12-2012

Design and Implementation of a Genre Hybrid Video Game that Integrates the Curriculum of an Introductory Programming Course

Cory Buckley
Clemson University, corybuckley@gmail.com

DESIGN AND IMPLEMENTATION OF A GENRE HYBRID VIDEO GAME THAT INTEGRATES THE CURRICULUM OF AN INTRODUCTORY PROGRAMMING COURSE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters
Digital Production Arts

---

by
Cory M. Buckley
December 2012

Accepted by:
Dr. Donald H. House, Committee Chair
Dr. Brian A. Malloy
Dr. Jan Holmevik

Abstract

Video games have a history of being exploited for education. However, all too frequently, the resulting educational video game is either transparent in its hijacking of video game media, or the educational content is not intelligently placed within the context of the game. In this paper we analyze existing educational video games and observe popular commercial video game mechanics to form a more player oriented development mindset. Our approach involves mingling game mechanics that are not commonly used together to create a genre hybrid educational video game with a seamlessly integrated introductory programming curriculum. We use a machine architecture visualization tool to allow the player to write and execute simple programs. This leads to a deeper understanding of foundational concepts by exposing the effects of the code on the underlying hardware. The result of our research is a playable, cross-platform educational game that aims to teach foundational programming skills. Methods discussed in this paper can be used by educational game developers to create more engaging educational experiences.

Table of Contents

List of Figures

List of Figures (Continued)

Page

List of Tables

Chapter 1

Introduction and Motivation

The Digital Production Arts program at Clemson University is a challenging program because it requires students to be dual natured. Both traditional art skills and technical skills are required for students to be successful. It is not uncommon for prospective students to have strong traditional art backgrounds and weak programming backgrounds. These students may find it rather difficult to adopt a programming mentality even with foundational programming courses that are specifically designed to cater to traditional artists. This has led educators in the program to wonder if there are more engaging ways to teach foundational computer programming to these students. This educational challenge has led to the research in this paper.

Computer technology and its use in society has undergone a wealth of change in the past ten years, while surveys show that traditional methods of instruction have remained relatively constant in educational institutions [36; 38]. Research by Kaila et al. [23] on the effectiveness of program visualization on programming literacy appears to be promising and provides a strong argument for continued research in program visualization. Traditional artists, with their visually oriented skill set, may be more receptive to teaching methods that utilize visualization tools. However, research shows that visualization is often not enough to provide students with a firm understanding of the underlying concepts [21; 36].

In the past decade video game sales have seen an exponential growth along with a consistent expansion in player demographics [43]. This increase in video game consumption provides motivation for the study of integrated educational video game solutions. Traditional art students who apply to programs such as Clemson University's Digital Production Arts program are actively seeking jobs in the film and video game industries. Therefore, the video game may serve as a more engaging medium for communicating foundational programming concepts to these students.

In this paper, we describe an approach to the design and implementation of a hybrid educational video game that uses game mechanics from many traditional video game genres. Thus, we refer to our approach as a *genre hybrid* video game. While most educational video games exhibit gameplay built around the goals of the educator, we propose a model of gameplay that uses the genre hybrid approach to build a game around the the interests of the player, resulting in a more engaging educational experience. To accomplish this goal, we analyze game mechanics that successful commercial video games use to attract their large consumer base. We then describe how certain game mechanics can be paired with educational content to create a video game with a more seamlessly integrated educational curriculum. We use this approach to develop a video game that aims to teach computer programming, resulting in a pilot version of a playable, cross-platform game.

First we review a few popular video game genres and the game mechanics that help classify them. We define sandbox games and serious games as well as explore their use as tools of education and entertainment. We also briefly explain the importance of game engines to the game industry and to the development of educational games. Next, we briefly cover Unity, the game engine used to build the educational game discussed in this paper. We review the current state of educational games, as well as games that teach computer programming concepts. We describe the design and implementation of a hybrid educational video game that aims to teach fundamental programming concepts using an organized curriculum. We also discuss the design and implementation of a visualization tool that operates on layered abstraction to create a sandbox environment for teaching introductory programming concepts. To conclude, we share our pilot study findings and draw conclusions about the methods proposed in this paper.

Chapter 2

Background

2.1 A Taxonomy of Video Games

This paper argues that the educational video game designer should seriously consider utilizing game mechanics that best match the subject matter of the educational content. Many commercial games make frequent use of these game mechanics. In fact, many of these game mechanics help to define a video game taxonomy. Much like literature and film, video games have a history of being identified by established genres that continue to evolve as new works are produced. A list of video game genres based on interactivity has already been established [45]. In this chapter we will identify a few of these genres and discuss their defining qualities and mechanics.

Role-Playing Games

Role-playing games, or RPGs, rely on the importance of an intricate narrative. The player is drawn into the game by taking on the role of one or more characters. A common game mechanic in this genre is the ability to increase a character's skills or attributes to more effectively confront the antagonist. This genre places much emphasis on the projection of the player onto the game character. A popular device to achieve this is the ability to customize the name, look, sound, and persona of the player's character. In doing so the game character becomes an extension of the player's identity and allows the player to become more invested [17].

The narrative in this game genre is typically punctuated by battle sequences where the player must defeat enemies. There are two popular battle styles in this genre. Turn-based battle refers to a game mechanic that pits the player's character, or party, against the opposing forces in a turn-by-turn battle sequence. During the player's turn all action ceases until commands have been issued and the battle can continue. Real-time battle refers to a system without turns, where actions are carried out as they are issued. Examples of this genre include the popular Final Fantasy series [12] (see Figure 2.1), The Legend of Zelda [34] series (see Figure 2.2), and the Pokemon series [10].

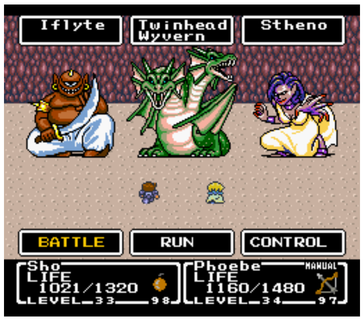


Figure 2.1: Turn-based battle system in Final Fantasy Mystic Quest

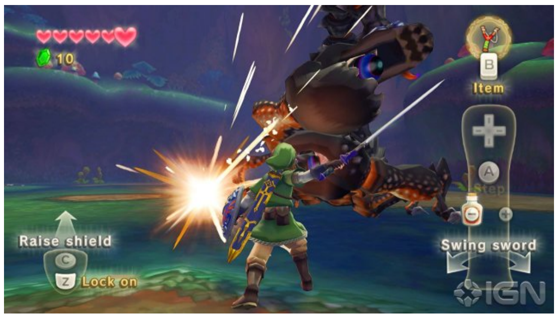


Figure 2.2: Real-time battle system in The Legend of Zelda: Skyward Sword

Action Games

Action games rely on the player's reflexes and controlled precision. Action games are often divided into incremental chapters called levels. Each level has a unique set of obstacles, collectable items, and weaponry. Levels may also end with a more challenging obstacle or enemy referred to as a boss. Typically the player is given a certain amount of health or lives that when depleted end the game. This happens to be the broadest genre encompassing many other subgenres including platform games, first-person shooters, fighting games, and racing games. For this reason, the action game is the predominant genre featured in home game consoles and in video arcades. Common iconic examples of this genre include the Super Mario Brothers series [32], the Sonic the Hedgehog series [37] (see Figure 2.4), and the Rayman series [44] (see Figure 2.3).

Figure 2.3: Boss battle action in Rayman Advance

Figure 2.4: Platforming action in Sonic the Hedgehog 4

Adventure Games

Some of the earliest video games invented hail from the Adventure games genre. Games in this genre are often text based and involve interaction that is not reflex based. Typically the player is presented with puzzles that are to be solved by interacting with the environment or with other game characters. The player is almost never given a time limit or presented with action based challenges, making this a genre that appeals to many non-gamers. True adventure games have suffered a fluctuation in popularity and as a result many of these games tend to be low budget. While this may be true in western economies, eastern cultures seem to fervently embrace the genre as visual novels, a subgenre of the adventure game, are extremely successful in the eastern market [39]. Popular adventure games include the Ace Attorney series [7] (see Figure 2.5), the Myst series [46], and Grim Fandango [28] (see Figure 2.6).

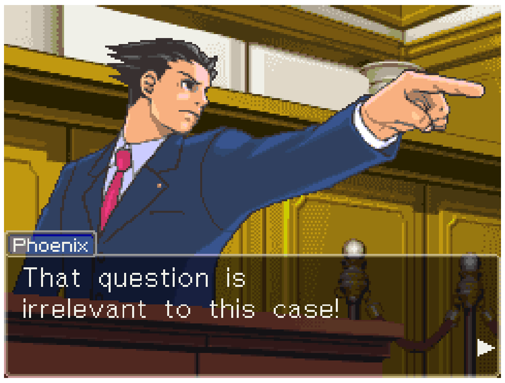


Figure 2.5: Text based adventure game Phoenix Wright: Ace Attourney

Figure 2.6: Point and click adventure game Grim Fandango

Hybrid Games

Many contemporary commercial video games make use of game mechanics from two or more traditional video game genres. These games are considered to be genre hybrid games. For example, the most recent games in the Castlevania franchise [25] are action platformers that incorporate a role-playing mechanic where game characters have skills and attributes that can be incrementally increased. Even though this game is a genre hybrid it is marketed as an action platformer due to the fact that the gameplay focuses primarily on the platforming game mechanic over the secondary role-playing game mechanics. Many video game companies with established franchises continue to extend them by releasing video games faithful to the traditional genre. However, for the sake of reaching new audiences it has become crucial to also expand successful franchises by investing in the production of hybrid video games [2]. Some hybrid video games become so successful that they eventually become independent franchises themselves. For example, the Mega Man franchise [6] is well known for it's platform action and unique game mechanics. Each boss enemy in Mega Man has a particular weapon that can be obtained by beating that enemy. Each enemy is weak to a particular weapon creating a game that exhibits mechanics similar to the rock-paper-scissors hand game. In the traditional Mega

Man series the game is played by jumping from platform to platform and shooting enemies to reach the end of the level, take on the boss, and collect the boss' weapon. Mega Man Battle Network, a recent addition to the Mega Man franchise, borrows the characters and game mechanics of the series and places them in the context of a role-playing video game. The player must control Mega Man and collect battle chips from enemies to incrementally increase the abilities of Mega Man. Certain enemies are weak to certain battle chips retaining the rock-paper-scissors mechanic. The game features a battle system that is a hybrid of the turn-based and real-time battle systems of the traditional role-playing video game genre. The original Mega Man Battle Network video game was so successful that several follow up titles have been released creating an entirely new franchise. Popular hybrid games include the previously mentioned Mega Man Battle Network franchise (see Figure 2.7) and the Paper Mario franchise [33] (see Figure 2.8).

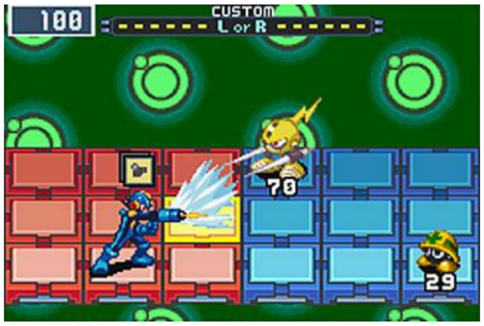


Figure 2.7: Action game mechanics from the role-playing game Mega Man Battle Network

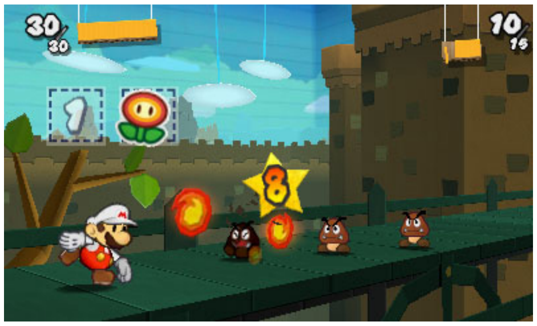


Figure 2.8: Action game mechanics from a turn-based battle in Paper Mario: Sticker Star

2.2 What is a Serious Game?

Video games that serve a purpose other than entertainment are considered serious games. These games do not form their own genre in the video game taxonomy. Types of serious games include educational, training, advertising, political, exercise, business, persuasive, military simulation, marketing, and medical games. While they are meant to serve another purpose, serious games can also be entertaining. The term was originally coined before the invention of the video game.

> Reduced to its formal essence, a game is an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context. A more conventional definition would say that a game is a context with rules among adversaries trying to win objectives. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement [1].

Many companies and schools use educational games, a subcategory of serious games, to train and teach [24]. The United States military invests in many training programs that utilize serious games, as video game development is cheaper and easier to access than special equipment and facilities [20]. Popular commercial serious games include Oregon Trail [9] (see Figure 2.9) and Wii Fit [31] (see Figure 2.10).

Figure 2.9: An educational game about pioneers on the Oregon Trail

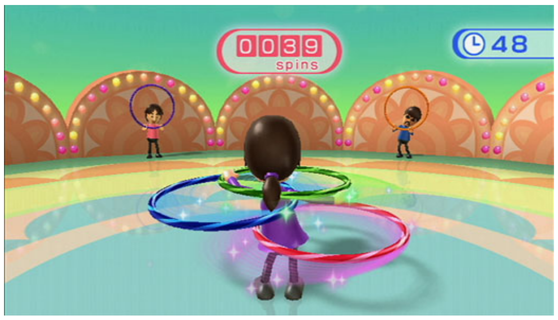


Figure 2.10: A calorie burning hula hoop mini-game in Wii Fit

2.3 What is a Sandbox Game?

A sandbox game is a game that allows the player to play with certain elements in the environment or interface with complete creative freedom. Like serious games, sandbox games do not form their own genre in the video game taxonomy. While playing a sandbox game, the player is allowed to be engaged without having to worry about whether they are playing the "right way." Typically, the player is given infinite health or lives and, like the role-playing genre, the player's character is usually highly customizable for more creative freedom. Popular commercial sandbox games include the LittleBigPlanet series [40] (see Figure 2.11) and Minecraft [29] (see Figure 2.12).

Figure 2.11: 4-player cooperation in the sandbox world of Little Big Planet

Figure 2.12: Exploring the sandbox environment of Minecraft

2.4 What is a Game Engine?

A game engine is a multifaceted tool used for the rapid production and deployment of video games. Game engines supply a framework for developers to create video games that run on personal computers and commercial game consoles. The features of game engines are varied but typically include integrated solutions for rendering, physics and collision detection, animation, networking, audio, artificial intelligence, memory management, scripting, and particle based visual effects [42]. It is often cheaper for a production studio to build or purchase a game engine for development, as multiple games can be built using the same engine. It is also possible, with some game engines, to develop games for multiple platforms, making it easier to reach a wider spectrum of video game consumers.

The Unity Game Engine

Unity [41] is a game engine that is not only used for games but for visualization as well. Unity is known for its cross-platform support featuring deployment to Windows, Mac, Xbox 360, Playstation 3, Wii, iPad, iPhone, and Android platforms. Developers using Unity can also deploy web-based games that run using Unity's web player plugin or Flash. What sets Unity apart from other game engines is that it includes a fully featured integrated development environment, equipped with property inspectors and live game preview. Programming languages supported by Unity include *JavaScript*, *C#*, and a *Python* derivative named *Boo*. Some of the features supported by the render engine include bump mapping, reflection mapping, shadow mapping, parallax mapping, and a shader language that supports fall back on alternative shaders, should the platform not be capable of utilizing a shader. The built-in physics engine is Nvidia's PhysX engine [35]. This engine supports rigid and soft body dynamics, character controllers, rag dolls, and simple cloth simulation. The professional version of Unity supports many advanced features, such as screen space ambient occlusion, lightmapping, global illumination, and pathfinding.

Chapter 3

Related Work

3.1 The State of Educational Games

While educational games do not often top the charts in popularity, many have experienced exceptional success in the video game market. Of these games, there seem to be two prevalent formulas for the creation of educational video games. One method takes a video game that is purely entertainment and injects educational content into the game. The other formula, most often used by educational game designers, takes educational content and shapes it into something that looks like a game.

Math Blaster [3] (see Figure 3.1) is a game that drills players with basic math problems to improve their math skills. Klopfer et al. [14] assert that while this educational game development formula may create relatively entertaining games it also has the potential to call the educational value of the game into question.

> If your spaceship requires you to answer a math problem before you can use your blasters, chances are you'll hate the game and the math [14].

Math Blaster does, however, incorporate a genre of gameplay similar to the nature of its educational content. The developers have paired fast paced mathematic drills with action based game mechanics. Re-Mission [18] (see Figure 3.2) makes use of the same action based shooting mechanics but differs in its educational content. The goals of the video game include increasing awareness about different types of cancer, educating the player on the proper nutrition to help fight cancer, and encouraging compliance with oral chemotherapy regimens. This educational content is more lecture based in nature and can be effectively communicated through the narrative without impeding on the action elements of the game.

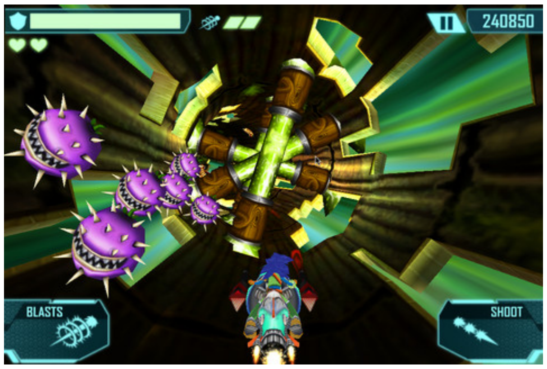


Figure 3.1: 3rd-Person Shooting Action in Math Blaster

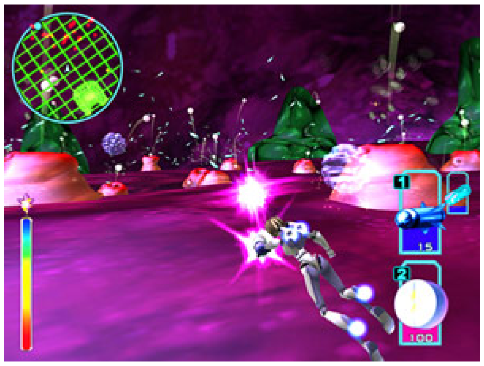


Figure 3.2: Shooting Lymphoma Cells in Re-Mission

One example of a game that is created by shaping educational content into a game is the Super Nintendo and MS-DOS game Mario is Missing [22]. This video game tries to present geography and history using the popular characters from the Super Mario Brothers series. The player controls Luigi as he journeys around the world answering questions (see Figure 3.3) about the geographic locations in which he is searching for his lost brother. At first, the game appears to be very similar to the popular Super Mario World. However, the game fails as an action platformer. While the side-scrolling and enemy bashing mechanics are somewhat present in the game they only serve as ornamentation to the educational content (see Figure 3.4). This shows how putting educational content into a game environment may have the potential to teach but does not often produce engaging games.

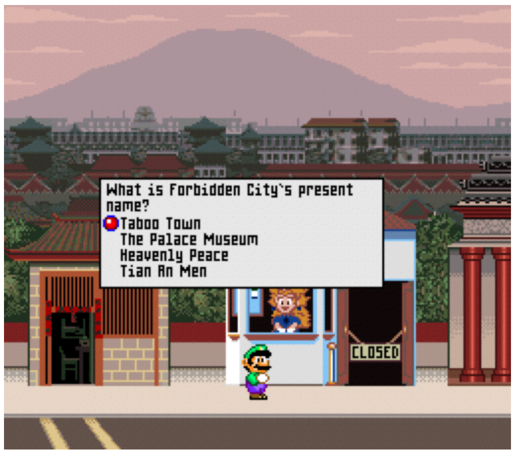


Figure 3.3: Geography and History Quiz questions in Mario is Missing

Figure 3.4: Uninteresting Platforming Action in Mario is Missing

A low budget seems to be a common factor in almost all educational games. However, video games do not need expensive graphics to be enticing. The Wii continues to outsell its competitors, the Playstation 3 and Xbox 360, [8] even though the graphics capabilities of the Wii are vastly inferior to those of its competitors [30].

3.2 Video Games that Teach Computing Concepts

Final Fantasy XII

Final Fantasy XII [11] is a role-playing game developed by the popular Japanese video game company Square Enix [13]. The twelfth game in the Final Fantasy series has introduced several particularly interesting game mechanics such as the gambit system. The gambit system allows players to program the behavior of the characters. In traditional Final Fantasy titles, the player uses a menu driven turn-based battle system to fight enemies. In Final Fantasy XII, the battle system is a hybrid of the real-time and turn-based battle systems. The player controls the game characters by issuing commands. The game character returns to an idle state after executing a command. During this idle state, the game character executes commands based on a set of predetermined rules called gambits (see Figure 3.5). For example, a player may decide to designate healing responsibilities to a certain character. The player would then assign a gambit that has the desired prerequisites, to execute a command when an ally's health is less than 70%, to the designated healing character. The player would then assign a command that acts as a parameter for the gambit such as cure (see Figure 3.6). When an enemy attacks a game character and lowers their health below 70%, the designated healing character will automatically execute the cure command. This rudimentary programming mechanic allows the player to experiment with conditional expressions.

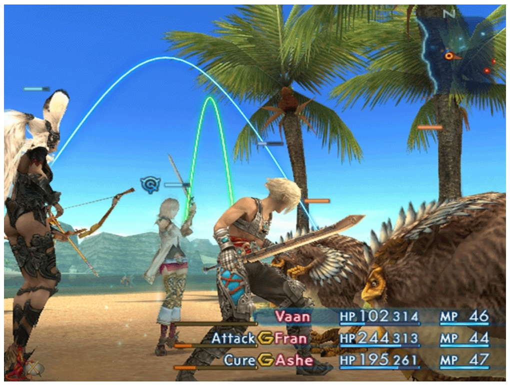
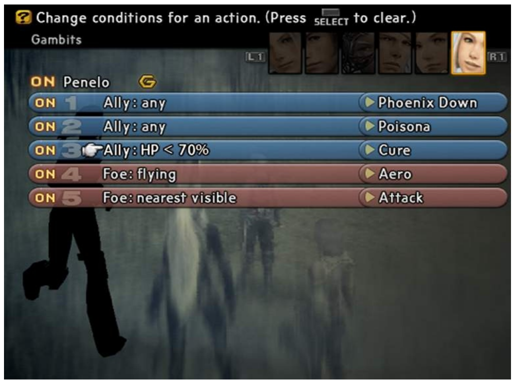


Figure 3.5: Battle Commands Being Executed by Gambits in Final Fantasy XII


Change conditions for an action. (Press SELECT to clear.)
Gambits
L1 R1
ON Penelo

ON 1	Ally: any	Phoenix Down
ON 2	Ally: any	Poisona
ON 3	Ally: HP < 70%	Cure
ON 4	Foe: flying	Aero
ON 5	Foe: nearest visible	Attack

Figure 3.6: Using Gambits to Execute Battle Commands in Final Fantasy XII

Light-Bot

Light-Bot [16] and Light-Bot 2.0 [15] are programming puzzle games that task the player with writing small programs that control the movement of a robot. In Light-Bot, the player is presented with a tool box of simple commands for the robot to obey. There is a main function that the player can drag commands into. The robot steps through the commands in order, just as a program steps through instructions. The goal is to get the robot to specially marked cells on the playing field and to use the light command (see Figure 3.7). Sometimes the player must light up several cells on the playing field, which can be difficult with a small amount of allotted space for commands. In these cases, the player is also allowed to use two subroutines.

These games introduce many rudimentary programming concepts such as modular implementation and recursion. Light-Bot 2.0 adds new game mechanics, such as commands that are only executed if the robot is a certain color. This mechanic makes conditionals possible.

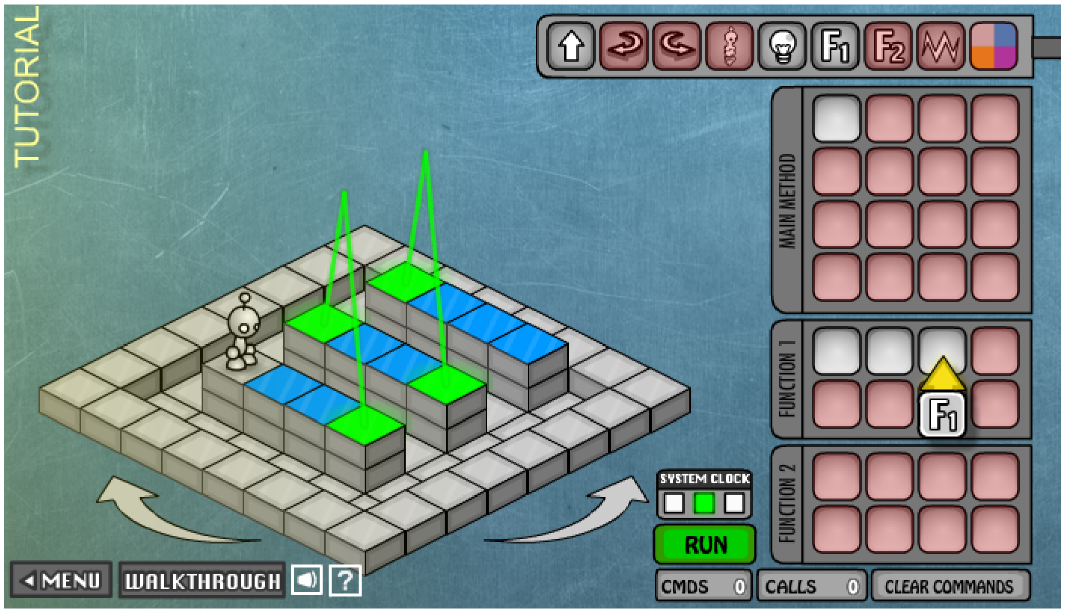


Figure 3.7: Programming puzzle in Light-Bot 2.0

Kodu

Kodu [26] is a sandbox game that allows the player to create their own game using a visual programming language. It was created to be accessible to people of all ages by substituting textual code for code that is built using the game controller and a rule-based language that focuses on sensory conditions (see Figure 3.8). Many high level programming conventions are absent including branching, number and string manipulation, subroutines, polymorphism, compiling, etcetera. The environment of the game is a nearly complete virtual environment, allowing players to manipulate the scenery using a visual editor (see Figure 3.9). The characters of the game are programmed by the player using sensory conditions such as the sequence "See - Fruit - Move - Towards." In this sequence the character moves towards the fruit when the fruit comes into its line of vision. Additionally, modifiers can be used to more precisely control the behavior of the programmable characters. Due to the sandbox nature of this tabula rasa environment, many different game genres can be modeled such as role-playing, action, adventure, puzzle, and platformers.

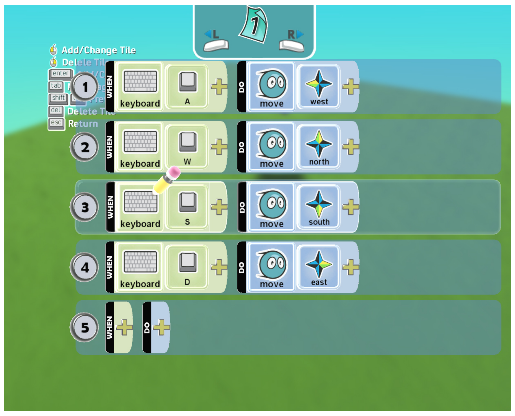


Figure 3.8: Programming Using a Sensory Based Language in Kodu

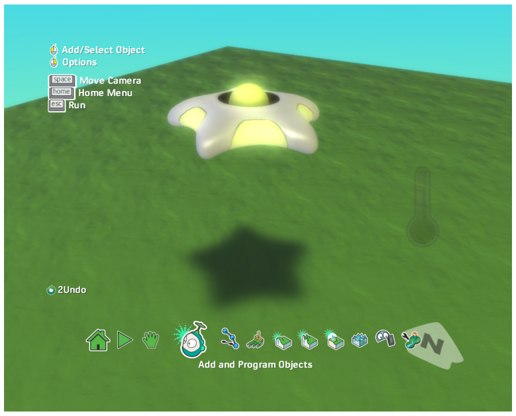


Figure 3.9: A Programmable Object in the Game Environment of Kodu

Code Hero

Code Hero [27] is a first person shooter game that aims to teach *Javascript* and *Unityscript* 3.10. The game is currently undergoing beta testing before its release. In Code Hero, the player is armed with a code ray that is capable of copying code, editing copied code, and shooting it at targets 3.11. The game presents programming topics in lessons organized into a curriculum. In order to progress in the game you must complete programming lessons that are explained using examples within the context of the game world. Often failure leads to a better understanding of concepts and ultimately leads to success. In Code Hero, players are expected to fail a lesson several times before they eventually master the concepts being exemplified in the lesson. The game does not, however, attempt to teach basic programming concepts through the paradigm of a simple machine. Instead, Code Hero focuses on high-level programming languages.

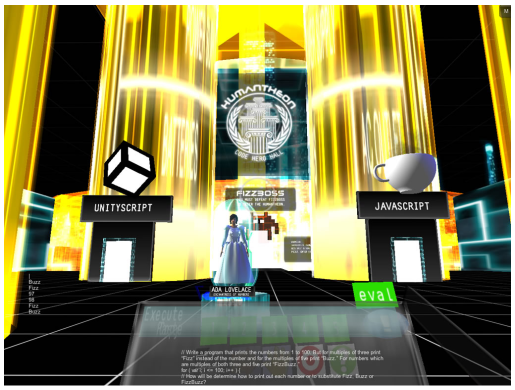


Figure 3.10: Choosing Between Unityscript and Javascript Tutorials in Code Hero

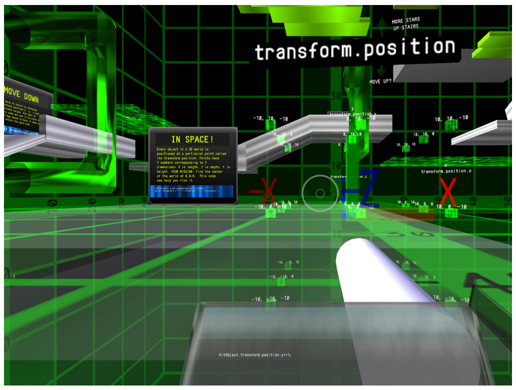


Figure 3.11: Shooting Code From the Code Ray in Code Hero

Chapter 4

Methodology

Exploiting the genre hybrid video game model in the construction of educational video games has many advantages. These games are naturally more engaging in their ability to present old game mechanics in a new and interesting way. Additionally, genre hybrids can use the game mechanics of one genre as a tool to weave the educational content into the game mechanics of another genre. Just as many genre hybrid games focus on a particular genre identified by the primary game mechanics, the game developer should choose a set of primary game mechanics that fit within the context of the educational content. For example, an educational game whose objectives include rapid problem drilling may find that it is easier to incorporate the educational content into the video game by making the primary game mechanics action oriented. The developer should then consider ways in which secondary game mechanics can be used to shape the concepts of the educational content into entertainment oriented gameplay.

In the design of our educational video game, we chose to have our primary game mechanics draw from the role-playing genre. We found this to be an appropriate match for a curriculum driven video game due to the quest-like nature of a predesigned set of increasingly difficult problems. Programs are constructed using a visualization tool that serves as a sandbox environment for the player to learn and experiment. We use adventure oriented game mechanics to communicate programming concepts through text based interaction with non-playable characters and the environment. Finally, we implement a hybrid battle system that uses both turn-based and real-time battle mechanics. This implementation allows the player to use programs written in the sandbox environment to help defeat the enemies.

4.1 Narrative and Visual Style

It is difficult to produce a large amount of photorealistic assets with a small team of developers. While there are industry professionals who believe narrative cannot be effectively communicated without photorealism [5], many video game consumers use the success of text-based adventure video games to rebut this claim [19]. For these reasons we chose to avoid photorealism in our artistic style. Instead, we drew inspiration from the main franchises of Nintendo, a company whose franchise sales [8] provide further proof that photorealism is not a requirement for engaging narratives in video games. Our characters and environments employ simple shading and whimsical designs that revolve around a digitally themed world.

The context of the educational content can help direct the educational game developer's artistic decisions. In our game, the player is asked to help create programs so it seemed natural to place the player inside the imaginary world of a computer. The opening scene introduces the CPU, who watches over the world inside of the player's computer (see Figure 4.1). The CPU informs the player of the dangerous viruses who have started to appear inside the world of the computer (see Figure 4.3). The player assumes the role of a digital knight (see Figure 4.2) who is charged with finding out why the viruses have been appearing and helping the inhabitants of the computer, the process bots, carry out their programming tasks along the way.

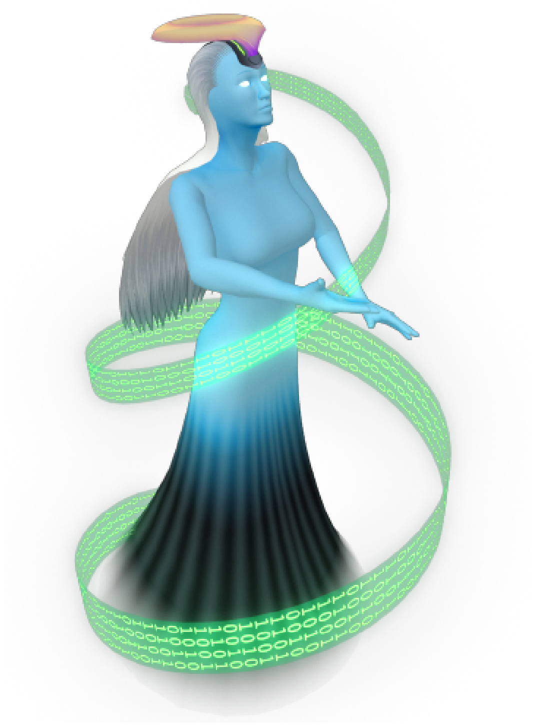


Figure 4.1: The central processing unit who watches over the computer world

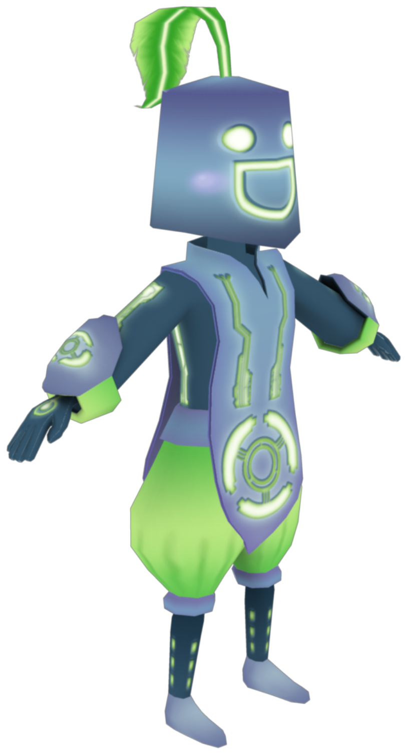
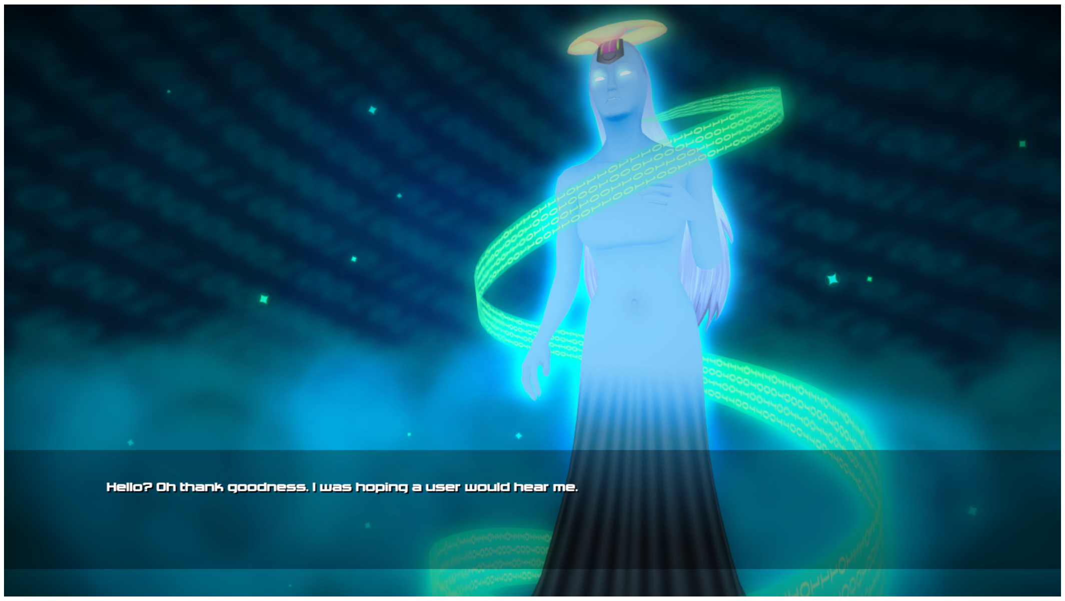


Figure 4.2: The digital knight controlled by the player



Hello? Oh thank goodness. I was hoping a user would hear me.

Figure 4.3: The CPU addressing the player in the introductory scene

4.2 The Architecture Simulator

The computer architecture simulator, shown in Figure 4.4, is an abstracted version of a simple computer architecture that serves as a visualization tool. There are four registers (D in Figure 4.4) whose values are initially unknown to the player. These values can be manipulated by using a set of five instructions: *input*, *output*, *assignment*, *if*, *else*, and *while*. As the player selects instructions from the instruction window (B in Figure 4.4) a program is assembled in the code window (C in Figure 4.4). The player can choose to either execute the program in its entirety or iterate through the instructions line by line using the *run* and *step* buttons (H in Figure 4.4). When the *input* instruction is executed, a keyboard appears allowing the player to provide an input value. The *assignment* instruction allows the player to assign values or expressions to a register. The *if* and *while* instructions allow the player to utilize branching using boolean expressions. The *else* instruction can only be used directly after an *if* code block. While the instructions are being executed, the data flow is illustrated by sparks that travel along wires that connect the code window to the registers. When output instructions are executed, the data flow is illustrated by sparks that travel along a wire connecting the registers to an output window (E in Figure 4.4). The speed of the computer and data flow visualization can be changed using a sliding bar (G in Figure 4.4) below the output window. A collection of buttons allowing the player to clear the code window, clear the output window, and reset registers are located underneath the output window. If the architecture simulator encounters an error the execution will halt and an error message will be displayed in the notification bar (A in Figure 4.4) above the instruction window. When the player is satisfied with a program, the exit button can be used to return to the game environment. When the player first encounters the architecture simulator, or terminal as it is referred to in the game, a process bot explains the interface by introducing the player to the various components of the visualization tool.

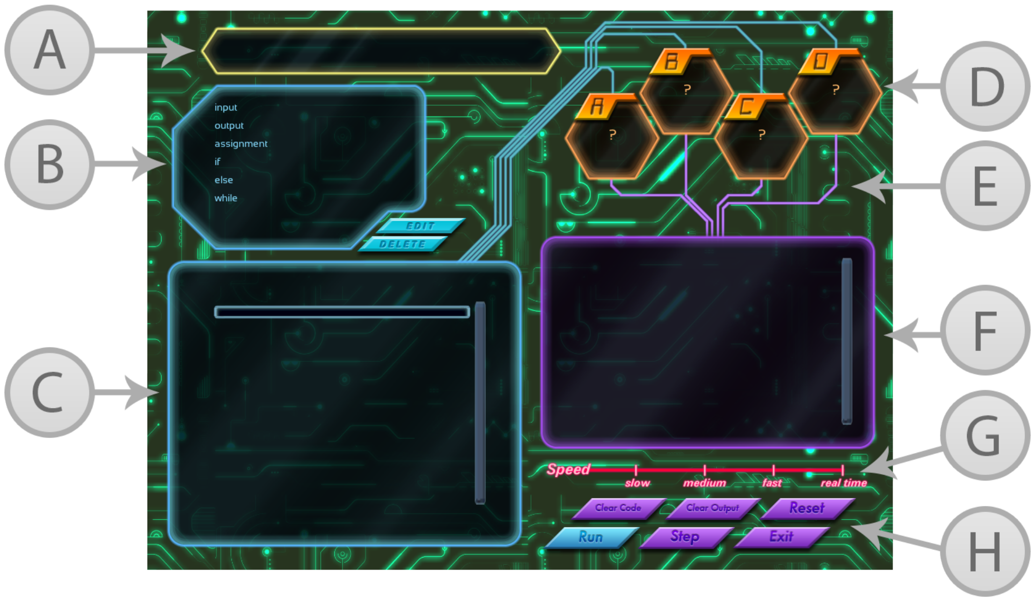


Figure 4.4: Breakdown of the visual elements in the Architecture Simulator

4.3 The Game Environment

The world inside the computer is divided into sectors that are comprised of a series of platforms. Each sector is connected to the central hub (see Figure 4.5). The terrain of each sector changes to give each sector a “world”-like identity. This is a common mechanic of the action platformer genre that acts as a visual cue to signal that the player has reached the next level. We use this mechanic to signal that the player has reached a new unit in the educational curriculum. Like many role-playing games, the player navigates the game environment from a third person perspective. The player cannot fall off the platform and is unable to jump. The architecture simulator, otherwise known as the terminal, can be found hovering over a predetermined location in each sector. Process bots can be found hovering over different areas of the platform. Some process bots are used to communicate programming information while others are used as an outlet for comic relief. A special process bot trades flops, the computer world currency, for items and upgrades to help the player in their quest against the viruses.

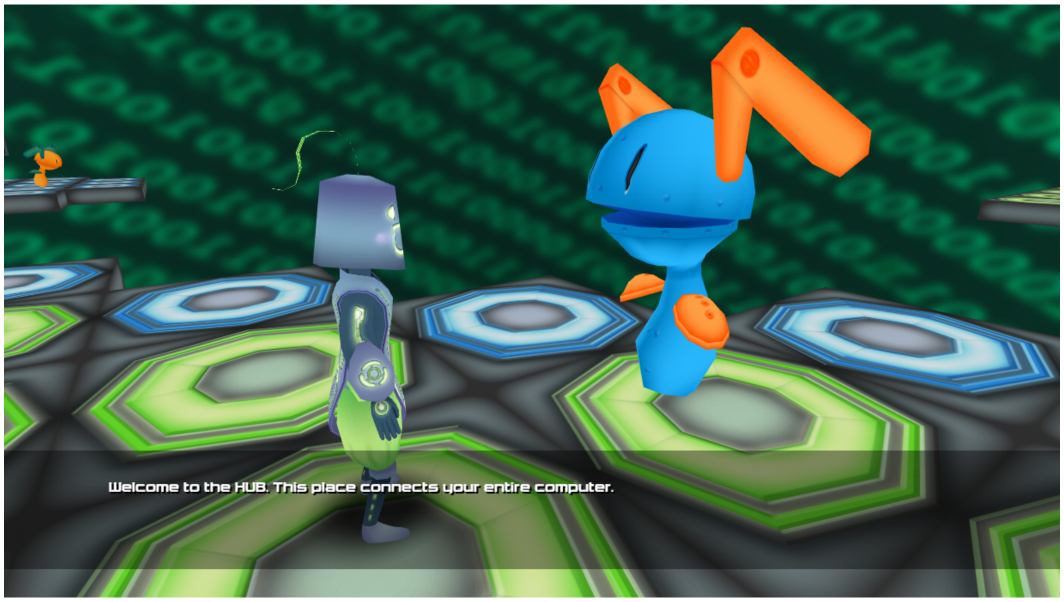

Welcome to the HUB. This place connects your entire computer.

Figure 4.5: Talking to a Process Bot on the central HUB

4.4 Programming Curriculum

Each sector is constructed such that the player's interactions with the process bots are loosely directed. This way, the developer can communicate information sequentially, allowing a lesson oriented curriculum to be easily incorporated. The curriculum is organized into a series of programming puzzles, as shown in Table 4.1, that progressively explore a basic set of programming skills. For example, the player learns about the input instruction from a process bot in the central hub before entering the first sector. The process bot closest to the entrance of the first sector explains the assignment instruction and also hints at the fact that swapping two numbers requires three memory registers. The player now has enough information to solve the *Swap 2* programming puzzle.

Table 4.1: Programming Puzzles and Rewards

Programming Puzzle	Reward for Completion
Input & Output	Unlock entrance to Jungle Sector
Reverse (2)	RAM Upgrade
Swap 2	Swap Macro
Output Biggest (2)	RAM Upgrade
Output Biggest (3)	Jump Macro
Select Biggest (2)	RAM Upgrade
Select Biggest (3)	Max-2 Macro
Sum (2)	RAM Upgrade
Sum (3)	Sum Macro
Sum (fixed n)	RAM Upgrade
Sum (variable n)	Blaze Command
Sum (to sentinel)	Unlock entrance to Deep Jungle Sector
Sort (2)	FLOPs Reward
Sort (3)	Sort Macro
Select Biggest (fixed n)	FLOPs Reward
Select Biggest (variable)	Max-3 Macro
Select Biggest (sentinel)	Unlock entrance to Jungle Sector Boss

4.5 The Battle System

Viruses are encountered randomly while walking around on the platform of the game environment. When a virus is encountered, gameplay is shifted from the exploratory game environment to an arena environment where the player must defeat viruses (see Figure 4.6).

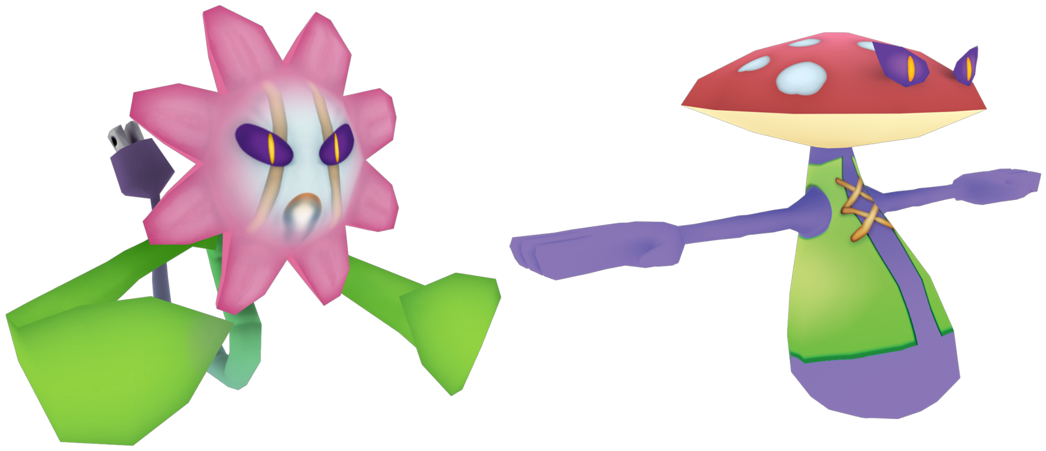


Figure 4.6: Viruses that the player must defeat during battle

The Battle Disk

The battle disk is a circular grid, representing the hard disk of a computer, on which the player battles viruses. Each character occupies a space on the grid with no two characters ever occupying the same space (see Figure 4.7). Viruses move around the grid and try to get close to the player. When a virus is on a space adjacent to the player, the virus is able to attack and lower the player's vitality. Likewise, a player can deplete the vitality of adjacent viruses by pressing the space key to attack with a debugging sword. The debugging sword is not a very effective weapon by itself, so the player must use battle commands to defeat viruses.

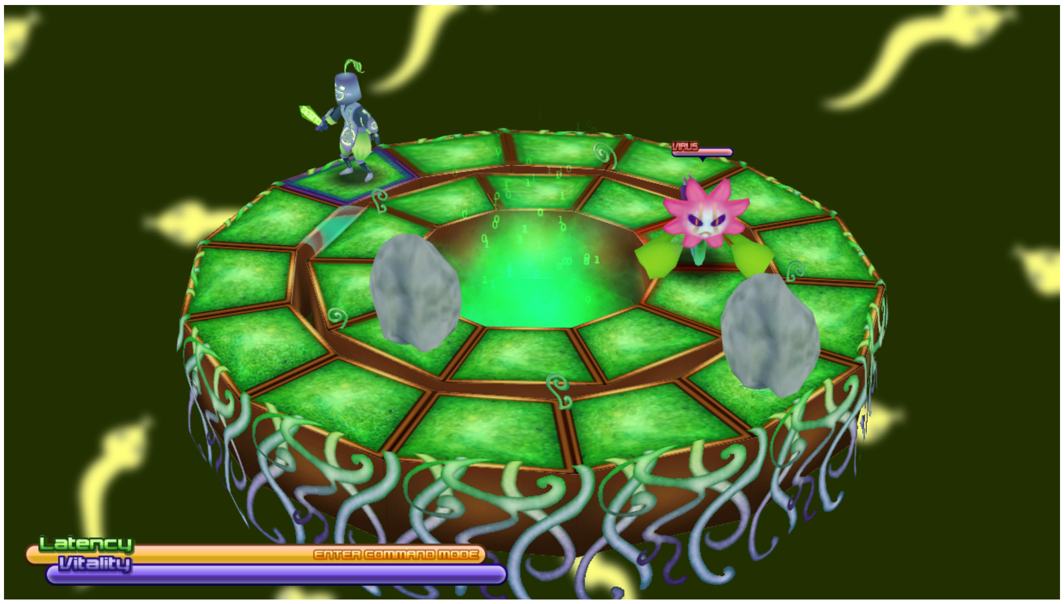


Figure 4.7: Battling a Virus on the Battle Disk

Command Mode

Command mode can be activated by pressing the return key when the player's latency meter is full. While command mode is active, the battle is paused, allowing the player to execute a more powerful command. During command mode, the player is presented with a queue of commands (see Figure 4.8). Table 4.2 lists all of the available commands in the game. Each command has an associated strength level denoted by a number in megabytes. The player can choose to either execute the command at the beginning of the command queue or manipulate the order of the commands in the queue using special functions called macros. The player is only allowed to manipulate a command if the appropriate RAM slot has been unlocked. If a macro is used to manipulate the command queue the player is unable to execute a command until the latency meter is full once again.

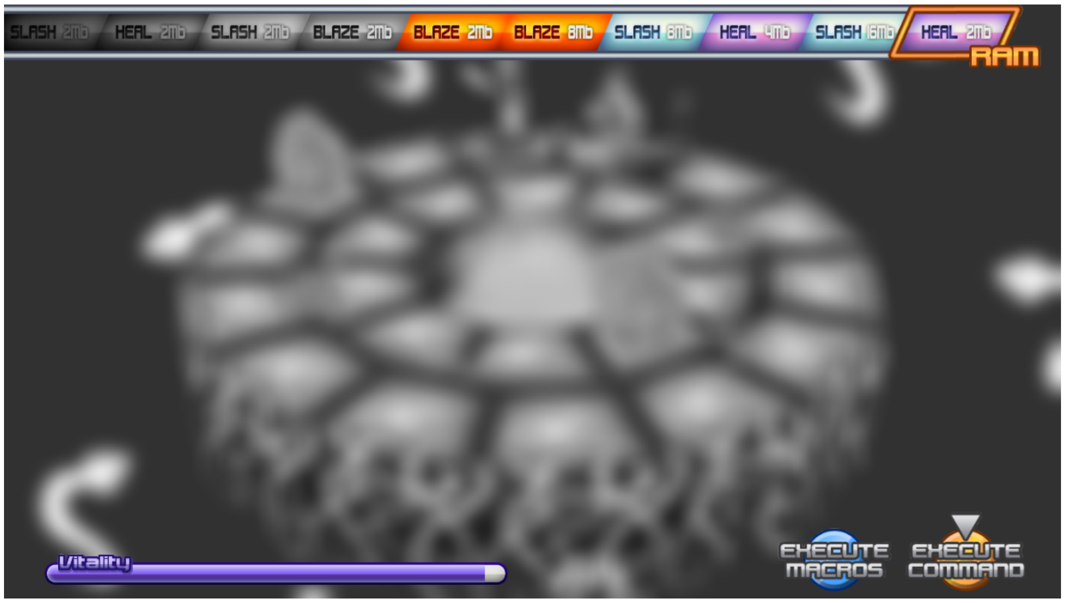
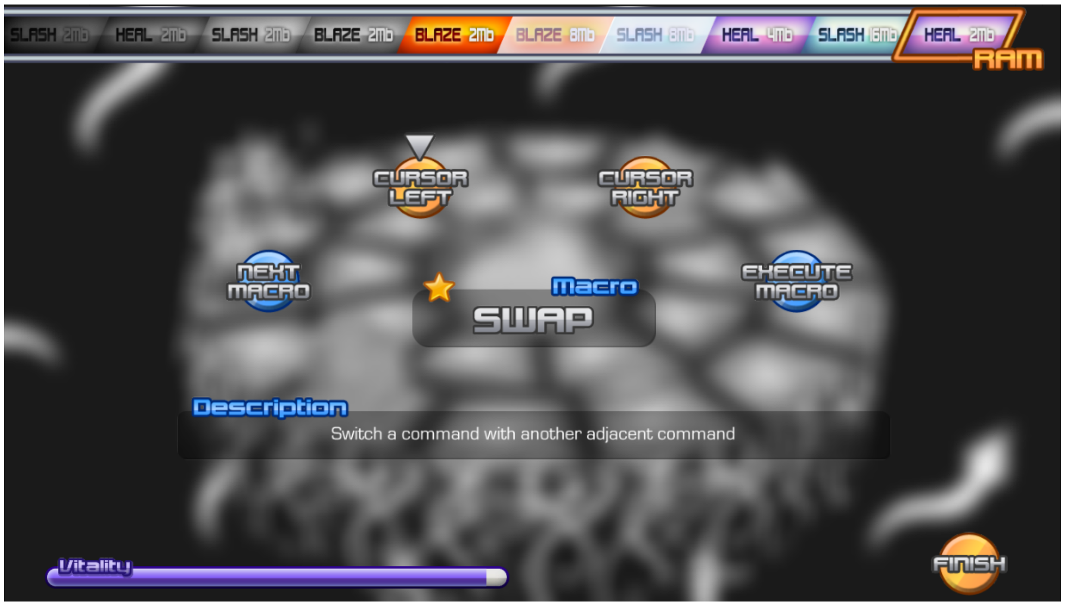


Figure 4.8: A variety of Commands in the Command Queue

Macros

The player unlocks macros by completing programming puzzles for the process bots. This concept ties the educational content to the battle system, giving the player motivation to complete as many programming puzzles as possible. Once a macro is unlocked, it can be used to manipulate the command queue during command mode (see Figure 4.9). When a macro is used, the latency meter takes more time to fill up during the next battle segment. Each macro has an associated complexity, as shown in Table 4.3, denoted by a number of star emblems. Simple macros have one star emblem, while the most complex macros have three star emblems. As more star emblems are used to manipulate the command queue, the speed of the latency meter will drop during the next battle segment. The player's goal is to use the least amount of macros to achieve the best outcome.



Switch a command with another adjacent command

Figure 4.9: Using the Swap Macro to swap two Commands in the Command Queue

Table 4.2: Available Commands

Command	Description	Strength Levels	Formula
Slash	Forcefully slash an enemy on the panel(s) in front of the player	2mb, 4mb, 8mb, 16mb, 32mb, Giga	Basic Command
Heal	Increases the player's vitality	2mb, 4mb, 8mb, 16mb, 32mb, Giga	Basic Command
Blaze	Creates a fire on the panel(s) in front of the player	2mb, 4mb, 8mb, 16mb, 32mb, Giga	Basic Command
Ignite	Creates a fire that travels around the battle disk	2mb, 4mb, 8mb, 16mb, 32mb	Blaze + Slash
Siphon	Creates a vortex that drains enemy vitality on the panel(s) in front of the player	2mb, 4mb, 8mb, 16mb, 32mb	Heal + Slash
Vigor	Increases the speed of the player's latency meter	2mb, 4mb, 8mb, 16mb, 32mb	Blaze + Heal

Table 4.3: Available Macros

Command	Complexity	Description
Swap	1 Star	Swaps two adjacent macros in the command queue
Jump	2 Stars	Swaps any command in the queue with the command at the front of the queue
Max-2	2 Stars	Maximizes the strength of a command to match the strength of an adjacent command
Sum	2 Stars	Combines two commands to make one powerful command
Sort	3 Stars	Sorts the command queue by strength
Max-3	3 Stars	Maximizes the strength of two commands to match the strength of an adjacent command

Chapter 5

Study

We conducted a pilot study to provide a preliminary evaluation of our design approach. We asked 31 high school juniors from the Clemson University Emerging Scholars program to volunteer one hour of their time to help evaluate our educational game. The test population, consisting of both male and female participants, was broken into two groups for Emerging Scholar classroom sessions. To assess improvement in programming skills, we composed a programming survey, as shown in Appendix A, that was administered before and after participants played the game. To gain some insight into the gameplay experience of the participant, a game experience questionnaire, as shown in Appendix A, was administered after the participants were instructed to play the game. This allowed us to determine what elements of the game need improvement as well as to identify the elements of the game that players enjoyed. While our ideal test population would be a group of individuals who are interested in learning programming, our actual test population consisted participants who were not necessarily interested in learning programming. One important issue that arose during testing was cross-platform performance. Due to the way the game handled player movement at lower frame rates on some platforms, some participants in the first test group experienced sluggish player movement. These issues were addressed by making minor tweaks to the code before testing the second group of participants.

5.1 Programming Survey

The programming survey included a basic explanation of instructions as well as an example of how they are used syntactically. The questions were formed around educational learning objectives outlined in *Bloom's Taxonomy* [4], allowing us to create an instrument that is able to evaluate learning based on a variety of cognitive skills. We are concerned with knowledge, comprehension, and critical thinking in foundational computer programming concepts. These ideas fall under the *Cognitive Domain* of the taxonomy, which features six levels of cognitive skill. We were only able to utilize five of the six levels in the taxonomy (see Table 5.1). Four of the five questions in the survey directly reflect programming puzzles outlined in the curriculum of the game.

After analyzing the data from the programming surveys, we noticed that most participants were unable to answer the free response and fill in the blank questions. This makes sense as these questions involve the use of cognitive skills from the more difficult levels of *Bloom's Taxonomy*: *Synthesis* and *Analysis*. Four participants were able to correctly answer the first question after playing the game for an hour. Before playing the game, only one participant was able to recognize that the else branch was executed in question 4. After playing the game for an hour, two participants were able to recognize that the else branch was executed in question 4. The distribution of answers to multiple choice questions collected from participants before playing the game, as shown in Table 5.2, show no indication of prior programming knowledge. When compared with the distribution of answers to multiple choice questions collected from participants after playing the game, as shown in Table 5.3, there seems to be an overall improvement. Results from question 2 had the most improvement with 3 students answering correctly before playing the game and 9 students answering correctly after playing the game. Results from questions 3 and 5 showed minor improvement.

5.2 Game Experience Questionnaire

The game experience questionnaire posed questions regarding perception of programming skill, overall reactions to gameplay, difficulties experienced, and suggested improvements. All questions were free response, requiring the participants to write their answers in a few short sentences. The data was evaluated using a qualitative analysis technique, known as *keyword analysis*, where common keywords and phrases are identified and organized into common themes. Assertions could then be made based on the number of participants whose responses contained keywords associated with a common theme. Results from the game experience questionnaire are shown in Table 5.4.

After analyzing the data from the game experience questionnaire, we found that the majority of respondents felt they had no programming experience prior to playing the game. Nearly half of the respondents said the game improved their programming knowledge. When asked about their programming skills after playing the game, one respondent answered saying “they improved and I got a better feeling for the game.” Overall, the responses indicated mixed emotions about the game. Some respondents indicated a purely positive reaction, while others indicated a purely negative reaction to the game. One respondent answered saying the game was “easy and fun once I figured out what was needed.” The majority of respondents referenced programming related elements as obstacles. For example, one respondent said one of the biggest obstacles was “trying to figure out the terminal.” The majority of respondents recommended improving the instruction by adding hints and examples. One respondent said the game should have “examples in the tutorials.” Several respondents recommended adding features to the game that were in the design documentation but never made it to the pilot version. For example, one participant mentioned adding the ability to customize the look of your character. Another participant recommended adding shops where the player is able to purchase upgrades. When asked about improvements that could be made to the game, a third of the participants said they would like to see less bugs, such as the previously mentioned sluggish player movement.

Table 5.1: Cognitive Skills Used in the Programming Survey

Question	Response Type	Cognitive Skills
1. Please write a program that takes two inputs (to registers A and B) and outputs the values in the opposite order that they were input.	Free Response	Synthesis
2. What instructions would you use to write a program that reads two numbers and outputs the largest number?	Multiple Choice	Comprehension
3. Please circle the program that takes 2 numbers as input and swaps them.	Multiple Choice	Knowledge
4. Consider the following program. What will the value be in each register after the program is run? What will the output be?	Fill in the Blank	Analysis & Application
5. Please circle the program that keeps track of the largest input number. The program should assume that the input numbers are arbitrary and it should terminate when a negative input value is read.	Multiple Choice	Knowledge

Table 5.2: Pre-Gameplay Programming Survey Results

Question	Correct Answer	A	B	C	D
2	C	6	11	3	10
3	B	15	4	12	0
5	C	2	12	4	12

Table 5.3: Post-Gameplay Programming Survey Results

Question	Correct Answer	A	B	C	D
2	C	1	11	9	7
3	B	18	5	3	2
5	C	2	8	7	10

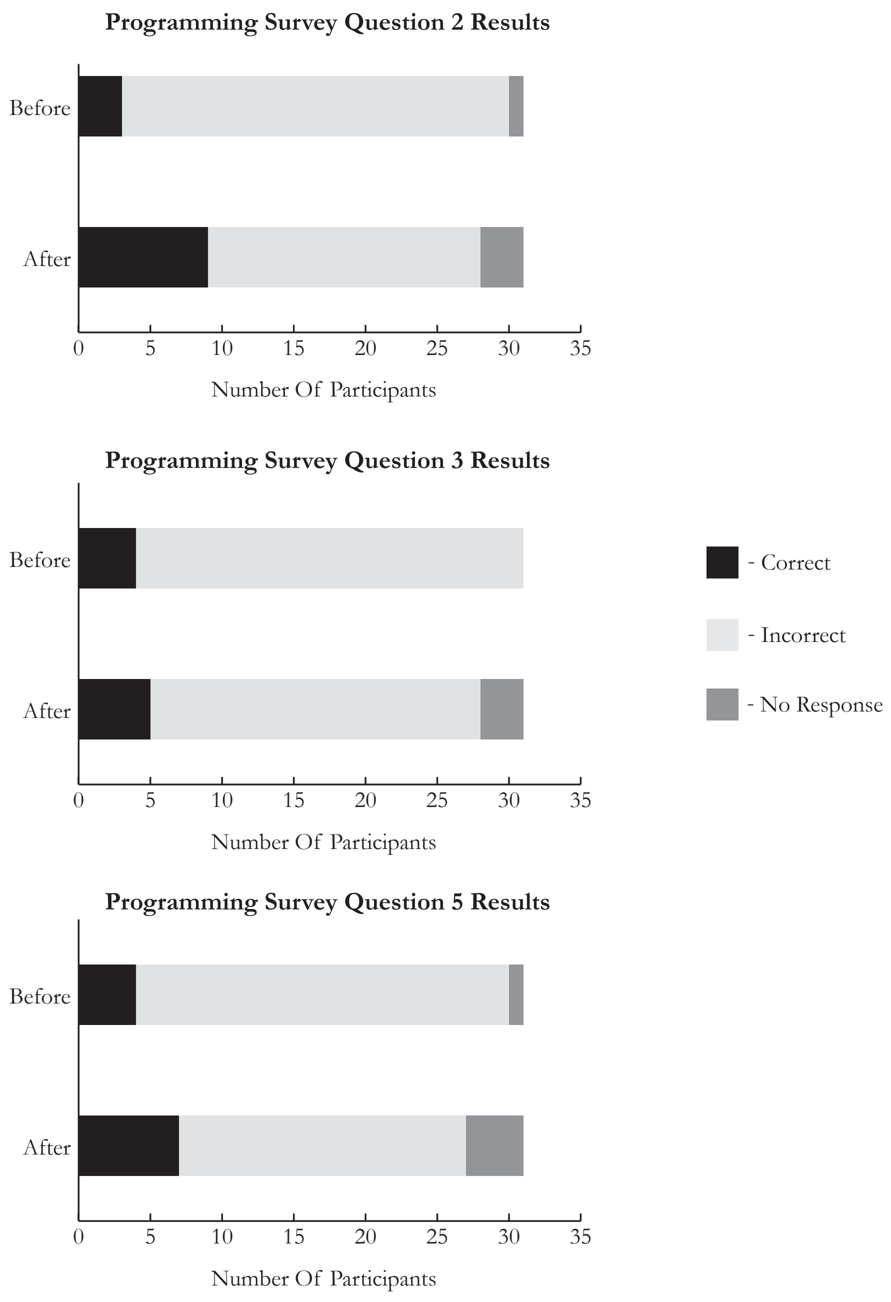


Figure 5.1: Programming survey results diagrams

Table 5.4: Game Experience Questionnaire Results

Question	Theme	Respondents who identified with theme
1	No Prior Experience	21 out of 26 respondents had little to no prior programming experience
2	Improvement	12 out of 26 respondents said the game improved their programming knowledge
2	No Improvement	7 out of 26 respondents said the game did not improve their programming knowledge
2	Some Improvement	2 out of 26 respondents said the game helped their skills but they are still confused
3	Positive Reaction	7 out of 29 respondents had a positive reaction to playing the game
3	Negative Reaction	12 out of 29 respondents had a negative reaction to playing the game
3	Mixed Emotions	9 out of 29 respondents had mixed emotions (i.e. Difficult, but fun)
4	Game Related Obstacles	9 out of 27 respondents classified game related elements as obstacles
4	Programming Obstacles	15 out of 27 respondents classified programming related elements as obstacles
5	Instructional Improvements	16 out of 28 respondents said the game could benefit from more or improved instruction
5	Feature Improvements	5 out of 28 respondents said they would like to see additional game features
5	Less Glitches	9 out of 28 respondents recommended addressing glitches

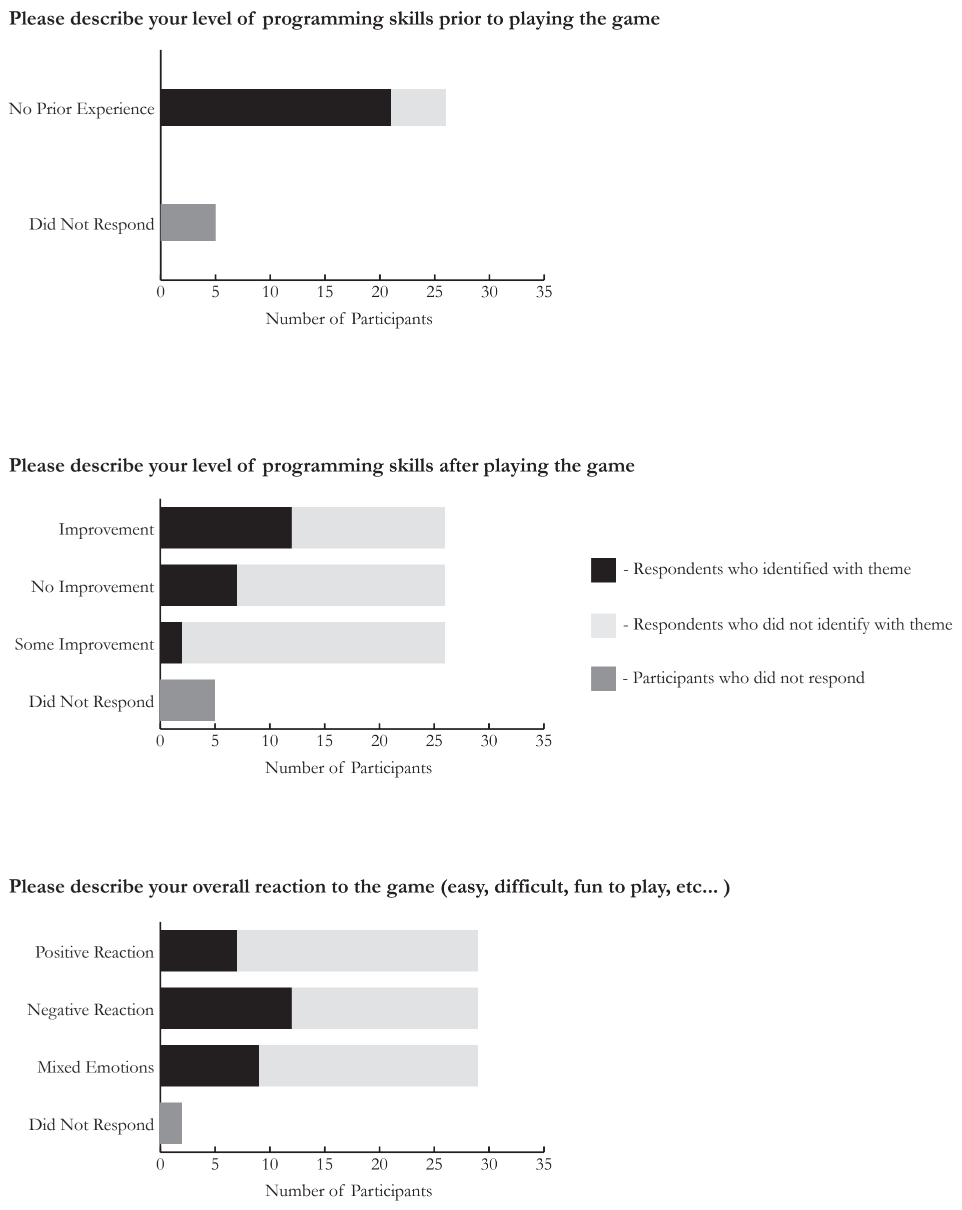


Figure 5.2: Questionnaire keyword analysis results diagrams

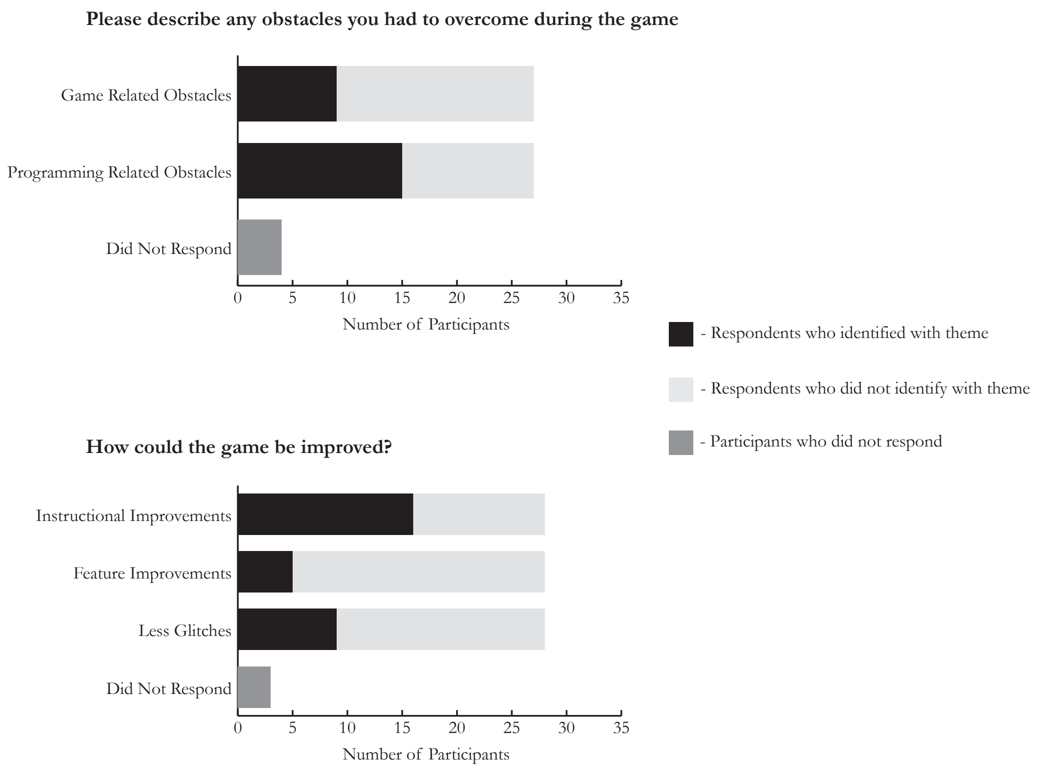


Figure 5.2: Questionnaire keyword analysis results diagrams

Chapter 6

Conclusion

We have described an approach to educational video game design that pairs mechanics from several game genres with various elements of a predesigned curriculum. We have shown that intelligently pairing educational content and interfaces with similarly natured game mechanics results in a more cohesive and engaging educational experience. Our pilot study results showed that students who played our educational video game for one hour were able to improve their computer programming skills. While the results of our pilot study provide motivation for further research, there are many aspects of our approach that can be improved upon.

We were reminded that while students enjoy the entertaining aspects of educational games, their experiences are shaped by both the entertaining and the educational elements. If the educational elements lack a thorough level of explanation, even in sandbox environments, the entertainment value suffers. A few mechanics that could have been implemented to provide a deeper understanding of the visualization tool are help buttons and sample programs. A help button would have been a useful way to remind the player of the goal of the current programming puzzle. If the player could see several example programs being executed in the visualization tool, with a thorough explanation of both the execution cycle and the data flow, it would make the sandbox environment seem less daunting. Research in the area of program validation could offer interesting ways to provide the player with feedback. By evaluating the player's program, the developers could offer new and interesting scoring mechanics as well as offer programming hints from within the visualization tool interface.

From a developers standpoint, there are a few problems that we would address if we were to attempt a second implementation. We started our development by establishing a video game design document. As we started to develop a more complete review of available game design literature we realized that our design document would need to be be changed. With each new game mechanic revision it became difficult to maintain the documentation. If we were to attempt a second implementation we would spend time reviewing literature on video game documentation methodology and establish a more feasible documentation protocol.

The pilot version of our educational video game only includes the first unit in our eight unit fundamental programming curriculum. If we were to attempt to implement additional units we could run into design conflicts. Our current methodology operates on the idea of fitting the educational content within the context of the game mechanics. When the educational content adds an additional layer of complexity, the gameplay mechanics must also change. For example, if we were trying to communicate the concept of tree data structures we would have to implement a gameplay mechanic that would communicate this concept better than the battle system's command queue. Successful implementation of additional units would require further research into gameplay mechanics that effectively fit within the context of each additional unit's educational content.

Another idea that could stand further investigation is the value of the visualization tool as a standalone supplement to introductory programming curricula. The sandbox nature of the visualization tool allows it to be used outside the context of the video game. This could provide students with a familiar classroom environment and provide educators with a useful standalone instructional aid.

Appendices

Appendix A

Programming Survey and Questionnaire

Part I – Programming Survey

This survey is designed to test the effectiveness of the video game's instructional methods. We do not expect you to answer every question correctly. Therefore, please do not be discouraged if you are unable to answer a question.

For questions 1 through 5 assume that you have four registers (A, B, C, and D) and each register has an initial value but it is unknown to you. Please write the simple programs using the following instructions.

Instruction	**Explanation**	**Example**
Input	Takes 1 register as a parameter	Input A
Output	Takes 1 register as a parameter	Output A
Assignment	Assigning a value to a particular register	A = B
If	Set of instructions executed under a condition	If(A = B) { … }
Else	Set of instructions executed in the event of an if statement's condition being evaluated as false	Else { … }
While	Set of instructions executed until a condition is considered true	While (A>0) { … }

1. Please write a program that takes two inputs (to registers A and B) and outputs the values in the opposite order that they were input.

2. What instructions would you use to write a program that reads two numbers and outputs the largest number?

a. Assignment, Output
b. Input, Output
c. Input, Output, If
d. Input, Output, While

3. Please circle the program that takes 2 numbers as input and swaps them.

a.
```
Input A
Input B
A = B
B = A
```

b.
```
Input A
Input B
C = A
A = B
B = C
```

c.
```
Input A
Input B
B = A
A = B
```

d.
```
Input A
Input B
C = B
A = C
B = A
```

4. Consider the following program. What will the value be in each register after the program is run? What will the output be?

```
A = 5
B = A + A
If ( A>B ){
    C = B + (2*A)
    D = 6
}
Else {
    C = A + 6
    D = (2*B) + A
}
While( D>0 ){
    Output D
    D = D - B
}
```

Register A	
Register B	
Register C	
Register D	
Output	

5. Please circle the program that keeps track of the largest input number. The program should assume that the input numbers are arbitrary and it should terminate when a negative input value is read.

a.
```
Input A
If( A>0 ){
    B = A
}
Output B
```

b.
```
Input A
B = A
If( A<0 ){
    Output B
}
Else{
    Input A
}
Output B
```

c.
```
Input A
While( A>0 ){
    Input A
}
Output A
```

d.
```
Input A
B = A
While( A>0 ){
    If( A>B ){
        B = A
    }
    Input A
}
Output B
```

Part 2 – Game Experience Questionnaire

For questions 1 through 5 please answer to the best of your ability in a few short sentences.

1. Please describe your level of programming skills prior to playing the game.

2. Please describe your level of programming skills after playing the game.

3. Please describe your overall reaction to the game. (easy, difficult, fun to play, etc…)

4. Please describe any obstacles you had to overcome during the game.

5. How could the game be improved?

Bibliography

[1] Clark C. Abt. *Serious Games*. University Press Of America, March 2002.

[2] Ernest Adams. *Fundamentals of Game Design*. New Riders, 2 edition, 2010.

[3] Knowledge Adventure. Math Blaster Official Website, August 2012. http://www.mathblaster.com/.

[4] Benjamin Samuel Bloom. *Taxonomy of Educational Objectives, Volume 1*. Longmans, Green, 1 edition, 1954.

[5] James Brightman. Games Must Achieve Photorealism in Order to Open Up New Genres Says 2K, August 2012. http://www.gamesindustry.biz/articles/2012-08-01-games-must-achieve-photorealism-in-order-to-open-up-new-genres-says-2k.

[6] Capcom. Megaman Official Website, August 2012. http://megaman.capcom.com/.

[7] Capcom. Phoenix Wright Official Website, August 2012. http://www.capcom.com/phoenixwright/.

[8] Video Game Charts. Video Game Charts, Game Sales, Top Sellers, and Game Data, August 2012. http://www.vgchartz.com/.

[9] The Learning Company. Oregon Trail Official Game Site, August 2012. http://www.oregontrail.com/.

[10] The Pokemon Company. The Official Pokemon Website, August 2012. http://www.pokemon.com/.

[11] Square Enix. Final Fantasy XII Official Website, August 2012. http://www.finalfantasyxii.com/.

[12] Square Enix. Final Fantsy 25th Anniversary, August 2012. http://discography.ff25th-anniversary.com/.

[13] Square Enix. Square Enix Glocal Website, August 2012. http://www.square-enix.com/.

[14] Eric Klopfer et al. Moving Learning Games Forward, 2009. http://education.mit.edu/papers/MovingLearningGamesForward_Ed Arcade.pdf.

[15] Armor Games. Light-bot 2.0 online game, August 2012. http://armorgames.com/play/6061/light-bot-20/.

[16] Armor Games. Light-bot online game, August 2012. http://armorgames.com/play/2205.

[17] James Paul Gee. Good Videa Games and Good Learning, 2005. http://www.jamespaulgee.com/sites/default/files/pub/GoodVideoGames Learning.pdf.

[18] HopeLab. Re-mission Official Website, August 2012. http://www.re-mission.net/.

[19] David Houghton. Why the notion that games need photoreal graphics to tell better stories is backward nonsense, whatever 2k's boss says, August 2012. http://www.gamesradar.com/why-the-notion-that-games-need-photoreal-graphics-to-tell-better-emotional-stories-is-backward-nonsense-whatever-2K-boss-says/.

[20] Jeremy Hsu. For the U.S. Mmilitary, Video Games Get Serious, August 2010. http://www.livescience.com/10022-military-video-games.html.

[21] Christopher Hundhausen, Sarah Douglas, John Stasko, and Andjohn T. Staskoz. A meta-study of algorithm visualization effectiveness, 2002.

[22] IGN. The Other Ma rio Games, Vol. 2: Ma rio is Missing. And many gamers left him that way., August 2008. http://retro.ign.com/articles/897/897225p1.html.

[23] Erkki Kaila, Teemu Rajala, Mikko-Jussi Laakso, and Tapio Salakoski. Effects of course-long use of a program visualization tool. In *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103*, ACE '10, pages 97–106, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[24] Brad King. Educators Turn to Games for Help, August 2003. http://www.wired.com/gaming/gamingreviews/news/2003/08/59855.

[25] Konami. Castlevania Official Website, August 2012. http://www.konami-castlevania.com/.

[26] Microsoft Research FUSE Labs. Kodu Game Lab, August 2012. http://fuse.microsoft.com/page/kodu.

[27] Primer Labs. Code Hero Official Website, August 2012. http://primerlabs.com/code-hero.

[28] Lucasarts. Grim Fandango Official Website, August 2012. http://lucasarts.grim-fandango.com/.

[29] Mojang. Minecraft Official Website, August 2012. http://www.minecraft.net/.

[30] PVC Museum. Video Game Console Specs, August 2012. http://www.pvcmuseum.com/games/console-specs/.

[31] Nintendo. Fitness Game for Nintend Wii, Wii Fit Plus, August 2012. http://www.wiifit.com/.

[32] Nintendo. Nintendo's Official Home for Mario, August 2012. http://mario.nintendo.com/.

[33] Nintendo. Paper Mario Game Info, August 2012. http://www.nintendo.com/games/detail/oWnd6EMCVBbdy9b-5rIkJfC64EEGPQTb.

[34] Nintendo. Zelda Universe, The Official Site of the Legend of Zelda Series, August 2012. http://zelda.nintendo.com/.

[35] Nvidia. Physx Product Overview, August 2012. http://www.geforce.com/hardware/technology/physx.

[36] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. *SIGCSE Bull.*, 39(4):204–223, December 2007.

[37] SEGA. Sonic the Hedgehog, August 2012. http://www.sonicthehedgehog.com/.

[38] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. Analysis of research into the teaching and learning of programming. In *Proceedings of the fifth international workshop on Computing education research workshop*, ICER '09, pages 93–104, New York, NY, USA, 2009. ACM.

[39] Siliconera. Visual Novals: A Cultural Difference Between The East And West, February 2011. http://www.siliconera.com/2011/02/17/visual-novels-a-cultural-difference-between-the-east-and-west/.

[40] Sony. Littlebigplanet Official Website, August 2012. http://www.littlebigplanet.com/.

[41] Unity Technologies. Unity Game Engine Official Website, August 2012. http://www.unity3d.com/.

[42] Alan Thorn. *Game Engine Design and Iimplementation.* Jones & Bartlett Learning, LLC, 1 edition, 2011.

[43] Investment U. The Videa Game Industry: An $18 Billion Entertainment Juggernaut, August 2008. http://seekingalpha.com/article/89124-the-video-game-industry-an-18-billion-entertainment-juggernaut.

[44] Ubisoft. Rayman Origins Official Website, August 2012. http://raymanorigins.us.ubi.com/.

[45] Mark J. P. Wolf. *The Medium of the Video Game.* University of Texas Press, 1 edition, February 2002.

[46] Cyan Worlds. Myst online: Uru live, August 2012. http://mystonline.com/.