Software Requirements Specification (SRS) MathMunition: Equation Siege

Team: Group 2

Authors: Aidan Javidpour, Joel Keaton, Anthony Kitowicz, Erson Ramirez Mendoza, Leon Zeltser

Customer: Middle School Teachers and Students

Instructor: Dr. James Daly

Table of Contents

1 Introduction

The topics that are to be discussed in this document include, the purpose and scope, a description of the product, the software requirements, various UML diagrams, and a sample prototype of the product.

- *Section 1*: This section goes over the purpose, scope of the project, various abbreviations and definitions, and lastly the organization of the rest of the SRS.

- *Section 2*: This section goes over the product's perspective and functions. It also goes over the users characteristics that are necessary to use the game, and then continues on to talk about constraints, assumptions, and apportioning of the various requirements.

- *Section 3*: This section simply goes over the games software requirements.

- *Section 4*: This section goes over the various UML diagrams for the game. That includes the use case diagrams, class diagrams, sequence diagrams, and state diagrams.

- *Section 5*: This final section shows how to run the prototype of the game and samples of the actual game.

1.1 Purpose

The purpose of this SRS document is to give an in-depth detailing of the educational game that our group has created and implemented. It will give a detailed explanation of the software which will be necessary to understand how it works. The intended audience for this is Professor Daly, the developers of this game, and the people testing the game.

1.2 Scope

MathMunition is an educational math based game that is developed in the Unity engine. The goal of this game is to create an interactive math game that will appeal to students of the 7th to 8th grade range that they can learn and have somewhat of a competition to keep them engaged. We plan on having the game downloadable and plan on having it in the browser on our website to make it easier to access if you choose to not download. The game will be programmed in C sharp since it is built in Unity.

The game should be able to teach the student basic algebra, specifically linear equations. With the timer implemented, it should help the student be able to solve these equations more quickly and efficiently. The visuals that are associated with the games functions also make it easy for the students to visualize the equations they are solving. Specifically the type of linear equations it gets the students into is equations in the slope intercept form. Slope intercept form is important when graphing and the cannon shooting is essentially a game-like way to visualize it and ease the students into the topic that could get more complicated as they advance to later grades.

1.3 Definitions, acronyms, and abbreviations

- *SRS*: Software requirements specification

- *MathMunition*: The game’s name

- *UML*: Unified Modeling Language

- *SW*: Software

- *HW*: Hardware

- *Mac*: Apple OS

- *Windows*: Microsoft OS

1.4 Organization

The rest of the SRS will contain an overall description of the software. Within the description will contain a product perspective and functions which will give detailed description of the various interfaces and functionality of the product. In the same overall description category, we will go over the users characteristics, constraints of the product, assumptions, and apportion of requirements. The next category that will be explored is the software specification requirements followed by the modeling requirements. After is simply the prototype with a description of how to run it and various samples of the product. Last in the SRS is our references and point of contact.

2 Overall Description

In this section, something that will be covered is the product perspective. The product perspective will give an oversight on the various interfaces, operations, and other system based details when referring to the software we are creating. Next, the product functions will be discussed, which will explain the various functions that our software will be able to perform along with diagrams to help visualize these functions. Next is the user characteristics, which will explain the needs users are required to have in order to understand and play the game. Then, the constraints section and assumptions and dependencies will follow. The assumptions and dependencies will go over the various assumptions about the hardware, software, environment, and user interactions. Finally, the apportioning of requirements section will go over the requirements that are beyond the scope of the current version of the software being created that could be addressed in future updates.

2.1 Product Perspective

This product is intended to be used both in and outside the classroom to help further understand linear equations in slope intercept form and efficiently solve them. The teacher, parent, and student themselves should be able to access and play the game as they choose to. This is to help encourage the cycle of learning further. The student needs to have or the teacher will need to provide a simple computer setup to actually play the game since this game will require a computer to play.

When playing the game, the user will get to select a difficulty and a level. The difficulty will correspond with the amount of time that the user has to complete the given problem. The highest difficulty will correspond to the least amount of time. The medium difficulty will correspond with slightly more time then the highest difficulty. The easiest difficulty will give you the longest time to solve the given linear equation.

Once you are in the game, you are given a math equation that requires you to enter information in the equation field. The equation field consists of two spots where you will enter in your 2 numbers that correspond to the correct equation coefficients. After pressing the Fire button, the cannon will launch the cannon ball to hit the castle using the specified equation. The goal is to hit the highlighted randomly selected block, which will cause the user to win the game when the block is destroyed. Points will also be awarded and shown on a highscore board.

In order to play the game, it is necessary to have a mouse and keyboard. The mouse is needed to navigate between screens and text fields. The keyboard is needed to enter the specific values to solve the various problems. The game will need Windows 10 or above to run the game. Since the game is written in Unity, it cannot run on a Mac unless you go through the website to play it. The game can be run by downloading it, or it can also be played on our website. The user will need the internet to download the game to play it.

2.2 Product Functions

The main functions of the game will be the solving of the linear equations in slope intercept form. When playing a level in the game, a question will be asked and the expected answer will be put into slope intercept form. After that, the user will enter the coefficients into the text input area. The cannon will fire in the exact line that was created from the equation. The cannon may or may not hit a brick from the structure in front of the cannon. The goal is to hit a specific brick and destroy that brick within the time limit to win. The time limit is determined based on the difficulty selected before you start the level.

The primary goal of this game is to target 7th-8th grade students in the American education system and teach them about simple linear equations and make it fun and easy for them to understand the concepts while also making it simpler. This game is not meant to replace the teaching of linear equations, but it is simply meant to make it easier and make the students more enthusiastic about the topic.

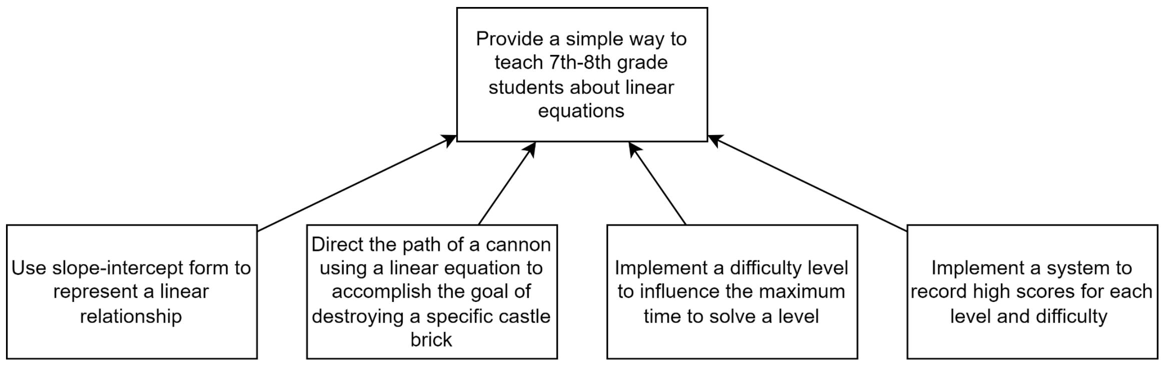

Provide a simple way to teach 7th-8th grade students about linear equations
Use slope-intercept form to represent a linear relationship
Direct the path of a cannon using a linear equation to accomplish the goal of destroying a specific castle brick
Implement a difficulty level to influence the maximum time to solve a level
Implement a system to record high scores for each level and difficulty

Figure 2.2: High-Level Goal Diagram

2.3 User Characteristics

The intended user of the product is around 12-13 years old. This means that the student should be in either 7th or 8th grade. The grade of 7th and 8th grade corresponds with the American education system. The person should be able to solve linear equations and find the equation of a slope of a line. That level of math is considered to be Algebra 1 level math. The user has to also be able to speak english.

2.4 Constraints

Like mentioned previously, this game needs to be played with both a mouse, keyboard, and computer. You need the mouse to navigate the game's UI, and the keyboard to type your answer, meant to represent the equation in slope intercept form.

Software restraints include the need to have a Windows device to download the game. In order to play on Mac OS you should be able to play in the browser on our website. The Windows device needs to be Windows 10 or higher to play the game.

Other constraints that we present are audience constraints due to this product being designed around the American education system, and the product being in English.

2.5 Assumptions and Dependencies

Users must have a functional computer with an internet connection. This computer should be running Windows 10 or higher and must be capable of running Unity. The computer should also have a monitor, mouse, and keyboard connected as peripherals. An internet connection is required to download the game but not play it

The player should have basic skills in English as well as basic problem solving skills. The player is assumed to be 12-13 years old and is in 7th or 8th grade, which should correlate to a basic understanding (or current learning) of algebra and linear equations.

2.6 Apportioning of Requirements

Requirements that we have determined to be beyond the scope of the current project include random level generation. Generating random levels would take too much time based on the time frame we were given for the project and limited knowledge on the use of Unity at the moment. However, randomly generating numbers for the question is within the scope for the current project so random number generation is not out of the project's scope. The other requirement that is beyond the scope of the current project would be the custom coordinate system we originally were planning to implement. Originally this coordinate system would have been the way to determine where the cannon blocks would be hit. However, creating an entire coordinate system for something like this would have stressed our time constraint.

3 Specific Requirements

1. There will be a main menu which will have several options.
    - 1.1. The level selection page which will have all the levels listed.
        - 1.1.1. The level selection screen shows if a level has been played and what the highest score on the level is.
    - 1.2. The help page will have instructions on how to play the game, how each level is won, and how the score is calculated.
    - 1.3. A difficulty selection page where the player is able to choose the difficulty.
        - 1.3.1. At different difficulties, the layout of the levels is unchanged but statistics such as high scores are different.
    - 1.4. An options menu will allow the player to adjust the volume of music and sound effects.
    - 1.5. The about page will list information about the game's developers and resources used.
        - 1.5.1. Includes list of the names of the developers.
        - 1.5.2. Will include information regarding game creation during the fall 2023 semester as part of Software Engineering I taught by Dr. James Daly at UMass Lowell.
        - 1.5.3. Includes list of all external resources used.
    - 1.6. A high scores menu will allow the player to see a list of high scores for all levels.
        - 1.6.1. The scores will be shown for the difficulty that is set.
2. Each level will have a castle on either the left or right side of the screen and a single cannon on the other side.
    - 2.1. The castle will be made up of square blocks.
        - 2.1.1. The shape of the castle will be predetermined.
        - 2.1.2. A random block in the castle in a predetermined column will be chosen as one that needs to be destroyed to win the level, and it is highlighted.
    - 2.2. A single cannon will be on the left or right on the screen.
3. Each level has a timer. If the timer reaches 0 before the level is finished, the game ends and the level is lost.
    - 3.1. The time in the timer is based on the difficulty chosen.
        - 3.1.1. On the easy difficulty there is a limit of 3 minutes.
        - 3.1.2. On the medium difficulty there is a limit of 2 minutes.
        - 3.1.3. On the hard difficulty there is a limit of 1 minute.
4. There will be space to enter a linear equation in slope intercept form ($y = mx + b$). Input spaces will require coefficients to be entered as integers.
    - 4.1. The cannon rotates based on the slope (m) entered.
    - 4.2. The cannon may change position on the y-axis depending on the user's input for (b).

4.2.1. As such, the base sprite the cannon is sitting on will change with the y-position the cannon is at.

4.3. The player will be able to see the path the ball will take depending on the coefficients entered.

5. Once the player is satisfied with the equation there will be a confirmation which will fire the cannon.
   5.1. The cannonball travels until hitting the first castle block along its path and destroying it.
      5.1.1. The cannonball hits as long as it comes into contact with a block's collision box.
      5.1.2. If the cannonball hits the ground or flies off the screen without hitting any castle block, the cannonball will get destroyed by invisible bounds.
      5.1.3. Destroying one block will not affect any other adjacent blocks.
6. The level ends and is considered won when the randomly chosen highlighted block is destroyed before time runs out.
7. After the level is finished, the number of cannonballs fired will be shown along with the time taken to complete it..
8. There will be a way to pause a level, which then shows a menu.
   8.1. The player cannot see the level when paused.
   8.2. The player is able to resume the level after any amount of time.
   8.3. The player is able to save progress and quit the level and continue it at a later time.
   8.4. The player is able to quit without saving.
9. Animations are added for cannonballs.
   9.1. The cannonball is shown flying along the screen.
   9.2. The cannonball explodes after hitting a block, which is then shown getting damaged before disappearing.
10. Music and sound effects are added.
   10.1. A song or several songs are playing in the background.
   10.2. Sound effects are played for cannons firing and cannonballs exploding.
   10.3. Sound effects are played for clicking options.
   10.4. A victory sound is played after the game is won.
   10.5. Music is different for different themes.

4 Modeling Requirements

4.1 Use Case Diagram

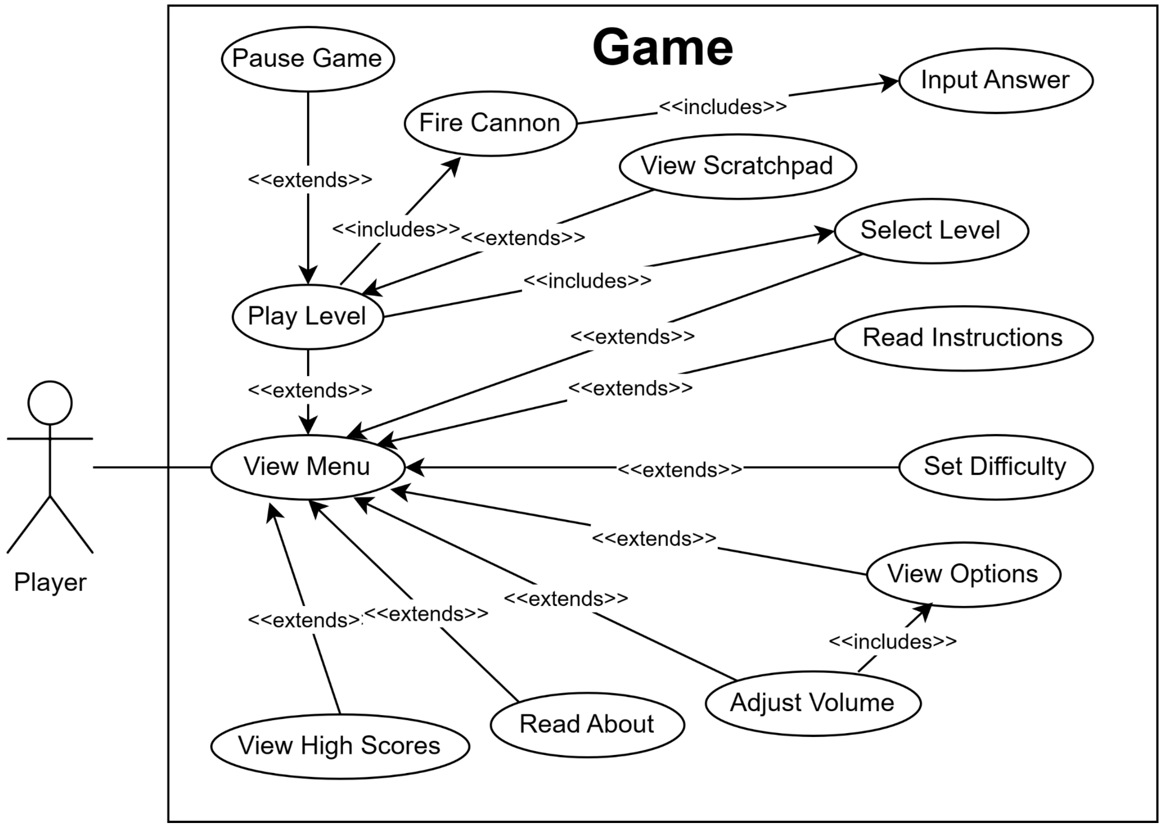


Figure 4.1: Use Case Diagram

The Use Case Diagram shows how the player, shown on the left, interacts with the game. It uses standard UML notation. Each oval represents a use case, or interaction the player has with the game. Use cases which are included in others, which are labeled <<includes>>, are subroutines needed to accomplish the case. Use cases which are extended, labeled by <<extends>>, are an extension of another use case, which may be accomplished as a result of accomplishing the use case it is extended from. Below are descriptions of each of the use cases.

Use Case Name:	Adjust Volume
Actors:	Player
Description:	The player adjusts the volume of the music and/or sound effects.
Type:	Primary
Includes:	View Options

Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	Fire Cannon
Actors:	Player
Description:	The player fires the cannon, causing it to shoot a cannonball and destroy the first block in its path, unless it hits the ground or flies off the screen first.
Type:	Secondary and essential
Includes:	Input Answer
Extends:	
Cross-refs:	
Uses cases:	

Use Case Name:	Input Answer
Actors:	Player
Description:	The player inputs a linear equation in slope intercept form.
Type:	Secondary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	

Use Case Name:	Pause Game
Actors:	Player
Description:	The player pauses the level so the timer no longer ticks down, in case they need to take a short break.
Type:	Secondary
Includes:	
Extends:	Play Level
Cross-refs:	

Uses cases:	

Use Case Name:	Play Level
Actors:	Player
Description:	The player plays the level, inputting an equation and making the cannon fire, which destroys blocks in a castle until they destroy a highlighted target block.
Type:	Primary and essential
Includes:	Fire Cannon, Select Level
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	Read About
Actors:	Player
Description:	The player can see the game’s authors and resources used.
Type:	Primary
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	Read Instructions
Actors:	Player
Description:	The player can read the instructions on how to play the game.
Type:	Primary
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	Select Level

Actors:	Player
Description:	The player sees the list of levels and chooses one of them to play.
Type:	Primary and essential
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	Set Difficulty
Actors:	Player
Description:	The player sets the difficulty of the levels, either easy, medium, or hard, which affect certain aspects of the gameplay such as the time limit the player has to finish a level.
Type:	Primary
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	View High Scores
Actors:	Player
Description:	The player is able to see the best scores for each level, which are different for every difficulty.
Type:	Primary
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

Use Case Name:	View Menu
Actors:	Player

Description:	The player is able to see the main menu, and from there is able to go to one of several screens, including the difficulty screen, the high scores, and choose a level to play.
Type:	Primary and essential
Includes:	
Extends:	
Cross-refs:	
Uses cases:	

Use Case Name:	View Scratchpad
Actors:	Player
Description:	At any time when playing the level, the player is able to open a scratchpad, which is a blank screen where they are able to draw freely, in order to do some quick calculations.
Type:	Secondary
Includes:	
Extends:	Play Level
Cross-refs:	
Uses cases:	

Use Case Name:	View Options
Actors:	Player
Description:	The player is able to see the options such as volume and what they are set to.
Type:	Primary
Includes:	
Extends:	View Menu
Cross-refs:	
Uses cases:	

4.2 Class Diagram

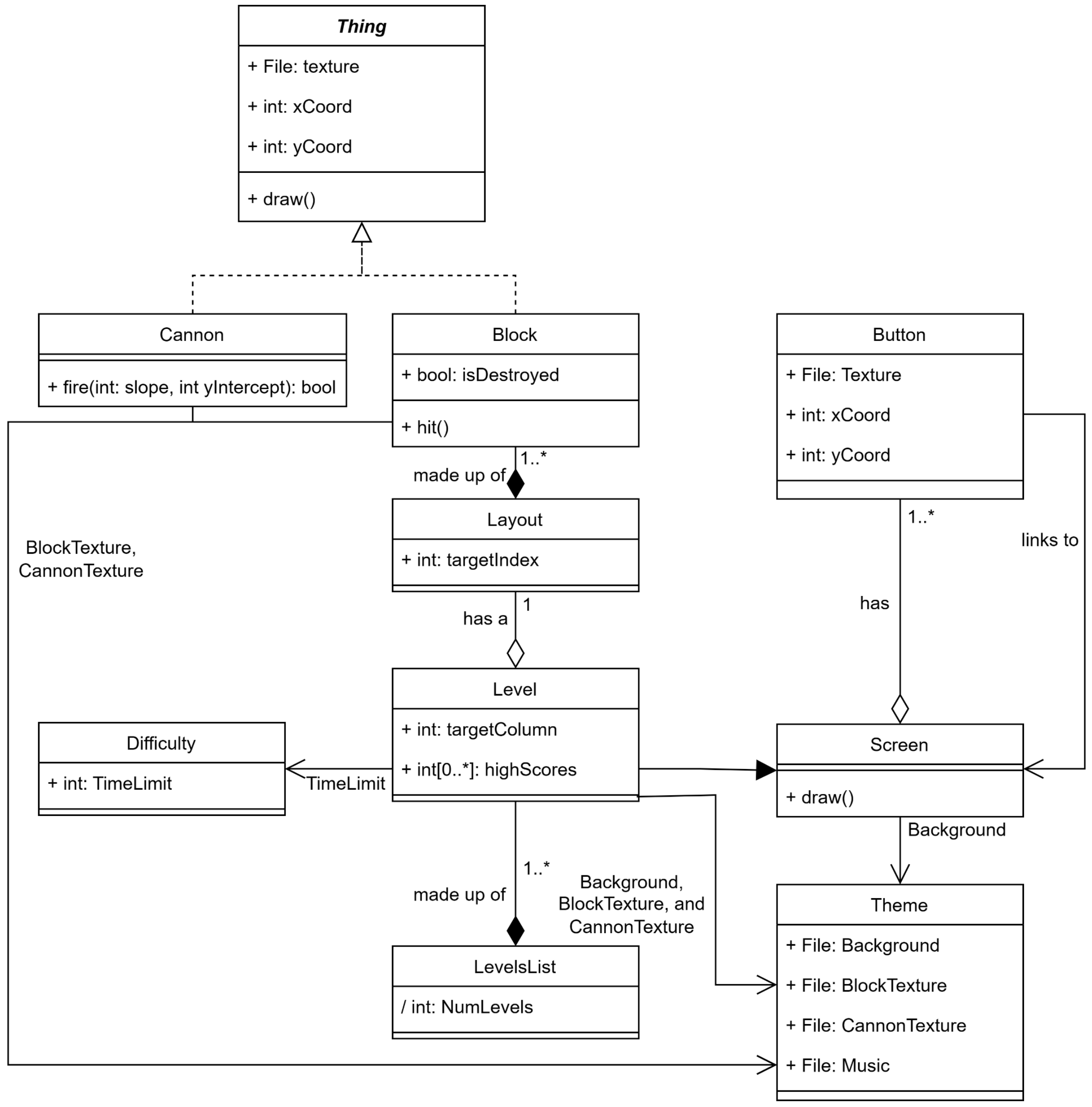


Figure 4.2: Class Diagram

The class diagram shows the classes that make up the program, and uses standard UML notation. Each box is a class, with its name being on the top section, the middle section having attributes, and the lower section having its methods. Types of all attributes, method parameters, and return types are listed along with their names. A + before a method or attribute name means it is public, while a / means it is calculated. Arrows show relationships between classes. The arrowheads display inheritance, with white meaning inheriting from a base class while black meaning inheriting from an already defined class. The empty arrows means a class “knows about” another class, and uses some of its values. The diamonds mean a class is part of another class. A black diamond means aggregation, or a class is entirely made up of another class, while a white diamond means composition, that a class is part of another class although it has other fields.

Element Name		**Description**
Block		The Block class represents the blocks the castle is made out of. It is instantiated on the game's start, and drawn when the player selects a level.
Attributes		
	isDestroyed:boolean	A boolean saying if a block has been destroyed. If true, it will not be displayed on the screen, else it is displayed.
Operations		
	hit():	A method that is called if a cannonball hits the block. The boolean isDestroyed is set to true and the block disappears from the screen.
Relationships	Block inherits from the Thing abstract base class, and the Layout class is composed of instances of the Block class. The Block class also takes its texture from an instance of the Theme class.	
UML Extensions		

Element Name		**Description**
Button		The Button class represents buttons on the menu screens, which the user presses to get to another screen.
Attributes		
	LinksTo:Screen	A pointer to the instance of a Screen class where the button links to.
	Texture:File	An image file with the button's texture.
	xCoord:int	The x coordinate of where the button sits on the screen.
	yCoord:int	The y coordinate of where the button sits on the screen.
Operations		
Relationships	The Button class links to another Screen class, which is another attribute of it. The Screen class also has a container with instances of the Button class, which is all the buttons on that screen.	
UML Extensions		

Element Name	**Description**
Cannon	The Cannon class represents the cannon which the player fires.

Attributes		
Operations		
	fire(slope:int, yIntercept: int):bool	Causes the cannon to fire a cannonball after the player inputs values for the slope and y intercept and returns true, unless the equation is not valid in which case it returns false and does nothing.
Relationships	The Cannon class inherits from the Thing abstract base class. It also takes its texture from an instance of the Theme class.	
UML Extensions		

Element Name		**Description**
Difficulty		The Difficulty class represents the difficulty of the game, which affects certain gameplay elements such as the time limit.
Attributes		
	TimeLimit:int	The time limit in minutes the player has to complete a level when this difficulty is chosen.
Operations		
Relationships	The Level class knows what difficulty the game is currently set to, and it takes the time limit from the corresponding Difficulty class.	
UML Extensions		

Element Name		**Description**
Layout		The Layout class has a container with several instances of the Block class, which represents the layout of the level.
Attributes		
	BlocksList:Block[1..*]	A container holding instances of the Block class.
	targetIndex:int	The index of a randomly chosen block to be the target.
Operations		
Relationships	The Layout class has instances of the Block class, which make up the castle in a level. Each Level class has a Layout class. When a level is chosen, the index of a randomly chosen block in the level's targetColumn is chosen as a target.	
UML Extensions		

Element Name	**Description**

Level		The Level class represents every level in the game.
Attributes		
	layout:Layout	An instance of the Layout class, which is the level's layout.
	targetColumn:int	The column in the castle from which a block is randomly chosen to be the target block.
	highScores:int[0..*]	The high scores achieved on the level.
Operations		
Relationships	The Level class inherits from the Screen class. It has an instance of the Layout class, as well as taking information about the level from the Difficulty and Theme class. The LevelsList class has a number of instances of the Level class.	
UML Extensions		

Element Name		**Description**
Levelslist		The LevelsList class has a list of all the levels in the game.
Attributes		
	AllLevels:Level[1..*]	A container with instances of the Level class.
	NumLevels:int	The number of levels.
Operations		
Relationships	The LevelsList class has a container with instances of the Level class.	
UML Extensions		

Element Name		**Description**
Screen		The Screen class represents all the screens in the program.
Attributes		
	ButtonList:Button[1..*]	A container with instances of the Button class, which is all the buttons shown on the screen.
Operations		
	draw():	Displays the screen with all buttons or other information.
Relationships	The Screen class has a container with instances of the Button class. It also takes the background from a Theme class. The Level class inherits from the Screen class.	
UML Extensions		

Element Name		Description
Theme		The Theme class represents the theme of the level and the game, which affects background music, sound effects, and the look of the blocks.
Attributes		
	Background:File	An image file to be used for the background.
	BlockTexture:File	An image file to be used as a texture for blocks.
	CannonTexture:File	An image file to be used as a texture for the cannons.
	Music:File	A music file which will play in the background.
Operations		
Relationships	Several classes take information from an instance of the Theme class for the look of the blocks and music.	
UML Extensions		

Element Name		Description
Thing		The Thing class is an abstract base class that represents something that is drawn in every level.
Attributes		
	texture:File	An image file of how the object will look like.
	xCoord:int	The x coordinate of where in the level the object will be located.
	yCoord:int	The y coordinate of where in the level the object will be located.
Operations		
	draw():	Draws the object on the screen.
Relationships	The Block and Cannon classes inherit from the Thing class.	
UML Extensions		

4.3 Sequence Diagram 1: Menus

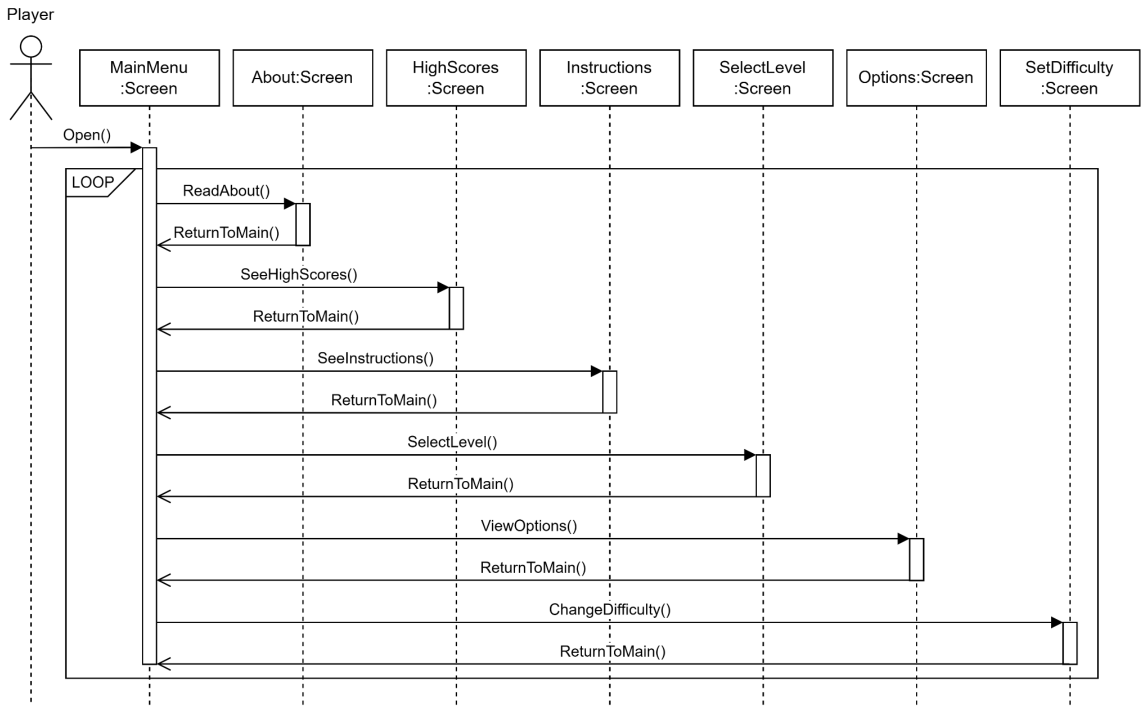


Figure 4.3: Menu Sequence Diagram

The Menu Sequence Diagram shows how the user goes through the various screens and menus. It uses standard UML notation. The stick figure on the left represents the player. Each box represents a certain instance of a class, and the arrows represent functions that cause a certain class to be active. Dotted lines under the names of classes show that they are running, and when the line turns into a box it means it is the main active class. The loop shows that the menus can be looped through indefinitely.

4.4 Sequence Diagram 2: Gameplay

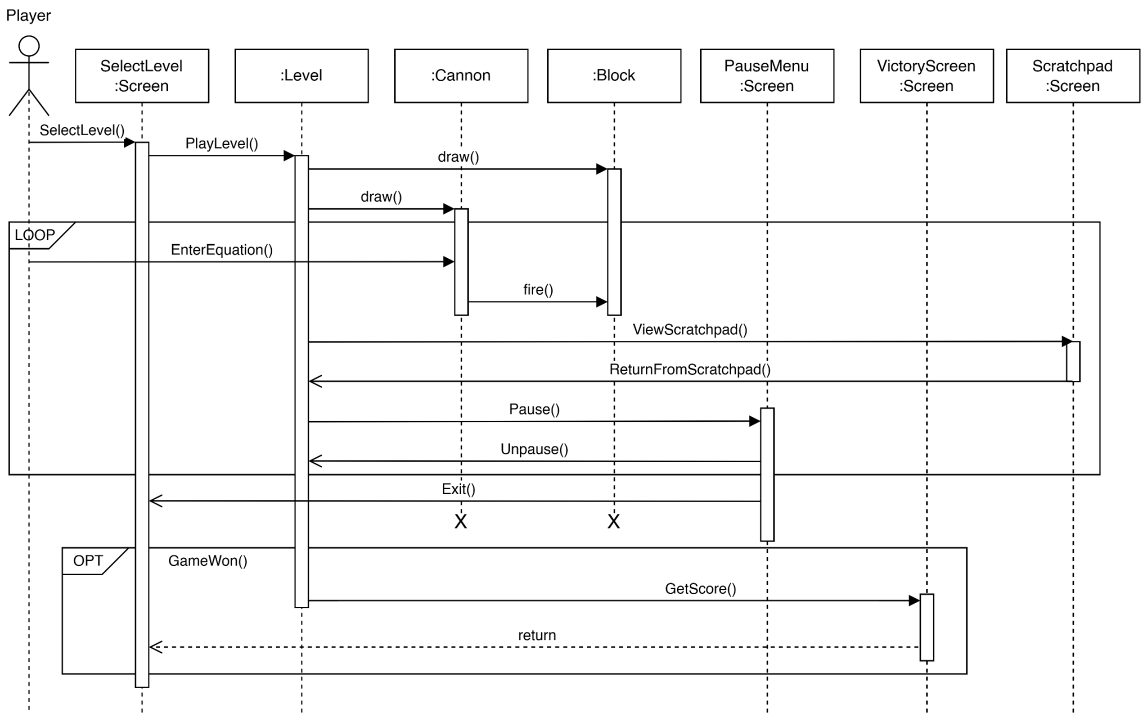


Figure 4.4: Gameplay Sequence Diagram

The Gameplay Sequence Diagram shows how the user can select a level then play it. It uses standard UML notation, the same as described for the Menu Sequence Diagram above. This diagram also has Xs at the end of the dotted lines of several classes, meaning that they are no longer active, and a box labeled OPT that only executes when the condition displayed on top of it is met.

4.5 State Diagram

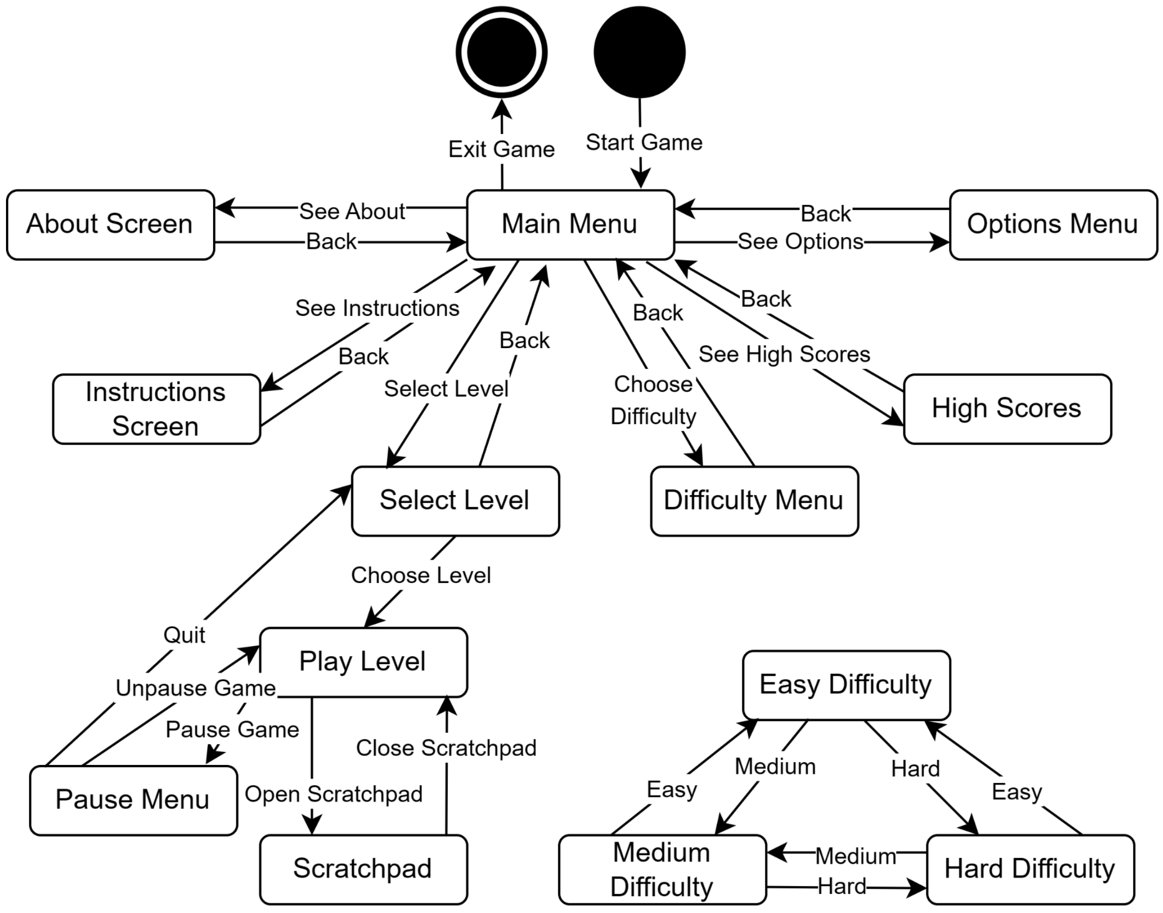


Figure 4.5: State Diagram

The state diagram shows how the program changes from the input of the user. In the top, the black circle to the right represents the program starting, while the circle to the left represents the program ending. Each box represents a state the program can be in, which is also equivalent to the screen it is on. Arrows represent how states can change, which are all buttons pressed by the player. The bottom right shows the difficulties, which can be changed by the player through the Difficulty Menu. They affect some gameplay elements in the level as well as the scores shown in the High Scores page.

5 Prototype

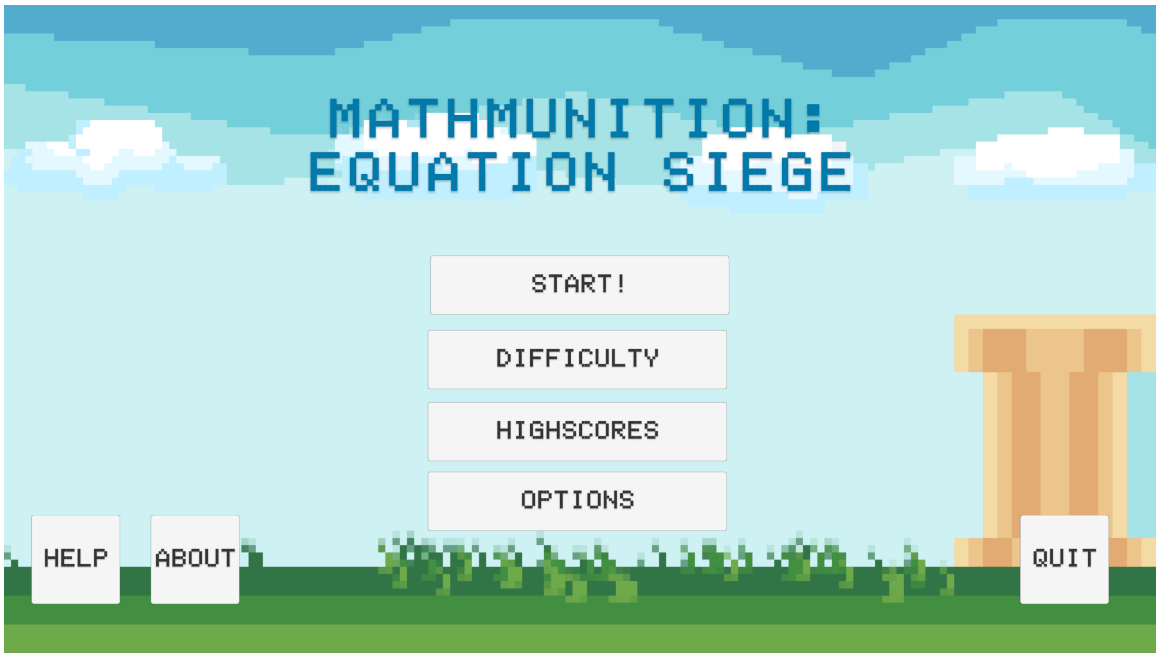
The existing iteration of the prototype showcases fundamental UI capabilities, allowing users to navigate between screens and transition into various levels. Notably, there is a lack of actual game logic in this current version of the game, which will be added later on.

5.1 How to Run Prototype

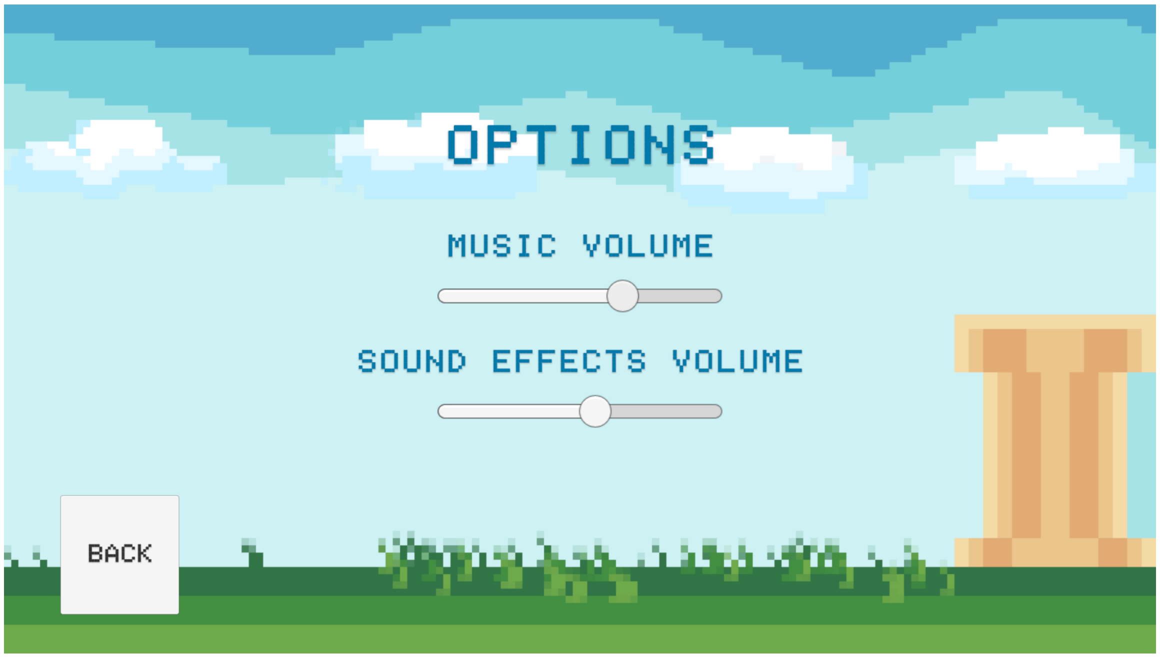
This prototype is designed to operate on Windows computers using Unity, and the project itself can be found on github. The project may be downloaded as a zip and loaded up as a Unity project file for further exploration. Additionally, a prototype executable will be included for users' convenience. Users will navigate the game using a mouse and keyboard.

5.2 Sample Scenarios

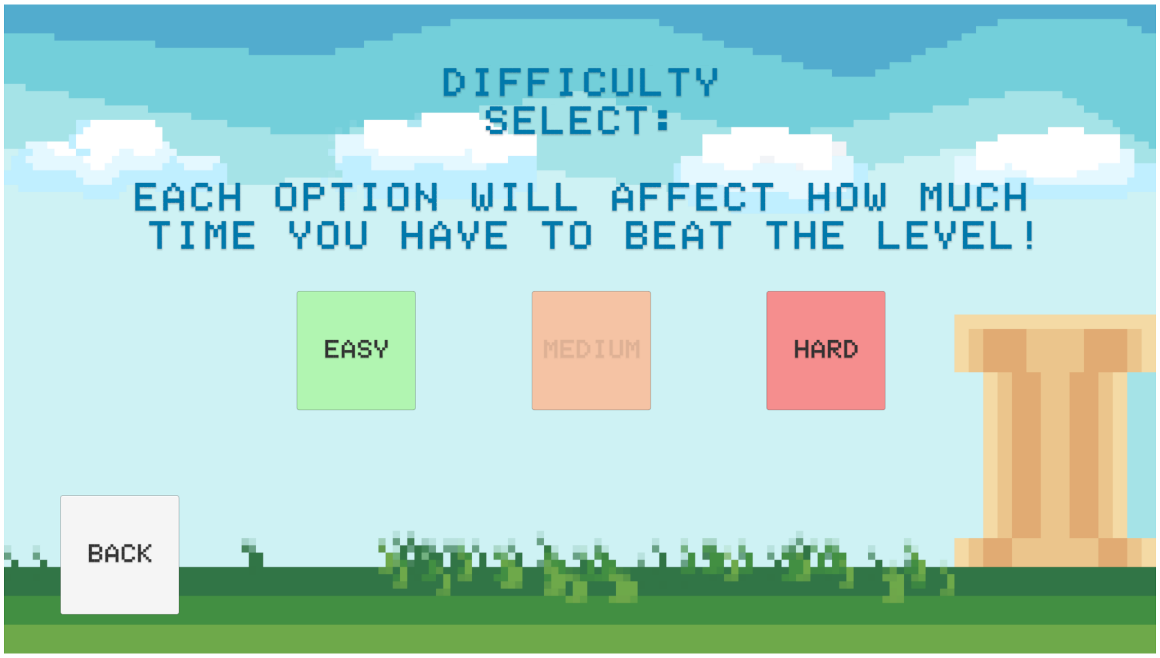
From the main menu, players will be able to navigate different menus using the buttons on screen.



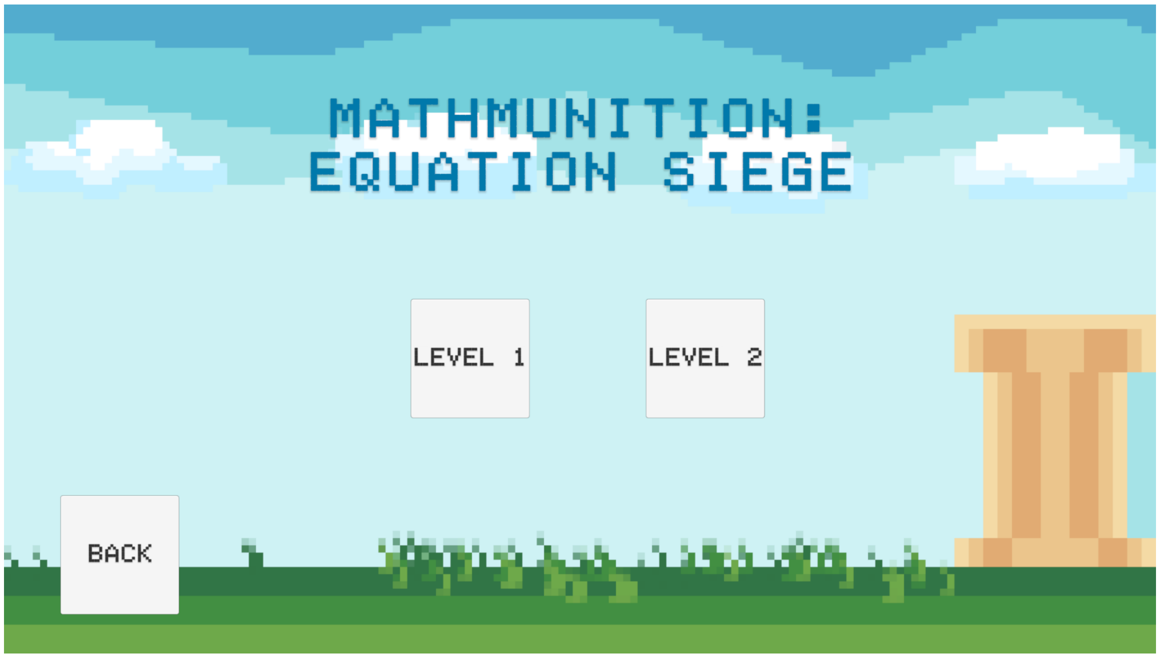
Players will be able to adjust settings related to the volume of sound effects and music in the options menu.



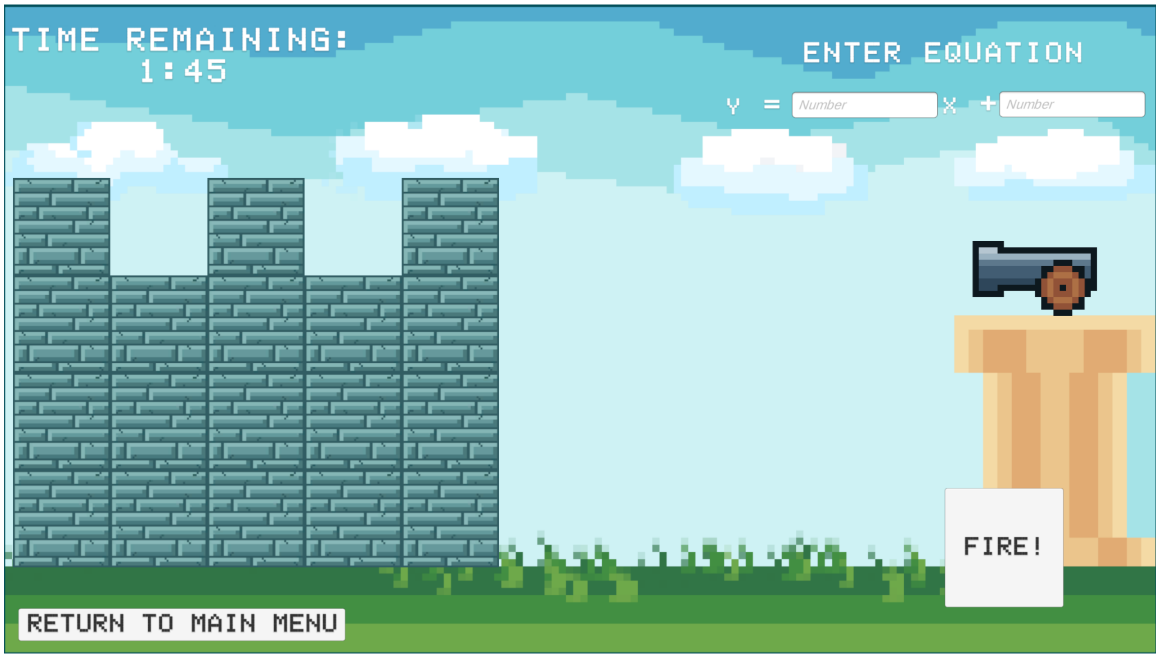
Players will be able to select overall difficulty for the game here in the difficulty menu. Defaults to medium difficulty.


DIFFICULTY SELECT:
EACH OPTION WILL AFFECT HOW MUCH TIME YOU HAVE TO BEAT THE LEVEL!
EASY
MEDIUM
HARD
BACK

Players will be able to select which level they would like to play here. Only 2 example levels are provided in this prototype.



In this example level, players will be able to interact with the equation coefficients. Only single integers are able to be input. Players can also interact with the fire button, but there will be no functionality. Only the return to menu button will function at this time.



6 References

[1] D. Thakore and S. Biswas, “Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks.” Proceedings of IEEE Military Communication, Atlantic City, October 2005.

[2] Massachusetts Department of Elementary and Secondary Education, “Massachusetts Mathematics Curriculum Framework — 2017.” Massachusetts Department of Elementary and Secondary Education, 2017, https://www.doe.mass.edu/frameworks/math/2017-06.pdf.

[3] Project Website: https://jojojo8359.github.io/SWE-Project/

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.