Software Requirements Specification (SRS)
FACTORBS

Team: Group 6
Authors: David Genis, Tom Kaplan, Shuyu Lin, Juan Fernando Ruiz, Nathaniel Garret
Customer: Teachers and students of 4th through 6th grade.
Instructor: Dr. James Daly

1 Introduction

The following subsections will have information that details the following FACTORBS project at hand. This document will cover a wide range of topics, beginning with an introduction in this current subsection followed by an overall (big picture) description of the document. After that, the document will be broken down into specific sections that go into detail the different aspects of FACTORBS and its many moving parts: the Specific Requirements for it, followed by its Modeling Requirements and the abstract representation that comes with it. A prototype is then shown that connects these designs, which is then followed up by the resources that were used within the project and the points of contact therein.

1.1 Purpose

The purpose of this document is multifaceted. One of the primary objectives is to cleanly and professionally describe FACTORBS for the developers to use and see. Within the SRS are the technical specifications and diagrams needed to bridge the gap from idea to a fully functioning prototype, and the SRS serves as a crucial tool to facilitate that process. Its purpose is to outline the project that was undertaken in detail and then describe FACTORBS from the perspectives of the developers, and then present that to its primary audience and prospective customers: teachers looking to understand the game, and school curriculum administrators, and for them to see the benefits that could be gleaned from educational games such as this one.

1.2 Scope

FACTORBS is a video game for 4-6th grade teachers to use as a teaching tool for students to learn early factoring. The game will feature a Cannon that the player can move by moving their mouse cursor around the screen and can shoot Orbs. The Orbs are shot at a trail of Orbs that follow a set path around the game map. The player must use their factoring skills to know where to shoot the Orbs.

1.3 Definitions, acronyms, and abbreviations

- FACTORBS: Name of the game
- HW: Hardware
- SW: Software
- Snake: a snake consists of exactly one snake tail and at least one snake orbs, moves along the path.
- Snake Tail: a snake tail is an entity that locates at the end of a snake. It is used to identify the location of the snake.

- Path: a path is a series of predefined locations that the snake moves along.
- Cannon: a cannon is an entity that can fire projectile orb and be controlled by the player.
- Fire: the frontmost orb in the cannon can be turned into an aimed projectile that attempts to collide with a snake orb.
- Snake Orb: a snake orb is an orb that has a number. It is controlled by the snake and moves along the path.
- Projectile Orb: a projectile orb is an orb that has a number and can be fired from the cannon. It can interact with the snake when it collides with a snake.
- Game Master: the game master is the central controller of the system.
- GoDot: a cross-platform, free and open-source game engine used in the system.
- SRS: Software Requirements Specification.

1.4 Organization

The remainder of the SRS document contains the following:

Section 2 will detail the overall description of the project, in detail. It will then be followed by 6 subsections. 2.1 will outline the context for the project, and how the different interfaces will play out. 2.2 Will focus on the different functions, both those that the user is able to interact with, and those in the background that they cannot see, as well as the corresponding diagrams. Subsection 2.3 specifies what the expectations for the user are. 2.4 will inform of the different possible software, hardware, or regulatory constraints that may appear, while 2.5 will detail the hardware, software, and user assumptions. Lastly, 2.6 encompasses the apportioning of requirements.

Section 3 consists of the Specific Requirements for the FACTORBS project.

Section 4 is dedicated to the Modeling Requirements, i.e. all of the different diagrams used to create FACTORBS. This consists of several kinds of diagrams: Use Case, Class, Sequence, and State.

Section 5 contains a preview/snapshot of the prototype. It is what the user will see when they play the game, how to play the game and the several scenarios that might play out in a session.

Lastly, section 6 contains the references used in the SRS document, and section 7 details the points of contact for FACTORBS.

2 Overall Description

This section will go over the overall description of the product in several sections.

1. The Product Perspective will be a technical description of the product.
2. The Product Functions will describe what the product does when used, which includes a simple description of how the game is interacted with.
3. User characteristics will describe what is expected of the users of the product to play and/or teach the game.
4. Constraints will describe technical constraints for the product.
5. Assumptions and Dependencies will describe the system, hardware, software, and user requirements for the product.
6. Apportioning of Requirements will describe what is beyond the scope of this stage of the project.

2.1 Product Perspective

Context: This product is a game meant to be played by students in a school context where teachers give them prior instructions on factoring, though they (and teachers/parents) can also play the game outside of classrooms.

User Interfaces: The user must have a mouse and may use an optional keyboard. The keyboard is used to press the ESC key and access the pause menu. The mouse is used to move the cursor around the screen, click the screen when needed, move the cannon around, clicking the left mouse to shoot orbs, and right mouse to switch orbs in the cannon. They also require a screen to visualize the game.

Hardware Interfaces: The game should be able to interface with a display, keyboard, and mouse.

Software Interfaces: There is no external software needed outside of the game itself.

Communication Interfaces: There are no communication interfaces, as the game does not need to communicate with other devices.

Memory: Other than the game files themselves, the game will only need to store a small file for the high score on that device. Not much space is needed for said file.

Operations: The game can run on computers with Windows, MacOS, or Linux.

Site Adaptation Requirements: The game has no needed site adaptation requirements.

2.2 Product Functions

The game has a central cannon that the user controls with their mouse and has two orbs in it at a time. They can shoot with the left-clicker and change the two orbs with the right-clicker. The game also has a trail around the map with a snake of orbs going through it. All orbs (in the cannon and the snake) have numbers on them. The speed of the orbs is dependent on the difficulty the player chooses. The actual gameplay entails shooting the orbs from the cannon at the snake and trying to divide the orbs in the snake.

When an orb is shot from the cannon and collides with a snake orb, it checks if the cannon orb is a factor of the snake orb. If it is *not*, the cannon orb is inserted behind the collided orb. If it *is* a factor, it splits the collided orb into multiple orbs of the value of *snake orb* divided by *cannon orb* and have a number of them equal to the value of *cannon orb*. When the value of any snake orb is equal to 1, it disappears. The goal of the game is to remove all the snake orbs, leading to the win screen. If the snake gets to the end of its trail, the player loses the game.

As previously stated, the goal of the game is to teach simple factoring to students from grade 4-6 in an engaging way. A high-level goal diagram is pictured below in figure 1:

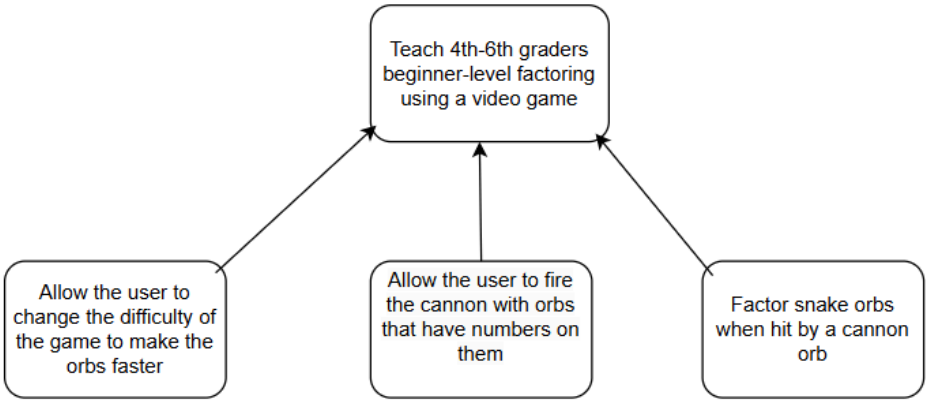


Figure 1: High-Level Goal Diagram

The High-Level goal diagram above shows the primary goal as teaching factoring using the video game, as well as the three secondary goals that will achieve the primary one. Those goals are (1) allow the user to change the difficulty, (2) allow the user to fire their cannon, and (2) factor the snake orbs in real time when a cannon hits them.

2.3 User Characteristics

The product is intended for students from 4th-6th grade. They should have a basic understanding of factoring and division that corresponds to the aforementioned grade range, as a beginner level is necessary to play the game. The user should have at least a beginner's understanding of English to read the menu words. The game is not intended to teach factoring by itself, as the teachers are expected to teach students simple factoring beforehand and use the game as a tool for instruction. The user should also be able to see well enough to read numbers and comprehend the game.

2.4 Constraints

The product must be a prototype of an educational video game for students from 4th - 8th grade. The game must be in English. The product must also not have any bugs or apparent issues.

2.5 Assumptions and Dependencies

The users must have a computer with one of the following operating systems: Windows, MaxOS, or Linux. The computer should be able to run the game and be connected to a display, mouse, and keyboard. It is also expected, but not required, to be used by students or their teachers and parents.

2.6 Apportioning of Requirements

There will not be any additional maps or levels, as the developers will only create one for this prototype. This may be an avenue for expansion in later releases. There will also not be any power-ups or different game modes other than the basic game.

3 Specific Requirements

1. When the game is first launched, it shall display a menu.
   1.1. The menu shall allow the player to start the game.
   1.2. The menu shall allow the player to quit the game.
   1.3. The menu shall allow the player to enter an options screen.
      1.3.1. The options screen shall allow the player to adjust the volume.
2. When the game is started, the game shall generate and display a ‘snake’.
   2.1. A snake is a finite sequence of ‘orbs’ that move along a set track.
      2.1.1. Orbs are circles containing a non-prime number.
   2.2. A snake’s movement speed shall be relative to its distance from the end of the track, fastest at the beginning and slowest at the end.
   2.3. The game shall end when any orb reaches the end of the track.
      2.3.1. When the game ends, it shall return the player to the menu.
   2.4. The game shall generate a new snake under the following circumstances:
      i. When there are no snakes on the screen
      ii. After a certain amount of time has passed since the last snake was generated.
3. The player shall be able to control the ‘cannon’.
   3.1. When the game is started, the cannon shall generate and display two orbs in sequence.
      3.1.1. When the cannon generates an orb, its number shall be a factor, prime or otherwise, of an orb in a currently-displayed snake, excluding one and the orb’s number.
   3.2. The cannon shall rotate to follow the mouse pointer.
   3.3. The cannon shall swap the order of its two orbs when the player presses the right mouse button.
   3.4. The cannon shall ‘fire’ when the player presses the left mouse button.
      3.4.1. When the cannon fires, it shall launch its frontmost orb in the direction of the mouse pointer.
   3.5. After firing an orb, the cannon shall move the remaining orb up and generate another orb to fill the secondary spot.
4. When an orb fired from the cannon collides with a snake orb, one of the following shall happen:
      i. If the projectile orb is not a factor of the collided orb, the projectile orb shall be inserted into the snake behind the collided orb.

ii. If the projectile orb is a factor of the collided orb, then the value of the collided orb and any contiguous orbs with the same value shall be divided by the projectile orb's value, and the projectile orb is inserted into the snake behind the collided orb.

4.1. All orbs with a value of one shall be removed from the snake and all following orbs shall be moved back to fill the space.

5. The game shall record the number of balls cleared during the current session as the player's score.

5.1. The current score shall be displayed to the player.

5.2. The game shall keep a record of the highest score.

6. The player shall be able to pause the game after it has begun.

6.1. The game shall suspend play while it is paused.

6.2. The game shall display a pause menu while it is paused.

6.2.1. The pause menu shall allow the player to resume the game.

6.2.2. The pause menu shall allow the player to restart the game.

6.2.3. The pause menu shall allow the player to return to the menu.

7. The game shall play background music while it is ongoing.

8. The game shall play sound effects under the following circumstances:

i. When an orb is fired

ii. When an orb collides with a snake

iii. When a sequence of orbs is cleared

iv. When the game is over

v. When a button is pressed in the menu

4 Modeling Requirements

4.1 Use Case Diagram

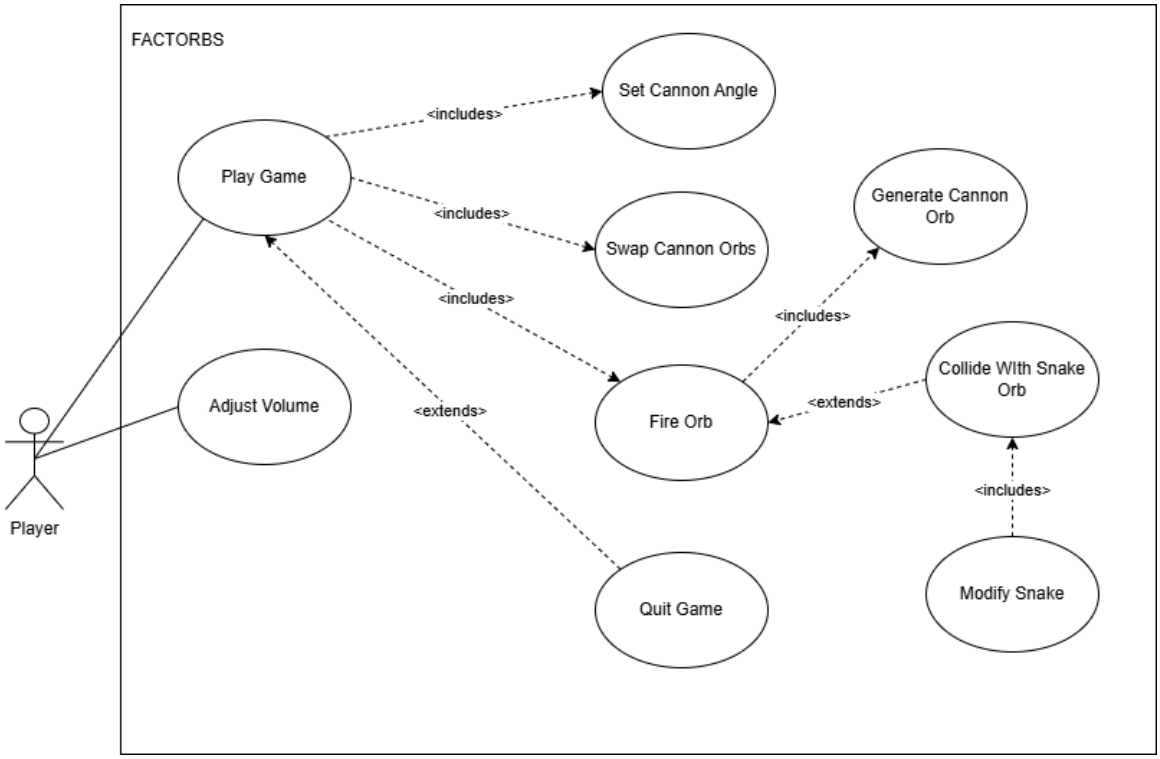


Figure 4.1: Use Case Diagram

The use case diagram provides an abstract visualization of how the user would interact with the game. Each circle represents an action a user could take, with solid lines connecting them to users who might take them and dashed lines connecting them to related actions. Dashed ‘extends’ lines are used for edge cases or variations on an existing action, while ‘includes’ lines are used for actions shared between several use cases

Use Case Name:	Play Game
Actors:	Player
Description:	The player selects the “start game” option in the main menu. The gameplay will then initialize.
Type:	Primary
Includes:	None
Extends:	None

Cross-refs:	Requirement 1.1
Uses cases:	None

Use Case Name:	Set cannon angle
Actors:	Player
Description:	The player moves their mouse while in game play.
Type:	Primary
Includes:	Play Game
Extends:	Return to Menu
Cross-refs:	Requirement 3.2
Uses cases:	None

Use Case Name:	Swap Cannon Orbs
Actors:	Player
Description:	The player provides the “right click” input in the game play. The ammos in the cannon queue will then be swapped.
Type:	Secondary
Includes:	Play game
Extends:	None
Cross-refs:	Requirement 3.3
Uses cases:	None

Use Case Name:	Fire orb
Actors:	Player
Description:	The player provides the “left click” input while in the game play. The game will create a projectile orb which moves in the direction of the current cannon angle.
Type:	Primary
Includes:	Play Game
Extends:	None
Cross-refs:	Requirement 3.4
Uses cases:	None

Use Case Name:	Generate Cannon Orb
Actors:	Player
Description:	After the player fires the cannon. The cannon copies and replaces the first ammo with the second ammo. Then, the cannon generates the second ammo.
Type:	Primary
Includes:	Fire Orb
Extends:	None
Cross-refs:	Requirement 3.5
Uses cases:	None

Use Case Name:	Collide With Snake Orb
Actors:	Player
Description:	A projectile orb collides with a snake orb. This projectile orb will then be removed. The snake will create a snake orb with the same number of the projectile orb and insert it into the snake body.
Type:	Secondary
Includes:	None
Extends:	Fire Orb
Cross-refs:	requirement 4
Uses cases:	None

Use Case Name:	Modify Snake
Actors:	Player
Description:	When a projectile orb collides with a snake. The adjacent orb will be divided by the number of the projectile orb (if they are divisible).
Type:	Secondary
Includes:	None
Extends:	Collide With Snake Orb
Cross-refs:	Requirement 4.ii
Uses cases:	None

Use Case Name:	Quit game
Actors:	Player
Description:	The player selects the “quit game” option in the main menu. The program will clean up used resources and close.
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirement 1.2
Uses cases:	None

Use Case Name:	Check score
Actors:	Player
Description:	The player can check the historic highest score.
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement 5.1
Uses cases:	None

Use Case Name:	Adjust volume
Actors:	Player
Description:	The player selects the “options” option in the main menu. Then adjust the game volume.
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement 1.3.1
Uses cases:	None

4.2 Class Diagram

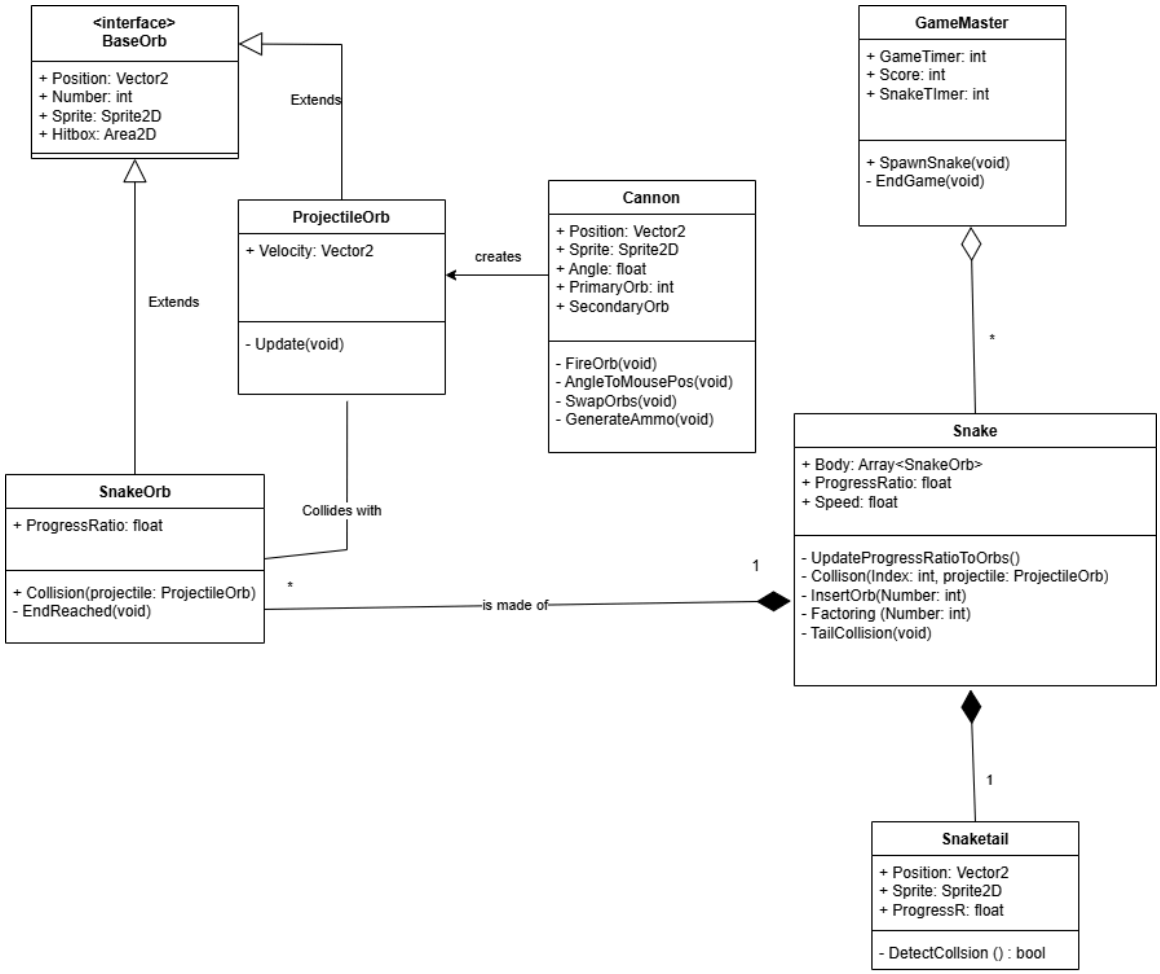


Figure 4.2: Class Diagram

The class diagram shows the structure of the game's classes, as well as the relationships between them. Each box is a class, and has subdivisions containing its name, attributes, and methods. Attributes and methods are marked with icons showing their privacy, with a plus denoting a public method or attribute and a minus denoting a private method or attribute. The arrows between boxes represent the classes' relationships, with a name and direction denoting the nature of the relationship. This nature is further specified by the type of arrow; white arrows are standard relationships, white diamonds are a 'composes' relationship, and black diamonds are a 'belongs' relationship.

Class: BaseOrb <interface>			
Description: The interface, virtual representation of an orb.			
Attributes			
Visibility	Name	Data Type	Description
Public	Position	Vector2	The current position of the entity.
Public	Number	int	An integer number that the entity contains.
Public	Sprite	Sprite2D	2D bitmap that represents the image of the entity.
Public	Hitbox	Area2D	2D area that defines the boundaries of the entity. Used for collision detection.
Methods			
Name		Visibility	Description
None			

Class: ProjectileOrb			
Description: Orbs that are created when the player fires ammo from the cannon. It moves at a constant velocity toward the direction the cannon faces when fired. It can collide with a snake orb and be inserted into the snake body.			
Attributes			
Visibility	Name	Data Type	Description
Public	Velocity	Vector2	The velocity of the entity. The entity moves accordingly in every update.

Methods			
Visibility	Name	Return Type	Description
Private	Update	void	Update the location of the entity using the velocity.

Class: Cannon			
Description: The cannon that the player has control over. It rotates its direction based on the player's mouse movement. It can fire projectile orb when the player provides "left click" input, and swap the ammos in the queue when the player provides "right click" input.			
Attributes			
Visibility	Name	Data Type	Description
Public	Position	Vector2	The current position of the entity.
Public	Sprite	Sprite2D	2D bitmap that represents the image of the entity.
Public	Angle	float	The current direction the entity faces.
Public	PrimaryOrb	int	An integer that the next fired projectile orb would have.
Public	SecondaryOrb	int	An integer that the orb after the next fired projectile orb would have.
Methods			
Visibility	Name	Return Type	Description
Private	FireOrb	void	Creating a projectile orb that has the integer equal to "PrimaryOrb" attribute. The orb would move along the direction based on the current "Angle"

			attribute. Copy and replace the value in “PrimaryOrb” from “SecondaryOrb”.
Private	AngleToMousePos	void	Rotate the cannon to face the direction based on the player’s mouse movement.
Private	swapOrbs	void	Swap the values of “PrimaryOrb” and “SecondaryOrb” attributes.
Private	GenerateAmmo	void	Generate a new value for “SecondaryOrb” attribute.

Class: SnakeOrb			
Description: The body of the snake. It can be created by either the snake generator or by colliding a projectile orb with the snake.			
Attributes			
Visibility	Name	Data Type	Description
Public	ProgressRatio	float	A ratio between 0 to 1, represents the progress of the entity on the path. 0 at the beginning, 1 at the end.
Methods			
Visibility	Name	Return Type	Description
Public	Collision	void	Collides with a projectile orb. If the number of this entity is divisible by the number of the projectile, divided by that number.
Private	EndReached	bool	Return if the entity has reached the end of the path.

Class: Snaketail			
Description: The tail of the snake. It is considered the essential and heart of the snake entity. Used to keep track of the snake's progress.			
Attributes			
Visibility	Name	Data Type	Description
Public	Position	Vector2	The current position of the entity.
Public	Sprite	Sprite2D	2D bitmap that represents the image of the entity.
Public	ProgressRatio	float	A ratio between 0 to 1, represents the progress of the entity on the path. 0 at the beginning, 1 at the end.
Methods			
Visibility	Name	Return Type	Description
Private	DetectCollision	bool	Return if the entity has collided with another snake.

Class: Snake			
Description: A snake consists of exactly one snake tail and at least one snake orb that moves along the predefined path. All snake orbs in the same snake are adjacent to others and closer to the end of the path, whereas the snaketail is closer to the beginning of the path. The closer the snaketail is to the end of the path, the slower the snake moves.			
Attributes			
Visibility	Name	Data Type	Description

Public	Tail	Snaketail	The tail of the snake. The progress ratio of this entity represents the location of the snake.
Public	Body	Array<SnakeOrb>	An array of snake orbs. This is considered the body of the snake.
Public	ProgressRatio	float	The progress ratio of the snake.
Public	Speed	float	The speed of the snake. This is considered a progress ratio per update.
Methods			
Visibility	Name	Return Type	Description
Private	UpdateProgressRatio	void	Moving the snake by updates the progress ratio
Private	Collision	void	The snake collides with a projectile orb. This would trigger InsertOrb and Factoring.
Private	InsertOrb	void	Instantiate and insert a new snake orb from the collided projectile orb.
Private	Factoring	void	Check if the number of the collided projectile orb shares a common factor of the adjacent snake orb. If so, divide them with the number.
Private	TailCollision	void	The snake collides with another snake (The snake tail of the front snake collides with the snake body of the rear snake). Combine the two collided snakes into one snake.

Class: Game Master			
Description: The central controller of the game play. This entity is unique and can only exist one instance at the same time per game play. It is responsible for keeping track of the time since the game started, the score, and the time since the last snake was generated.			
Attributes			
Visibility	Name	Data Type	Description
Public	GameTimer	int	The time in second since the beginning of the game.
Public	Score	int	The local score of this gameplay.
Public	snakeTimer	int	The time in second since the last snake was generated.
Methods			
Visibility	Name	Return Type	Description
Public	spawnSnake	void	Spawn a new snake from the beginning of the path.
Private	EndGame	void	End the current game play. Notify the player that a snake has reached the end of the path and direct the player back to the main menu.

4.3 Sequence Diagrams

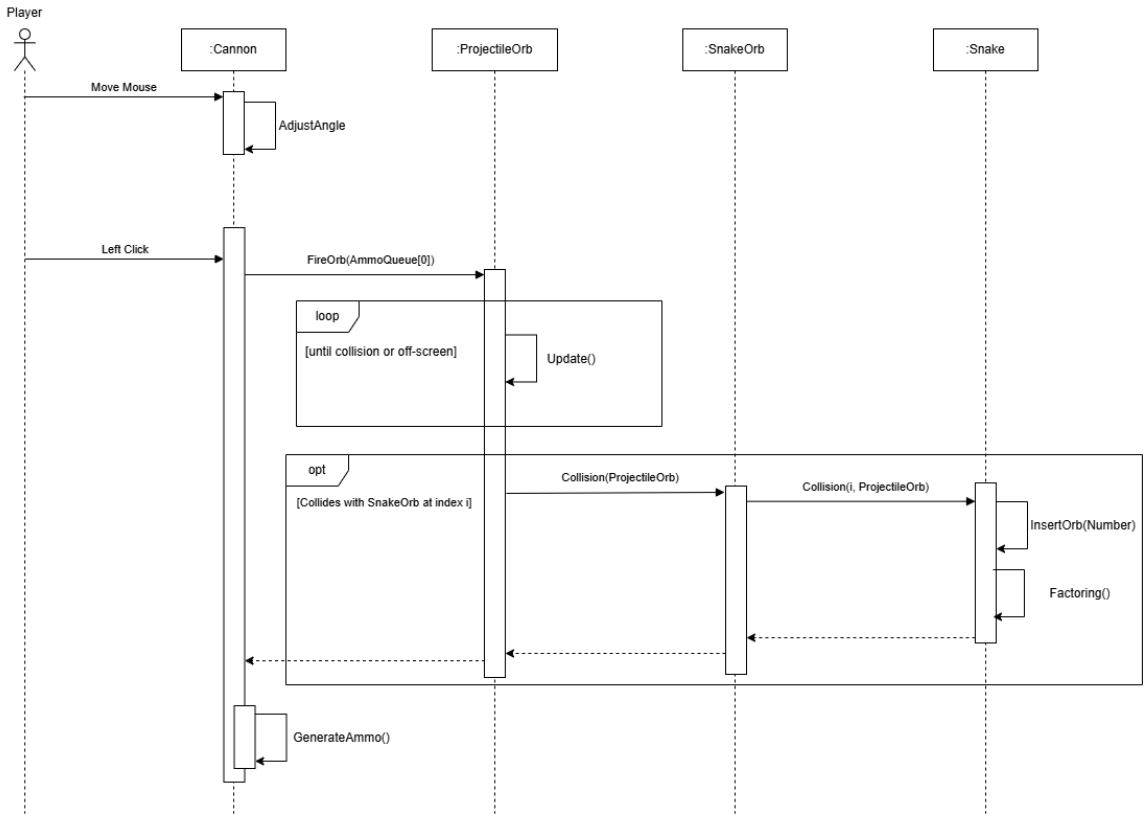


Figure 4.3.: Sequence Diagram I

The first sequence diagram depicts the class interactions that occur when the player fires a projectile orb at a snake. Each box with a descending dotted line represents an object in the game, and each rectangle along that line is a process being executed. The arrows between lines are actions or functions starting a process in another object. Loop boxes represent repeated actions, while opt boxes represent optional steps that only occur under specific scenarios. The general structure is that the cannon listens for a mouse click, then generates a ProjectileOrb, which stays in a movement update loop. If it collides with a SnakeOrb it notifies the Snake, which handles the collision results internally.

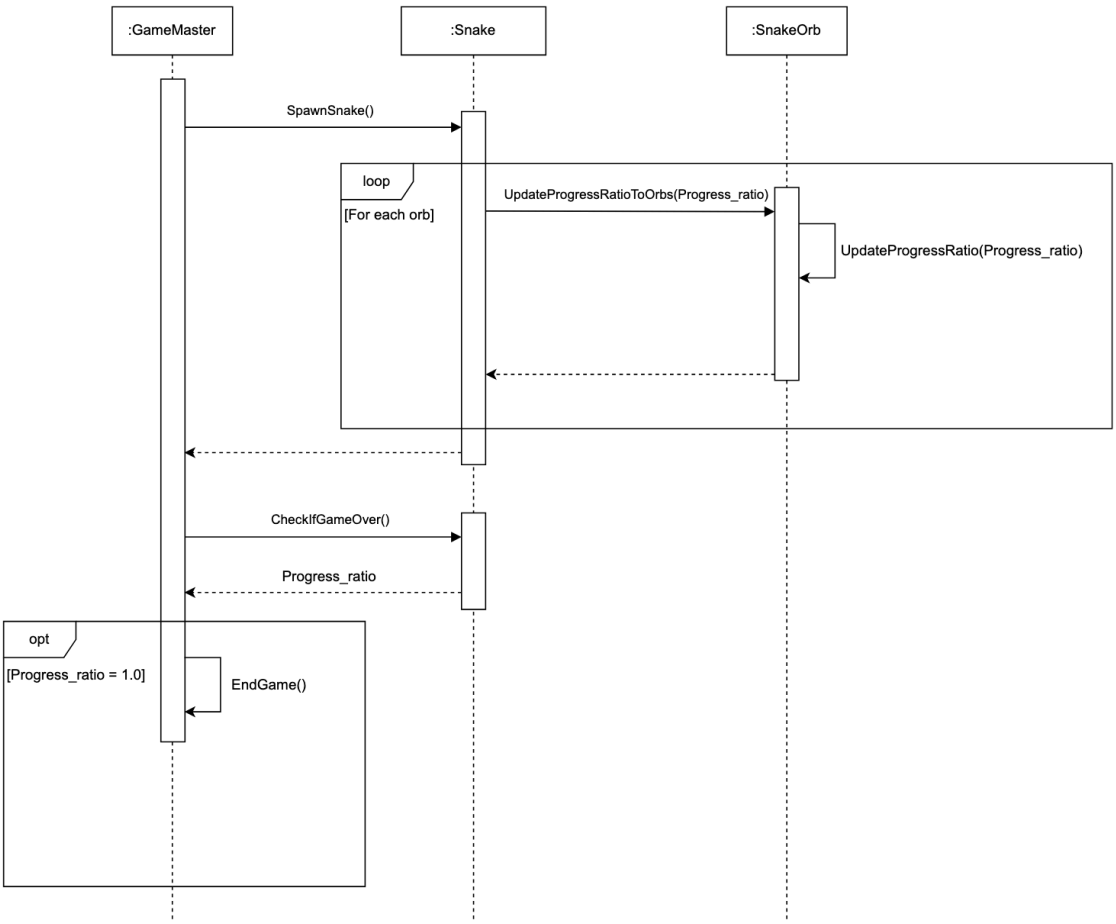
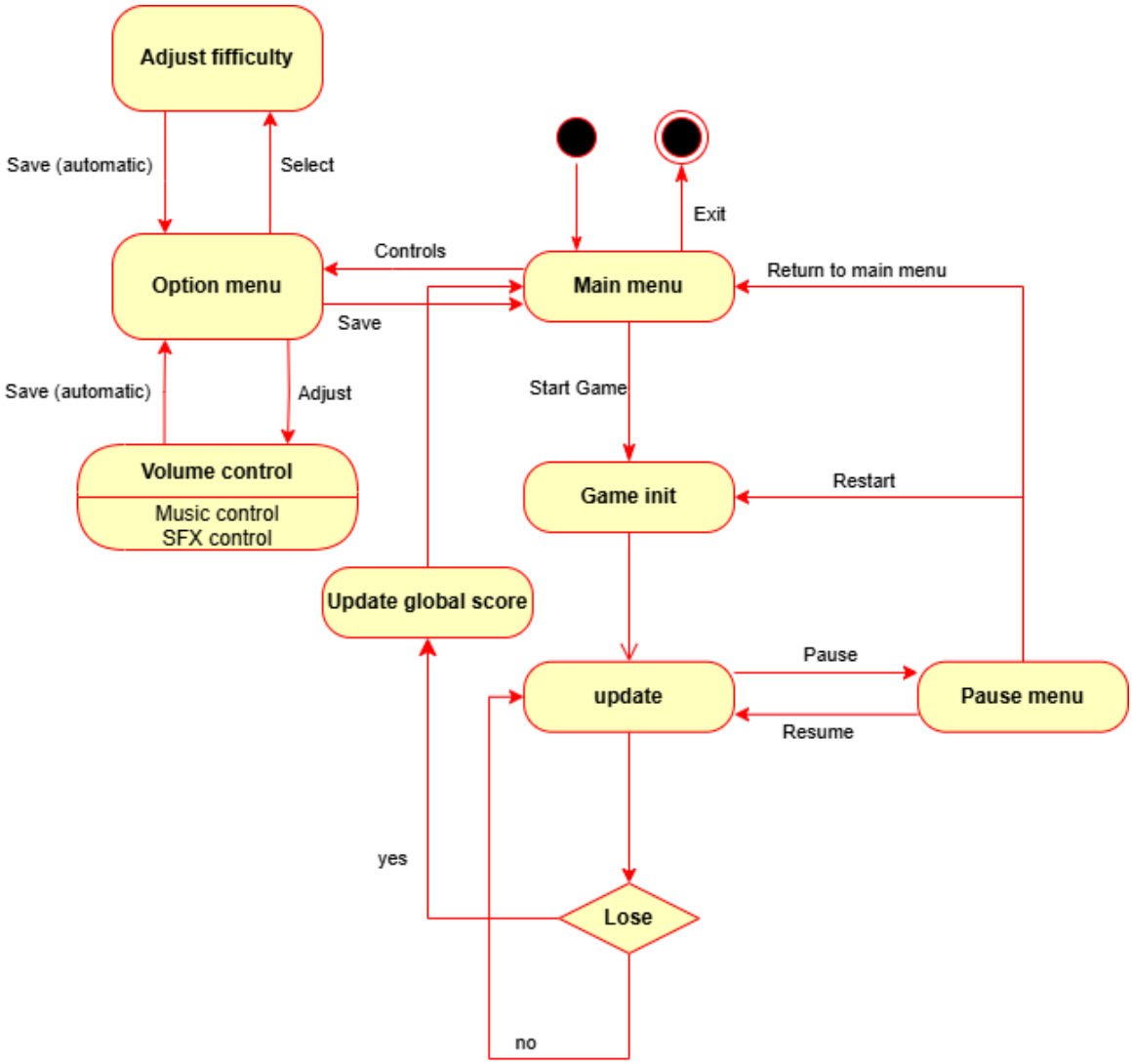


Figure 4.4: Sequence Diagram II

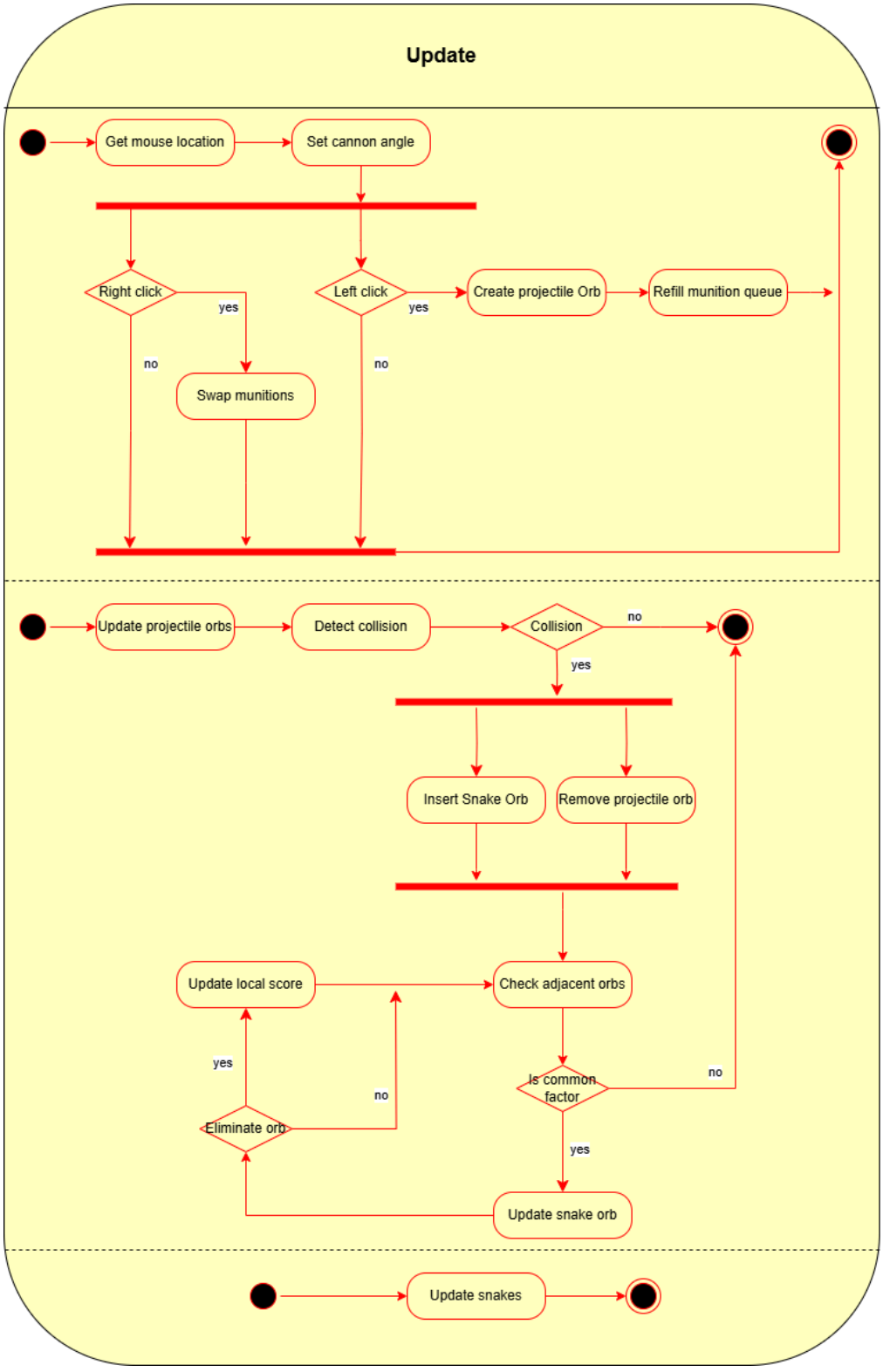
The second sequence diagram depicts the class interactions managed by the GameMaster, and uses the same general structure and components as the first sequence diagram. The GameMaster can create a new Snake, which then populates itself with SnakeOrbs. It also watches Snakes to see when they reach the end of the path, and executes the EndGame procedure.

4.4 State Diagram

4.4.1 Main



4.4.2 Update


Update
Get mouse location
Set cannon angle
Right click
yes
no
Swap munitions
Left click
yes
no
Create projectile Orb
Refill munition queue
Update projectile orbs
Detect collision
Collision
no
yes
Insert Snake Orb
Remove projectile orb
Update local score
Check adjacent orbs
yes
no
Eliminate orb
Is common factor
no
yes
Update snake orb
Update snakes

The state diagrams depict the program as a finite automata, passing control between a series of states based on internal logic. Each diagram starts at the plain black circle and ends at the ringed circle, with rectangles representing potential states, diamonds representing decision points, and horizontal bars representing forks and joins in which multiple paths are taken separately. The first diagram is for the menu, and shows how different sub-menus and menu based actions can be reached from each part of the menu. The second diagram is for various update loops in the game itself, including cannon control, orb collision, and snake movement.

5 Prototype

The prototype will open to a main menu, from which the player can navigate to the options menu, start the game, or quit. In the options menu, the player can control the master, music, and sound effects volume using sliders; as well as return to the main menu. The game itself consists of ‘snakes’ moving along a set path on the screen. Snakes are composed of one or more ‘orbs’, and each orb contains a numerical value. There is also a stationary cannon, which the player controls using the mouse. The player can move the mouse to aim the cannon, right-click to swap the currently loaded and secondary orb, and left-click to fire the currently loaded orb in the aimed direction. The player can clear snake orbs by hitting them with their factors, which divides the number of those orbs. Once an orb reaches a value of 1, it is cleared and removed from the snake, increasing the player’s score. If a snake reaches the end of the path, the game ends and the player's score is displayed and saved.

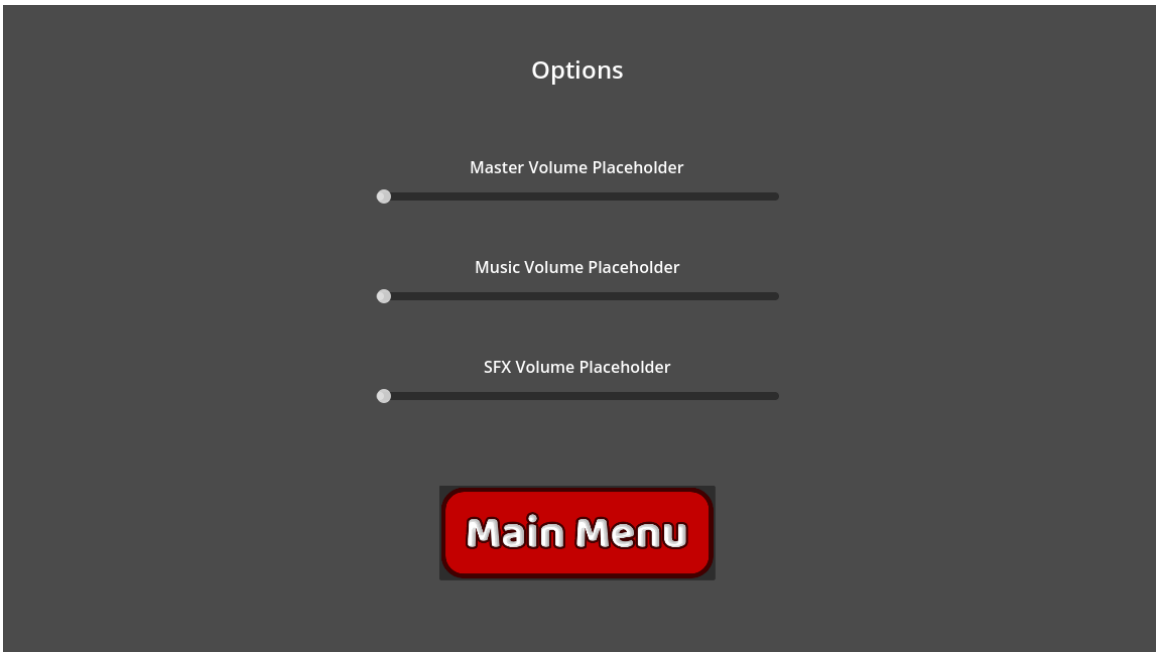
5.1 How to Run Prototype

There are two ways to run the prototype. The first is by playing the embedded browser version that will be on the team's website using Godot's HTML 5 web build feature. The second is by downloading the game on the team's website. This will download an executable that can be clicked and played by the user.

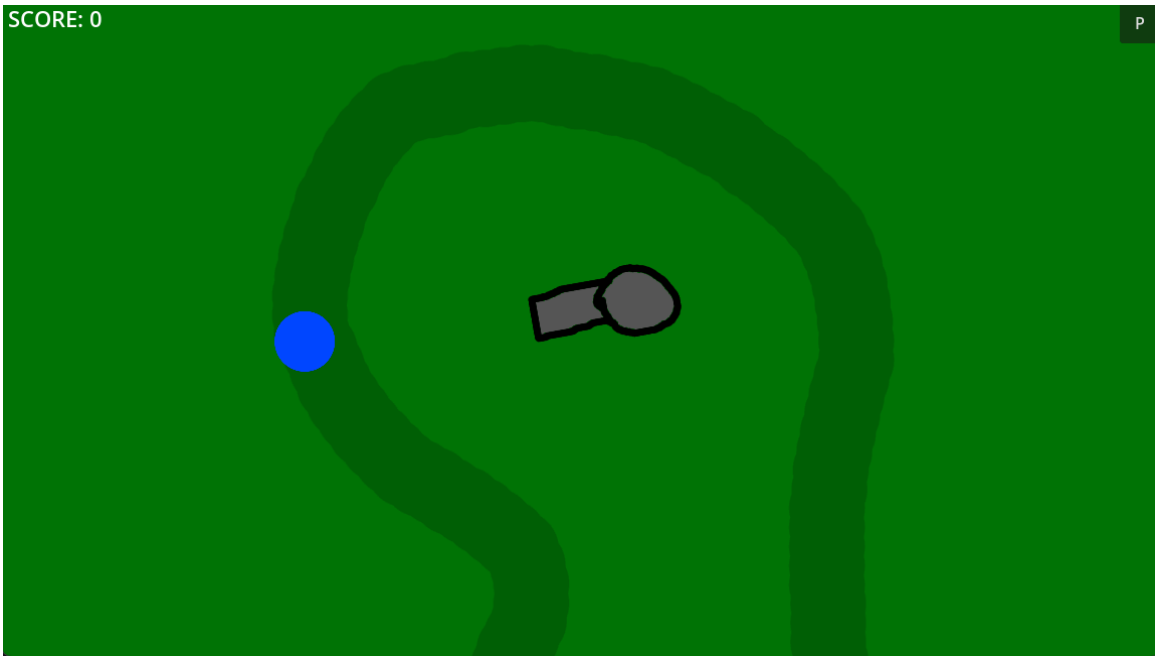
5.2 Sample Scenarios

The main menu will allow the player to start the game, navigate to the options menu, and quit.

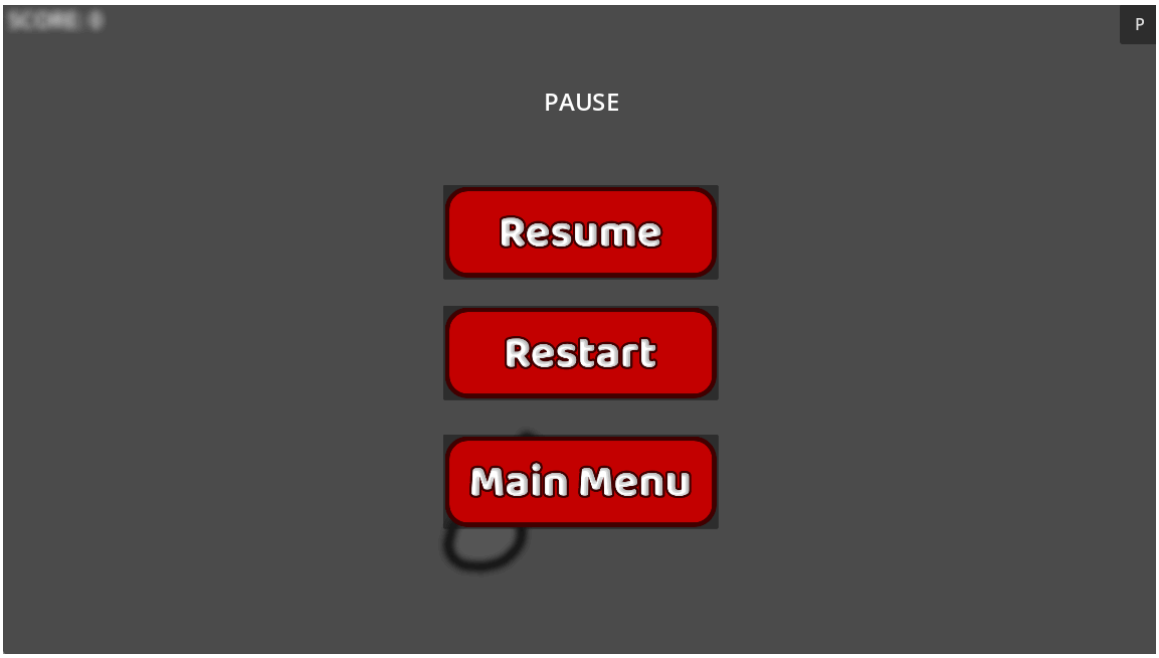
From the options menu, the player can adjust the master, music, and sound effects volume using sliders, or return to the main menu.



While the game is running, the player can control the cannon's direcetion by moving the mouse.



While the game is paused, the player can access the pause menu. From here, the player can resume the game, restart the game, or return to the main menu.



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 Curriculum Framework. (2017). MATHEMATICS Grades Pre-Kindergarten to 12. “, pp. 47, Malden, MA. Massachusetts Mathematics Curriculum Framework — 2017

[3] Common Core State Standards Initiative - Preparing America’s Students For College & Career , “Common Core State Standards For Mathematics”, pp. 29, ch. “OPerations and Algebraic Thinking”. Math Standards.indb

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.