming sequence" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.6. Having played the game, I think I know how "functions" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.7. Having played the game, I think I know how "decision making" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.8. Having played the game, I think I know how "loops" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.9. Having played the game, I think I have the problem solving abilities required for learning computer programming (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution).

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.10. Having played the game, I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4. Computational thinking – Step 4/5

Please rate each of the following programming constructs according to your game experience and usage in the game.

4.1. I think playing this game requires thinking logically and evaluating conditions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.2. I think this game developed/ has the potential to develop my ability to think algorithmically.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.3. I think the run-time mode (run button) in this game simulates how computer algorithms work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.4. I think the debug mode (debug button) in this game was useful to detect errors in my solutions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.5. Sharing ideas / strategies with a friend was helpful for designing my solutions during the game-play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I did not share ideas / not applicable to me

5. Attitude to learning computer programming through game-play – Step 4/5

5.1. I think this game is useful for learning how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play the game enough to decide this

Please provide your feedback about the following questions. Please notice that this game is only a prototype and your positive/negative comments will guide the future versions of it.

Do you think this game was helpful to you? If so how?

Do you think using this game to support tutorials is a good idea?

Would you like to see any improvements in the game? If so what type of improvements?

.....

.....

.....
.....
.....
.....
.....
.....
.....
.....
.....

THANK YOU FOR YOUR PARTICIPATION

The PGS study – post study questionnaire

Hi!

Welcome to the second part of our research - the post-game questionnaire. Now that you played the game we need your feedback in order to decide whether or not this game works for you.

Similar to the pre-game questionnaire, this questionnaire consist of several parts which are:

1. Participation number
2. Game experience
3. Computer programming
4. Computational thinking
4. Attitude to learning computer programming through game-play

We need your unique number in order to match your answers from this questionnaire with the answers you have given us in the previous questionnaire.

Once again, thank you for participating. We really appreciate your contribution!

1. Participant number – Step 1/5

1.1. Please enter the unique number you have been given to use for this questionnaire:

We will never attempt to identify you from your unique number and your data will be kept confidential to this research.

2. Game Experience – Step 2/5

Please rate each of the followings according to your game-play experience.

2.1. How far have you been able to go through in the game?

☐ Only played level 1 - introducing sequence
☐ Played level 2 and/or 3 – introducing functions (methods)
☐ Played level 4 – introducing decision making (selection)
☐ Played level 5 – introducing loops (iteration)
☐ Played level 6 – all
☐ Completed the game

2.2. I believe this game is easy to learn to play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.3. I think this game presents a good example of how computer programs are put together.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

2.4. I think this game improved/ has the potential to improve my understanding of how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3. Computer programming – Step 3/5

Please rate each of the followings according to your game-play experience.

3.1. Having played the game, at what level do you consider your programming/skills now?

- ☐ I have very good knowledge/skills in computer programming
- ☐ I have good knowledge/skills in computer programming
- ☐ I am neither good nor bad
- ☐ I have poor knowledge/skills in computer programming
- ☐ I have very poor knowledge/skills in computer programming
- ☐ I don't know / I did not play enough to decide this

3.2. Having played the game, I think I have intrinsic motivation (motivation that is driven by an interest or enjoyment) to learn computer programming.

- ☐ Strongly agree
- ☐ Agree
- ☐ Neither agree nor disagree
- ☐ Disagree
- ☐ Strongly disagree
- ☐ I don't know / I did not play enough to decide this

3.3. Having played the game, I think I know how "programming sequence" works in computer programming.

- ☐ Strongly agree
- ☐ Agree
- ☐ Neither agree nor disagree
- ☐ Disagree
- ☐ Strongly disagree
- ☐ I don't know I did not see this programming construct in the game

3.4. Having played the game, I think I know how "functions" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.5. Having played the game, I think I know how "decision making" works in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.6. Having played the game, I think I know how "loops" work in computer programming.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know I did not see this programming construct in the game

3.7. Having played the game, I think I have the problem solving abilities required for learning computer programming (e.g. being able to divide problems into smaller units that can be dealt with individually and then combine these to form a solution).

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

3.8. Having played the game, I can easily visualise programming constructs in my head from given problems.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4. Computational thinking – Step 4/5

Please rate each of the following programming constructs according to your game experience and usage in the game.

4.1. I think playing this game requires thinking logically and evaluating conditions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.2. I think this game developed/ has the potential to develop my ability to think algorithmically.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.3. I think the run-time mode (run button) in this game simulates how computer algorithms work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.4. I think the debug mode (debug button) in this game was useful to detect errors in my solutions.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play enough to decide this

4.5. Sharing ideas / strategies with a friend was helpful for designing my solutions during the game-play.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I did not share ideas / not applicable to me

5. Attitude to learning computer programming through game-play – Step 4/5

5.1. I think this game is useful for learning how computer programming constructs work.

☐ Strongly agree

☐ Agree

☐ Neither agree nor disagree

☐ Disagree

☐ Strongly disagree

☐ I don't know / I did not play the game enough to decide this

Please provide your feedback about the following questions. Please notice that this game is only a prototype and your positive/negative comments will guide the future versions of it.

Do you think this game was helpful to you? If so how?

Do you think using this game to support tutorials is a good idea?

Would you like to see any improvements in the game? If so what type of improvements?

.....
.....
.....
.....
.....
.....
.....
.....
.....

THANK YOU FOR YOUR PARTICIPATION