Computer Science Education: A Game to Teach Children about Programming

by

Xiaoxiao Wang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2017 by the
Graduate Supervisory Committee:

Brian Nelson, Chair
Pavan Turaga
Erin Walker

ARIZONA STATE UNIVERSITY

May 2017

ABSTRACT

Computational thinking, the fundamental way of thinking in computer science, including information sourcing and problem solving behind programming, is considered vital to children who live in a digital era. Most of current educational games designed to teach children about coding either rely on external curricular materials or are too complicated to work well with young children. In this thesis project, Guardy, an iOS tower defense game, was developed to help children over 8 years old learn about and practice using basic concepts in programming. The game is built with the SpriteKit, a graphics rendering and animation infrastructure in Apple's integrated development environment Xcode. It simplifies switching among different game scenes and animating game sprites in the development. In a typical game, a sequence of operations is arranged by players to destroy incoming enemy minions. Basic coding concepts like looping, sequencing, conditionals, and classification are integrated in different levels. In later levels, players are required to type in commands and put them in an order to keep playing the game. To reduce the difficulty of the usability testing, a method combining questionnaires and observation was conducted with two groups of college students who either have no programming experience or are familiar with coding. The results show that Guardy has the potential to help children learn programming and practice computational thinking.

DEDICATION

To my parents and friends.

ACKNOWLEDGMENTS

First of all, I would like to thank my thesis advisor Dr. Brian Nelson for the continuous support and patience. His knowledge and experience in the field have inspired me a lot in this study. Then, I would like to express my gratitude to my committee members Dr. Pavan Turaga and Dr. Erin Walker for meetings and suggestions to improve the quality of this work. Special thanks to my friends who participated in the usability testing of the game.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Overview

Due to the huge demand for it in our society, computer science has become a vital part of science, technology, engineering, and mathematics (STEM) learning [1]. Computational thinking (CT), the fundamental way of thinking and problem solving in computer science, requires thinking at multiple levels of abstraction such as consideration of available resources and efficiency [2]. It involves the process of understanding and analyzing problems, heuristic reasoning, designing systems, solving problems, and evaluating solutions by drawing on the concepts fundamental to computer science [2]. Because CT is ingrained in everyday lives, it becomes critical and beneficial to everyone, not just computer scientists [2]. Since first proposed by Jeanette Wing in 2006, CT education, especially for K-12 students, has been studied by a growing community of educators and computer science experts [3]. Based on their research, it is wildly accepted that learning and practicing programming is a better approach to learn CT than only completing introductory activities [3]. The problem-solving, information-sourcing, project management and algorithms in programming are significant to introducing CT to children [4]. Furthermore, programming offers a chance for students to express themselves and participate in social networks [5]. Through the process of computer programs being created, used, repurposed and shared, student programmers increase their capacity to participate in today's digital society [5].

Because of the growing attention paid to CT, more and more students are taking computer science AP courses in high school [6]. However, whether programming courses should be added as a requirement for all students in the already packed K-12 curricula remains a question [3]. While computer science education for undergraduates and above has matured, for K-12 students it is still in the early stages of development in the US [7]. Even though a group of movements [8] have been focusing on teaching programming to children, it still lacks the strategy and solid research foundation necessary to bring CT to all students [7].

In such a situation, teaching programming through games seems to be a feasible approach. Games have a set of rules in the virtual worlds and give the player an identity or a role to play in these worlds [9]. At some point, games become simulations that easily involve players and teach them the information they need to know. The capability of games expands the possibility of education in various ways due to the limitlessness of digital virtual worlds. Furthermore, games are considered as an immensely entertaining and attractive interactive technology that helps form affinity groups among players [10]. Besides the educational potential it has, teaching programming through games does not require any instant change in the current K-12 education system since it can be conducted outside 'normal' school curricula.

While the outcomes of game-based education and whether it is appropriate for learning programming remain controversial in our society [11], some coding-based games have been developed for K-12 students. However, the influence these games have on children's after-school life and their understanding of programming still needs to be studied.

1.2 Objective

The objective of this thesis is to design and implement an iOS tower defense game to teach children about programming. The game is designed to introduce programming basics to chidren aged 8 or above as well as elder students without coding experience. The focus of game relies on general conceptions and principles in coding instead of teaching a specific programming language.

1.3 Challenges

Several challenges to be addressed to build such a game are discussed below.

1.3.1 The Genre of Game

Most of coding games described in the literature were developed based on early programming languages such as LOGO [12]. Due to this fact, the contents of these games were designed to have students use a sequence of directions to move a game character [12]. Conversely, the genre of the game developed for this thesis is defined as tower defense. Teaching students how to integrate coding into this specific kind of game genre becomes a challenge.

1.3.2 The Playability for Children

The main purpose of teaching through games is to get children more involved in learning than they are through the current education system. Played in after-school time, this game should be designed to be engaging enough to encourage children to play it. The visual style, level settings, mission progress and affinity groups need to be taken into consideration.

1.3.3 Typing in the Game

Typing code and interacting with the computer are vital in real programming experience. The balance between integrating actual typing of code into the game to support learning, and avoiding so much text entry that the game becomes less fun becomes another challenge.

1.4 Contributions

This thesis draws from different knowledge bases. It involves study in game design, programming instruction and visual design work. This thesis is considered as an attempt to experiment with approaches to helping children learn CT. It will be a beneficial example for reaching out to kids with computer science.

CHAPTER TWO

RELATED WORKS

2.1 The Foos

The Foos is a platform designed by codeSpark to help children make their own games and learn to code through playing them [13] and its target players are kids at any age above four. In a typical game scene, the player is required to drag various command modules from the bottom toolbar. By arranging commands in a certain sequence and running them, the player can make the character move around to collect gems and eventually arrive at the target destination for a given level. The basic process of finishing a level involves analyzing the requirements, coming up with a solution, sequencing the command modules to implement the solution, running the sequence to test the result, and fixing the problems happen in the running time. The modules of the game include movements in four basic directions, jumping, looping, and other functional commands such as throwing a firecracker to clear obstacles in the way.

Figure 2.1 A Typical Game Scene in The Foos

The Foos has set a good example for finding a balance between a fun game and a game that teaches both CT and programming concepts. The process of a game is a simple simulation of writing a program without typing code since CT is well integrated in it. Command modules that the player uses through the game provide an introduction to variables and functions in coding. At the same time, the level setting, cartoon style, and no-text tutorials of The Foos make it approachable and entertaining for children. Besides the core gameplay elements, The Foos also provides game modes through which the player can build custom game levels and share them with others. This platform not only enables children to get involved in social networks under a free environment of interactions, but also offers them a chance to think like a game designer and programmer.

While The Foos has done a good job in building an easy-to-play game about coding, the real teaching relies on its corresponding curriculum [14]. In its materials, a formal classroom setting is required for activities like group discussions to conduct the teaching [14]. In such case, whether The Foos curriculum can be added into current education system and how much the game itself helps children learn programming still need to be explored in studies.

2.2 Scratch

Scratch is a simple visual programming language developed by MIT Media Lab. It aims to help children easily create and share their own interactive digital projects such as video games, animations, newsletters [15]. With different shaped programming blocks in Scratch's library, children are able to play with materials like sounds and images to build their own works. Specifically, programming is done by snapping commands

(sometimes with parameters) together to move various objects on the background [16]. Similar to the way people play with LEGO bricks, “Scratchers” design and build their own projects freely with these components.

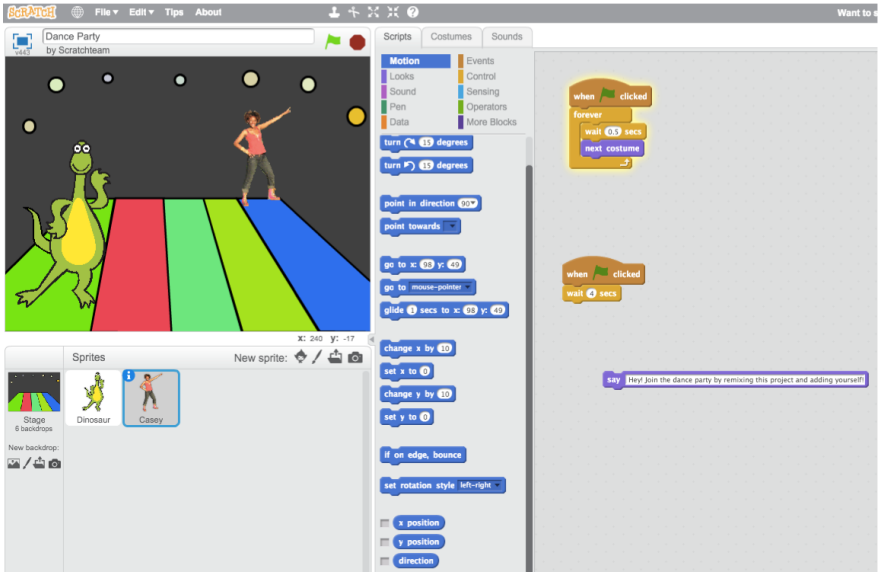

Figure 2.2 User Interface in Scratch

Since it came out, Scratch has been studied and tested by different groups of education researchers. The common opinion among these studies is that Scratch helps elementary students develop programming concepts and engage problem solving [17]. Like real programming, Scratch offers children a chance to think as a designer and creator by expressing themselves through projects they build. The way of adding up command blocks and parameters also leads directly to actual coding structures. By providing an open environment of graphical command blocks, Scratch excels at helping children transition to real computer programming. Likewise, Scratch also provides a platform where children can share their works with each other. Discussion forums and

online activities have played significant roles in promoting people to communicate and participate in social networks.

On the other hand, being a tool to learn programming (rather than a game), Scratch may not appeal to children as much as The Foos. Research shows that students tend to find options and commands too complicated in Scratch, which causes difficulty in the process of building projects [17]. Moreover, some students are expecting "something cooler" in Scratch considering the visual effects [17]. Therefore, the future plan for Scratch team is to focus on lowering the technical floor and making it more approachable for kids [15].

2.3 CodeCombat

CodeCombat is a 2D browser-based game that teaches people about coding at various levels of skill [18]. In a typical CodeCombat game, the player needs to plan ahead to help the game character collect all the gems and move to final destination. With multiple commands provided based on the items the character owns, the player has to write the corresponding code in the right window to use them. Once the RUN button is hit, the character will execute the sequence of commands such as moving in a specified direction or attacking a specific enemy (in this case the name of enemy is required as a parameter). As the player makes progress in the game, more programming concepts like loops and conditions are introduced. Besides the learning aspect, CodeCombat is also considered as a role-playing game as the user plays a hero adventuring through different levels and switch from various scenes [19].

Figure 2.3 A Game Scene in CodeCombat

With the combination of role-playing and typing code, CodeCombat is a useful tool for children to learn and practice programming. On the right side of the game interface, all the parameters and functions are stated clearly in text, which is basically a simplified real coding window. Through this approach, CodeCombat is able to help its players improve their understanding of programming. Furthermore, the fantasy backstory and settings of CodeCombat appeal to a young generation of players [20]. Research shows most participants find the visual design and cartoon style of CodeCombat easy and engaging in the game [21].

However, when it comes to coding education for K-12 students, CodeCombat may be a little too specialized. CodeCombat makes players pick a specific coding language such as Javascript or Python before they start the game. Because of this design approach, the game narrows the education of CT down to learning a certain programming language, which might not be appropriate for children, and may focus too strongly on programming a specific language over CT. Other than that, excessive text input may also cause kids at the younger age lose patience with the game.

2.4 Summary

Besides the three learning tools described above, other coding games such as Lightbot [22] and Cargo-Bot [23] have been studied. Based on the experience of these games and analysis of them, several things were considered in the game design of this thesis.

- Visual design and character settings should be friendly to children as in The Foos.
- Symbolic and graphic expressions should be used to avoid excessive text-based explanation and text input in the game interface.
- Parameters should be added into coding while working to ensure the complexity is not set too high. Game progress is supposed to move forward gradually with guidance through tutorials.
- A sufficient, but not excessive, amount of text should be used to represent code structures like in CodeCombat; but the text pattern is not supposed to indicate any specific programming languages.
- Various game elements such as maps and items should be added into the game world to keep the player highly involved.
- A reward system should be designed to encourage the player to come up with optimized solutions during every game.

CHAPTER THREE

GAME DESIGN

3.1 Overview

Guardy is an iOS tower defense game that teaches children about programming. The aim of this game is to help the player learn about CT and basic coding structures through playing the game. The target audience of this game is children aged eight and above, but it may also be beneficial to adults who have no programming background.

Like most tower defense games, the player views a preview of incoming enemies and plans ahead with a sequence of commands to defeat the enemies in each level of the game. Once the player has created a defensive plan and hits the play button, the attack of minions begins and the player's commands are executed to destroy them. The goal of the game is to defend attacks and prevent a certain number of minions from passing. During the game, the player's goal is to come up with solutions to stopping the minions, using series of different commands similar to coding functions. In this way, children get to practice problem solving and CT as they play various levels of game.

3.2 Conceptions

Overall, a tower defense game itself is similar to the CT process in programming. The preview of incoming enemies and existing commands has resemblance to the requirements for a coding project and available resources in the coding environment. From thinking through the given problem to designing a plan to defend against the attack, the process of successfully playing the game is analogous to coming up with a solution to

a specific programming problem with CT. Moreover, running the commands to test if a defense plan works is the same as running a program to test the result of coding. Through readjusting the defense plan based on feedback, the player gets to practice recursive thinking and efficiency in CT.

Specifically in Guardy, there are a few other CT conceptions behind the game.

- Programming can be conceptualized as a battle. The game helps players get to know that coding is not just creating a stack of commands but also a fun way to engage in CT and its problem-solving process.
- The commands the player uses represent functions in programming. Through the game, the player needs to learn how to arrange these commands to complete the level. The sequence of command lines indicates the structure of coding and designing systems in CT.
- Differences among minions relate to the concepts of classes and objects. Classification is a vital part of CT and in this game the player has to recognize different types of minions and use corresponding commands to deal with them.
- Loops are implemented in the game. Loops are helpful when repeated work is needed in programming. In the same way, loops are provided in the game as a tool to run the same operation for multiple times.
- Conditional variations are considered. Just like loops, conditionals used in coding are integrated in this game. In the later phases of Guardy, unknown minions are added into the preview window as question marks. The type of these enemies can change under different conditions. Thus, the player can

practice nondeterminism and decision-making in CT by covering all the possible types of unknown minions.

- A carefully considered amount of typing gives children a closer look at programming. In the final several levels of the game, the player is required to type the command lines instead of clicking on the buttons. Typing the code into the program is necessary for coding and the simplified typing process helps teach children the rigor of implementing a solution in CT.
- Feedback is given in the game to help players optimize their solutions. Besides the result of planned commands after tapping the play button, the players are also notified if their solutions are ill-structured. For example, players will see a syntax error warning if they use loops function incorrectly. In this way, the player gets to learn CT through receiving feedback and evaluating solutions.

3.3 Gameplay

In total, Guardy has thirty game levels. Once one level is complete, the next level will be unlocked to be played.

3.3.1 Play Flow

At the beginning of each level, a preview window showing incoming enemies is displayed on the top left of the scene as in Figure 3.1, so that the player can see the specific sequence of minions in the level. Meanwhile, available commands to deal with these minions are provided as buttons in the bottom toolbar. Based on that, the player is required to analyze the minions and find a plan with different commands in an order to destroy all enemies. When the player taps a command button, text of that command will

be written down to the command list window on the top right, showing the current sequence of commands he/she has selected. Once the player finishes adding and arranging the commands in the list, he/she can tap the start button to release the minions and they will appear from the left and move toward the character, following the path in the map. At the same time, the commands are executed as minions come out so that the player can see real-time feedback as the battle unfolds on the map. In case the result of selected commands does not turn out well, once the start button is tapped, it will be replaced by a stop button that can stop the running process and allow the player to rearrange commands in the list. With the start button and the stop button, players can always test and debug their commands to find the best solution to eliminate all the enemies. If players let a certain number of minions pass through during the game, they will fail the level and have to replay it. Otherwise, they get reward coins as the level is completed, and are able to move to the next level.

Figure 3.1 Basic Play Flow in Guardy

3.3.2 Challenge Structure

The performance of the player is rated on three components—destroying all enemies, the character's remaining health points, and the number of used commands. In each level of Guardy, the player starts with four health points. If a minion passes through and hits the character, the player loses one health point and the minion will be removed. When all the minions are eliminated by commands, the level is complete and the player receives one star. If the player manages to maintain all four health points until the end of level, one more star will be rewarded. The third star will be given if the player completes the level with a minimum number of commands. Every star earned means more coins for the player. Because of this rewards system, the ultimate mission for the player is to come up with a solution consists of the fewest number of commands needed to eliminate every enemy minion without the character getting hit.

3.3.3 Game Progression

As the levels progress, the difficulty and complexity of game rise too. Not only do scenes and potential minion paths change, but also minions and defensive tools get more complicated in the later levels. For example, besides basic minions that can be destroyed by fire or ice spells, there are higher-level minions that can take multiple shots and colored minions can only be killed by correspondingly colored towers. For tools, loops that repeat the same operation multiple times and conditionals that run different commands under various situations will be available as players unlock more levels.

3.4 Mechanics

3.4.1 Physics and Movements

In Guardy world, movements are strictly controlled by moving objects straight from one location to another with no gravity, outside force or friction applied in the system. Minions and spells involving movement (such as the fireball) are programmed to follow a certain path with a set of coordinates in every level. Similarly, tower bullets are shot out by being moved in a straight line from the position of tower to the target enemy.

3.4.2 Conflicts

Objects in the game scene are assigned with different masks. Whenever two objects overlap on the screen, one can identify the other with their masks to see if they would cause a conflict. Once a conflict happens, masks are used again to detect which object needs to be removed or have a clash effect applied to it. For example, when an enemy object crashes into the player character in the game, conflict between them will be detected and the enemy will disappear while the character will lose one health point.

3.4.3 Communications

The player's input methods include buttons and typing. Buttons are for navigating among different scenes and for functional features such as buying items from the store. In early phases of game levels, spells and tools also appear in the form of buttons. Meanwhile, text input is only available in the last seven levels of the game. In these levels, the player is supposed to type commands with the keyboard while buttons are unavailable.

On the other hand, graphics, text, and sound effects are used to output information to the player. Specifically, the visual look of every button is designed to indicate its

function and reduce the amount of text in the game. However, warning signs and explanation windows include text to ensure that the player has a good understanding of the game goals and functions. In addition, different sounds are played in various scenarios to match the messages displayed.

3.4.4 Economy

Golden coins are the currency used in the game. Whenever the player completes a level, a certain number of coins will be rewarded. The number of coins varies based on the number of rating stars the player gets at end of the level. Besides that, additional coins will be added to the account if the level is completed for the first time or a pet (details in the next paragraph) is used in a battle. With these coins, the player can purchase different kinds of items in the store to improve the experience of battles.

3.4.5 Items

Three kinds of items—weapons, armor, and pets are added into the game to increase player engagement. A weapon improves the speed of spell casting of the character while armor gives the character one more health point. When the character is battling with a pet, he/she gains extra coins when the level is completed.

At the beginning of the game, most of items are locked in the store. As the player completes levels and moves forward, more expensive items will be unlocked and become available in the store. Once the player buys the item in the store, it can be equipped or unequipped in the character scene.

3.4.6 Minions

There are seven kinds of enemy minions in Guardy. At first, two kinds of basic minions include a dark blue one that can only be killed by a fireball and a dark green one

that can only be destroyed by an ice spell. In addition, there are two kinds of high-level minions in (light blue and light green colors) that move faster than basic ones. Both of them can be destroyed by fireballs or ice spells, but the blue one takes two shots while the green one takes three shots. Finally, there are specialty minions in three colors—red, gray and brown. These enemies can only be eliminated by the correspondingly colored towers, which can detect and shoot color-matched minions automatically.

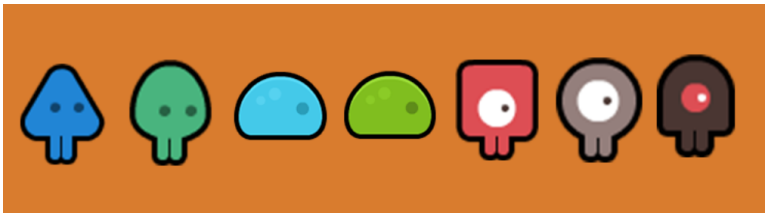

Figure 3.2 Enemies in Guardy

In addition, when conditionals are introduced in the game, there will be question marks in the preview window of coming enemies. In the case, question marks will become different kinds of minions based on the value of the condition when they are released.

3.5 Levels

In Guardy, levels are distributed based on how programming concepts are introduced to players. From basic operations to complicated commands, players gradually learn the way these components are arranged together to solve the problems.

3.5.1 Basics and Fireball

In the first two levels, the player is supposed to get familiar with what a typical game scene looks like. Basic operations introduced in these two levels include clicking

the preview window, adding a fireball spell to the list of player commands, deleting the last command, releasing minions and running commands, and stopping the commands. Meanwhile, to ensure the simplicity, only fireballs and dark blue minions are included in these two levels.

3.5.2 Ice Spell

Level three and four will introduce dark green minions to the game. The corresponding spell for them is an ice spell, which is also available in the button bar at the same time. For these two levels, players are able to continue playing around with basic operations while getting used to the ice spell command.

3.5.3 Loops

Loops will show up at levels five and six as a pair of parentheses. In Guardy, loops only have five options—times the commands inside them can be repeated (two, three, four, five or nine). In these two levels, easy loop patterns are integrated in the sequence of enemy minions to help the player learn loops. With loops, players are required to conduct more thinking than previous levels to find the repeat in the problem.

3.5.4 High-level Minions

Light blue enemies and light green enemies are introduced in levels seven and eight respectively. As mentioned before, these two kinds of minions need multiple shots of either the fireball or ice spell in order to be destroyed. The appearance of them increases the difficulty for the player to find the pattern in an enemy sequence and use loops. Thus, levels nine and ten don’t bring in any new concepts but only help players practice previous operations.

3.5.5 Towers

From level eleven to level fourteen, towers and their corresponding enemies are added into the game. In each level, there are only a limited number of spots for building towers and one tower is good enough to destroy all the minions that match a given tower's color. The player needs to plan ahead to decide which towers should be used in the level. At same time, the mix of tower minions and other minions raises the difficulty of finding loop patterns.

3.5.6 Starbomb

When too many high-level minions appear in the enemy sequence, the spell casting of the character might not be fast enough. In levels fifteen and sixteen, star bomb is introduced to slow down the speed of present enemies on the map. In these two levels, players need to figure out how to integrate star bomb into loops to gain more time for their spell castings.

3.5.7 Conditionals

In real programming languages, conditionals are variables that change values under different situations. In Guardy, question marks appear in the enemy sequence as unknown variables in levels seventeen and eighteen. They can be one kind of basic minions or tower minions. The possibilities of their type will be displayed next to the preview window. With that information, the player needs to cover all situations with the provided "if" parentheses. Just like coding, when the condition is not met, the commands inside the parentheses will be skipped. Otherwise, they will be executed. The introduction of conditionals brings in factors to encourage the player to conduct comprehensive CT.

3.5.8 Practice Levels

After all the commands available in Guardy are introduced, later levels are just for players to practice the problem solving process. From levels nineteen to twenty-two, different minions are mixed with question marks in the enemy sequence and the number of minions also increases. The solutions to these levels are open, thus players have to master the use of tools and analyze the problems to come up with the best plan.

3.5.9 Typing

From levels twenty-three to thirty, buttons are replaced by a text field where the player can input text with the keyboard. Available command samples are displayed in the toolbar so players know what to type for a specific function. The difficulty of the levels and tools provided are the same as those of practice levels. With the change of the input form, the player is able to grab a deeper understanding of programming as real coding involves a lot of typing.

3.6 Interface

3.6.1 Visual System

The visual system of Guardy is divided into six scenes—main menu, store scene, character scene, information scene, level selection scene, and game scene. Main menu is the beginning scene of the game and it has four buttons leading to other four scenes except game scene. Store scene displays items available in the store and the number of coins the player has. All purchases are conducted in this scene with buttons and selections. Character scene shows players the items they have and also provides a preview window for them to put on or take off items for the character. Information scene

has three pages to explain the basics of the game and one page to credit the people or resources involved in the project. Level select scene displays thirty levels and their ratings. Every level in the level scene is a button that leads to its game scene when it is clicked.

Game scene consists of a game map and its path, the character, and four heads-up displays—enemy window, toolbar, command window, and game over window. Enemy window on the left top of scene displays the sequence of incoming minions in this level and it can be hidden or shown with a click during the game. When the question marks appear in the level, the possibilities of the question mark are also attached to the enemy window. Toolbar, located at the bottom the scene, contains all the available command buttons in the game. When a text field replaces buttons in later levels, toolbar will remove the buttons and display command samples. On the top right of the scene, command window displays the commands the player has picked. It also has a menu button which leads back to level selection scene and a play button to release enemy minions. After the play button is clicked, it turns into a stop button that can stop the running process of enemies. In addition, on the top right of the command window, there is a label displaying the number of coins the player has. Game over window will pop up in the middle of the scene when the player completes the level or fails. It displays the ratings of the level and bonus coins. Meanwhile, the game over window also has three buttons with which players can go back to the level scene, replay the level, or play the next level.

Figure 3.3 Game Scene in Guardy

3.6.2 Music and Sound Effects

In Guardy, a piece of background music is played in all scenes except the game scene. Sound effects are also added to different operations including button clicks, warnings, explosions, putting on items, buying items, casting spells, enemies getting hit, completing a level, and failing a level.

3.6.3 Help System

Tutorials, warning signs, and information scene are integrated to help the player quickly get familiar with the game. Tutorials are displayed before the game starts when a new game conception is introduced in the level. They appear in the form of simple illustrations with arrows and some text. Warning signs are implemented in various occasions; such as when loop or if parentheses have a syntax error or the player types in wrong commands. Information scene, as mentioned, gives players a simple overview of enemies and items.

3.7 Visual Design

Because the target audience of Guardy is children, the intended style of visual design is a colorful and simple cartoon. A jungle theme user interface is applied in all scenes to keep the elements vivid and consistent. Besides that, the character and enemies are also distinguished by bright colors to help children easily classify them. To avoid excessive text in the game, graphics are also used in buttons to indicate their functions. Finally, items and game maps are also designed in 2D cartoon style to keep all elements compatible.

CHAPTER FOUR

GAME DEVELOPMENT

4.1 Overview

As an iOS game, Guardy is developed on Apple's integrated development environment Xcode. The programming language Swift and game framework SpriteKit are used for the game.

Swift, Apple's own programming language, is made specifically for iOS and Macs. Apple has been improving its performance since it came out, thus it became the most suitable language to write iOS apps [24]. Swift's simplified syntax makes code concise and efficient and its Automatic Reference Counting tracks and manages the app's memory usage [25]. At the same time, its type safety and new variable type—optionals—also ensures that errors can be caught and fixed early in the development process [26].

SpriteKit is Apple's built-in framework for developing 2D games. It is an infrastructure that renders graphics and animates sprites in the game. Besides its support operations and actions for sprite nodes, SpriteKit also has an integrated library to simulate physics in the scene [27]. According to the requirements and purpose of Guardy, SpriteKit has the tools needed to develop this game.

4.2 Modeling

4.2.1 MVC Design Pattern

The Model-View-Controller (MVC) pattern is the architectural pattern Apple recommends for iOS development. In short, this pattern divides objects into three

layers—model, view, and controller. The model layer encapsulates the data specific to the app and handles various operations to process the data [28]. The view layer, as its name indicates, is in charge of displaying views to the user. Because of its visibility to users, user interactions with the app are in fact with the view layer [29]. In the MVC pattern, the view layer normally does not process operations or logics but only displays data from the model layer. However, consistency is required for the view objects because users often reuse and reconfigure the view in apps [28]. Therefore, most of time the view layer is decoupled from the model layer and a controller layer is brought in to coordinate these two layers. Controller layer communicates between view objects and model objects through interpreting users' actions in the view layer and conducting operations on the data of the model layer. Besides that, controller objects also perform tasks like setup and arranging the order of operations.

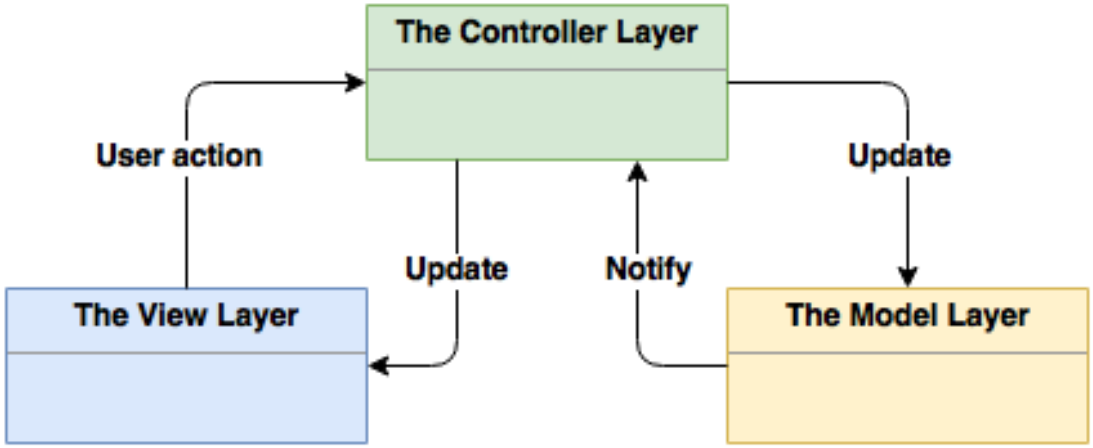

Figure 4.1 MVC Design Pattern

Specifically in Guardy, MVC pattern is applied in the early stages of design. Data including level settings, ratings, items, number of coins, and the character is stored in the model layer for the use in the game. In the view layer, a UIView and multiple SKScene

classes are used to present the view to players and display nodes like buttons and labels. Controller objects are integrated in all the classes to decide how to react to the player's actions in the game. For example, when the player clicks a button in the view to add an item, the controller gets the action through functions touchesBegan and touchesEnded. Then, it plays the sound for the operation in the model, changes the data of that item, and stores it back to the model.

4.2.2 Class Diagram/Screen Frames

In SpriteKit, SKScene is used to represent a scene of contents, therefore each functional scene is implemented as a class in Guardy. But due to the special structure of SKScene and features in SpriteKit, a traditional class diagram might not be suitable for modeling the game. Thus a combination of class diagram and screen frames is used in the design stage to model Guardy.

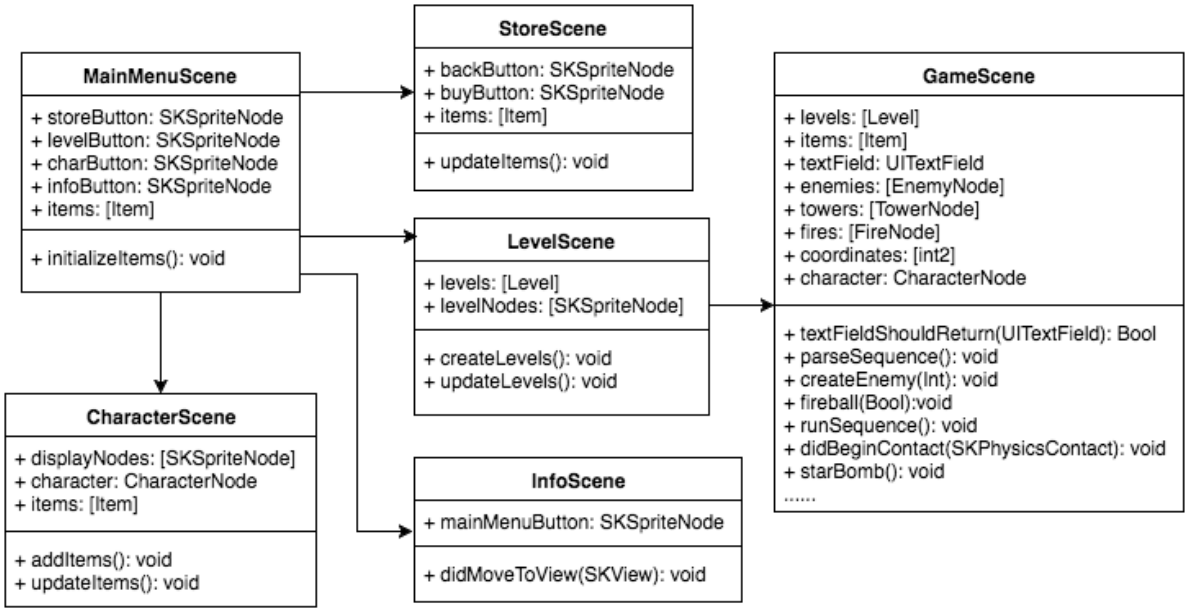

Figure 4.2 Class Diagram/Screen Frames of Guardy

4.3 Implementation

Based on the modeling, the Guardy world is implemented with functions in different parts including data storage, connections, actions, etc. In this thesis, only the implementation of several core functions is discussed below.

4.3.1 Data Storage

Data that needs to be stored in the game includes level settings, level information, items, and the number of coins. Level settings consist of enemy sequence, tower spots, coordinates of the path, number of buttons, and other requirements in each level. All the settings are translated into dictionaries and stored in property list files of the project. Before a level starts, game scene will access the file, load the corresponding level setting and set up the view based on it.

Level information indicates if the level is locked and the ratings of the level. Data of level information and items exists as two arrays of level objects and item objects respectively. Values of their elements are encoded with different keys and stored under the document directory of the game. When the data is needed in the game, the controller will decode files and get the values with the keys. In the same way, the number of coins is encoded and stores through the path in NSUserDefaults.

```
class Level: NSObject, NSCoding{

    // MARK: Properties

    var locked: Bool
    var rate: Int

    // Archive Paths
    static let DocumentsDirectory = NSFileManager().URLsForDirectory(.
        DocumentDirectory, inDomains: .UserDomainMask).first!
    static let ArchiveURL = DocumentsDirectory.
        URLByAppendingPathComponent("levels")

    // Types
    struct PropertyKey {
        static let lockedKey = "locked"
        static let rateKey = "rate"

    }
```

Figure 4.3 Part of Declaration of Level Information

4.3.2 Game Objects

In SpriteKit, every object in the game is represented as an SKNode. According to the varieties of game objects in Guardy, they are created as multiple subclasses of SKNode. Each subclass has its own functions and also inherits or overrides functions from their parent classes. Figure 4.4 indicates relationships among these object classes.

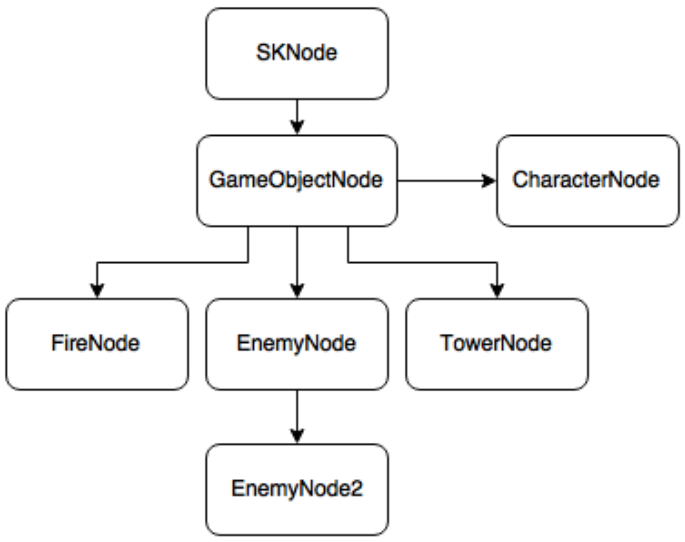

Figure 4.4 Tree Diagram of Nodes in Guardy

4.3.3 Command Parser

During every game, the player inputs a list of commands that are stored in an array of strings. Before these commands being run, loops and conditionals inside them need to be parsed. The process of parsing includes three steps in total. At first, the syntax of commands should be examined to ensure every pair of parentheses of loops or conditionals is closed and in the right order. Secondly, to avoid unnecessary computation in the program, conditionals are parsed first. When the value of the condition is false, commands inside the conditional parentheses will be removed by array operations. Finally, loops will be translated into repeated operations and inserted back to the array. With the parsed array of commands, the program will go through them one by one and execute the corresponding functions in the scene.

4.3.4 Tower Functions

Towers in Guardy detect their enemy minions and shoot them automatically. Both towers and their minions are tagged with an integer to indicate their type. When an enemy enters in a tower's range and their tags match, the tower will append the enemy into its shooting sequence. For every enemy in the tower's shooting sequence, the tower will rotate to the enemy, shoot a projectile to it and then remove it from the sequence. These actions of towers are implemented with SKActions, which are used to rotate and move nodes in the scene. On the other hand, when the enemy leaves the shooting range, the tower will just remove it from its shooting sequence.

```
class TowerNode: GameObjectNode {

    var sprite: SKSpriteNode!
    var identifier: Int!
    var enemiesInRange = [EnemyNode]()

    let rotationAngle = CGFloat(M_PI * -1)
    let rotationRate = 4.0

    init(sprite: SKSpriteNode, identifier: Int) {
        super.init()
        self.sprite = sprite
        self.identifier = identifier
        self.addChild(sprite)

        rotateNodeForever(sprite, byAngle: rotationAngle, duration:
            rotationRate)
    }
```

Figure 4.5 Part of Declaration of Tower Node

4.3.5 Collisions

Collisions between objects in Guardy are simulated by the use of SKPhysicsBody. For every object in the map, it is assigned a certain sized shape and a mask. When two shapes overlap in the view, a collision happens if their masks match as well. Function didBeginContact is called automatically whenever a collision happens, so inside the function, physical contacts of the objects will be handled. Incidents like when a bullet hits an enemy or an enemy crashes into the character are all simulated by collisions.

```
// MARK: Physical Collision
func didBeginContact(contact: SKPhysicsContact) {

    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!

    if nodeA is EnemyNode {

        let enemyA = nodeA as! EnemyNode

        if nodeB is BobiNode {
            let bobi = nodeB as! BobiNode
            bobi.collisionWith(enemyA)
        }
        else if nodeB is FireNode {
            enemyA.collisionWith(nodeB)
        } else if nodeB.name == "bullet" {
            enemyA.collisionWith(nodeB)
        }
```

Figure 4.6 Part of didBeginContact Function

4.3.6 Text Input

Since SpriteKit does not provide any tool for text input, typing has to be implemented with a UITextField. As an interface of UIKit, the text field is added into the scene as a subview of the view controller. The elements of text field such as font and text color can be easily set with its well-declared attributes, but its location becomes a problem when the keyboard pops up and blocks the view of it. Therefore, the whole view should be moved up to make the text field above the keyboard when the player is typing. To solve this problem, the default center of built-in class NSNotificationCenter is brought in. It is able to add observers for events like KeyboardWillShow and KeyboardWillHide in this case. When these events happen, functions will be called to adjust the position of the view.

```
// Add the notification center to detect the keyboard.
NSNotificationCenter.defaultCenter().addObserver(self,
    selector:  #selector(self.keyboardWillShow), name:
    UIKeyboardWillShowNotification, object: nil)
NSNotificationCenter.defaultCenter().addObserver(self,
    selector: #selector(self.keyboardWillHide), name:
    UIKeyboardWillHideNotification, object: nil);
```

Figure 4.7 Code of Adding Observers

4.4 Testing

The goal of testing is to verify and validate if a piece of software is built based on requirement specifications. In this regard, the testing process is carried out in steps below.

4.4.1 Unit Testing

To understand more about the internal workings of the game and ensure operations are performed as expected, unit testing is adopted to test several functions in

the game scene. XCTest is the framework provided in Xcode to write units test for apps. Its simplicity and tight integration into Xcode development environment make it easy to test units in Guardy. Specifically, test cases are written for testing the command parser and other functions. When the test cases fail, bugs will be detected by the program, and the corresponding code can be fixed right away.

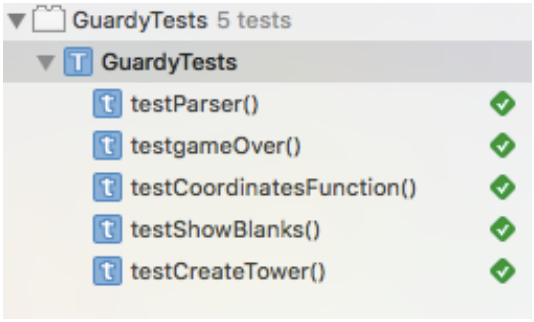

Figure 4.8 Test Cases of XCTest

4.4.2 Black Box Testing

Contrary to unit testing, black box testing, also known as behavioral testing, focuses on inputs and outputs of the game instead of its internal structures. For Guardy, black box testing is run based on the requirement specifications and a set of use case scenarios to simulate the user experience. A playable prototype of Gurady is downloaded on a device to conduct black box testing. To ensure functions are complete and meet the requirements, normal operations are performed first on the game to detect errors. After that, the prototype is sent to several end-users to test exceptional operations. When errors occur during the black box testing, they are reported and related classes and functions will be checked manually to fix the code.

Several behavioral problems of Guardy were detected in the black box testing. One scenario was after an end-user added commands to the full command list, tapping

the undo button did not remove the last command. This issue was fixed later by redesigning the array of command list to ensure it would not append more commands when the list is full. Another case was after an end-user typed a wrong command in the text field, the whole command was removed so all the text had to be typed in again. This problem was solved by keeping the text of wrong input in the text field to be fixed by the player. Besides these, other issues like the incorrect display of the number of coins were fixed in their related code functions.

4.5 Deployment

As an iOS game, Guardy is designed to run on any iOS device. But because settings and sizes of iOS devices vary, an iPad Mini 2 was chosen to deploy the game for usability testing. With the features in Xcode, the game can be launched on the device through provisioning, registering the developer's information, and transferring files of the project to the device.

CHAPTER FIVE

USABILITY TESTING

5.1 Overview

For games and software, usability testing is always a useful and straightforward way to receive feedback and test if they are designed well for the target audience. But traditional methods for testing productivity software may not be a good choice for games because they are essentially different [30]. Moreover, other than regular games, Guardy aims to engage players across learning activities around programming. For such an educational game, the usability testing should focus more on learning effectiveness, user engagement, and player interactions with the game itself [31]. Activities involved in the usability testing include planning, identifying the usability model, preparing the session, selecting subjects, conducting the test, and analyzing the data [32].

In terms of the specific method of testing in this work, a combination of observational analysis and questionnaire was adopted. During the testing process, subjects are allowed to interact with the prototype while the investigator observes how they figure out the way to play the game, and the type of problem solving they go through during their playing [33]. According to the observation and side notes the investigator takes, issues affecting the user's interactions can be detected directly. The questionnaire method, similar to traditional methods, is more based on evaluating the performance of game. In the survey, subjects need to rate the attributes of the game like difficulty or visual design, and answer open questions about their opinions toward the game. With the

model involving both observation and questionnaires, the usability of Guardy can be tested in a comprehensive way.

5.2 Subjects

Even though the target audience of Guardy is children over eight years old, to reduce the complexity, the usability testing was conducted with 10 college students. Research has shown that five users are enough to discover most usability problems with software [34], but due to the educational purpose of Guardy, additional subjects were added into the testing process to determine the educational impact on programming. The 10 subjects were divided into two groups—students who barely have any programming experience and those who are familiar with coding. Each of the 10 subjects played through the game and looked for usability errors meanwhile the programming impact of the game could be indicated through comparison of two groups' reactions.

5.3 Process

During this study, each subject took part in a roughly 30-minute usability testing session individually. At first, they went over the information scene, the store scene, and the character scene of Guardy on the device to get an overview of the game. Then, subjects were asked to play the game from level one and vocalize their thinking process in every level. At the same time, the investigator observed the subject's behaviors and wrote down notes about their behaviors. During the subject's playing, a few practice levels were skipped over to reduce the time of the session. After completing all the levels that introduce new conceptions and some practice levels, each subject was asked to rate

the performance of Guardy and give opinions on the questionnaire to end the session. At last, notes from the observation and answers to the questionnaire were processed and organized for data analysis.

5.4 Question Sample

As mentioned above, the questionnaire includes rating the performance of Guardy and open questions about programming. Meanwhile, the observer of the test also has a set of topics to focus on during the observation.

Table 5.1 Questions on the Questionnaire

Questionnaire of Guardy	
Q1	Rate the difficulty of understanding how to play the game. (1-10)
Q2	Rate the difficulty of understanding enemy minions. (1-10)
Q3	Rate the difficulty of understanding the store and item system. (1-10)
Q4	Rate how well the game introduces new programming concepts. (1-10)
Q5	Rate the difficulty of playing the game. (1-10)
Q6	Rate the visual design of the game. (1-10)
Q7	Rate the sound effects of the game. (1-10)
Q8	Rate the pleasure of playing the game. (1-10)
Q9	How would you describe programming before playing the game?

Questionnaire of Guardy	
Q10	How would you describe programming after playing the game?
Q11	What kind of problem solving did you go through while playing the game?
Q12	Is this game suitable for children over 8 years old to play?
Q13	Would this game help children over 8 years old learn programming?

Table 5.2 Observation Topics

Observation Topics	
Q1	Has the subject finished the levels on time?
Q2	Has every level been completed in the most efficient way?
Q3	Has the subject found any trouble in one or more levels?
Q4	Based on vocalized reactions, has the subject conducted computational thinking?
Q5	Other notes.

5.5 Data Collection

Information collected is also divided into two groups as the subjects are. Group 1 consists of subjects who have no programming experience and Group 2 consists of those

who already have a good understanding of programming. From questions 1 to 8 on the questionnaire, answers of subjects are averaged in two groups to rate the performance of Guardy as shown in Figure 5.1. Meanwhile, subjects' answers to questions 9 to 13 are attached in Appendix A, and the result of observations will be discussed in the next chapter.

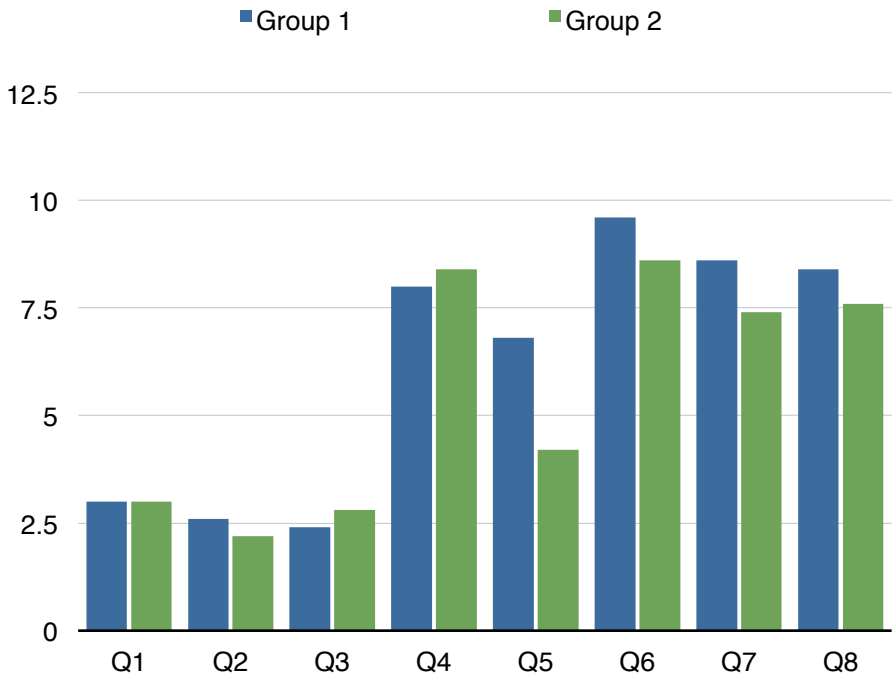

Figure 5.1 Bar Chart of the Answers to Questions 1-8

CHAPTER SIX

RESULTS AND DISCUSSION

With data collected from the usability testing, a fairly comprehensive evaluation of Guardy can be given through analysis. In general, three parts of the data need to be discussed in this work.

6.1 Performance

Bar chart in Figure 5.1 shows the testing results of Guardy's performance. In general, two groups' answers to these questions are almost identical except for Q5. Regarding to the difficulty of the game, Group 1 has an average rating of 7 while the mean of ratings from Group 2 is 4. The obvious difference between 7 and 4 indicates level settings of Guardy do require a programming-like mind to solve problems while the general high ratings show that current levels are challenging for both groups to some extent.

Other than Q5, ratings of other questions demonstrate that Guardy is fairly good as a game. Based on the ratings of Q1 to Q3, both two groups of subjects do not have much trouble understanding the basics of the game. In Q4, the ratings are all above 8, which demonstrate that Guardy achieves its purpose of introducing programming concepts to players. As for Q6 to Q8, Guardy is considered a high-quality game among all subjects.

6.2 Open Questions

As shown in Appedix A, through the comparison of answers to Q9 and Q10, most subjects received a better understanding of programming after playing Guardy. Specifically, for subjects in Group 1, programming was abstract to them before but later, they were able to use words like “input and output” and “patterns” to talk about coding. Even for Group 2, subjects seemed to describe programming in more details after their playing.

For Q11, all subjects mentioned that during their problem solving they conducted significant steps of CT such as looping, sequencing, and mathematical thinking. But in general, subjects from Group 2 described their CT in a more comprehensive way as they talked about efficiency and complexity of the solution.

In terms of Q12 and Q13, most subjects were positive that Guardy could engage children over 8 to play and learn about programming. However, some of them also showed concerns about whether the game was too difficult for children. But in general, most of subjects agreed that Guardy makes programming more approachable for children and people without programming experience.

6.3 Observation

As the supplement of questionnaire in this usability testing, observation was mostly focused on subjects’ behaviors during playing the game. First of all, observations reveal that the amount of time subjects spent on levels does not completely depend on the amount of programming experience they have. Some subjects from Group 1 completed levels surprisingly fast and the reason behind it, as they described, might be that they are

familiar with math and logic practices in daily life. Secondly, some of subjects in both groups were able to come up with simpler ways than the standard solution to solve a couple of problems. In this case, Guardy does provide them with a relatively open environment to think of other possibilities.

Finally, even though all the subjects had not heard of the concept of CT, based on their verbalized thoughts, all of them had conducted CT while playing the game. When they looked at the preview of incoming enemies and counted the number of hits, they were understanding and analyzing the problem. When they tapped buttons to write down the list of commands, they were expressing their plans and forming solutions. Moreover, once the battle started on the map, they got to evaluate their solutions according to the feedback and fix errors to improve the performance. Overall, basic steps of CT process such as prefetching, computing, decision-making, algorithms, backtracking etc. were conducted during their playing.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

Guardy is an iOS tower defense game that has the potential to teach children over 8 years old about programming. It simplifies the structures and methods in real coding and brings them into game battles to make them appealing to children. However, user testing revealed that concerns about the difficulty of the game existed for players with little or no programming background, and thus some changes should be made to level settings and the introduction of loops and conditionals. Furthermore, the typing system and command syntax should be simplified to engage children even in the later levels. Overall, Guardy has presented a feasible way to help children learn CT and problem solving outside the school curricula system.

For future work, besides the problems found from usability testing, several solutions to extend the impact of Guardy can be implemented. More characters and storylines can be added to the game to form a complete system that keeps children more involved and interested. Meanwhile, network functions should be considered in the future development to create a community for children to share their solutions or ideas. In addition, more programming conceptions like parameters can be integrated when more levels are designed in the game.

REFERENCES

[1] Levis, Phillip. "Education and Job Opportunities in STEM, 2008." Education and Job Opportunities in STEM, 2008. February 2, 2012. Accessed April 03, 2017. http://csl.stanford.edu/~pal/ed/.

[2] Wing, Jeannette M. "Computational thinking." Communications of the ACM 49, no. 3 (2006): 33. doi:10.1145/1118178.1118215.

[3] Grover, S., and R. Pea. "Computational Thinking in K-12: A Review of the State of the Field." Educational Researcher 42, no. 1 (2013): 38-43. doi:10.3102/0013189x12463051.

[4] Hosford, Grant. "Do Your Kids Need to Learn to Code? Yes! But Not for the Reasons You Think." The Huffington Post. May 29, 2015. Accessed April 03, 2017. http://www.huffingtonpost.com/smart-parents/do-your-kids-need-to-lear_b_7473058.html.

[5] Kafai, Yasmin B., and Quinn Burke. Connected code: why children need to learn programming. Cambridge, MA: The MIT Press, 2016.

[6] "AP Program Participation and Performance Data 2015 – Research – The College Board." Research. Accessed April 03, 2017. https://research.collegeboard.org/programs/ap/data/archived/ap-2015.

[7] Lynch, Tom Liam. "Saving Computer Science Education from Itself." Medium. December 01, 2015. Accessed April 03, 2017. https://medium.com/@tomliamlynch/saving-computer- science-education-from-itself-3bd3e3c300a2#.yrl4rrx75.

[8] "About Us." Code.org. Accessed April 03, 2017. https://code.org/about.

[9] Shaffer, David Williamson. "Epistemology: The Debating Game." In How Computer Games Help Children Learn, 23-40. New York: Palgrave Macmillan, 2006.

[10] Gee, James Paul. "The Social Mind: How Do You Get Your Corpse Back After You've Died?" In What Video Games Have to Teach Us about Learning and Literacy, 188-209. New York: Palgrave Macmillan, 2007.

[11] Dörner, Ralf, Stefan Göbel, Michael Kickmeier-Rust, Maic Masuch, and Katharina Zweig. *Entertainment Computing and Serious Games: International GI-Dagstuhl Seminar 15283, Dagstuhl Castle, Germany, July 5-10, 2015, Revised Selected Papers*. Cham: Springer International Publishing, 2016.

[12] "Alternatives to Scratch." Scratch Wiki. Accessed April 03, 2017. https://wiki.scratch.mit.edu/wiki/Alternatives_to_Scratch.

[13] "CodeSpark Academy with The Foos." Accessed April 03, 2017. http://thefoos.com/.

[14] The Foos Curriculum-Introduction to Computer Science Grades K-5. codeSpark. May, 2016. Accessed April 03, 2017. http://thefoos.com/wp-content/uploads/2016/05/Hour-of-Code-Curriculum.pdf

[15] Resnick, Mitchel, Brian Silverman, Yasmin Kafai, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, and Jay Silver. "Scratch." *Communications of the ACM* 52, no. 11 (2009): 60. doi:10.1145/1592761.1592779.

[16] Maloney, John, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. "The Scratch Programming Language and Environment." *ACM Transactions on Computing Education* 10, no. 4 (2010): 1-15. doi:10.1145/1868358.1868363.

[17] Filiz KALELIOĞLU, Yasemin GÜLBAHAR, The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. Informatics in Education, 2014, Vol. 13, No. 1, 33–50.

[18] Project Proposal for PyTeacher, Eugene de Beste, Mark Grivainis, University of Cape Town. Accessed April 03, 2017. https://people.cs.uct.ac.za/~dbseug001/res/docs/proposal.pdf.

[19] Gube, Jacob. "5 Games That Teach You How to Code." Six Revisions. September 30, 2015. Accessed April 03, 2017. http://sixrevisions.com/resources/games-that-teach-how-to-code/.

[20] Kumar, Deepak. "Digital playgrounds for early computing education." *ACM Inroads* 5, no. 1 (2014): 20-21. doi:10.1145/2568195.2568200.

[21] Orehovacki, Tihomir, and Snjezana Babic. "Evaluating the quality of games designed for learning programming by students with different educational background: An empirical study." *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015. doi:10.1109/mipro.2015.7160414.

[22] "Lightbot." Lightbot. Accessed April 03, 2017. https://lightbot.com/resources.html.

[23] "Cargo-Bot – iPad." Accessed April 03, 2017. https://twolivesleft.com/CargoBot/.

[24] Solt, Paul. "Swift vs. Objective-C: 10 reasons the future favors Swift." InfoWorld. May 11, 2015. Accessed April 03, 2017. http://www.infoworld.com/article/2920333/mobile- development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html.

[25] Mathias, Matthew, and John Gallagher. *Swift Programming: the Big Nerd Ranch Guide*. Atlanta, GA: Big Nerd Ranch, 2016.

[26] "The Swift Programming Language (Swift 3.1): The Basics." The Swift Programming Language (Swift 3.1): The Basics. March 27, 2017. Accessed April 03, 2017. https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID309.

[27] "Apple Game Frameworks Category." Ray Wenderlich. Accessed April 03, 2017. https://www.raywenderlich.com/category/apple-game-frameworks.

[28] "Cocoa Core Competencies." Model-View-Controller. October 21, 2015. Accessed April 03, 2017. https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[29] "Model-View-Controller (MVC) in iOS: A Modern Approach." Ray Wenderlich. Accessed April 03, 2017. https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.

[30] Moreno-Ger, Pablo, Javier Torrente, Yichuan Grace Hsieh, and William T. Lester. "Usability Testing for Serious Games: Making Informed Design Decisions with User Data." *Advances in Human-Computer Interaction* 2012 (2012): 1-13. doi:10.1155/2012/369637.

[31] S. de Freitas and M. Oliver, "How can exploratory learning with games and simulations within the curriculum be most effectively evaluated?" *Computers and Education*, vol. 46, no. 3, pp. 249–264, 2006.

[32] Diah, Norizan Mat, Marina Ismail, Suzana Ahmad, and Mohd Khairulnizam Md Dahari. "Usability testing for educational computer game using observation method." *2010 International Conference on Information Retrieval & Knowledge Management (CAMP)*, 2010. doi:10.1109/infrkm.2010.5466926.

[33] M. Macleod and R. Rengger, "The development of DRUM: a software tool for video-assisted usability evaluation," in *Proceedings of the 5th International Conference on Human- Computer Interaction (HCI '93)*, pp. 293–309, August 1993.

[34] R. A. Virzi, "Refining the test phase of usability evaluation: how many subjects is enough?" *Human Factors*, vol. 34, no. 4, pp. 457–468, 1992.

[35] Salen, Katie, and Eric Zimmerman. *Rules of play: game design fundamentals*. Cambridge, Mass.: The MIT Press, 2010.

APPENDIX A

ANSWERS TO OPEN QUESTIONS

Questions:

Q9: How would you describe programming before playing the game?

Q10: How would you describe programming after playing Guardy?

Q11: What kind of problem solving did you go through while playing the game?

Q12: Is this game suitable for children over 8 years old to play?

Q13: Would this game help children over 8 years old learn programming?

Answers:

Subject 1 (Group 1)

A9: Writing commands to make something function properly.

A10: Writing commands to complete specific tasks.

A11: Trying to figure out how to consolidate the repetition of shooting.

A12: Yes.

A13: For first ten levels, yes.

Subject 2 (Group 1)

A9: Orchestrating a system to be used on a computer.

A10: A system with inputs and outputs.

A11: Math and patterns.

A12: Yes.

A13: Yes.

Subject 3 (Group 1)

A9: Creating some software on the computer.

A10: Starting by ground, building up a process, each thing is connected to each other.

A11: Recognizing enemies and a lot of math.

A12: I think so.

A13: It might be hard for younger ones.

Subject 4 (Group 1)

A9: Sounds like complicated, time-consuming, boring and difficult work.

A10: Not that hard as I thought it would be. It might be a little fun to program.

A11: Logic thinking, looking for patterns, making sure the solution works.

A12: Yes, except the typing part. Children might lose patience with it.

A13: Yes, but the part without typing might not leave an impression to kids.

Subject 5 (Group 1)

A9: Writing code to get machine to do stuff.

A10: Understanding problems, and trying to get the computer to do stuff in the most efficient way.

A11: Figuring out how many hits are needed and how to loop them.

A12: Yes, levels might be hard but the game is simple to play.

A13: Yes.

Subject 6 (Group 2)

A9: A process contains certain functions and types.

A10: Same as the last question.

A11: Looping, balancing between slowing down enemies and attacking them, which indicates complexity and accuracy in real programming.

A12: Yes.

A13: Yes.

Subject 7 (Group 2)

A9: Using the logics to solve problems.

A10: Not hard, but it requires a lot of thinking.

A11: Figuring out the most efficient way to write code with least command lines.

A12: Yes.

A13: Yes, it would help a lot. This game makes loop and if statements more approachable. I wish I had this game before I learned Matlab.

Subject 8 (Group 2)

A9: Finding solutions to solve problems.

A10: Finding the most efficient ways to solve the problem.

A11: Understanding the limit of requirements and finding patterns.

A12: Yes.

A13: Yes. This game involves syntax, patterns and using them to construct algorithms, which makes coding more approachable to people.

Subject 9 (Group 2)

A9: Logics.

A10: Making decisions, repeating to perform tasks.

A11: Understanding constraints, planning, coming up with the solution, classifying minions, and finding patterns.

A12: Yes.

A13: Yes.

Subject 10 (Group 2)

A9: Different languages the computer uses to dictate their actions.

A10: Series of instructions used to dictate the computer's actions.

A11: Assessing what the issue is, thinking of what's available to use, making choices, testing it out and iterating decisions.

A12: Yes.

A13: Yes. One good thing about this game is children will learn logic and sequencing easily instead of any specific programming language.