Software Requirements Specification (SRS) MathMunition: Equation Siege

Team:	Group 2
Authors:	Aidan Javidpour, Joel Keaton, Anthony Kitowicz, Erson Ramirez Mendoza, Leon Zeltser
Customers:	Middle School Teachers and Students
Instructor:	Dr. James Daly

Table of Contents

1 Introduction

This SRS (Software Requirements Specification) document will give an overview about MathMunition: Equation Siege, a math-based educational game meant to teach students about linear equations. It will include explanations of its purpose and software requirements from the user's and developer's side, and information about a prototype of the game.

Section 1, which is this section, goes over the purpose and scope of the project, provides definitions for terms used in the SRS, and presents the document's overall organization. The next section, Section 2, includes an overall description of the product including the perspective, constraints, some assumptions about the user, and requirements for future development. Section 3 has a list of requirements for this prototype from the user's perspective, then Section 4 has diagrams displaying the requirements from the developers' side. Section 5 discusses the prototype and how to run it. Section 6 lists all sources this document cites, and finally Section 7 has a point of contact for the project.

1.1 Purpose

The purpose of this SRS document is to give an in-depth detailing of MathMunition for the stakeholders of this project and others who it may be of interest to, including but not limited to Dr. Daly, game testers, middle school math teachers, and those involved in planning middle school mathematics curriculum. This document describes the developers' intention for the game, listing its requirements from the user's and developer's side, and gives an explanation of a prototype made by the developers, as well as listing possible requirements for future versions of the game.

1.2 Scope

MathMunition is an educational video game which has the look and feel of a 2D platform style game, written in C# and developed using Unity. The goal of the game is to teach linear equations to 7th and 8th grade students, and be engaging as well as educational. It includes competitive elements such as a score and time limit for completing levels to help increase engagement.

In the game, players will enter linear equations into a field, then will fire a cannon. The cannonball will travel in the trajectory of the equation. The visuals in the game, including flying cannonballs, will make it easy for the player to visualize these equations. Linear equations in the game are specifically in slope-intercept form, since it is generally the most commonly used format to write them. Levels come with a time limit for completion and a score derived from the number of cannonballs fired, with a bonus added for completing the level, encouraging students to solve equations quicker and more efficiently.

Linear equations in slope-intercept form are the only topic that this game teaches. MathMunition will not be a replacement for learning linear equations, but rather a companion to help students learn about and be more excited about the subject.

The game will be available to download from the project website [1] so an internet connection is not necessarily required to play, although it can also be played in the browser from the website if the player chooses to not download it.

1.3 Definitions, Acronyms, and Abbreviations

- *SRS*: Software Requirements Specification

- *MathMunition*: Or MathMunition: Equation Siege, the name of the game that this document describes.

- *Unity*: A game engine developed by Unity Technologies which MathMunition is developed using.

- *UML*: Unified Modeling Language

- *OS*: Operating System

- *WebGL*: JavaScript API used for rendering games on a browser.

- *Block*: A square that makes up the castle in each level, which the player is able to destroy by firing the cannon at.

- *Target:* A block that if the player destroys, they win the level.

- *Cartesian Coordinate System:* A coordinate system bound by the X-axis and Y-axis, which are two lines that are perpendicular to each other, and each point is specified by its distance from both axes.

- *Coordinate:* A point on the Cartesian coordinate system represented in the form (a,b), where a is the x coordinate, or the distance from the point to the y-axis, and b is the y coordinate, or its distance from the x-axis.

- *X-Axis:* A horizontal line on the Cartesian coordinate system. All points along it are at y=0.

- *Y-Axis:* A vertical line on the Cartesian coordinate system. All points along it are at x=0.

- *Linear Equation*: An equation which creates a line on the Cartesian coordinate system.

- *Slope-intercept form*: Linear equations in the form y = mx+b, where m is the slope and b is the y-intercept, or y coordinate at which the line intersects the y-axis.

- *Y-intercept*: Coordinate at which a line intersects the y-axis.

1.4 Organization

The rest of the SRS will contain an overall description of MathMunition and a prototype developed by the team. It is organized as follows:

- *Section 2*: This section goes over the product's perspective and functions.
    - *Subsection 2.1*: Product Perspective, which describes the context of the game and its purpose.
    - *Subsection 2.2*: Product Functions, which gives a description of the game and the goals the team intend to meet by creating it, and includes a diagram with the project goals.
    - *Subsection 2.3*: User Characteristics, which describes expectations about players of the game.
    - *Subsection 2.4*: Constraints, which describes constraints that limited the development of the prototype.
    - *Subsection 2.5*: Assumptions and Dependencies, which describes assumptions made about the game and dependencies the game requires to function.
    - *Subsection 2.6*: Apportioning of Requirements, which describes requirements that are outside the current scope of the project and may be addressed in future versions of the game.

- *Section 3*: This section has a numbered list of software requirements for the project from the user's perspective. It lists several top-level or base requirements and sub-requirements related to them.

- *Section 4*: This section describes requirements from the software developer's perspective, and has five UML diagrams along with descriptions of them and other information relating to them.
    - *Subsection 4.1*: Use case diagram, which describes all ways the player can interact with the software and how these cases are related to one another. Following it are descriptions of all use cases.

    - *Subsection 4.2*: Class diagram, which shows all classes in the program, their attributes, methods, and how they interact with each other. Following it are descriptions of all classes, their attributes, and methods.
    - *Subsection 4.3*: Menu sequence diagram, which shows how the player can interact with the menus in the game.
    - *Subsection 4.4*: Level sequence diagram, which shows how the player interacts with elements of the game when playing a level.
    - *Subsection 4.5*: State diagram, which shows states the program can be in and what actions can cause it to be in a different state.

- *Section 5*: This section explains details about the prototype, how to download and run it, and describes several possible scenarios that may occur when running the game.

- *Section 6*: This section lists external sources which this document has cited.

- *Section 7*: This section has a point of contact for this project.

2 Overall Description

This section will provide a description about MathMunition, first discussing the product perspective and basic functions, then information about the user, constraints, assumptions, and finally possible future requirements which were not addressed when developing the current prototype.

2.1 Product Perspective

This product is intended to be used both in and outside the classroom to help students further understand linear equations, specifically writing an equation of a line that goes through two specific points in slope-intercept form. It is intended for 7th and 8th grade students, which according to the Massachusetts Department of Education are the grades where linear equations are taught at [2]. The game itself is independent from external software systems, so teachers, parents, and students can all access and play the game with no additional system integrations.

The game will have the following operational constraints:

- System Interfaces
    - Since the game is not connected to an external software system, it has no system interface constraints.
- User Interfaces
    - The game requires a screen to display its graphics. This would logically be a screen connected to the operating system (OS) the game will be running on.
    - The game requires the use of numeric keys found on a standard keyboard.
    - The game also requires the use of inputs from a mouse or other pointing device (including positional data and use of navigational buttons, primarily the “left-click” or “primary click” button).
- Hardware Interfaces
    - The hardware the game will be run on must have the proper support for both a keyboard and mouse. This will vary by system, so specific setup configurations will be omitted here.
- Software Interfaces
    - The game requires the use of the Unity Player package, since the game is being developed in Unity.
        - However, since the game should run for Windows and web platforms, the requirements for Unity Player for these platforms should be met.
- Communications Interfaces
    - The game does not require any communications interfaces.
        - However, communication interfaces outside the game’s scope may be needed to access the game itself.
- Memory Constraints
    - The game will require a small amount of disk space in order to store the high scores file.
- Site Adaptation Requirements

- The game does not require, nor has the capacity for, changes in order to accommodate specific sites or use cases.

Figure 1 shows how the game will fit into the Massachusetts education system.

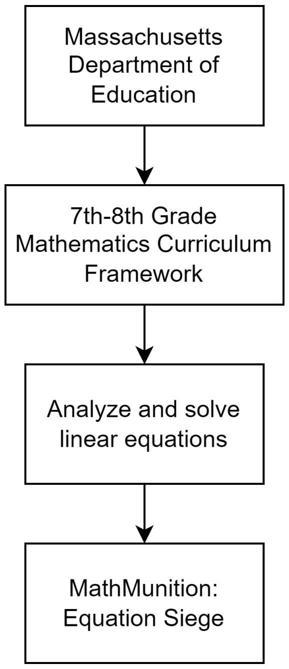

Massachusetts Department of Education
7th-8th Grade Mathematics Curriculum Framework
Analyze and solve linear equations
MathMunition: Equation Siege

Figure 1: Massachusetts Education System Diagram

2.2 Product Functions

Gameplay in MathMunition revolves around solving linear equations. On one side of the screen is a cannon, and on the other side is a castle made up of several blocks. One of these blocks is designated as the target, which if destroyed will cause the player to win the level. The player fires the cannon by entering an equation in slope-intercept form then pressing the fire button, and this equation is the trajectory of the cannonball. The player will just enter the slope as the cannon's location is set to a random value on the y-axis, and it changes after each shot. The cannonball travels along the screen until hitting a block, the ground, or flying off the screen. If it hits a block, the block is first damaged, and after three hits is destroyed. There is a limited amount of time to complete the level based on the difficulty, and the player loses if they do not destroy the target block before the time runs out.

As shown in Figure 2, the small goals that are present for the game are not meant to replace teaching linear equations to 7th-8th grade students, but to make it much simpler and more fun, to make the students or users want to learn the subject.

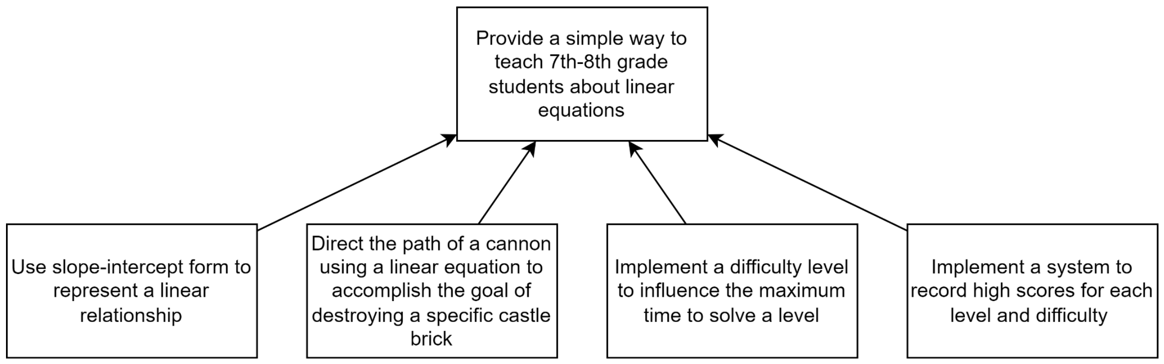

Provide a simple way to teach 7th-8th grade students about linear equations
Use slope-intercept form to represent a linear relationship
Direct the path of a cannon using a linear equation to accomplish the goal of destroying a specific castle brick
Implement a difficulty level to influence the maximum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

Figure 2: High-Level Goal Diagram

2.3 User Characteristics

Players should have at least a basic grasp on linear equations, which as outlined by the Massachusetts Department of Education standards is supposed to be covered in 7th or 8th grade [2] (or 12-14 years old in the American education system). That level of math is considered to be Algebra 1 level math. The user has to also be able to read English. Other intended users of the product are teachers of 7th and 8th grade math classes and parents of students in these classes, and anyone else involved in planning curriculum for those grades.

2.4 Constraints

One major constraint during the development of the game is the amount of time given to complete it. This leads to other constraints, including not offering a mobile platform option and not including accessibility features in the game. Many features originally envisioned by the team had to be scrapped as only one member was familiar with the tools the team used. The game should be reliable enough to run without any game-breaking issues, however. The game will have text content only written in English. Because the game will not have communication capabilities, there are no security constraints needed.

2.5 Assumptions and Dependencies

Players must have a functional computer with an internet connection, needed to download or otherwise access the game. If running the Windows release of the game, this computer should be running Windows 10 or higher and must be capable of running Unity. Otherwise, the computer should be able to view the game in a web browser and should meet the requirements for the WebGL Unity Player platform. The computer should also have a monitor, mouse, and keyboard connected as peripherals.

2.6 Apportioning of Requirements

There are a number of requirements which the team has determined as beyond the scope of this project due to time constraints but could be added to future versions.

One such requirement is adding a random level generator, to randomly decide the shape of the castle, position of the target block, and position of the cannons, with cannons positioned on points besides just the y-axis. Making it would require making a custom Cartesian coordinate system as Unity does not have one built in, and because of that the team decided against implementing it in the prototype. A related requirement is adding more levels. Three exist in the prototype which the team determined was sufficient for demonstration purposes.

Referenced in the last requirement was having cannons on positions besides just the y-axis. Implementing this would also require a custom coordinate system, and a check to make sure the equation entered goes through the point the cannon is at.

Two more requirements which were dropped due to time constraints are a scratchpad, which the player can open at any time during gameplay to do some calculations, and a way to save and quit the game when paused.

There are also two ways the game could be expanded. One is by adding additional themes, such as an industrial theme or a maritime one with a ship instead of a castle, either to individual levels or have it apply to all levels through the settings. Another way is expanding it beyond slope-intercept form, adding either other forms for linear equations, or other equations entirely such as parabolas.

3 Specific Requirements

1. There will be several screens the player is able to go through.
    1.1. There will be a main menu that the player sees after starting the game.
        1.1.1. The player will be able to get to all other screens from the main menu.
        1.1.2. The player is able to quit the game from the main menu.
    1.2. Each other screen will have a way of returning to the main menu.
    1.3. There will be a level selection screen which lists all levels.
        1.3.1. Each level will say if it has been completed or not.
        1.3.2. If a level has been completed, the best score for that level for the currently selected difficulty is shown.
    1.4. There will be a help page with instructions on how to play the game.
        1.4.1. The page describes how levels are won and lost.
        1.4.2. The page describes how the score is calculated.
    1.5. There will be a difficulty selection screen where the player is able to choose between difficulties.
    1.6. There will be an options menu which will allow the player to adjust the volume of music and sound effects.
    1.7. There will be a high scores screen that shows a list of the best scores for every level for the currently set difficulty.
    1.8. There will be an about page that will list information about the game's developers and resources used.
        1.8.1. The page includes a list of the names of the developers.
        1.8.2. The page includes information regarding game creation during the fall 2023 semester as part of Software Engineering I taught by Dr. James Daly at UMass Lowell.
        1.8.3. The page includes a list of all external resources used.

2. There will be an easy, medium, and a hard difficulty.
    2.1. The default difficulty is medium.
    2.2. At different difficulties, the layout of the levels is unchanged.
    2.3. The difficulty will determine the time the player has to complete a level.
        2.3.1. On the easy difficulty the player has 3 minutes.
        2.3.2. On the medium difficulty the player has 2 minutes.
        2.3.3. On the hard difficulty the player has 1 minute.
    2.4. Statistics such as high scores are different for different difficulties.

3. Each level will have a castle on either the left or right side of the screen and a single cannon on the other side.
    3.1. The castle will be made up of blocks.

3.1.1. The shape of the castle will be predetermined.
3.1.2. A random block in the castle in a predetermined column will be chosen as the target.
3.1.2.1. The target block will be highlighted.
3.2. A single cannon will be on the left or right on the screen along the y-axis.
3.3. There will be a way of returning to the main menu from the level.

4. During gameplay the player will be able to fire the cannon as much as they want until the level is either won or lost.
4.1. The cannon will be set to a random y value.
4.1.1. The cannon will move to a different randomly chosen height after each shot is fired.
4.1.2. The cannon sprite will move up and down to the decided height.
4.2. There will be space to enter a linear equation in slope-intercept form.
4.2.1. The y-intercept will not be able to be changed.
4.2.2. The player can enter any numerical value as the slope.
4.3. Once the player is satisfied with the equation they will be able to confirm it which fires the cannon.
4.3.1. The cannonball travels until it either hits a castle block, the ground, or flies off the screen.
4.3.1.1. If the cannonball hits a block, the block will be destroyed.
4.3.1.1.1. Destroying one block will not affect any other adjacent blocks.
4.3.1.1.2. An explosion animation will play then the block which was hit will disappear.
4.3.1.2. If the cannonball hits the ground or flies off the screen without hitting any castle block, the cannonball will get destroyed by invisible bounds.
4.4. Each level has a timer. If the timer reaches 0 before the level is finished, the game ends and the level is lost.
4.4.1. The player is taken to the loss screen.
4.5. If the target block is destroyed before the timer runs out, the game ends and the level is won.
4.5.1. The player is taken to the victory screen.
4.6. After the level ends the player will be shown a score in the victory or loss screen.
4.6.1. The score is based on how many blocks the player destroyed, with a large bonus added if they destroyed the target block and won the level.
4.6.2. The player is able to return the main menu from the screens.
4.6.3. The player is able to restart the level from the screens.

5. There will be music and sound effects.
    5.1. The player is able to adjust the volume of the music and sound effects.
    5.2. A song or several songs are playing in the background.
    5.3. Sound effects are played for cannons firing and cannonballs exploding.
    5.4. Sound effects are played for clicking options.
    5.5. A victory sound is played after the game is won.
    5.6. Another sound is played if the game is lost.

4 Modeling Requirements

4.1 Use Case Diagram

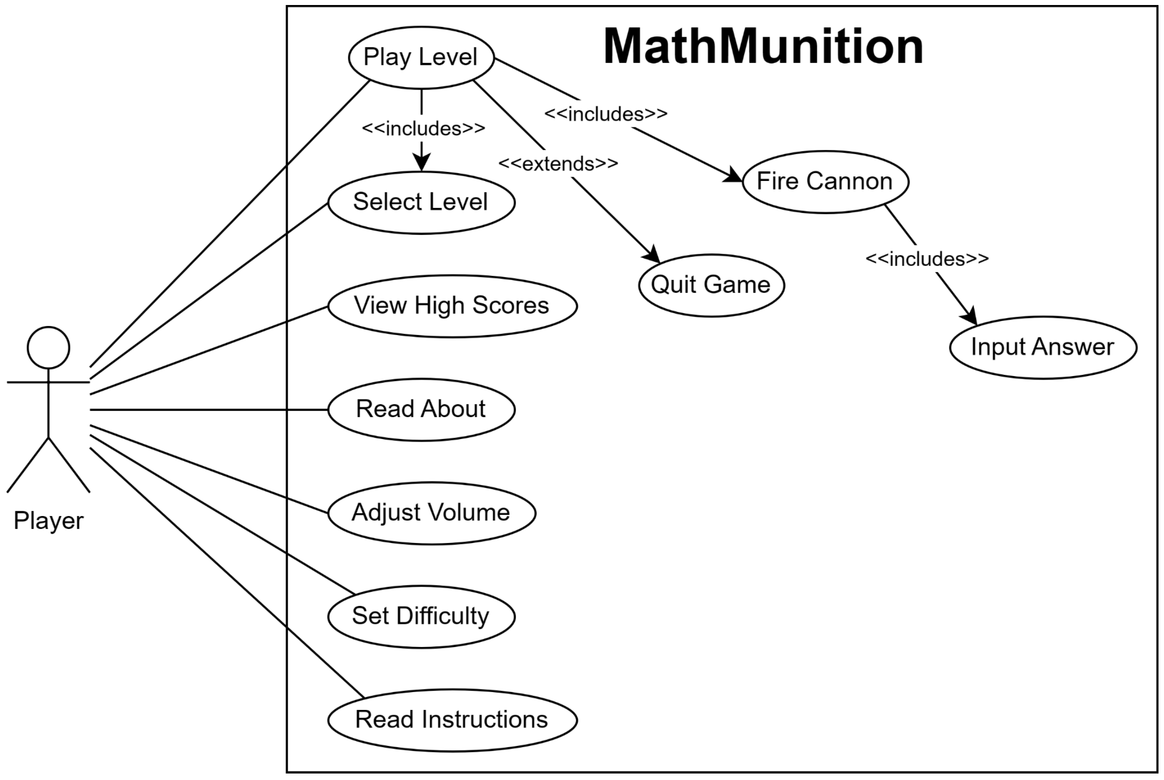


Figure 3: Use Case Diagram

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, labeled with <<includes>>, are subroutines needed to accomplish the case. Use cases which are extended, labeled with <<extends>>, are an extension of another use case, which may be accomplished as a result of accomplishing the use case it is extended from.

There are seven primary use cases. Six of them, Read Instructions, Set Difficulty, Adjust Volume, Read About, View High Scores, and Select Level, are all accessible from the main menu. The last one, Play Level, first requires the player to choose a level to play. During gameplay the player will repeatedly fire the cannon, which requires them to first input an equation. They are also able to pause the game at any time, and from there they can quit the game.

Below are descriptions of each of the use cases along with requirements from section 3 which they reference.

Use Case Name:	Adjust Volume
Actors:	Player
Description:	The player adjusts the volume of the music and/or sound effects.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 6.1
Uses cases:	N/A

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:	N/A
Cross-refs:	Requirement 4.3 and its subrequirements
Uses cases:	Performed as a subroutine of Play Level use case, Input Answer use case must be performed every time before the cannon can be fired.

Use Case Name:	Input Answer
Actors:	Player
Description:	The player inputs a linear equation in slope-intercept form.
Type:	Secondary and Essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 4.2 and its subrequirements
Uses cases:	Performed as a subroutine of Play Level use case, Fire Cannon use case will always be performed after the answer is inputted.

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:	Pause Game, View Scratchpad
Cross-refs:	Requirement 4 and its subrequirements
Uses cases:	The player first needs to choose a level, or do the Select Level use case before they can play the level. While playing they will repeatedly perform the Fire Cannon use case. Quit Game is a use case they can also perform while playing the level.

Use Case Name:	Quit Game
Actors:	Player
Description:	The player can quit the level without saving progress.
Type:	Secondary
Includes:	N/A
Extends:	Play Level
Cross-refs:	Requirement 5.5
Uses cases:	This use case can only be performed as an extension of Play Level.

Use Case Name:	Read About
Actors:	Player
Description:	The player can see a list of the game’s authors and the external resources which were used.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1.8 and its sub requirements
Uses cases:	N/A

Use Case Name:	Read Instructions

Actors:	Player
Description:	The player can read the instructions on how to play the game.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1.4 and its sub requirements
Uses cases:	N/A

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:	N/A
Extends:	N/A
Cross-refs:	Requirement 1.3 and its subrequirements
Uses cases:	The Play Level use case includes the Select Level use case, and will be performed after a level is elected.

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:	N/A
Extends:	N/A
Cross-refs:	Requirement 2 and its sub requirements
Uses cases:	N/A

Use Case Name:	View High Scores
Actors:	Player

Description:	The player is able to see the best scores for each level that they completed. The scores for the currently set difficulty are shown.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1.7
Uses cases:	N/A

4.2 Class Diagram

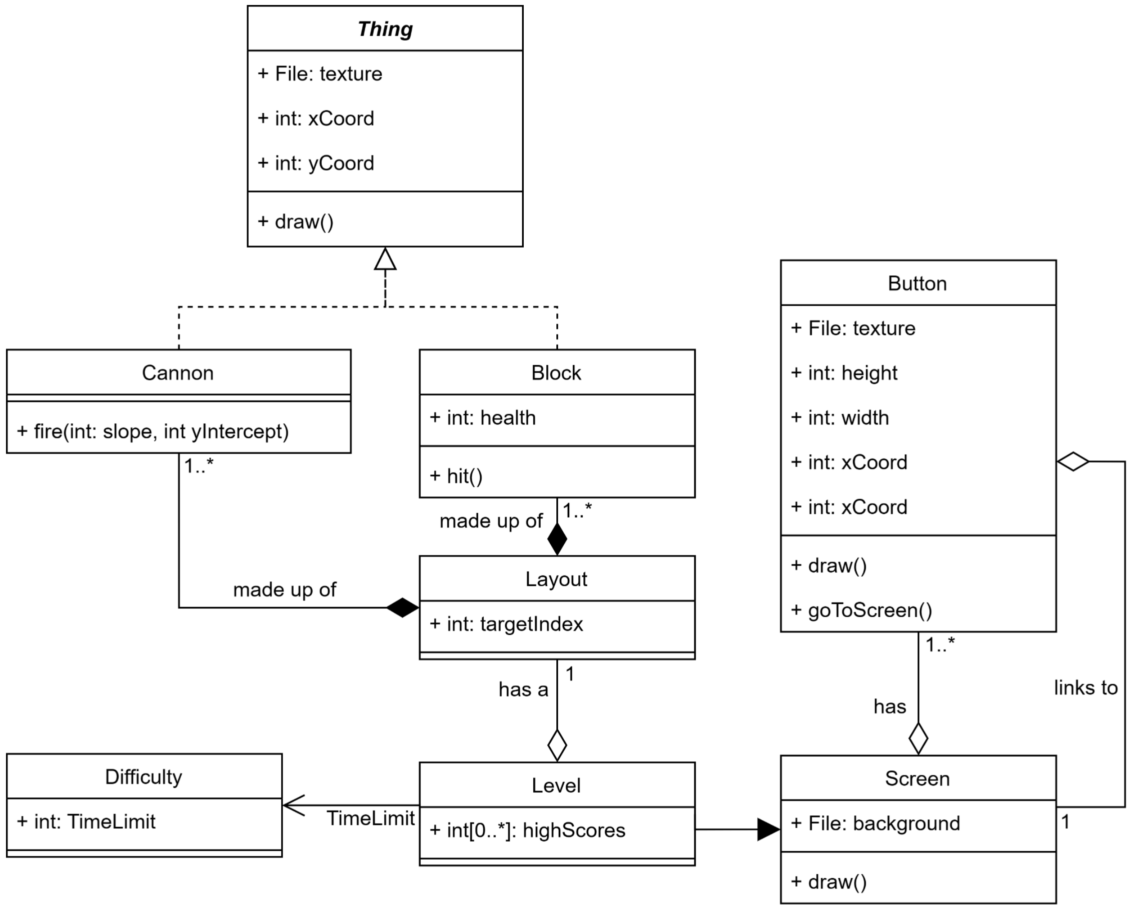


Figure 4: Class Diagram

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 from other attributes. 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.

The Thing class is an abstract base class which represents objects in each level. The Block and Cannon class inherit from it, and a container with these classes makes up the Layout class, with the Layout class also having an index of which block is the target.

The Layout class is one element of the Level class, which represents the level itself. The Difficulty class is a separate class that the Level class knows of, and it holds the time limit to complete the level, which is different across difficulties. The Level class inherits from the Screen class, which represents all screens including the main menu. It also has several instances of the Button class, which are buttons the player clicks to get to another screen. Each instance of the Button class links to an instance of the Screen class.

Element Name		**Description**
Block		Represents the blocks that the castle is made out of.
Attributes		
	health:int	The amount of health a block has. When the health reaches 0 the block is destroyed.
Operations		
	hit():	Called when the block is hit by a cannonball. It reduces the health the block has by 1.
Relationships	The Block class inherits from the Thing Class. A container of instances of the Block and Cannon classes make up the Layout class.	
UML Extensions		

Element Name		**Description**
Button		Represents a button that the user can click to go to another screen.
Attributes		
	texture:File	An image file which will be how the button looks like.
	height:int	The height in pixels of the button.
	linksTo:Screen	The address of an instance of a Screen class with the button links to.
	width:int	The width in pixels of the button.
	xCoord:int	The X coordinate of where on the screen the button will be.
	yCoord:int	The Y coordinate of where on the screen the button will be.
Operations		
	draw():	Called by the draw() method in the Screen class. Draws the button to the screen.
	goToScreen():	Called when the button is clicked, the game goes to the screen the button links to.

Relationships	The linksTo attribute is a pointer to an instance of the Screen class, and it is the screen that the game goes to if this button is clicked. The Screen class also has a container with several instances of the Button class.	
UML Extensions		

Element Name		**Description**
Cannon		Represents the cannon that the user fires at the castle.
Attributes		
Operations		
	fire(slope:int. yIntercept:int):	Called after the user enters the equation. Fires the cannon. The slope and yIntercept parameters are the slope and y-intercept of the cannonball that will be fired respectively.
Relationships	The Cannon class inherits from the Thing Class. A container of instances of the Block and Cannon classes make up the Layout class.	
UML Extensions		

Element Name		**Description**
Difficulty		Represents the difficulty, and holds gameplay information which may vary based on the difficulty.
Attributes		
	TimeLimit:int	The time limit in seconds which the user has to complete a level at the current difficulty.
Operations		
Relationships	The Level class knows about the Difficulty class and takes information such as the time limit from it.	
UML Extensions		

Element Name		**Description**
Layout		Represents all blocks and cannons that are in a level.
Attributes		
	blocks:Block[1..*]	A container holding instances of the Block class.
	cannons:Cannon[1..*]	A container holding instances of the Cannon class.

	targetIndex:int	The index of which block is the target block.
Operations		
Relationships	The Layout class holds several instances of the Block and Cannon class. The Level class has one instance of the Layout class as an attribute.	
UML Extensions		

Element Name		**Description**
Level		Description
Attributes		
	layout:Layout	An instance of the Layout class which holds information about blocks and cannons in the level.
	targetIndex:int[0..*]	A container of all the best scores which were achieved on this level.
Operations		
Relationships	The Level class has a single instance of the Layout class as an attribute, which holds information about blocks and cannons in it. It also knows of the Difficulty class and takes information about gameplay such as the time limit from it. The Level class also inherits from the Screen class.	
UML Extensions		

Element Name		**Description**
Screen		Represents every screen in the game such as the main menu and level select screen.
Attributes		
	background:File	An image file which will be the background of the screen.
	buttons:Button[1-..*]	A container with several instances of the Button class, which will all be drawn on the screen.
Operations		
	draw():	Called when the player enters a screen. Draws the screen and all elements within it such as buttons.
Relationships	The Screen class has a container with several instances of the Button class. Every instance of the Button class also has a pointer to one instance of a Screen class. The Level class inherits from the Screen class.	
UML Extensions		

Element Name		**Description**
Thing		The Thing class represents every object that is part of the game.
Attributes		
	texture:File	An image file which will be how the thing looks like.
	xCoord:int	The X coordinate of where on the screen the thing will be.
	yCoord:int	The Y coordinate of where on the screen the thing will be.
Operations		
	draw():	Called by the draw() method in the Screen class. Draws the thing to the screen.
Relationships	The Block and Cannon class both inherit from the Thing class.	
UML Extensions		

4.3 Sequence Diagram 1: Menus

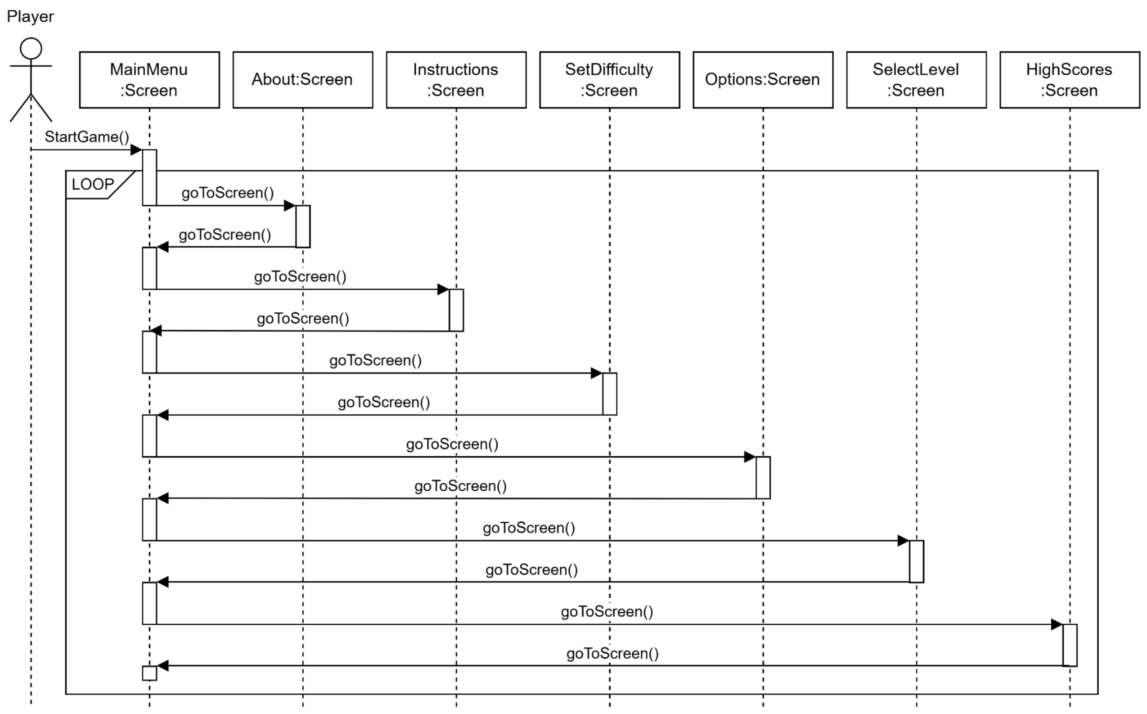


Figure 5: Menu Sequence Diagram

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.

After starting the game, the player sees the Main Menu. From there, they are able to press a button to visit one of six other screens, about, instructions, set difficulty, opinions, select level, and high scores. In each of these screens the player is able to press a back button to return back to the main menu.

4.4 Sequence Diagram 2: Gameplay

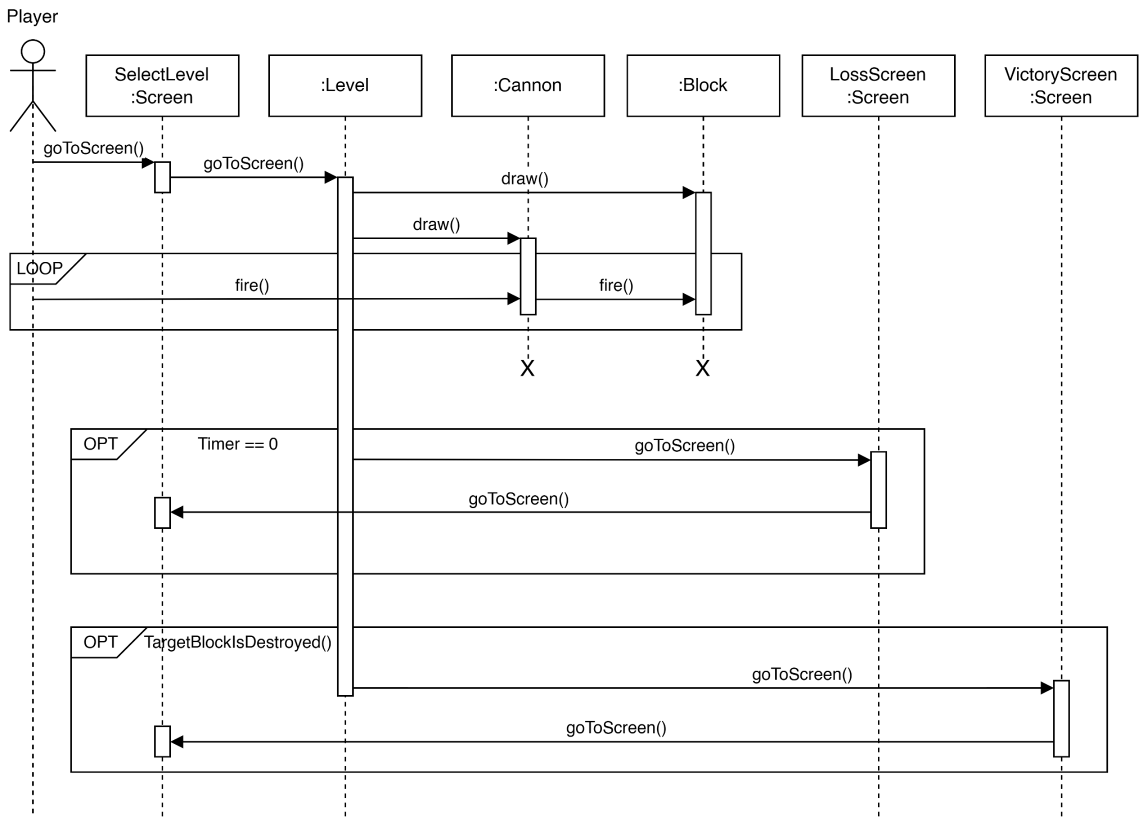


Figure 6: Gameplay Sequence Diagram

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.

The player first goes to the Select level screen and picks a level to play. When they choose a level, the game first draws a cannon and the blocks. The player then fires the cannon by entering an equation, until either the timer runs out, at which point they lose the game, or until they destroy the target block, at which point they win. The player is taken to either the Loss Screen or the Victory Screen, and is then able to return to the main menu from them.

4.5 State Diagram

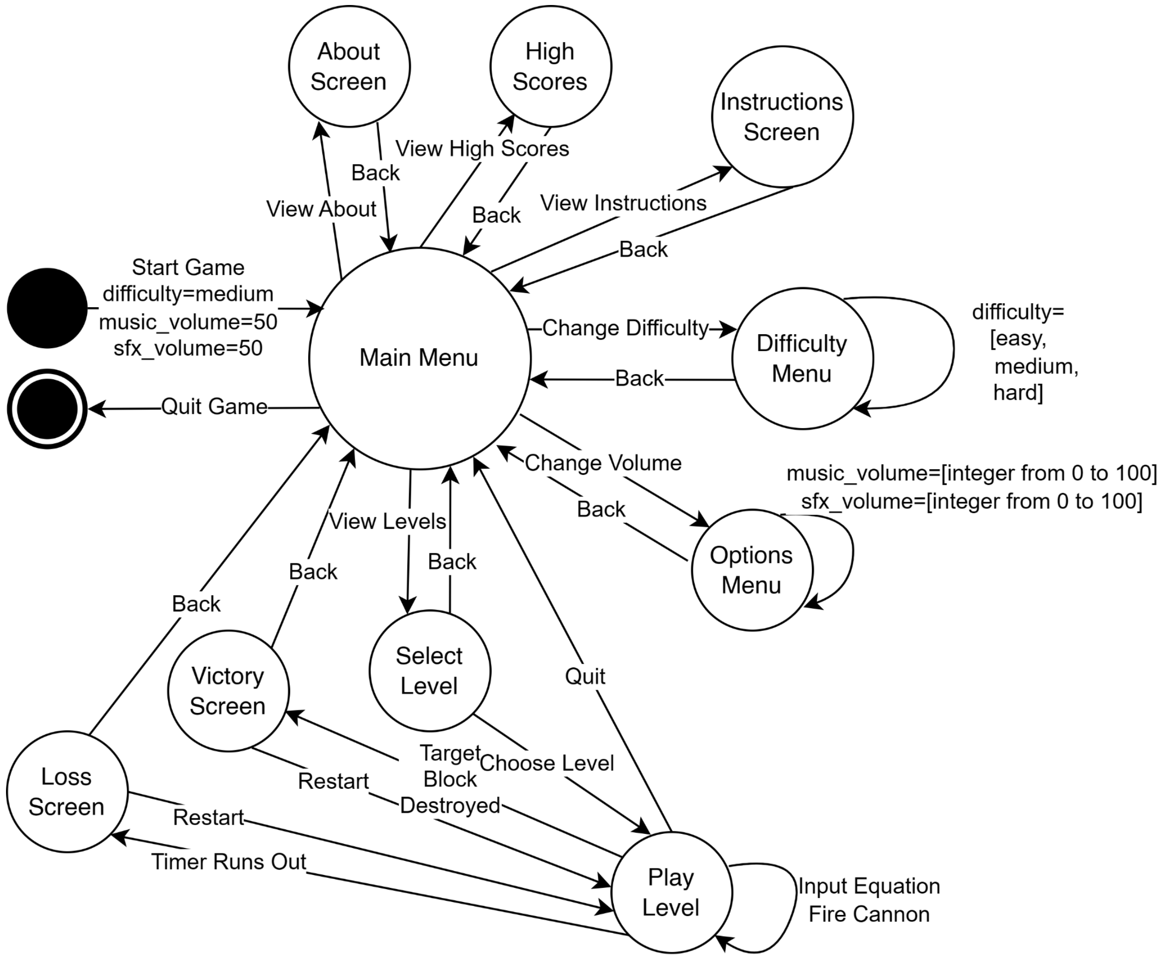


Figure 7: State Diagram

The state diagram shows how the program changes from the input of the user. In the top right, the black circle on top represents the program starting, while the circle below it represents the program ending. Each circle with text 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 done by actions the player does such as button presses. Arrows looping to a state show actions which do not result in states being changed, while doing other changes such as to variables.

After starting the game, the player enters the Main Menu. A quit button allows them to exit the game. From the Main Menu the player is able to press buttons to go to the About Screen, High Scores, Instructions Screen, Difficulty Menu, Options Menu, and Select Level, and return to the Main Menu from all of these. In the Difficulty Menu the player is able to change the difficulty to easy, medium, or hard, with medium being the default. In the Options Menu the player is able to change the music and sound effects volumes from 0 (mute) to 100, with 50 being the default for both.

Finally from the Select Level screen, once they choose a level they enter the Play Level state and play the game, inputting equations to fire the cannon. At any time the player is able to pause the level then return to it or quit the level, returning to the Main Menu. When the target block is destroyed, the player wins the level, but if the timer runs out before then the player loses. The player is then taken to the Victory Screen or Loss Screen respectively, after which they can go back to the Main Menu or restart the level.

5 Prototype

The prototype has three levels, and is meant to show the basic gameplay features which have been described above. All three levels have a cannon on one side of the screen and a differently shaped castle on the other. The player is able to enter an integer value as the slope in the top of the screen to fire the cannon, destroying blocks until either destroying the target block or running out of time.

5.1 How to Run Prototype

There are two versions of the prototype, both of which can be found on the project website [1] by clicking “Play” in the top right corner. There is a downloadable one which can only run on Windows, that needs to be extracted after being downloaded then can be run by going to the “Mathmunition final build” folder and running “My application.exe”. There is also a browser version, which can be played by clicking a link to it in the website.

5.2 Sample Scenarios

When the player opens the game, the main menu shows up, which can be seen below in Figure 8.

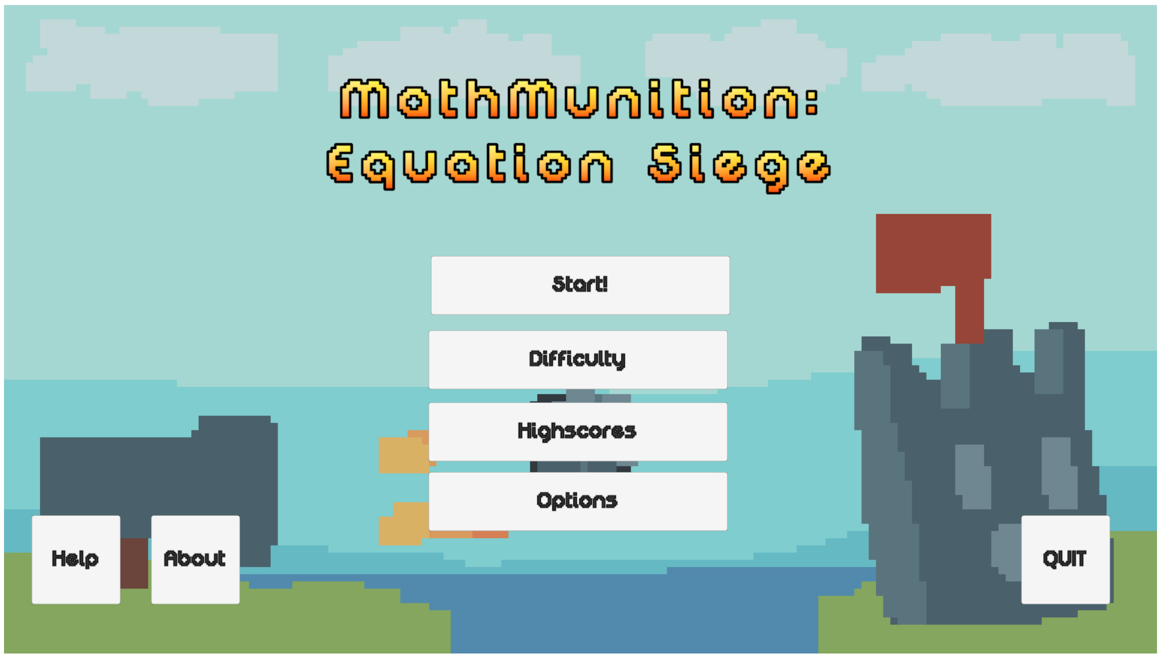
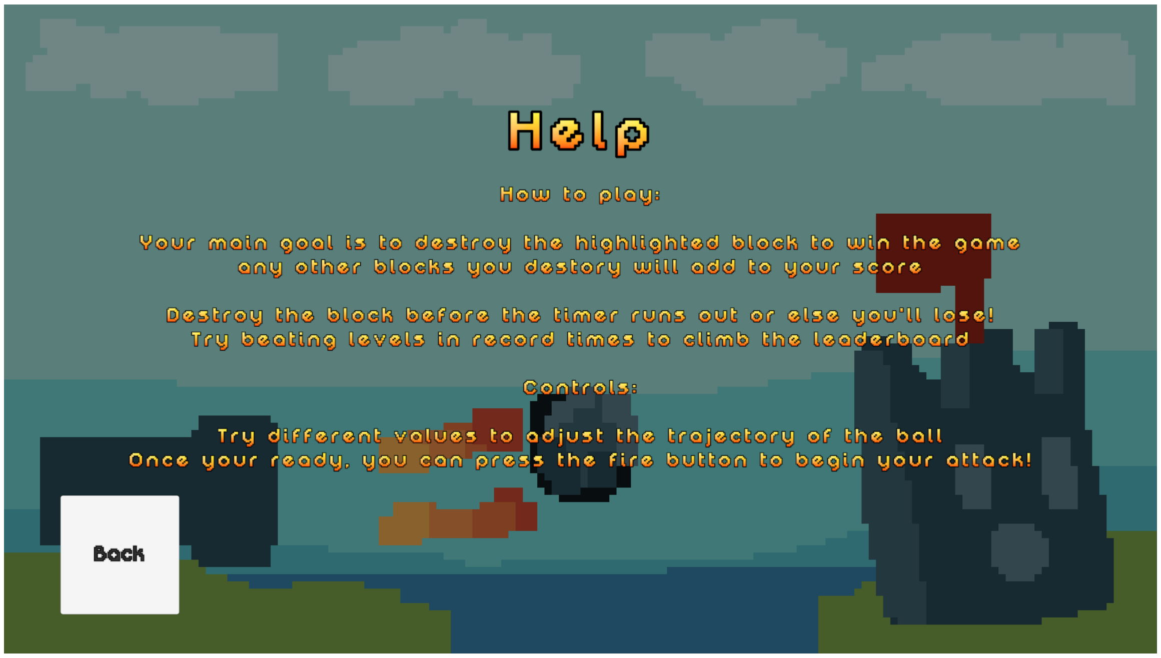


Figure 8: Main Menu

Clicking the “Help” button takes the player to the help screen, which shows instructions on how to play the game, and can be seen in Figure 9. The back button takes the player back to the main menu.


Help

How to play:

Your main goal is to destroy the highlighted block to win the game
any other blocks you destory will add to your score

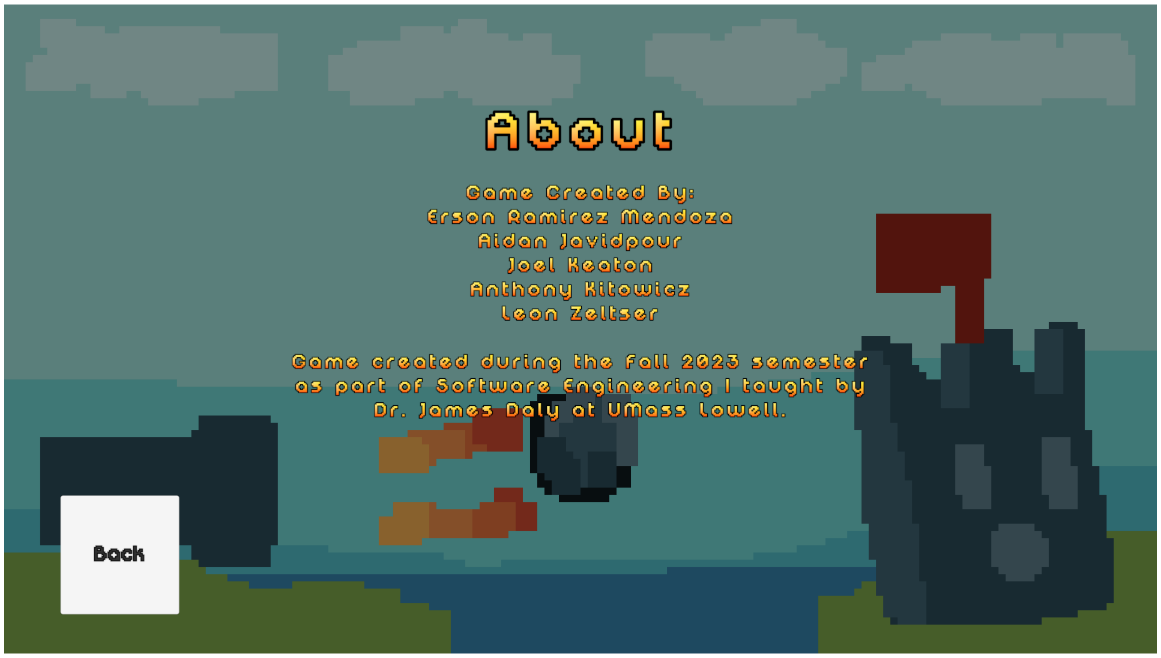
Destroy the block before the timer runs out or else you'll lose!
Try beating levels in record times to climb the leaderboard

Controls:

Try different values to adjust the trajectory of the ball
Once your ready, you can press the fire button to begin your attack!

Figure 9: Help Screen

Clicking the “About” button takes the player to the about screen, which shows information about the game’s authors and when it was made. It can be seen in Figure 10. The back button takes the player back to the main menu.


About

Game Created By:
Erson Ramirez Mendoza
Aidan Javidpour
Joel Keaton
Anthony Kitowicz
Leon Zeltser

Game created during the Fall 2023 semester
as part of Software Engineering I taught by
Dr. James Daly at UMass Lowell.

Figure 10: About Screen

Clicking the “Options” button takes the player to the options screen, as seen in Figure 11. The player is able to adjust the volume of music and sound effects with the two sliders. The back button takes the player back to the main menu.

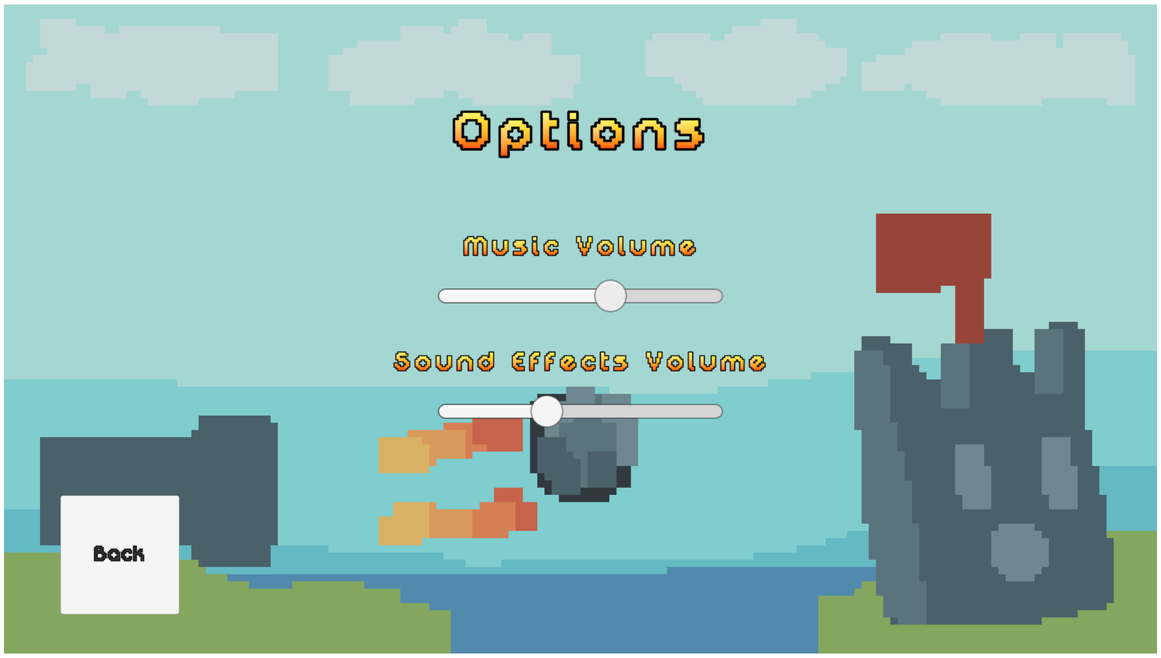


Figure 11: Options Screen

Clicking the “Difficulty” button takes the player to the difficulty select screen, which can be seen in Figure 12, and where the player can choose between easy, medium, and hard by pressing one of the buttons, with medium being the default. The back button takes the player back to the main menu.

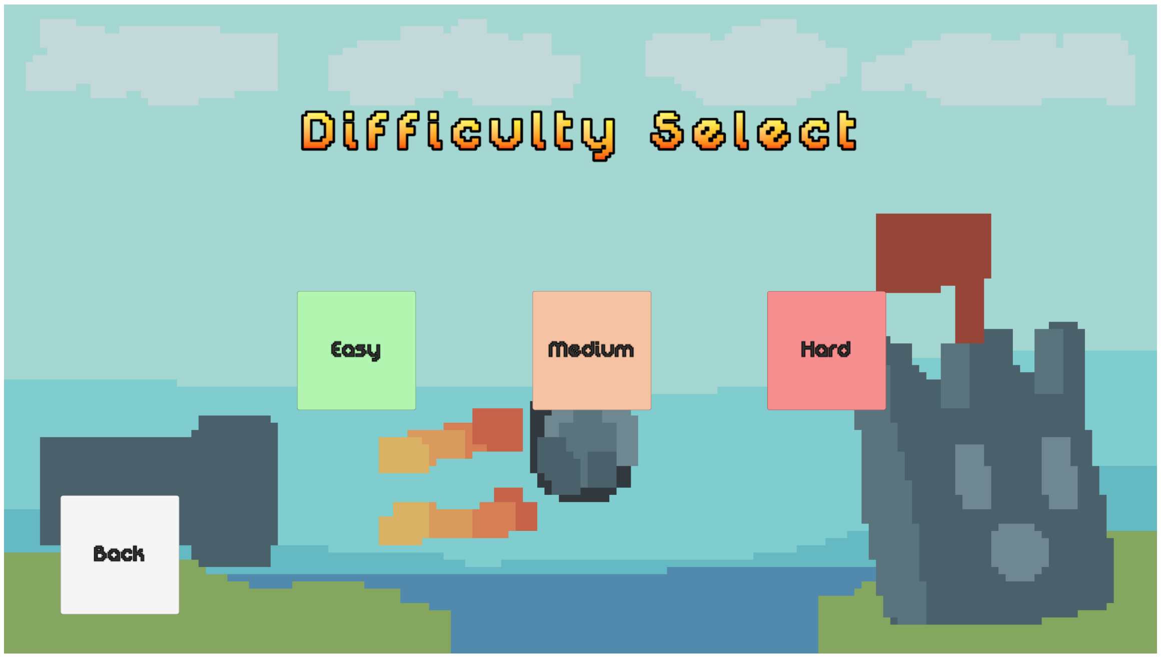


Figure 12: Difficulty Select Screen

Clicking the “Start!” button takes the player to the level select screen, shown in Figure 14. They can then choose one of the three levels to play. The back button takes the player back to the main menu.

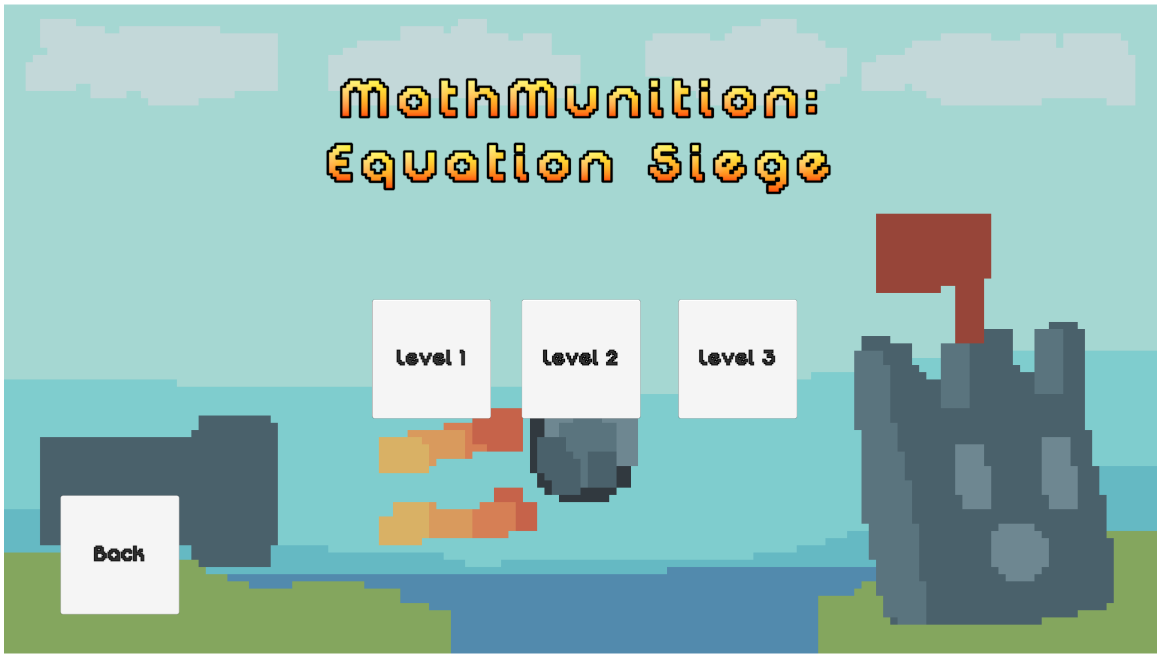


Figure 14: Level Select Screen

Figure 15 shows level 1 after several shots were fired and blocks destroyed. The gray blocks have a health of 3, the purple a health of 2, and the red a health of 1. The top left corner has the time remaining to complete the level, the top right has space to enter an equation, and the button to fire the cannon is on the bottom right. The score is shown on the bottom left, along with a button to return to the main menu. The target block is in the middle and has a star on it.

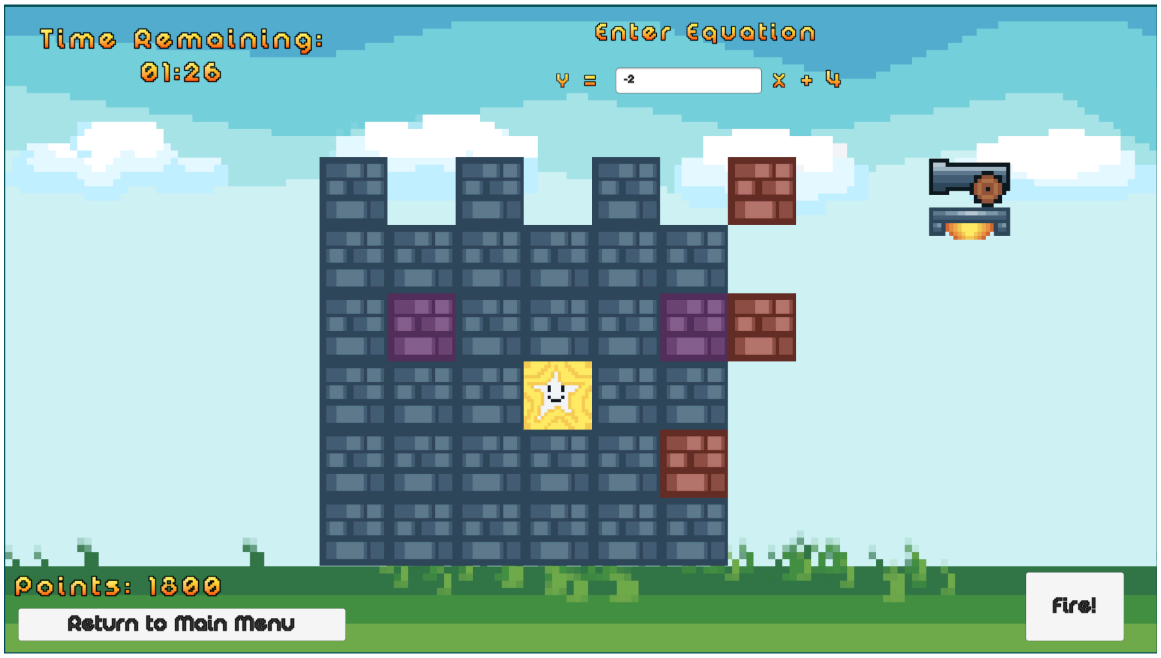


Figure 15: Level 1

Figure 16 shows the castle after more shots were fired, and more blocks were destroyed.

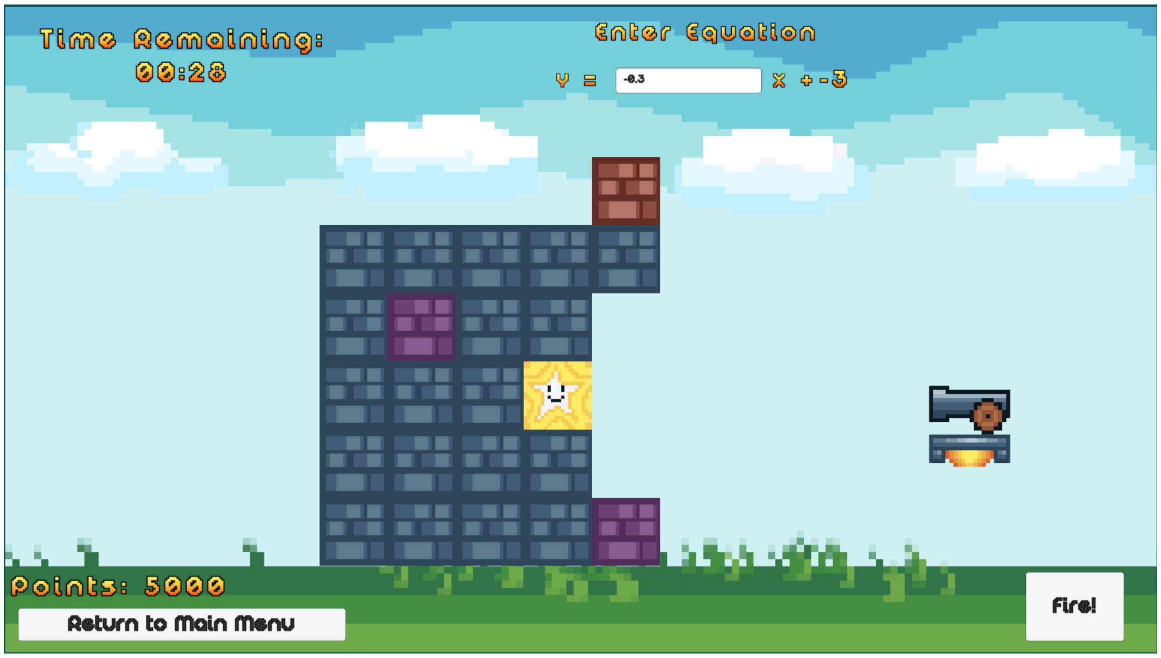


Figure 16: Level 1 Continued

Figure 17 shows the victory screen after the target block is destroyed. It shows the score, and the player can restart the level or return to the main menu.

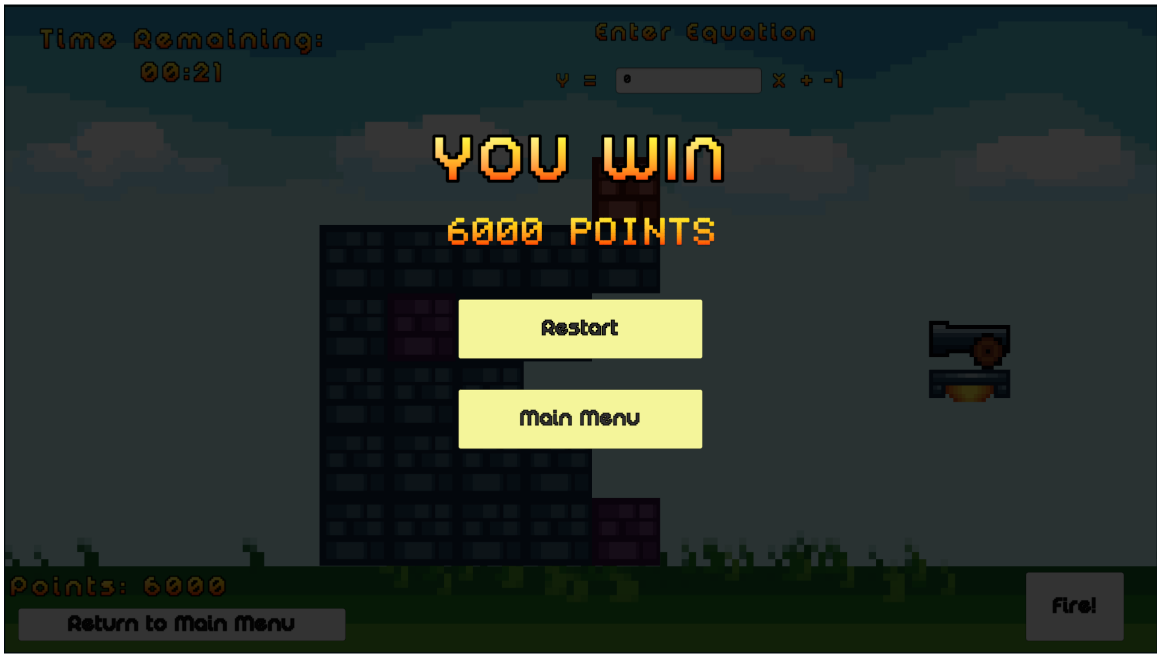


Figure 17: Victory Screen

Figure 18 shows a different level, this time with the cannon on the other side of the screen, and a different layout for the castle.

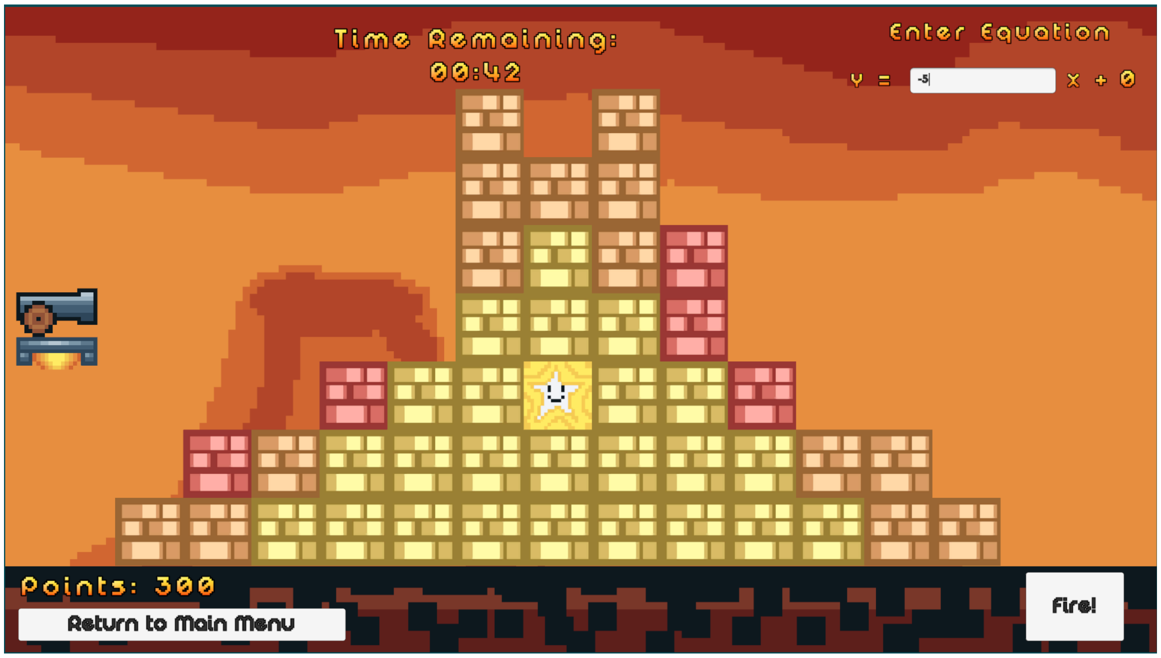


Figure 18: Another level

If the player runs out of time, they will be taken to the game over or loss screen, shown in Figure 19. It also shows the score, and has buttons to restart the level or return to the main menu.

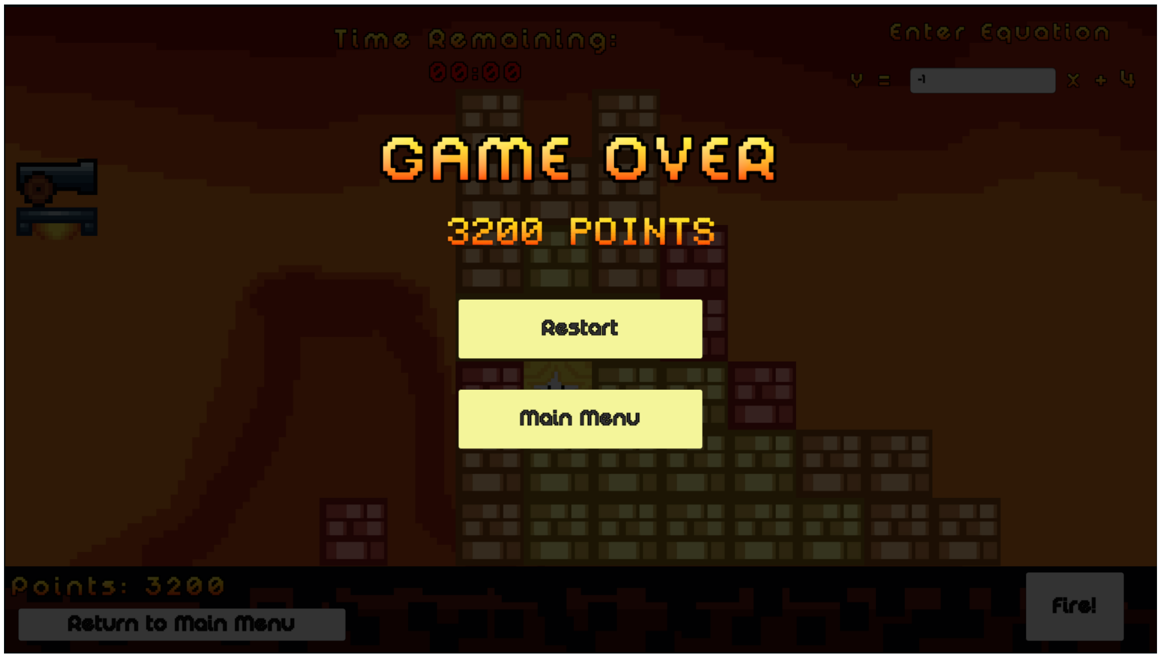


Figure 19: Loss Screen

The high score screen, shown in Figure 20, is accessed by pressing the “Highscores” button on the main menu. It has a list of the best scores in each of the difficulties. The back button takes the player back to the main menu.

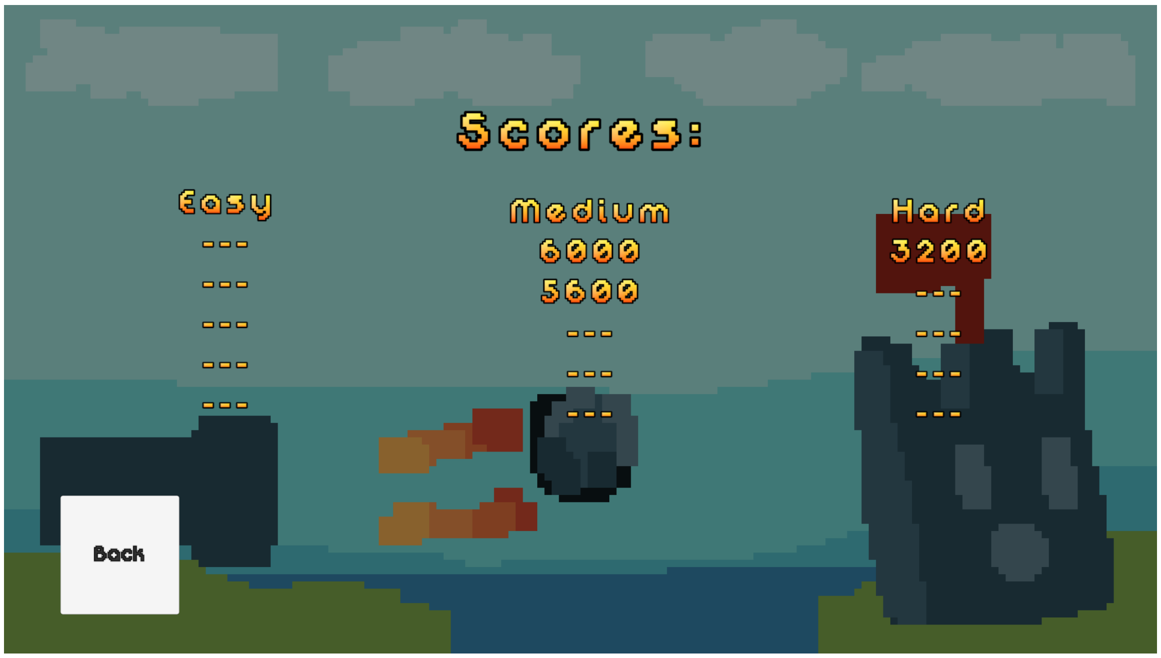


Figure 20: High Score screen

6 References

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

[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] "IEEE Recommended Practice for Software Requirements Specifications," in IEEE Std 830-1998 , vol., no., pp.1-40, 20 Oct. 1998, doi: 10.1109/IEEESTD.1998.88286. https://ieeexplore.ieee.org/document/720574

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.