Software Requirements Specification (SRS)
FACTORBS

Team: Group 6
Authors: David Genis, Tom Kaplan, Shuyu Lin, Juan Fernando Ruiz, Nathaniel Garret
Customers: Middle School Teachers and students in grades 4 through 6.
Instructor: Dr. James Daly

1 Introduction

The following subsections will have information about FACTORBS, an educational computer game that aims to teach and reinforce factoring to 4-6th graders. 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 specify 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 (an educational video game aimed to give young students a head start at a core component of mathematics) for the developers to use and see. Within this software requirements specification are the technical specifications and diagrams needed to bridge the gap from an abstract idea to a fully functioning prototype. The SRS serves as a crucial tool to facilitate this process. The project aims to detail the development of FACTORBS and present it to its target audience—teachers, school curriculum administrators, and young middle school students—highlighting the game's educational potential from both the developers' and users' perspectives.

1.2 Scope

FACTORBS is a video game for 4th to 6th grade teachers to use as a teaching tool for students to learn entry-level factoring. Within the scope of FACTORBS, entry-level factoring will involve the skills of breaking down simple algebraic expressions into their more basic components by finding the different numbers they factor into. This is a foundational skill that will be used throughout all of mathematics and obtaining a strong base foundation will improve their ability within future mathematical courses they will take in the future. The game will achieve this through a gameplay loop that features a cannon that the player can move with their mouse cursor, and left clicking will then shoot orbs containing a number. These orbs are then 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, and in doing so will factor the path orbs down until they disappear. This will help teach factoring by making the players use factoring knowledge in a fun and active learning environment to clear the orbs.

1.3 Definitions, acronyms, and abbreviations

- FACTORBS: Name of the game
- HW: Hardware
- SW: Software
- Orb: A circle that has a number value written on it.
- Snake Orb: A snake orb is an orb that has a number. It is controlled by the snake and moves along the path.
- Snake: A group of snake orbs on a path consisting of exactly one snake tail and at least one snake orb.
- 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.
- 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. It contains 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.

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

2.1 Product Perspective

Factoring is an important skill that students begin to learn in Elementary School and is important for many mathematics concepts later in their student careers. FACTORBS is a game meant to be played by students in a school context where teachers give them prior instructions on factoring, though they, as well as their teachers and parents, can also play the game outside of classrooms.

The user must have a mouse and may use a 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 and move the cannon around. It is also used to shoot orbs from the cannon. The user also requires a screen to display the game.

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 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, the value of the snake orb is divided by the value of the cannon orb. Another orb of that value is then inserted behind that orb. When the value of any snake orb is equal to 1, it disappears. Adjacent orbs of the same value are divided together, and can thus be turned to 1 and disappeared together. 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 the game progresses, the numbers get larger and the snakes get faster, requiring faster and more intentional play.

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

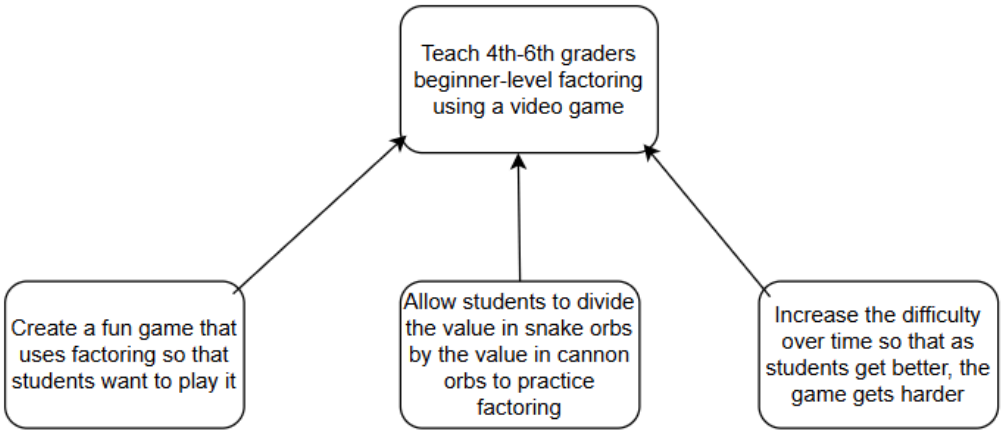


Figure 1: High-Level Goal Diagram

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) make a fun game for students so that they actively want to learn, (2) teach factoring to students by making them interact with the game through factoring, and (3) force students to improve by making the game harder over time.

2.3 User Characteristics

The product is intended for students from 4th to 6th grade, or approximately age 9-12. 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. Lastly, the game could also be played by teachers or parents of the students.

2.4 Constraints

The product must be comprehensible and educational to students from 4th to 8th grade. The game must follow any and all standards set by the Massachusetts Mathematics Curriculum Framework. The game must follow Common Core standards as described by the Common Core State Standards Initiative. It must be safe for consumption by students grades 6 and up for compliance, and must adhere to any restrictions set forth by said rating.

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. If the player wants to play the game through the browser, the computer must be connected to the internet.

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 number.
        2.1.2. When a snake is generated, its orbs contain non-prime numbers.
    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.
    2.4. When two snakes collide with each other, the two snakes shall merge together into one snake that has all the orbs of both snakes, and in the same order.
    2.5. When an orb is removed from the snake, all following orbs shall be moved back to fill the space.
    2.6. 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, the projectile orb is destroyed and one of the following shall happen:
    i. If the projectile orb is not a factor of the collided orb, a new orb shall be inserted into the snake behind the collided orb with a value equal to that of the projectile 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 a new orb of the same value is inserted into the snake behind the collided orb.

    4.1. After a collision, all orbs with a value of one shall be removed from the snake.
5. The game shall keep track of the player's score for the current session, which starts at 0.
    5.1. Each time an orb is cleared by the player, the score increases by 1.
    5.2. The current score shall be displayed to the player, visible on the top-left side of the screen.
    5.3. The game shall keep a record of the highest score.
6. When the game is over, it shall display a game over screen.
    6.1. The game over screen shall display the score for this session as well as the highest score.
    6.2. The game over screen shall allow the player to restart the game.
    6.3. The game over screen shall allow the player to return to the main menu.
7. The player shall be able to pause the game after it has begun.
    7.1. The game shall suspend play while it is paused.
    7.2. The game shall display a pause menu while it is paused.
        7.2.1. The pause menu shall allow the player to resume the game.
        7.2.2. The pause menu shall allow the player to restart the game.
        7.2.3. The pause menu shall allow the player to return to the menu.
8. The game shall play background music while it is ongoing.
9. 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

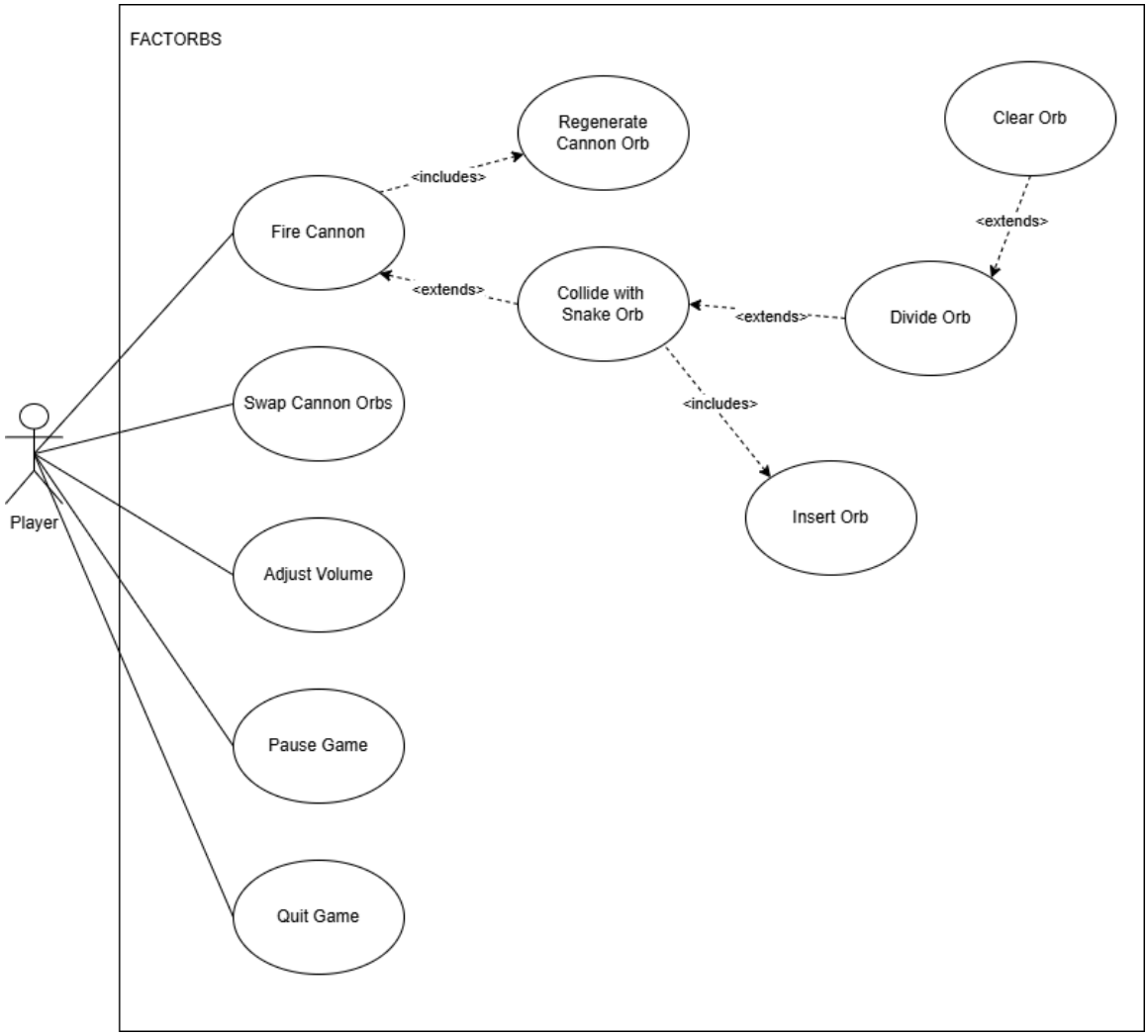


Figure 4.1: Use Case Diagram

The use case diagram provides an abstract visualization of how the user would interact with the game. Each oval 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:	Fire Cannon
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:	Regenerate Cannon Orb
Extends:	None
Cross-refs:	Requirement 3.4
Uses cases:	None

Use Case Name:	Swap Cannon Orbs
Actors:	Player
Description:	The player provides the “right click” input in the game play. The cannon’s primary orb and secondary orb will then be swapped.
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirement 3.3
Uses cases:	None

Use Case Name:	Regenerate Cannon Orb
Actors:	Player
Description:	The cannon’s secondary orb becomes the primary orb. Then, the cannon generates a new secondary orb.
Type:	Secondary
Includes:	None
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, and inserted into the snake at the point of collision. Then, the snake will be modified depending on whether the projectile orb was a factor of the snake orb or not.
Type:	Secondary
Includes:	Insert Orb
Extends:	Fire Cannon
Cross-refs:	Requirement 4
Uses cases:	None

Use Case Name:	Insert Orb
Actors:	Player
Description:	Happens when a projectile orb collides with a snake orb. An orb is inserted into the snake at the point of collision.
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement 4
Uses cases:	None

Use Case Name:	Divide Orb
Actors:	Player
Description:	Happens when a projectile orb collides with a snake orb and the projectile orb is a factor of the snake orb. The snake orb and any contiguous orbs of the same value are divided by the value of the projectile orb.
Type:	Secondary
Includes:	None
Extends:	Collide With Snake Orb
Cross-refs:	Requirement 4.ii
Uses cases:	None

Use Case Name:	Clear Orb
Actors:	Player
Description:	Happens when an orb is divided and its final value is 1. The orb is then deleted from the snake, and the player's score increases.
Type:	Secondary
Includes:	None
Extends:	Divide Orb
Cross-refs:	Requirement 4.1
Uses cases:	None

Use Case Name:	Pause Game
Actors:	Player
Description:	The player presses the "pause" button. The game will be paused and display a pause menu until the player presses the "resume" button.
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirement 7
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:	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

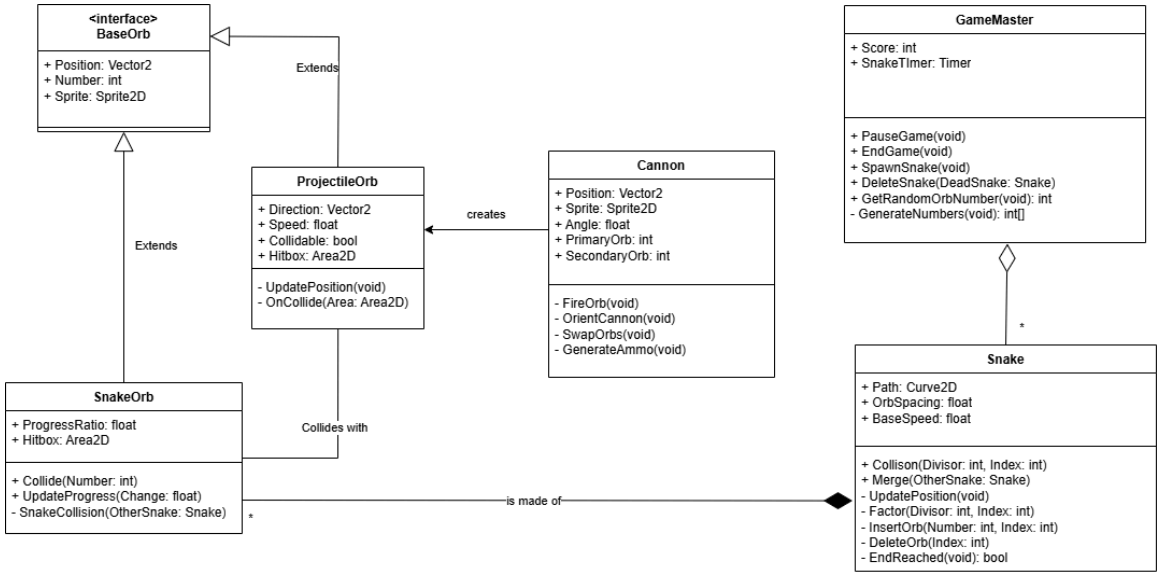


Figure 4.2: Class Diagram

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.
Methods			
Visibility	Name	Return Type	Description
None			

Class: ProjectileOrb			
Description: Orbs that are created when the player fires the cannon. It moves at a constant velocity toward the direction the cannon was facing when fired. It can collide with snake orbs.			
Attributes			
Visibility	Name	Data Type	Description
Public	Direction	Vector2	The unit vector representing the direction of the entity.
Public	Speed	Vector2	The speed of the entity.
Public	Collidable	bool	Defines whether or not the OnCollide function will do anything. Used to avoid

			double-collisions.
Public	Hitbox	Area2D	2D area that defines the boundaries of the entity. Used for collision detection.
Methods			
Visibility	Name	Return Type	Description
Private	UpdatePosition	void	Update the location of the entity based on Direction and Speed.
Private	OnCollide	void	Defines collision behavior. Does nothing if Collidable is set to false. Calls a SnakeOrb's Collide function when colliding with it.

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. Copies and replaces the value in PrimaryOrb from SecondaryOrb, and then calls GenerateAmmo to create a new number for SecondaryOrb.
Private	OrientCannon	void	Set Angle to face the current location of the mouse. Rotate Sprite to match.
Private	SwapOrbs	void	Swap the values of the PrimaryOrb and SecondaryOrb attributes.
Private	GenerateAmmo	void	Generate a new value for the SecondaryOrb attribute.

Class: SnakeOrb			
Description: The orbs that make up the body of the snake. It is created when a Snake is generated or when ProjectileOrb collides with a 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.
Public	Hitbox	Area2D	2D area that defines the boundaries of the entity. Used for collision detection.
Methods			
Visibility	Name	Return Type	Description
Public	Collide	void	Calls the parent Snake's Collision function at the SnakeOrb's index with the number of the .
Public	UpdateProgress	void	Updates ProgressRatio and moves the orb along the path.
Private	SnakeCollision	void	Detects when this orb collides with an orb from another snake, and calls the parent Snake's Merge function to combine the two snakes into one.

Class: Snake			
Description: A snake consists of a chain of one or more SnakeOrbs that moves along a predefined path. The closer the snaketail is to the end of the path, the slower the snake moves.			
Attributes			
Visibility	Name	Data Type	Description
Public	Path	Curve2D	The path along which the SnakeOrbs will move.
Public	OrbSpacing	float	The distance between SnakeOrbs in the chain.
Public	BaseSpeed	float	The speed of the snake, used to update the position of the

			SnakeOrbs.
Methods			
Visibility	Name	Return Type	Description
Private	UpdatePosition	void	Called to move all SnakeOrbs along the path based on BaseSpeed.
Public	Collision	void	Logic for what happens when a projectile orb collides with a SnakeOrb that is part of this Snake. This triggers InsertOrb and possibly Factor.
Private	InsertOrb	void	Instantiate and insert a new SnakeOrb into the Snake.
Private	DeleteOrb	void	Delete a SnakeOrb from the Snake, and move all other SnakeOrbs accordingly to fill the gap.
Private	Factor	void	Called by Collision if the ProjectileOrb is a factor of the collided SnakeOrb. Divides the SnakeOrb and contiguous SnakeOrbs of the same value by the number of the ProjectileOrb, and removes them from the Snake if the resulting value is 1.
Public	Merge	void	Copy all the SnakeOrbs of another Snake and insert them into the front of this one, then delete the other Snake. Called when two Snakes collide to combine them into a single longer 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	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	PauseGame	void	Pauses the game and displays a pause menu through which the player can resume the game, restart, or return to the menu.
Public	SpawnSnake	void	Spawn a new snake from the beginning of the path, using GenerateNumbers to choose numbers for the orbs.
Private	GenerateNumbers	int[]	Returns an array of numbers to be used for generating new snakes. The numbers that can be generated depend on the Score, with higher numbers being generated when Score is higher.
Public	GetRandomOrbNumber	int	Choose an orb at random from any currently active Snake and returns its number.
Public	DeleteSnake	void	Delete a Snake, removing it

			from play.
Private	EndGame	void	End the current game play. Display a game over screen where the player can see their score for this session as well as their highest score. If the game resulted in a new high-score, save it.

4.3 Sequence Diagrams

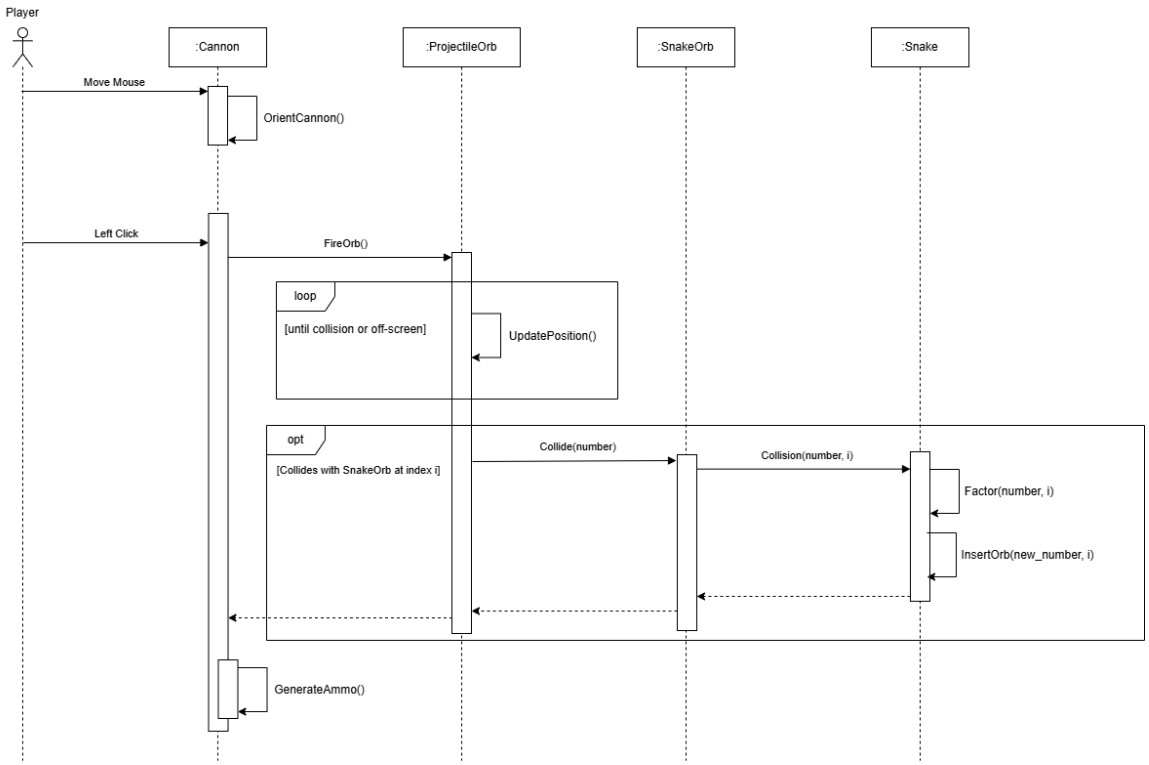


Figure 4.3.: Sequence Diagram I - Fire Cannon and Collision

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.

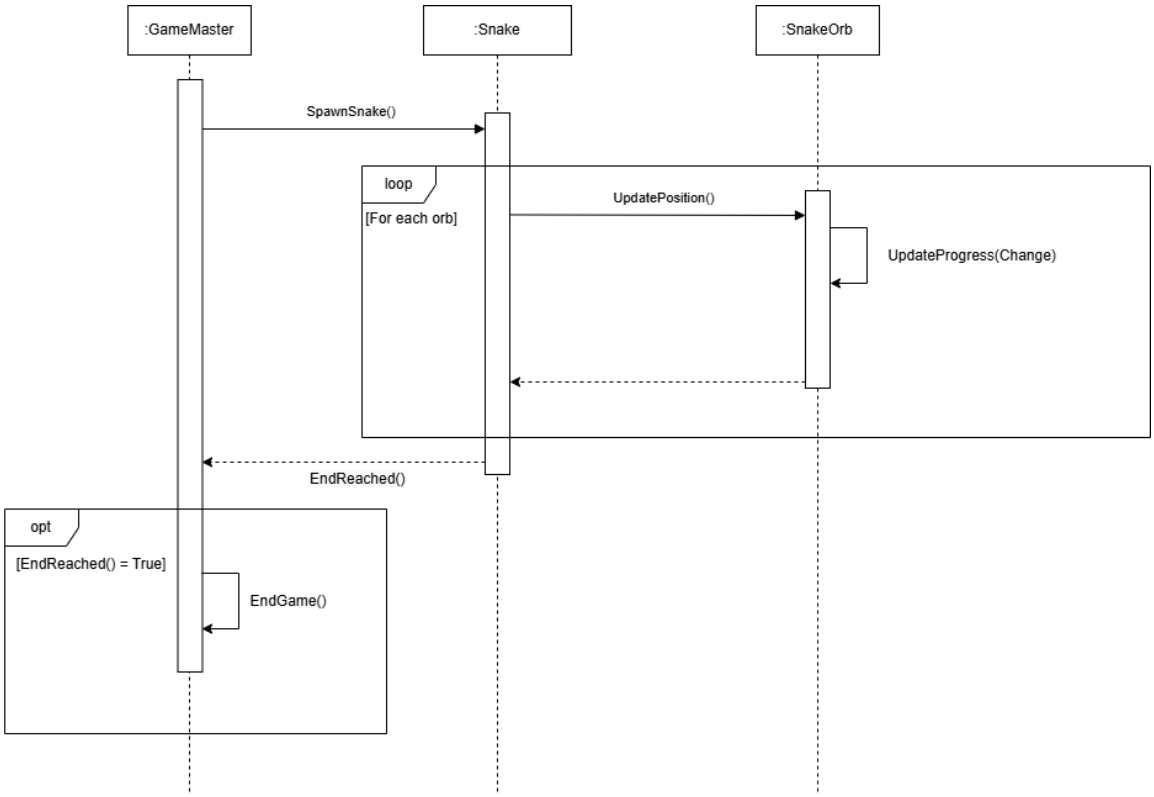
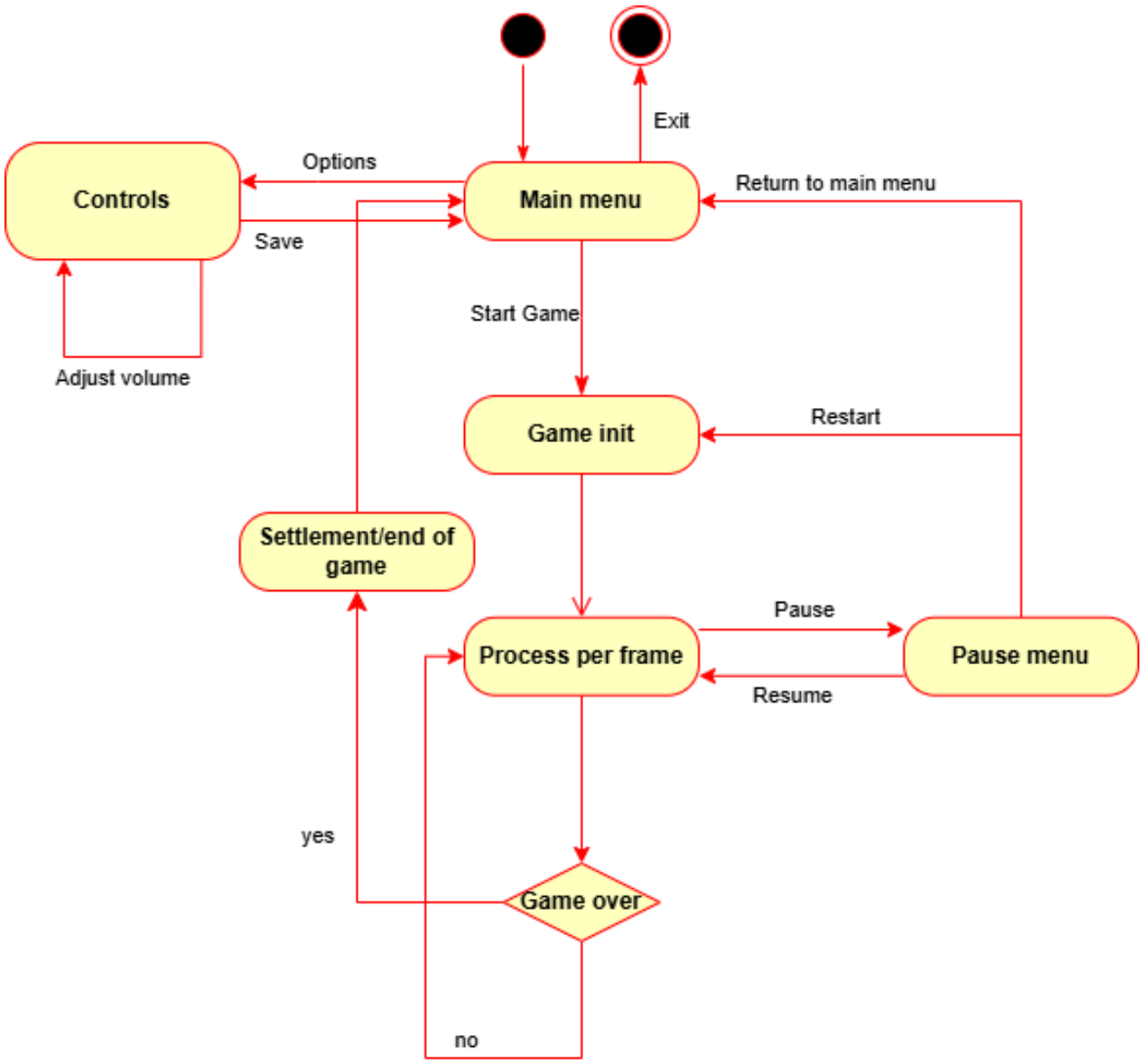


Figure 4.4: Sequence Diagram II - GameMaster and Snake Movement

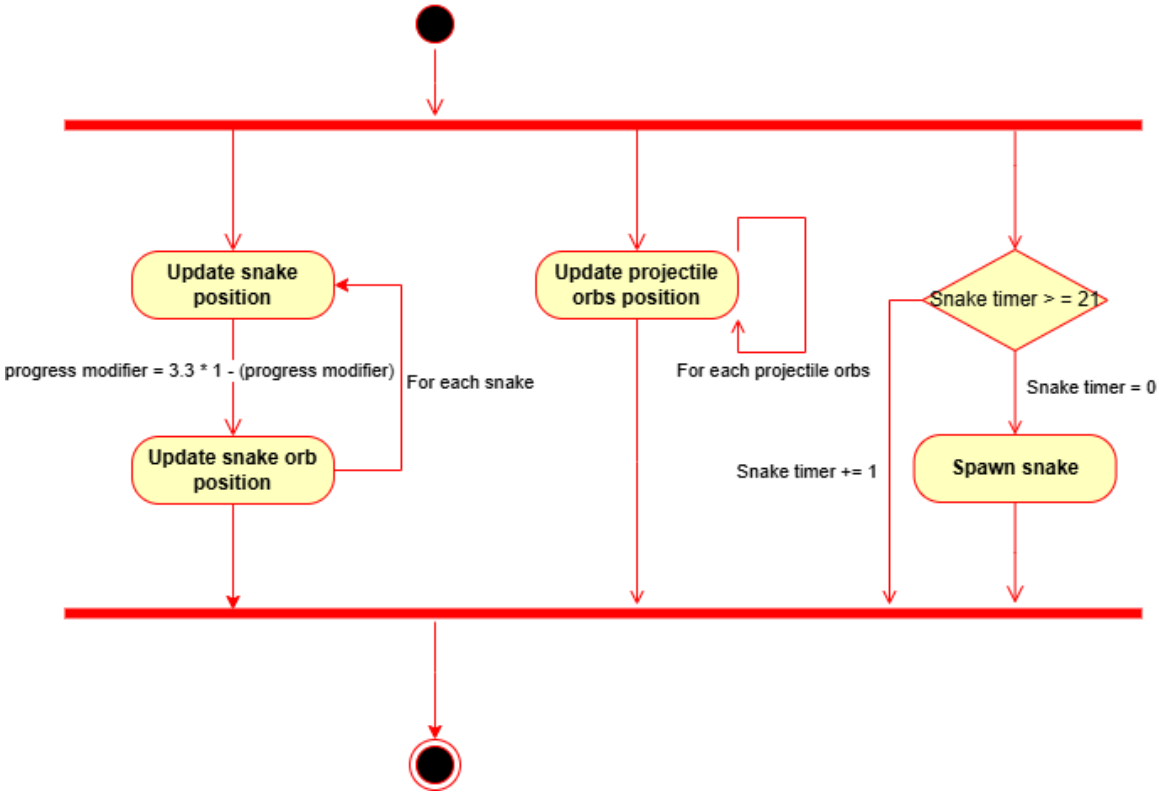
The second sequence diagram depicts the class interactions managed by the GameMaster, and uses the same general structure and components as the first sequence diagram. The GameMaster can create a new Snake, which then populates itself with SnakeOrbs. It also watches Snakes to see when they reach the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

4.4 State Diagram

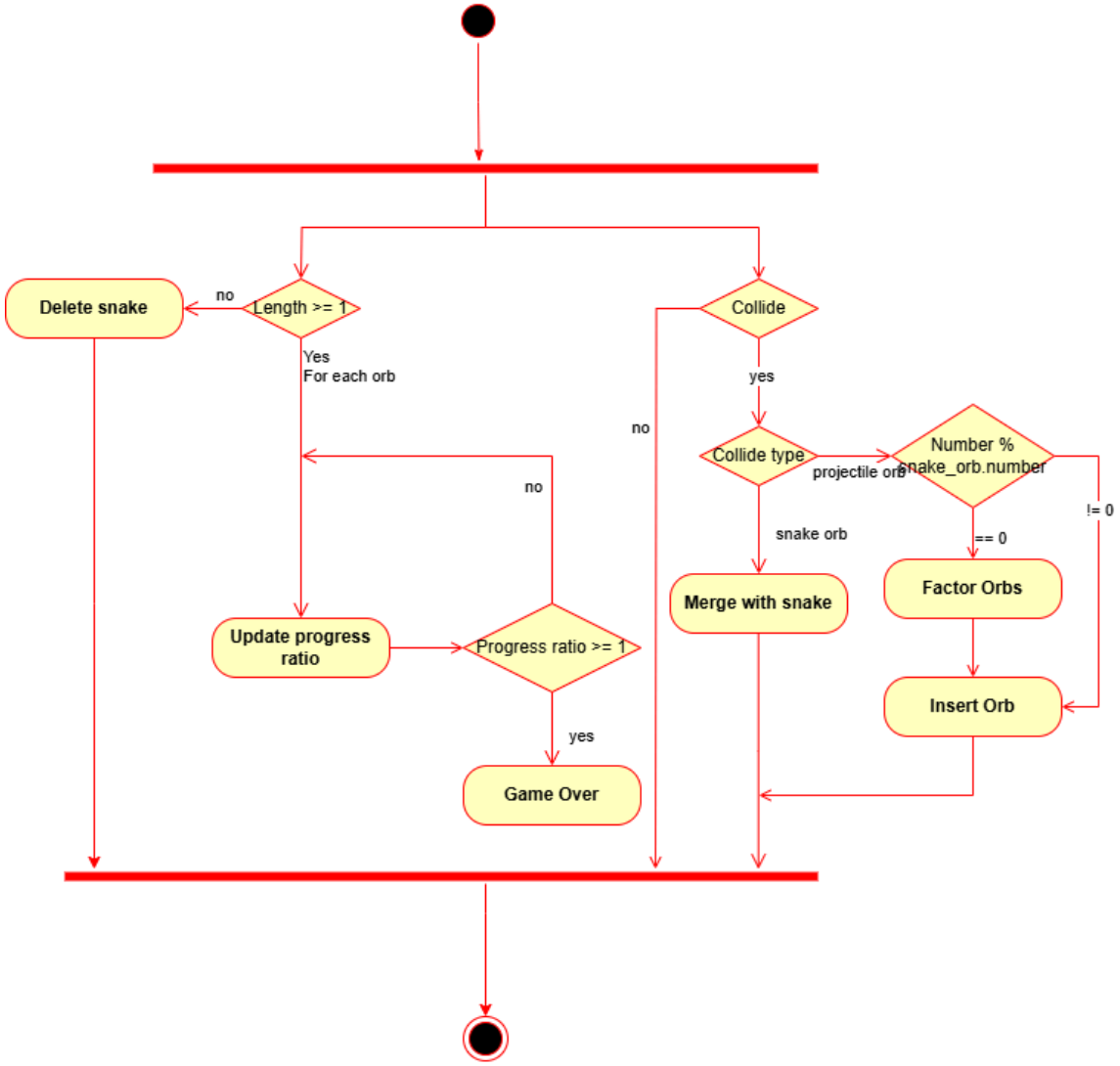
4.4.1 Main



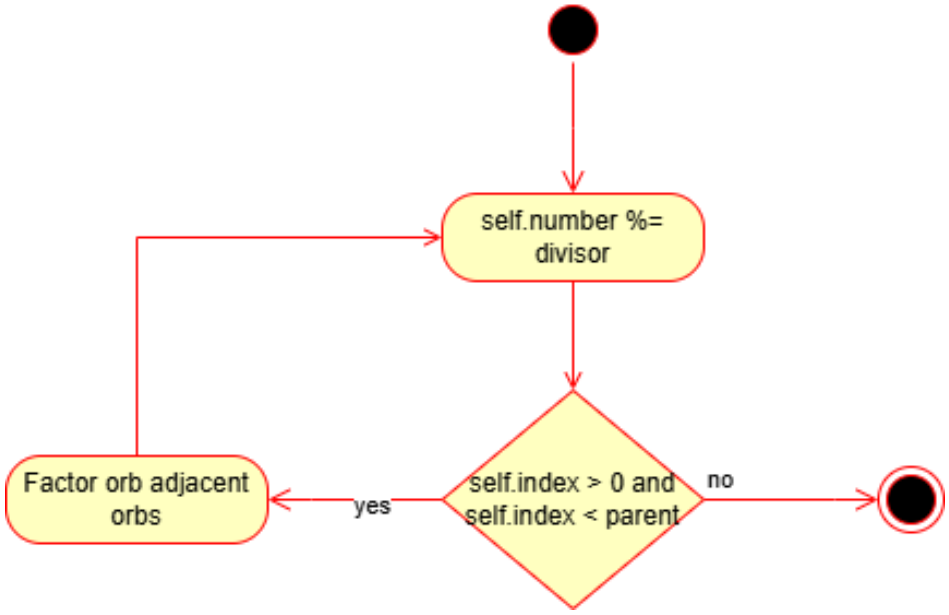
4.4.2 Process per frame



4.4.3 Update snake orb Position



4.4.4 Factor orbs (group)



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

FACTORBS is an arcade game that tests the user's ability to recognize a number's factors, the numbers that divide it exactly with no remainder. Snakes of orbs will approach on a set path, and the player must stop them from reaching the end by clearing the orbs. They do this by firing orbs from a cannon, which can be controlled using the mouse. When a fired orb hits a snake orb that it is a factor of, the snake orb's value is divided by the number of the fired orb. The player must be careful as hitting the wrong orb causes the fired orb to be inserted into the snake, making it longer. The more orbs the player can clear, the higher their score at the end of the game.

5.1 How to Run Prototype

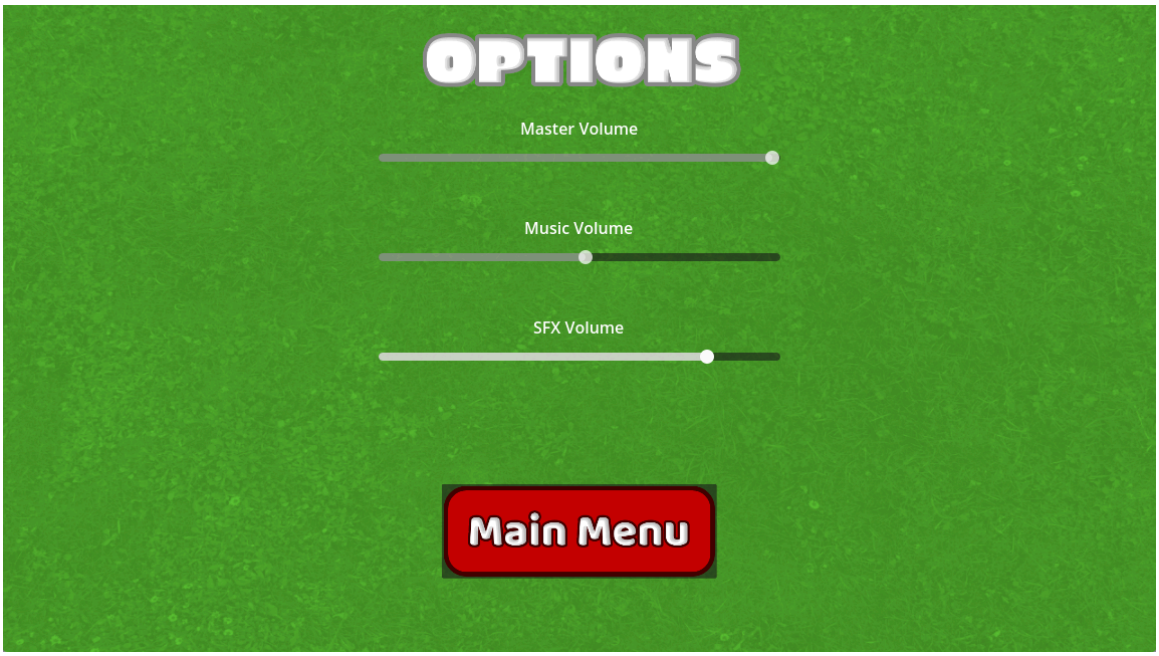
There are two ways to run the prototype. The first is by playing the embedded browser version that will be on the team's website using Godot's HTML 5 web build feature. The game can be found at the team's website under the "How to Play" tab: https://juruiz031.github.io/swe1/index.html

The second is by downloading the game on the same website, also under the "How to Play" section. This will download an executable that can be run and played by the user. There are three options for the download: Windows, MacOS, and Linux. https://juruiz031.github.io/swe1/index.html

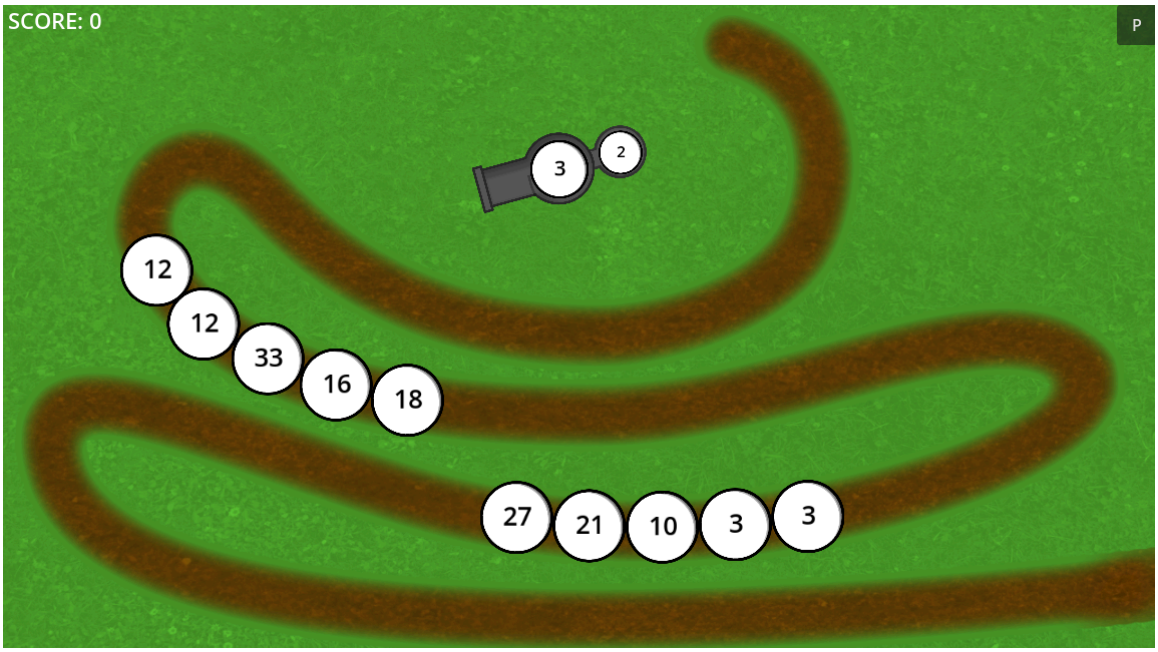
5.2 Sample Scenarios

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.

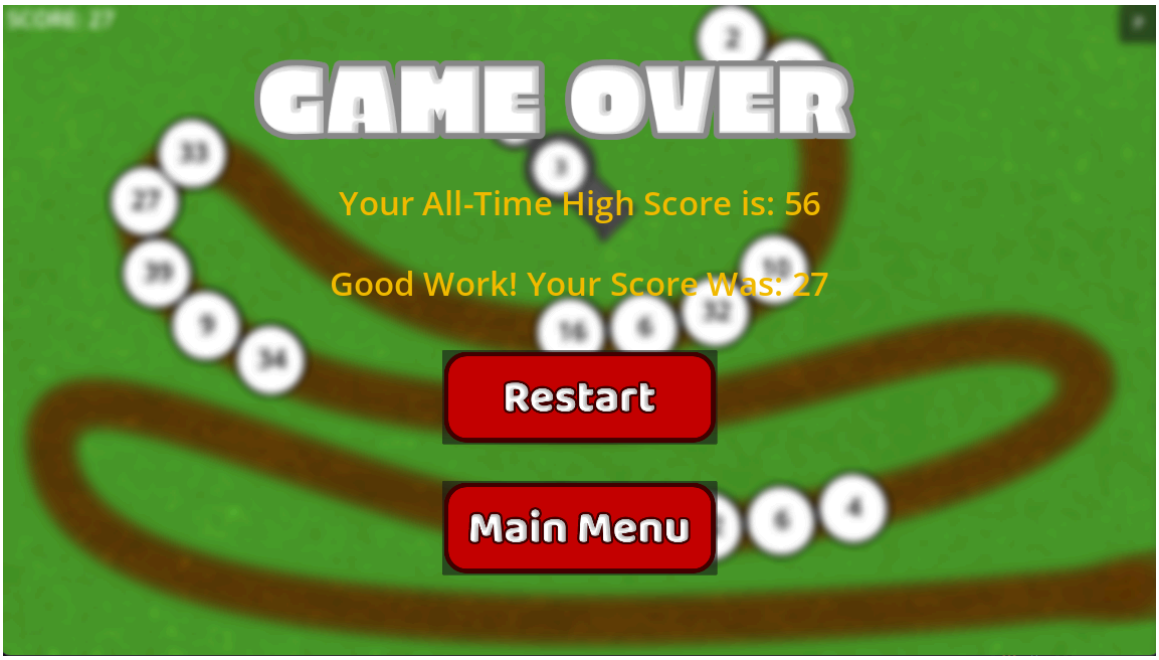


While the game is running, the player can control the cannon's direction by moving the mouse, and fire it by clicking the left mouse button.



The player can access the pause menu by clicking the "escape" key or pressing the "P" button on the top-right. From here, the player can resume the game, restart the game, or return to the main menu using the buttons shown.

When a snake reaches the end of the path, the game ends and a game over screen is displayed which shows the player's high score and the score for the current game session. From here, the player can either restart the game or return to the main menu.



6 References

[1] FACTORBS Website. Educational Game Prototype

[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

[4] Sparkle Unleashed. Sparkle Unleashed | 10tons

[5] Photopea. Photopea | Online Photo Editor

[6] Draw.io draw.io

[7] Godot Engine. https://godotengine.org/

[8] Audio. https://pixabay.com/ All audio fully aligns with Pixabay’s license, seen here: https://pixabay.com/service/license-summary/

a. FireCannon: https://pixabay.com/sound-effects/thud2-47105/

b. Button Click: https://pixabay.com/sound-effects/button-202966/

c. Clearing Orbs: https://pixabay.com/sound-effects/ding-36029/

d. Game Over: https://pixabay.com/sound-effects/game-over-39-199830/

e. Successful Factoring: https://pixabay.com/sound-effects/ding-101377/

f. Wrong Buzz: https://pixabay.com/sound-effects/wronganswer-37702/

g. Background Music:

https://pixabay.com/music/video-games-puzzle-game-bright-casual-video-game-music-249202/

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.