A VIDEO GAME FOR TEACHING INTRODUCTORY PROGRAMMING

THE EDUCATIONAL EFFECTIVENESS OF A COOPERATIVE AND COMPETITIVE VIDEO GAME FOR TEACHING INTRODUCTORY PROGRAMMING

By
SAMANTHA CHAN

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the degree
Master of Applied Science in Software Engineering

McMaster University

MASTER OF APPLIED SCIENCE (2014) McMaster University
(Software Engineering) Hamilton, Ontario

TITLE: The Educational Effectiveness of a Cooperative and Competitive Video Game for Teaching Introductory Programming
AUTHOR: Samantha Chan
SUPERVISOR: Dr. Spencer Smith and Dr. Christopher Anand
RESEARCH ETHICS NUMBER: 2013 176 (McMaster University Research Ethics Board)
NUMBER OF PAGES: ix, 166

For my mother, who taught me to always learn from my experiences

Abstract

The subject of computer programming is highly practical and it is crucial that beginners participate in hands-on experimentation as part of the learning process. Unfortunately, many first year engineering students that are new to this discipline are often intimidated by the material and unmotivated to review or practice the concepts on their own. The purpose of this study is to measure the success of using a cooperative and competitive video game as a pedagogical tool in software engineering education. The video game that was developed for this research is called Space Race and it harnesses the power of group discussion to encourage students to share their individual understandings of basic programming concepts. This dissemination of knowledge within groups was able to teach many students new concepts that they did not understand previously.At least 67% of the students stated that the game motivates them to review course material. The game was well-received with at least 82% of the students that played Space Race agreeing that they would recommend that others also learn basic programming concepts with this game. Although the game does not directly teach students new concepts, it allows the instructors to identify what concepts students struggle with. Space Race encourages students to ask the instructor questions when they do not understand. In some cases, game participants outperform non-participants on course exams. On the final course exam, all of the statistically significant ($p<0.05$) comparisons (42% of the relevant questions) showed a performance improvement of game participants, with a maximum grade improvement of 41%. The findings also suggest that some students can retain the knowledge obtained from Space Race for at least 7 weeks. The results of this study provide strong evidence that a video game can be a successful pedagogical tool for software engineering education.

Acknowledgments

This work would not have been possible without the support and help I have received from very many wonderful people over the years. In the course of completing this research, I was reminded of how very little I know and how much I rely on the experience and intellect of others. I also could not have finished this research, with my sanity intact, without the incredible encouragement and kindness I have received from so many amazing people I have met along the way.

I would like to thank the entire Engineering I Department and the Computing and Software Department for their continuous encouragement and assistance. Thank you, Michael Curwin, for all of the invaluable technical support and advice you provided. Thank you, Jessica Anderson, Helena Collins, Tina Macala, Ruth Nicholson, Joanne Squires, Kristina Trollip, and Teresa Trimboli for being so helpful, and encouraging. Thank you, Joanne Bannister, for being such an incredible friend. Deep thanks to Dr. Tom Doyle, Dr. William Farmer, Dr. Robert Fleisig, and Dr. Colin McDonald for being so accommodating and supportive. To the Engineering I students that I have taught or that have participated in my research: thank you for making this work possible and for brightening my day with your kind words of support.

I would also like to thank all of the great peers that I had the fortunate opportunity to work with. You have all made my time here so unforgettable and enjoyable. To Sari Latif and Leo Li, thank you for listening to all my crazy ideas; providing invaluable help and direction; and just making me laugh. Thank you, Monika Bialy and Widmer Bland for being my friends and helping me with my graduate courses. For being such great listeners, game-testers, advice-givers, and friends: thank you so much, Shawn Simon, Alessandro Minali, Mohsin Khan, Jamie Counsell, Taralyn Schwering, JJ Booth, Omar Boursali, Ian Sinke, Karl Good, Alexander Halliwushka, and Taylor Sorgini.

I also want to send a big thank you to Dr. Christopher Anand, Kevin Browne, and Andrew Curtis for giving me such great advice on ethics approval, game and research design, and statistical analysis. I would also like to give huge thanks to Michael Viveros for working so unbelievably hard to help me develop Space Race. Michael's remarkable work was what allowed Space Race to be such a success. Thank you again for making all the necessary adjustments for Space Race and for smiling through it all.

My sincerest appreciation and thanks to Dr. Spencer Smith for believing in me, providing me with outstanding guidance, and giving me the freedom to spread my wings. I truly could not have done this without you. Thank you for constantly reassuring me and providing me with such incredible opportunities. I have learned so much from you and I will be sure to take this knowledge with me everywhere I go.

Finally, as always, thanks to my family for believing in me from the start. Thank you, Paulo Fetalvero, for being impossibly patient with me. Thank you, Dad, for reminding me of the importance of school. To my Brother, thank you for supporting me in your own way;

whether that means driving me to school or never questioning the work I do, thank you. For the wonderful meals and support, thank you, Grandma. To my Grandpa, thank you for inspiring me to always improve myself and to constantly seek out new knowledge. Finally, I want to say thank you to my superhuman Mom for being my best friend and biggest fan.

Contents

Chapter 1

Introduction

Computing is an important discipline for future engineers and scientists to understand. However, it is challenging to motivate and teach computing to many of today's students in higher education. This thesis considers the use of video games to teach this material more effectively than current, more traditional, methods. A new cooperative and competitive video game is developed and then analyzed for this study. Student attitudes towards gaming in education is surveyed. The game is also assessed in its short term impact and longer term impact on knowledge retention.

This chapter begins with a statement of the problem that the researcher would like to address. It is then followed by a discussion of the solution that has been proposed. The contributions of this study will also be highlighted and a general mapping of the thesis will be provided.

1.1 Problem Statement

Computer programming is a complex subject that cannot be taught through conventional lecturing and explanation alone. The practical nature of programming requires that those new to this discipline must engage in hands-on experimentation to become skilled in this field. Students in first year engineering at McMaster University are required to take Engineering Computation (ENG 1D04), an introductory course to the fundamentals of computer programming. The current curriculum prescribes 3 hours of mandatory hands-on lab activities over the course of 12 weeks to provide students with an opportunity to explore programming first hand. Unfortunately, this is an insufficient amount of time for beginners to become experienced with programming. When this is coupled with a general lack of motivation from students to study the material on their own, many students find themselves struggling with the course content.

Observation has shown that a majority of students enrolled in ENG 1D04 are unmo-

tivated to learn computer programming during their regular study. This is partly because students are intimidated by new syntax and lengthy code examples shown in textbooks and course slides. Students have also been observed to learn programming concepts more successfully through discussion in a group. They are much more capable at solving programming lab assignments in a group as opposed to individually. Anecdotes from students also suggest that they prefer this method of learning to working individually. This observation matches the findings of other researchers, as discussed in Chapter 3.

Efforts have been made through the new Experiential Playground and Innovation Classroom (EPIC) lab at McMaster University to encourage students to learn through experience [21]. Bonus marks are rewarded for students that volunteer to complete any or all four lab activities being offered. Almost all the students enrolled in ENG 1D04 of Term 1 of the 2012-2013 Academic Year completed the first lab activity and 65% of the students completed at least 1 of the remaining 3 lab activities. This leaves room for improvement. More efforts need to be directed in motivating students on learning through experimentation.

1.2 Proposed Solution

To improve student engagement with programming course material, the project proposes a solution that introduces a video game into teaching practices. Specifically, a cooperative and competitive game, Space Race, was designed to teach students basic programming concepts. This was an additional activity that was integrated into the EPIC lab. The incentive for this project stems from the tendency of games to engage users and their ability to cater to the needs of this generation's learners. Moreover, games provide an ideal environment for students to actively learn programming through self-exploration and experimentation. This notion is explored in depth in Chapter 2. Finally, as mentioned above, students were observed to learn more effectively in groups, rather than individually. A game could take advantage of this fact and encourage these group discussions by incorporating social interaction between players.

1.3 Thesis Contributions

The findings of this research can provide software engineering educators and pedagogical researchers with insight on the feasibility and effectiveness of a video game as a teaching tool. As seen in Chapter 3, teaching computer programming through a game in higher education remains a relatively new field and more research must be conducted to evaluate the effectiveness of such an approach. Specifically, this study will contribute the following:

- Provide evidence that a majority of students in engineering would like to see video

games incorporated into education and feel that their inclusion could motivate students to engage with course material

- Demonstrate how cooperation and group discussion can be harnessed effectively through a game to teach students basic computer programming concepts
- Highlight the importance of feedback in educational video games
- Prove that a video game can potentially have a positive effect on student knowledge in programming immediately after gameplay
- Show that an educational video game that teaches programming can potentially positively impact student understanding of course material
- Reveal that programming knowledge obtained in a video game can be used in a non-gaming context
- Indicate that the understandings acquired through a video game that teaches programming can be retained for at least 7 weeks by some students that play the game

1.4 Thesis Organization

The thesis is organized according to the chronological order of steps that were followed in designing this research experiment.

To begin, Chapter 2 will orient the reader with the theoretical background for this project including learning theories such as experiential learning and collaborative learning. Games will also be explored and defined. Specifically, flow theory, competition, and cooperation in games will be mentioned. The rationale for game-based learning will also be presented.

Chapter 3 will investigate the motivations for injecting game-based learning into software engineering education. The chapter will also explore the role that cooperation can play in learning programming. Finally, previous research on video games as a pedagogical tool for computing and software education will be presented.

Chapter 4 will cover the gameplay, level design, and game mechanics of Space Race. Space Race was the game that was used for this study. An understanding of how the game is played is imperative to understanding the results of this study.

Chapter 5 mentions the experimental procedures that was followed by the researcher. This chapter will describe the participants of the study, the data that was collected for analysis, and the data collection timeline.

Chapter 6 states the results of a survey that was used to gauge student attitudes towards the incorporation of games in education as well as their perceived prior experience with

programming. The results of the survey will also reveal the video gaming habits of the student population within this study.

Chapter 7 discloses the feedback and comments that the researcher received from students regarding the playability, teachability, and the cooperative and competitive aspects of Space Race.

Chapter 8 presents a discussion of the results of various exams and quizzes that were used to assess the educational effectiveness of Space Race. The exam results of game participants will also be contrasted and compared to non-participants.

Finally, Chapter 9 will deliver conclusions that can be derived from the results of this study. The chapter will also discuss the limitations of this study as well as the future work that can be done to both improve Space Race and the education that students can receive in software engineering.

Chapter 2

Theoretical Background

This chapter introduces and examines learning theories that support the incorporation of video games in education. Specifically, the progression from behavioural to cognitive to constructivist perspectives will be presented with an emphasis on the constructivist approach, since it most closely aligns with the goals of experiential and collaborative learning.This chapter will also formally define games and review some characteristics of games like their tendency to engage players. Finally, research literature presented will also be on recent empirical studies that have investigated the feasibility of game-based learning.

2.1 Learning Theories

Throughout history, many researchers and philosophers have tried to understand and develop theories on how people learn. This led to the development of many philosophies of education that attempted to unify pedagogy, curriculum, learning theory, and the purpose of education. Up until the late 1950s, behaviourism was the leading theory in education [12]. Behavioural theorists believed that learning would take place as a result of responses to stimuli in the environment which was reinforced by teachers; the mind was a "black box" that could be studied and manipulated by observing and controlling external behaviour. For those that followed this philosophy, teaching through conditioning, by supplying the appropriate reinforcer for each desired behaviour, was the primary technique [27].

By the late 1950s, cognitivism was the primary educational philosophy [12]. Proponents of the cognitive theory believed the mind was more than a "black box" and mental processes such as thinking, memory, and problem-solving needed to be understood to explain how learning occurs. As such, the individual learner was more important than the environment for cognitivist theorists. Instructional design was influenced by their understanding of how the human mind works and the focus was on building intelligence and cognitive development [27]. In more recent years, constructivism has taken over as the

prevailing paradigm for instructional designers [12].

Jean Piaget, a French philosopher, is commonly credited for the formalization of the constructivism learning theory [37]. The constructivism perspective builds upon the cognitivist approach. Like cognitivists, constructivists believe that knowledge is constructed by the individual. However, a greater focus is placed on the role of interactions between external experiences and internal ideas in the learning process. When applying this approach to pedagogical design, students are encouraged to learn by increasing their participation in the appropriation of their knowledge. This type of instructional strategy sees self exploration and questioning as necessary components to the learning process. Consequently, when teaching within the constructivist mode, learners become actively involved so that they may build knowledge for themselves; they do not simply mirror and reflect what they have read. In this style of active learning, teachers become facilitators of a framework of free exploration and, as such, learners bear the responsibility of learning [37].

The constructivist theory is composed of many suppositions, but Savery and Duffy have condensed them to three fundamental ideas [61]:

1. *Understanding is in our interactions with the environment*
2. *Cognitive conflict or puzzlement is the stimulus for learning and determines the organization and nature of what is learned*
3. *Knowledge evolves through social negotiation and through the evaluation of the viability of individual understandings*

The first idea states that one cannot consider what is learned independently from how it is learned; this is the essential concept of constructivism. To expand, what an individual learns is a function of the context, content, learner's activity, and the objectives of the learner. Since an individual's understanding is constructed independently, this understanding cannot be shared with others but group discussion can be a means of testing the compatibility of separate understandings. The second proposition identifies cognitive conflict or puzzlement as the primary source of motivation for learning. This intrinsic motivation to understand is responsible in determining what the learner attends to and what is constructed from experiences. Essentially, the goal of the learner is pivotal to considering what will be learned. The final supposition establishes social collaboration as a critical part in developing individual understanding. Discussion with people is a mechanism that an individual can utilize to test their personal understanding. Furthermore, other individuals can provide alternative views that challenge thinking. These collaborative groups are necessary to enrich, interweave, and expand understandings of different phenomena [61].

Research conducted by Tynjala in 1999 [65] examined the benefits of a constructivist learning environment versus a traditional learning environment in universities. Tynjala argues that traditional forms of university instruction focuses on the acquisition of inert

knowledge. Such knowledge is suitable in instructional settings but is often not transferable to a real working environment where problems are much more complex and social factors can influence outcomes. Within this traditional framework, students are encouraged to simply memorize and reproduce the acquired knowledge; much like one would do in a behaviourist learning environment. This passive reception of information is vastly different than the active learning that takes place with the continuous process of construction and reconstruction of perceived phenomena in a constructivist learning environment. Tynjala found that students within a constructivist learning environment acquired more diversified knowledge. Students that learned in the constructivist schema reported that the knowledge they acquired could be practically applied. In addition, they also felt a development in their thinking and communication skills, which helped them articulate their thoughts and function as a more effective team member. In summary, the findings of the study suggests that learning outcomes generated in a constructivist environment better fulfills the requirements of working life. This could be attributed to the constructivist student's perceived enhancement of personal transferable skills [65].

These philosophies of constructivism and constructivist learning environments are important for this study because of the way these ideas are apparent in certain types of video games. In addition, several researchers have discovered that learning with well-designed video games follow constructivist principles [17]. For instance, video games can create an environment through which learners can explore. While navigating immersive virtual worlds with rich media, learners can practice skills that can be transferred to the real world. Furthermore, games can present a context for problem-solving. Finally, collaboration and learning from peers, which is fundamental to the constructivist approach, can be facilitated through multi-player video games or collaborative gameplay in the same physical space [69].

In the following subsections, experiential learning and collaborative learning will be explored. Both of these learning styles have characteristics that are congruent with the principles of constructivism. Each of these styles support the argument that games can be used to create a constructivist learning environment, which in turn will create a space where effective learning can take place.

2.1.1 Experiential Learning

As mentioned previously, constructivism posits that students learn better through self-exploration and deriving personal meaning from that experience. It is no surprise then that experiential learning, which strives to instruct through experience, stems from constructivist principles. In the experiential learning model, experience plays a central role in the learning process. Kolb's experiential learning model (Figure 2.1) consists of four stages. It begins with the student actively taking on the learning process by exposure to tangible experiences. The second stage requires the student to reflect on the experience.

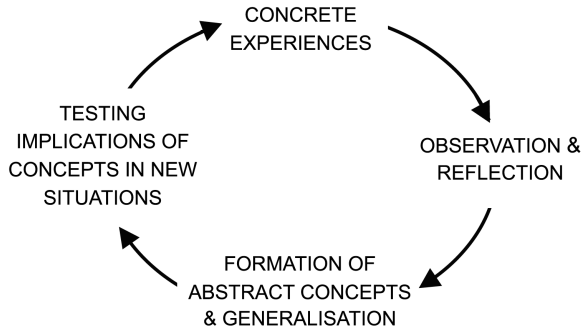


Figure 2.1: Kolb's Experiential Learning Cycle [32]

This reflection is then followed by the generation of abstract concepts, which must then be generalized. Finally, the learner must modify and plan the next learning session. Generally, this model places importance on the continuous nature of learning and the appropriate feedback; this is key to providing a basis for a continuous process of goal-motivated action [32].

For a long time, designers of digital learning environments have looked to experiential learning theory for direction [35]. The ideas of experiential learning provides a solid foundation for the integration of games and pedagogy. The virtual worlds created by games have been found to be highly interactive since they can provide dynamic feedback, experimentation and exploration opportunities. As such, video games provide an excellent platform for interaction and feedback to take place; these are essential components of the experiential learning cycle [29]. When players are immersed in these virtual worlds, they must examine the environment, reflect on the situation, and generate a theory about what something in this circumstance might mean, and re-adjust the virtual world to observe what effect it has. This process closely matches that of the experiential learning cycle.

2.1.2 Collaborative Learning

A central component of constructivist learning is the collaboration of students. When students group together, they can share and clarify ideas and opinions, while enhancing their communication skills and learning from one another. Collaboration enables each individual to work to their strengths, expand their critical thinking skills, exercise their creativity, confirm their understandings, and become familiarized with different learning styles, skills, and perspectives [69]. When considering group-based learning, Strijbos stipulates that it is composed of an intertwining of cognitive, social, and motivational components [63]. As seen in Figure 2.2 motivation can affect cognition through persistence in practice and alter social aspects, such as group cohesion and student responsibility. Cognitive development,

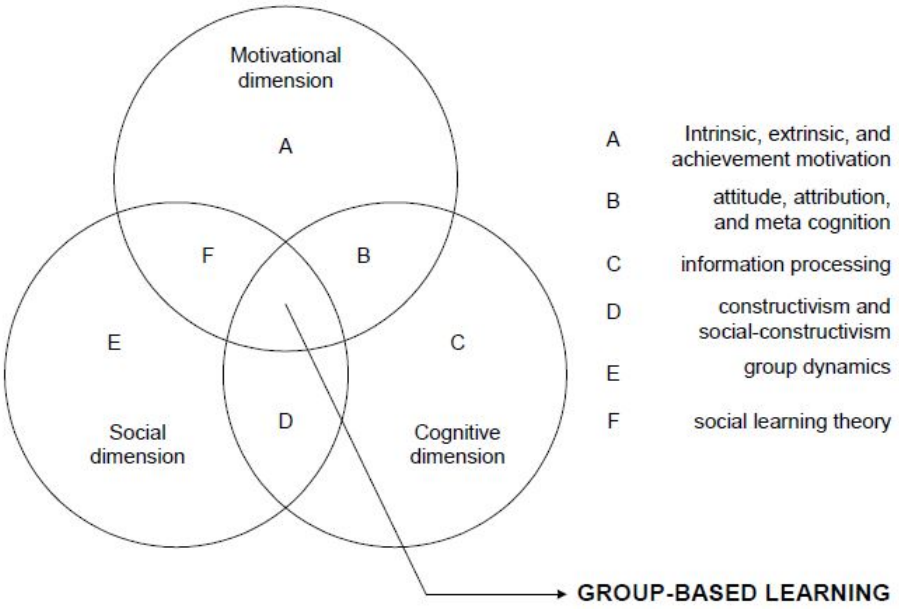


Figure 2.2: Theoretical Perspective of Group-Based Learning [63]

on the other hand, can lead to increased self-confidence, which can raise motivation to learn and also improve social skills. This, in turn, may stimulate cognitive gains through more effective collaboration [63]. Vygotsky's contributions to the field of social constructivism is particularly involved with the collaborative element of learning. According to his theory, problem solving skills of specific tasks can be evaluated based on a student's ability to perform these tasks independently; with help from others; and not at all, even with help from others. The second component takes place in a collaborative environment where individuals can learn from their more knowledgable peers [67]. This ability to solve problems with others is transferable from the classroom to "real life" [54].

Collaborative learning can take place when players play multiplayer online games. Research has shown that, in multiplayer online games, individual players collaborate in teams where each person contributes a different, but overlapping, set of skills [24]. These groups often share knowledge and values with others both inside the game and in real life. During gameplay, teams create a distributed and dispersed knowledge within a community much like in the workplace [24].

2.2 Games

After exploring some educational theories that support the incorporation of game-based learning in school, this section will look at some characteristics of games and gaming. Independent features of games and gaming will be reviewed for their potential to provide educational value. Finally, the engaging nature of games will be discussed.

2.2.1 Definition of Games

Although everyone is familiar with the concept of a game, it is difficult to exactly define what a game is. Given that there are very many different types of games, when one attempts to provide a definition of "games", it seems as though games could be very many things. It is perhaps easier to identify what games are not. Specifically, how do games differ from regular play? In 1949, Huizinga defined play as a free activity taken under by an individual that is consciously outside the "ordinary". Furthermore, this activity is not taken "seriously" but it nonetheless absorbs the player intensely and completely [25]. These are elements that are also found in gaming. However, the main difference between play and gaming lies within the territory that these activities take place. In play, the individual's world is open-ended and make-believe and world-building are key. Contrastingly, games take place in confined worlds where the rules of engagement are defined and gamers are encouraged to interpret and optimize their tactics to fit into this defined space [6].

There is no common definition in literature for "game" and there are varying perspectives from different fields of study. However, most agree that although many different types of games exist, they all share common characteristics. Avedon, in particular, has identified ten structural elements of games by combining the work of mathematicians and behaviourists. These ten elements are [3]:

1. *Purpose of the game*
2. *Procedures for action*
3. *Rules governing action*
4. *Number of required players*
5. *Roles of participant*
6. *Participant interaction patterns*
7. *Results or pay-off*
8. *Abilities and skills required for participation*
9. *Physical setting and environmental requirements (not always present)*
10. *Required equipment (not always present)*

These components can be used to produce a "game-like" activity. An activity becomes increasingly more "game-like" with the presence of more of the characteristics in Avedon's list.

2.2.2 Flow in Games

Perhaps one of the biggest points of interest in games is their tendency to engage players and immerse them into another world. When players are both engaged and immersed within the game, they are said to be experiencing "flow". The concept of flow was first introduced by Csikszentmihalyi after studying people perform activities such as chess and dance [13]. When individuals are experiencing flow, they are said to be in a state of complete absorption and this results in the optimal experience of the activity that elicited this state [13].

Observations have shown that a person is more likely to be in a state of flow when they are performing an activity with a defined set of goals that require specific responses. It is easy to see then that games, with defined goals and rules that make it very clear to the player what to do and how, can be very effective in putting players into flow. Flow can also occur when a person dedicates their skills to solving a problem that is manageable but still challenging. This can, in turn, bring about an intention to learn new skills to cope with new challenges. Games often present obstacles and challenges that can bring about a state of flow for players [14].

Flow can also be considered as a source of mental energy. This is because it can focus an individual's attention and motivate action. Like any other sources of energy, flow can be harnessed for both constructive and destructive purposes. It has been shown that learning, a constructive purpose, is positively affected by flow [68]. As such, educational games should be designed in such a way to induce flow so that the player's learning experience can be optimized. Generally speaking, educational games must strive to provide players with challenges that are related to the main task so that flow is possible. It is important that these challenges relate to the main task and are not, in fact, artifacts that distract from gameplay. Also, if the game offers challenges that match the player's existing skills, the probability of experiencing flow is increased. If the skill required to overcome a challenge far exceeds that of the player's skill, the player may exhibit anxiety, which hinders flow from occurring. Contrastingly, if the task is too simple, the player may feel bored, which will also prevent flow. As a general outline, game designers should design the game such that challenges increase in difficulty as players develop their skills. This should keep the player in a state of flow while playing the game [68].

2.2.3 Competition and Cooperation in Games

Social competition in games arises when a player competes with one or more opponents that can either be controlled by another person or by a computer. When social competition

occurs, a player will perform actions to maintain their own interests at the disadvantage of their opponent. During gameplay, each player will monitor how their individual performance compares to the performance of others and how this difference is expected to continue for the rest of the competition. These social comparisons will affect the player's emotional state, self-esteem, and mood. Better performance will positively affect the player's emotions, self-esteem, and mood, while worse performance will have negative effects [66]. Emotional states are caused by intense feelings that are directed at someone or something, whereas moods are less intense and the often (though not always) lack a contextual stimulus. Most experts believe moods linger longer than emotional feelings. However, both emotions and moods can affect each other. For example, defeating an opponent may cause an individual to experience an intense emotion of joy, which can put them into a good mood over the next few days [26].

Research has found that challenge and competition can be an important reason for the enjoyment felt by computer game players [66]. However, the extent of this enjoyment will differ from person to person, as some individuals may prefer engagement in competitive situations more than others. From this, one can surmise that people that are more comfortable with competitive situations will seek out games that offer this more so than people who are less interested in social comparisons. Individuals may also seek out competitive situations for their psychological consequences. Competition allows one to maintain or enhance their self-esteem and to bring about positive moods. However, it is important to note that not all individuals may seek out competition for these psychological benefits; this is dependent on their social value orientation. A person's actions can be categorized into three outcomes: competitive, individualistic, and cooperative. Individualistic outcomes occur when one tends to their own benefits. While a cooperative orientation means that individuals care about their own benefits in addition to the benefits of others. Finally, a person with a competitive orientation, will maximize their personal benefits in relation to the benefits of others. Those that exhibit a competitive orientation will be most likely to put themselves in a competitive situation [66].

Since not all individuals may see competition as motivation to play games, it may be advantageous to incorporate a cooperative component to games, to motivate those that have a cooperative orientation. This can be done in games that support competitive groups. Within this context, individuals cooperate within a group but compete across groups. Empirical studies have shown that individuals show higher levels of cooperation within their teams when there is competition between groups [7]. This is because of the player's tendency to view their group mates as collaborators rather than competitors. Also feelings of guilt are registered when the individual does not feel that they have contributed as much as their group members [7]. Those that are more individualistic in nature are also more likely to contribute under group competition, as opposed to a standard public situation [57]. Consequently, games that incorporate both competition and cooperation in the form of group competition may increase players' motivation to participate and play the game.

2.3 Game-Based Learning

Having considered learning theories that support the inclusion of games in education and formally defined games, this section talks about game-based learning, a combination of the aforementioned topics.

In recent years, video games have become a pervasive part of everyday life. This is a direct consequence of the introduction of mobile devices such as smart phones and tablets, which allow easy access to video games at any time. Furthermore, the explosive advancement in computing power and graphics enabled the creation of "serious games" such as high definition first-person shooters, driving games, open world games, action-adventure games, and real-time strategy games [36]. Surveys results published in 2008 by the Pew Internet & American Life Project have shown that 76% of students (82% of full-time and 69% of part-time students) who are 18 or older are regular or occasional game players [1]. It is not surprising then that there is a renewed interest in the educational potential of video games [20]. With the rapid progression of video games, contemporary education researchers and game developers both have much to explore. For example, the educational potential in an early text-based adventure game is vastly different from a high-definition action-adventure game. As such, there is a high demand for research-based educational video games that challenge the existing method of game-based learning.

2.4 Rationale for Game-Based Learning

When reviewing the literature on game-based learning, an assumption is commonly made that the rationale for this method is that games are intrinsically motivating. Also, the argument made by supporters of game-based learning is that if the motivational factors associated with playing games could be transferred to learning then the learning would be more successful. However, most of these researchers do not consider that game playing is not necessarily motivating to all individuals. For the individuals that are not motivated to play games, another reason must be made to use games educationally. Another argument for the incorporation of video games in education is the changing profile of modern learners.

Present-day students are drastically different from the students of the past. The current educational system was designed for the students of the past and may no longer fit the needs of today's students. Prensky identifies these modern students as being *digital natives* or "Games Generation" learners [55]. These digital natives have grown up with computer games, television, and other media. Prensky argues that these mediums have taught them how to learn instinctively and, as such, this generation is cognitively different from previous generations. The ten cognitive changes in people of the Games Generation are [55]:

1. Games Generation learners process information at a much faster pace than traditional learners

2. Games Generation learners can multitask and process information from multiple sources at the same time
3. Games Generation learners are drawn towards graphics and images rather than textual information
4. Games Generation learners do not follow a linear path through learning materials
5. Games Generation learners will expect to work collaboratively with others rather than alone
6. Games Generation learners play a more active role in obtaining information and planning their learning
7. Games Generation learners view play and work as being closely related and playing games to learn will feel natural
8. Games Generation learners expect quick reward and feedback and will quickly become demotivated if they do not get rewards for their effort
9. Games Generation learners do not part with fantasy play as they grow up like the traditional learners of the past; they are willing to go a lot further with their imaginations
10. Games Generation learners are quick to familiarize themselves with new technology and welcome change and advancement

It is important to note that these changes in cognition are based on experience, not empirical studies [55]. Nevertheless, they provide a compelling argument for a new approach in education. When one reviews the characteristics of Games Generation learners listed above, it becomes obvious that video and computer games are able to satisfy these modern students' learning needs.

2.5 Edutainment versus Educational Games

Games, without computers, have long been used as a pedagogical tool in instructing students. The fun that students derive from playing these games serve as motivation to remain engaged with the material [59]. With advances in technology, game developers and pedagogical researchers alike realized the potential of multimedia in improving games used in instruction, which could result in higher-quality education. This awareness motivated the creation of the "edutainment" industry in the seventies. Edutainment is a broad term

that represents the union of education and entertainment. It embodies the delivery of knowledge through multiple media platforms, including video games [20]. The eighties and early nineties were spent developing multiple games of diverse genres such as educational adventure games and drill-and-practice games. The development of these games was motivated by the business market, as opposed to cognitive or educational principles. Consequently, by the late nineties, such edutainment games slowly lost their appeal. A majority of the edutainment titles currently left in the market are now targeted towards young and pre-school children [59].

Edutainment and educational games can be differentiated in their interactivity. Edutainment games usually force players to practice repetitive skills or rehearse memorized facts, whereas educational games require players to strategize, test hypotheses, or solve problems. These educational games usually feature a system, rewards, or goals that motivate the players. The content to be learned is also relevant to the narrative plot. Usually, unlike educational games, edutainment titles fail to transmit non trivial knowledge. Nevertheless, edutainment games that employ *skill and drill* have also demonstrated gains in learning [18]. For example, in 2004, researchers found that a math facts game designed for second graders encouraged players to complete a larger number of problems at an increased level of difficulty [38]. This math facts game was played on handheld computers and those that played this game completed nearly triple the number of problems in 19 days than those that used paper worksheets. The learners employing edutainment game also voluntarily increased the degree of difficulty in the game as they continued to play [38].

Chapter 3

Literature Review

Many university students must study the basics of programming because it is relevant in many fields of technology. Unfortunately, some students experience great difficulty in both understanding and applying the basics taught in these courses. The difficulty of the subject stems from its practical nature and the presence of many abstract concepts. The classes of basic programming courses are often large and heterogenous in nature; students exhibit a large range of familiarity with the subject [34]. As such, it is often difficult to design course content that is relevant for experts and manageable by novices.

This chapter will review some of the current methods that have been employed to teach programming. Collaboration-based learning in the form of paired programming will be explored in addition to game-based learning. It is important to understand the current methodology used for instruction and its effects so that improvements can be made.

3.1 Collaboration to Learn Programming

As mentioned in Chapter 2, social collaboration can be highly beneficial for learners. The following criteria has been established for tasks that benefit most from social collaboration [64]:

- *The task is complex or conceptual*
- *Problem solving is desired*
- *Divergent thinking or creativity is desired*
- *Mastery and retention are important*
- *Quality of performance is expected*
- *Higher-level reasoning strategies and critical thinking are needed*

It is clear that these qualities are all inherent in programming. In 1991, Flor [22] observed and recorded verbal and non-verbal conversations between two programmers collaborating on a software maintenance task. He noticed that each member of the pair was able to add their previous experience, task-relevant knowledge, and perspective to the problem through collaboration. This increased the possibility of generating more diverse plans, which lead to a greater ability to solve the problem [22]. His observations are consistent with the advantages of collaborative learning. Furthermore, other research has shown that collaboration is an effective pedagogical approach for introductory programming [9, 15]. Also, collaborative learning has been found to be particularly effective for computer science because of the grand scale of complexity and how it continuously changes [56].

When teaching students about programming, paired programming has often been used as a form of collaborative learning. When students perform paired programming, each pair sits side-by-side at one computer to work on the same program, or problem, at the same time. Research done by Cliburn in 2003 [9] on students in an introductory programming course at a small college shows that pair programming produces better projects in a shorter time-frame. Also most students enjoyed the class more when they were asked to program in pairs and it was perceived that the group work experience would be transferable to the skills required in the industry. Finally, exam scores were comparable with courses that did not incorporate paired programming [9]. Research conducted by McDowell et al. on 600 students in an introductory programming class yielded similar positive results. Like Cliburn, they found that students who programmed in pairs generated better programs and performed equally well on the final exam as those who programmed independently. Interestingly, they found that those that programmed in pairs also completed the course at higher rates [46]. These findings support the notion that social collaboration can be used to great effect in teaching introductory programming.

3.2 Game-Based Learning to Teach Programming

This section begins with a review of the interest students in higher education have in seeing games in computing classes. It will then examine the literature on the use of computer games to instruct software engineering concepts.

3.2.1 Demand and Interest from Students

Research conducted by Kumar and Khurana in 2012 [4] shows that students in the field of computing are showing a great interest in a gamified approach for the learning process. A survey was conducted upon 207 students enrolled in a post graduate program in computer applications (higher education). Three parameters were explored within this survey:

- Identification of the students perspective with the current pedagogy system

- Students' outlook towards games
- Willingness to accept innovation in pedagogies and other factors

Results showed that 86% of the students would like the instructor to incorporated game-oriented pedagogies into classes. 94% of the students believed that fun learning in classes would elevate the quality of classes and encourage comfort with courses by lowering the barriers associated with fear of course content. 87% stated they would be comfortable with new game-oriented pedagogies in place of PowerPoint or whiteboard presentations and videos. 46% of the students believed that motivating students would be the best way to engage students in the classes while another 45% believed that incorporation of new pedagogies in class, such as games, would be best. The remaining 9% believed that mandatory attendance be the best source of engagement [4]. These findings show a desire among students for the transition from traditional pedagogy to game-oriented pedagogies in computer programming courses.

3.2.2 Previous Research

As mentioned in Chapter 2, game-based learning can be highly effective. However, there is still a lack of empirical evidence to suggest that its application in software engineering or computing is effective. It is important to mention again that game-based learning here refers to the use of games as a tool to teach programming; it does not refer to the *development* of games as a method of teaching programming. As such, many popular block-based graphical languages such as StarLogo The Next Generation [70], Scratch [42], Alice2 [31], and Cleogo [10] will not be considered. These novice programming environments are not considered games, since students have to build a game, not play one, to learn programming.

Very little research has been done to explore game-based learning as a method of instruction for programming. Furthermore, most of the research that has been completed does not definitively compare the effects of game-based learning to more traditional methods. Moreover, almost none of the research has evaluated whether the programming knowledge acquired from the game is transferable to contexts outside of the game and none of the games employ collaboration or teamwork in gameplay. In fact, many games have been proposed by researchers, but they have either not been implemented or not investigated [11]. Table 3.1 lists some games that have been proposed by different researchers for implementation in software engineering or computer science education.

From Table 3.1, one can see that the educational effectiveness of the game-based learning approach in teaching programming is severely limited and in many cases non-existent. The following sub-sections will cover the few conclusions that other researchers have drawn after developing and studying the effects of a video game or a computer-based game that teaches programming or computing concepts to students.

Table 3.1: Games Proposed for Implementation in Software Engineering Education

Author(s)	Game(s)	Programming Topics	Evaluation
Rathika Rajaravivarma (2005) [58]	Number and Word Games	basic programming skills: terminal and file IO, string manipulation, control structures, random numbers, arrays, exception handling, algorithm design	No evaluation
Wen-Chih Chang, Yu-Min Chou (2008) [39]	"*Bomberman*" computer game	introductory C programming language	No evaluation
Mathieu Muratet et al. (2009) [49]	Real-time strategy game	basic programming skills (same as above)	Evaluation proposed-results not published
Emily Oh Navarro, Andre van der Hoek (2005) [50]	SimSe-Interactive Simulation Game	software engineering process (project management)	No evaluation
Charles Wesley Ford, Steve Minsker (2003) [23]	"*TREEZ*" computer game	Tree traversal techniques and iterative and recursive traversal algorithms	No evaluation
Andrew Martin (2000) [45]	Simulation game	information systems development	No evaluation

3.2.2.1 ToonTalk-Teaches Abstract Programming Concepts Through Concrete Actions

An example of a game that attempts to teach programming was developed by Kahn in 1999. ToonTalk is an animated interactive world that allows users to build a large range of computer programs. This interactive puzzle game is unique in that players do not construct programs by typing text or arranging icons; it does so by taking actions in its world. For example, the player can perform primitive operations such as addition by building a stack of numbers. In this fashion, ToonTalk strives to make the abstractions for computation concrete. Demos also supplement the puzzle games. In this situation, the player observes a playback, with synchronized narration, of a solution to a programming construct or technique. However, in this passive mode, the learner typically learns superficially, unless puzzle solving or free form constructions proceeds after viewing.

To test the success of this game, two classes of 24 children in a fourth grade class

Figure 3.1: Example of a room and question in LearnMem1 [52] (message is shown in Greek)

were asked to play it. These children had no prior exposure to ToonTalk and only two had any experience with computer programming. They were paired up and observed to use Toontalk for three 40-minute sessions. Nearly all the pairs of children were able to solve the first 25 puzzles without assistance. Since ToonTalk is designed so that each progressive puzzle builds on programming concepts introduced in previous puzzles, the completion of 25 puzzles suggests that the children are successfully learning programming concepts [28].

This experiment did not compare instruction through the game to more traditional methods. As such, one cannot tell whether the video game taught any better than regular instruction would have. Furthermore, Kahn did not test these children outside of the game environment to evaluate whether they are able to apply these programming concepts outside of the ToonTalk world.

3.2.2.2 LearnMem1- Teaches Basic Computer Memory Concepts

LearnMem1 is a game designed to introduce students to basic computer memory concepts; it conforms to the Greek high school computer science curriculum. It is a game that encourages active learning by providing an environment that combines learning material in the form of webpages with game playing. This environment (Figure 3.1) is structured around three rooms in the form of mazes and players travel through these mazes to review different learning materials and to solve questions in progressive levels of difficulty. The student has to accomplish the mission of reaching and collecting a termination flag in each room to successfully complete the game. Obstacles to be dealt with include questions to be

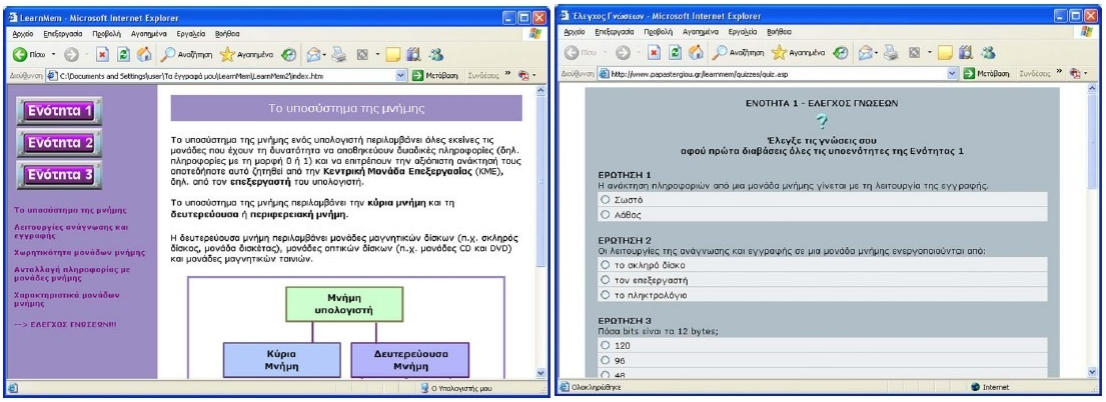


Figure 3.2: Example Learning Material (left) and Quiz (right) in LearnMem2 [52] (message shown in Greek)

answered, doors to be opened, walls to be blasted by setting off bombs, and moving robots to impede the player's progress. Explanatory feedback regarding the player's response follows each question answered. Also, in each room there is a "golden book" that contains a listing of all the learning material the student has access to and a navigation menu for this material.

The player starts the game off with a number of lives and zero points. Their lives decrease by one for ever incorrect answer. Points are gained with correct answers or bonus items found in each room. These bonus items may also provide the player with extra lives. The game terminates when the player loses all of their available lives. Once this happens, the player may choose to resume the game from the last room they were able to reach in the previous game or they may stop playing.

Another non-gaming application called LearnMem2 was used as a non-game-based learning approach. This application had identical learning objectives and content to LearnMem1. LearnMem2 consisted of three thematic units that matched the rooms in LearnMem1. The learning material for each thematic unit was accessible through a navigation hyperlink. After browsing through the learning material, the student was prompted to complete a quiz. Successful completion of the quiz would allow the students to progress to the next thematic unit. The feedback that is provided for each question in the quiz is identical to the feedback given in LearnMem1. Screenshots of LearnMem2 can be seen in Figure 3.2

Research was conducted on two randomly selected high schools in town of central Greece: Trikala. The test population was comprised of 88 total students that attended the course 'Computer Science and Computer Applications'. In each school, participants were randomly assigned to one of two groups. Group A played LearnMem1 and Group B learned through LearnMem2. In total, six classes from the high school participated in the research;

three used LearnMem1 and three used LearnMem2.

Pre-tests and post-tests were used to evaluate the effectiveness of the gaming and non-gaming learning applications. The pre-test and post-test consisted of the same 30 questions on computer memory but the questions appeared in different orders. Analysis of the pre-test showed that there was no statistically significant difference in the performance between the LearnMem1 group and LearnMem2 group [$F(1,86)=2.625$, $p=.109$]. However, there was a statistically significant difference in the performance of the post-test in favour of the LearnMem1 group [$F(1,83)=8.853$, $p=.004$] which suggests that the gaming application was a more effective teaching tool than the non-gaming application. A survey was also provided after the students completed LearnMem1 and LearnMem2 to understand their opinions towards the application. The students that used LearnMem1 found their application to be significantly more appealing and educationally fruitful than their peers that used LearnMem2. The survey revealed that LearnMem1 was deemed to be more engaging, effective, active, and "relaxed" than LearnMem2.

The study was able to demonstrate that the gaming approach was both more effective in teaching computer memory concepts and more motivational to students than the non-gaming approach. However, it is important to note that the gaming approach was not compared to a more traditional approach. Rather, it was compared to another form of ICT (information and computer technology)-based learning. Also, LearnMem1 lacked sophisticated graphic designs, sound effects, and storylines of immersive multiplayer games that students play outside school. From the feedback surveys, students identified this as something they would like to see improved. However, since this simple game was able to have a positive effect on students' knowledge acquisition and motivation, one can surmise that perhaps a more sophisticated game would yield even greater results. Finally, the long term effects of this game was not investigated so it is not clear whether the gaming approach would help students retain the knowledge that they have gained [52].

3.2.2.3 Program Your Robot- Practice Introductory Programming Constructs Through a Game

Program Your Robot is a serious game designed to allow higher education students to practice introductory programming concepts within an environment that allows them to attain skills like algorithm building, debugging, and simulation. The premise of the game is for the player to assist a robot escape from a series of obstacles by generating an escape plan called a *solution algorithm*. These players will build their solution algorithm by giving the robot various commands to perform. The commands can either be categorized as action commands or programming commands. Action commands have a direct effect on the robot; these commands cause the robot to go forward, turn left, etc. Programming commands affect the action commands through functions, decision making structures, and looping mechanisms. All of these commands are dragged from toolbars and dropped into slots to

Figure 3.3: Game Screen for a Level in Program Your Robot [30]

build the solution algorithm. The game screen for Program Your Robot can be seen in Figure 3.3.

The game consists of six levels with each level presenting a different challenge that aims to teach a different programming construct. Each level also becomes progressively more difficult to complete. To pass a level, players must reach a destination point called the teleporter by developing their solution algorithm.

Twenty-five students who were studying degrees within the computer science discipline at University of Greenwich were asked to provide an initial evaluation of the game. Since the participants were studying in different degree programs, their programming knowledge were also considerably different. Participants provided positive feedback and said that they enjoyed the game. They also thought that this type of game-based approach could enhance the problem solving abilities of more novice programmers. Participants also provided feedback on improvement of the game mechanics and user interface.

Although this game is similar to the game developed for this study and the players are similar to the target audience for this study, there is much still remaining to be explored with regards to the educational value of such a game. The study population for Program Your Robot was small so the significance of their feedback is questionable. Also, no tests were conducted to evaluate whether learning took place before and after the game; there were no quantifications. Finally, it would be interesting to see if this game would be well-received if it was incorporated into an introductory programming course [30].

3.2.2.4 Robocode

Robocode was started by Mathew Nelson in 2001 and it is an open source educational game. The purpose of Robocode is to help people learn how to program in Java or .NET framework programming languages. Players write software that controls a miniature tank. These miniature tanks are then placed in a playing field to fight other identically-built (but differently programmed) tanks. The robots can be programmed to move, shoot at, and scan for other robots. The Robocode framework provides a set of rules that every robot has to follow. Re-usable object structures are also available to players to ease the development of robots.

There are several leagues for Robocode. These leagues feature 1-on-1, melee (free for all with more than two bots) and teams competition. The rankings for of each these leagues are also publicly available. International competitions for Robocode are held by international online coding festivals. The Robocode community is very large and diverse.

In one research study, 500 surveys were sent out to Robocode community members and 83 members responded. This survey discovered that 80% of the participants perceived that their programming skills increased after playing Robocode. Results also demonstrated that participants in various education levels and expertise levels all enjoyed playing the game. They also found that the opportunity to learn new programming skills and to have fun was the greatest sources of motivation to participate in Robocode [41].

While Robocode is a successful game with many participants, its incorporation into education may not be as effective. One research study tried to incorporate Robocode into course curriculum but they discovered that many students were able to find open-source code for their robots online. Students were caught attempting to program tanks from pieces of existing code from online sources instead of implementing original designs [5].

3.2.2.5 6 Tablet Video Game Applications to Teach Introductory Science Concepts

In Fall 2012, Browne conducted a series of experiments at McMaster University to investigate the success of 6 tablet video games designed to teach introductory computer science concepts [8]. The primary goal of this study was to determine whether game-based learning or gamification could increase student satisfaction and engagement in course material. Other goals include investigating what game features increase satisfaction and how educational software can be effectively integrated into the classroom.

The 6 independent tablet games were designed to teach the following concepts: binary search, binary numbers, CPU/assembly language, polynomial graphs, quicksort algorithm, and Dijkstra algorithm. All of these games were designed to teach the relevant concepts by representing them interactively. For example, the Binary Search app, shown in Figure 3.4, asks students to find a golden egg hidden in a row of coconuts. When the student taps on a coconut, the coconut will either reveal the golden egg or an arrow that points in the

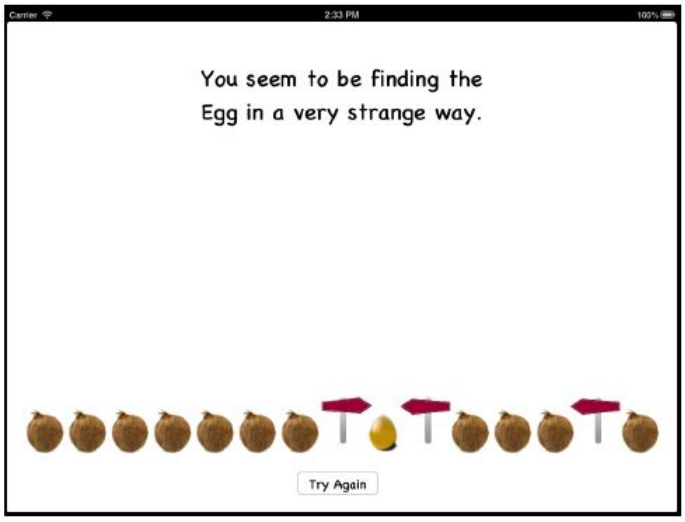


Figure 3.4: Binary Search App [8]

direction of the golden egg. Once the golden egg has been found, if the student did not follow a search sequence that conforms to binary search, the game will suggest to the student to find the egg in a different manner. In this way, the concept is represented in an interactive manner. In addition, game elements such as objectives, rewards, penalties, levels, narrative, multiplayer, and in-game assistance was included in the games at different degrees.

The experiment was conducted on 101 first year computer science students over the span of 6 weeks where one game is introduced every week during a 50 minute lab session. Three identical quizzes, relating to the topic being taught, was issued to students at different points in time during each lab session. One quiz was issued before instruction, another after the first lesson, and then the final quiz at the end of the second lesson. Students were divided into two groups, one group would receive a conventional lecture for their first lesson and then the game for their second lesson. The other group would, alternatively, play the game for their first lesson, and then receive a conventional lecture for their second lesson. In addition to the quizzes, students were also given questionnaires to complete to provide feedback for the usability of each game and to express their attitudes towards game-based learning.

Browne found that participants generally performed better on Quiz 3 than on Quiz 2 for all 6 games, after they had experienced both instructional methods. This result is consistent with questionnaire responses where the majority of participants recommended future instruction should incorporate both traditional and game-based instruction. There was also no consistent trend in differences in performance on Quiz 2 amongst the two groups for the

6 games game. Overall, participants expressed that they preferred instruction with the apps more than through traditional academic instruction. Browne did not investigate whether knowledge obtained through the game could be retained overtime [8].

The success of existing projects with educational games provides the motivation for the game used in this study (Space Race). To contrast with the projects summarized above, the current project has the following characteristics and goals:

- There are a large number of participants (485 students total)
- The educational game will be designed to teach introductory programming concepts to students in higher education (university)
- The educational game will incorporate cooperation and competition
- The effectiveness of the educational game will be quantified
- The longer term impacts of the educational game on learning will be measured (7 weeks after gameplay)
- The transferability of the knowledge gained from the educational game to the real world will be evaluated

Details of the game used for this study, Space Race, can be found in the next Chapter: Game Design.

Chapter 4

Game Design

Space Race is a multi-player Android tablet game that was developed for this research project. Levels 2, 3, and 4 was inspired by **Spaceteam**, a cooperative Android and iPhone game developed by **Sleeping Beast Games 2012**© [62].

This chapter explores the design elements of Space Race that are relevant for research purposes. In particular, the details of gameplay are established and the programming concepts present in the game are listed. For brevity, irrelevant details such as menu design, sound effects, and graphics design have been omitted.

4.1 Design Goals

Two primary goals were set to maximize the effectiveness of the game by motivating students to engage with the content.

1. **Cooperation between students to reinforce learning of programming concepts.**

   Since there is evidence that group discussion and cooperation can be highly benficial for learners, the game should encourage communication between students during gameplay [69]. It should be designed such that progression is dependent upon teamwork between students to maximize cooperation.

2. **Competition between groups of students to motivate learning.**

   An aspect of competition should be included in the game design. This, combined with cooperation, should prompt students to engage participate in gameplay which, in turn, will increase opportunities for learning [7].

Figure 4.1: Leaderboard for Space Race

4.2 Game Overview

This section will summarize Space Race's overall gameplay and level design.

4.2.1 Gameplay

Every player must have their own Android tablet to play Space Race. The game is played in teams of four for all four levels. The players are able to create their own team along with their team names. They may choose to stay in the same teams while playing all four levels, or they may opt to switch teams between levels.

The game features cooperation amongst team members and competition between teams. Players progress through a level by correctly answering or executing a series of questions or instructions served by the game. Game performance is dependent upon the time it takes to complete the instructions in a level; the shorter the time, the better the performance. The best time for each team is recorded per level on a public leaderboard that players are able to access through the main menu in the game (Figure 4.1). The times are listed from best to worst according to team names.

4.2.2 Level Design

The game is comprised of four levels. Each advancing level introduces new programming concepts that rely on a firm understanding of concepts taught in the previous levels. The game mechanics between Levels 2, 3, and 4 are very similar, while the game mechanics of Level 1 are different. The game is designed this way because the concepts introduced in Level 1 are much more basic than the other levels. Each level of Space Race was designed to be completed in an hour or less. On average, teams completed each level within 45 minutes. The programming language being taught in all four levels is Python version 2.7.

4.3 Level 1 Design

The sections below describe the design components of Level 1. They provide design details on the programming concepts introduced, the game controls, the game screen, and the game mechanics.

4.3.1 Programming Concepts

The following programming concepts are introduced in Level 1:

- Python built-in types
  - Boolean type
  - Numeric types: `int`, `float`, `long`
  - Sequence type: `str`
- Python built-in operations
  - Comparison operations: <, <=, >, >=, ==, `!=`
  - Numeric operations: `+`, `-`, `*`, `**`, `/` (emphasis on integer division), `%`, `int()`, `float()`, `round`, `abs()`
  - Sequence operations: `+`, `str()`
- Variable assignment
- Python Built-In Exceptions
  - `TypeError`
  - `NameError`
  - `SyntaxError`
  - `ValueError`

Figure 4.2: Game Screen for Level 1

4.3.2 Game Screen

Figure 4.2 depicts the game screen for this level. A timer, the team's high score (if applicable), the overall high score for all teams (if applicable), and the number of questions completed out of 45 are displayed on the game screen. The game screen also includes the following elements:

1. **Question Monitor**- displays a block of Python code and the last line asks the players the value of a single variable
2. **Answer Buttons**- provides four distinct values for the variable in question shown in the Question Monitor, with one correct answer
3. **Panel Type**- displays the individual player's Python data type
4. **Correctness Indicator**- indicates whether a chosen answer is "Correct" or "Incorrect"

4.3.3 Game Controls

The player can interact with Level 1 through four touch buttons and a shake motion. The four touch buttons are used to select answers and the shake motion is used to indicate an Error response.

4.3.4 Game Mechanics

The gameplay of Level 1 is very similar to that of a multiple choice quiz game. However, modifications were made to ensure the design goal of cooperation between team members is attained.

Every player on the team of four will view the same game screens. The only difference in information being displayed is the Python data type shown in the Panel Type display. This data type is randomly assigned to each player at the beginning of the game and they may be one of the following: `int`, `str`, `bool`, or `float\long`. Every player on the team will have a different data type.

The objective of this level is to answer all the questions in the shortest time possible. Since all four players will see the same question in the Question Monitor and the same four values for the Answer Buttons, the player to select the correct answer must have the data type that corresponds to the data type of the variable in question in the Question Monitor. Refer to Listing 4.1 for a simple example of a question that could be shown in the Question Monitor.

Listing 4.1: An example of a question shown in the Question Monitor of Level 1

```
1 X= 6
2 Y= 5
3 Z= X+Y
4
5 Z=?
```

Assume `10`, `11`, `12`, `13` are the values for the four Answer Buttons given the question in Listing 4.1. The player with the Panel Type of `int` must be the player to select the answer of `11`, since the data type of variable `Z` (line 5) is also of type `int`. For the question to advance, the correct answer must be selected by the player with the correct data type. This encourages the team to converse and generate the correct answer together.

In some cases, the code block shown in the Question Monitor may raise Python exceptions (ie. there is an error in the question). An example of this is shown in Listing 4.2.

Listing 4.2: An example of a question with an error shown in the Question Monitor of Level 1

```
1 X=6
2 Y==X
3
4 Y=?
```

For this question, since a Python `NameError` exception is raised by line 2, all four players must shake their tablets to acknowledge this error and advance to the next question.

The level is complete when all 45 questions have been answered correctly. Each iteration of this level generates the same questions in the same order. The questions served by the game are shown in Appendix A. This ensures that every team receives the same questions and each team has an equal opportunity to improve their overall time with every iteration of this level. However, the data type that the players are assigned may differ through different iterations of Level 1, since the data type is randomly assigned to each player.

4.4 Level 2 Design

The programming concepts, game controls, game screen, and game mechanics of Level 2 are outlined in the sections below.

4.4.1 Programming Concepts

In addition to the programming concepts introduced in Level 1, the following concepts are introduced in Level 2:

- Python built-in operations
  - Sequence operations: indexing and slicing `s[i]`, `s[i:j]`, `len()`
- Python Built-In Exceptions
  - `ZeroDivisionError`

For this level, students are provided with a "Cheat Sheet" that concisely summarizes the concepts that are introduced in this level. This "Cheat Sheet" can be found in Appendix B.

4.4.2 Game Screen

The game screen for this level can be seen in Figure 4.3. The game screen for this level includes a timer, the team's high score for that level (if applicable), the overall high score for that level for all teams (if applicable), and the number of instructions executed out of 112. The game screen also includes the following:

1. **Instruction Display**- displays a unique instruction to be executed
2. **Control Panel** (also simply called Panel)- displays the game controls with each control labeled by a unique control name and paired with its submit button

Figure 4.3: Game Screen for Level 2

3. **Previous Values Display**- displays the most recent correct value for each game control on the control panel, if applicable

4. **Correctness Indicator**- indicates whether a submitted control value is "Correct" or "Incorrect"

Every player will have the same controls but all the controls will have unique names. That is, the control names are never duplicated; either on one panel or across different panels.

4.4.3 Game Controls

All of the touchscreen controls that the player can interact with for this level are displayed together on a single control panel on their screens (seen in Figure 4.3). Each touchscreen control on the panel is paired with a touch button that is used to submit that control's selected value. These **Submit Button**s will turn green if the control's submitted value is correct, red when the control's submitted value is incorrect, and yellow if the incorrect control has been changed.

The interface for the controls was selected to be as intuitive as possible by making use of "cultural conventions". Norman [51] describes cultural constraints as conventions that are shared by a cultural group. For example, the fact that one should click on a button through touch to cause a desired effect is a cultural, learned convention [51]. These commonly learned conventions by tablet users was applied to the game design to make it more intuitive

for players to interact with the game. The following controls that these players can interact with are labeled with a unique name and are shown on each player's control panel:

- two **NumberPicker**s with a predefined range of 0-5
  - Players swipe up and down on a wheel-like mechanism to select an integer value from its predefined range
  - Used to represent Python `int` values
- an **Alphapad** with keys `A`, `B`, `C`, `D`, and `del` for deleting characters
  - The typed characters are shown in a display above the keypad
  - Used to represent Python `str` values
- a **Numpad** with integer key values `0-9`, a decimal (`.`), and `del` for deleting characters
  - The typed value is shown in a display above the numpad
  - Used to represent Python `int` and `float` values
- a **Switch** with values `True` or `False`
  - Players swipe the control left and right to select `True` or `False`, respectively
  - Used to represent Python `bool` values

In addition to the touchscreen controls, players can also trigger a game event with a shake motion. This shake motion is, again, used to acknowledge a Python exception.

4.4.4 Game Mechanics

The game mechanics of Level 2 are different from Level 1, but they are also designed to encourage cooperation amongst team members.

The objective for this level is for players to execute all the instructions in 4 different programs in the shortest time possible. Before the game begins, each player is randomly assigned a unique program from four available programs. During the game, the players are served one instruction at a time from their assigned program. A new instruction is served to a player when the previous instruction has been correctly executed. Instructions indicate the value that controls on the control panel should be set to. It is possible for a player to receive instructions for a control on their own control panel, or for a team member's control panel.

Figure 4.4a is an example of a possible game scenario in Level 2. Player 1 has received the instruction `Aileron= 9%2`. This is an example of an instruction a player can receive

that affects a control on their own panel. To correctly execute the instruction, Player 1 must set their `Aileron` control to the value of `1` and press the matched Submit Button. Once the instruction has been cleared, Player 1 will be served the next instruction from their program.

The game scenario presented in Figure 4.4 also demonstrates an instance where a player's given instruction does not refer to a control on that player's control panel. Player 2 has been served the instruction `Message= "DAB"`. However, the `Message` control is found on Player 4's control panel. In this instance, Player 2 must verbally communicate to Player 4 that `Message` should be set to `DAB`. Once Player 4 completes this task correctly, Player 2's instruction will be cleared.

Finally, Player 4 is presented with an instruction that raises a Python `SyntaxError` in Figure 4.4d. To clear this instruction, every team member must shake their tablets to acknowledge the exception.

Similar to a regular computer program, some instructions will require controls to be set to a new value that is dependent on that control's current value or another control's current value. As an example of this, a possible instruction a player may receive is: `Feedback= Feedback+"C"`. The player with the `Feedback` control would then refer to their Previous Values Display to determine the current value of `Feedback`. This current value is the previously correct value that the control was set to. Assuming that the current value of `Feedback` is `DCA`, the player would have to set `Feedback` to `DCAC` to clear the instruction.

Table 4.1: Level 2 Programs and Panel Distribution

Program 1			Program 2		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
1	Aileron	`int`	2	Flaps	`int`
4	Elevator	`int`	1	Slats	`int`
4	Rudder	`float/int`	1	Hover	`float/int`
3	Text	`string`	4	Message	`string`
2	Break	`bool`	3	Horn	`bool`
Program 3			Program 4		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
3	Torque	`int`	4	Roll	`int`
2	Pitch	`int`	3	Yaw	`int`
2	Yoke	`float/int`	3	Altitude	`float/int`
1	Speaker	`string`	2	Feedback	`string`
4	Light	`bool`	1	Lock	`bool`

(a) Player 1

(b) Player 2

(c) Player 3

(d) Player 4

Figure 4.4: Example Level 2 Space Race Game Scenario

Since a control's value may depend on that control's previous value or another control's current value, it is imperative that the game be designed to ensure that multiple players will not provide conflicting instructions for a single control. The following design decisions were made to avoid conflicts:

1. Each player is randomly assigned one of four program/panel pairs:
    - Program 1 with Panel 1
    - Program 2 with Panel 2
    - Program 3 with Panel 3
    - Program 4 with Panel 4
2. Each program will only affect 5 predefined variables of set types
3. The 5 predefined variables are preassigned to one of four panels

Table 4.1 illustrates the program/panel pairs that are available for Level 2. For example, if a player is assigned Program 1, that player would also be assigned the controls associated with Panel 1 on their panel: Aileron (`int`), Slats (`int`), Hover (`float/int`), Speaker (`string`), and Lock (`bool`). The Program 1 instructions the player receives would affect controls Aileron (`int`), Elevator (`int`), Rudder (`float/int`), Text (`string`), and Break (`bool`). This setup ensures that each player will receive the same distribution of control types on their panel as well as in their program instructions. Furthermore, the distribution is designed such that each player is guaranteed to provide instructions for every team member. For each program, there exists two controls that belong to the same panel (ex. Elevator and Rudder for Panel 4 in Program 1). This is because these paired controls will have instructions in the program that depend on each other's values.

The game is designed to maximize player involvement and multi-player interactions throughout the level. To accomplish this, a pattern is set for the order that instructions from a program are issued for each team member and, consequently, each panel. To illustrate, Program 1 has instructions that affect the panel controls in this order: Panel 1, Panel 2, Panel 3, and then Panel 4. Every program follows a different pattern; this is shown in Appendix A. This ensures that a player will not repetitively provide instructions to the same team member. It also serves to guarantee that every player will begin the game by providing instructions to a different player. The only exception to this pattern is when an instruction with an error/exception occurs. For those scenarios, every player is involved, since an error can only be cleared when all team members shake their tablets.

For research purposes, the game has to be designed to ensure that every player on the team receives the same learning experience. To achieve this, every program has the same type of instructions. However, the type of instructions do not occur in the same order

across different programs. For instance, `Aileron = 9%2` is instruction # 1 from Program 1 and `Torque= 5%3` is instruction # 4 from Program 3. These instructions are of the same type; they both test the player's understanding of how the modulus (`%`) operator functions. However, they both occur at different points in their respective programs. By the end of the level, all the players will have given and received a set of instructions that covers the same range of topics as everyone else.

With the way the game is designed, it is possible for players to finish all the instructions for their programs at different points in time. A player will be given the instruction to "`Listen to teammates!`" if they have completed their programs before everyone else. They will then have to continue to participate in the game by executing the instructions verbally communicated from their team members. The level ends when all four programs have been executed to completion.

4.5 Level 3 Design

Like Level 1 and 2, the programming concepts, game controls, game screen, and game mechanics of Level 3 are described in the subsequent sections.

4.5.1 Programming Concepts

In addition to the programming concepts introduced in Level 1 and 2, the following concepts are added in Level 3:

- Python built-in types
  - Sequence type: `list`- including nested `list`s of one level
- Python built-in functions
  - `range()`
- Python file input
  - `open()`, `file.read()`, `file.readline()`, and `file.readlines()`
- Python built-in operations
  - Sequence operations: indexing of nested lists `s[i][j]`, slicing `s[i:j:k]`
  - Mutable sequence operations: `s.index()`
- Python built-in string method

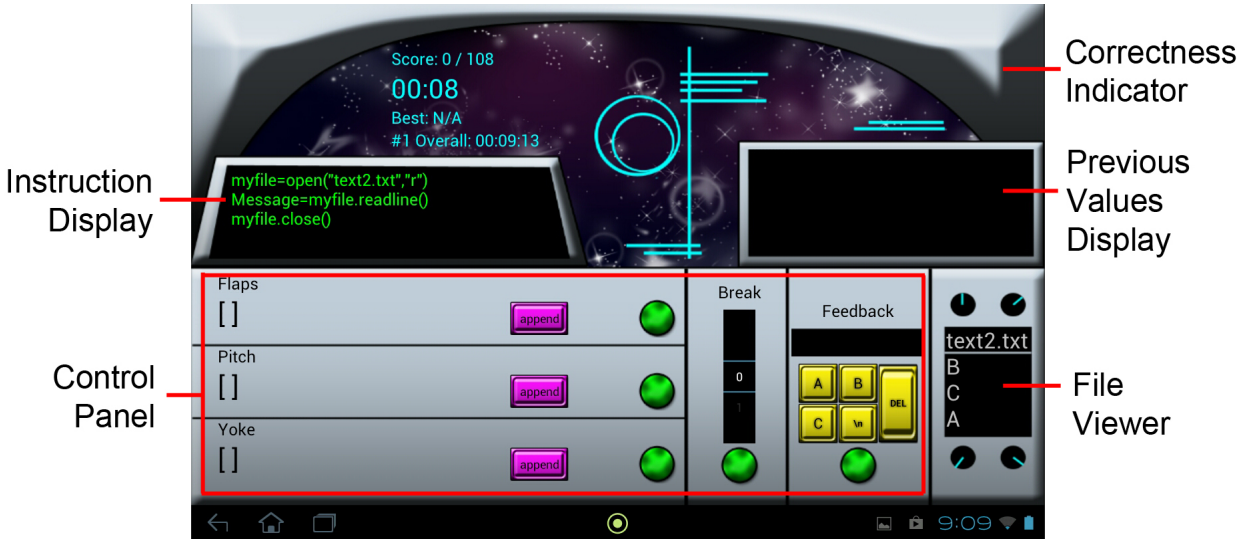


Figure 4.5: Game Screen for Level 3

  - `str.split()`

- Python built-In exceptions
  - `IndexError`
- Aliasing

Similar to Level 2, students are provided with a "Cheat Sheet" that summarizes the concepts that appear within this level. This "Cheat Sheet" can be found in Appendix B.

4.5.2 Game Screen

Figure 4.5 shows the game screen for this level. The game screen for this level includes all of the same information that is displayed in Level 2. However, the number of instructions to be executed are out of 108 instead of 112. Also, a new display component is introduced:

1. **File Viewer**- displays content within a text file

As with Level 2, every player will have the same controls but all the controls will have unique names.

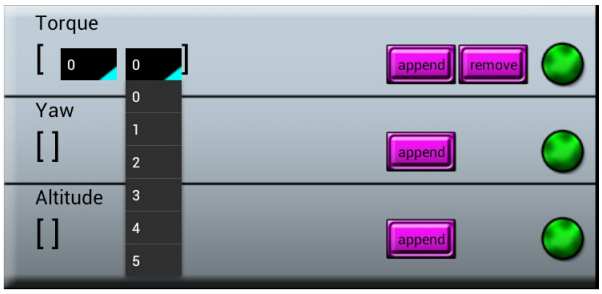


Figure 4.6: `List`- type Controls in Level 3

4.5.3 Game Controls

The game controls for Level 3 follow the same scheme and behaviour as Level 2; each player has a set of controls with their corresponding **Submit Button**s on their panel. The **Submit Button** in Level 3 functions in the same way as Level 2.

The main difference between the control panels for Level 2 and Level 3 is the introduction of a control that represents a new type: Python `lists`. Also, controls of type `bool` and `float/int` are absent in Level 3. A listing of the controls available in Level 3 are provided below. Descriptions for the controls that were mentioned previously in Level 2 are omitted.

- a **NumberPicker** with a predefined range of 0-5
- an **Alphapad** with keys `A`, `B`, `C`, `\n`, and `del` for deleting characters
- three groups of controls with each group consisting of five **Spinner**s, an Append **Button**, and a Remove **Button** (shown in Figure 4.6)
  - Used to represent Python `list` values
  - Each **Spinner** is a drop-down menu with predefined values that represent the list's elements
    - an `int_list`= a group of 5 **Spinner**s with preset values: `0`, `1`, `2`, `3`, `4`, and `5`
    - two `str_list`(s)=2 groups of 5 **Spinner**s with preset values: `""`, `"A"`, `"B"`, `"C"`, `"A\n"`, `"B\n"`, and `"C\n"`
  - The visibility of the **Spinner**s are used to represent the presence/absence of elements in the `list`. If all the **Spinner**s in a group are invisible, the list is empty.

- The Append **Button** is visible when there are less than 5 **Spinner**s visible. When clicked, a new element will be added to the `list` by making the next **Spinner** visible.
- The Remove **Button** is visible when there is one or more **Spinner**(s) visible. When clicked, the last element in the `list` will be removed by making the right-most **Spinner** invisible.

In addition to the touchscreen controls, players can also trigger a game event with a shake motion. This shake motion is, again, used to acknowledge a Python exception.

4.5.4 Game Mechanics

The objective and game mechanics for Level 3 are the same as Level 2. The only difference lies in the controls/control types that are available and the introduction of files. As with Level 2, all the players are randomly assigned a program and panel pair. For this level, the program and panel pairs are also matched with a specific file. This is shown in Table 4.2. The file I/O instructions in any given program will only refer to the file that is matched with that program. For example, a player assigned with Program 1 may receive the instruction shown in Listing 4.3. One can see that "`text1.txt`" is opened on line 1. This file is the file matched with Program 1. This design limits confusion by ensuring that a player will never have to ask their team members for the contents of an opened file. Furthermore, it brings the player's focus towards understanding how the lines of code should be interpreted rather than where the file is located.

Listing 4.3: An example of a file I/O instruction from Program 1 of Level 3

```
1 myfile= open("text1.txt","r")
2 Text= myfile.read()
3 myfile.close()
```

The programming concept of aliasing is also presented for the first time in Level 3. Since aliasing involves the access of a data location in memory by different symbolic names in the program, the game must be designed to reflect this notion. This means that any instruction involving aliasing must require the player to set all the aliased variables/ controls to the appropriate values. Listing 4.4 is an example of an instruction that involves aliasing. For this example, `Rudder` and `Elevator` are aliased. As such, both controls need to be set to the correct value and submitted before the instruction is cleared. A hint in the form of a comment (seen on Line 3 in Listing 4.4) reminds the players that two control must be changed because this gameplay differs slightly from what players are familiar with.

Table 4.2: Level 3 Programs, File, and Panel Distribution

Program 1			Program 2		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
1	Aileron	int_list	2	Flaps	int_list
4	Elevator	str_list	1	Slats	str_list
4	Rudder	str_list	1	Hover	str_list
3	Text	string	4	Message	string
2	Break	int	3	Horn	int
text1.txt			text2.txt		
A B C			B C A		
Program 3			Program 4		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
3	Torque	int_list	4	Roll	int_list
2	Pitch	str_list	3	Yaw	str_list
2	Yoke	str_list	3	Altitude	str_list
1	Speaker	string	2	Feedback	string
4	Light	int	1	Lock	int
text1.txt			text2.txt		
C A B			B A C		

Listing 4.4: An example of an instruction involving aliasing in Level 3

```
1 Rudder= Elevator
2 Rudder[2]= "A"
3 #Hint: Two controls changed!
```

4.6 Level 4 Design

The programming concepts, game controls, game screen, and game mechanics of Level 4 are described in the ensuing sections.

4.6.1 Programming Concepts

The programming concepts included in Level 4 are an extension of the topics covered in Levels 1 through 3. The primary emphasis of Level 4 is `for`- loops. The topics covered also include the introduction of the following:

- Python built-in functions
  - `list()`
- Python built-in list methods
  - `list.append()`
- Python built-in string methods
  - `str.strip()`
- Python flow control tools
  - `for` statements- including nested `for`-loops of one level
- Python file input
  - `for` line `in` file

The "Cheat Sheet" provided for players for this level can be found in Appendix B.

4.6.2 Game Screen

Like the previous two levels, every player will have the same controls, but all the controls will have unique names except for the **Terminate Loop** button. Figure 4.7 depicts the game screen for Level 4. The information available in Levels 2 and 3 are also available in Level 4. The instructions to be completed are out of 102 for this level. Unlike Level 3, all the players will view the same file in their **File Viewer**s. The contents of this file are shown in Listing 4.5.

Listing 4.5: Level 4 test1.txt file content for all players

```
1 C
2 A
3 C
```

Figure 4.7: Game Screen for Level 4

4.6.3 Game Controls

The game controls for Level 4 are very similar to Level 2 and 3. The available controls in Level 4 are listed below and descriptions for controls that have appeared in previous levels have been omitted for conciseness.

- a **NumberPicker** with a predefined range of 0-5
- an **Alphapad** with keys `A`, `B`, `C`, `\n`, and `del` for deleting characters
- two groups of controls with each group consisting of five **Spinner**s, an Append **Button**, and a Remove **Button**
  - Used to represent Python `list` values
  - Each **Spinner** is a drop-down menu with predefined values that represent the list's elements
    - an `int_list` with preset values: `0`, `1`, `2`, `3`, `4`, and `5`
    - a `str_list`(s) with preset values: `""`, `"A"`, `"B"`, and `"C"`
- a **Button** labelled "`Terminate Loop`" is used to indicate termination of a `for`-loop

A shake motion is, again, used to acknowledge the presence of a Python exception in the instructions of any of the programs.

4.6.4 Game Mechanics

The game mechanics for this level are similar to Level 2 and 3 but there are some significant differences because of the addition of `for`-loops. To begin with, Level 4 shares the same goals as Level 2 and 3 with regards to the completion of all instructions in 4 programs. These 4 programs are, once again, paired with a panel and randomly assigned to each player at the beginning of the game. As mentioned previously, these panel and program pairs are matched with the file shown in Listing 4.5. The program and panel pairs can be seen in Table 4.3.

Table 4.3: Level 4 Programs and Panel Distribution

Program 1			Program 2		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
1	Aileron	`int_list`	2	Flaps	`int_list`
1	Elevator	`str_list`	2	Slats	`str_list`
1	Rudder	`string`	2	Message	`string`
1	Text	`int`	2	Hover	`int`
Program 3			**Program 4**		
Panel#	Control Name	Control Type	Panel#	Control Name	Control Type
3	Torque	`int_list`	4	Roll	`int_list`
3	Pitch	`str_list`	4	Yaw	`str_list`
3	Speaker	`string`	4	Feedback	`string`
3	Yoke	`int`	4	Altitude	`int`

It is important to note that these 4 programs are no longer separate and distinct programs. The programs follow a pattern; instructions alternate between one Level 2 and 3-style instruction and a group of instructions for a `for`-loop. For simplicity, moving forward, the Level 2 and 3- style instructions will be referred to as "regular instructions". All the regular instructions in a given program for this level will affect the controls on the panel that is paired with that program. After the completion of one regular instruction, all the players will be told to "`Wait for other teammates`" until every team member executes their regular instruction. Once this occurs, all the players will advance to the same `for`-loop instructions. One may notice from Table 4.3 that players will not be providing instructions to team members with different control panels. This is done to facilitate the gameplay for the `for`-loop instructions.

From Appendix A, one can see that the `for`-loop instructions are the same across all four programs. The first `for`-loop instruction is shown in Listing 4.6. It is worth noting again that all four players will see the same for-loop instruction at the same point in time.

Listing 4.6: First `for`-loop instruction for Level 4

```
1 for Rudder in range(3):
2     Hover= Hover+1
3     Torque.append(Rudder)
4     Roll[Rudder]=Rudder
```

The controls that are affected in this group of instructions are `Rudder` from Program 1, `Hover` from Program 2, `Torque` from Program 3, and `Roll` from Program 4. Since one control from each panel is affected in the group of instructions, all of the team members will be able to participate in the execution of the `for`- loop.

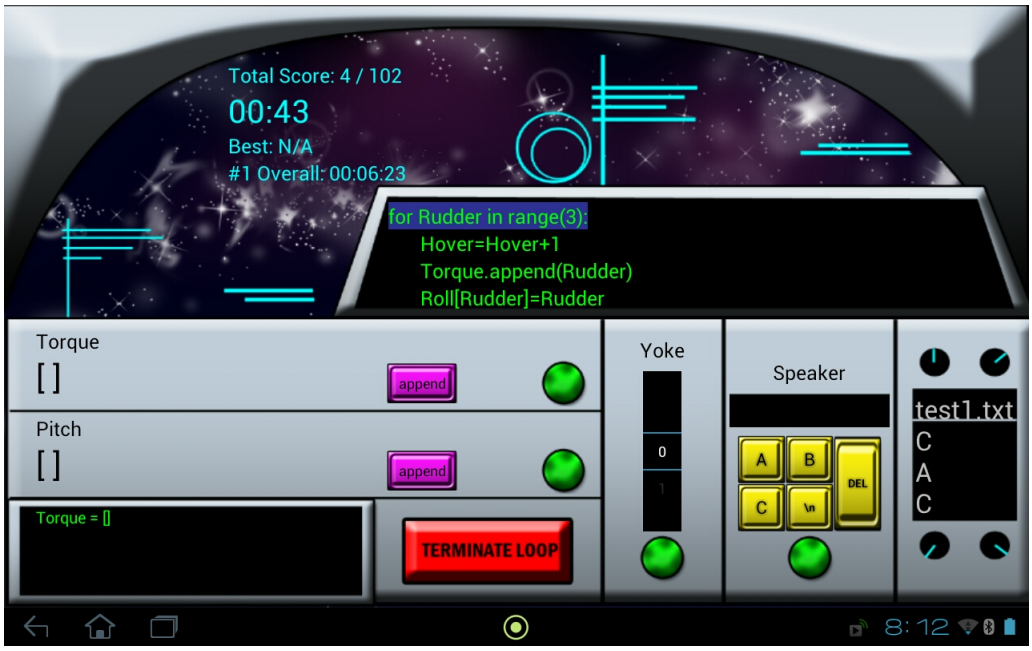
To start with, the first line from the group of `for`-loop instructions is highlighted (shown in Figure 4.8a). This highlighter indicates the line to be executed. The highlighter will advance once the instruction is executed by the correct player 4.8b. After the loop body is executed, the highlighter will loop back to the first line, much like a `for`-loop in programming. For the example shown in Listing 4.6, the highlighter will travel through the loop 3 times and end at Line 1. Once the loop has been executed to completion, all 4 players must press the **Terminate Loop** button to indicate that the `for`-loop has terminated. This will then prompt the game to serve the next set of regular instructions.

This gameplay for the `for`-loop forces the players to traverse through the loop like a computer program. When the players take turns executing their instructions repeatedly, they receive first-hand experience with the looping mechanism of a `for`-loop.

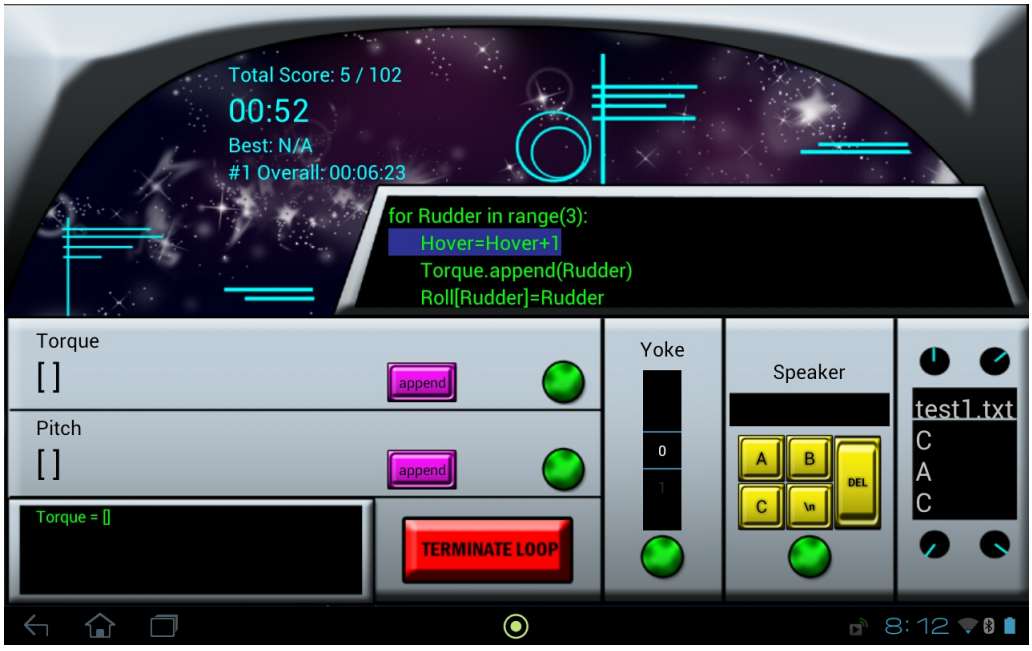
For research purposes, the gaming experience was designed to be as equal as possible amongst different players. To that effect, all of the programs have the same types of regular instructions. These same-type regular instructions may occur at different time points across different programs. To further equalize the gaming experience for different team members, a group of `for`-loop instructions generally places players in different "roles" and the players take turns fulfilling these different roles between different `for`-loop instructions. These "roles" are either variables that play an essential part in determining how a loop iterates or they are variables that are directly affected by the looping mechanism. Listing 4.7 is used to classify these roles. From Line 1 in Listing 4.7, the player with control `Speaker` plays the role of the "Iterating Variable" while the player with the `Feedback` control plays the role of the "Sequence Variable". The player responsible for the `Hover` control in Line 2 and the player responsible for the `Aileron` control in Line 3 plays the role of the "Loop Body Variable"s.

Listing 4.7: Second `for`-loop instruction for Level 4

```
1 for Speaker in Feedback:
2     Hover= Slats.index(Speaker)
3     Aileron.append(Hover)
```



(a) First Line Highlighted



(b) Second Line Highlighted

Figure 4.8: Level 4 Space Race For-Loop Example

These roles do not always appear in the same proportion across different instances of `for`-loop instructions. To exemplify this, a group of instructions with nested `for`-loops may have two "Iterating Variable" roles and two "Loop Body Variable" roles. However, every effort has been made to distribute these roles evenly across all players. In other words, every player should experience each role approximately the same number of times. This, combined with the fact that all four players simultaneously see the same block of instructions should serve to equalize the gaming experience.

Like Level 2 and 3, the level ends when all the instructions from all 4 programs have been executed correctly.

Chapter 5

Experimental Procedure

This chapter presents an overview of the experimental procedures for this study. It will begin with a description of the participants and the participation rates for each level of Space Race. Continuing on, the selected methodologies for data collection will be outlined as well as the data collection timeline. This chapter will not describe the results of the data, it simply describes what type of data will be collected for research purposes. The results will be discussed in Chapters 6, 7, and 8

5.1 Participants

The study was conducted with voluntary participants from the first-year engineering student population of McMaster University in the Winter Term of the 2013-2014 Academic Year. Specifically, these 485 students were enrolled in ENG 1D04-Engineering Computation, a 12 week introductory programming course. There were no restrictions placed on these volunteers based on their characteristics (e.g. age, gender, location, affiliation, etc.). All participants were informed of the risks involved with participation as approved by the McMaster Research Ethics Board.

This group of participants was further divided into two groups: one control group that did not play Space Race and the other group that did play Space Race. The participants of Space Race will henceforth be identified as the experimental group. The members of the experimental group were offered an incentive in the form of bonus marks for ENG 1D04 for participation Space Race. Specifically, 0.5% was added to the student's final ENG 1D04 grade for every level played with the opportunity to earn an extra 0.5% if the student played all four levels. This adds up to a maximum of 2.5% bonus ENG 1D04 marks if the volunteer played all four levels of Space Race. Since students were not required to play all four levels, the number of participants in the experimental and control group varied across the four levels. Table 5.1 shows the number of participants in each level of Space Race.

Table 5.1: Space Race Participation Numbers

Level	Experimental Group	Control Group
1	236	249
2	214	271
3	194	291
4	190	295

5.2 Surveys and Assessments

This section provides an overview of the methods used to obtain data relevant to this study. It includes a description of the surveys and quizzes administered to the student population, as well as the ENG 1D04 exams that were used as an indicator of course performance. In addition, a summary of the data collection timeline is included.

5.2.1 Surveys

The entire population was given the opportunity to participate in Survey A (seen in Appendix D). The purpose of this survey is to assess their pre-existing knowledge in programming. It also measured the students' habits with video games as well as their attitudes toward video games and their inclusion in higher education. It features 26 Likert scale [40] questions, 6 multiple choice questions, and 2 short answer questions. The Likert scale [40] questions had a response range of *Strongly Agree, Agree, Neither Agree nor Disagree (neutral), Disagree, and Strongly Disagree*. The multiple choice questions had options that were specific to each question. The survey was completed online by the student population and completion was optional.

The experimental group was also asked to complete Survey B online upon the completion of each level of Space Race (seen in Appendix D). This means that the experimental group had the opportunity to complete this survey 4 times. Students used this survey to evaluate the perceived usability, effectiveness, and quality of the game. Furthermore, this survey allowed students to report how the game affected their motivation to study ENG 1D04 course material. It features 25 Likert scale [40] questions that had a response range of *Strongly Agree, Agree, Neither Agree nor Disagree (neutral), Disagree, and Strongly Disagree*. It also included 4 short answer questions. Completion of the survey was, again, voluntary.

The Likert questions of Survey B can be broken down into four categories: playability, learnability, team cooperation, and competition. Questions relating to playability asks students to rate Space Race according to their user experiences and their perceived enjoyment of the game. Learnability questions are used to gauge a student's self-perceived learning and relevance of the educational material in Space Race. The success to which communica-

tion and cooperation is used as a means of teaching through Space Race is assessed through team cooperation questions. Finally, competition questions are used to indicate whether students perceive the competition aspect of Space Race to motivate repeated gameplay and revision of course content. The Likert scale questions were chosen because this type of scale is used in many questionnaires so it would be familiar to participants and it is also relatively straightforward to develop [60]. A five-point scale is chosen because it gives a significant level of discrimination without forcing the participant toward an opinion. The results of these surveys are discussed in Chapter 6 and 7.

5.2.2 Pre and Post-Game Quizzes

In many experimental design studies a pre-test followed by the intervention or treatment, followed by a post-test is commonly used to collect performance data. The difference between the pre and post-test is used to measure the effectiveness of the intervention in producing a desired result. Game-based learning research, in particular, has used this technique to measure whether learning takes place after exposure to the educational game. For example, this approach has been used to great effect by Ebner and Holzinger in their study of an educational game in civil engineering [19]; Vivrou, Katsionis, and Manos in their study of the educational effectiveness on an educational virtual reality game [44]; and Papastergiou in her study of digital game-based learning in high school computer science education [52].

For this experiment, two identical quizzes were given to the experimental group to complete before and after playing a level in Space Race (seen in Appendix C). The quizzes feature 5 multiple choice questions that relate to the material covered in the level to be played. Solutions to the quiz are not offered after completion of the pre or post-quiz. The quizzes are completed once by the students on the the Android tablets. That is, the students are not asked to complete the quiz again if they choose to re-play the level. All questions have to be completed before the quiz can be submitted. The students were also instructed to complete the quiz individually.

The difference in performance in pre and post-quizzes is an indicator of whether Space Race was able to immediately teach students certain programming concepts. Improved performance on the post-quiz suggests that learning took place after the intervention of Space Race. Conversely, if performance on the post-quiz is worse than that of the pre-quiz, this would indicate that Space Race is teaching the incorrect concepts to students. Finally, equal performance on both quizzes would suggest that Space Race has had no influence on student knowledge. While this pre and post-quiz technique is great for showing the immediate effects of Space Race, it cannot indicate whether learned knowledge is retained by students over time. The approach for measuring this is discussed in the next section. The results for these pre and post-game quizzes are presented in Chapter 8

5.2.3 Course Exams

Exams used in ENG 1D04 are completed by the entire student population and thus, differences in performance on the exams between the control and experimental group can be used to show that Space Race has caused different levels of learning. Furthermore, since the majority of exam questions are designed by the instructor of ENG 1D04, and not by the researcher, the questions should have a minimal bias. That is, the questions will not be designed to target the material covered in Space Race explicitly. As a result of this, the exam can also indicate whether the concepts learned in Space Race can be applied to a context that is unrelated to Space Race. Finally, since the final exam takes place at the end of the term, 7 weeks away from when Space Race is last played, it can serve as an indicator on the retention of material learned in Space Race.

ENG 1D04 uses two midterm exams and one final exam to evaluate students' proficiency in the course material. All of the questions on the exams are in a multiple-choice format and they cover material taught in the course. Midterm 1 tests all of the content covered in weeks 1 through 5 while Midterm 2 covers material taught in weeks 6 through 9. The final exam tests students on the information covered in the entire course, from week 1 through to week 12.

All of the exam questions are designed and set by the course instructor with the exception of three questions. Questions 5, 6, and 7 in the final exam were designed and set by the researcher. These questions were selected from the pre and post-quizzes; question 5 from Level 3, question 6 from Level 2, and question 7 from Level 1. The results from these questions are presented in Chapter 8 and they will be used to gauge whether knowledge directly obtained from Space Race has been retained.

5.3 Data Collection Timeline

Table 5.2 outlines the weeks that performance data was collected. Starting from Week 3, each Space Race level is available for students to play with for one week. This is because the content of each Space Race level is designed to incorporate the concepts taught in the ENG 1D04 lectures, labs, and tutorials from the previous week. For example, Space Race Level 1 covers the content taught in the ENG 1D04 course during weeks 1 and 2.

Table 5.2: Experimental timeline according to ENG 1D04 Weeks

Week #	Item
Week 1	ENG 1D04 begins
Week 2	Survey A
Week 3	Space Race Level 1, Survey B
Week 4	Space Race Level 2, Survey B
Week 5	Space Race Level 3, Survey B
Week 6	Space Race Level 4, Survey B, and Midterm 1
Week 10	Midterm 2
Week 12	ENG 1D04 ends
Week 13	Final Exam

Chapter 6

Student Attitudes and Prior Experience

This chapter will begin with a discussion of the programming experience that students of the experimental and control group have had prior to taking the ENG 1D04 course. The proceeding sections will then explore students' video gaming habits along with their attitudes towards video games in education. All of the aforementioned information was obtained from the responses of Survey A, which can be seen in Appendix D. As mentioned previously in Section 5.2.1, members of the control and experimental group could volunteer to complete Survey A. For this chapter, students belonging to the experimental group are defined to be any student that has played at least one level of Space Race. In total, there were 394 respondents for Survey A; 181 belongs to the control group and the other 213 are part of the experimental group. Not all students chose not to answer all of the questions on the survey.

The last section of this chapter will comment on whether past programming experience, video gaming habits, and attitudes towards educational video games had an influence on whether students chose to participate and play in Space Race.

6.1 Previous Programming Experience

From Survey A, question 1, students were asked if they had any programming experience prior to taking ENG 1D04. The experimental group and control group had nearly an identical proportion of students that answered "yes". From the experimental group, 40% of the respondents stated they had experience with programming. Similarly, 41% of the control group respondents also claimed to have experience with programming. Question 2 and Question 3 were used to get a better sense of what type and how much experience these students had with programming in the past. From Question 2, students were asked how many courses (high school, college, extracurricular courses) they have taken that have involved computer programming (excluding ENG 1D04). The results are seen in Figure 6.1.

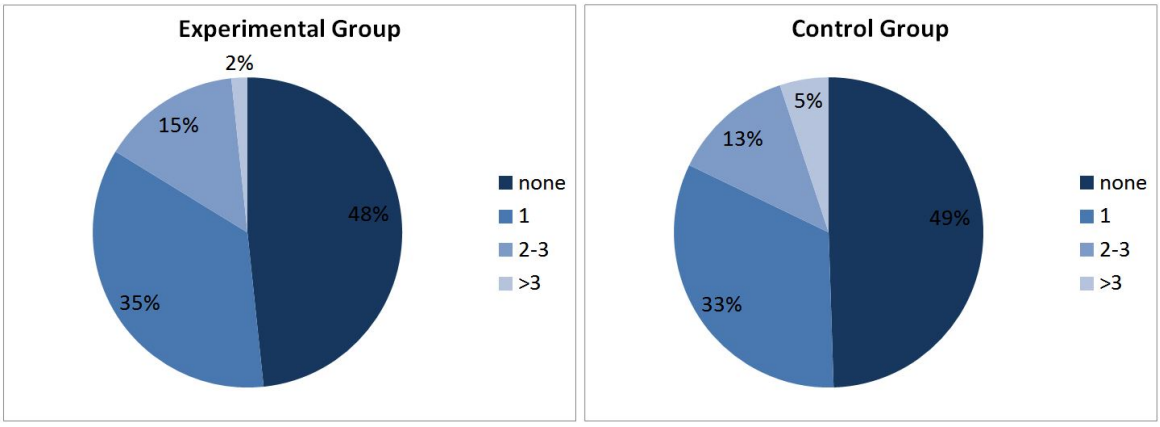


Figure 6.1: Number of Courses Students Have Taken Involving Programming Prior to ENG 1D04

Although 60% of the respondents reported they had no programming experience prior to taking ENG 1D04 in Question 1 for both groups, only approximately 48% stated they had never taken a programming course prior to ENG 1D04. This may be because some respondents chose not to answer one question or the other. Students may also have misinterpreted either one of the questions. Despite this, both groups still have a very similar distribution of responses.

Question 3 was used to further understand the depth of familiarity that students have had with computer programming in the past. Students were asked how many lines of code they wrote in their single largest simple program. From the 213 individuals in the experimental group, only 87 students (41%) have written a program before and the average number of lines for their largest programs is 459 lines. The standard deviation for this group is large: 1355.204 with only 48 students (55%) that have written at least 100 lines in their largest program. Similarly, from the 181 students in the control group, only 59 students (33%) have written a program before and the average number of lines for their largest programs is 613 lines. Once again, the standard deviation for this group was large: 1140.344 with 42 students (71%) that have written at least 100 lines in their largest program. Although the number of lines of code in a program is not an accurate indicator of how proficient a student may be with programming, it can still help to clarify what students interpret to be "programming experience". From the results stated above, it appears that many students' "programming experience" does not involve writing computer programs. Furthermore, one could theorize that a student that has written a maximum of 2 lines of code in their largest program has had less practice with programming compared to another student that has written 2000 lines in their largest program. To summarize, at least 60% of the students from both groups have had no experience in producing computer programs. For the students that have produced programs, 55% of experimental group and 71% of the control group have

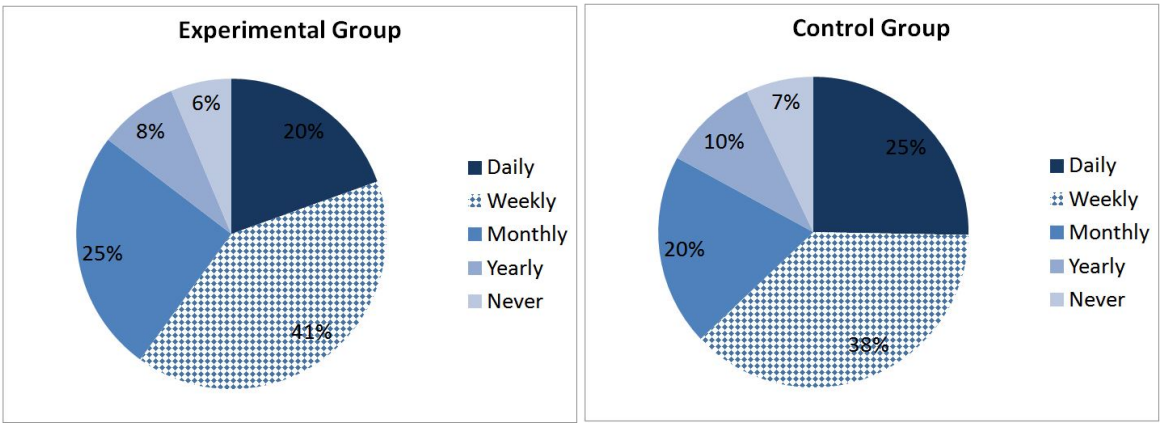


Figure 6.2: Frequency at which Students Play Video Games

written programs that are at least 100 lines long. Although a lower proportion of students in the control group have written programs, on average, their programs are longer than that of their peers in the experimental group.

Finally, when students were asked which programming languages they have used prior to taking the ENG 1D04 course, only a small proportion of each group claimed to have experience with Python, the language used in ENG 1D04. From the control group, 14% of the students have used Python. Java was the top response with 23%. Like the control group, Java was the most frequently used language amongst the experimental group with 23% and only 9.9% of the students having used Python.

From the survey, the researcher is unable to precisely define the level of skill in programming that students have from either the experimental or the control group. However, based on the information above, it appears that there are not any significant differences between the two groups regarding the students' perceptions of their experience with programming. Lastly, the responses give the impression that a large majority of the students would be considered novice programmers.

6.2 Video Gaming Habits

Students were surveyed to gain a better understanding of their video gaming habits. Question 5 from Survey A was used to gauge the frequency with at which students play video games. The results are in Figure 6.2. There does not appear to be a large difference in the distribution of responses from students. Students from the control group only appear to play video games slightly more frequently than the students in the experimental group. In both groups, only 6-7% of the students reported that they never play video games.

Aside from video gaming habits, students also had an opportunity to report on whether

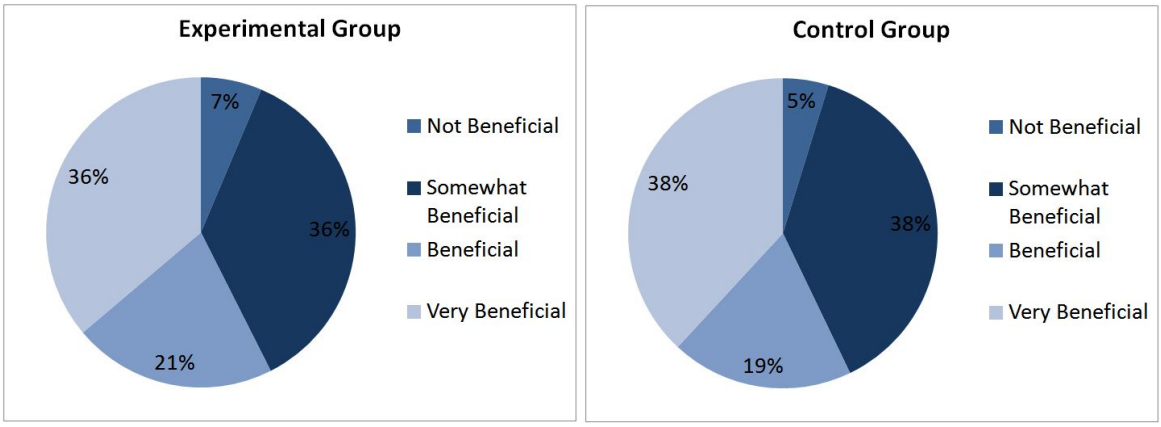


Figure 6.3: Student Opinion on the Incorporation Video Games in Education

they have ever used an educational video game as part of a course in the past (Question 7). From both groups, only a very small proportion of students have played an educational video game as part of a course: 7.1% of the control group and 13% of the experimental group.

Students that have played an educational video game before were asked to comment on their perception of how beneficial the game was to their overall learning experience (Question 9). The question did not explicitly state that the educational video game had to be played as part of a course in the past. 21 individuals (12%) from the control group and 27 students (13%) from the experimental group responded to the question. Figure 6.3 displays the proportion of responses from the respondents. Again, the distribution of responses are nearly identical between the two groups. Furthermore, more than half of the respondents felt that the educational video game they played was at least beneficial if not very beneficial. Lastly, there was an equal percentage of students that found the video game either very beneficial or somewhat beneficial between the two groups.

From the results shown above, it can be seen that the experimental and control group have approximately equivalent video gaming habits. Additionally, only slightly more students from the experimental group, about 5%, have played an educational video game before as part of the course. Finally, for the students that have played an educational video game before, at least half believe that the experience was beneficial to their learning.

6.3 Attitude Towards Educational Video Games

This section will cover students' viewpoints on the inclusion of video games into school curricula as well as their inclination to play video games to learn. Some of the Likert responses of Survey A provide information on this topic. The relevant questions and the

responses students provided are shown in Figure 6.4 and Figure 6.5. The results are shown with diverging stacked bar charts. This format has been chosen because it is easy for the reader to compare whether the respondents generally agree or disagree with the statements in the survey. The percentage of respondents who agree with a given statement are shown to the right of the zero line; the percentage of those who disagree are shown to the left. The percentage of respondents that have chosen "Neither Agree nor Disagree" on the survey are split in half and are shown in a neutral blue colour.

When comparing the graphs visually, it appears there are no significant differences in the distribution of responses between the experimental group and control group. To confirm, a Mann-Whitney U test [43] was performed using SciPy [16], a Python-based open-source software. This non-parametric test requires that the two samples are statistically independent and that the observations are ordinal. This test works well for this scenario since the experimental and control group are independent from each other. Additionally, Likert responses are ordinal; the response scale is arbitrary and the perceived distance between each item is subjective but there is a defined ranking from the response options. The null hypothesis for this test states that the two samples are derived from the same population; that is, there are no significant differences in the distribution of responses between one group and the other. When the null hypothesis is rejected then it means that the sample distributions differ in center, spread, and/or shape. When the forms of the distributions are similar, then the rejection of the null hypothesis is taken to mean that one sample tends to have a larger median than the other. The Mann-Whitney U test provides two values: the U-statistic and the p-value. The smaller the U-statistic, the less likely that the differences have occurred by chance alone. The p-value, if it is lower than 0.05, can indicate that the result is significant at a level of 0.05; the null hypothesis can be rejected. The results of the Mann-Whitney U test can be seen in Table 6.1. Questions 12, 21, and 34 have significantly different results. There is very weak evidence that the results of the other questions are significantly different; they are very similar.

When students were asked if "*[they] would enjoy playing video games to learn*" in Question 12, 67% of the experimental group and 64% of the control group agreed (either "Agree" or "Strongly Agree" chosen) that they would. Alternatively 5.4% of the experimental group and 8.3% of the control group disagreed with a remaining 27% and 28% choosing to neither agree nor disagree, respectively. This shows that approximately the same proportion of students either agree or disagree that they would enjoy playing video games to learn. However, a greater percentage of the experimental students (23% versus 13%) chose to strongly agree with the statement which most likely accounts for the significant difference in results in the Mann-Whitney U test for Question 12. From this, one can conclude that more students from the experimental group would derive more joy from playing a video game to learn than the control group, although almost the same proportion of students would feel at least some enjoyment from playing video games.

The response from Questions 20 to 23 can be used to gain an understanding on how

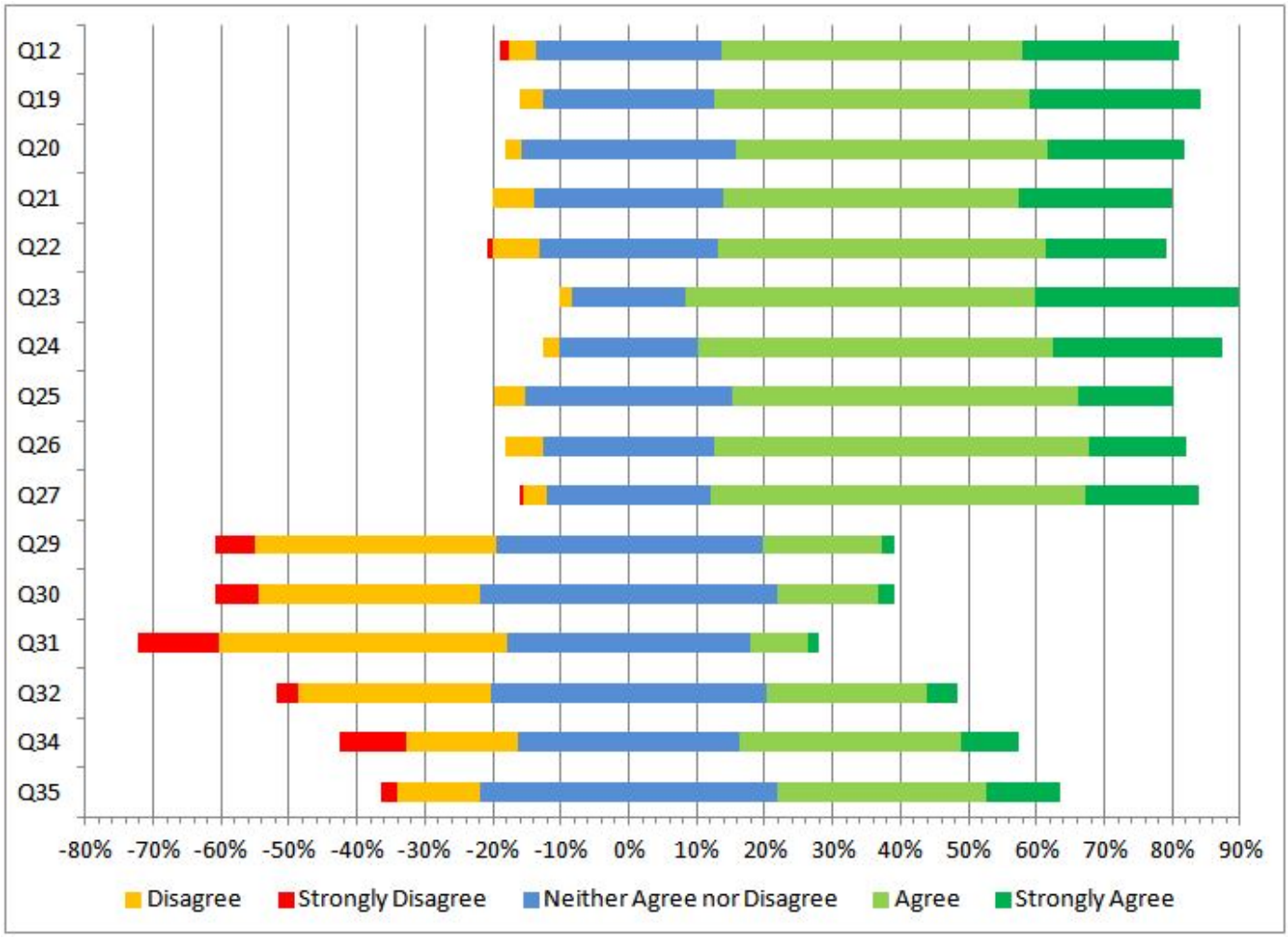


Figure 6.4: Experimental Group Survey A Likert Response Results

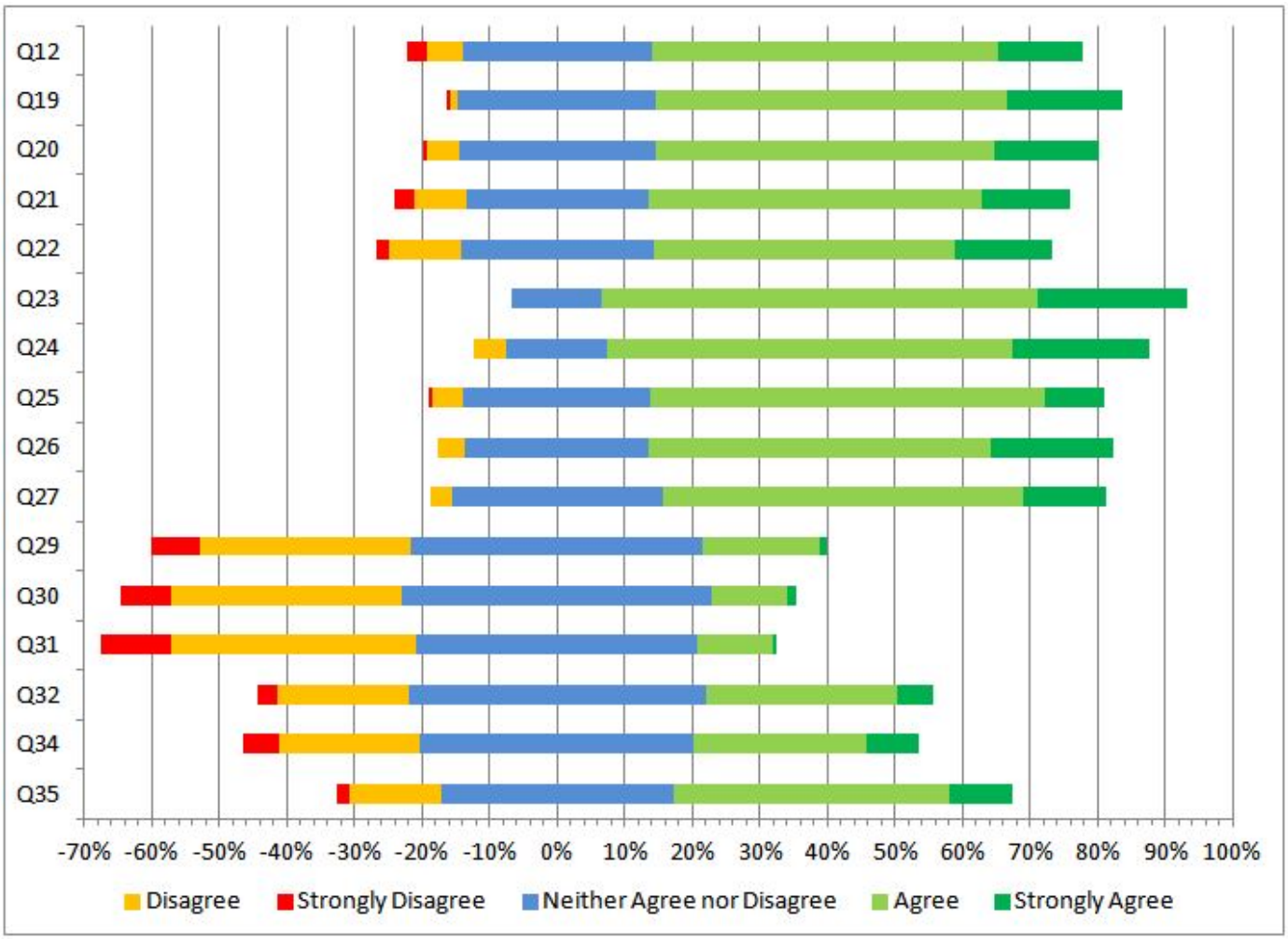


Figure 6.5: Control Group Survey A Likert Response Results

Question	U-statistic	p-value	Significant at p<0.05?
Q12	15390	0.029	Yes
Q19	16198	0.099	No
Q20	16599.5	0.227	No
Q21	15679.5	0.031	Yes
Q22	15495	0.053	No
Q23	16674	0.322	No
Q24	16592.5	0.349	No
Q25	17073	0.355	No
Q26	16573.5	0.319	No
Q27	15755	0.079	No
Q29	17126	0.455	No
Q30	16203.5	0.158	No
Q31	16101.5	0.099	No
Q32	15518.5	0.038	Yes
Q34	16539.5	0.222	No
Q35	16327.5	0.156	No

Table 6.1: Mann-Whitney U test Results Between Experimental and Control Group

students feel a video game would impact their curiosity on a subject and motivation to learn. Out of these three questions, only Question 21 had a significant difference in results between the experimental and control group. Like Question 12, there is approximately the same proportion of students that agree and disagree that "*video games can motivate students to learn*" between the two groups. However, the experimental group had 10% more students choosing to strongly agree with the statement versus the control group. From this, one can conclude that more students from the experimental group feel more strongly that video games can motivate students to learn. When viewing the remaining questions, it is clear that a large majority of students that chose to either agree or disagree with the statements have chosen to agree. This indicates that, across both groups, most students feel that video games can do the following:

- "*Video games can teach students how to set and reach goals*"
- "*Video games can motivate students to seek out additional information that will help them succeed*"
- "*Video games can stimulate a student's curiosity on a topic*"

The first three statements above are all very important for learning according to the fundamental ideas of the constructivist theory [61]. From Section 2.1, it was shown that,

according to constructivist principles, puzzlement is the stimulus for learning and determines the organization and nature of what is learned. If the first three statements above are true, then video games can greatly influence and enhance the learning experience. If students agree with the above statements, then it suggests that they are open to the idea that video games can affect their learning experience.

Questions 24 and 26 implicitly ask questions on whether video games can be an effective part of experiential learning. Experiential learning was discussed in Section 2.1 and it was revealed that student exploration with tangible experiences and personal reflection on the feedback received from those experiences is part of Kolb's experiential learning model. At least 77% of the students from the experimental and control group agreed that "*video games can provide a safe way to test new ideas or try new techniques*". Furthermore, a minimum of 69% of the students from both groups agreed that "*video games can provide immediate feedback to students*". It is important to note that, at most, 4.8% of the students chose to disagree with either one of those statements in both groups; the rest chose to neither agree nor disagree. This information indicates that students feel video games can have a role to play in experiential learning.

Questions 25, 31, and 32 comment on student attitudes towards the incorporation of video games in education. Most students from both groups (at least 67.1%) agree that "*video games can be used to complement course objectives*" (Question 25); only, at most, 4.8% disagree with the statement. Also, across both groups, at least 35% more students disagree that "*the use of educational video games does not belong in the post-secondary classroom*"(Question 31) as opposed to agree. There were significant differences in the distribution of responses for the statement given in Question 32: "*In general, students will not take educational video games seriously*". For both groups, at least 40% of the students chose to neither agree nor disagree with the statement. The experimental group had more students disagree with the statement: 3.5% more disagreed. Conversely, the control group had more students agree with the statement: 11% more agreed. This indicates that there is a greater proportion of students from the control group that feel that students, in general, will not take educational video games seriously.

Finally, students were asked to comment on how they perceived gender could influence the enjoyment and benefits that an educational video game could provide through Questions 34 and 35. Perhaps surprisingly, both groups had a slightly greater proportion of students that agreed that "*in general, males will enjoy using educational video games in classes more than females will*" (Question 34); 7.2% more in the control group and 15% more in the experimental group. However, in both groups more students agreed that "*in general, females will benefit from using educational video games as much as males*" than disagreed (Question 35): 27% more agreed in the experimental group compared to 35% more that agreed in the control group.

Chapter 7

Game Reception and Feedback

This chapter will provide details on the results of Survey B (seen in Appendix D), which allowed students to provide feedback on each level of gameplay in Space Race. These results will be used throughout the chapter to support the qualitative observations made by the researcher while watching student participants play Space Race. Based on these observations, the playability and teachability of Space Race will be discussed. Furthermore, the cooperative and competitive aspects of Space Race and their contributions to the effectiveness of Space Race as a teaching tool will be explored.

7.1 Survey B Results

A detailed description for the design of Survey B can be found in Section 5.2.1. The Likert response results of Survey B (questions 1 through 24) from Level 1, Level 2, Level 3, and Level 4 can be seen in Figures 7.1, 7.2, 7.3,and 7.4, respectively. The responses are shown with diverging stacked bar charts like the Likert responses from Survey A in Chapter 6. The number of respondents for Survey B in each Level are shown in Table 7.1.

Table 7.1: Number of Survey B Respondents

Level	Number of Respondents
1	180
2	180
3	92
4	98

A quick look through the results show that respondents generally hold the same opinion towards a statement on the survey. That is, a great majority of respondents respond in the

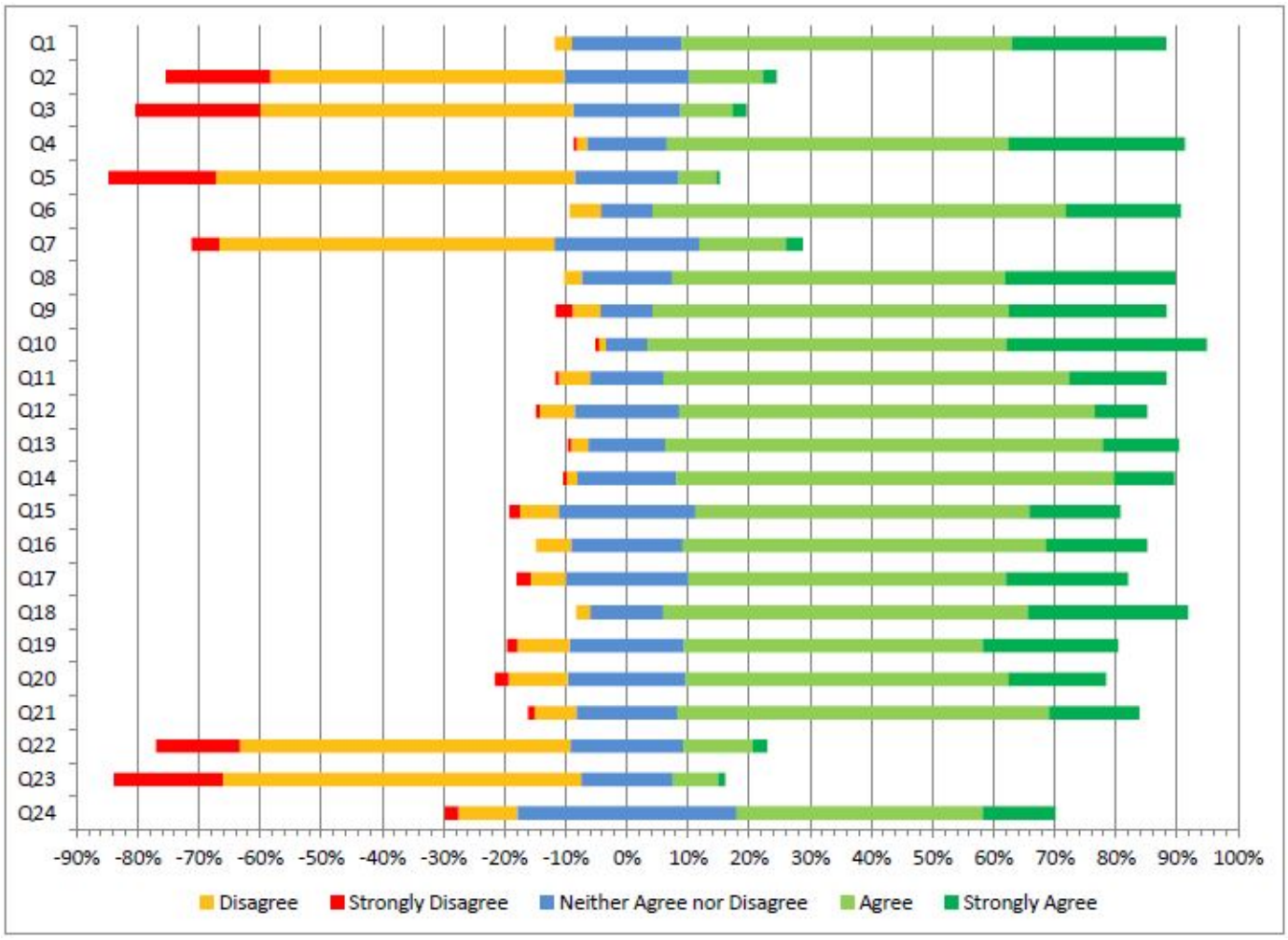


Figure 7.1: Level 1 Survey B Likert Response Results

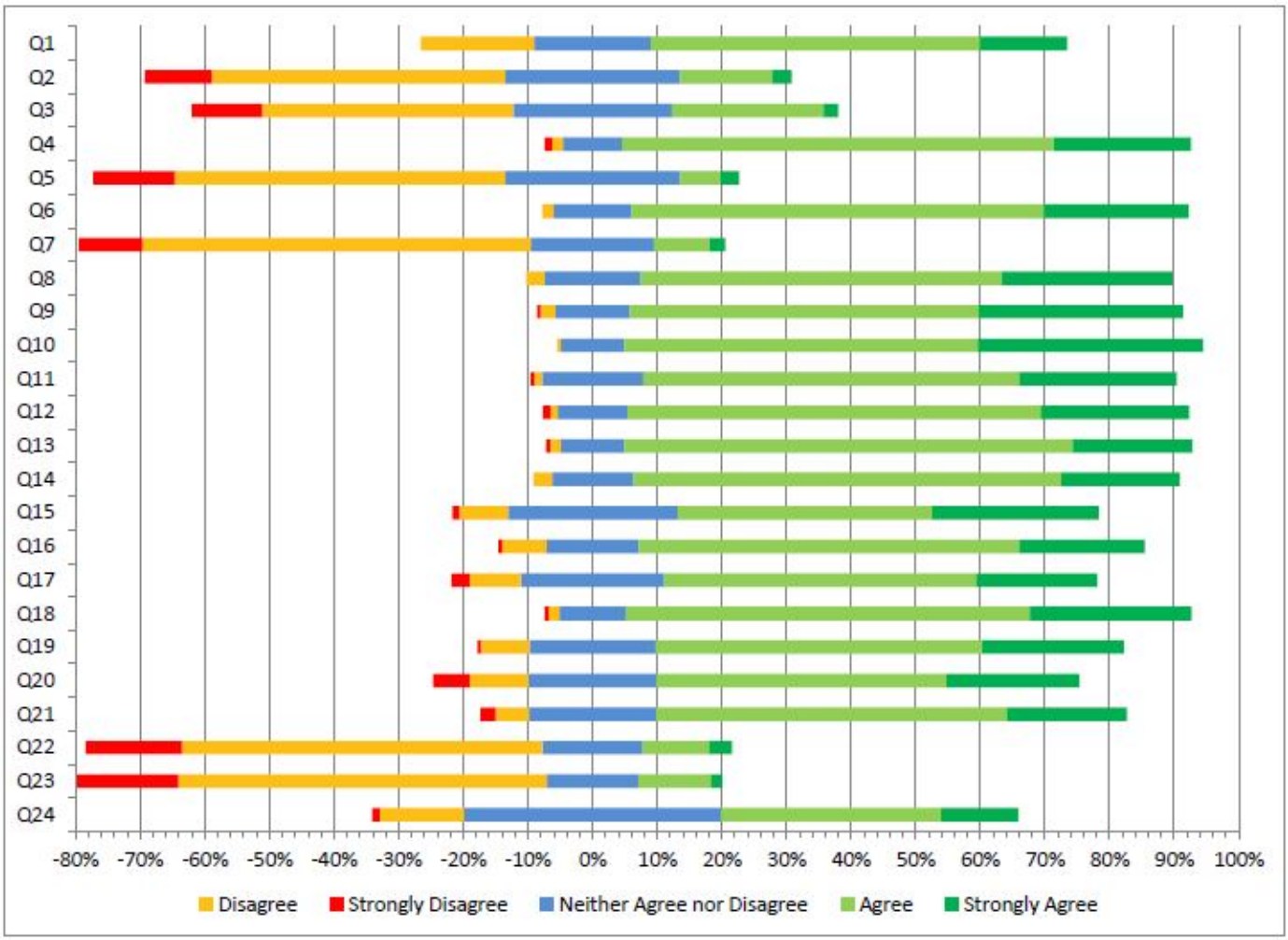


Figure 7.2: Level 2 Survey B Likert Response Results

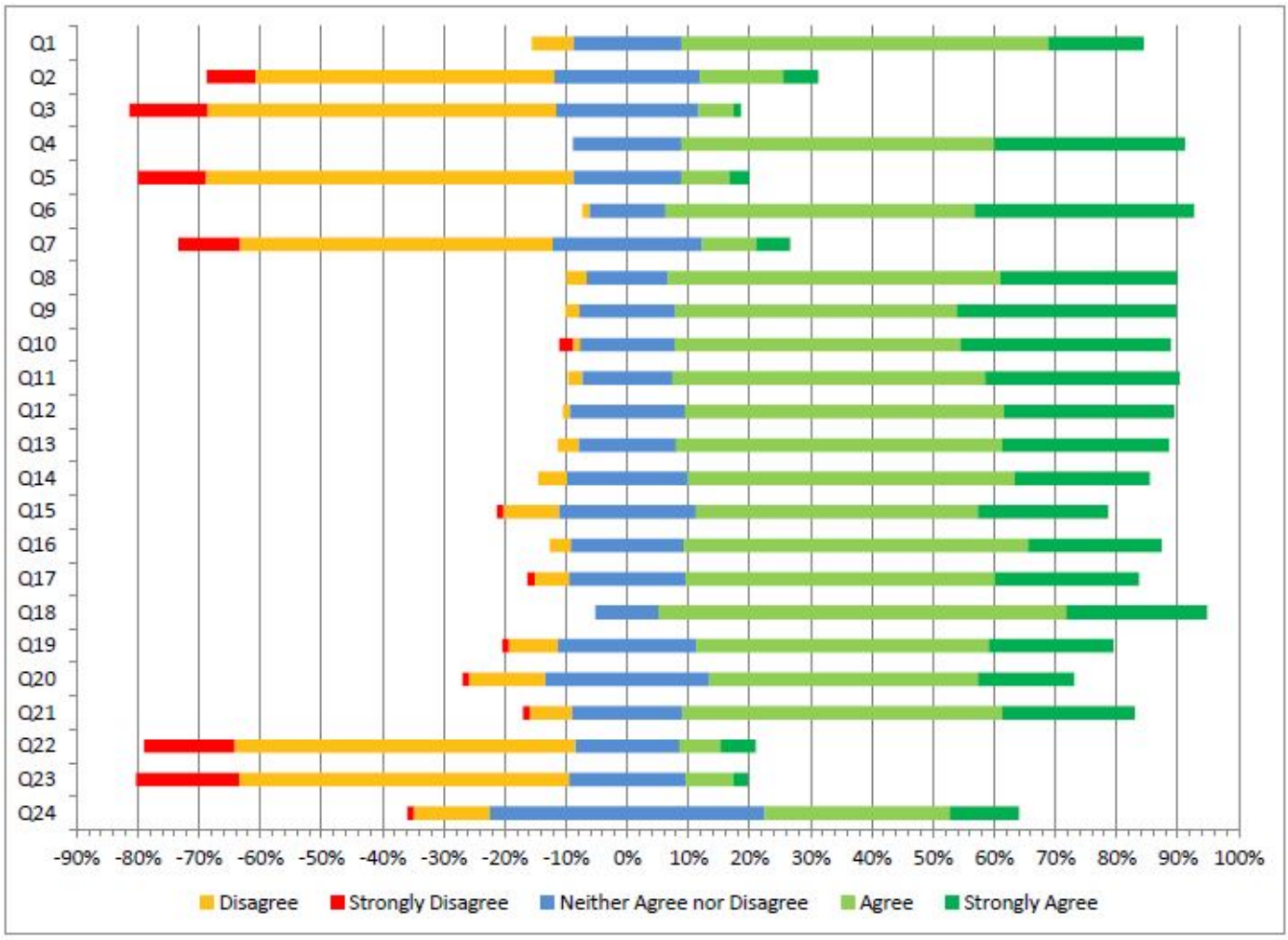


Figure 7.3: Level 3 Survey B Likert Response Results

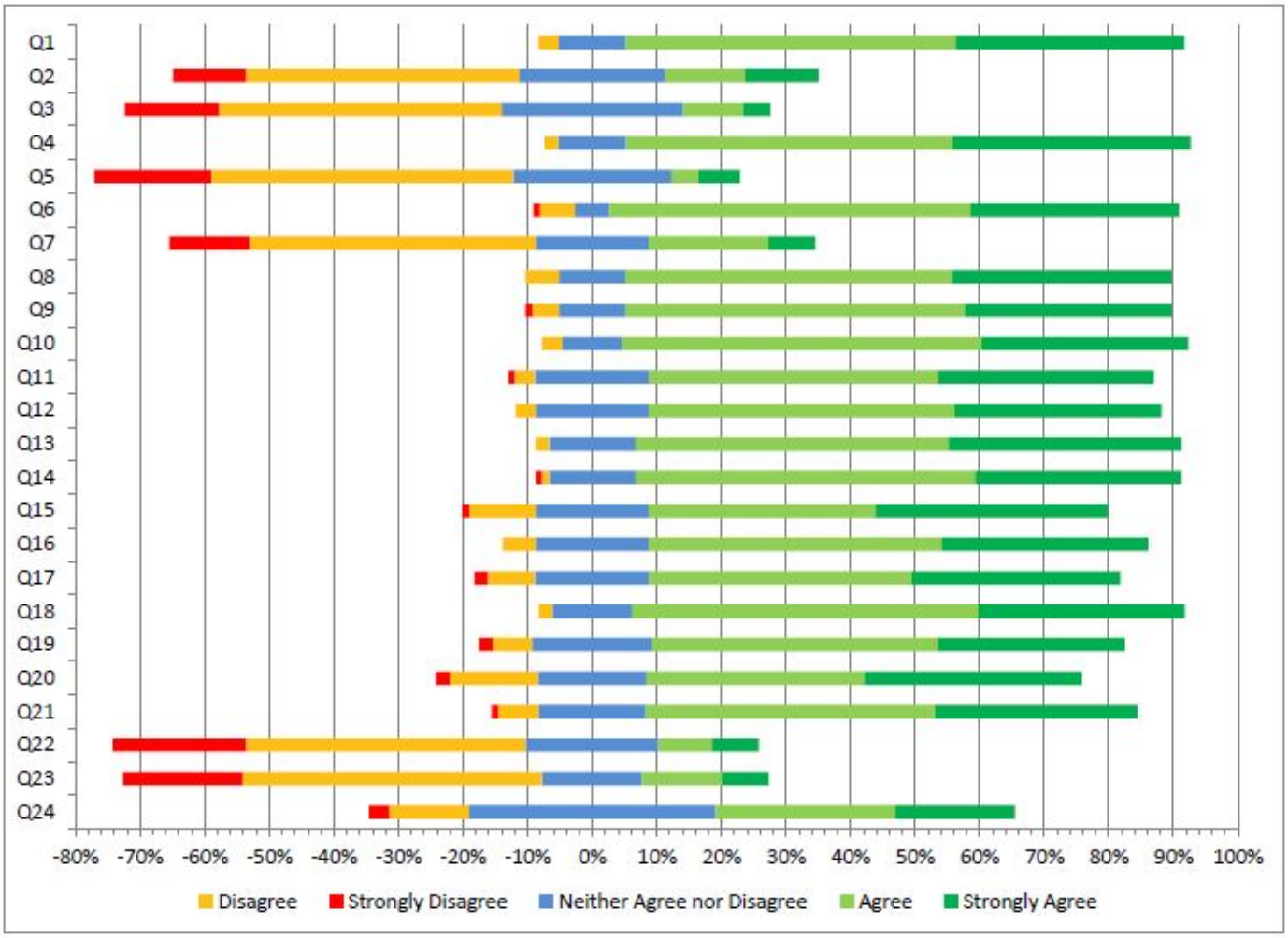


Figure 7.4: Level 4 Survey B Likert Response Results

Table 7.2: Survey B Level 1,2,3,4 Kruskal-Wallis H-test

Question Number	H-statistic	p-value	Significant at p<0.05?
Q1	32.861	0	Yes
Q2	7.752	0.051	No
Q3	22.865	0	Yes
Q4	4.083	0.253	No
Q5	6.419	0.093	No
Q6	7.894	0.048	Yes
Q7	7.139	0.068	No
Q8	1.389	0.708	No
Q9	2.171	0.538	No
Q10	1.117	0.773	No
Q11	5.624	0.131	No
Q12	19.038	0	Yes
Q13	11.028	0.012	Yes
Q14	11.175	0.011	Yes
Q15	4.62	0.202	No
Q16	4.062	0.255	No
Q17	4.788	0.188	No
Q18	0.717	0.869	No
Q19	1.616	0.656	No
Q20	4.214	0.239	No
Q21	4.224	0.238	No
Q22	0.264	0.967	No
Q23	2.848	0.416	No
Q24	1.802	0.614	No

same manner for any given statement; either agreeing or disagreeing.

To aid in simplifying the analysis of the results, each question was evaluated to determine if the results across the four levels of the game can be combined. A Kruskal-Wallis H-Test [33] was performed, using SciPy [16], to compare and evaluate if there were any significant differences between the four groups of responses. This test is chosen because Likert scale responses are ordinal and there are four categorical groups: Level 1, Level 2, Level 3, and Level 4 respondents. The Kruskal-Wallis H-test tests the null hypothesis that the response distributions are equal. The p-value indicates how likely any differences observed would have occurred if responses were drawn from the same population. That is, a small p-value would indicate that it would be rare to observe differences, if there was no relationship between the group and the value being tested. The results of the Kruskal-

Wallis H test performed on Questions 1 through 24 on Survey B across the four categorical groups can be seen in Table 7.2. If the p-value for a given responses is not less than 0.05, one cannot say that there is a significant difference in attitudes on the survey for the Likert responses between the four categorical groups. However, if there is a significant result then one can make the conclusion that at least one of the groups is different form the other groups. The test does not identify where the differences occur or how many differences actually occur [33].

The results of Table 7.2 indicate that there are significantly different responses from at least one group for Questions 1, 3, 6, 12, 13, and 14. Since the game mechanics of Level 2, 3, and 4 are very similar while the game mechanics for Level 1 are different, the Kruskal-Wallis H-Test was performed again, but this time only on the responses for Levels 2, 3, and 4. The results can be seen in Table 7.3 and they indicate that for Levels 2, 3, and 4, only Questions 1 and 3 have responses that are significantly different. Going forward, all of the questions that had insignificant differences between the four levels will be analyzed using the results for all of the data pooled together. Alternatively, all questions that yielded a significantly different result between the four levels will be analyzed further to identify what may have caused these differences in responses.

The following subsections will review the Likert responses according to the categories that were specified in Section 5.2.1: playability, teachability, team cooperation, and competition. The discussion of results from the Likert responses of Survey B will be supplemented by some additional feedback that was provided by respondents through the short answer questions of Survey B.

7.1.1 Playability

Questions 1-5 and 24 on the Likert response portion of Survey B relate to playability. Playability questions allow students to rate Space Race according to their user experience. It also allows the student to comment on how much they enjoyed playing Space Race.

The statement from Question 1, "*The video game was easy to play*", was met with a large majority of agreement (either "Agree" or "Strongly Agree"); over 60%, for all four levels. However, there was a gradual increase in the proportion of students that agreed with that statement across the last three levels: 64% for Level 2, 76% for Level 3, and 86% for Level 4. This may account for the significant difference in results seen from the Kruskal Wallis H-test in Table 7.3. The gradual increase in the percentage of students that agreed with the statement suggest that students found the game easier to play as they continued from Level 2 to Level 4. This may be because of the similar game mechanics across those levels. From the survey, a student commented that they "*appreciated the fact that the format of level 3 was similar to that of level 2. [They] were able to get straight into the game!*". Alternatively, 79% of the students in Level 1 agreed that the game was easy to play. This suggests that students found the game mechanics of Level 1 easier to manage than in the

Table 7.3: Survey B Level 2,3,4 Kruskal-Wallis H-test

Question Number	H-statistic	p-value	Significant at p<0.05?
Q1	27.41	0	Yes
Q2	0.549	0.76	No
Q3	10.798	0.005	Yes
Q4	4.165	0.125	No
Q5	0.458	0.795	No
Q6	4.001	0.135	No
Q7	4.314	0.116	No
Q8	1.311	0.519	No
Q9	0.088	0.957	No
Q10	0.929	0.628	No
Q11	0.887	0.642	No
Q12	0.152	0.927	No
Q13	4.012	0.135	No
Q14	4.67	0.097	No
Q15	2.696	0.26	No
Q16	2.119	0.347	No
Q17	4.586	0.101	No
Q18	0.625	0.732	No
Q19	1.487	0.475	No
Q20	3.863	0.145	No
Q21	3.081	0.214	No
Q22	0.073	0.964	No
Q23	0.935	0.626	No
Q24	0.459	0.795	No

other levels. This result makes sense given the simple game mechanics for Level 1.

Perhaps one of the main reasons why students found Level 1 easier to play than Level 2 and 3 is the potential for chaos to be introduced in Level 2 and 3. For Levels 2 and 3, the game was designed to simultaneously provide each player with a different instruction. The intention behind this design decision was to ensure that all students would be simultaneously involved in the game. Although the designer was aware that some chaos may be introduced, it was believed that this minimal disorder would increase players' immersion into the game. However, this chaos may have also introduced more distraction than desired. A student suggested that perhaps "*maybe only 1 or 2 members has a question on [their] screen or else it gets kind of hectic*". Level 4 had the greatest number of respondents that felt the video game was easy to play. This may have been because there is less

chaos introduced in this level than in Levels 2 and 3. This was achieved by having students alternate between an instruction for their own panels and a `for`-loop that all team members worked to execute together (more details given in Chapter 4). Many students commented that they liked this format because "*the `for`-loops included all of the group members*". Another student stated that "*[they liked] the idea of working together in an orderly manner to execute the loops. It was a good way to make sure [they] were all following the code being executed*". To summarize, it appears that Space Race could increase ease of play, by introducing more order to gameplay.

The statement from Question 3, "*It was difficult to learn how to play the video game*", provided insight to the learning curve students faced to play the video game. According to the Kruskal Wallis H-test results of Table 7.2 and Table 7.3, Question 3 had significantly different results for the four levels. Level 1 had the greatest percentage of students that were in disagreement (either "Disagree" or "Strongly Disagree") with that statement: 72%. On the other hand, Level 2 had the lowest percentage of students that disagreed with that statement: 50%. Level 3 had a larger proportion of students that disagreed: 70%. Finally, Level 4 saw a decrease again in the proportion of students that disagreed with the statement: 58%. The game mechanics of Level 2 are more involved than Level 1 so it is not surprising that more people found it more difficult to learn how to play Level 2. The increase in the number of students that did not find it difficult to learn how to play the game in Level 3 can be explained again by the fact that Level 3 has nearly identical game mechanics to Level 2. Lastly, Level 4 may have seen a decrease in the portion of students that did not find it difficult to learn how to play the game, in comparison to Level 3, because `for`-loops are introduced in this level. However, it is important to note that students felt Level 4 was still easier to learn how to play than Level 2. This would be expected since the underlying game mechanics of Level 4 do not differ much from Level 2 and 3 despite the introduction of `for`-loops.

A large majority of the feedback on the survey mentioned the need for a more interactive tutorial to teach students how to play the game. A student said that they "*like the game however the instructions were not very intuitive and it was hard to initially learn how to play. After learning how to play it was more fun.*". An example of the current tutorial page, which is shown before each level of gameplay, can be seen in Figure 7.5. The page features the game screen that students will see for each level along with explanations for each game control that students can read after clicking on the appropriate button. The researcher observed that the students rarely, if at all, read the instructions before playing the game. A wide majority of the students preferred to skip the instructions and proceed directly to gameplay. This may contribute to the perceived difficulty in learning how to play the game. It also suggests that the traditional method of defining game rules through text is no longer as effective and perhaps, as a student suggested, "*...a video or example would be much more effective*". While a video tutorial could be more interesting for students to review, it is not clear whether a tutorial presented in a different manner would have

any effect on each player's ability to learn how to play the game. Research by Andersen et al. has shown that the value of tutorials is highly dependent upon the complexity of the game; they have a surprisingly negligible effect on player engagement in games that are less complex and more similar to existing games in the same genre [2]. Furthermore, providing help on-demand, during gameplay, can either have positive, negative, or negligible effects on player engagement. Their results concluded that perhaps tutorials are not necessary for games that can easily be learned through experimentation. For these types of games, players seem to learn more through experimentation than from reading text [2]. Given the relatively simplistic game mechanics of Space Race, perhaps a complete tutorial is not necessary; it could either be omitted entirely or help on-demand can be provided during gameplay. However, given the unpredictable effects of help on-demand (positive, negative, or negligible effects), this tutorial implementation should be tested to avoid unforeseen negative effects.

While playing the game, students experienced some minor technical difficulties. The greatest issue was that the tablets would sometimes struggle to maintain connection with the server because of the weak wireless signal. In the feedback section of the survey some students suggested the researcher "*provide a much more reliable internet connection*". If one of the players were disconnected during gameplay, the game would bring all of the other team members back to a waiting room where they could resume the game once the disconnected player reconnected to the server. This was a minor inconvenience; it did not hinder gameplay too much because the game state would be saved. It appears to have had some effects on the student's perception of the benefits of the game as the survey indicates that at most 24% agreed with the statement that "*the video game required too much technical support to be beneficial*".

Overall, it appears that a large majority of the students enjoyed playing the game with at least 82% students agreeing with the statement "*I enjoyed playing the video game*" on the survey across all the levels. Level 2 had the highest proportion of students that were in agreement with that statement: 88%. A student commented that they "*enjoyed the whole game, [because] it helped [them] learn some basic programming techniques and cleared up some uncertainty.*". A student suggested that there should be a "*storyline element [added to the game] as it makes it much more immersive and easier to remember when told as a narrative*". Another student even offered a storyline where "*four astronauts have crash landed on a planet and need to recode their computer in order to take off again*" suggesting that this storyline would "*draw the players in to the game*".

When students were asked if they "*[wished] there was more gameplay time in the video game*" (Question 24), Level 1 respondents had the highest proportion of students that agreed with the statement at 52% while Level 3 had the lowest with 42%. This was also the question where the greatest number of students selected "Neither Agree nor Disagree" as their response. The game was designed so that each level could be completed within an hour because of the time constraints students faced with their busy schedules.

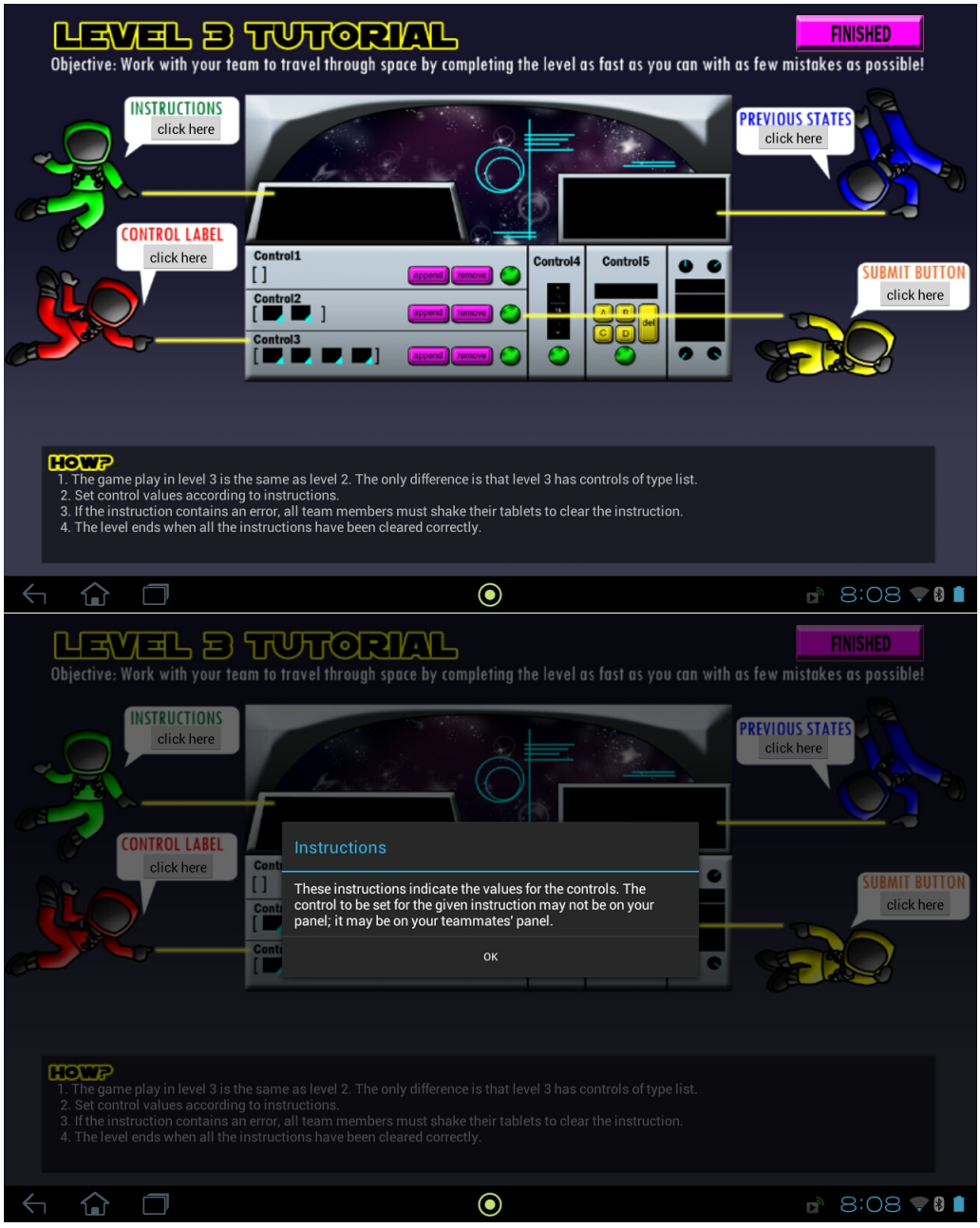


Figure 7.5: Space Race Level 3 Tutorial Page

An average of the best times for each level can be seen in Table 7.4. As expected, Level 3 had the longest time while Level 1 had the lowest time which may explain the results of Question 24.

Table 7.4: Averages of the Best Times for each Level of Space Race

Level	Average Time
1	0:07:58
2	0:19:07
3	0:32:14
4	0:23:49

From Table 7.4, it can be seen that Level 1 has an extremely low average best time. In reality, most teams took approximately 30 to 45 minutes to complete the first level the first time through. All of the groups completed the first level as intended the first time through the game. However, many groups discovered that they could achieve a much faster time if they "cheated" the second time through. These groups would continuously press random buttons and shake their tablets to advance in the game. This is a problem with Level 1 since there are no consequences to making mistakes. This was noted by a few students in the survey as some students suggested that the the game should "*incorporate the effect of wrong answers into the final score*". Alternatively other students said that "*[they] liked that [they] don't lose points when [they] get the questions wrong*". While including the number of incorrect answers to the score in the first level could have prevented cheating, this method would not be necessary for the other levels since it is much more difficult, and nearly impossible, to cheat. Furthermore, it is not clear if a score is necessary to encourage students to answer questions accurately. For example, a student said that "*the running clock was a very interesting part of the game, it encouraged [them] to think fast and try [their] hardest to answer the questions accurately and quickly*". This suggests that the clock was sufficient to prompt the student to try and answer the questions as accurately as possible without guessing. Additionally, the researcher observed that all of the groups were more interested in learning the material correctly rather than guessing and pressing random numbers for Levels 2, 3, and 4, and the first iteration of Level 1.

7.1.2 Teachability

This section explores students' perceptions of how effective Space Race was at teaching basic programming concepts. It will also comment on whether Space Race was able to motivate students to study the course material and whether students prefer the game over traditional teaching methods. Questions 6-8, 11, 12, 15-17, 22, and 23 on Survey B are the

Likert responses that relate to the teachability of the game.

When students were asked if "*the video game helped [them] to learn the basic programming concepts being presented*" (Question 6), at least 85% students answered with agreement in all four levels. Level 4 had both the most students that agreed with the statement (88%) and disagreed with the statement (6.5%). The Kruskal Wallis H-Test results shown in Table 7.2 reveal that there was a significant difference in the results between the four groups. However, this can most likely be attributed to the difference in the proportion of students that chose "Strongly Agree" as opposed to "Agree" since all four levels had approximately the same proportion of students that chose "Neither Agree nor Disagree" and "Disagree"/"Strongly Disagree"; all were comparatively low. Notably, Level 3 and Level 4 had almost double the proportion of students that selected "Strongly Agree" versus Level 1.

For Question 8, at least 82% of the students felt that they "*would recommend that others try to learn the basic programming concepts with [Space Race]*" for all four levels. However, from the survey, a student has noted that "*the game confirms knowledge [; it] doesn't really teach*". Some other students expressed the same feelings and felt that Space Race needed "*a better response system to incorrect answers*". For example, a student suggested that "*hints [appear] if a team continuously got a certain question wrong*". To summarize, students felt that the game needed more feedback response to teach students concepts instead of simply confirming correct knowledge. Even though there was a lack of feedback from the game, which prevented the game from directly teaching students concepts, it certainly created multiple opportunities for team members to teach each other new concepts. This will be discussed in more detail in the following section (Section 7.1.3). Furthermore, the game pushes students to reference their "*Cheat Sheets*" (Appendix B) in Levels 2, 3, and 4 for assistance and feedback. Most students felt that "*the cheat sheet was very helpful in summarizing the game*".

From the researcher's observations, although the game does not directly teach students new concepts, it definitely creates many "teachable moments". An instructor was present while the students were playing the game and when a team failed to execute an instruction correctly after much discussion, they would raise their hands to ask for additional assistance. This allowed the instructor to gain a better understanding of what the students did not understand. Moreover, it helped the students identify concepts that they were not familiar with. Finally, although the game cannot teach students directly, it is certainly an excellent pedagogical tool because it encourages students to ask the instructor questions when they do not understand. A student is perhaps more likely to receive help from an instructor when they are working with team members that are also unable to solve the problem at hand. This could be because a student that is less assertive and may normally not ask for help from a teacher may have a team member that is more assertive and not as shy to ask questions. Also, there is less shame associated with not understanding a concept if their peers also do not understand the concept. This, in turn, could lower a student's anxiety towards requesting their instructor, an authoritative figure, for help. Finally, asking

an instructor for help in the context of a game could perhaps be lower risk than asking a question in another context; for example, in lectures, tutorials, or labs.

Question 15 was used to assess students' preference for an educational video game over traditional teaching methods. At least 65% of the students agreed with the following statement: "*I would prefer to learn basic programming concepts through a video game as opposed to other traditional teaching methods such as lecture tutorials, or textbooks*". A student also said that '*when [they] played this game [they] felt as if [they] were studying. [They] would enjoy using this game as a break from the regular studying for tests and exams.*". Another student has stated that they "*learned a lot from this game as [they] were able to apply what [they learned] in lectures and tutorials*". Finally, a student has said that "*the interactions between team members bolstered the understanding of concepts and created a better learning experience than just reading a textbook.*".

The results of Question 16 and 17 can be used to gauge how effective Space Race was in motivating students to be more interested in the subject of programming. When students were asked if "*the video game stimulated [their] curiosity on the basic programming concept being presented*" (Question 16), at least 74% agreed and at most 7.4% disagreed with the statement for all the levels . Furthermore, at least 67% agreed that "the video game has motivated [them] to review the course material" across all four levels. This is important because it suggests that the game has the side effect of motivating students to seek an understanding of programming outside the context of the game. Furthermore, curiosity in a subject is a key component to learning that material from the constructivist perspective as mentioned previously in Section 2.1.

Finally, the results of Questions 22 and 23 of the survey can indicate whether the level of difficulty of the game was a hindrance to learning the programming concepts being taught. It appears that a large majority of the students felt that the game was neither too challenging nor too easy to teach the basic concepts. This is supported by the results that at most 15% felt that the "*the video game was not effective in teaching basic programming concepts because it was not challenging enough*" (Question 22) while at most 20% felt that "*the video game was not effective in teaching basic programming concepts because it was too challenging*" for all of the levels. A student also commented that they "*[enjoyed] the level of difficulty and the relevance of the questions with regards to the course material*".

7.1.3 Cooperation

Perhaps the most important aspect of Space Race is its ability to teach was the incorporation of cooperation in the game design. Questions 9, 10, 13, 14, 18, and 21 provide insight on how cooperation was able to affect how students learned while playing the game.

From Question 9, across all levels, as little as 2.3% and at most 7.4% disagreed with the statement that "*[their] team members helped [them] to better understand the basic programming concepts being presented*" for all four levels. Furthermore, only 0 to 2.3%

disagreed with the statement that "*cooperation between team members makes this video game an effective teaching tool for learning basic programming concepts*" (Question 18) for all the levels. From this, one can conclude that cooperation played a large part in the effectiveness of Space Race as a teaching tool. Students have also provided the following feedback with regards to cooperation in Space Race:

- "*I really enjoyed the cooperative aspects of the game because it does a tremendous job of encouraging teamwork and sharing individual knowledge for the benefit of a common goal...*".
- "*When we were stuck on a question, the team would offer suggestions and we would get through the question together. This was helpful when questions were hard and especially when the player did not know the correct answer on their own.* "
- "*[My team members] helped me a lot , as sometimes my initial guess was incorrect, but they [would] correct me and explain the concept.*"
- "*My team members helped me to understand the various tools being presented and helped to clarify things I did not quite understand.*"
- "*Communicating with team members encourages discussion of concepts which I like*"
- "*I really liked the cooperative nature of this video game. Individually, we may have struggle with the concepts; but as a team, we conquered them.*"

The quotes given above certainly highlights how cooperation influenced gameplay in Space Race and facilitated learning. These findings are consistent with the theories discussed in Chapter 2. For example, students did, in fact, find that they could share and clarify ideas and learn from each other when they grouped together to play Space Race. Furthermore, it was mentioned previously that, according to Vygotsky's contributions to the field of social constructivism, problem solving skills of specific tasks can be evaluated based on a student's ability to perform tasks with others [67]. This ability is cultivated when students learn in a collaborative environment and is transferable from the classroom to "real life". This notion was echoed by one of the students in their feedback: "*I liked how team member cooperation results in better final scores. [This represented] real life situations [since] working in cooperative teams translated to better success.*".

In Chapter 2, it was also mentioned that cooperation could encourage students to work harder since feelings of guilt are registered when the individual does not feel that they have contributed sufficiently. Question 21 on the survey targets this concept. At least 73% agreed with the statement that "*[their] role as an effective team member motivates [them] to review course material*" while at most 8.0% disagreed with the statement for all four levels. A student has said that "*[their] team member really motivated [them] to actually*

work hard, and not mess up, since it's a team effort. It encouraged [them] to review [their] knowledge about Python.".

Each group had individuals with different programming skill levels and degree of understanding of course material. From the feedback that has been mentioned so far, it is clear that the less informed students felt they were able to learn from their more capable peers. It would be interesting to investigate whether these more knowledgeable students were still able to feel that Space Race contributed to their understanding. Questions 13 and 14 of the survey can provide some insight as to whether the provision and reception of verbal cues was able to reassure students of their understanding of concepts. The results of the Kruskal Wallis H-test shown in Table 7.2 reveal that there were significant differences between Levels 1, 2, 3, and 4 for these two questions. This can, again, be attributed to the proportion of students that selected "Strongly Agree" versus "Agree". From all four levels, at least 81% of the students agreed that "*[they] felt reassured of [their] knowledge in the programming concepts presented when [they] gave verbal cues to [their] team members in the game.*" (Question 13) while at least 76% of the students felt "*[they] felt reassured of [their] knowledge in the programming concepts presented when [they] received verbal cues from [their] team members in the game.*" (Question 14). From the comments received on the survey, it seems like most of the students that felt more capable than their team members still enjoyed the game and felt a sense of pride in being able to instruct others. For example, a student stated that they "*[were] the lead of the team so [they] did not learn anything from [their team members] however, [they] taught them a lesson or 2 [sic] about Python.*". It is important to note that there were very few comments on the survey where students felt their team members did not teach them anything new. Some of the "negative" comments could even have a positive interpretation. For example, a student said that "[their teammates] slowed [them] down but [it] was fun to play with them". Another said "[their teammates were] great but they need to be more efficient". It was much more common for students to comment that their team members were able to teach them a concept they did not understand. To quantify, 9 out of 231 (0.039%) respondents from all four levels for Question 26, "*How did your team members affect your ability to learn the basic programming concepts being presented*", had responses that could be interpreted as being "negative".

7.1.4 Competition

In this last section, the effects of competition on Space Race will be explored. Questions 19 and 20 provide some insight on this topic. When students were surveyed, at least 68% of the students agreed that "*competition between teams [made] this video game an effective teaching tool for learning basic programming concepts*" (Question 19) across the four levels. This is lower than the proportion of students that felt cooperation between teams made the video game effective in teaching basic programming concepts. However, this is

still a considerably high percentage of students that felt competition added to the game's effectiveness as a teaching tool. In particular, a student "*thought the game was fun...[they liked] the leaderboard because it [challenged] other teams to get a high score*". When students were asked if "*competition between teams [motivated them] to review course material*" (Question 20), at least 60% of the students agreed for all the levels. None of the students reported feeling less motivated to participate as a result of the competitive aspect of the game.

The researcher observed that fewer teams were interested in competing with other teams than in successfully completing each level. To illustrate, a student commented that when they played the game with their team, "*they assisted [him/her] and [he/she] assisted them. When someone did not understand [since they were] not that competitive [they] stopped and helped each other through it*". For some groups, the competitive aspect was less important than the cooperative aspect of the game. An unexpected side effect was the competition that some students felt with the peers within their group. This form of competition was not at the expense of their team members; it was more individualistic in nature. A student felt that they needed to "*try to be the smartest one in the group*". This could serve as motivation for students to review the material. Another student put it simply: "*some [people] knew more, so I wanted to know more*".

Chapter 8

Educational Effectiveness of Game

This chapter will quantify the educational effectiveness of Space Race. Specifically, the pre and post quiz results will be analyzed first followed by the ENG 1D04 exam results. The quiz results demonstrate the immediate effects of Space Race, while the ENG 1D04 exam results demonstrate whether the knowledge obtained from Space Race can be retained over time. The exams also provide an opportunity to examine whether the understanding of programming through Space Race can be transferred to a context outside of the game.

When performing statistical analysis that determines the significance of results, there are two important numbers: alpha (α) and p-value. The p-value is derived from every test statistic and it represents the probability that the observed results occurred by chance alone. For example, a p-value of 0.1 means that there is a 10% chance that the observed results occurred randomly through chance. A significance level of the study must be designated to determine the threshold value that p-values can be measured against. This numerical value is known as alpha. It determines how extreme observed results must be to reject the null hypothesis of any significance test. In the past, researchers most commonly selected 0.05 as the alpha value and this is generally accepted as the standard. This is also the alpha value that has been chosen for this experiment; the significance of the results in the following sections will use an alpha of 0.05. The results can be called statistically significant if the test statistic produces a p-value that is less than or equal to alpha. Contrastingly, any p-value that is greater than alpha would mean that results are not statistically significant. All statistical tests within this chapter were performed using SciPy [16].

8.1 Benchmark for Student Abilities

Since members of the experimental group volunteered to play Space Race, it is important to first assess whether these individuals were pre-disposed to excelling in ENG 1D04 before the exam results of the experimental group can be compared to the control group. That

is, there must be confirmation that there was minimal self-selection bias so that causation can be attributed to Space Race rather than a biased sample group. Furthermore, by benchmarking the abilities of students in the experimental group, one can compare the effects of Space Race between "stronger" and "weaker" students by analyzing the differences in post-quiz results.

A method of benchmarking must be employed to determine the distribution of students in both the experimental and control group. To do this, a combination of the Math 1ZA3 (Engineering Mathematics I) and Physics 1D03 (Introductory Mechanics) exam marks for each student were used. Both of these courses are mandatory for all first year McMaster Engineering students. They are usually taken in Term 1 of the academic year, although some students may opt to take the courses during the summer or at a later time if they decide to divide their first year of study into two years or more. The latter option is rare amongst the first year engineering student population. As such, most of the ENG 1D04 term 2 students, the research participants, will have already taken these courses. Math and physics marks were chosen as indicators because research has shown that misunderstandings in physics, mathematics, and computer programming display strong commonalities with one another. These misunderstandings can arise as the result of parallel or identical causes that occur in each domain [53]. With support from this theory, the researcher has chosen to use math and physics marks as predictors of success in ENG 1D04, since a student's misunderstanding of concepts in either one of those courses may translate into a programming course.

Each student in both the control group and experimental group was assigned a score which is calculated in the following manner:

$$\begin{aligned} M &= \text{Math 1ZA3 Final Exam Mark} \\ P1 &= \text{Physics 1D03 Midterm 1 Exam Mark} \\ P2 &= \text{Physics 1D03 Midterm 2 Exam Mark} \\ Score &= Average(M, Average(P1, P2)) \end{aligned}$$

When using the above equation to determine each student's score, the following methods were applied to special cases:

1. If only one of either Midterm 1 (*P1*) or Midterm 2 (*P2*) was available for Physics, the available mark was used.

2. If only one of either Physics (*Average(P1,P2)*) or Math (M) was available, the available mark was used.

3. If neither a Physics or Math mark was available, the student was omitted from the evaluations used to determine the educational effectiveness of Space Race.

A total of five students could not be assigned a score based on the criterion mentioned above. Table 8.1 shows the number of students that were omitted from the evaluations

according to the experimental and control groups. The number of participants that were eliminated from the study are much smaller than the original sample they were removed from. As such, the elimination of these participants should have a negligible effect on the observed results.

Table 8.1: Number of Participants Eliminated

Level	Experimental Group	Control Group
1	2 out of 236	3 out of 249
2	2 out of 214	3 out of 271
3	2 out of 194	3 out of 291
4	1 out of 190	4 out of 295

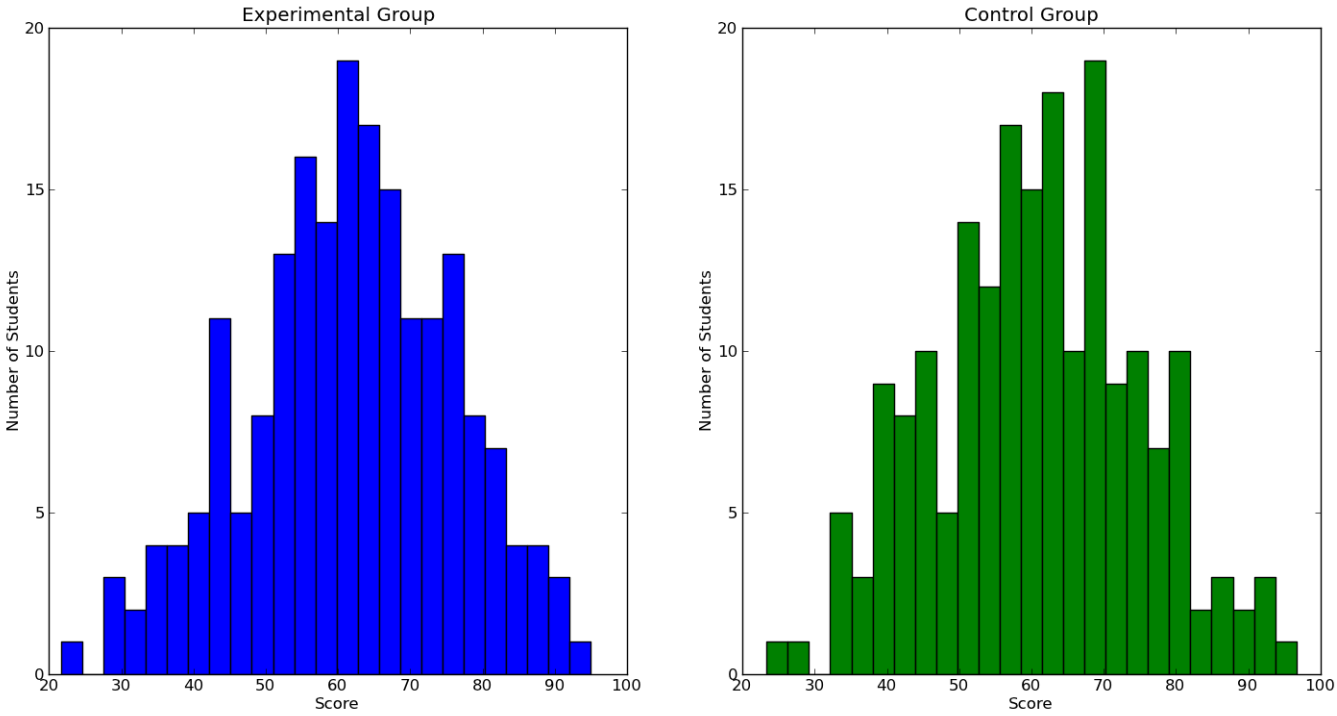
Figure 8.1 shows the distribution of scores between the experimental and control group for each level of Space Race. A quick review of the histograms suggests that the distribution of scores between the experimental and control group are very similar.

A two-sided t-test was performed for each level to confirm that the differences between the distributions of the two groups are insignificant. Since the t-test assumes the samples are normally distributed, the normality test from SciPy was used to verify that both the experimental and control group scores are normally distributed. The normal test in SciPy tests the null hypothesis that a sample comes from a normal distribution. The results are shown in Table 8.2. Given that all the p-values are larger than the level of significance of 0.05, the null hypothesis cannot be rejected. As such, the distribution of scores for both the experimental and control group can be assumed to be normal.

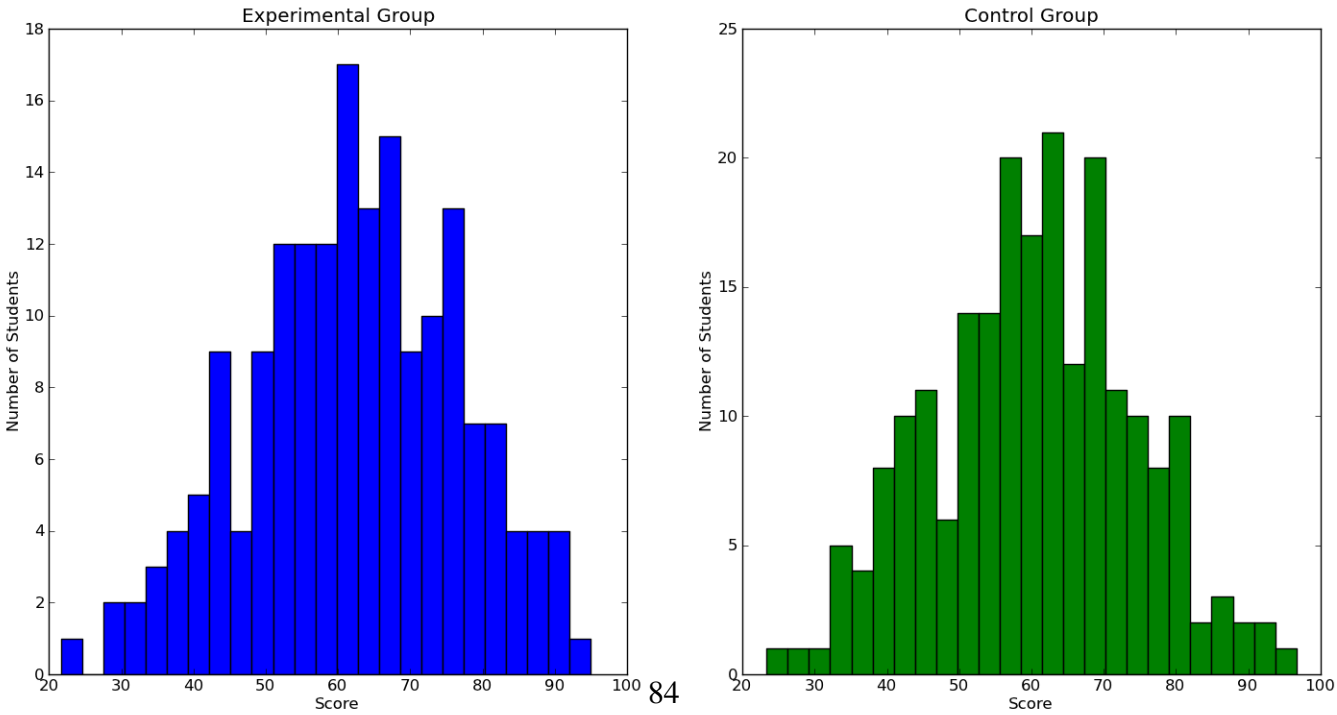
Table 8.2: Normality Test Results for Experimental and Control Group Scores

Level	**Experimental Group p-value**	**Control Group p-value**
1	0.421	0.595
2	0.364	0.699
3	0.081	0.669
4	0.355	0.668

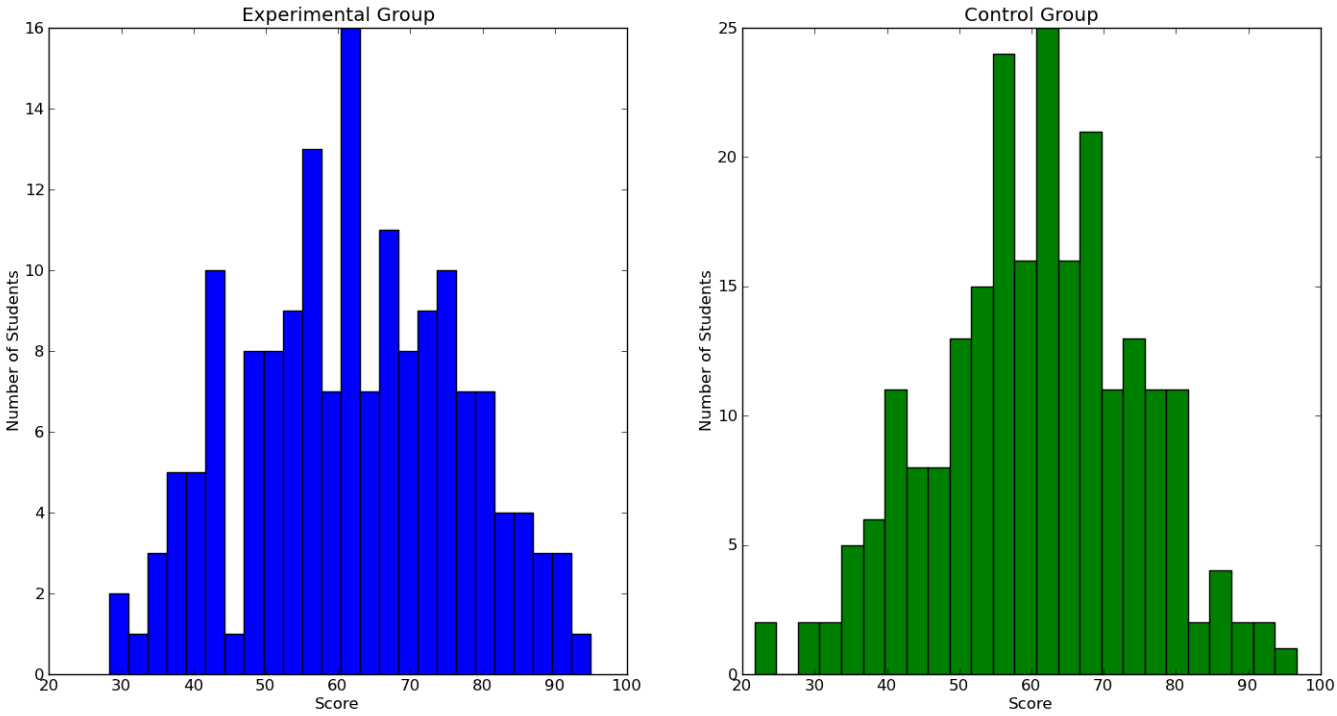
Having established normality for the distribution of scores for both the control and experimental group, the t-test was performed with the null hypothesis that the two independent samples, the experimental and control group, have the same distribution. The p-value can be used to interpret the results of this test. If a large p-value is observed, then the null hypothesis cannot be rejected; this means there is not a significant difference in the



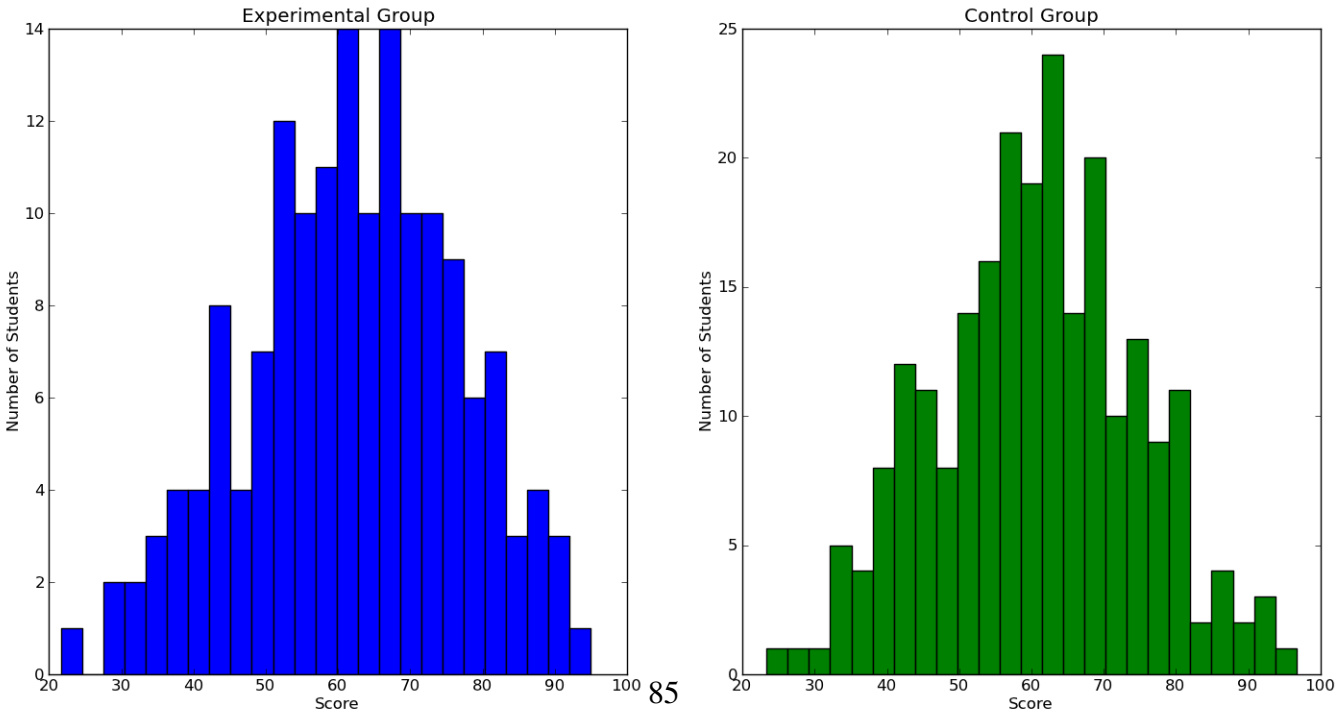
(a) Level 1



(b) Level 2



(c) Level 3



(d) Level 4

Figure 8.1: Score Distributions for Experimental and Control Group by Levels

distribution of scores.

Table 8.3: T-test Results for Comparison of Experimental and Control Group Scores

Level	t-statistic	p-value
1	0.5214	0.6024
2	1.2440	0.2142
3	1.0046	0.3157
4	0.8312	0.4064

The results of the t-test can be seen in Table 8.3 for each level. It can be seen that the p-value for all four levels is well above 0.05. Consequently, the null hypothesis cannot be rejected. This suggests that there is very weak evidence the distribution of scores between the experimental and control group are not the same. This conclusion suggests that there is minimal self-selection bias in the experimental and control group thus strengthening the argument that any differences in performance between the two groups can be attributed to the influence of Space Race.

8.2 Pre and Post Quiz Results

As mentioned previously in Section 5.2.2, the pre and post quizzes are identical and they are completed by the students before and after every level of gameplay in Space Race. These quizzes can be seen in Appendix C. Students were not told which questions they answered correctly for both the pre and post-quiz; students did not receive feedback on quiz performance. Table 8.4 shows the percentage of students that answered correctly on the pre and post-quiz. The table also includes the number of students that successfully completed the pre and post-quiz questions for each level (indicated by N). This number may be less than the number of students that actually played and completed that level of Space Race because some students were unable to submit the post quiz due to technical issues. The quiz data, both pre and post, for these students has been eliminated from the results. While considering the results of the pre and post quiz, it is also important to note that some students may not have taken the quizzes seriously. This may have been because there were no consequences for selecting an incorrect answer. Some of the quiz results may have been affected by this.

McNemar's test was chosen to verify the significance of the differences between the pre and post-quiz results for each question. McNemar's test can be used when the data being investigated is paired and nominal and the outcome of interest is a proportion [47]. In this case, the pair-matched data comes from each individual's results before and after playing

Space Race. The null hypothesis in this instance is that the proportion of subjects that answer a question correctly is the same before and after playing Space Race. The results of McNemar's test can be seen in Table 8.4. The table shows that 65% (13/20) of all the results are significant. Additionally, all of the results that are significant are cases where the proportion of students that were correct in the post-quiz exceeded that of the pre-quiz (positive delta correct). On the other hand, all of the negative delta correct results were relatively small and not statistically significant.

Table 8.4: Pre and Post Quiz McNemar's Test Results

Question	Pre Correct(%)	Post Correct(%)	Delta Correct (%)	McNemar's Chi-Square Statistic	P-Value	Significant at p<0.05?
LEVEL 1	**N= 230**					
1	17.83	40.87	23.04	44.59	0	Yes
2	26.09	30.43	4.35	1.79	0.181	No
3	71.3	69.13	-2.17	0.36	0.547	No
4	39.57	46.96	7.39	6.42	0.011	Yes
5	19.13	60	40.87	83.36	0.000	Yes
LEVEL 2	**N= 188**					
1	96.28	95.21	-1.06	0.33	0.564	No
2	23.94	28.72	4.79	3.24	0.072	No
3	26.6	30.85	4.26	2	0.157	No
4	53.19	54.79	1.6	0.36	0.549	No
5	69.15	85.11	15.96	22.5	0.000	Yes
LEVEL 3	**N= 171**					
1	66.67	73.68	7.02	5.14	0.023	Yes
2	30.41	46.2	15.79	13.75	0.000	Yes
3	66.08	81.29	15.2	16.1	0.000	Yes
4	90.06	90.64	0.58	0.04	0.835	No
5	17.54	82.46	64.91	98.57	0.000	Yes
LEVEL 4	**N= 167**					
1	79.04	92.81	13.77	17.06	0.000	Yes
2	61.08	69.46	8.38	6.53	0.011	Yes
3	49.1	59.88	10.78	9	0.003	Yes
4	72.46	85.03	12.57	13.36	0.000	Yes
5	31.14	61.68	30.54	40.02	0.000	Yes

Level 1 had two questions where the difference in results were not significant: Question 2 and Question 3 (abbreviated to L1Q2 and L1Q3). The insignificance of the results suggests that there was no change in the proportion of students that were correct before and after playing Space Race. Both of these questions were not directly from Space Race Level 1 and required students to abstract and apply what they observed in Space Race to answer the questions. L1Q2 required students to understand Question 43 from Level 1 of the Space Race game. That question emphasized that floats are approximations because of its fixed precision. Hence, `1e30+1==1e30` is true in Python (`1e30` is scientific notation for $1.0x10^{30}$). This question prompted many students to ask the instructor for an explanation

of the results. It was established in the previous chapter that it is easy for students to cheat in Level 1 and there is a general lack of feedback from the game. This combination could have resulted in the poor results on L1Q2. Although many students asked for clarification, other students may have guessed the correct answer and moved on without asking for clarification. In this case, the lack of feedback from the game failed to correct or augment their understanding. The results of L1Q2 emphasize the importance of providing feedback in the game when it is easy for students to proceed in the game without really knowing what they are doing. In that case, the presence of an instructor does not necessarily facilitate learning.

L1Q3 is another question that is not explicitly taught in Level 1 of Space Race; it requires students to understand that integer division always takes place in Python when both the numerator and denominator are integers. This happens regardless of whether there is a remainder value or not. The lack of improvement in this question could be due to the lack of exposure that students have had to integer division in Level 1, or it could mean that students were unable to understand `m%n!=0`. This section of the question may have confused students.

Out of the four levels, students improved the least on the quiz in Level 2 with only one of the positive delta correct cases being significant. L2Q1 saw negative improvement with a delta of -1.1%. However, the percentage of students that were correct before and after playing Space Race is over 95%. This suggests that the question was perhaps too easy; the concept it tested was already understood by the large majority of students prior to playing Space Race.

L2Q2 was used to assess whether students were able to make the connection that immutable values assigned to variables do not change unless a new value is assigned to that variable. The question for L2Q2 is the following:

> **After running the following lines of code:**
> ```
> X=2
> float(X)
> ```
> **the value of X is 2.0. Is this statement true or false?**
>
> a) true
>
> b) false

When students are given an instruction in Space Race like "`Yoke=float(7/2)`", they must change their `Yoke` control to `3.0` because a value has been assigned to their `Yoke` control. Without assignment, none of the immutable values for their controls would change in Level 2. Unfortunately, it appears that students were unable to abstract their understanding from Space Race and apply it to L2Q2. Perhaps, L2Q2 would have been met with more success if there was an explicit example from Space Race where an instruction does not cause any change for any of the controls because assignment has not taken place. Also, it may have

been helpful if the game provided explicit feedback that assisted students in making a connection between the game mechanics and programming concept. In this instance, the game could have revealed to the player that changing control values because of an instruction is much like how the values from variables are changed from variable assignment.

The result for L2Q3 is surprising because of its close relationship with L2Q5. L2Q5 asks students to identify, from a list of types, which type is not mutable. 85% of the students were able to correctly identify `strings` as the correct answer after playing Space Race, a 16% improvement from the pre-quiz. However, when reviewing the results of L2Q3, only 31% of the students were able to select the correct answer. The question for L2Q3 is the following:

> **Assuming variable X is a non-empty string of length 3. Which of these generates a runtime error?**
>
> a) `X[0]="E"`
>
> b) `Y= X[1:2]=="C"`
>
> c) `Y= X[:2]`
>
> d) `Y= X[2:0]`

L2Q3 requires students to identify the option that produces an error; the correct answer is the one where the line of code attempts to change the current string by doing the following: "`X[0]="E"`". The results from L2Q3 and L2Q5 suggests that students know strings are not mutable but they do not understand what this means. Furthermore, all of the options from L2Q3 appear in Level 2 of Space Race. That is, all of the incorrect answers are instructions that students have to execute in Level 2 and the correct answer is an "Error" result in Level 2. The distribution of responses for L2Q3 is shown in Table 8.5. It is worth noting that the instructor noticed that slice notation for strings was intimidating for many students. A lot of students had to ask questions for clarification on instructions that involved slice notation for strings. Since correctly answering such a question requires students to either identify the correct answer or eliminate the incorrect answers, it is unclear whether more practice with slice notation would have improved response rates on the post-quiz. The inability to recognize the "Error" option in L2Q3 as the correct answer may perhaps highlight Space Race's failure to consistently teach students to identify exceptions. As stated in Chapter 4, when more than one player has an instruction with an error, the first error that appears is the first error that is cleared after all four players shake their tablets. It is possible that sometimes two errors are "in-play" and when everyone shakes their tablets with the intention of clearing the second error, the first error is cleared instead. If the student with the first error is not paying attention to their instructions, they may not notice their instruction before it gets cleared. Again, feedback from the game to explain why their error was cleared would be beneficial in this instance.

Table 8.5: L2Q3 Answer Distribution

Answer Option	Pre-Quiz(%)	Post-Quiz(%)
`X[0]="E"`- correct	26.60	30.85
`Y=X[1:2]=="C"`	31.38	31.91
`Y=X[:2]`	7.98	8.51
`Y=X[2:0]`	34.04	28.72

Level 3 and Level 4 had the greatest number of significant results. The only insignificant result is for L3Q4 which, like L2Q1, had a high proportion of students correct for both the pre and post quiz results. This again, may indicate that the question was too simple and could not measure whether a concept could be taught by Space Race. L3Q5 had the highest improvement out of all the questions across the four levels with a delta correct of 65%. This question is pulled directly from Space Race; it is almost identical to one of the instructions that appears in the programs of Level 3. The question for L3Q5 is the following:

If x="BCBC".split("C"), what is the value of x?

a) `["B", "B"]`

b) `["C", "C"]`

c) `["B", "B", ""]`

d) `["", "C", "C"]`

Even when this concept was explained on the "Cheat Sheet", this was one of the instructions that most students struggled with within the game. This is not surprising since the percentage of students that got L5Q1 correct in the pre-quiz is the lowest: 18%. While playing the game, most students would often require additional help from the instructor to execute the instruction when it first appears during gameplay. This instruction is different from the other instructions because it is less intuitive and more difficult to simply guess, thus providing the instructor with an opportunity to explain the concept in more depth. This suggests that perhaps the game is most effective when it is supplemented with explanations from an instructor.

8.2.1 Comparing the Effects of Space Race on Different Students

To compare the effects of Space Race on "stronger" and "weaker" students, all the students within the experimental group will be sorted into three separate bins based on the *Score* they received. The calculations and grades used to obtain these scores were shown in Section 8.1. Each student will be sorted one of the three following categories:

- **Bin 0**: $Score < 50\%$
- **Bin 1**: $50\% \leq Score \leq 75\%$
- **Bin 2**: $75\% < Score$

The students in Bin 0, Bin 1, and Bin 2 represent the weakest, average, and strongest students, respectively. After sorting the students into their respective bins, the quiz results can be obtained for each group. These results are shown in Table 8.6, Table 8.7, and Table 8.8.

Table 8.6: Pre and Post Quiz McNemar's Test Results for Bin 0

Question	Pre Correct(%)	Post Correct(%)	Delta Correct (%)	McNemar's Chi-Square Statistic	P-Value	Significant at p<0.05?
LEVEL 1	**N= 42**					
1	21.43	35.71	14.29	6	0.014	Yes
2	26.19	30.95	4.76	0.4	0.527	No
3	71.43	73.81	2.38	0.09	0.763	No
4	52.38	52.38	0	0	1	No
5	14.29	54.76	40.48	17	0	Yes
LEVEL 2	**N= 31**					
1	96.77	90.32	-6.45	1	0.317	No
2	16.13	22.58	6.45	2	0.157	No
3	25.81	35.48	9.68	3	0.083	No
4	45.16	41.94	-3.23	0.33	0.564	No
5	70.97	87.1	16.13	5	0.025	Yes
LEVEL 3	**N= 30**					
1	53.33	70	16.67	2.78	0.096	No
2	13.33	30	16.67	3.57	0.059	No
3	63.33	73.33	10	1.29	0.257	No
4	90	90	0	0	1	No
5	13.33	90	76.67	23	0	Yes
LEVEL 4	**N= 30**					
1	66.67	90	23.33	5.44	0.02	Yes
2	56.67	66.67	10	1.8	0.18	No
3	36.67	50	13.33	2.67	0.102	No
4	70	86.67	16.67	2.78	0.096	No
5	20	56.67	36.67	11	0.001	Yes

Results show that 6/20 (30%), 11/20 (55%), and 10/20(50%) of the improvements are significant for Bins 0, 1, and 2, respectively. This indicates that Space Race has had a larger positive impact on the average and stronger students than on the weaker students. Also, it appears that the impact of Space Race on stronger and average students are approximately the same.

Table 8.7: Pre and Post Quiz McNemar's Test Results for Bin 1

Question	Pre Correct(%)	Post Correct(%)	Delta Correct (%)	McNemar's Chi-Square Statistic	P-Value	Significant at p<0.05?
LEVEL 1	**N= 117**					
1	13.68	39.32	25.64	25	0	Yes
2	25.64	26.5	0.85	0.04	0.847	No
3	72.65	67.52	-5.13	0.95	0.33	No
4	37.61	47.01	9.4	5.26	0.022	Yes
5	18.8	62.39	43.59	42.64	0	Yes
LEVEL 2	**N= 94**					
1	96.81	95.74	-1.06	0.14	0.705	No
2	26.6	26.6	0	0	1	No
3	26.6	29.79	3.19	0.47	0.491	No
4	51.06	54.26	3.19	0.69	0.405	No
5	70.21	88.3	18.09	12.57	0	Yes
LEVEL 3	**N= 82**					
1	70.73	73.17	2.44	0.33	0.564	No
2	40.24	57.32	17.07	7	0.008	Yes
3	64.63	82.93	18.29	9.78	0.002	Yes
4	87.8	90.24	2.44	0.33	0.564	No
5	18.29	81.71	63.41	46.62	0	Yes
LEVEL 4	**N= 89**					
1	83.15	94.38	11.24	6.25	0.012	Yes
2	65.17	69.66	4.49	1	0.317	No
3	48.31	59.55	11.24	4.55	0.033	Yes
4	73.03	86.52	13.48	8	0.005	Yes
5	33.71	60.67	26.97	16.94	0	Yes

8.3 Exam Results

The following section will compare the midterm and final exam results between the experimental and control group. These results provide insight into how well knowledge obtained from Space Race is retained over time. Furthermore, the results support an investigation into whether the knowledge attained in Space Race is applicable in a context outside of the game world.

8.3.1 Midterm Exam Results

This section compares the performance of the experimental group to the control group on the midterm exams. Details of the midterm exams can be found in Section 5.2.3 and the questions can be found in Appendix E.1 and E.2. The questions on the exam are all in a multiple-choice format. The researcher chose ten questions from Midterm 1 and two questions from Midterm 2 to investigate. These questions were chosen because they tested material that was covered in Space Race. Only two questions from Midterm 2 were selected, since the material tested in Midterm 2 covers mainly weeks 6 through 9 of ENG 1D04, while Space Race only covers material from weeks 1 through 5. After the questions

Table 8.8: Pre and Post Quiz McNemar's Test Results for Bin 2

Question	Pre Correct(%)	Post Correct(%)	Delta Correct (%)	McNemar's Chi-Square Statistic	P-Value	Significant at p<0.05?
LEVEL 1	**N= 39**					
1	28.21	46.15	17.95	5.44	0.02	Yes
2	28.21	46.15	17.95	4.45	0.035	Yes
3	71.79	71.79	0	0	1	No
4	35.9	38.46	2.56	0.14	0.705	No
5	17.95	69.23	51.28	20	0	Yes
LEVEL 2	**N= 34**					
1	97.06	100	2.94	1	0.317	No
2	23.53	44.12	20.59	7	0.008	Yes
3	20.59	26.47	5.88	2	0.157	No
4	55.88	64.71	8.82	1.29	0.257	No
5	64.71	73.53	8.82	1.8	0.18	No
LEVEL 3	**N= 31**					
1	70.97	80.65	9.68	3	0.083	No
2	29.03	51.61	22.58	4.45	0.035	Yes
3	67.74	87.1	19.35	4.5	0.034	Yes
4	90.32	93.55	3.23	0.33	0.564	No
5	16.13	83.87	67.74	21	0	Yes
LEVEL 4	**N= 26**					
1	76.92	92.31	15.38	4	0.046	Yes
2	57.69	65.38	7.69	1	0.317	No
3	69.23	88.46	19.23	5	0.025	Yes
4	76.92	88.46	11.54	3	0.083	No
5	30.77	69.23	38.46	10	0.002	Yes

were chosen, each question was assigned one or more levels of Space Race according to its relevance to the material covered in those levels. For example, if a question is assigned Levels 1 and 2 then the core concept required to solve that question is covered in both Level 1 or Level 2. This is necessary to determine which students belong to the experimental and control group when comparing results between the two samples, especially since not everyone that participated completed all of the levels. A student is considered to belong to the experimental group if they have played either one of the levels that have been assigned to that question. For example, a question that has been assigned Level 1 and Level 2 would have students that have played either Level 1 or Level 2 in the experimental group.

Table 8.9 and Table 8.10 shows the proportion of students that were correct in both the experimental and control group for each question of the midterm exams. The assigned levels for each question are also listed along with the delta correct value, which shows the difference in percentage of students that were correct between the experimental and control group.

The Chi-square statistic was chosen to test the significance of each of the results. This statistic can be used to investigate whether the distribution of the categorical variables differ from one another [48]. It compares the counts of categorical responses between two or more independent groups. In this scenario, the two independent groups are the exper-

imental and control group, while the categorical responses are either correct or incorrect for an individual question. The null hypothesis posits that the proportion of students that are correct do not differ between the experimental and control group. The results of the Chi-test is shown in both Table 8.9 and Table 8.10.

Table 8.9: Midterm 1 Chi-Square Test Results

Question	Levels	Experimental Group Correct(%)	Control Group Correct(%)	Delta Correct (%)	Chi-Square Statistic	P-Value	Significant at p<0.05?
1	1	54.69	50.00	4.69	0.64	0.423	No
5	1 2	89.58	93.33	-3.75	1.22	0.270	No
9	1	96.35	94.44	1.91	0.40	0.527	No
10	4	46.75	36.24	10.51	3.71	0.054	No
13	3	51.92	46.76	5.16	0.77	0.380	No
17	4	74.03	63.30	10.72	4.27	0.039	Yes
20	3 4	35.26	29.63	5.63	1.07	0.300	No
23	2 3 4	59.30	47.50	11.80	4.71	0.030	Yes
27	3 4	70.51	63.43	7.09	1.73	0.188	No
28	3	70.51	57.87	12.64	5.69	0.017	Yes

Table 8.10: Midterm 2 Chi-Square Test Results

Question	Levels	Experimental Group Correct(%)	Control Group Correct(%)	Delta Correct (%)	Chi-Square Statistic	P-Value	Significant at p<0.05?
5	1	78.80	75.14	3.67	0.50	0.479	No
27	3 4	45.70	30.37	15.32	8.30	0.004	Yes

The results show that 4/12 (33%) of the questions had a significant difference in performance between the experimental and control group. All of these significant results are for a positive delta correct value with the largest improvement being 15% for Question 27 on Midterm 2. That is, all of the significant results are represented by cases where the experimental group outperforms the control group. None of the negative delta correct results were significant. From this, one can conclude that students from the experimental group either perform equally well or better than the students from the control group. There also does not seem to be a trend in determining which types of questions would guarantee that students from the experimental group would outperform the control group. The only pattern that can be drawn from the results is that any question involving Level 1 did not result in a positive delta correct value that is significant. This suggests that Level 1 does not have

a significant impact on the experimental students' results on the midterm exams.

8.3.2 Final Exam Results

The Final Exam results are presented in the same manner as the Midterm 1 and Midterm 2 results in Section 8.3.1. The questions (seen in Appendix E.3) and their associated Space Race level were, once again, assigned by the researcher. Also, the Chi-square statistic was used again to test the significance of results, as shown in Table 8.11. The conclusions that can be drawn from the results of the Final Exam differ from the results of the midterm exams because it occurs at a much later time point in the experimental timeline. There is at least a total of 7 weeks between each student's last exposure to Space Race and their completion of the Final Exam. The performance of the experimental group on the Final Exam can indicate whether the students were able to retain the concepts they learned in Space Race over a greater time period.

Table 8.11: Final Exam Chi-Square Test Results

Question	Levels	Experimental Group Correct(%)	Control Group Correct(%)	Delta Correct (%)	Chi-Square Statistic	P-Value	Significant at p<0.05?
4	1	51.83	51.08	0.76	0.00	0.965	No
5 (L3Q5)	3 4	77.56	36.20	41.37	61.22	0.000	Yes
6 (L2Q3)	2 3	48.26	32.68	15.57	8.83	0.003	Yes
7 (L1Q5)	1	80.63	63.98	16.65	12.26	0.000	Yes
11	4	88.16	82.67	5.49	1.73	0.189	No
16	1 2	84.82	84.95	-0.13	0.01	0.913	No
17	1	37.70	43.55	-5.85	1.11	0.293	No
21	4	84.87	87.11	-2.24	0.22	0.640	No
23	3 4	82.69	79.19	3.51	0.51	0.474	No
29	3 4	76.28	61.54	14.74	8.42	0.004	Yes
44	3 4	33.33	21.27	12.07	6.27	0.012	Yes
46	1	64.92	67.74	-2.82	0.22	0.638	No

As stated in Section 5.2.3, questions 5, 6, and 7 on the final exam were set by the researcher, not the instructor. The questions are as follows:

Question 5 (L3Q5): If x="BCBC".split("C"), what is the value of x?

a) `["B", "B"]`

b) `["C", "C"]`

c) `["B", "B", ""]`

d) `["", "C", "C"]`

Question 6 (L2Q3): Assuming variable X is a non-empty string of length 3. Which of these generates a runtime error?

a) `X[0]="E"`

b) `Y= X[1:2]=="C"`

c) `Y= X[:2]`

d) `Y= X[2:0]`

Question 7 (L1Q5): Which of these generates a runtime error?

a) `"b"*4`

b) `X=6!=5`

c) `X=str(5)`

d) `X=int("b")`

e) `X=4L/2`

These questions are identical to the pre and post quiz questions given in Space Race: Question 5 from L3Q5, Question 6 from L2Q3, and Question 7 from L1Q5. Question 5 and Question 7 were chosen because both questions had a relatively large improvement from the pre-quiz to the post-quiz. L3Q5 (Question 5) had a delta correct of 65%, while L1Q5 (Question 7) had a delta correct of 41% from pre to post-quiz. Question 6 (L2Q3) was chosen because there was no significant change in results between the pre and post-quiz responses with a delta correct of 4.3% and both the pre and post-quiz responses had a low proportion of students that got the answer correct (27% and 31%, respectively). The final exam results for these questions could indicate if the students that completed the quiz questions correctly retained their knowledge over several weeks.

Table 8.12 shows a comparison of how students that completed the post-quiz for each relevant question performed on the exam. For example, for Question 5 (L3Q5), 141 students had the correct answer on their post quiz; of those 141 students, only 114 students had the correct answer on their final exam (81%). Alternatively, 30 students got Question 5 (L3Q5) incorrect on their post-quiz but 15 of those students got the correct answer on the final exam (50%). The results show that 55% of the students that had the correct answer on their post-quiz for Question 6 selected the correct answer again on the final exam. However, Question 5 and Question 7 showed significantly better retention rates, as 80% of the

students that selected the correct answer in the post-quiz selected the correct answer again in the final exam. These results indicate that not all of the students retained their knowledge after playing Space Race, but the majority did. In particular, questions that saw a large delta correct from pre to post-quiz results had a higher retention rate. Comparatively, a question that had no significant change in pre to post-quiz performance had a lower retention rate. In all cases, some of the students that originally selected the incorrect answer on the post-quiz, chose the correct answer on the final exam. It is not clear whether these students performed better on the final exam because Space Race motivated them to investigate the topic being tested, but it could be a contributing factor.

Questions 5, 6, and 7 had the largest difference in performance between the experimental group and the control group as seen in Table 8.11. Out of these questions, Question 5 had the biggest delta correct value (41%). Interestingly, Question 5 (L3Q5) also had the greatest delta correct value for the pre and post quiz as seen in Table 8.4. This could suggest that most of the students that did not play Space Race failed to learn the concept tested in Question 5 without the intervention of Space Race.

Overall, only 5/12 (42%) of the results from Table 8.11 were significant. None of the negative delta correct values were significant results. This, again, suggests that students that played Space Race either performed equally well or better than the students that did not play Space Race. For Questions 5, 6, and 7, it is unclear whether the experimental group performed better than the control group simply because they have seen the questions before. Although, it is important to note that the post-quiz did not provide any feedback to students. That is, students had no idea whether their answer was correct or incorrect unless they sought the answer out themselves.

Finally, a comparison of knowledge retention results amongst stronger, average, and weaker students, belonging to the experimental group, was performed on Question 5 (L3Q5), Question 6 (L2Q3), and Question 7 (L1Q5). Students belonging to the experimental group were divided again into Bin 0, Bin 1, and Bin 2 (shown in Section 8.2.1). Table 8.13 shows the percentage of students that got the correct answer on the post-quiz and the final exam from each bin. For example, 70.37% of the students from Bin 0 answered Question 5 (L3Q5) correctly on the post-quiz and the final exam. The table also shows the percentage of students that selected the incorrect answer on the post-quiz and then selected the correct answer on the final exam. For example, 30.00% of the students from Bin 0 had the incorrect answer on the post-quiz for Question 6 (L2Q3), but a correct answer on the final exam. These results suggests that the weaker students, represented by Bin 0, are less likely to retain the knowledge obtained from Space Race than the average and stronger students, represented by Bin 1 and Bin 2, respectively. There also appears to be similar retention rates amongst average and stronger students. When reviewing the results of the students that did not select the correct answer on the post-quiz but later rectified their errors and selected the correct answer on the final exam, it appears, again, that the average and stronger students perform better than the weaker students. Also, in most cases, the aver-

Table 8.12: Final Exam Question Results Compared to Post-Quiz Question Results

	Post-Quiz Correct Sample		
	Post-Quiz Correct	**Exam Correct**	**Percentage (Exam/Post-Quiz)**
Question 5 (L3Q5) N= 171	141	114	80.85%
Question 6 (L2Q3) N= 188	58	32	55.17%
Question 7 (L1Q5) N= 230	138	111	80.43%

	Post-Quiz Incorrect Sample		
	Post-Quiz Incorrect	**Exam Correct**	**Percentage (Exam/Post-Quiz)**
Question 5 (L3Q5) N= 171	30	15	50%
Question 6 (L2Q3) N= 188	130	51	39.23%
Question 7 (L1Q5) N= 230	92	70	76.09%

age students and stronger students perform equally well, with the exception of Question 5 (L3Q5) where the average students outperform the stronger students. This indicates that the weaker students are less likely to correct their mistakes after the intervention of Space Race in comparison to the average and stronger students.

Table 8.13: Final Exam Question Correct Results Compared to Post-Quiz Question Results for Bins 0, 1, and 2

	Post-Quiz Correct Population		
	Bin 0	**Bin 1**	**Bin 2**
Question 5 (L3Q5)	70.37%	86.57%	80.77%
Question 6 (L2Q3)	45.45%	67.86%	55.56%
Question 7 (L1Q5)	65.22%	84.93%	83.33%

	Post-Quiz Incorrect Population		
	Bin 0	**Bin 1**	**Bin 2**
Question 5 (L3Q5)	0.00%	73.33%	25.00%
Question 6 (L2Q3)	30.00%	42.42%	41.67%
Question 7 (L1Q5)	63.16%	75.00%	75.00%

Chapter 9

Conclusions

In this final chapter of the thesis, the work of the previous chapters is brought together and discussed to identify what the study, as a whole, has discovered. The thesis began with an identification of a problem: students in higher education are unmotivated to review introductory programming course material and this reluctance to engage with the material has had a negative impact on their learning. From this, the researcher proposed that a video game could be used to improve student engagement with new programming knowledge.

Chapter 2 highlighted the potential for video games to facilitate experiential and collaborative learning; both of which stem from a constructivist learning approach. The chapter also provided a rationale for the incorporation of video games in education. This rationale does not simply stem from the supposition that individuals are intrinsically motivated to play games. It is also supported by the argument that video games provide a stimulating environment that satisfies the needs of modern day learners known as the "Games Generation". The chapter also highlights how a combination of cooperation and competition can motivate participation in games from people of all orientations: competitive, cooperative, or individualistic.

A review of literature was presented in Chapter 3. This chapter revealed that social collaboration can be highly beneficial for learners and programming is a topic that is well suited for learning through collaborative work. The chapter also showed that there is a demand and interest from students in higher education for a game-based learning approach to learn software engineering concepts. Finally, the chapter concluded with a description of the limited amount of work researchers have done in the past to explore the viability and effectiveness of a game in instructing computing concepts. This highlighted the need for more research in this field of study with an emphasis on the demand for larger test groups and quantification of results.

Chapter 4 provided a description of the design goals for Space Race, the educational video game used for the study. The two primary goals of implementing cooperation and competition within the game were set based upon the information obtained in Chapter 2.

This chapter also gave a comprehensive description of the game mechanics of each level in Space Race. The programming concepts that were taught in each level were also listed.

The experimental procedure and data collection timeline was outlined in Chapter 5. In addition to this, participant sizes and incentives for participation was specified. The chapter also provided a detailed description of the surveys and assessments that were used to collect data from participants.

The purpose of Chapter 6 was to provide the reader with more insight on the student population that this research was conducted upon. From surveys, the researcher was able to conclude that an overwhelming majority of students could be considered novice programmers with very few students that have had practical experience with the subject. The chapter also revealed that students from both the experimental and control group have approximately the same video gaming habits. Only 7.06% of the control group and 12.62% of the experimental group have played an educational video game in the past. Even so, 63.69% of the control group and 67.32% of the experimental group agreed that they would enjoy playing video games to learn.

The feedback for Space Race was presented in Chapter 7. Space Race was well-received with at least 82.22% of the students reporting that they enjoyed playing the video game for all four levels. Some students commented that playability could be improved by decreasing the chaos in Levels 2 and 3 and by improving the game tutorials. At least 85% of the students, for all four levels, felt that Space Race helped them to learn the basic programming concepts being presented. However, some students also expressed that the game does not directly teach programming concepts. They suggested that more feedback in the game could remedy this problem. The researcher also highlighted how Space Race was an effective teaching tool because it can help instructors identify student weaknesses by lowering student anxiety towards asking questions. The chapter also confirmed that the collaborative aspect of the game was imperative to the teachability of Space Race. As evidence of this, only 0 to 2.27% of the students disagreed that cooperation between team members made this video game an effective teaching tool for learning basic programming concepts across all four levels. Moreover, the cooperation in Space Race has motivated at least 72.84% of the students to review course material so that they can be effective team members. Finally, the competition present in Space Race appears to have motivated at least 59.55% of the students to review their course material. Also, some teams placed less value in competing with other teams; instead, they focused their efforts towards helping each other to successfully complete each level.

Lastly, Chapter 8 quantified the educational effectiveness of Space Race. Analysis of the pre and post-quiz results showed that 13/20 (65%) of the results were significantly different. All of these significantly different results were improvements on the post-quiz after the intervention of Space Race. The greatest improvement had 64.91% more students correct on the post-quiz compared to the pre-quiz. The chapter determined that there was minimum self-selection bias from the experimental group thus strengthening the argu-

ment that differences in performance between the experimental and control group could be attributed to Space Race. A comparison of the midterm performance between the experimental and control group revealed that 4/12 (33.33%) of differences were significant and all of these differences were in favour of the experimental group. A maximum difference of 15.32% more experimental students got a question correct in comparison to the control students. The final exam had 5/12 (41.67%) questions that were significantly different in result between the experimental and control group. These different results were, again, in favour of the experimental group. 41.37% was the maximum difference in the number of experimental students that were correct for a question in contrast to the control group. Three of these questions on the final exam were identical to the quiz questions. At most, 80.85% of the experimental students that got the answer correct on the post-quiz also got the question correct on the exam. This result provides the basis for the conclusion that knowledge obtained in Space Race was retained for at least 7 weeks by some participants.

This thesis has explored the effectiveness of a cooperative and competitive video game in improving student understanding of basic programming knowledge. Based on the results of this study, game-based learning can be effectively used to teach basic computer programming concepts to students in higher education. Students can find the game enjoyable and those that play the game can outperform other students that do not play the game on course exams. In particular, cooperation within the game world can be used effectively to encourage students to teach one another. Also, in some cases, the knowledge learned in the game can be transferable to a non-gaming context and this knowledge can be retained by some students. From the findings described above, it appears that the proposed solution of incorporating a video game into course curriculum to teach programming and motivate students to engage with course material was more effective than not.

9.1 Limitations and Future Work

There are limitations to the study that was conducted and there is still work to be done to gain a better understanding of how video games can be used to teach programming. There are a vast number of ways that a game can be designed to teach computer programming. Space Race is only one example of a video game that can be incorporated into course curriculum. There is still much left to investigate with regards to the educational effectiveness of a video game. Specifically, this study was able to show, in a limited manner, that knowledge acquired from gameplay can be retained by some students over 7 weeks. This was shown with three questions on the final exam that was also presented in the pre and post-quiz. In future work, a much stronger argument can be made for knowledge retention if there were more questions or a different experimental design.

This study also demonstrated that students that played Space Race, an educational video game, were able to outperform students that did not play Space Race on course exams in

some instances. However, it does not directly compare the educational effectiveness of the video game to more traditional methods of teaching. One could argue that gaming participants did better in some cases simply because they spent more time on computing. Te question is whether non-gaming participants, that spent the same amount of time with traditional methods of learning, such as reading a textbook or attending a tutorial, would perform equally well, or surpass the performance of gaming participants.

Space Race was, in general, well-received by students. Many students enjoyed playing the game and found it to be beneficial for their learning of basic programming concepts. However, there is still space for many improvements and additions in Space Race. Perhaps one of the greatest enhancements that can be added to Space Race is the introduction of feedback during gameplay. This feedback provided by the game could ensure that the correct knowledge is being transferred to the students when the instructor is not available to answer questions. Overall, this feedback could improve the teachability of Space Race. However, this could be attained at the cost of decreasing collaboration. An investigation should be conducted to measure the trade-off between cooperation and in-game feedback.

For Levels 2 and 3, some students felt that there was too much chaos introduced during gameplay. This chaos arises because all four players were simultaneously given different instructions to execute. Chaos was initially introduced in the game to ensure that the game would remain engaging and exciting throughout gameplay. However, the chaos incorporated into the game also hindered the students' abilities to learn. This could be remedied by altering the game to serve instructions to two players at a time instead of all four. By decreasing chaos, it is possible that a student's ability to learn new concepts while playing Space Race could be increased. However, more research must be conducted to find a balance between the fun and problems that chaos brings into Space Race.

The game tutorial for Space Race can also be improved. As mentioned previously, many students did not read the instructions and expressed an interest in seeing a more interactive tutorial. Future work could investigate and compare the effectiveness of video tutorials and on-demand help provided during gameplay.

Finally, the range of introductory programming topics currently covered in Space Race is limited. There is certainly space for more levels to cover more programming concepts. These topics include, but are not limited to, conditional statements, function definitions, `while`-loops, classes, and algorithm development.

Given the relative success of Space Race and its potential as a teaching tool, it is hoped that the work presented here will motivate other educators and researchers to consider and evaluate the inclusion of video games in higher education to engage the learners of this generation.

Bibliography

[1] Alexandra MacGill Amanda Lenhart, Sydney Jones. Adults and video games. Technical report, Pew Internet & American Life Project, 2008.

[2] Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Rich Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovic. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 59–68. ACM, 2012.

[3] Elliott M Avedon. The structural elements of games. *The study of games*, pages 419–426, 1971.

[4] Parul Khurana Balraj Kumar. Gamification in education-learn computer programming with fun. *International Journal of Computers and Distributed Systems*, 2:46–53, 2012.

[5] Kevin Bierre, Phil Ventura, Andrew Phelps, and Christopher Egert. Motivating OOP by blowing things up: an exercise in cooperation and competition in an introductory java programming course. 38(1):354–358, 2006.

[6] Brad Paras Jim Bizzocchi. Game, motivation, and effective learning: An integrated model for educational game design. In *Digital Games Research Association: Changing Views- Worlds in Play*, 2005.

[7] Maxwell N Burton-Chellew, Adin Ross-Gillespie, and Stuart A West. Cooperation in humans: competition between groups and proximate emotions. *Evolution and Human behavior*, 31(2):104–108, 2010.

[8] Kevin Browne Christopher Anand. Gamification and serious game apapproach for introductory computer science tablet software. In *Gamification 2013*, Stratford Ontario, October 2013. University of Waterloo.

[9] Daniel C Cliburn. Experiences with pair programming at a small college. *Journal of Computing Sciences in Colleges*, 19(1):20–29, 2003.

[10] Andy Cockburn and Andrew Bryant. Cleogo: collaborative and multi-metaphor programming for kids. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, pages 189–194. IEEE, 1998.

[11] Thomas M Connolly, Mark Stansfield, and Thomas Hainey. An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416–428, 2007.

[12] P. A. Cooper. Paradigm shifts in designed instruction: from behaviourism to cognitivism to constructivism. *Educational Technology*, 33:12–19, 1993.

[13] Isabella Selega Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge University Press, 1992.

[14] Mihaly Csikszentmihalyi. *Finding flow: The psychology of engagement with everyday life*. Basic Books, 1997.

[15] Timothy H. DeClue. Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *J. Comput. Sci. Coll.*, 18(5):49–56, May 2003.

[16] SciPy developers. Scipy.org. http://www.scipy.org/, 2014. Accessed: July 19, 2014.

[17] Mary Jo Dondlinger. Educational video game design: A review of the literature. *Journal of Applied Educational Technology*, 4:21–31, 2007.

[18] Mary Jo Dondlinger. Educational video game design: A review of the literature. *Journal of Applied Educational Technology*, 4(1):21–31, 2007.

[19] Martin Ebner and Andreas Holzinger. Successful implementation of user-centered game based learning in higher education: An example from civil engineering. *Computers & Education*, 49(3):873 – 890, 2007.

[20] S. Egenfeldt-Nielsen. Overview of research on the eduational use of video games. *Digital Kompentanse*, 1:184–213, 2006.

[21] McMaster Engineering. Epic experiential playground and innovation classroom. `http://epiclab.mcmaster.ca/`. Accessed: July 7, 2014.

[22] Nick V. Flor and Edwin L. Hutchins. A case study of team programming during perfective software maintenance. In *Empirical studies of programmers: Fourth workshop*, page 36. Intellect Books, 1991.

[23] Charles Wesley Ford Jr. and Steven Minsker. TREEZ-an educational data structures game. *Journal of Computing Sciences in Colleges*, 18(6):180–185, 2003.

[24] James Paul Gee. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)*, 1(1):20–20, 2003.

[25] Johan Huizinga and Richard Francis Carrington Hull. *Homo ludens. A study of the play-element in culture.[Translated by RFC Hull.].* Routledge & Kegan Paul, 1949.

[26] David Hume. Emotions and moods. *Organizational Behavior*, pages 258–297, 2013.

[27] David Hung. Theories of learning and computer-mediated instructional technologies. *Educational Media International*, 38(4):281–287, 2001.

[28] Ken Kahn. A computer game to teach programmming. In *National Educational Computing Conference: NECC '00: Spotlight on the Future: Technology for the NEw Millenium*, 1999.

[29] Slava Kalyuga. Enhancing instructional efficiency of interactive e-learning environments: A cognitive load perspective. *Educational Psychology Review*, 19(3):387–399, 2007.

[30] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences*, 47:1991–1999, 2012.

[31] Caitlin Kelleher, Dennis Cosgrove, David Culyba, Clifton Forlines, Jason Pratt, and Randy Pausch. Alice2: programming without syntax errors. In *User Interface Software and Technology*. Citeseer, 2002.

[32] David A Kolb et al. *Experiential learning: Experience as the source of learning and development*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1984.

[33] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[34] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*, volume 37, pages 14–18. ACM, 2005.

[35] Timo Lainema. *Enhancing organizational business process perception: Experiences from constructing and applying a dynamic business simulation game*. Turku School of Economics and Business Administration, 2003.

[36] R. C. Landers, R. N. Callan. *Casual Social Games as Serious Games: The Psychology of Gamification in Undergraduate Education and Employee Training*. Springer London, 2011.

[37] Marie Larochelle, Nadine Bednarz, and James W Garrison. *Constructivism and education*. Cambridge University Press, 1998.

[38] Jeremy Lee, Kathleen Luchini, Benjamin Michael, Cathie Norris, and Elliot Soloway. More than just fun and games: Assessing the value of educational video games in the classroom. In *CHI'04 Extended Abstracts on Human Factors in Computing Systems*, pages 1375–1378. ACM, 2004.

[39] Frederick Li, Jianmin Zhao, TimothyK. Shih, Rynson Lau, Qing Li, and Dennis McLeod. Introductory C programming language learning with game-based digital learning. In *Advances in Web Based Learning - ICWL 2008*, volume 5145 of *Lecture Notes in Computer Science*, pages 221–231. Springer Berlin Heidelberg, 2008.

[40] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22:140, 1932.

[41] Ju Long. 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, 2007.

[42] John Maloney, Leo Burd, Yasmin Kafai, Natalie Rusk, Brian Silverman, and Mitchel Resnick. Scratch: A sneak preview. In *Proceedings of the Second International Conference on Creating, Connecting and Collaborating Through Computing*, C5 '04, pages 104–109, Washington, DC, USA, 2004. IEEE Computer Society.

[43] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[44] Maria Vivrou George Katsionis Konstantinos Manos. Combining software games with education: Evaluation of its educational effectiveness. *Educational Technology & Society*, 8:54–65, 2005.

[45] Andrew Martin. The design and evolution of a simulation/game for teaching information systems development. *Simulation & Gaming*, 31(4):445–463, 2000.

[46] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. The effects of pair-programming on performance in an introductory programming course. *SIGCSE Bull.*, 34(1):38–42, February 2002.

[47] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[48] David S Moore. *Chi-square tests*. Defense Technical Information Center, 1976.

[49] Mathieu Muratet, Patrice Torguet, Jean-Pierre Jessel, and Fabienne Viallet. Towards a serious game to help students learn computer programming. *International Journal of Computer Games Technology*, 2009:3, 2009.

[50] Emily Oh Navarro and André van der Hoek. Simse: an educational simulation game for teaching the software engineering process. In *ACM SIGCSE Bulletin*, volume 36, pages 233–233. ACM, 2004.

[51] Donald A Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999.

[52] Marina Papastergiou. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education*, 52(1):1 – 12, 2009.

[53] David N Perkins and Rebecca Simmons. Patterns of misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research*, 58(3):303–326, 1988.

[54] Maja Pivec, Olga Dziabenko, and Irmgard Schinnerl. Aspects of game-based learning. In *3rd International Conference on Knowledge Management, Graz, Austria*, pages 216–225, 2003.

[55] Marc Prensky. *Digital Game-Based Learning*, volume 9. Paragon House, 2007.

[56] David Preston. Pair programming as a model of collaborative learning: A review of the research. *J. Comput. Sci. Coll.*, 20(4):39–45, April 2005.

[57] Tahira M. Probst, Peter J. Carnevale, and Harry C. Triandis. Cultural values in intergroup and single-group social dilemmas. *Organizational behavior and human decision processes*, 77(3):171–191, 1999.

[58] Rathika Rajaravivarma. A games-based approach for teaching the introductory programming course. *SIGCSE Bull.*, 37(4):98–102, December 2005.

[59] M. Rebetez, C. Betrancourt. Education research in cognitive and educational sciences. *Cognitie, Creier, Comportament/ Cognition, Brain, Behaviour*, 11:131–142, 2007.

[60] Colin Robson. *Real World Research*. Wiley, 3 edition, 2011.

[61] John R. Savery and Thomas M. Duffy. Problem based learning: An instructional model and its constructivist framework. *Educational technology*, 35(5):31–38, 1995.

[62] Henry Smith. Spaceteam. `http://www.sleepingbeastgames.com/spaceteam/`, July 2014. Accessed: July, 2, 2014.

[63] Jan-Willem Strijbos. *The Effect of Roles on Computer-Supported Collaborative Learning*. PhD thesis, Open University of the Netherlands, 2004.

[64] Jacqueline S. Thousand and Richard A. Villa. *Creativity and Collaborative Learning: A Practical Guide to Empowering Students and Teachers*. Paul H Brookes Pub Co, 1995.

[65] Paivi Tynjala. Towards expert knowledge? a comparison between a constructivist and a traditional learning environment in the university. *International Journal of Educational Research*, 31:357–442, 1999.

[66] Peter Vorderer, Tilo Hartmann, and Christoph Klimmt. Explaining the enjoyment of playing video games: the role of competition. In *Proceedings of the second international conference on Entertainment computing*, pages 1–9. Carnegie Mellon University, 2003.

[67] Lev Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, 1978.

[68] L. K. Ryan L. Webster, J. Trevino. The dimensionality and correlates of flow in human- computer interaction. *Computers in Human Behaviour*, 9:411–426, 1993.

[69] Nicola Jane Whitton. *An investigation into the potential of collaborative computer game-based learning in Higher Education*. PhD thesis, Napier University, 2007.

[70] Eric Klopfer Susan Yoon. Developing games and simulations for today and tomorrow's tech savvy youth. *TechTrends*, 49:33–41, 2005.

Appendix A

Level Solutions for Space Race

SOLUTIONS FOR LEVEL 1

#	Code	Question	Option A	Option B	Option C	Option D	Answer	Panel
1	X= 5+6	X=?	56	10	11	12	C	INT
2	X=4%2	X=?	2	0	4	1	B	INT
3	X=5/2	X=?	2.5	2	3	1	B	INT
4	X= “a”+2	X=?	a2	aaa	aa	2a	ERROR	ERROR
5	X=”a”*5	X=?	a5	a*5	5a	aaaaa	D	STR
6	X= 5.0/2.0	X=?	2	2.0	2.5	1	C	FLOAT
7	X=5 Y=3 Y=X	Y=?	True	False	5	3	C	INT
8	X=5L/2	X=?	2	2.5	0	1	A	LONG
9	X= 5+3/4	X=?	2	6	5.75	5	D	INT
10	X=5L>5	Z=?	True	False	5	4	B	BOOL
11	X= 5/2.0	X=?	2	0.5	1	2.5	D	FLOAT
12	X= 5+6.0	X=?	11	10	56	12	A	FLOAT
13	X=5 Y=5.0 Z= X==Y	Z=?	5	5.0	True	False	C	BOOL
14	X=3 Y==X	Y=?	3	5	True	False	ERROR	ERROR
15	X=2*3	X=?	8	0	1	6	D	INT
16	X=”b” Y=”a” Z=X+Y	Z=?	ab	ba	a+b	c	B	STR
17	X=9%2	X=?	4	1	4.5	5	B	INT
18	X=9%2.0	X=?	4	1	4.5	5	B	FLOAT
19	X=”a” Y=”b” X=”c” Z=X+Y	Z=?	ab	ba	cb	bc	C	STR
20	X=”a” Y=”A” Z=X<Y	Z=?	aA	Aa	True	False	D	BOOL
21	X=5 Y=”5” Z=X==Y	Z=?	True	False	5	five	D	BOOL
22	X=”aaaa”%2	X=?	a	0	aaaa	aa	ERROR	ERROR

23	X=str(5)	X=?	True	False	5	five	C	STR
24	X=(16%6)/2	X=?	0	1	2	3	C	INT
25	X=3.0**2	X=?	9	6	1.5	1	A	FLOAT
26	X=”False”	X=?	0	1	True	False	D	STR
27	X= 6!=5	X=?	6	5	True	False	C	BOOL
28	X= “5”+5	X=?	10	55	5	55555	ERROR	ERROR
29	X=True Y=False X=Y	X=?	True	False	1	0	B	BOOL
30	X=round(5.5)	X=?	5	6	5.1	5.2	B	FLOAT
31	X=int(5.5)	X=?	5	6	5.5	5.6	A	INT
32	6=X 6.0=Y X==Y=Z	Z=?	6	6.0	True	False	ERROR	ERROR
33	X=abs(-7.5)	X=?	-7.5	-8	7.5	8	C	FLOAT
34	X=5.0/2	X=?	2	1	0.5	2.5	D	FLOAT
35	X=3 Y=4 X=Y Y=X	Y=?	3	4	True	False	B	INT
36	X=3 Y=4 X=Y-7/X	X=?	2	3	4	5	A	INT
37	X=float(3)	X=?	3	2	1	0	A	FLOAT
38	X=”a” Y=”” Z=”b” W=X+Y+Z	W=?	a b	a	ba	ab	D	STR
39	X=”a” Y=str(5) Z=X+Y	Z=?	5a	a5	a	5	B	STR
40	X=5./2	X=?	2	1	2.5	0.5	C	FLOAT
41	X=int(“a”)	X=?	97	a	1	0	ERROR	ERROR
42	X=4.5 int(X)	X=?	4	4.5	5	3	B	FLOAT
43	X= (1e30- 1e30)+1 Y= 1e30- (1e30+1) Z=X==Y	Z=?	1	0	True	False	D	BOOL
44	X=int(5.6)/2	X=?	1.8	3	2	1	C	INT
55	X=float(1/4)	X=?	0	0.25	1	4	A	FLOAT

SOLUTIONS FOR LEVEL 2

Variable		Aileron	Elevator	Rudder	Text	Break
Panel #		**1**	**4**	**4**	**3**	**2**
		int	**int**	**float/int**	**string**	**bool**
Program 1	**Panel #**	0	0			TRUE
Aileron= 9%2	1	1				
Break= "a"<"A"	2					FALSE
Text= "CAD"	3				CAD	
Elevator=2**2	4		4			
Elevator=3/0	ERROR					
Aileron= Aileron+3	1	4				
Break= (5%2)!=2	2					TRUE
Text= Text+"B"	3				CADB	
Rudder= float(5/2)	4			2.0		
Aileron= 0**0	1	1				
Break= 3L==3	2					TRUE
Text=Text[:2]	3				CA	
Elevator= int(5.6)	4		5			
2=Aileron	ERROR					
Aileron= 37%4+4	1	5				
Break= (5/2)>2	2					FALSE
Text="DAB"*2	3				DABDAB	
Rudder=Elevator/2	4			2		
Aileron=len("CAD")	1	3				
Break="BCAD"[1:2]=="C"	2					TRUE
Text=Text[1]	3				A	
Rudder= Elevator+12.0	4			17.0		
Text[0]="C"	ERROR					
Aileron=int(9.0/2)	1	4				
Break=len("A B")==2	2					FALSE
Text=Text[4:3]	3				EMPTY	
Elevator=len("0123")	4		4			
Text="DCA"[3]	ERROR					

		Flaps	Slats	Hover	Message	Horn
Panel #		**2**	**1**	**1**	**4**	**3**
		int	**int**	**float/int**	**string**	**bool**
Program 2	**Panel #**	0	0			TRUE
Message="DAB"	4				DAB	
Horn="B">"b"	3					FALSE
Flaps=len("AB")	2	2				
Slats=2**2	1		4			
Message=Message+"C"	4				DABC	
Horn=(4%2)!=2	3					TRUE
Flaps=Flaps+3	2	5				
Hover= float(9/2)	1			4.0		
Slats=5/0	ERROR					
Message=Message[1]	4				A	
Horn= 4L==4	3					TRUE
Flaps=0**0	2	1				
Slats=len("012")	1		3			
Message[1]="B"	ERROR					
Message="CA"*3	4				CACACA	
Horn=(9/2)>4	3					FALSE
Flaps= 24%5-2	2	2				
Hover= Slats/2	1			1		
3=Flaps	ERROR					
Message=Message[:5]	4				CACAC	
Horn="CDAB"[1:2]=="D"	3					TRUE
Flaps= int(8.0/3)	2	2				
Hover= Slats+4.0	1			7.0		
Message="BDA"[3]	ERROR					
Message=Message[2:1]	4				EMPTY	
Horn=len("B C")==2	3					FALSE
Flaps=8%4	2	0				
Slats=int(2.7)	1		2			

Variable		Torque	Pitch	Yoke	Speaker	Light
Panel #		3	2	2	1	4
		int	int	float/int	string	bool
Program 3	Panel #	0	0			TRUE
Speaker="BAC"	1				BAC	
Pitch=2**2	2		4			
Light="C">"c"	4					FALSE
Torque= 5%3	3	2				
Speaker[2]="D"	ERROR					
Speaker= Speaker+"D"	1				BACD	
Yoke=float(7/2)	2			3.0		
Light=(9%2)!=4	4					TRUE
Torque=len("ABC")	3	3				
Speaker=Speaker[2]	1				C	
Pitch=int(5.9)	2		5			
Light= 2L==2	4					TRUE
Torque= 0**0	3	1				
Pitch=4/0	ERROR					
Speaker=Speaker[1:0]	1				EMPTY	
Yoke=Pitch/2	2			2		
Light=(7/2)>3	4					FALSE
Torque= 26%3+2	3	4				
Speaker= "CD"*2	1				CDCD	
Yoke=Pitch+6.0	2			11.0		
Light="BADC"[1:2]=="A"	4					TRUE
Torque=int(10.0/4)	3	2				
2=Torque	ERROR					
Speaker=Speaker[:3]	1				CDC	
Pitch=len("0123")	2		4			
Light=len("D C")==2	4					FALSE
Torque= Torque+1	3	3				
Speaker="DCC"[3]	ERROR					

Variable		Roll	Yaw	Altitude	Feedback	Lock
Panel #		4	3	3	2	1
		int	int	float/int	string	bool
Program 4	Panel #	0	0			TRUE
4=Roll	ERROR					
Yaw=len("01234")	3		5			
Roll=7%3	4	1				
Feedback="DCA"	2				DCA	
Lock="d"<"D"	1					FALSE
Altitude=float(11/2)	3			5.0		
Roll=Roll+4	4	5				
Feedback=Feedback+"C"	2				DCAC	
Lock=(7%2)!=3	1					TRUE
Feedback[3]="B"	ERROR					
Yaw=int(3.6)	3		3			
Roll=0**0	4	1				
Feedback=Feedback[:3]	2				DCA	
Lock=len("B C")==2	1					FALSE
Feedback="BDDA"[4]	ERROR					
Altitude=Yaw/2	3			1		
Roll=35%3+2	4	4				
Feedback=Feedback[2:0]	2				EMPTY	
Lock= 1L==1	1					TRUE
Altitude=Yaw+6.0	3			9.0		
Roll=int(6.0/4)	4	1				
Feedback="BAD"*2	2				BADBAD	
Lock="BDCA"[1:2]=="D"	1					TRUE
Yaw=3/0	ERROR					
Yaw=2**2	3		4			
Roll=len("CDA")	4	3				
Feedback=Feedback[2]	2				D	
Lock=(11/2)>5	1					FALSE

SOLUTIONS FOR LEVEL 3

text1.txt
A
B
C

Variable		Aileron	Elevator	Rudder	Text	Break
Panel #		1	4	4	3	2
		int_list	str_list	str_list	string	int
Program 1	Panel #					
Aileron=range(4)	1	[0,1,2,3]				
Break=len(range(3))	2					3
myfile=open("text1.txt","r") Text=myfile.read() myfile.close()	3				A\nB\nC	
Elevator="ABABA".split("B")	4		["A","A","A"]			
Break=[2,3,4].index(1)	ERROR					
Aileron=range(2,5)	1	[2,3,4]				
Break=["A",2,"C"].index("A")	2					0
Text=["A",["B","C"]][1][0]	3				B	
Rudder=Elevator+["B"]	4			["A","A","A","B"]		
Aileron=range(0,6,2)	1	[0,2,4]				
Break=len(["A","B",[4,5]])	2					3
myfile=open("text1.txt","r") Text=myfile.readline() myfile.close()	3				A\n	
Elevator="A B C".split()	4		["A","B","C"]			
Text[0]="B"	ERROR					
Aileron=range(3,0)	1	[]				
Break=len(range(1,4))	2					3
myfile=open("text1.txt","r") Text=myfile.readline() Text=myfile.readline() myfile.close()	3				B\n	
Rudder=Elevator	4		["A","B","A"]	["A","B","A"]		

Rudder[2]="A" #Hint:Two controls changed!					
Aileron=range(4,0,-1)	1	[4,3,2,1]			
Break=len(range(4,0))	2				0
myfile=open("text1.txt","r") Text=myfile.read() Text=myfile.read() myfile.close()	3			""	
myfile=open("text1.txt","r") Rudder=myfile.readlines() myfile.close()	4		["A\n","B\n","C"]		
Aileron[4]=5	ERROR				
Aileron[-1]=4	1	[4,3,2,4]			
Break=len(["BC","A","C"][0])	2				2
Text="".join(["A","B"])	3			AB	
Elevator="BCBC".split("C")	4	["B","B",""]			

text2.txt
B
C
A

Variable		Flaps	Slats	Hover	Message	Horn
Panel #		2	1	1	4	3
		int_list	str_list	str_list	string	int
Program 1	**Panel #**					
myfile=open("text2.txt","r") Message=myfile.readline() myfile.close()	4				B\n	
Horn=len(range(5))	3					5
Flaps=range(1,4)	2	[1,2,3]				
Slats="B C A".split()	1		["B","C","A"]			
myfile=open("text2.txt","r") Message=myfile.readline() Message=myfile.readline() myfile.close()	4				C\n	
Horn=["B",3,"A"].index("A")	3					2
Flaps=range(4)	2	[0,1,2,3]				
Hover=Slats+["A"]	1			["B","C"."A","A"]		
Flaps[4]=2	ERROR					
myfile=open("text2.txt","r") Message=myfile.read() myfile.close()	4				B\nC\nA	
Horn=len(range(2,6))	3					4
Flaps=range(2,0)	2	[]				
Slats="CBCBC".split("B")	1		["C","C","C"]			
Message=["C",["A","B"]][1][0]	4				A	
Horn=len(["A","C",[3,1,],"B"])	3					4
Flaps=range(0,5,2)	2	[0,2,4]				
myfile=open("text2.txt","r") Hover=myfile.readlines() myfile.close()	1			["B\n","C\n","A"]		
Horn=[1,3,5].index(2)	ERROR					
myfile=open("text2.txt","r") Message=myfile.read() Message=myfile.read() myfile.close()	4				“”	
Horn=len(["A","CA","ACD"][1])	3					2
Flaps=range(3,0,-1)	2	[3,2,1]				
Slats="ACAC".split("C")	1		["A","A",""]			

Message="".join(["C","A"])	4			CA	
Horn=len(range(3,1))	3				0
Flaps[-1]=4	2	[3,2,4]			
Hover=Slats Hover[2]="A" HASHTAGHint:Two controls changed!	1		["A","A","A"]	["A","A","A"]	
Message[1]="B"	ERROR				

text3.txt
C
A
B

Variable		Torque	Pitch	Yoke	Speaker	Light
Panel #		3	2	2	1	4
		int_list	str_list	str_list	string	int
Program 3	**Panel #**					
myfile=open("text3.txt","r") Speaker=myfile.read() myfile.close()	1				C\nA\nB	
Pitch"ACACA".split("C")	2		["A","A","A"]			
Light=["B",1,"C"].index("C")	4					2
Torque=range(3)	3	[0,1,2]				
Light=[1,4,3].index(2)	ERROR					
myfile=open("text3.txt","r") Speaker=myfile.readline() myfile.close()	1				C\n	
Yoke=Pitch Yoke[2]="B" HASHTAGHint:Two controls changed!	2		["A","A","B"]	["A","A","B"]		
Light=len(range(4))	4					4
Torque=range(2,4)	3	[2,3]				
Speaker[0]=A	ERROR					
myfile=open("text3.txt","r") Speaker=myfile.readline() Speaker=myfile.readline() myfile.close()	1				A\n	
Pitch="C B B".split()	2		["C","B","B"]			
Light=len(["A","BC","A"][1])	4					2
Torque=range(3,0)	3	[]				
Speaker="".join(["A","C"])	1				AC	
myfile=open("text3.txt","r") Yoke=myfile.readlines() myfile.close()	2			["C\n","A\n","B"]		
Light=len(range(2,3))	4					1
Torque=range(0,6,2)	3	[0,2,4]				
Speaker=["A",["C","B"]][1][0]	1				C	
Pitch="BCBC".split("C")	2		["B","B",""]			
Light=len(["B","C",[3,"B"],"A"])	4					4
Torque[-1]=3	3	[0,2,3]				

Torque[3]=2	ERROR		
myfile=open("text3.txt","r") Speaker=myfile.read() Speaker=myfile.read() myfile.close()	1		“”
Yoke=Pitch+["C"]	2	["B","B","","C"]	
Light=len(range(4,1))	4		0
Torque=range(4,0,-1)	3	[4,3,2,1]	

text4.txt
B
A
C

Variable		Roll	Yaw	Altitude	Feedback	Lock
Panel #		4	3	3	2	1
		int_list	str_list	str_list	string	int
Program 4	**Panel #**					
Lock=[1,4,5].index(2)	ERROR					
Yaw="BCBC".split("C")	3		["B","B",""]			
Roll=range(4)	4	[0,1,2,3]				
myfile=open("text4.txt","r") Feedback=myfile.readline() myfile.close()	2				B\n	
Lock=len(range(5))	1					5
Altitude=Yaw+["A"]	3			["B","B","","A"]		
Roll=range(2,3)	4	[2]				
myfile=open("text4.txt","r") Feedback=myfile.readline() Feedback=myfile.readline() myfile.close()	2				A\n	
Lock=len(range(1,5))	1					4
Yaw="A B C".split()	3		["A","B","C"]			
Roll=range(0,4,2)	4	[0,2]				
myfile=open("text4.txt","r") Feedback=myfile.read() myfile.close()	2				B\nA\nC	
Lock=len(["B","","AA"][1])	1					0
Roll[2]=1	ERROR					
Altitude=Yaw Altitude[2]="B" HASHTAGHint:Two controls changed!	3		["A","B","B"]	["A","B","B"]		
Roll=range(3,0,-1)	4	[3,2,1]				
Feedback="".join(["B","C","A"])	2				BCA	
Lock=len(["C","A",[4,"B"],"B"])	1					4
Feedback[0]="A"	ERROR					
Yaw="CBCBC".split("B")	3		["C","C","C"]			
Roll[-1]=0	4	[3,2,0]				
myfile=open("text4.txt","r") Feedback=myfile.read()	2				“”	

Feedback=myfile.read() myfile.close()			
Lock=len(range(3,0))	1		0
myfile=open("text4.txt","r") Altitude=myfile.readlines() myfile.close()	3	["B\n","A\n","C"]	
Roll=range(2,0)	4	[]	
Feedback=["C",["B","A"]][1][0]	2	B	
Lock=["A",0,"C"].index("C")	1		2

SOLUTIONS FOR LEVEL 4

text1.txt
C
A
C

Variable		Aileron	Elevator	Text	Rudder
Panel #		1	1	1	1
		int_list	str_list	string	int
Program 1	Panel #				
$Elevator=["A","B"]*2	1		["A","B","A","B"]		
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	1				0
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	1				1
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	1				2
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	TERMINATE				
$Aileron=range(3,0)	1	EMPTY			
2for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	1	[0]			
2for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	1	[0.1]			
2for Speaker in Feedback: TABHover=Slats.index(Speaker)	1	[0,1,0]			

TABAileron.append(Hover)		
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	TERMINATE	
$Text="ABAB"[1:]	1	BAB
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	TERMINATE	
$Rudder= len(range(3,1))	1	0
0for Rudder in 2: TABFlaps[Rudder]=0 TABFeedback=Pitch[Rudder]	ERROR	
$Aileron=range(5,0,-2)	1 [5,3,1]	
0for Hover in range(3,0): TABTorque.append(Hover) TABElevator[Hover]=Feedback[Hover]	TERMINATE	
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	1	0
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	1	1
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	1	0
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	1	1
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
0for Hover in range(2): TABRoll.append(Hover)	TERMINATE	

TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)			
$Rudder=len("ABCA"[2:1])	1		0
0for Speaker in Feedback: TABRudder=Rudder+1 TABMessage=Slats[0] TABYoke=Altitude+2	TERMINATE		
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	1		C\n
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	1		A\n
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	1		C
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	TERMINATE		
$Elevator=list("BCAB")	1	["B","C","A","B"]	
1for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	1	["B","C","A",""]	
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE		
1for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	1	["B","C","",""]	
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE		
0for Hover in range(1,3): TABElevator[-Hover]=""	TERMINATE		

```
TABfor Yoke in range(Hover):
TABTABFeedback="".join(Yaw[Yoke:Yoke+1])
```

text1.txt
C
A
C

Variable		Flaps	Slats	Message	Hover
Panel #		2	2	2	2
		int_list	str_list	string	int
Program 1	Panel #				
$Hover=len("ABC"[2:0])	2				0
1for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	2				1
1for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	2				2
1for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	2				3
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	TERMINATE				
$Slats=["C","A"]*2	2		["C","A","C","A"]		
1for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	2				0
1for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	2				1
1for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	2				0
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	TERMINATE				
$Flaps=range(5,0,-2)	2	[5,3,1]			
2for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	2			C	

2for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	2		A
2for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	2		C
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	TERMINATE		
$Message="CABCA"[1:4]	2		ABC
0for Rudder in 2: TABFlaps[Rudder]=0 TABFeedback=Pitch[Rudder]	ERROR		
$Slats=list(Message)	2	["A","B","C"]	
0for Hover in range(3,0): TABTorque.append(Hover) TABElevator[Hover]=Feedback[Hover]	TERMINATE		
0for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	2		0
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE		
0for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	2		1
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE		
0for Hover in range(2): TABRoll.append(Hover) TABRudder in range(2): TABYoke=Torque.index(Rudder)	TERMINATE		1
$Message="BACBA"[1:]			ACBA
2for Hover in range(2): TABRoll.append(Hover) TABRudder in range(2): TABYoke=Torque.index(Rudder)	TERMINATE		

1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	TERMINATE	
$Hover=len(range(2,0))	2	0
0for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	2	1
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE	
0for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	2	2
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE	
0for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE	1

text1.txt
C
A
C

Variable		Torque	Pitch	Speaker	Yoke
Panel #		3	3	3	3
		int_list	str_list	string	int
Program 3	**Panel #**				
$Torque=range(3,0)	3	EMPTY			
2for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	3	[0]			
2for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	3	[0,1]			
2for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	3	[0,1,2]			
TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder					
$Yoke=len("CDC"[2:0])	3				0
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	3			C	
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	3			A	
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	3			C	
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	TERMINATE				
$Pitch=["B","A"]*2	3		["B","A","B","A"]		
1for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	3			B	
1for Altitude in range(len(Text)):	3			A	

TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])		
1for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	3	B
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	TERMINATE	
$Torque=range(3,-1,-1)	3 [3,2,1,0]	
0for Rudder in 2: TABFlaps[Rudder]=0 TABFeedback=Pitch[Rudder]	ERROR	
$Speaker="CCACAC"[1:]	3	CACAC
0for Hover in range(3,0): TABTorque.append(Hover) TABElevator[Hover]=Feedback[Hover]	TERMINATE	
3for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	3	3
3for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	3	2
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
3for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	3	3
3for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	3	2
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
0for Hover in range(2): TABRoll.append(Hover)	TERMINATE	

TABRudder in range(2): TABYoke=Torque.index(Rudder)		
$Pitch=list(Speaker)	3	["C","A","C","A","C"]
2for Hover in range(2): TABRoll.append(Hover) TABRudder in range(2): TABYoke=Torque.index(Rudder)	TERMINATE	
3myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	3	["","A","A",""]
3myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	3	["C","C","C"]
3myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	3	["","A","A",""]
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	TERMINATE	
$Yoke=len(range(2,-1))	3	0
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	3	0
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE	0
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	3	0
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	3	1
2for Hover in range(1,3): TABElevator[-Hover]=""	TERMINATE	0

TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	
0for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE

text1.txt
C
A
C

Variable		Roll	Yaw	Feedback	Altitude
Panel #		4	4	4	4
		int_list	str_list	string	int
Program 4	**Panel #**				
$Roll=range(5,0,-2)	4	[5,3,1]			
3for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	4	[0,3,1]			
3for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	4	[0,1,1]			
3for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	4	[0,3,2]			
0for Rudder in range(3): TABHover=Hover+1 TABTorque.append(Rudder) TABRoll[Rudder]=Rudder	TERMINATE				
$Feedback="ACAC"[1:]	4			CAC	
0for Speaker in Feedback: TABHover=Slats.index(Speaker) TABAileron.append(Hover)	TERMINATE				
$Altitude=len(range(4,2))	4				0
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	4				0
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	4				1
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	4				2
0for Altitude in range(len(Text)): TABSpeaker=Text[Altitude] TABMessage="".join(Slats[Altitude:Altitude+1])	TERMINATE				

$Yaw=["C","B"]*2	4	["C","B","C","B"]
0for Rudder in 2: TABFlaps[Rudder]=0 TABFeedback=Pitch[Rudder]	ERROR	
$Roll=range(3,1)	4	EMPTY
0for Hover in range(3,0): TABTorque.append(Hover) TABElevator[Hover]=Feedback[Hover]	TERMINATE	
1for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	4	[0]
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
1for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	4	[0,1]
2for Hover in range(2): TABRoll.append(Hover) TABfor Rudder in range(2): TABTABYoke=Torque.index(Rudder)	TERMINATE	
0for Hover in range(2): TABRoll.append(Hover) TABRudder in range(2): TABYoke=Torque.index(Rudder)	TERMINATE	
$Feedback="ABACA"[3:1]	4	EMPTY
2for Hover in range(2): TABRoll.append(Hover) TABRudder in range(2): TABYoke=Torque.index(Rudder)	TERMINATE	
2myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	4	C
2myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	4	A
2myfile=open("test1.txt","r")	4	C

for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)			
1myfile=open("test1.txt","r") for Text in myfile: TABFeedback=Text.strip() TABPitch=Speaker.split(Feedback)	TERMINATE		
$Yaw=list("BACA")	4	["B","A","C","A"]	
3for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	4		B
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE		
3for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	4		B
3for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	4		A
2for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE		
0for Hover in range(1,3): TABElevator[-Hover]="" TABfor Yoke in range(Hover): TABTABFeedback="".join(Yaw[Yoke:Yoke+1])	TERMINATE		

Appendix B

Cheat Sheets for Space Race

CHEAT SHEET FOR LEVEL 2

OPERATORS

==	**Equality**	3==3 is True
!=	**Non-equality**	3!=3 is False
******	**Exponent**	3**2 is 9
%	**Modulus** (remainder)	5%2 is 1
/	**Division** **Integer division** gives an integer result rounded down towards negative infinity	5/2 is 2 5.0/2 is 2.5
+	**Addition/ Concatenation**	5+2.0 is 7.0 “A”+”C” is “AC”
=	**Assignment Operator** Values from right operand assigned to left operand	A=2 2=A gives SyntaxError

INTEGERS

int()	**Converts** a string or number to an integer, if possible. All digits after a decimal will be truncated for a floating point number.	int(4.5) is 4 int(“9”) is 9 int(“a”) gives Value Error

STRINGS

ASCII	ASCII codes represent **text in computers**. “A” is represented as 65. “a” is represented as 97	“A”<”a” is True
Index	Used to specify **positions of characters** within a string. Positive indexing begins at 0. Negative indexing starts at the end of the string with -1.	“ABCD”[0] is “A” “ABCD”[2] is “C” “ABCD”[-1] is “D” “ABCD”[-2] is “C”
Immutable	A string’s state **cannot be modified** after it is created.	x=”ABCD” x[0]=”D” gives TypeError
Slice Notation	Used to represent a substring (a “slice” of the string) **String[start:end]** = substring from index start to index end-1 If **start>end** then substring is always an empty string. **String[start:]** = substring from index start to end **String[:end]** = substring from start to index end-1 **String[:]** = copy of entire string	“ABCD”[1:3] is “BC” “ABCD”[1:2] is “B” “ABCD”[3:1] is “” “ABCD”[1:] is “BCD” “ABCD”[:3] is “ABC” “ABCD”[:] is “ABCD”
len()	Returns the number of characters in a string	len(“ABC”) is 3

CHEAT SHEET FOR LEVEL 3

STRINGS

.split()	Returns a **list of substrings** in a given string, **using a separator** (if specified) as the delimiter. If no delimiter is specified, any consecutive whitespace is used as the separator.	"papacy".split("a") is ["p","p","cy"] "banana".split("a") is ["b","n","n",""] "aardvark".split("a") is ["","","rdv","rk"] "a b c d".split() is ["a","b","c","d"]
"".join()	Returns a string composed of the strings in a given list	"".join(["a","b","c"]) is "abc"

FILES

Assume "testfile.txt" has the following lines:
CAT
DOG
BAT
		Assume
		myFile= open("testfile.txt","r")
		#sample code goes here
		myFile.close()

.readline()	Reads **next line** from the file as **a string**. Retains newline	myFile.readline() is "CAT\n"
.read()	Reads **entire file** until EOF (end of file) is reached as **a string**	myFile.read() is "CAT\nDOG\nBAT"
.readlines()	Reads **entire file** until EOF is reached as a **list of strings** with each string representing a line from the file	myFile.readlines() is ["CAT\n","DOG\n","BAT"]
File Pointer	The file pointer begins at the start of the file and it **specifies the current file position**. Once a line is read, the file pointer **advances** to the next line. A line cannot be re-read unless the current file position is set back to that line.	

LISTS

.index()	Returns **first index of value** in a list. Raises ValueError if the value is not present.	["A","B","B"].index("B") is 1 ["A","B","B"].index("C") gives ValueError
range()	Returns **a list** with an arithmetic progression **of integers**. Accepts as arguments [start,]stop[,step] (start and step are optional). The **stop point is not part of the generated list**! If **start>stop and step is positive or not given** then the returned list is empty.	range(3) is [0,1,2] range(1,4) is [1,2,3] range(0,8,2) is [0,2,4,6] range(4,1) is [] range(4,1,-1) is [4,3,2]
len()	Returns the **number of elements** in a list.	len([0,1,2]) is 3
Nested Lists	Lists formed with **lists as elements**. For a list within a list (one sublevel), you can index sub-list elements with the format: **[outer][inner]**.	X=[1,2,["a","b"],3] X[2][0] is "a"
Mutable	Lists **allow in-place modification** after it has been created. Can use positive or negative indexing, or slice notation, to refer to element(s) in a list.	X=["A","B","C"] X[0]="D" X is ["D","B","C"]
Aliasing	Can occur for values that are **mutable**. Happens when one variable's mutable value (for ex. list) is assigned to another variable. Both **variables** are then **referring to the same value**. When an **in-place change** is made to the value through one variable, the other variable "sees" the **same change**.	X=[1,2,3] Y=X Y[0]=4 Y is [4,2,3] X is [4,2,3]

CHEAT SHEET FOR LEVEL 4

STRINGS

.strip()	Returns string with **leading and trailing whitespaces removed**	“ \ta\n”.strip() is “a”

LISTS

.append()	Appends **object to the end** of the current list.	[“A”,”B”].append(“C”) is [“A”,”B”,”C”]
list()	Returns a list initialized with the sequence/iterable’s items. If the sequence is a string, a sequence of characters, each **element in the list will be a character** from that string.	list(“abc”) is [“a”,”b”,”c”]
*****	Provides **copies** of the list, concatenated	[“A”,”B”]*2 is [“A”,”B”,”A”,”B”]

FOR- LOOPS

for iterating_var **in** sequence**:**
statement(s)

```
>>for var_i in [1,2,3]:
        print var_i
1
2
3
>>>for var_j in “abc”:
        print var_j
a
b
c
```

iterating_var	This variable is assigned one item from **sequence** for every repetition/iteration of the loop
sequence	Examples of sequence types in Python include **strings, lists,** and **tuples**
statement(s)	This block of code is executed once for each element in **sequence**. In other words, it is executed once for every iteration of the for-loop
loop termination	The loop terminates when all the items from **sequence** have been exhausted. That is, there are no more elements in **sequence** to assign to the **iterating_var**. If **sequence** is empty, the **statement(s)** are never executed as there are no elements to assign to **iterating_var**.

nested for-loop

```
>>>for var_i in [“outer1”,”outer2”]:
        for var_j in [“inner 1”,”inner2”]:
            print var_i, var_j
outer1 inner1
outer1 inner2
outer2 inner1
outer2 inner2
```

A nested for-loop is a **loop within a loop**. For every iteration of the outer for-loop, the inner for-loop must execute to completion.

FILES

Assume “testfile.txt” has the following lines:
CAT
DOG
BAT

for loop

A for loop can be used to **read a line** from the file **for every iteration** of the loop. Think of the file as a **sequence of lines**.

```
>>>myFile=open(“testfile.txt”,”r”)
>>>x=[]
>>>for var_i in myFile:
        x.append(var_i)
>>myfile.close()

x is [“CAT\n”,”DOG\n”,”BAT”]
```

Appendix C

Pre and Post Quizzes

***Note: all correct answers are indicated with a tilde (~)**

LEVEL 1

1. **In Python, what is the value of expression float(1/2)?**
    a) 0
    b) 0.0~
    c) 0.5
    d) 1
    e) 1.0
2. **There exists a positive (nonzero) floating point number e such that 1+e=1. Is this statement true or false?**
    a) true~
    b) false
3. **In Python, for integers m and n, the type of the expression m/n will be float when m%n!=0. Is this statement true or false?**
    a) true
    b) false~
4. **Concatenation of strings in Python is commutative. That is, for strings x and y: x+y=y+x. Is this statement true or false?**
    a) true
    b) false~
5. **Which of these generates a runtime error?**
    a) "b"*4
    b) X=6!=5
    c) X=str(5)
    d) X=int("b")~
    e) X=4L/2

LEVEL 2

1. **In Python, x=3 and 3=x have the same meaning. Is this statement true or false?**
    a) true
    b) false~
2. **After running the following lines of code:**
   X=2
   float(X)
   the value of X is 2.0. Is this statement true or false?
    a) true
    b) false~
3. **Assuming variable X is a non-empty string of length 3. Which of these generates a runtime error?**
    a) X[0]="E"~

b) Y= X[1:2]=="C"
c) Y= X[:2]
d) Y= X[2:0]

4. **If you double the memory of your computer, what effect does this have on the number of ASCII values that can be represented?**
    a) Doubles the number of ASCII values available
    b) Allows including both capital and lowercase letters
    c) There is an increase, but the specific results depend on the particular architecture of your computer
    d) All ASCII values will become unicode values
    e) There is no change in the number of ASCII values~
5. **In Python, which kind of value is not mutable?**
    a) Strings~
    b) Lists
    c) Objects
    d) None of the above

LEVEL 3

1. **The elements of a list can have any type, as long as the elements are not of the list type; that is, lists of lists are not possible. Is this statement true or false?**
    a) true
    b) false~
2. **Suppose the file named LOTR contains the following two lines of ASCII characters:**
   One Ring to rule them all,
   One Ring to find them.
   After running the following lines of code:
   f = open (" LOTR " ," r")
   x = f . readline ()
   y = f . readline ()
   f . close ()
   what will be the value of y?
    a) "One Ring to rule them all,\n"
    b) "One Ring to find them.\n"~
    c) "One Ring to rule them all,\nOne Ring to find them.\n"
    d) ""
3. **After running the following lines of code:**
   x=[1,2,3]
   y=x
   y[0]=4
   the value of x is [1,2,3]. Is this statement true or false?
    a) true

b) false~

4. **If x= range(3,0), the value of x is [3,2,1,0]. Is this statement true or false?**
   a) true
   b) false~
5. **If x="BCBC".split("C"), what is the value of x?**
   a) ["B","B"]
   b) ["C","C"]
   c) ["B","B",""]~
   d) ["","C","C"]

LEVEL 4

1. **In Python, what is the value of the expression ["a", "b"]*2?**
   a) ["aa","bb"]
   b) ["a2","b2"]
   c) ["a","b","a","b"]~
   d) ["a","a","b","b"]
2. **For m>=0 and n>=0 the value of num is the same after executing**
   num = 0
   for i in range (n):
   num = num + 3
   as it is after executing
   num = 0
   for j in range (m , m + n) :
   num = num + 3
   Is this statement true or false?
   a) true~
   b) false
3. **If a is an expression of type long, and m is an integer with m>=0, after the following lines of code are executed:**
   x = 1
   for i in range (m):
   x = x * a
   what is the value of x?
   a) a + m
   b) a * m
   c) a * a
   d) a ** m~
4. **After executing the following lines of code:**
   for i in range(2):
   for j in range(3):
   print "Hello World"
   How many times is Hello World printed?

a) 2
b) 3
c) 6~
d) 8

5. What is the value of x if x=list("ABC")?

a) ["ABC"]
b) ["A","B","C"]~
c) []
d) undefined because of TypeError

Appendix D

Experimental Surveys

Survey A

Please note that question order may change

1. Prior to taking ENG 1D04- Engineering Computation, did you have programming experience?
    a. Yes
    b. No

2. How many courses (high school, college, extracurricular courses) have you taken that have involved computer programming (excluding ENG 1D04)?
    a. None
    b. 1
    c. 2 -3
    d. >3

3. How many lines of code did you write in your single largest simple program? (Leave blank if you have not written a program before)

4. Programming languages you have used prior to taking ENG 1D04 (you may select more than one):
    a. None
    b. C
    c. C++
    d. Fortran
    e. Java
    f. Matlab
    g. Python
    h. Other
    i. If other, please specify language(s):

5. I play video games
    a. Daily
    b. Weekly
    c. Monthly
    d. Yearly
    e. Never

6. Each video gaming session lasts _____ hours (leave blank if you do not play video games)

7. Have you ever used an educational video game as part of a course?
    f. Yes
    g. No

8. If you answered yes to the previous question, please describe your experience below. Include game title (if not known, provide a brief description), approximate grade, and subject used in. Feel free to add any other relevant details.

9. If you have used an educational video game, to what extent did you find it beneficial to your overall learning experience? (Select N/A if you have never used an educational video game)
    h. Not beneficial at all
    i. Somewhat beneficial
    j. Beneficial
    k. Very Beneficial
    l. N/A

*****The following questions will have these options: strongly agree, agree, neither agree nor disagree (neutral), disagree, strongly disagree*****

10. I consider myself to be an avid gamer
11. I play video games for fun
12. I would enjoy playing video games to learn
13. Excitement is the most important part a good video game
14. Interactivity is the most important feature of a good video game
15. Story/ Plot is the most important feature of a good video game
16. Variety is the most important feature of a good video game
17. Graphics is the most important feature of a good video game
18. Competition between players the most important feature of a good video game
19. Video games can be powerful teaching tools
20. Video games can teach students how to set and reach goals
21. Video games can motivate students to learn
22. Video games can motivate students to seek out additional information that will help them succeed
23. Video games can stimulate a student's curiosity on a topic

24. Video games can provide a safe way to test new ideas or try new techniques
25. Video games can be used to complement course objectives
26. Video games can provide immediate feedback to students
27. Video games can be an effective tool to teach basic programming concepts
28. Video games are a waste of time
29. Video games should only be used as a leisure activity
30. Educational games require too much special equipment and technical support to be beneficial
31. The use of educational video games does not belong in the post- secondary classroom
32. In general, students will not take educational video games seriously
33. Students cannot learn well through video games
34. In general, males will enjoy using educational video games in classes more than females will
35. In general, females will benefit from using educational video games as much as males

Survey B- Gameplay Feedback Survey

Please note that question order may change
******The following questions will have these options: strongly agree, agree, neither agree nor disagree (neutral), disagree, strongly disagree******

Please help us rate the video game by answering the following questions:

1. The video game was easy to play
2. The video game required too much technical support to be beneficial
3. It was difficult to learn how to play the video game
4. I enjoyed playing the video game
5. The video game was too slow to play efficiently
6. The video game helped me to learn the basic programming concepts being presented
7. I felt confused trying to understand the basic programming concepts with this video game
8. I would recommend that others try to learn the basic programming concepts with this video game
9. My team members helped me to better understand the basic programming concepts being presented
10. I enjoyed cooperating with team members to successfully complete the game level
11. Video games, in general, are an effective tool to teach basic programming concepts
12. I feel confident that I understand the basic programming concepts presented in the video game
13. I felt reassured of my knowledge in the programming concepts presented when I gave verbal cues to my team members in the game
14. I felt reassured of my knowledge in the programming concepts presented when I received verbal cues from my team members in the game
15. I would prefer to learn basic programming concepts through a video game as opposed to other traditional teaching methods such as lecture, tutorials, or textbooks.
16. The video game stimulated my curiosity on the basic programming concept being presented
17. The video game has motivated me to review the course material

18. Cooperation between team members makes this video game an effective teaching tool for learning basic programming concepts

19. Competition between teams makes this video game an effective teaching tool for learning basic programming concepts

20. Competition between teams motivates me to review course material

21. My role as an effective team member motivates me to review course material

22. The video game was not effective in teaching basic programming concepts because it was not challenging enough

23. The video game was not effective in teaching basic programming concepts because it was too challenging

24. I wish there was more gameplay time in the video game

The following question is in short answer form.

25. What were some features that you did or did not like about the video game?

26. How did your team members affect your ability to learn the basic programming concepts being presented?

27. What are some improvements you would incorporate into the video game?

28. Is there anything important that I forgot or something you think would help me understand the usability of the game in learning basic computer programming concepts?

Appendix E

Exam Questions

E.1 Midterm 1

Question 1
Let the variables `x` and `y` be bound to values of type float. If `x + y == x` evaluates to `True`, then the value of `y` must be `0.0`. Is this statement true or false?

1. True.

2. False. **(correct answer)**

Question 5
For all expressions `a` and `b` of type `int` with `b` not equal to 0, `a / b` is an expression of type `int`. Is this statement true or false?

1. True. **(correct answer)**

2. False.

Question 9
In Python, = and == both mean *equal*. Is this statement true or false?

1. True.

2. False. **(correct answer)**

Question 10
When evaluated, both of the following code fragments print the same sequence of numbers:

```
# Code Fragment 1
x = 0
for i in range(5):
  x = x + i + 1
  print x

# Code Fragment 2
for i in range(1,6):
  x = i
  for j in range(i):
    x = x + j
  print x
```

Is this statement true or false?

1. True. **(correct answer)**
2. False.

Question 13
After the following code is executed, how are the contents of the files named `iceland` and `denmark` related?

```
f = open("iceland","r")
x = f.read()
f.close()
g = open("denmark","w")
y = ""
for i in x:
  y = i + y
g.write(y)
g.close()
```

1. The contents of `iceland` and `denmark` are identical.
2. The contents of `iceland` and `denmark` are identical except that the sequence of lines in `iceland` are reversed in `denmark`. That is, the first line in `iceland` is the last line in `denmark`.
3. The contents of `iceland` and `denmark` are identical except that the sequence of characters in `iceland` are reversed in `denmark`. That is, the first character in `iceland` is the last character in `denmark`. **(correct answer)**
4. The contents of `iceland` and `denmark` are identical except that the characters in each line in `iceland` are reversed in `denmark`.

Question 17
When executed, which of the following code fragments print the numeral 5?

```
# Code Fragment 1
sum = 1
for i in range(5):
  sum = sum + 1
print sum

# Code Fragment 2
sum = 1
for i in "00" * 2:
  sum = sum + 1
print sum

# Code Fragment 3
sum = 1
for i in [2,2,2,2]:
  sum = sum + 1
print sum
```

1. Fragments 1 and 2.
2. Fragments 1 and 3.
3. Fragments 2 and 3. **(correct answer)**
4. Fragments 1, 2, and 3.

Question 20
A for loop can be used to compute a summation like

$$\sum_{i=m}^{n} a_i.$$

Which for loop computes the value of

$$\sum_{i=2}^{1} i.$$

1. ```
   num = 0
   for i in range(2):
     num = num + i
   ```

2. ```
   num = 0
   for i in range(1,2):
     num = num + i
   ```

3. ```
   num = 0
   for i in range(2,2):
     num = num + i
   ```

   (correct answer)

4. ```
   num = 1
   for i in range(2,1,-1):
     num = num + i
   ```

Question 23
If `s` is of type `str` and `len(s) > 1` evaluates to `True`, what is returned when `type(s[0])` is evaluated?

1. `<type 'char'>`.
2. `<type 'str'>`. **(correct answer)**
3. `<type 'int'>`.
4. `<type 'list'>`.

Question 27

Let `a` be an expression of type `str` and `b` be an expression of type `int`. Which piece of code computes `a * b`?

1. ```
   x = ""
   for i in range(b):
     x = x + a
   ```

   (correct answer)

2. ```
   x = ""
   for i in range(b):
     x = x * a
   ```

3. ```
   x = a
   for i in range(b):
     x = x + a
   ```

4. ```
   x = ""
   for i in range(a):
     x = x + b
   ```

Question 28

```
a = [0,0,0]
b = [0,0,0]
c = b
a[0] = 1
b[1] = 2
c[2] = 3
```

What are the values of `a`, `b`, and `c`?

1. `[1,0,0]`, `[0,2,0]`, and `[0,0,3]`.
2. `[1,0,0]`, `[0,2,0]`, and `[0,2,3]`.
3. `[1,0,0]`, `[0,2,3]`, and `[0,2,3]`. **(correct answer)**
4. `[1,2,3]`, `[0,2,3]`, and `[0,2,3]`.

E.2 Midterm 2

Question 5
If two real numbers are exactly representable as floating point numbers, then the result of a arithmetic operation on them will also be exactly representable as a floating point number. Is this statement true or false?

1. True.

2. False. **(correct answer)**

Question 27
A for loop can be used to compute a summation like

$$\sum_{i=m}^{n} a_i.$$

Which for loop computes the value of

$$\sum_{i=2}^{1} i.$$

1. ```
   num = 0
   for i in range(2):
     num = num + i
   ```

2. ```
   num = 0
   for i in range(1,2):
     num = num + i
   ```

3. ```
   num = 0
   for i in range(2,2):
     num = num + i
   ```

   (correct answer)

4. ```
   num = 1
   for i in range(2,1,-1):
     num = num + i
   ```

E.3 Final Exam

Question 4
Recall that `(x + y) + z = x + (y + z)` is the law of associativity for the operator `+`. For which Python type does the law of associativity for the operator `+` fail?

1. `int`.
2. `long`.
3. `float`. **(correct answer)**
4. `str`.

Question 5
If `x = "BCBC".split("C")`, what is the value of `x`?

1. `["B","B"]`.
2. `["C","C"]`.
3. `["B","B",""]`. **(correct answer)**
4. `["","C","C"]`.

Question 6
Assuming variable `X` is a nonempty `string` of length 3, which of these generates a runtime error?

1. `X[0] = "E"`. **(correct answer)**
2. `Y = X[1:2] == "C"`.
3. `Y = X[:2]`.
4. `Y = X[2:0]`.

Question 7
Which of these generates a runtime error?

1. `X = "b" * 4`
2. `X = 6 != 5`
3. `X = str(5)`
4. `X = int("b")` **(correct answer)**
5. `X = 4L / 2`

Question 11
In Python, what is the value of the expression `2 * [1]`?

1. `[1]`.
2. `[1,1]`. **(correct answer)**
3. `[2]`.
4. `[2,2]`.

Question 16
When a voltage (battery) is connected to a simple circuit containing a resistor and a capacitor, the current in the circuit increases as an exponential function of time. The following program is intended to calculate the current I in a simple circuit at a given time, with resistance doubled. The formula for calculating current in this circuit is:

$$I = \frac{V}{2R}\left(1 - e^{\frac{-t}{2RC}}\right)$$

where the parameters for this calculation are the voltage supplied V, the resistance R, the capacitance C, and the time t.

```
import math
V = input("Enter a value for the voltage, V: ")
R = input("Enter a value for the resistance, R: ")
C = input("Enter a value for the capacitance, C: ")
t = input("Enter a value for the elapsed time, t: ")
I = %*\textit{E}*)
print "The current for V, R, C, and t =", I
```

What should be the value of expression *E* for this code to work as expected?

1. `V / (2 * R) * ( 1 - math.exp( - t / ( 2 * R * C) ))`
2. `V / (2.0 * R) * ( 1 - math.exp( - t / ( 2 * R * C) ))`
3. `V / (2.0 * R) * ( 1 - math.exp( - (1.0 * t) / ( 2 * R * C) ))` **(correct answer)**
4. All of the above.

Question 17
In Python, arithmetic involving values of type _____ may yield mathematically incorrect results.

1. `int`.
2. `long`.
3. `float`. **(correct answer)**
4. `str`.

Question 21
If `a` and `b` are expressions of type `list`, what is the value of `x` after the following code is executed?

```
x = []
for i in a:
  x = x + b
```

1. `a + b`.
2. `a * b`.
3. `len(a) * b`. **(correct answer)**
4. `a ** b`.

Question 23
After the following code is executed how are the contents of the files named `rudolph` and `albert` related?

```
f = open("rudolph","r")
x = f.read()
f.close()
g = open("albert","w")
c = 0
for y in x:
  c += 1
  if y == '\n':
    g.write(str(c) + '\n')
    c = 0
g.close()
```

1. The contents of `rudolph` and `albert` are identical.
2. The contents of `rudolph` and `albert` are identical except that the newline characters in `rudolph` have been removed in `albert`.
3. `albert` is obtained from `rudolph` by replacing each line in `rudolph` with the number of characters in that line. **(correct answer)**
4. `albert` holds the number of characters, the number of words, and the number of lines in `rudolph`.

Question 29
A for loop can be used to compute a summation like

$$\sum_{i=m}^{n} a_i.$$

Which for loop computes the value of

$$\sum_{i=2}^{1} i.$$

1. ```
   num = 0
   for i in range(2):
     num = num + i
   ```

2. ```
   num = 0
   for i in range(1,2):
     num = num + i
   ```

3. ```
   num = 0
   for i in range(2,2):
     num = num + i
   ```

   (correct answer)

4. ```
   num = 1
   for i in range(2,1,-1):
     num = num + i
   ```

Question 44
What is the value of $\sum_{i=1}^{2} \left(\prod_{j=3}^{1} ij \right)$?

1. 2. **(correct answer)**
2. 3.
3. 18.
4. 54.

Question 46
Which of the following types in Python represents rational numbers in scientific notation?

1. `int`.
2. `long`.
3. `float`. **(correct answer)**
4. `complex`.