

Software Requirements Specification (SRS)

Grammar Grable

Team: Group 4

Authors: Darin Oti Abankwa, Nick Stearns, Andrew Brandao, Jarrett Williams, Andrew Clark

Customer: Massachusetts Department of Education

Instructor: Dr. James Daly

Table Of Contents

1 Introduction.....	3
1.1 Purpose.....	3
1.2 Scope.....	3
1.3 Definitions, acronyms, and abbreviations.....	4
1.4 Organization.....	4
2 Overall Description.....	5
2.1 Product Perspective.....	5
2.2 Product Functions.....	5
2.3 User Characteristics.....	6
2.4 Constraints.....	6
2.5 Assumptions and Dependencies.....	7
2.6 Apportioning of Requirements.....	7
3 Specific Requirements.....	8
4 Modeling Requirements.....	10
4.1 Use Case Diagram.....	10
4.2 Class Diagram.....	16
4.3 Sequence Diagrams: 1.....	25
4.4 Sequence Diagrams: 2.....	26
4.5 State Diagram.....	27
5 Prototype.....	29
5.1 How to Run Prototype.....	29
5.2 Sample Scenarios.....	29
6 References.....	36
7 Point of Contact.....	37

1 Introduction

The following document contains information about the video game prototype Grammar Grable. This prototype is an educational computer game that aims to educate students from the grades of 4th-6th grade on vocab, definitions, and punctuation. The document will then provide a detailed breakdown of the prototype, beginning with an overview of the game's concept. It will then cover various design elements, including specific requirements, modeling needs, and the overall design philosophy. To help illustrate the game's layout and mechanics, images of gameplay and modeling diagrams are included in later sections. Finally, a resources section will be provided to acknowledge the many sources that contributed to the development of this project.

1.1 Purpose

The Software Requirements Specification (SRS) document provides a complete description of all function, specification, and constraints for the software system Grammar Grable. The primary purpose of this document is to define what and why of the system, not the implementation. It acts as the core agreement between the development team and stakeholders, showing a clear and shared understanding of the project's objectives and expected outcomes.

The intended audience for this document is as follows:

- Customers and Stakeholders: This includes the future players, clients such as school districts wishing to understand a potential purchase, and Prof. Daly. This document allows them to better understand the game's purpose and intentions.
- Development Team: The software engineers, designers, and architects will use this document while developing and maintaining the game. It is intended to preserve an understanding of the game's purpose and design philosophy, and document its inner workings.

1.2 Scope

Grammar Grable is an edutainment based game designed to increase student engagement and incentivise the learning of multiple grammar concepts, including vocabulary, punctuation, and parts of speech. Students struggle with maintaining and studying grammar due to a lack of engagement. This game will tie these concepts to a fun puzzle escape room-esque game where they must find the key item corresponding to the correct answer to open the door and advance to the next level. This must be done while avoiding a teacher NPC who is pursuing them. This will allow students to learn while enjoying a fun and engaging game. While the game supports grammar learning, it will not act as a full grammar curriculum or replace formal classroom instruction. It will not provide standardized testing capabilities or teach content beyond the targeted grade 4–6 curriculum. The game will not offer multiplayer gameplay or online competition. Additionally, it will not collect personal student information. The system is intended to supplement learning through interactive gameplay.

1.3 Definitions, acronyms, and abbreviations

Below is a list of definitions and acronyms that are used in our game that are not inherently understandable without a proper introduction.

Grammar Grable: The title of the game that this document describes.

Player: The person playing the game.

Key/item: Interactable items that correspond to answers that players will use to solve a question.

Inventory: Where items collected by the player's character are stored.

Room/level: A subsection of the map that the player can walk around contained within 4 walls.

Enemy/Teacher: NPCs (non-player characters) that the player must avoid.

Health: An attribute of the player character which is decreased when questions are incorrectly answered or enemies contact the player.

Obstacles: Objects within a level that a player cannot move through and must go around.

Question: Prewritten English language problems with prescribed correct and incorrect answers.

Doors: Interactable objects which obstruct movement. Interacting with a door causes a question to be presented.

Popup: Text that appears when the player grabs an item or interacts with a door.

Detection: An enemy becoming aware of and starting to pursue a player character.

DetectionCone: The implementation of how an enemy in the level detects a player, it is a hidden cone that moves with the enemy. A player entering the detection cone causes the player to become detected.

Main Menu: The menu presented when the game first loads.

Normal Gameplay: The game state in which a player character can move about a level, collect items, interact with doors, and lose health to enemies.

1.4 Organization

The remainder of this document comprises 6 additional sections. Section 2 provides a detailed description of the product, encompassing its prospective functions, user characteristics, constraints, assumptions, and allocated requirements. Section 3 presents a comprehensive list of all specific requirements that the software must fulfill. Section 4 offers a technical overview of the modeling requirements that the software must satisfy, which includes a use case diagram, a class diagram, two sequence diagrams, and a state diagram. Section 5 outlines the prototypes of the software that will be delivered prior to the final product. Section 6 contains a compilation of references, while Section 7 identifies the point of contact.

2 Overall Description

Grammar Grable is designed as an educational game involving various vocabulary and definition based questions. The questions will use 4th-6th grade level vocabulary. It is a two-dimensional level-based exploration and puzzle game.

The following section explains the context of the project and these following subsections: the product's context and functions, a description for the specific user characteristics for the ideal users of this product, the limitations and restrictions of the project, the assumptions and dependencies of the project.

2.1 Product Perspective

Grammar Grable is an educational exploration and puzzle game that is a standalone application with the goal of educating elementary and early middle school students on vocabulary and definitions. The game functions as part of a broader educational environment, where students use the application during personal study time or in classroom settings on their own or school provided computers to reinforce vocabulary skills. The game will not interact with any external source and can be run on most computers without issue.

Students playing Grammar Grable will learn about a variety of different vocabulary and definitions based on the difficulty they select. This game will help them prepare for the vocabulary exams and quizzes they might receive as well as reading comprehension as they will understand more words as they read.

2.2 Product Functions

The product is an edutainment game. Its major functions are vocabulary based questions varying in difficulty, an engaging challenge of enemy avoidance, and vocabulary discernment tasks to improve reading comprehension. Grammar Grable functions as an edutainment game where the player navigates a 2D environment, collecting vocabulary-based items, using them to unlock rooms by selecting the vocabulary corresponding to the definition.

Once the game itself is started, a player-controlled character can move about the level. The objective is to find specific items that can be used to unlock each room's door. Each of these items will either have a word or a definition that will be displayed to the user. These items can then be used on the locked door in the level by selecting the correct one from the player's inventory with their mouse.

Figure I is a high-level diagram for Grammar Grable. Represented in this diagram is our goal to create a fun and engaging way for students to study and how it is broken down. Using a range of difficulties that the user can choose, and making each play slightly different with varying questions, we can prevent students feeling the gameplay is too difficult, too easy, or too repetitive. In addition, there is an engaging component of a moving enemy to instill a sense of competitiveness.

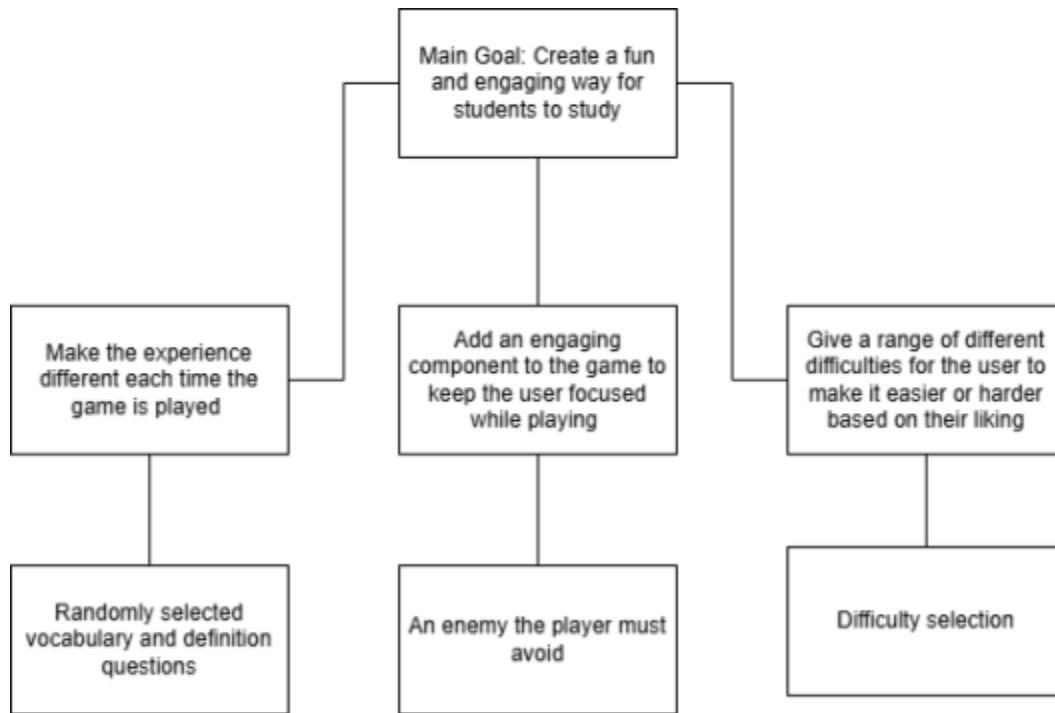


Figure 1: Breakdown Of Game

2.3 User Characteristics

The user is expected to have basic knowledge of a computer and how to use a keyboard and mouse to be able to play. The target audience for Grammar Grable is 4th to 6th grade students. They are expected to have learned enough basic vocabulary in order to understand the definitions and be able to associate them with their words. These vocabulary questions will assist them with their understanding for general reading vocabulary, mathematics vocabulary, and science vocabulary as listed for the 4th to 6th level in the Massachusetts Department of Education Curriculum.

2.4 Constraints

The game must be understandable and educational for students in grades 4-6, ensuring that in both classroom and personal use, players can be engaged with the content without confusion. All vocabulary must abide by the guidelines in the Common Core State Standards for English Language Art to support learning objectives and curriculums of students in schools. The game also needs to be safe for students in grade 4 and above considering students will be interacting with devices and be exposed to on screen content. To ensure an appropriate learning environment, the game must abide by all age-rating requirements and restrictions.

2.5 Assumptions and Dependencies

It is assumed that the user will use a modern computer with either macOS or Windows 11 as their operating system. The user's computer should have a functioning keyboard and mouse. To download the game the users must have internet access, however to play the game when downloaded and installed, no internet access is required. Game questions and instructions were written assuming that the user is a proficient English speaker.

2.6 Apportioning of Requirements

There will be no level-select, or more rooms and locations outside of the basic map. This game is designed only for a prototype meaning there will only be the preset levels created in order for there be enough gameplay to demonstrate the game's concept. For future improvements, the map can be expanded, and a level-select may be implemented, however for the basis of a prototype, only basic functionality is implemented. Different types of grammar questions can be added in the future to make it more difficult or varied.

3 Specific Requirements

1. The game shall consist of multiple levels.
 - 1.1. Each level shall be a traversable environment containing walls, interactable doors, obstacles, collectable items, and enemies.
2. The game shall feature a player character that can be controlled by the player to move about the environment.
 - 2.1. The player character's movement shall be constrained by the environment.
 - 2.1.1. The character shall not be able to move through walls.
 - 2.1.2. The character shall not be able to move through locked doors.
 - 2.1.3. The character shall not be able to move through obstacles.
 - 2.2. The player character shall have an inventory which stores items collected on a given level.
 - 2.3. The player character shall have a health attribute.
 - 2.4. The player character shall advance to the next level by completing the current level.
3. Each game level shall contain a grammar question which must be answered correctly to unlock the door.
 - 3.1. The grammar questions shall be designed to match the English curriculum for students in grades 4 through 6.
 - 3.1.1. All questions shall have a clear, unambiguous answer.
 - 3.1.2. The grammar questions shall be randomly selected from pools of questions separated by difficulty (e.g. Easy, Medium, Hard).
 - 3.2. Each level shall contain key items that are solutions to grammar questions.
 - 3.2.1. Key items shall be visually distinct and placed throughout the level.
 - 3.2.2. Key item types shall include, but are not limited to: Nouns, Verbs, Adjectives, Adverbs, Punctuation Marks, and Correctly Spelled Words.
 - 3.2.3. Key items shall be added to the player's inventory once interacted with.
4. The core gameplay shall involve solving grammar-based puzzles to progress through multiple levels while avoiding enemies.
 - 4.1. Locked door objects shall present grammar questions to the player upon the player character interacting with the door.
 - 4.2. A player must select a key item from their inventory to answer a question.
 - 4.2.1. To correctly answer a question, the player must select the item which corresponds to the correct answer.
 - 4.3. After a level's question is successfully answered the next level shall load.
 - 4.3.1. The game shall be completed when the final level's question is successfully answered.
 - 4.4. Incorrect answers shall resume play on the current level and cause a loss of player health.
 - 4.4.1. The game shall end if a player loses all their health.

5. The game shall feature enemies that can detect the player.
 - 5.1. Enemies shall pursue detected players and cause the player to lose health on contact.
6. The game shall provide a comprehensive menu system.
 - 6.1. The game shall display a Start Menu upon launch.
 - 6.2. The Start Menu shall provide options to start a new game and access other menus.
 - 6.2.1. The Start Menu shall give the player the ability to select the difficulty level (e.g. Easy, Medium, Hard).
 - 6.3. The game shall have a Controls Menu that displays the input mappings to the player.
 - 6.4. The game shall have a Pause Menu, accessible during gameplay.
 - 6.4.1. The Pause Menu shall provide options to resume, quit, and view the Controls Menu.
 - 6.5. Upon completing the game, an End Game menu shall congratulate the player for completing the game.
7. The game shall provide clear visual feedback to the player based upon user actions.
 - 7.1. Descriptive text shall appear when the player interacts with key objects, including locked doors, key items, and non-player characters.
 - 7.2. The game shall provide clear visual or audio feedback upon successful or failed puzzle attempts.

4 Modeling Requirements

This section contains diagrams that illustrate various components of the game. These include a use case diagram, a class diagram, two sequence diagrams, and a state diagram. Each diagram is accompanied by a detailed description that clarifies its purpose and the aspects it models.

4.1 Use Case Diagram

The use case diagram describes the user's interactions with the game where the primary objective is to open a door and advance to the next level. When trying to open a door, a question will always be asked and the user can either give a correct or incorrect answer. If the player answers correctly, they progress to the next level. If on the final level the player wins the game. Incorrect answers result in the player losing health.

Answering the question correctly requires having added the correct item to the player inventory, making adding items a significant subgoal. The player also has the option to pause the game, during which they can view the controls, change the difficulty, resume play, or quit the game.



Figure 2: Use Case Diagram

Use Case Name:	Add Item
Actors:	Player
Description:	The player is close enough to the object to interact with it. When interacted with, the item is picked up and stored in the player's inventory.
Type:	Secondary and essential
Includes:	None
Extends:	None
Uses cases:	The correct item is needed to correctly answer the question.

Use Case Name:	Answer Question
Actors:	Player
Description:	When a door is interacted with, the game will enter the Question Asked state. Here a question will be asked which the player can answer by selecting a key item in their inventory. The answer will be evaluated as correct or incorrect.
Type:	Primary and essential
Includes:	None
Extends:	None
Uses cases:	Correctly answering the question is required to open the door.

Use Case Name:	Change Difficulty
Actors:	Player
Description:	The player can change the difficulty of the game when they are in the pause menu.
Type:	Secondary
Includes:	None
Extends:	Pause
Uses cases:	Used to alter the game's difficulty.

Use Case Name:	Correct Answer
Actors:	Player
Description:	While in the Question Asked state the player selects the correct key item in their inventory. For most levels this will cause the next level to load. If on the final level the game will be won.
Type:	Secondary and essential
Includes:	None
Extends:	Answer Question
Uses cases:	Required to open a door and advance to the next level.

Use Case Name:	Incorrect Answer
Actors:	Player
Description:	When the player answers the question incorrectly, they will lose health and the Question Asked state will be exited, and normal gameplay will resume.
Type:	Secondary
Includes:	None
Extends:	Answer Question
Uses cases:	Punishes the player if they give an incorrect answer. Occurs in the question asked state.

Use Case Name:	Load Next Level
Actors:	Player
Description:	After a question has been successfully answered the next level will be loaded and normal gameplay resumes. If the player was on the final level then the game will be won instead.
Type:	Primary and essential
Includes:	None
Extends:	Correct Answer
Uses cases:	Occurs after a question has been successfully answered. All levels need to be loaded and completed for the game to be won.

Use Case Name:	Open Door
Actors:	Player
Description:	The player interacts with the door prompting a question to be asked. The door can be opened, advancing the player to the next level if the question is answered correctly.
Type:	Primary and essential
Includes:	Answer Question
Extends:	None
Uses cases:	Opening doors is required to advance to the next level, completing all levels is required to win the game.

Use Case Name:	Pause Game
Actors:	Player
Description:	The game is paused and a menu with options to resume, view controls, or adjust the difficulty is shown.
Type:	Secondary
Includes:	None
Extends:	None
Uses cases:	To pause the game and allow for the difficulty to be changed or controls viewed.

Use Case Name:	Quit Game
Actors:	Player
Description:	While the game is paused the player can select an option to exit the game and return to the main menu.
Type:	Secondary
Includes:	None
Extends:	Pause Game
Uses cases:	When the player wishes the exit to the main menu the game can be paused and quit game selected.

Use Case Name:	Resume Game
Actors:	Player
Description:	The player resumes the game if it is currently paused, the game will enter the normal gameplay state.
Type:	Secondary
Includes:	None
Extends:	Pause
Uses cases:	Returns a paused game to normal gameplay.

Use Case Name:	View Controls
Actors:	Player
Description:	While the game is paused the player can open a new screen showing the in-game controls.
Type:	Secondary
Includes:	None
Extends:	Pause Game
Uses cases:	Used while the game is paused to view the game's control scheme.

Use Case Name:	Win Game
Actors:	Player
Description:	The player has completed all levels and won the game. After winning the game will exit to the End Game menu.
Type:	Primary and essential
Includes:	None
Extends:	Correct Answer
Uses cases:	The game is won when the final level's door is opened.

4.2 Class Diagram

This diagram details the components of each class and outlines the relationships between the various classes within the game. The class diagram has nine major classes with one enumeration type and an interface class. The `PlayerCharacter` class houses attributes and methods tied to the playable character, such as the player's health, list of the player's inventory, and the character's `RigidBody`. `PlayerCharacter`'s methods include functions that move the character, keep track of the player's health, interactions with interactable objects, and using and collecting `Item` objects. The `Enemy` class inherits from the `Interactable` interface and houses only one additional attribute, its ID. The `Enemy` class has methods that relate to moving, detecting the player, and following a detected player. The `DetectionCone` class handles field of view detection between objects, and is used within the `Enemy` class's detection method.

The main interactable objects in the game have their own classes: `Door` and `Item`. The `Door` class has an ID so which door is being interacted with can be determined. The `Door` class also has a method which calls the `SceneManager` to display a question. The `Item` class contains an attribute to track whether or not it corresponds to the level's correct answer. Upon selection in the Question Asked state, an item calls a method to evaluate if it is the correct key. If an incorrect item is used, items have a method to decrease a player's health. `PlayerCharacter`'s store added `Item` objects in an inventory field. The `Door` and `Item` classes inherit from the `Interactable` interface.

`QuestionManager` and `QuestionBank` are central to the game's question logic. Upon a level loading, the `QuestionManager` class is responsible for randomly selecting `Question` objects from the `QuestionBank`. When a `Question` is selected, the correct answer, and one of several incorrect answers will be stored in the `QuestionManager`. The `QuestionManager` creates the key items present on any given level and ensures that one will always be the correct answer. The `QuestionManager` uses the `Difficulty` enumeration to store the player's selected difficulty, and draws `Questions` based on this setting.

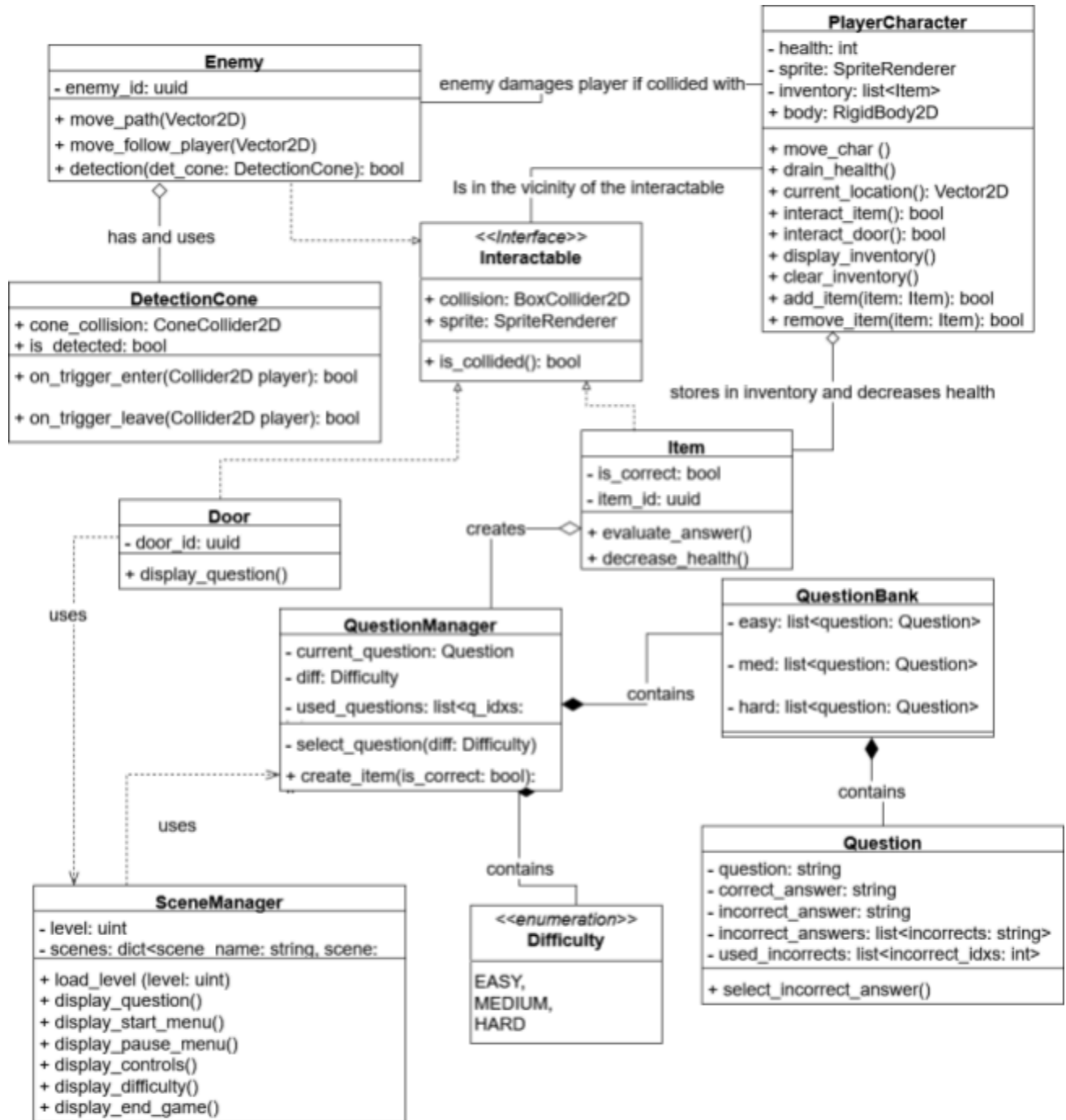


Figure 3: Class diagram

Class Data Dictionary

Element Name		Description
DetectionCone		Responsible for detecting collisions within a field of view. Determines if objects of a specific type are within the detection range.
Attributes		
	cone_collision: ConeCollider2D	Instance of ConeCollider which checks if collisions are detected within the (2-dimensional) cone shaped detection area.
	is_detected: bool	Value to track if a detection is occurring.
Operations		
	on_trigger_enter(player: Collider2D): bool	Returns true if a player character's collider has entered the detection area, false otherwise.
	on_trigger_leave(player: Collider2D): bool	Returns true if a player character's collider has left the detection area, false otherwise.
Relationships	Contained and used by the Enemy class.	

Element Name		Description
Difficulty		An enumeration for the game's difficulty. Questions are stored by difficulty, questions are drawn from the player-set difficulty pool.
Attributes		
	EASY	The easiest game setting.
	MEDIUM	A moderate difficulty setting.

	HARD	The most challenging game setting.
Relationships	Used by the QuestionManager class.	

Element Name		Description
Door		
Attributes		
	door_id: uuid	An identifier for doors.
Operations		
	display_question()	Upon a player interacting with it, Door calls the SceneManager to display the question.
Relationships	Derived from the Interactable class.	

Element Name		Description
Enemy		An NPC with the ability to detect a player. The enemy will pursue a detected player, dealing damage to them on contact.
Attributes		
	enemy_id: uuid	A unique identifier for the enemy.
Operations		
	move_path(loc: Vector2D)	Moves the enemy along a prescribed path.
	move_follow_player(loc: Vector2D)	The enemy pursues a detected player.

	detection(det_cone: DetectionCone): bool	Returns true when an enemy detects a player character.
Relationships	Contains and uses the DetectionCone class.	

Element Name		Description
Interactable		An interface class possessing the core properties of objects which interact with each other.
Attributes		
	collision: BoxCollider2D	An instance of a Unity collider. Collisions occur when objects' colliders intersect with other colliders.
	sprite: SpriteRenderer	Each Interactable object has a sprite that is shown in game.
Operations		
	is_collided(): bool	Returns true if an interactable object is colliding with a PlayerCharacter.
Relationships	Parent class to Enemy, Door, and Item classes. Interacts with the PlayerCharacter class.	

Element Name		Description
Item		Items have a correct or incorrect property, can be collected by players, and are selected by players to answer questions.
Attributes		
	is_correct: bool	Set when an item is created. This determines if the item correctly solves a level's question.

	item_id: uuid	A unique identifier for items.
Operations		
	evaluate_answer(): bool	Checks is_correct and then controls the flow of the game. If true, the next level will be loaded. If false, the player loses health, the item will be destroyed, and the game continues.
	decrease_health()	Causes the player to lose health after an incorrect answer is selected.
Relationships	<p>Derived from the Interactable class.</p> <p>Created by the QuestionManager class.</p> <p>Can be stored in a PlayerCharacter's inventory.</p> <p>Can call for a PlayerCharacter to lose health.</p>	

Element Name		Description
PlayerCharacter		The player's in-game avatar. Moves about levels collecting key items and answering questions to open doors and advance to the next level.
Attributes		
	health: int	Players lose health from contact with the enemy and incorrect answers. The game ends if health reaches zero.
	sprite: SpriteRenderer	Responsible for visually representing the player in-game.
	inventory: list<Item>	Stores items added by the player.
	body: Rigidbody2D	Object used to determine if a player character is colliding with an interactable object.

Operations		
	move_char()	Moves the player character throughout the game environment.
	drain_health()	Decreases a player's health after enemy contact or an incorrect answer.
	current_location(): Vector2D	Returns a player's current location within the game as a Vector2D.
	interact_item(): bool	Checks if an Item object is within range.
	interact_door(): bool	Checks if a Door is within range. If one is, the level's question will be displayed.
	display_inventory()	Displays all items in the player's inventory.
	clear_inventory()	Removes all items from the player's inventory.
	add_item(item: Item)	Adds an item to the player's inventory.
	remove_item(item: Item)	Removes an item from the player's inventory.
Relationships	<p>Takes damage from the Enemy class.</p> <p>Interacts with the Item and Door classes.</p> <p>Can store and take damage from Item objects.</p>	

Element Name		Description
Question		Contains a question, a correct answer, and several incorrect answers which are randomly assigned if a question is used.
Attributes		
	question: string	String of the question.

	correct_answer: string	String of the correct answer.
	incorrect_answer	String of the selected incorrect answer.
	incorrect_answers: list<incorrects: string>	A list of prescribed incorrect answers for a given question.
	used_incorrects: list<incorrect_idx: int>	A list of the indexes of incorrect_answers which have been selected for a given question. Intended to prevent repetition.
Operations		
	select_incorrect_answers()	Randomly selects an incorrect answer from a prescribed list of incorrect answers. Stores index of used incorrect answers in incorrect_answers to prevent repetition.
Relationships	Stored in the QuestionBank class.	

Element Name		Description
QuestionBank		Stores lists of Question objects separated by difficulty.
Attributes		
	easy: <question: Question>	List of the easy questions
	med: <question: Question>	List of the medium difficulty questions.
	hard: <question: Question>	List of the hard questions.
Relationships	Stores Question objects. Used and contained by the QuestionManager class.	

Element Name	Description
--------------	-------------

QuestionManager		Responsible for assigning a question for a level and creating items which correspond to correct and incorrect answers.
Attributes		
	current_question: Question	The Question object assigned to the current level.
	diff: Difficulty	Enumeration storing the player's selected difficulty.
	used_questions: list<q_idx:int>	A list of the indexes of Questions which have already been selected. Used to prevent repetition.
Operations		
	select_question(diff: Difficulty)	Randomly selected an unused question based on the player-selected difficulty.
	create_item(is_correct: bool): Item	Creates an item which is either correct or incorrect. There will only be one correct item per level.
Relationships	<p>Used by SceneManager.</p> <p>Contains a QuestionBank object.</p> <p>Creates Item objects.</p> <p>Has the Difficulty enum.</p>	

Element Name		Description
SceneManager		Controls the overall flow of the game. Responsible for loading levels and displaying menus.
Attributes		
	level: uint	The current level id.

	scenes: dict<scene_name: string, scene: Scene>	Dictionary with scene_names as keys and Scenes as values. Stores all scenes that will be used during the game.
Operations		
	load_level(level: uint)	Loads a level.
	display_question()	Loads the scene which contains the question for a given level.
	display_start_menu()	Loads the start menu.
	display_pause_menu()	Loads the pause menu.
	display_controls()	Loads the controls menu.
	display_difficulty()	Loads the change difficulty menu.
	display_end_game_menu()	Loads the end game menu.
Relationships	Calls QuestionManager. Is called by Door.	

4.3 Sequence Diagrams: 1

Figure 4 illustrates a player's progression through a single level of the game. First, the SceneManager loads the level. As part of the level loading process, the QuestionManager self-calls a method to randomly select a question from the QuestionBank. This then creates two item objects, an item corresponding to the correct answer and an item corresponding to an incorrect answer, which are scattered throughout the room. When close enough to these objects, the player then interacts with them, adding them to their inventory. Next, the player interacts with a door, which prompts the level's question to be shown.

Two scenarios are then described. In the first, the player selects the correct item, the item evaluates whether or not it is correct, and because it is both objects are destroyed and the next level loads. In the second, the player answers incorrectly, loses health, and the selected item is destroyed. Normal play then resumes without the level changing.

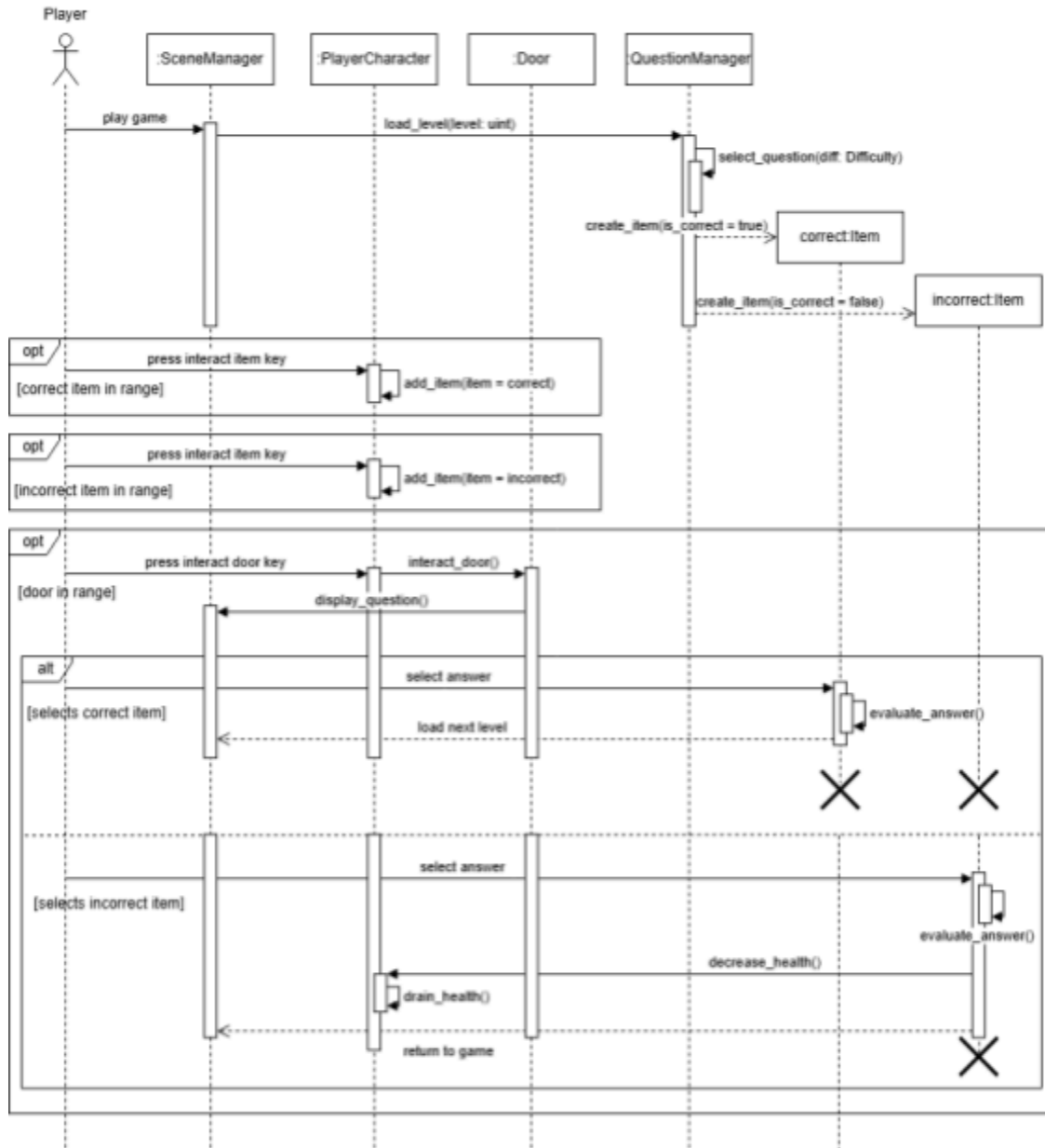


Figure 4: Sequence diagram 1

4.4 Sequence Diagrams: 2

Figure 5 shows an enemy in the level follows a certain set path throughout the level. As soon as the enemy detects the player within its Detection Cone, it then will start following the player. As soon as the player collides with the enemy, the player then loses health. If the player escapes the enemies vision, the enemy reverts to following its main path.

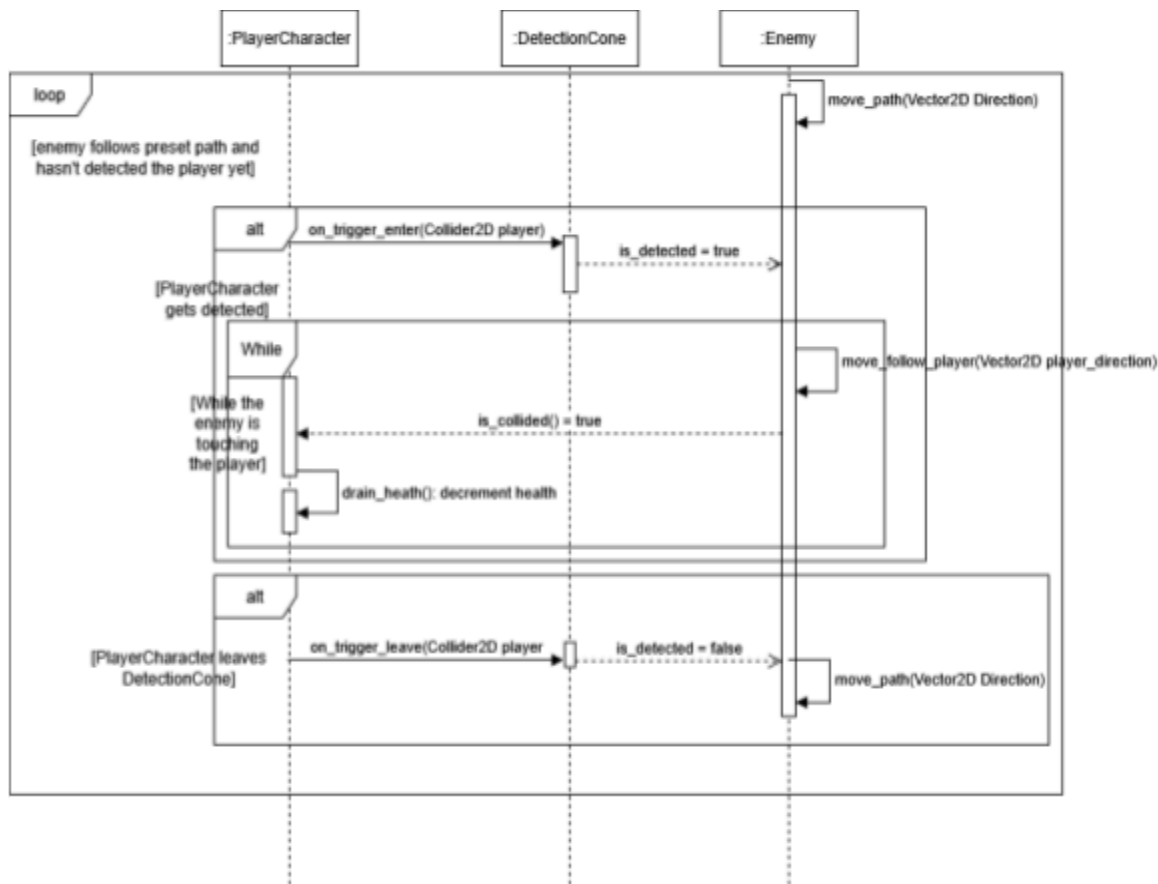


Figure 5: The sequence for the enemy detecting and attacking the player

4.5 State Diagram

The state diagram describes the control flow of the game. When the game is first started the player will be at the main menu. Here they have options to view and change the difficulty, view the controls, exit, and play the game. When playing the game a level will first be loaded, then the game will be in a normal gameplay state where the player can move about and collect items. When a player interacts with a door, a new state will be entered where a question is asked. An incorrect answer by the player leads back to normal gameplay, while a correct answer causes the next level to load. If the player has reached the last level, a correct answer will win the game and the player will return to the main menu.

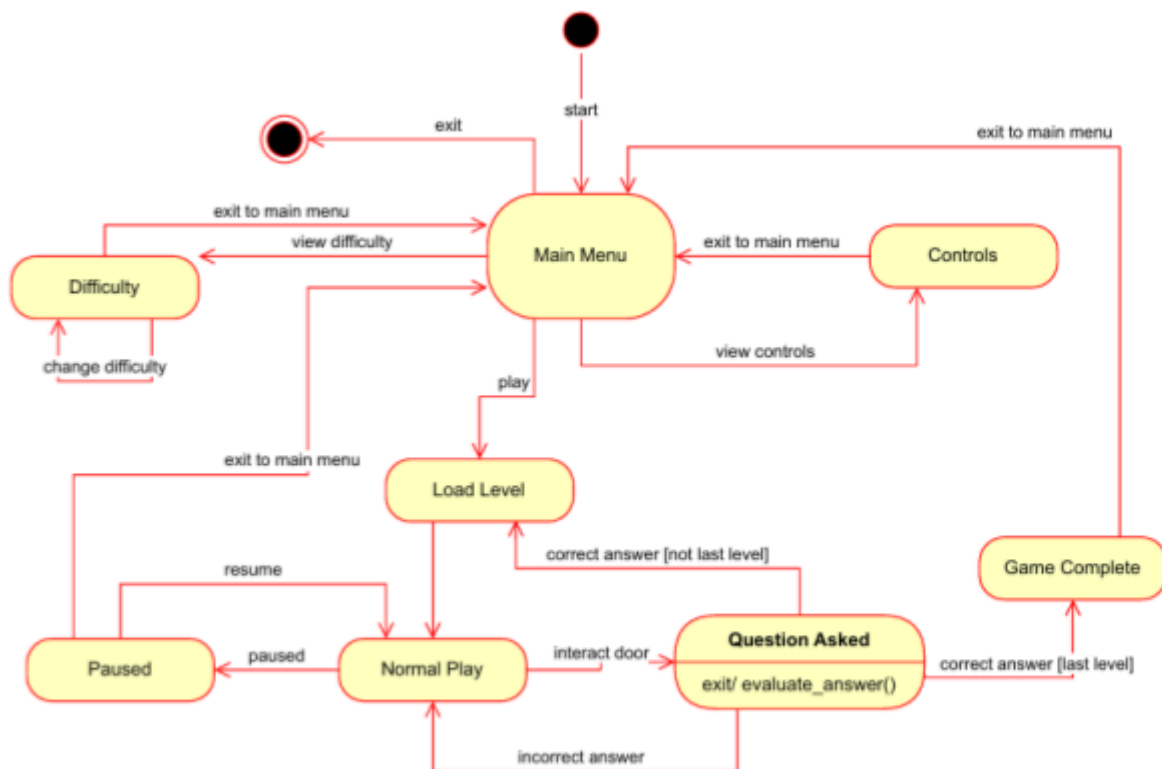


Figure 6: State Diagram

5 Prototype

In terms of system functionality, the prototype will be playable and will feature 3 different difficulty options for the player: easy, medium, and hard. This difficulty determines the set of questions the player will get. Easy has 4th grade vocab questions, medium has 5th grade questions, and hard has 6th grade questions. Once the play button is selected the main game will initiate where the player will traverse around the level to find the various grammar clues to reach to the next level. Gameplay will display enemy navigation, hierarchical question system for checking if the player acquired the correct answer, and level progression system to move on to the next level if grammar questions were answered successfully.

5.1 How to Run Prototype

The user needs either a mac computer or a windows computer and uses a mouse and keyboard. They must navigate to our website - <https://andrewbrandao88.github.io/GrammarGrable/> - and navigate to the download tab. Follow the instructions on the page and click the correct Download button for your computer. Once the button is clicked, a zipped folder should be downloaded. In a windows OS system, the user should unzip the folder to a location of their choosing. The user should then open up the folder titled, GrammarGrable windows release, and double click GrammarGrable.exe within that folder. In a MacOS system, the unzipped file should already be extracted within the downloads folder, move it to applications, double click GrammarGrable, or right click and select Open.

5.2 Sample Scenarios

5.2.1 The main menu will allow the player to select the difficulty of the questions, view the controls, and start or stop the game by using the left-click on the mouse.

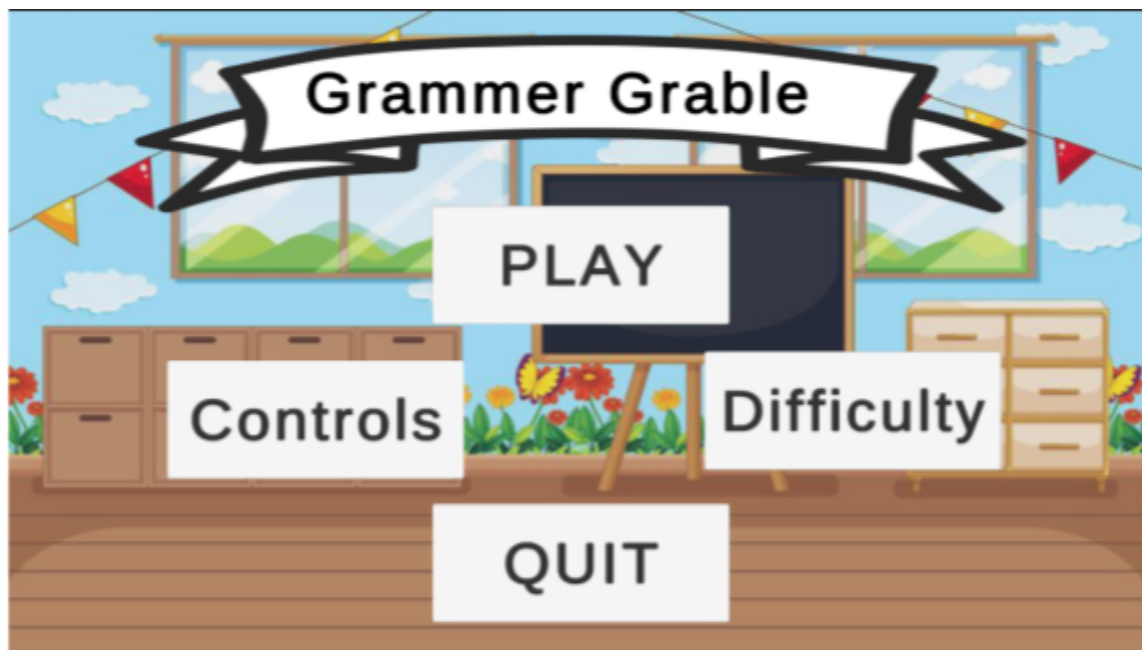


Figure 7: Main Menu

- 5.2.2 Difficulty selection menu with a back button to be able to return to the main menu still using the left click for all use on the menu.

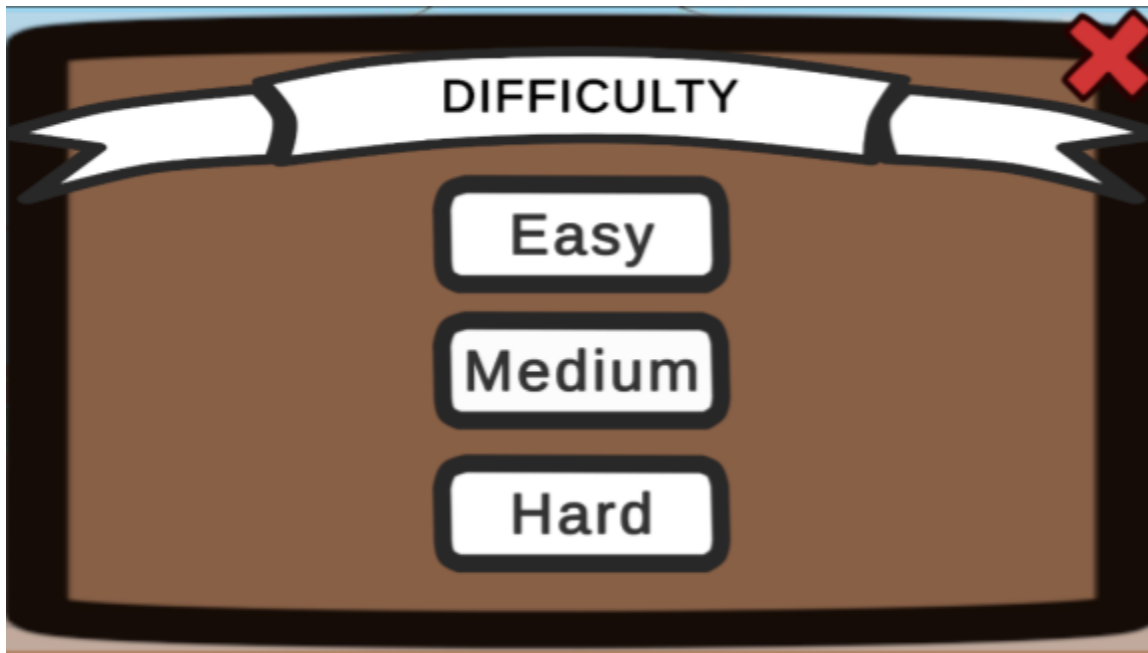


Figure 8: Difficulty Panel

- 5.2.3 Controls menu to display the controls to the user in order for them to understand how to play.

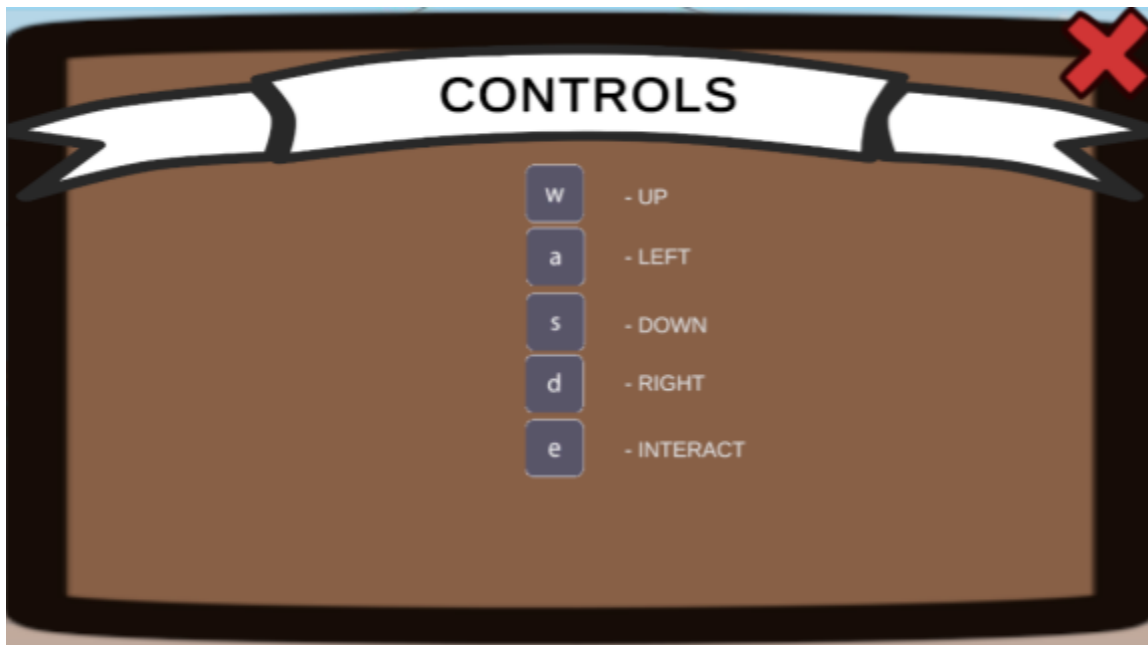


Figure 9: Control Panel

- 5.2.4 Below is the pause menu from in-game by clicking the pause button in the bottom right corner of the screen to allow you to then either resume, go back to the main menu, or view the controls.

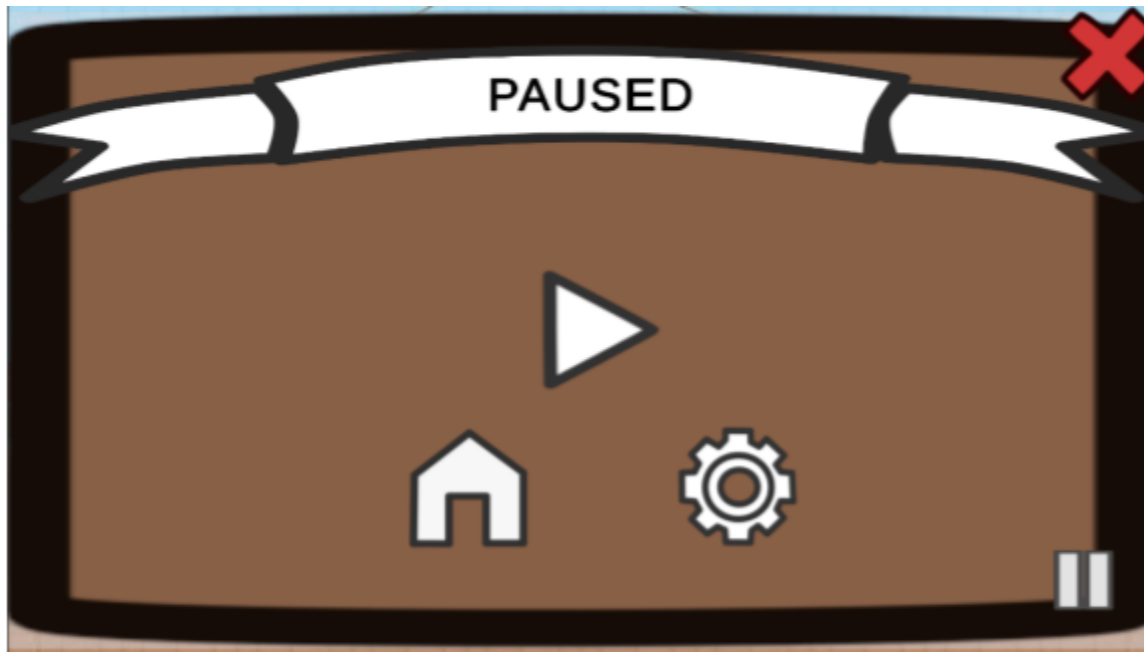


Figure 10: Pause Menu

- 5.2.5 After clicking on the Play button with the mouse, the player can interact with items by walking over them and pressing the interact key E on the keyboard.



Figure 11: Player Interacting With Item

- 5.2.6 The player interacts with the locked door to start the question, the player must now select the items in their inventory in order to answer the question presented to them. These words as appearing over the items in the inventory are

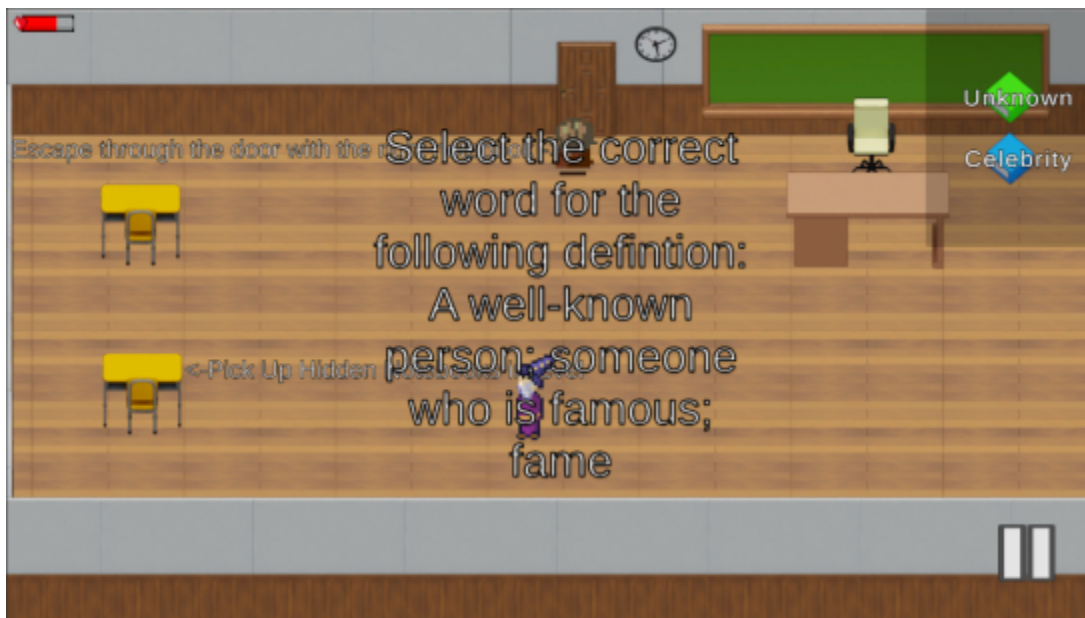


Figure 12: Player Interacting With Door

- 5.2.7 The player left-clicks on an item in the inventory that doesn't answer the question. If correctly guessed, the player will move onto the next level. The player must do all of this while avoiding the enemy. If correctly chosen, it loads the next scene.

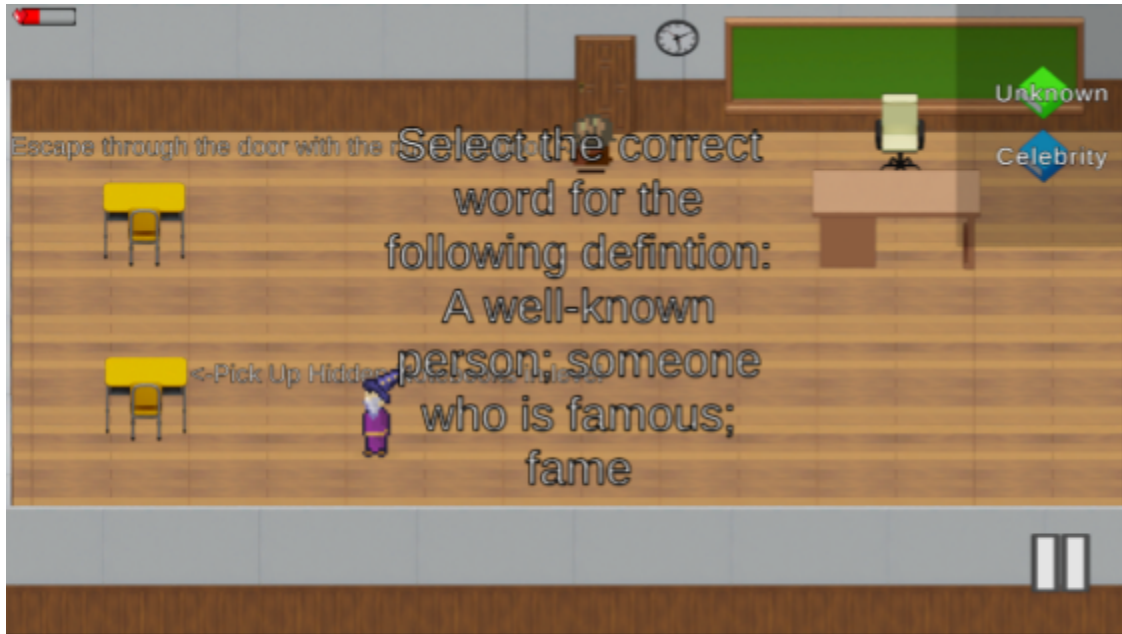


Figure 13: Player Using Item On Door

5.2.8 It then allows the user to either return to the main menu or continue to the next level of the game.



Figure 14: Continue Level Scene

- 5.2.9 The enemy preset on each level patrols to any number of patrol points waiting to see the player in its cone of vision.



Figure 15: Enemy Patrolling throughout the level

- 5.2.10 Once the player gets hit by the enemy enough they can get a game over screen in which they can either restart the current level or go back to the main menu to try again.

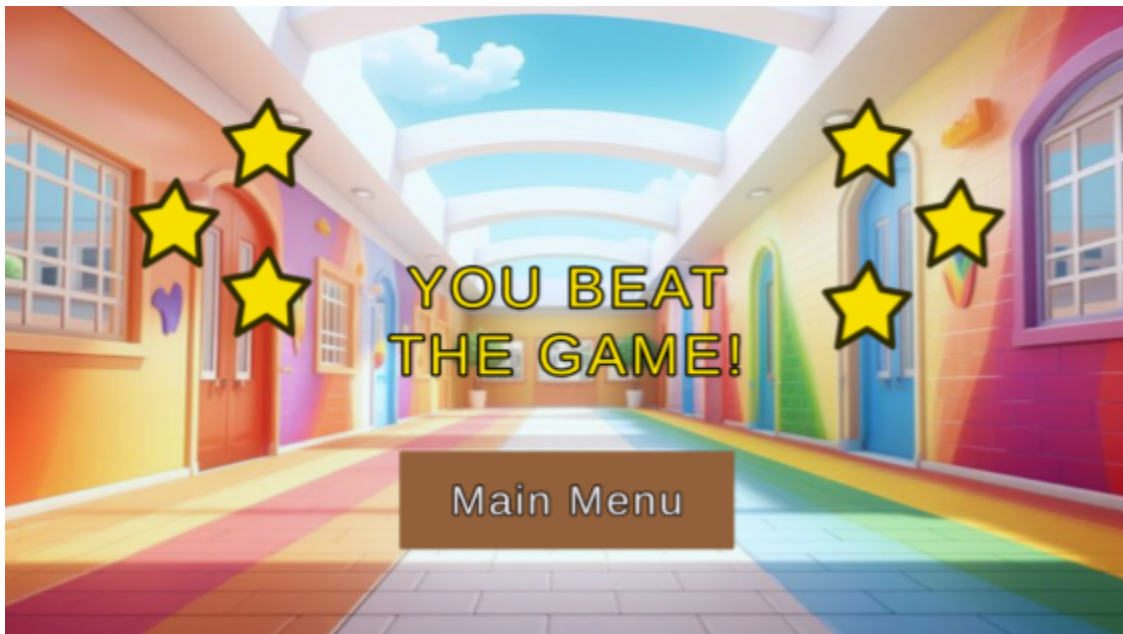


Figure 16: Win game Screen

6 References

- [1] Website: [Grammar Grable](#)
- [2] Unity Technologies, Unity Game Engine, Version Unity 6. [Online]. Available: <https://unity.com/products/unity-engine>
- [3] *Empty classroom interior with chalkboard for UI*, brgfx(Freep!k), 2025: https://www.freepik.com/free-vector/empty-classroom-interior-with-chalkboard_16845438.htm#fromView=search&page=1&position=0&uuid=12e28d48-307e-47c3-892d-463516ef9af0&query=school+Cartoon+background
- [4] *Full Keyboard Key Free Vector and PNG*, Designer Baba ([pngtree.com](#)), accessed 2025 https://pngtree.com/freepng/full-keyboard-key_6569504.html?sol=downref&id=bef
- [5] *UI Prototype Base*, VerzatileDev(itch.io), 2024: <https://verzatiledev.itch.io/ui-prototype-base/download/eyJleHBpcmVzIjoxNzYzNDZMzkwLCJpZCI6MjEyODQyNn0%3d.JygM3qBIq%2bZsiKnnMa9hvdGkOuQ%3d>
- [6] *Free 2D mega pack*, Brackeys(Unity asset Store), 2020, <https://assetstore.unity.com/packages/2d/free-2d-mega-pack-177430>
- [7] *2D School Classroom Asset Pack*, styloo([itch.io](#)), accessed 2025 : <https://styloo.itch.io/2dclassroom>
- [8] *Tiny RPG Town Environment*, Ansimuz (Unity asset Store), 2017 <https://assetstore.unity.com/packages/2d/environments/tiny-rpg-town-environment-88293>
- [9] *Grade 4 Vocabulary Words*, mrgobourne ([quizlet.com](#)), accessed 2025 <https://quizlet.com/332484570/grade-4-vocabulary-words-flash-cards/>
- [10] *5th Grade Vocabulary List*, FaulconD([quizlet.com](#)), accessed 2025, <https://quizlet.com/263029718/5th-grade-vocabulary-list-flash-cards/>
- [11] *6th Grade Vocabulary Word List*, Ellie_Mcelhenny5([quizlet.com](#)), accessed 2025, <https://quizlet.com/264988896/6th-grade-vocabulary-word-list-flash-cards/>

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.