Software Requirements Specification (SRS)
Mini-Golf Mania

Team: Team 5
Authors: Austin Nguyen, Braden Maillet, Edward Alderman, and Christopher Lambert
Customer: Grades 6-8 Instructors
Instructor: Dr. Daly

1 Introduction

This Software Requirements Specification (SRS) document provides a comprehensive overview of the system requirements and design for **Mini-Golf Mania**, an educational game that reinforces algebra concepts in a fun and interactive way. The document is structured in a hierarchical approach, starting with a high-level overview and gradually diving into detailed technical aspects. It aims to guarantee clarity and alignment with the project's goals while maintaining logical progression. Topics covered in this document include the product's purpose, functional and non-functional requirements, system constraints, and user characteristics. This document also describes the modeling requirements, constraints, and a prototype that will help stakeholders visualize the game.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the core features, functionality, and constraints of the Mini-Golf Mania game. It provides a clear and detailed description of the system's objectives to make sure that all collaborators have a shared understanding of what the game will achieve and how it will function. This document serves as a foundation for the design, development, and testing phases of the game while guiding the project team through the implementation and validation processes. The intended audience for this document includes developers, project managers, clients who commission the product, customers who purchase the product, and end users. Developers will use this document to guide the game's design and implementation, while project managers will rely on it to ensure the project is completed in full and within scope. Clients will use the document to verify that the game meets their expectations, customers will refer to it to understand the features they are purchasing, and end users will interact with the features outlined in the document.

1.2 Scope

The software product to be produced is an edutainment game developed using Unity and WebGL. The game is designed to combine the engaging mechanics of mini golf with educational content focused on algebra concepts, specifically linear equations. The application domain of this software is the edutainment sector, focusing on both educational and entertainment value. The main objective of the game is to reinforce algebraic concepts through practical and hands on problem solving within the context of a game. It will allow players to interact with algebraic expressions and observe the impact of their modifications through the golf ball's trajectory effectively visualizing the relationship between equations and graphical representations like a graphing calculator.

The software will focus on gameplay where players will enter algebraic expressions to determine the trajectory of a golf ball. Each level will present unique challenges designed to reinforce algebraic concepts. The game will include a user interface with input fields for algebraic expressions and a slider to control the ball's trajectory, along with a level progression system that tracks the player's progress and attempts.

On the contrary, the software will not focus on advanced algebraic concepts beyond linear equations, nor will it feature multiplayer functionality or in-depth lessons. The game will be designed for individual learning.

1.3 Definitions, acronyms, and abbreviations

UML: Unified Modeling Language

Input field: Area in which user can input their answer

MMA: Mini-Golf Mania. The name of the edutainment game

SRS: Software Requirements Specification

Trajectory: The path that the golf ball follows being hit. In the game, the trajectory is determined by a linear algebraic expression entered by the player.

Linear Equation: A first-degree polynomial equation that describes a straight line.

UI (User Interface): The part of the software that allows users to interact with the game, including buttons, sliders, and other interactive elements.

Unity: A cross-platform game engine used for creating 2D and 3D games.

WebGL: A web-based graphics library used for rendering 3D and 2D graphics in a web browser.

1.4 Organization

This SRS document is structured to provide a clear and detailed description of the requirements for **Mini-Golf Maina**. The document is organized into sections that progressively describe the system's purpose, scope, functionality, and other critical aspects.

The rest of the document is organized as follows:

- **Section 2: Overall Description** – This section provides an overview of the product, its context, major functions, user characteristics, and any constraints or assumptions related to the system. It also covers the system's product perspective, including interfaces with other systems and hardware.
- **Section 3: Specific Requirements** – This section enumerates the detailed functional and non-functional requirements for the game, organized in a hierarchical manner. Each requirement will be described and referenced with corresponding use cases and other specifications.
- **Section 4: Modeling Requirements** – This section details the modeling of system components, including diagrams such as use case, class, sequence, and state diagrams, to provide a graphical representation of the system's structure and behavior.

- **Section 5: Prototype** – This section discusses the prototype of the game, including the system functionality, scenarios, and screenshots that demonstrate how the prototype will represent the final product.
- **Section 6: References** – This section lists all documents and resources referenced in the SRS, including any external sources for definitions, libraries, or standards used.
- **Section 7: Point of Contact** – This section provides the contact details of the person or team responsible for the project, allowing participants to reach out for further clarification or inquiries.

2 Overall Description

This section provides a comprehensive overview of the game's development and operational context. It describes how the game fits within its intended environment, including the interfaces it relies on and any constraints on its design and implementation. There are key features and functions outlined that focus on how the game will interact with users and provide educational value. Characteristics of the target audience are mentioned, along with assumptions about their environment and algebraic proficiency. This section also addresses the constraints impacting development, such as hardware requirements and legal considerations, and highlights potential features that could be included in future versions of this document.

2.1 Product Perspective

MMA is an interactive web-based application designed to teach algebra concepts via gameplay. The game will be developed using Unity and WebGL to ensure compatibility across various platforms, such as web browsers, desktop computers, and mobile devices. MMA will serve as a standalone product, focusing primarily on teaching algebra in a fun and interactive way. Interaction with the user's device will take place through a graphical user interface, where users input mathematical equations that affect the trajectory of the golf ball. The game will not require external hardware components, instead it will rely on modern web technologies for communication and rendering.

Interface constraints include the following:

- **System Interfaces**: MMA will interface with web browsers, supporting both desktop and mobile environments, with basic HTML5 and WebGL support required.
- **User Interfaces**: The primary user interface will consist of input fields for equations as well as visual representation of the game, so the user will need a screen, keyboard, and mouse or a mobile device capable of performing the same functions.
- **Hardware Interfaces**: MMA will require minimal hardware, working on any standard desktop or mobile device capable of running WebGL enabled browsers.
- **Software Interfaces**: MMA will rely on Unity for game mechanics and WebGL for rendering within the browser, so users will need a device capable of running any popular web browser.
- **Operations**: MMA will operate offline once loaded, requiring no constant internet connection otherwise the initial connection to download the contents of the game.

2.2 Product Functions

MMA will feature several key functions to achieve its goal of teaching algebra through interactive gameplay:

- **Equation Input**: Players can input algebraic expressions, in the form of linear equations, which determine the trajectory of the golf ball.
- **Graphical Representation**: MMA will visually represent the trajectory of the ball based on the player's input equation.
- **Physics Simulation**: The game will simulate realistic physics for the golf ball's movement, taking angles and the player's input equation into account.
- **Level Progression**: MMA will include multiple levels with increasing difficulty, requiring players to apply more complex algebra concepts as they advance.
- **Scoring System**: The game will track scores based on accuracy and time, rewarding players for completing levels quickly and correctly.
- **User Interface**: MMA will offer a clean, intuitive UI that facilitates easy interaction, allowing users to input equations and engage with the game.
- **Instructional Feedback**: The game will have the option to provide instructional feedback offering hints or guidance for solving algebraic problems when necessary.

2.3 User Characteristics

The intended users for MMA are students primarily in grades 7 and 8, who are learning Algebra I. These users are expected to have a foundational understanding of algebraic principles, including linear equations and expressions, which are typically introduced in 7th grade. Users should be comfortable working with linear equations in point-slope, slope-intercept, and standard form. By 8th grade, students are expected to have developed proficiency with linear relationships, equations and expressions as recommended by the Massachusetts Department of Education's advisory on the Mathematics Curriculum Framework. The game is designed to challenge and engage users who are proficient in algebraic reasoning therefore MMA targets higher-level middle school students who can apply mathematical concepts to solve interactive challenges.

2.4 Constraints

Developmental Constraints: The development of MMA is constrained by the limited time and resources available to the project. Developers must adhere to a strict schedule and agile method to deliver a functional prototype and subsequent versions. Also, the use of Unity as the development platform imposes a dependency on team members having familiarity with the engine and its scripting environment, potentially limiting design flexibility.

Legal Constraints: The game must comply with the educational clauses stated in Section 110 of U.S. copyright law, which are applicable to all resources compiled for the game. This includes ensuring that third-party assets such as texture packs are used in accordance with their licensing terms and conditions.

Hardware Limitations: The game targets WebGL as its deployment platform, which necessitates compatibility with modern web browsers and operating systems. Devices with outdated hardware or limited graphics capabilities may not perform optimally, posing a limitation in terms of accessibility for all potential users.

Other Considerations: While safety and security are not primary concerns for this game, the project must ensure that any hosted web-based deployment adheres to basic web security standards, such as securing the hosting platform against vulnerabilities.

2.5 Assumptions and Dependencies

Assumptions:

The game assumes that end users have access to devices with standard hardware capabilities, including a CPU and graphics card sufficient for running modern web browsers. Users will use browsers such as Chrome, Firefox, Safari, or Edge, all of which must support WebGL for rendering. Users should be able to interact with the game using a mouse or touchscreen, depending on the platform. The game also assumes users have a stable internet connection to access the game through a web browser. Lastly, it assumes that the user has a good foundation in algebraic concepts, particularly linear equations, as the game requires these skills to solve in game challenges related to trajectory and ball movement.

Dependencies:

Key dependencies include WebGL support in the user's browser engine to render the game's graphics and the developer's device compatibility with Unity WebGL to build and test the game's browser capabilities.

2.6 Apportioning of Requirements

The current prototype focuses on core functionality essential for demonstrating the basic concept of the game. This includes a main menu with operational "Start" and "Quit" buttons, functional input fields for equations, a "Shoot" button, level progression, and a basic win condition. Upon winning, a window prompts the player to either continue or quit the game, ensuring a functional game flow.

Functionality that will be implemented later in development include a ball sprite for visual representation, basic physics found in mini-golf, an equation validation system to ensure input accuracy, a trajectory calculator to compute the ball's path based on user input, a built-in coordinate system for mechanics, and problem sets for each level.

3 Specific Requirements

1. Each level will take the form of an overhead view of a mini-golf course.
    - 1.1. The view of the mini-golf course will be overlaid with a labelled grid representing the first quadrant of the cartesian plane.
    - 1.2. The course will feature a “tee” as a starting point for the ball.
    - 1.3. The course will feature a “hole” for the destination of the ball.
    - 1.4. The course will feature zero or more obstacles between the starting point and tee and the hole.
2. Players will decide the trajectory of a golf ball by entering or augmenting an expression.
    - 2.1. Once an expression has been entered, an overlay of the projected trajectory of the ball will be rendered over the golf course. This overlay will update whenever the expression is altered.
    - 2.2. The ideal trajectories for each level will be designed to reinforce specific concepts from Algebra 1.
3. Once Players have decided a trajectory, they will decide the distance the ball travels along that trajectory by moving a slider.
    - 3.1. As the slider goes up, a putter will be shown moving back slightly.
4. Upon releasing the slider and pressing the shoot button, the ball will move along the trajectory to a final position determined by the distance slider.
    - 4.1. If the ball’s final position is within a specified distance from the hole, it will enter the hole.
    - 4.2. If the ball enters the hole, a celebratory message will be displayed, and a “continue” button appears to take the player to the next level.
        - 4.2.1. On the final level, the “continue” button will instead return the player to the main menu.
    - 4.3. If the ball hits a wall or obstacle, then the ball will bounce with respect to the obstacle.
    - 4.4. If the ball misses the hole, the ball will be returned to its starting position, and the number of attempts for the level will be incremented for the player.
5. The game will display gameplay options and statistics on a user interface, which surrounds the display of the golf course.

5.1. The input field for the player to enter their trajectory equation and the slider to select the distance the shot will travel will be displayed in this user interface.

5.2. The user interface will also host a button to exit the current game.

6. The game will display a main menu upon launch.

6.1. The main menu will contain a quit button to exit the program.

6.2. The main menu will contain a start button, which will bring the player to the first level of a new game.

6.3. The main menu will contain a High Scores button, which will bring the player to a page displaying the local record score for each level.

7. The game will contain a minimum of three levels.

7.1. Each level will focus on a different form of linear equation. (point-slope, slope-intercept, standard)

4 Modeling Requirements

This section defines how the system's requirements are translated into models that connect user needs with system functionality. It includes descriptions of use cases that comply with requirements, a class diagram outlining the system's core components, and a data dictionary explaining attributes and relationships. There are also state diagrams depicting the behavior of key class instances.

4.1Use Case Diagram

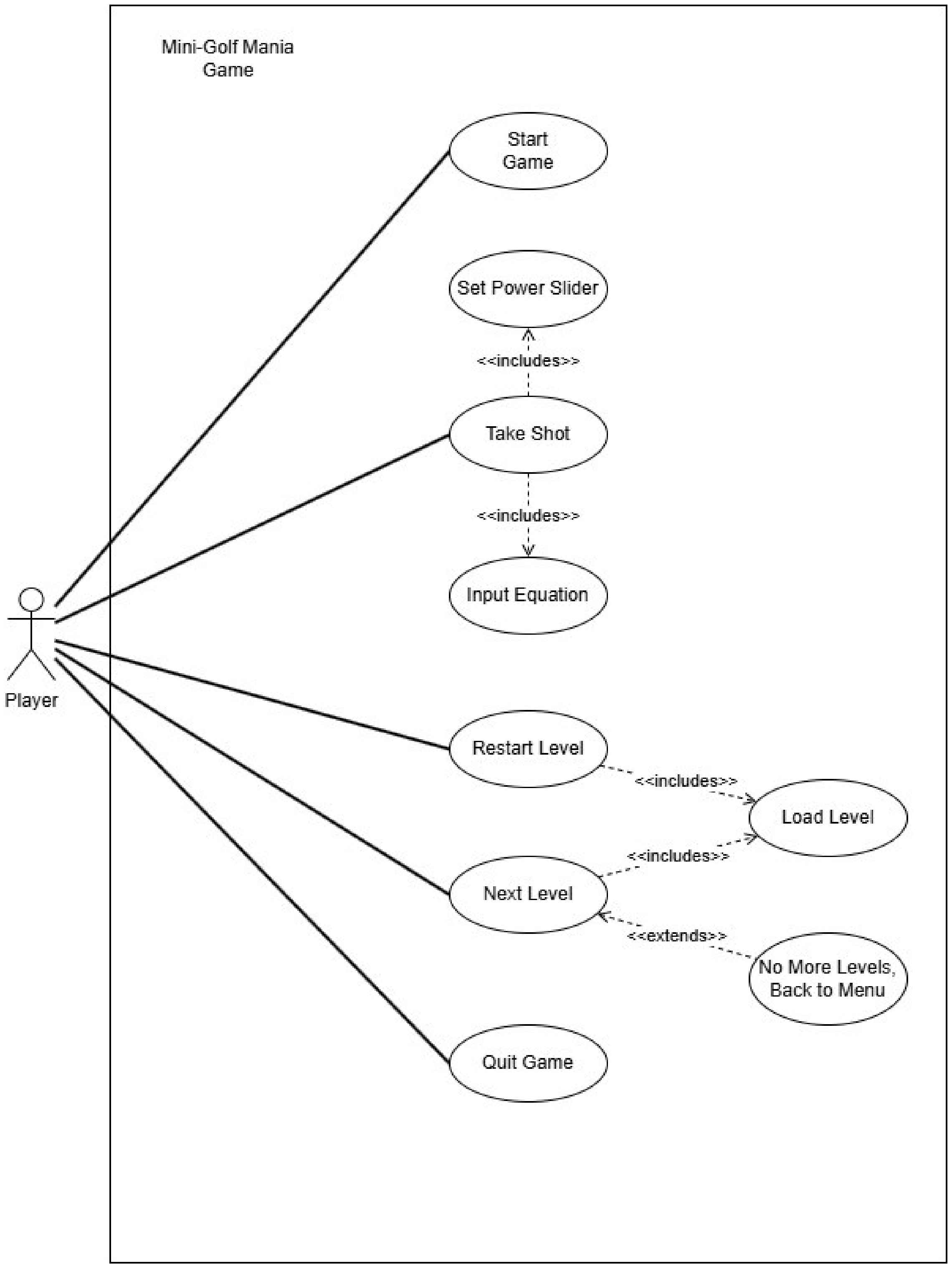

Figure 4.1.1 Use Case Diagram

Use Case Name:	Start Game
Actors:	Player
Description:	The button to click to start to game.
Type:	Primary and essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 6.1
Uses cases:	N/A

Use Case Name:	Take Shot
Actors:	Player
Description:	The player has input the parameters and clicks the shot button.
Type:	Primary and essential
Includes:	Set Power Slider, Input Equation
Extends:	N/A
Cross-refs:	Requirement 4
Uses cases:	

Use Case Name:	Restart Level
Actors:	Player
Description:	The button to click to restart a level.
Type:	Primary and essential
Includes:	Load level
Extends:	N/A
Cross-refs:	Requirement 4.3
Uses cases:	N/A

Use Case Name:	Next Level
Actors:	Player
Description:	The button to click go to the next level
Type:	Primary and essential
Includes:	Load Level
Extends:	N/A
Cross-refs:	Requirement 4.2
Uses cases:	N/A

Use Case Name:	Quit Game
Actors:	Player
Description:	The button to click to quit the game
Type:	Primary and essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 6.2
Uses cases:	N/A

Use Case Name:	Load Level
Actors:	Player
Description:	The function that is called to render the games levels when the player progresses
Type:	secondary and essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 4.2 & 6.2
Uses cases:	Next Level, Restart Level

Use Case Name:	Set Power Slider
Actors:	Player
Description:	The parameter the player will set to determine how far the golf ball will roll.
Type:	secondary and essential
Includes:	N/A
Extends:	N/A
Cross-refs:	***
Uses cases:	Take Shot

4.2 Class Diagram

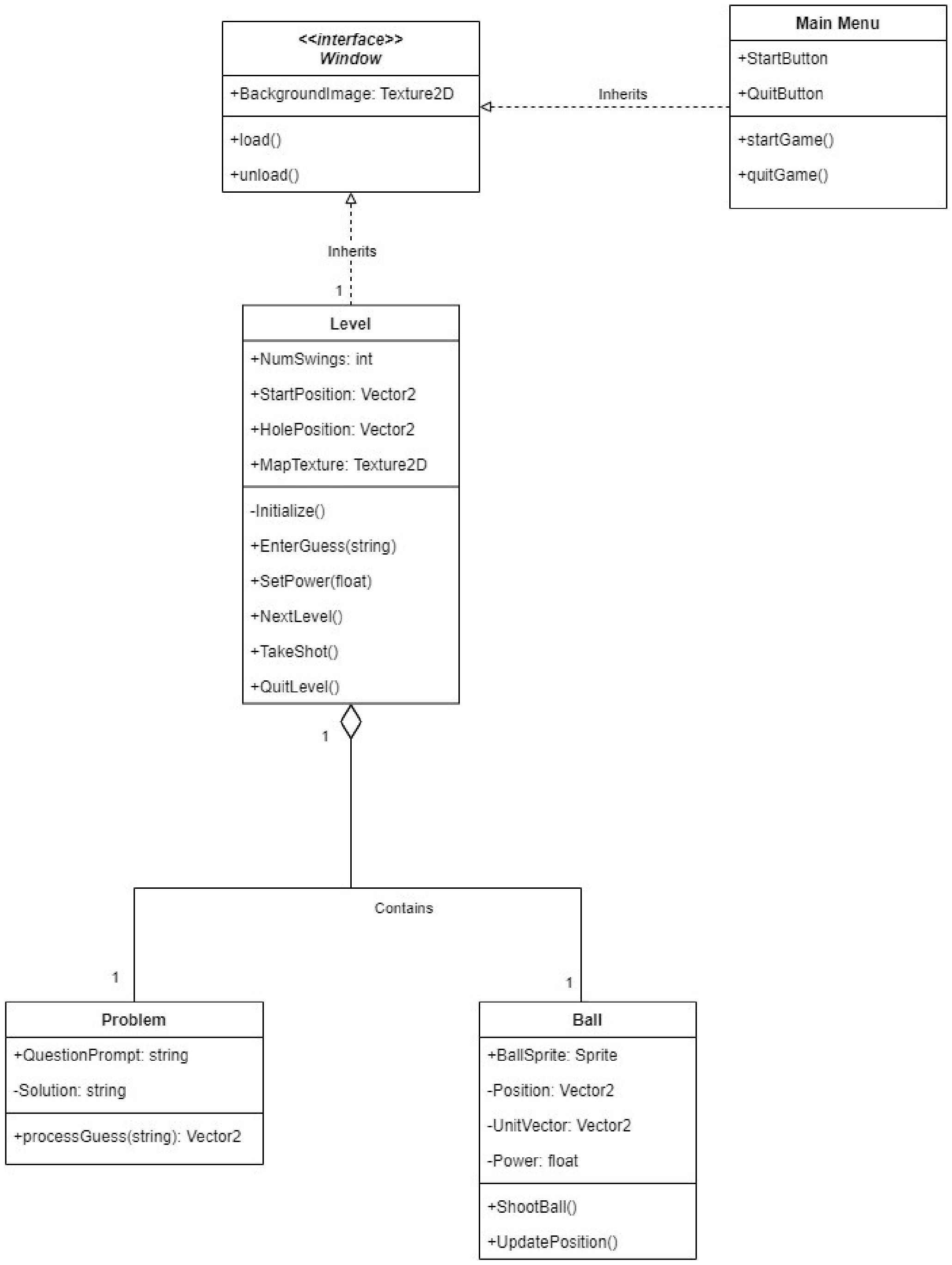

Figure 4.2.1 Class Diagram

Element Name		Description
Ball		Manages the depiction, and movement of the ball.
Attributes		
	BallSprite: Sprite	A sprite depicting the ball.
	Position: Vector2	The current position of the ball.
	UnitVector: Vector2	The unit vector derived from processing the player’s guess in the form of a Vector2. This is multiplied by Power to determine the velocity of the ball.
	Power: float	Multiplies UnitVector to determine the ball’s velocity.
Operations		
	ShootBall (Vector2, float): void	Accepts the X and Y components of the ball’s velocity as returned by Problem’s ProcessGuess operation, and a the power to scale the ball's velocity by as a float returned by Level’s SetPower operation.
	UpdatePosition(): void	Called once per frame while the ball is in motion to update the position of the ball.
Relationships	Level contains one Ball.	
UML Extensions	N/A	

Element Name		Description
Level		Manages the game's individual stages, layout, objectives, and progression criteria.
Attributes		

	NumSwings: int	The number of times the player has attempted the level without winning.
	StartPosition: Vector2	The starting position for the ball on this level.
	HolePosition: Vector2	The position of the hole for this level.
	MapTexture: Texture2D	The texture for the map of the level.
Operations		
	Initialize (): void	Loads the texture for the level as the BackgroundImage. Positions the ball at its starting point. Loads the level's question into the UI.
	EnterGuess (string): void	Accepts the string entered by the player as the solution to the problem.
	SetPower(int): void	Sets the power for the shot to be taken.
	NextLevel(): void	Loads next level if there is one. If not, returns to Main Menu.
	TakeShot(): void	Triggers the ShootBall operation in Ball.
	QuitLevel(): void	Returns the player to the Main Menu.
Relationships	Level inherits from the Window interface. Each Level contains one Problem and one Ball.	
UML Extensions	N/A	

Element Name		**Description**
Main Menu		Allows a player to start a new game or close the window.
Attributes		

	StartButton: Button	A button which triggers the startGame operation.
	StartButton: Button	A button which triggers the quitGame operation.
Operations		
	StartGame (): void	Loads first level.
	QuitGame (): void	Closes the game window.
Relationships	The MainMenu inherits from the Window interface.	
UML Extensions	N/A	

Element Name		**Description**
Problem		This class contains the problem to be presented to the player for the level.
Attributes		
	QuestionPrompt: string	The text for the question to be presented to the player for the current level.
	Solution: string	The correct answer to the problem.

Operations		
	ProcessGuess(string): Vector2	This accepts the player’s guess at the solution for the problem as an argument, and returns a Vector2, representing the X and Y components of the ball’s velocity for the entered equation, assuming a total velocity of 1.
Relationships	Each Level contains one Problem.	
UML Extensions	N/A	

4.3 Sequence Diagrams

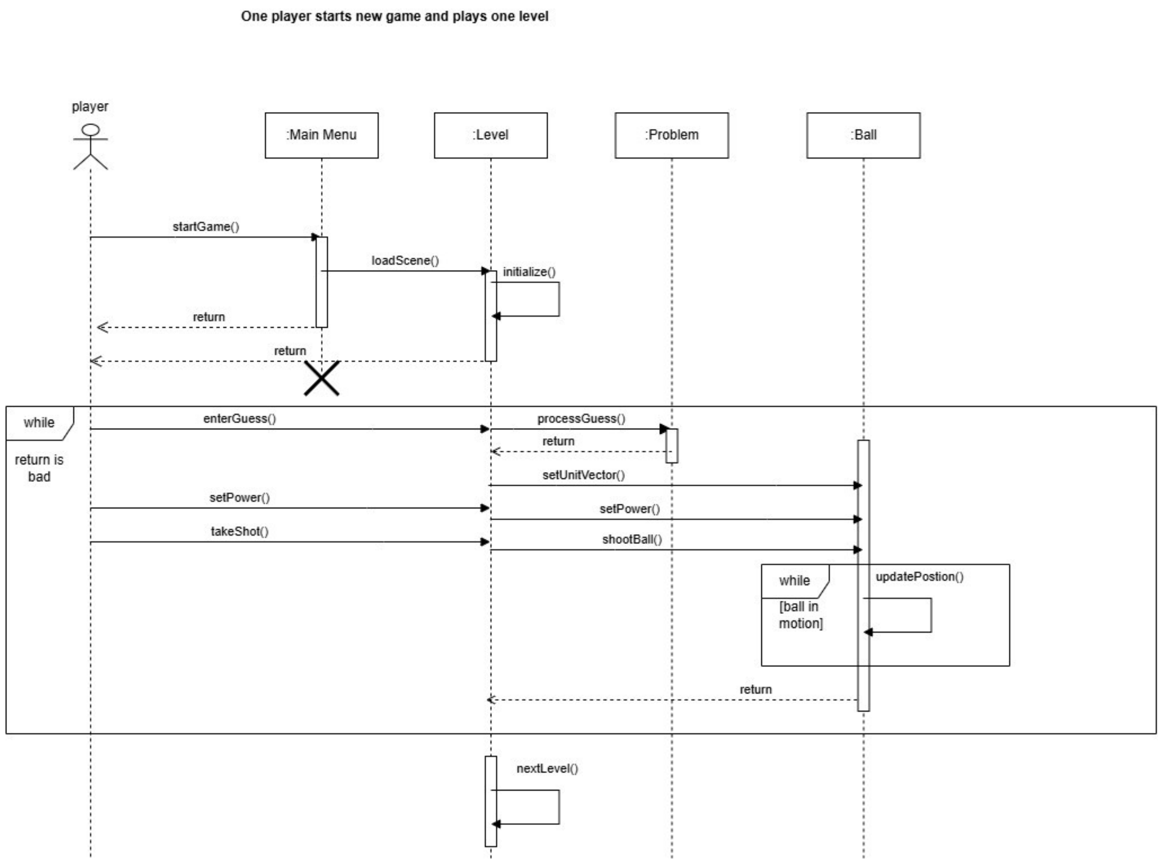

Figure 4.3.1 Sequence Diagram 1: Player Plays One Level

This sequence diagram demonstrates the gameplay process starting from the main menu to the first level of the game. The player initiates the game by selecting "Start Game," triggering the startGame() function in the Main Menu, which then calls loadScene() to initialize the first level. The player interacts with the game by entering a guess, setting the power, and shooting the ball, with feedback loops for invalid returns until successful progression. Once completed, the nextLevel() function advances the game to the next stage.

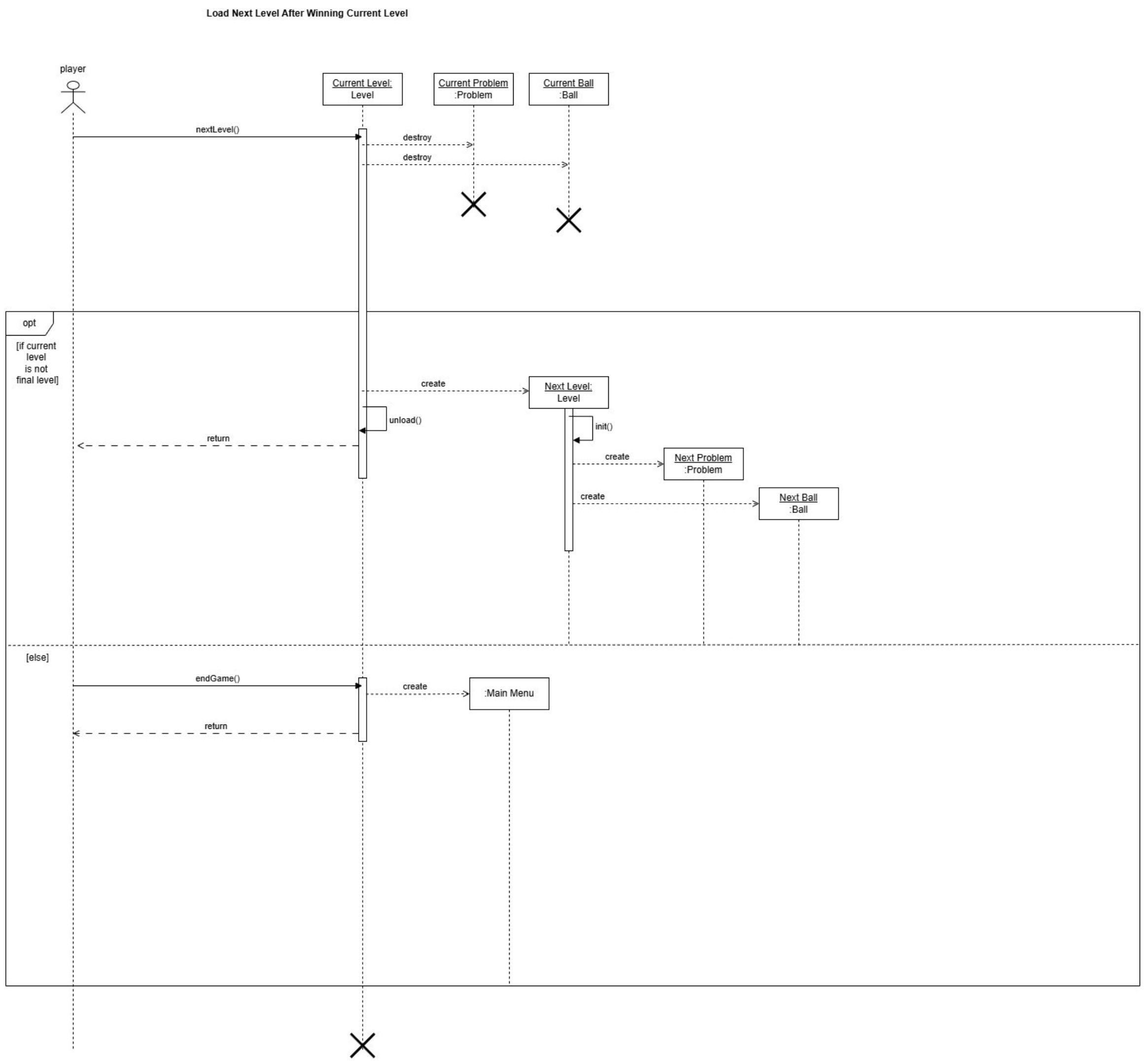

Figure 4.3.2 Sequence Diagram 2: Game Session

This sequence diagram outlines the game's handling of level progression. Upon winning the current level, the nextLevel() function is invoked, destroying the objects associated with the current level, problem, and ball. If the current level is not the final level, new objects for the next level, problem, and ball are created and initialized. Otherwise, the game ends with the endGame() function, returning the player to the main menu.

4.4 State Diagrams

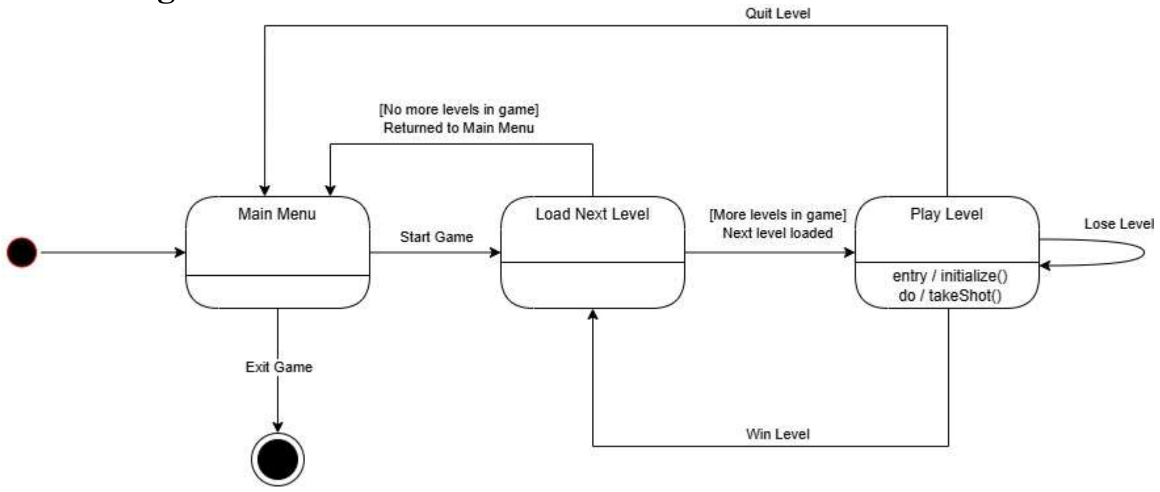

Figure 4.4.1 State Diagram 1: Full Game State Diagram

This state diagram captures the states and transitions of the game as a whole at a high level. Upon launching the game, the Main Menu is displayed. If the player selects Exit Game, then the game exits. This is considered its final state. If the player instead selects Start Game, then the game transitions into the Loading Next Level. If there is a level to load it is loaded, the game will transition into the Play Level state. Otherwise, if there are no levels left to load, the game transitions back to the Main Menu state. Upon entering the Play Level state, the level is initialized. While in the Play Level state, the player will take shots at the hole. If the player misses the hole, they lose the level and the game transitions back to the Play Level state, initializing the level and allowing the player to attempt the level again. If the player wins the level, the game transitions to the Load Next Level state again. If the player chooses to quit the level, the game transitions back to the Main Menu state.

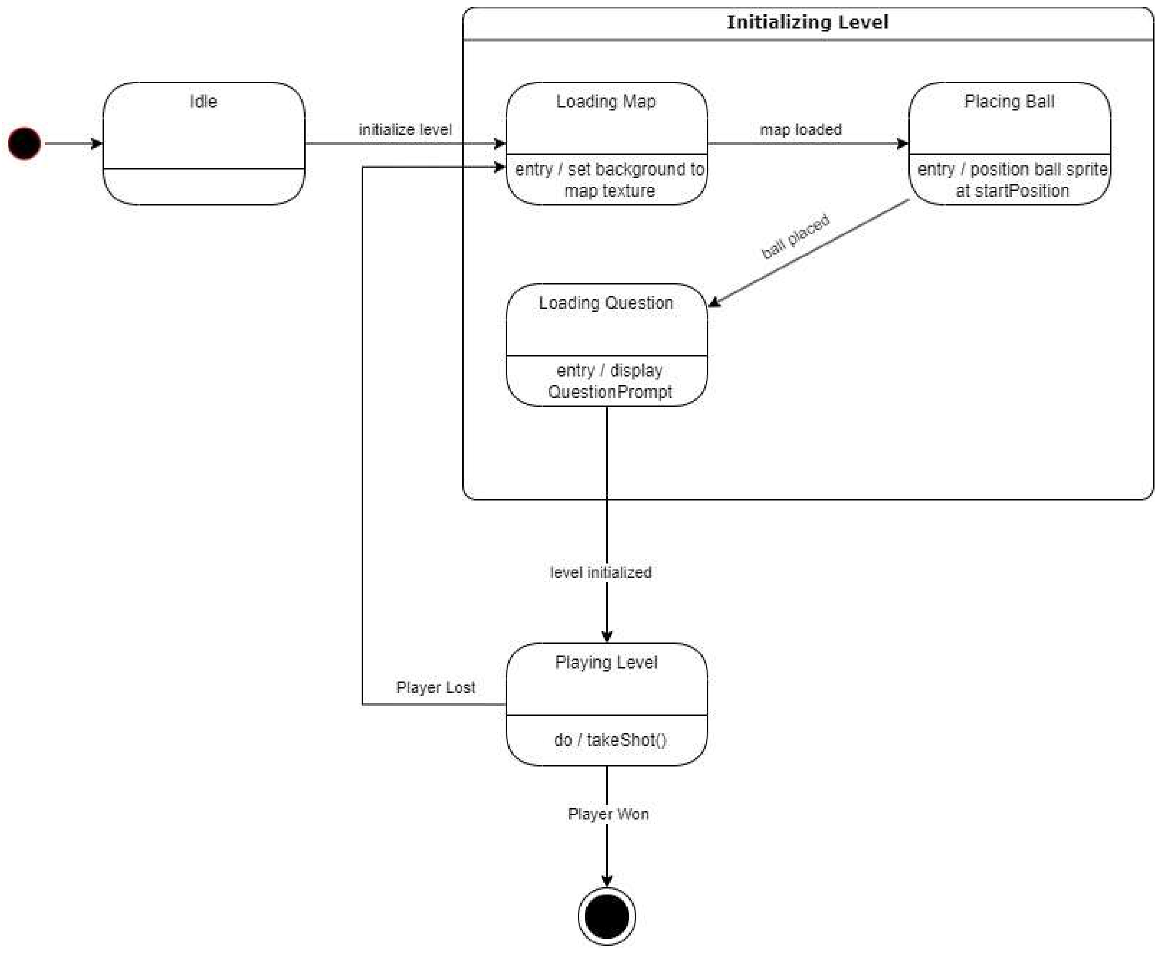

Figure 4.4.2 State Diagram 2: Level State Diagram

This state diagram outlines the states of the Level class throughout the central gameplay loop. The Level starts idle, and transitions into the Initializing Level composite state once initialize() is called. Once the map is loaded, it transitions into the Placing Ball state, in which it sets the position of the ball and loads its sprite. Once the ball is placed, the Level transitions into the Loading Question state, in which the question prompt for the level is loaded and rendered onto the user interface. Once that has finished, initialization is complete, and the Level transitions into the Playing Level state, in which the player takes shots at the hole. If the player wins the level, the Level transitions into its final state. If the player loses, the Level transitions back into the Initialing Level composite state, beginning with Loading Map.

5 Prototype

The current prototype consists of a main menu and three holes. The prototype does not contain any game functionality and consists only of user-interface elements. The main menu allows you to start or quit the game. The subsequent holes give you areas for answer input, a prompt box as well as a button to “shoot” the ball. In this prototype the answer is always correct and once the shoot button is pressed a win screen is displayed that allows the user to quit or continue.

5.1 How to Run Prototype

To run the MMA prototype, the following steps are required:

1. **Setup Unity:**
    - Download and install **Unity Hub** from Unity's official website.
    - Once Unity Hub is installed, use it to download and install the **Unity Editor**. Ensure that you are using a version of Unity compatible with the project.
2. **Clone the GitHub Repository:**
    - Clone the project repository from GitHub using the following link: https://github.com/chrislambert3/EduGame.git.You can use GitHub Desktop or any Git client for this purpose.
3. **Import Project into Unity Hub:**
    - After cloning the repository, open Unity Hub.
    - Click on **Add Project from Disk** and navigate to the location where the repository has been cloned.
    - Select the project folder and add it to Unity Hub.
4. **Open the Project:**
    - Once the project is added to Unity Hub, click on the project to open it in the Unity Editor.
5. **Configure Game Window:**
    - In the Unity Editor, navigate to the **Game** window and set the resolution from **Free Aspect** to **Full HD** (1920x1080).
    - Scale the view back out to ensure the entire game screen is visible.
6. **Running the Game:**
    - Press the **Play** button located at the top center of the Unity Editor window to run the game. The prototype will start running, and you can interact with the game as intended.

System Configurations:

- **Operating System:** The prototype is compatible with **Windows, macOS**, and **Linux**. Ensure your operating system meets the minimum requirements for running Unity projects.
- **Hardware Requirements:** A PC or laptop with at least 8GB of RAM, a modern multi-core CPU, and a GPU with OpenGL 3.0 or higher is recommended for optimal performance.
- **Web Accessibility (for Prototype V2):**
    - Prototype V2 will be hosted as a WebGL build and should be accessible via a web browser (e.g., Chrome, Firefox).
    - Ensure your internet connection is stable when accessing the prototype online.

- **Plugins and Dependencies:**
    - Ensure you have the required Unity plugins and packages installed. Any necessary packages will be included in the Unity project and should automatically be configured upon opening the project.

5.2 Sample Scenarios

When a player starts the game, they will be prompted with the main menu. From this screen the user will have the ability to start or quit the game.

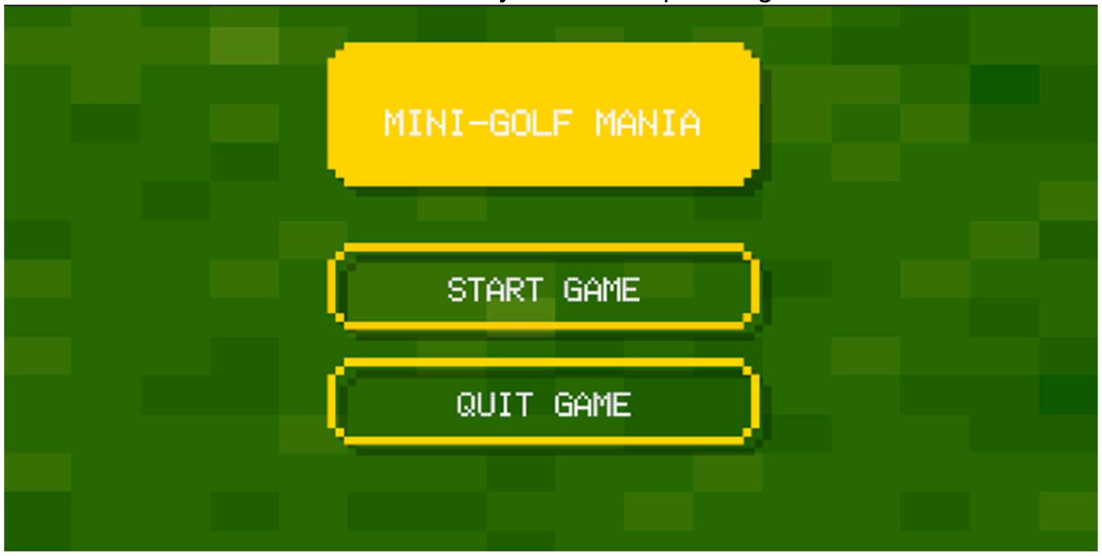

Figure 5.2.1: Main Menu Screen

If the player chooses to start the game, they will be brought to the first hole of the course (Figure 5.2.2). Here they will be prompted with a question as you can see in the bottom left. They will be able to input an answer within the "input" box and power slider. Once the player is confident with their answer, they will click the shoot button to enter their answer.

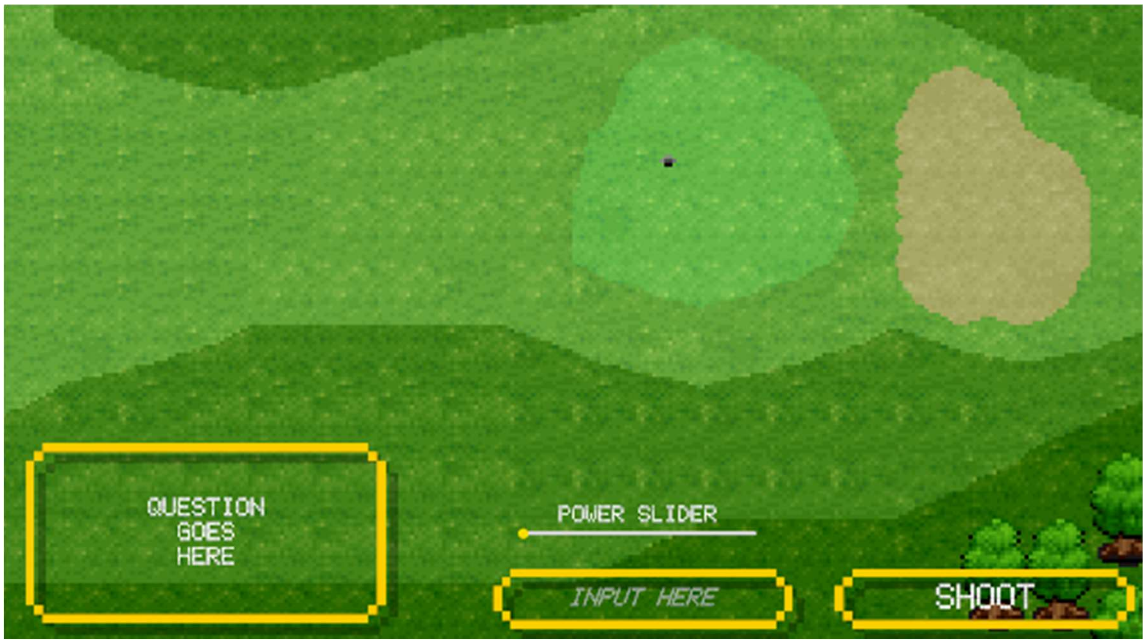

Figure 5.2.2: Hole 1

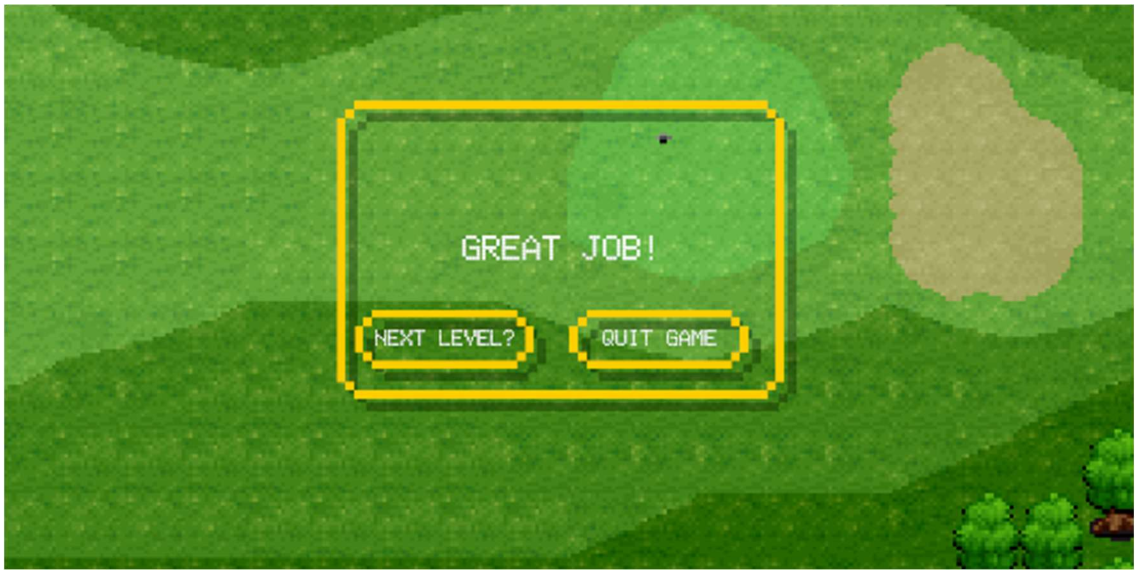

Figure 5.2.3: Hole 1 Win Screen

In the case that the players answer is correct, they will be prompted with this screen. Here they will be able to continue to the next hole or quit the game.

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] U. S. C. Office, “Chapter 11: Subject matter and scope of copyright,” Chapter 1 - Circular 92 | U.S. Copyright Office, https://www.copyright.gov/title17/92chap1.html#110 (accessed Nov. 19, 2024).

[3] “Copyright in the classroom,” Copyright in the classroom | UC Copyright, https://copyright.universityofcalifornia.edu/use/teaching.html#:~:text=Teachers%20and%20students%20have%20certain,108%20of%20US%20Copyright%20Law (accessed Nov. 19, 2024).

[4] “Dinky Tiny Golf Asset Pack - free by Mike,” itch.io, https://pixelbitsnbytes.itch.io/dinky-tiny-golf-free (accessed Nov. 19, 2024).

[5] A. Nguyen, E. Alderman, C. Lambert, and B. Maillet, Mini-golf mania, https://ealderman-uml.github.io/EduGame-Website/ (accessed Nov. 19, 2024).

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.