Software Requirements Specification (SRS)
Leximon

Team: Group 2

Authors: Charlie Norton, Platon Supranovich, Samuel Stanley, Christopher Nguyen, Evan Sykowski

Customer: 4th to 5th grade English Language Arts teachers

Instructor: Dr. James Daly

Contents

1 Introduction

This Software Requirements Specification (SRS) describes Leximon, a top-down turn-based battle game where players use their knowledge of synonyms and antonyms to defeat game-controlled enemies [17]. This document is divided into different sections and subsections which cover different information about the project. These sections introduce the project and the structure of this SRS, describe the purpose of the Leximon game, lay out its full requirements, provide diagrams representing key parts of the Leximon game, and give instructions on how to run the prototype.

1.1 Purpose

The purpose of this SRS is to provide a complete definition of the functional and non-functional requirements for Leximon. This is so that the development team has a clear blueprint for implementation and testing. The SRS establishes realistic expectations and facilitates future expansion. This document serves as the primary reference for developers, testers, and stakeholders throughout the project lifecycle. For the customers, which are 4th and 5th grade English Language Arts teachers, the SRS communicates how Leximon will reinforce synonym and antonym recognition through engaging, self-contained battles without requiring grading or much preparation from teachers.

1.2 Scope

The result of this project will be a prototype of Leximon built in the Godot game engine containing a simple world with just enough functionality to engage with a full-featured turn-based battle system. Designing, testing, and improving the battle system is the main technical goal of this prototype.

Even in its prototype state Leximon will be playable by its 4th and 5th grade intended audience. Students will be able to engage in battles that remind them of what synonyms and antonyms are. Students will be presented with a list of words during battle and the effectiveness of their attack or defense will be determined by if they correctly identify a synonym or antonym. The objective of these battles will be to help students practice identifying synonyms and antonyms of words and to expand their vocabulary.

The delivered product is a playable prototype that contains only one battle arena. It also contains a limited set of words/enemies. It will not include world exploration, multiple levels, or player progression other than players' expanded vocabulary. It will also not include account systems, multiplayer, mobile support, or teacher analytics/dashboard features.

Leximon addresses the common 4th-5th grade struggle of vocabulary retention by turning synonym/antonym practice into an engaging Pokémon-style battle, providing immediate feedback and repeated exposure in a format students already love.

1.3 Definitions, acronyms, and abbreviations

- The Game: Leximon, the product described by this SRS.
- Game State: The data which determines the state of each object in The Game.
- The Computer: A device on which The Game runs.
- Player: The person who interacts with the game by providing input via The Computer.
- Type: A single-word label for a category of words which are synonyms or antonyms of each other.
- Action: A single word belonging to one or more Types.
- Dictionary: A collection of one or more Actions.
- HP: Health Points.
- Armor: A multiplier which affects how HP is changed.
- Character: An object in The Game which has a Type, a Dictionary, a maximum HP value, a current HP value, and an Armor value.
- Inventory: The collection of values owned by a Character.
- Player Character: The Character controlled by the Player.
- Enemy: A Character controlled by The Game which is hostile to The Player Character.
- Attack: An attempt by a Character to reduce another Character's HP.
- Defense: An attempt by a Character to increase their own Armor value.
- Relation Engine: An engine to determine the effectiveness of an Attack or Defense based on the relation between the Action chosen and the Type of the Character it affects.
- Turn: An opportunity for the Player Character or an Enemy to Attack or Defend while interacting.
- Round: A single loop of Turns, one for the Player Character and one for the Enemy.
- Battle: A sequence of Rounds.

1.4 Organization

The rest of this SRS will be structured as follows:

- **Section 2:** A description of The Game which includes its context, primary functions, user characteristics, constraints, dependencies, and future expansions.
- **Section 3:** An enumerated list of the complete requirements for The Game.
- **Section 4:** Use Case, Class, Sequence and State diagrams which describe The Game's behavior and requirements in visual form. A short description of what each diagram depicts is included.
- **Section 5:** A description of the prototype, how to run it, and examples of using it.
- **Section 6:** A list of sources referenced or used in this SRS.
- **Section 7:** Point of contact.

2 Overall Description

This section will describe an overview of The Game. First the background, overall educational intention and the requirements to play the game will be detailed. Then the main functions and high-level educational goals will be described. After that the characteristics of the intended users of The Game are described. Then the constraints on the development of The Game are defined and then the assumptions and dependencies. Finally the requirements beyond the scope of the project are listed.

2.1 Project Perspective

One of the key linguistic concepts students learn near the end of elementary school is synonyms and antonyms. These constructs can greatly enhance students' ability to identify and define new vocabulary words. For many people it is easier to understand the meaning of a word if presented with a synonym or antonym than when presented with its definition. Synonyms are crucial for creating variation in writing and can often make the difference between boring and interesting literature.

Leximon is a game which presents students with an opportunity to practice identifying synonyms and antonyms in a fun and engaging way. The Game is intended to be introduced to students by teachers as a practice and learning tool which may be played in class or on students' own time. It is a standalone product which enhances existing education about synonyms and antonyms. Figure 1 demonstrates how Leximon fits into its intended educational system.

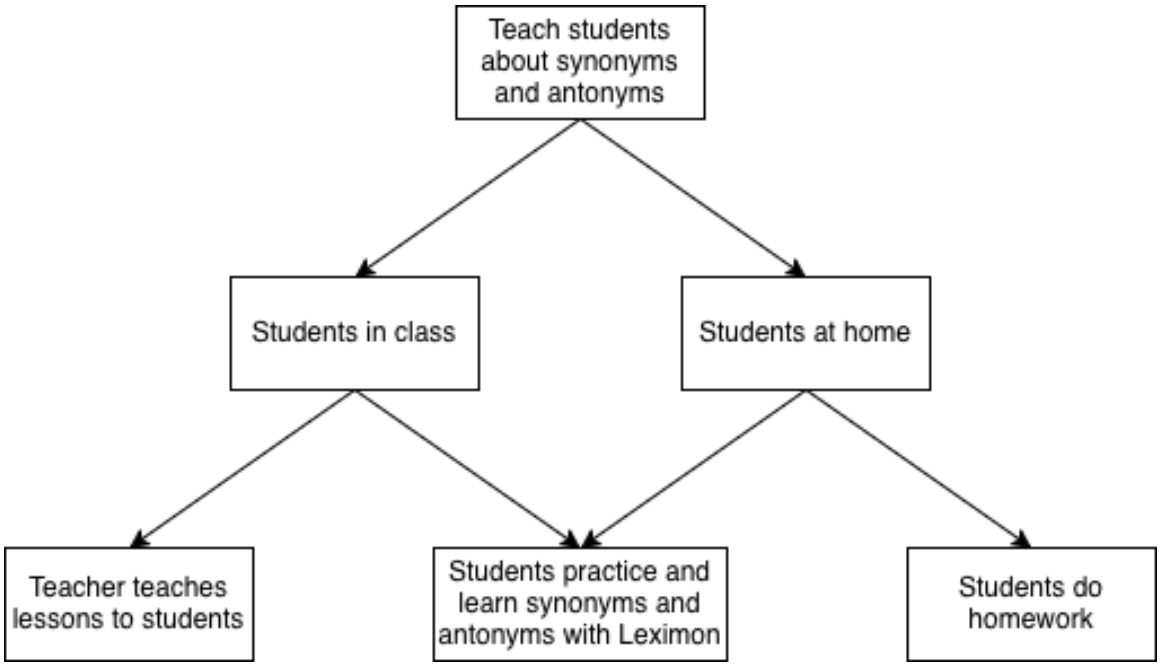

Figure 1: The bigger system Leximon fits into

The Game requires a computer with a keyboard, mouse and monitor to be played. Currently Godot does not support compiling C# for web, so The Game must be downloaded as a program and run on The Computer [16]. The Computer must run the Windows operating system.

2.2 Project Functions

The primary function of The Game is to engage the Player in turn-based Battles where they practice their knowledge of synonyms and antonyms and learn new words by identifying synonyms and antonyms. See the left-most goal in Figure 2. Players will navigate a two-dimensional top-down world and interact with enemies they see to start Battles. In a Battle Players will be presented with a list of Actions and will be shown the Type of each Character. Players will select an Action from the list and choose either to Attack or Defend. If they choose to Attack then the Action they choose must be an antonym of the Enemy's Type to be effective. If they choose to Defend then the Action they choose must be a Synonym of their own Player Character's Type to be effective.

For example, suppose the Enemy has the Type "Cold" and the Player Character has the Type "Dark". To perform an effective Attack, the Player might choose the Action "scorching", since this word is an antonym of the Enemy's Type. To perform an effective defense, the Player might choose the Action "dim", since this word is a synonym of the Player Character's Type. In this way, Players will test their knowledge of both synonyms and antonyms, which fulfills part of the top-most goal in Figure 2.

While not in Battle Players may view their Dictionary containing all their words and the definitions of their words. Since a word may have multiple meanings for different contexts, multiple definitions corresponding with each relevant Type will be displayed. Players may also see the definitions of the visible words during a Battle. This teaches Players new words so that they may learn their synonyms and antonyms during Battle which, in turn, reinforces the definitions they learned. This is the right-most goal in Figure 2, which fulfills the teaching part of the top-most goal.

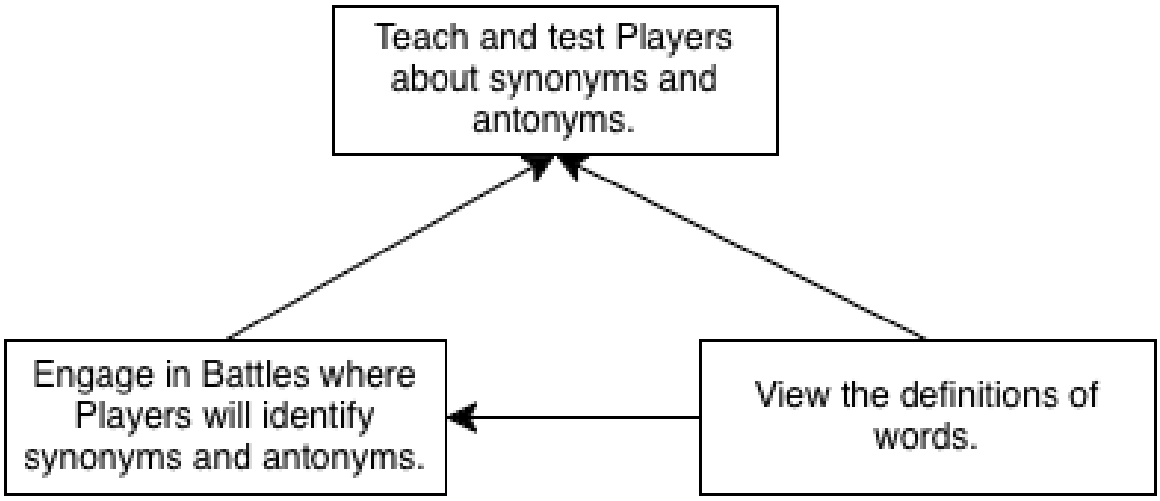

Figure 2: High Level Goals

2.3 User Characteristics

The intended users of The Game are students in grades 4 and 5, the grades where students learn about synonyms and antonyms [11]. Alternatively students in different grades who are learning about synonyms and antonyms are also intended users. The Game is intended for students who are already at least somewhat familiar with what synonyms and antonyms are but still need additional practice identifying them or need to practice learning new words using synonyms and antonyms. Users should also be comfortable using a mouse and keyboard to play video games.

2.4 Constraints

Technical Constraints: The development of The Game is constrained by the capabilities of the Godot game engine with C#. While the game engine is very permissive in the features which may be implemented, it only has the capability to compile C# projects for Windows, Linux and macOS, not for the web [16]. This means that the constraints of the Players' computers must be taken into account.

Developmental Constraints: The development of The Game is also constrained by its developers and its timescale. Since this project is the first time several of its

developers have used Godot or C#, learning both these tools presents a constraint on the complexity and quantity of the work which may be done on The Game. Finally, this project is constrained by the amount of time given to complete it.

Legal Constraints: All assets and code used in The Game must adhere to U.S. copyright law and must provide attribution when applicable.

Safety Constraints: Physical safety is not a concern for Leximon since it has no direct hardware control, but user privacy is a consideration. Therefore only the data necessary to operate The Game will be collected and none of it will be transmitted off The Computer.

2.5 Assumptions and Dependencies

Assumptions: The Game assumes that users have access to computers with a keyboard, monitor and mouse running the Windows operating system. It also assumes that users' computers are powerful enough to render simple 2D graphics. Lastly, The Game assumes that users are at least somewhat familiar with what synonyms and antonyms are.

Dependencies: The Game depends on the user's computer supporting the Vulkan, Direct3D 12 or Metal rendering APIs, which are the default rendering drivers used by Godot [15].

2.6 Apportioning of Requirements

The current prototype focuses mainly on the Battle system, testing if the system is technically feasible, educational, and fun. In the future, the world outside of Battles may be expanded to include a story and main quest, Characters other than Enemies, different ways to obtain Actions and different worlds which may be visited. Battles may be expanded to include more types of Actions than just Attack and Defend, variations on the strength and quantity of Enemies, and an experience system to allow Players to level up. The ability to play and store multiple separate Game States may also be added.

3 Specific Requirements

1. The Game will have a Hub World containing the Player Character and five Enemies.
    (a) The Player Character will be able to move freely around the Hub World.
    (b) Each time the Player Character enters the Hub World enemies will be spawned at random locations until there are five enemies in the Hub World.
2. While in the Hub World the Player will be able to open an Inventory Menu.

(a) Actions from the Player Character's Dictionary will be displayed.

i. Selecting an Action will display the Types it belongs to and the definitions of the Action corresponding to each Type.

(b) The Player Character's Type will be displayed.

(c) The Player Character's current and maximum HP will be displayed.

3. A new Game State will be initialized using the following attributes:

(a) The Player Character will start at the upper left corner of the Hub World.

(b) Every Action in The Game will be added to the Player Character's Dictionary.

(c) All Characters will have 100 current and maximum HP.

(d) All Characters will have 1.0 Armor.

(e) The Player Character's Type will be randomized.

4. While in the Hub World the Player will be able to save the current Game State.

(a) The Game State will be saved persistent storage on The Computer.

(b) The position and data of every Character will be saved.

5. Upon starting the game the player will be shown a Main Menu scene with the name of The Game and several options.

(a) An option labeled "Start" will retrieve the saved Game State and apply it to The Game. If no saved Game State is available, a new Game State will be initialized and applied to The Game.

(b) An option labeled "About" will open The Game's website in an external web browser.

(c) An option labeled "Exit" will close The Game.

6. While in the Hub World the Player will be able to exit to the Main Menu.

(a) Exiting to the Main Menu will save the current Game State.

7. The Player Character interacting with an Enemy in the Hub World will start a Battle.

(a) Starting a Battle will move the Player Character and the Enemy to the Battle Scene.

i. The Enemy's Type will be randomized.

ii. The Player Character and the Enemy will be displayed facing each other.

iii. The Type, HP, and Armor value of each Character will be displayed.

iv. A list of five Actions selected from the Player Character's Dictionary at the start of the Battle will be displayed.
    A. One synonym of the Player Character's Type or one antonym of the Enemy's Type will be selected.
    B. One antonym of the Player Character's Type or one synonym of the Enemy's Type will be selected.
    C. Three Actions will be randomly selected from the rest of the Player Character's Dictionary.

(b) Each time it is the Player Character's turn the Player will select a displayed Action.
    i. Upon selecting an Action the Player will be prompted to choose whether to perform an Attack or Defense using the Action.
    ii. Attacks by the Player Character will deal a normal damage of 10 HP.
    iii. Defenses by the Player Character will normally increase their Armor value by 7% each time.

(c) The strength of the Player Character's Attacks will be determined by the Relation Engine.
    i. An Attack's damage will be unchanged if the Action belongs to a Type which is an antonym of the Enemy's Type.
    ii. Otherwise, an Attack will deal 1/4 damage if the Action belongs to a Type which is a synonym of the Enemy's Type.
    iii. An Attack will deal 0 damage if none of the Types the Action belongs to are antonyms or synonyms of the Enemy's type.

(d) The strength of the Player Character's Defenses will be determined by the Relation Engine.
    i. A Defense's effectiveness will be unchanged if the Action belongs to a Type which is a synonym of the Player Character's Type.
    ii. Otherwise, a Defense will be 1/4 effective if the Action belongs to a Type which is an antonym of the Player Character's Type.
    iii. A Defense will not increase Armor value if none of the Types the Action belongs to are antonyms or synonyms of the Player Character's type.

(e) Each time it is the Enemy's turn The Game will randomly select Attack or Defense.
    i. Attacks by the Enemy will deal 7 HP of damage.
    ii. Defenses by the Enemy will increase their Armor value by 5% each time.

(f) All Attack values will be divided by the Armor value of the Character they affect after the Relation Engine is run.

(g) The Battle will end when one of the Characters runs out of HP.

i. The Player Character running out of HP will be considered a Player Character Loss.
   A. The Enemy will be returned to their position in the Hub World before the Battle started.
   B. The Enemy's current HP will be reset to their maximum HP and their Armor value will be reset to 1.0.
ii. The Enemy running out of HP will be considered a Player Character Win.
   A. The Enemy will be removed from the Hub World.
   B. The Player Character's Type will be randomized every third Player Character Win.
iii. The Player Character will be returned to their position in the Hub World before the Battle started.
iv. The Player Character's current HP will be reset to their maximum HP and their Armor value will be reset to 1.0.
v. The Hub World will be shown to the Player.
vi. The current Game State will be saved.

4 Modeling Requirements

The following section presents all project-related diagrams, including a use case diagram, a class diagram, two sequence diagrams and a state diagram. Each diagram is accompanied by descriptive content to provide additional context and clarification.

4.1 Use Case Diagrams

Figure 3 shows the use cases for the Player, demonstrating how they interact with The Game and how different parts of The Game interact internally to perform concrete use cases. Uses cases are represented by ovals and each use case directly connected to the Player is an action the Player directly takes. Use cases pointed to by an `includes` arrow are used by the pointing use case to perform their functionality. Use cases pointing to another use case with an `extends` arrow are conditionally used by the other use case. Lose Battle and Win Battle are marked as such because these are the two possible outcomes from ending a battle.

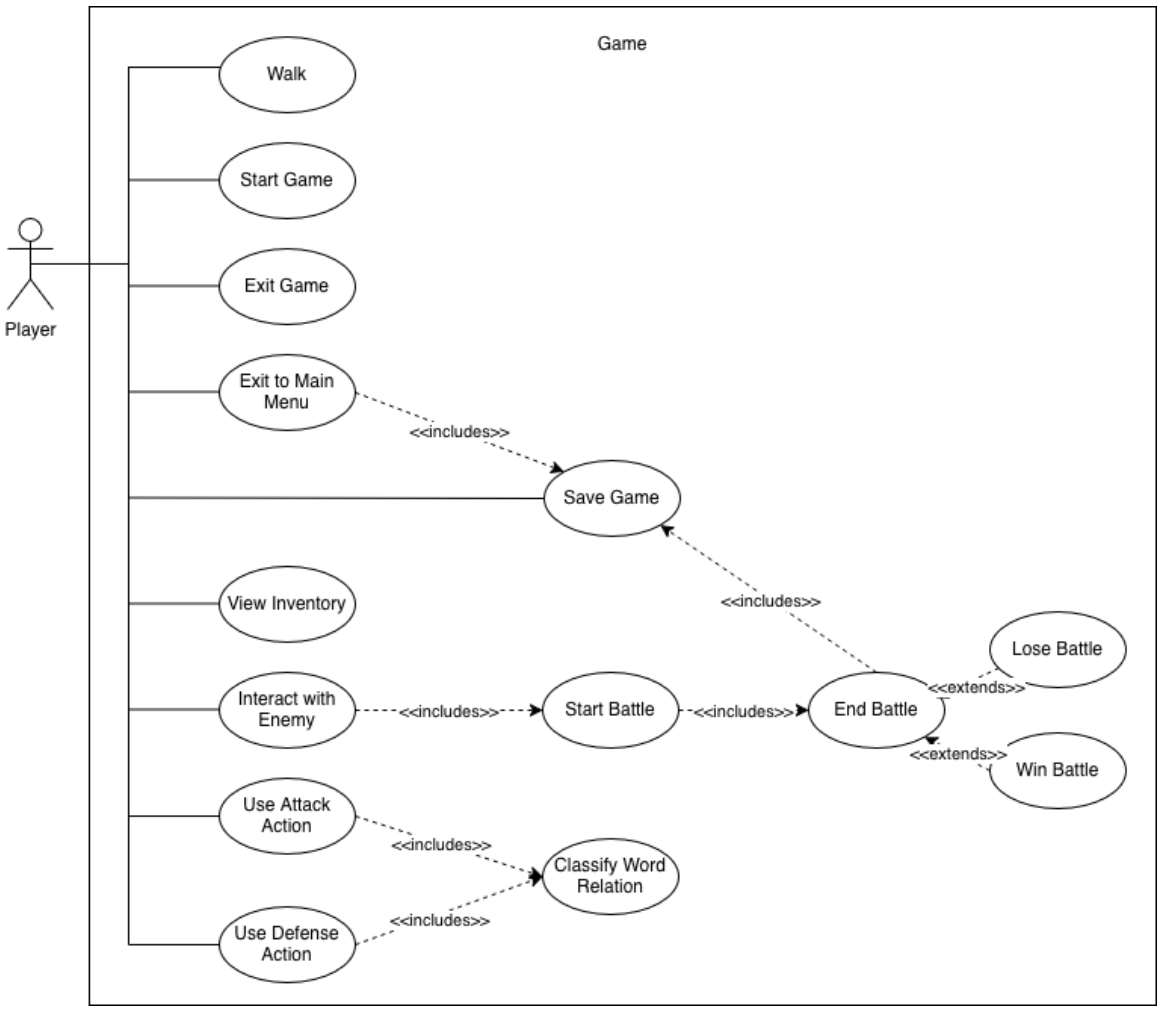

Figure 3: Use Case Diagram

Use Case Name	Walk
Actors	Player
Description	Move the Player's position in the Hub World. The Player may walk in any direction using the keyboard but may not walk past the edge of the Hub World.
Type	Primary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 1.a
Use Cases	N/A

Use Case Name	Start Game
Actors	Player
Description	The button in the Main Menu to retrieve the saved Game State and apply it to The Game. Initializes a new Game State if none exists.
Type	Primary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 5.a
Use Cases	N/A

Use Case Name	Exit Game
Actors	Player
Description	The button in the Main Menu to exit the game.
Type	Primary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 5.c
Use Cases	N/A

Use Case Name	Exit to Main Menu
Actors	Player
Description	A button accessible from the Hub World to return to the Main Menu. Automatically saves the Game State.
Type	Primary
Includes	Save Game
Extends	N/A
Cross-refs	Requirement 6
Use Cases	N/A

Use Case Name	Save Game
Actors	Player, System
Description	Saves the current Game State, either through a button accessible in the Hub World or through internal methods.
Type	Primary
Includes	N/A
Extends	N/A
Cross-refs	Requirements 4, 6.a, and 7.g.vi
Use Cases	Exit to Main Menu, End Battle

Use Case Name	View Inventory
Actors	Player
Description	View the Player's Dictionary, Type, current and maximum HP from the Hub World.
Type	Primary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 2
Use Cases	N/A

Use Case Name	Interact with Enemy
Actors	Player
Description	While in close proximity to an Enemy in the Hub World, a button may be pressed to interact with them, which starts a Battle.
Type	Primary
Includes	Start Battle
Extends	N/A
Cross-refs	Requirement 7
Use Cases	N/A

Use Case Name	Start Battle
Actors	System
Description	Changes the scene to the battle scene and places the interacted with enemy in the opponent's slot.
Type	Secondary
Includes	End Battle
Extends	N/A
Cross-refs	Requirement 7.a
Use Cases	Interact with Enemy

Use Case Name	End Battle
Actors	System
Description	End the current battle and return the player to the hub world. Restore the player's HP
Type	Secondary
Includes	N/A
Extends	Lose Battle, Win Battle
Cross-refs	Requirement 7.g
Use Cases	Start Battle

Use Case Name	Lose Battle
Actors	System
Description	Return the player and enemy to the hub world. Restore both of their HP
Type	Secondary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 7.g.i
Use Cases	N/A

Use Case Name	Win Battle
Actors	System
Description	Return the player to the hub world. Restore their health.
Type	Secondary
Includes	N/A
Extends	N/A
Cross-refs	Requirement 7.g.ii
Use Cases	N/A

Use Case Name	Use Attack Action
Actors	Player
Description	While in a Battle the Player may use an Action to Attack. The Game will classify the word relation and apply the appropriate damage to the Enemy.
Type	Primary
Includes	Classify Word Relation
Extends	N/A
Cross-refs	Requirement 7.b.i
Use Cases	N/A

Use Case Name	Use Defense Action
Actors	Player
Description	While in a Battle the Player may use an Action to Defend. The Game will classify the word relation and apply the appropriate Armor value to the Player Character.
Type	Primary
Includes	Classify Word Relation
Extends	N/A
Cross-refs	Requirement 7.b.i
Use Cases	N/A

Use Case Name	Classify Word Relation
Actors	System
Description	Determines if the two words are synonyms, antonyms, or neither.
Type	Secondary
Includes	N/A
Extends	N/A
Cross-refs	Requirements 7.c and 7.d
Use Cases	Use Attack Action, Use Defense Action

4.2 Class Diagram

Figure 4 shows the structure of the main classes that make up the game. Each class includes the fields and methods that define its role as well as how it interacts with other elements of the program. Boxes represent classes, where the top segment is the name of the class, the middle is the class's attributes and the bottom segment is the class's methods. Lines represent relationships between classes, and the arrows on the lines represent the kind of relationship. White triangular arrows represent inheritance, and dashed lines represent inheritance from an interface. White diamond-shaped arrows represent aggregation while black diamond-shaped arrows represent composition.

At the core of The Game is the `Character` class, which stores information about the Player and Enemies, which are child classes. The `BattleUI` class handles the logic for interactions between the Player and an Enemy interact and controls the turn flow during a Battle. The `WordManager` class handles Actions and resolves the relationships between Actions and types during Battles. The `HubWorld` class manages the Hub World and the Characters in it, starts Battles, and allows the Player to quit to the Main Menu.

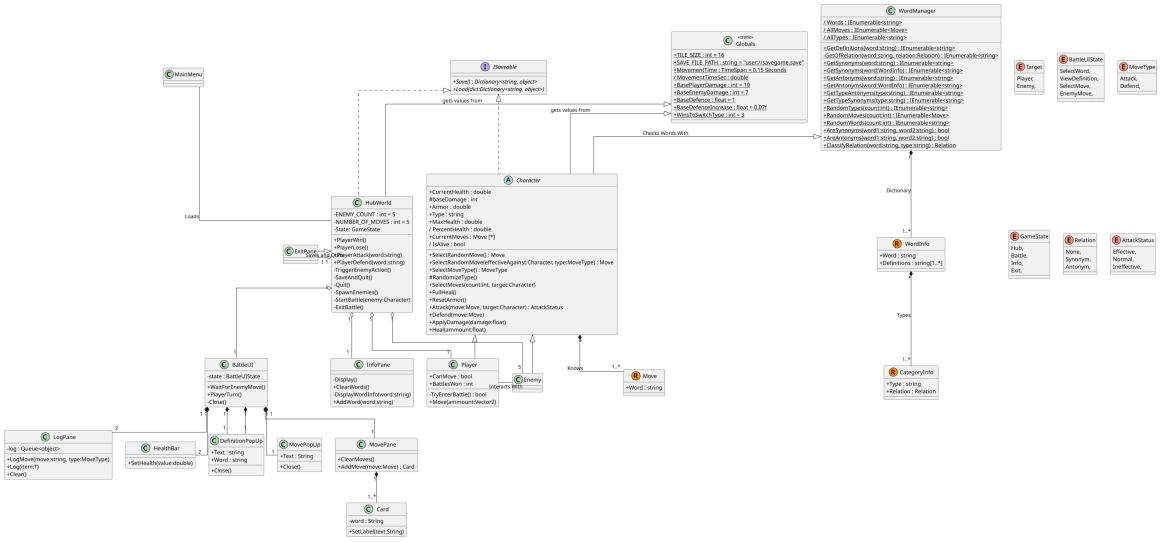

Figure 4: Class Diagram

Class Name	ISaveable	
Description	An item that can be saved and loaded as a JSON object	
Attributes	Save(): Dictionary<string, object>	Converts the item into a JSON object
	Load(dict: Dictionary<string,object>)	Loads the items from a JSON object

Class Name	Character		
Description	The Base Class for all things that can enter battles and know words		
Extends	ISaveable		
Attributes	CurrentHealth	double	The remaining health of the character
	baseDamage	Int	The damage done against opponents before any modifiers
	Armor	double	Damage applied to this character is divided by this value
	Type	string	One of the word categories, affects effectiveness of different Words against the character
	MaxHealth	double	The most health the character can have
	PercentHealth	double	The character's CurrentHealth divided by the MaxHealth
	CurrentMoves	Move[*]	The moves currently available for the Character to use
	IsAlive	Bool	If the character's health is above 0
Operations	SelectRandomMove(): Move		Return a random Move from CurrentMoves
	SelectRandomMove(effectiveAgainst: Character, type: MoveType) : Move		Return a random move from CurrentMoves, weighted to be

		more effective against effectiveAgainst, for the move type
	SelectMoveType(): MoveType	Weighted random selection for either Attack or Defend
	RandomizeType()	Set the type to a random one selected from the types in WordManager
	SelectMoves(count: int, target: Character)	Set CurrentMoves to count randomly select moves, with one guaranteed to be effective for either target or self, and one guaranteed to be normal for either target or self, if possible
	FullHeal()	Set CurrentHealth to MaxHealth
	ResetArmor()	Set Armor to 1
	Attack(move: Move, target: Character): AttackStatus	Use move to attack target, returns how effective the attack was based on its type and the target's type
	Defend(move: Move)	Defend self, using move as the word
	ApplyDanage(damage: float)	Calculate damage reduction from Armor, and reduce current health by reduced damage
	Heal(ammount: float)	Increase CurrentHealth by ammount, up to MaxHealth
Relationships	Knows Move	The character knows a number of moves which can be used
	Gets values from Globals	The character retrieves some constant values from Globals
	Gets words from WordManager	The character checks with the word manager to know which words are valid and how they relate to each other

Class Name	Player		
Description	Represents the player character, holds their stats and list of words		
Extends	Character		
Attrubutes	CanMove	Bool	If the player is currently able to move
	BattlesWon	Int	The current number of battles the player has won

Operations	TryEnterBattle()	Finds the nearest enemy in range, if it exists enter battle with it
	Move(amount: Vector2)	Moves the player by amount, updating sprite if needed and forbidding out of bounds movement
Relationships	Interacts with Enemy	The player can interact with an enemy to enter battle with it. The player can attack that enemy while in a battle
	Contained by HubWorld	The HubWorld contains the Player, triggers its saving and loading, and communicates with it about the state of the game

Class Name	Enemy	
Description	Represents an NPC which can be battled	
Extends	Character	
Relationships	Interacts with Player	Enters a battle when the player initiates. Attacks the player in the battles
	Contained by HubWorld	The HubWorld contains ENEMY_COUNT enemies, triggers its saving and loading, and communicates with it about the state of the game

Class Name	Move		
Description	Represents a Move a Character can make. A wrapper around a string		
Attributes	Word	string	Value wrapped by move
Relationships	Known by Character		A character knows some number of moves that it can use in attacks and defence

Class Name	HubWorld		
Description	The main manager of the game, contains all game elements and communicates with them		
Extends	ISaveable		
Attributes	ENEMY_COUNT	int	The number of enemies that can be spawned on the map
	NUMBER_OF_MOVES	int	How many moves the player is given to choose from during a battle turn
	State	GameState	The current state that the game is in
Operations	PlayerWin()		Leaves the current battle, increases the number of player wins, and resets the map

	PlayerLose()	Leaves the current battle, and resets the map
	PlayerAttack(word: string)	Resolves the player's attack with word
	PlayerDefend(word: string)	Resolves the player's defence with word
	TriggerEnemyAction()	Causes the enemy to perform an Action, and resolves its effects
	SaveAndQuit()	Write the HubWorld and any saveable members to the game's save file, then return to the main menu
	Quit()	Return to the main menu
	SpawnEnemies()	Creates and places ENEMY_COUNT, enemies at random positions on the map
	StartBattle(enemy: Character)	Begins a battle of the player against the enemy
	ExitBattle()	Leave a battle and reset the stats of all Characters
Relationships	Contains Player	The HubWorld contains the Player, triggers its saving and loading, and communicates with it about the state of the game
	Contains Enemies	The HubWorld contains ENEMY_COUNT enemies, triggers its saving and loading, and communicates with it about the state of the game
	Contains ExitPane	The HubWorld contains the ExitPane, and triggers actions based on received signals
	Contains InfoPane	The HubWorld contains the InfoPane, and triggers actions based on received signals
	Contains BattleUI	The HubWorld contains the BattleUI, and triggers actions based on received signals
	Loads Main Menu	Loads the MainMenu when ExitPane signals to Quit
	Gets Values from Globals	The HubWorld retrieves some constant values from Globals

Class Name	Globals
Description	A collection of constants read only values to be accessed across the game

Attributes	TILE_SIZE	int	The size of a tile in pixels
	SAVE_FILE_PATH	string	The Godot resource path to the savefile
	MovementTime	TimeSpan	Time for the player to move one tile
	MovementTimeSec	double	MovementTime in seconds
	BasePlayerDamage	int	The damage a player does before modifiers
	BaseEnemyDamage	int	The damage an enemy does before modifiers
	BaseDefence	float	The armor value characters start with
	BaseDefenceUbcrease	float	The multiplier to apply when increasing defense
	WinsToSwitchType	int	The number of wins the player should achieve before changing type

Class Name	WordManager		
Description	Loads words from disk, determines relations between words, and provides words that fit certain criteria		
Attributes	Words	IEnumerable<string>	All words contained by the WordManager
	AllMoves	IEnumerable<Move>	Same as Words, but converts each word to a Move
	AllTypes	IEnumerable<string>	All word Types
Operations	GetDefinitions(word:string) : IEnumerable<string>	Get all the definitions of the word	
	GetOfRelation(word:string, relation:Relation) : IEnumerable<string>	Get all words of the relation type to the provided word	

	GetSynonyms(word:string) : IEnumerable<string>	Get synonyms of the provided word
	GetSynonyms(word:WordInfo) : IEnumerable<string>	Get synonyms of the provided word
	GetAntonyms(word:string) : IEnumerable<string>	Get antonyms of the provided word
	GetAntonyms(word:WordInfo) : IEnumerable<string>	Get antonyms of the provided word
	GetTypeAntonyms(type:string) : IEnumerable<string>	Get words that are antonyms of the provided type
	GetTypeSynonyms(type:string) : IEnumerable<string>	Get words that are synonyms of the provided type
	RandomTypes(count:int) : IEnumerable<string>	Return count random types
	RandomMoves(count:int) : IEnumerable<Move>	Return count random moves
	RandomWords(count:int) : IEnumerable<string>	Return count random words
	AreSynonyms(word1:string, word2:string) : bool	Checks if the two words are synonyms
	AreAntonyms(word1:string, word2:string) : bool	Checks if the two words are antonyms
	ClassifyRelation(word:string, type:string) : Relation	Return the relation between the provided word and type
	GetDefinitions(word:string) : IEnumerable<string>	Get the definitions of the provided word
Relationships	Has a Dictionary of WordInfo	Maps the word to its WordInfo

Class Name	WordInfo		
Description	Represents the data that corresponds to a word		
Attributes	Word	string	Word that is represented
	Definitions	string[1..*]	The definitions for the Word
Relationships	Has CategoryInfo	Has 1 or more CategoryInfo, which represent the types of the word	

Class Name	CategoryInfo		
Description	Represents a Type for a Word		
Attributes	Type	string	Type that is represented
	Relation	Relation	How the word is related to the Type

Relationships	Owned by WordInfo	A WordInfo can have 1 or more CategoryInfo to represent its relations to different types

Class Name	BattleUI		
Description	The UserInterface for the battle mechanic		
Attributes	State	BattleUIState	The current state of the BattleUI
Operations	WaitForEnemyMove()		The BattleUI should prohibit player interaction until the enemy's turn is resolved
	PlayerTurn()		Indicate that the player can interact
	Close()		Reset and Close the BattleUI
Relationships	Owns DefinitionPopUp		The BattleUI owns a DefinitionPopUp that it shows, hides, and updates based on signals from other members
	Owns MovePopUp		The BattleUI owns a MovePopUp that it shows, hides, and updates based on signals from other members
	Owns MovePane		The BattleUI owns a MovePane that it shows, hides, and updates based on signals from other members
	Owns HealthBar		The BattleUI owns a HealthBar for the player and the opponent. It shows, hides, and updates these based on signals from other members
	Owns LogPane		The BattleUI owns a LogPane for the player and the opponent. It shows, hides, and updates them based on signals from other members
	Contained by HubWorld		The HubWorld contains the BattleUI, and shows, hides, and updates it as needed

Class Name	InfoPane	
Description	The Player Inventory	
Operations	Display()	Show the InfoPane
	ClearWords()	Remove all words from the UI
	DisplayWordInfo(word string)	Display the details of the provided word
	AddWord(word: string)	Add the word to the UI
Relationships	Contained by HubWorld	The HubWorld contains the InfoPane, and shows, hides, and updates it as needed

Class Name	DefinitionPopUp

Description	A pop up to view the definition of a word		
Attributes	Text	string	The definitions
	Word	string	The word being viewed
Operations	Close()	Close the popup	
Relationships	Owned by BattleUI	The BattleUI owns a DefinitionPopUp and shows, hides, and updates it as needed	

Class Name	MovePopUp		
Description	The Popup to select how to use a move		
Attributes	Text	String	The word that the move uses
Operations	Close()		Close the pop up
Relationships	Owned by BattleUI		The BattleUI owns a MovePopUp and shows, hides, and updates it as needed. The MovePopUp signals to the UI how to use the select word

Class Name	MovePane	
Description	The list of Words to select for a battle turn	
Operations	ClearMoves()	Clear all the moves from the MovePane
	AddMove(move: Move): Card	Adds a Move to the UI, and returns the resulting Card
Relationships	Owned by BattleUI	The BattleUI owns a MovePane and shows, hides, and updates it as needed. The MovePane signals to the UI about what word to use
	Owns Cards	Owns a Card for each Move

Class Name	Card		
Description	A card to display and select a word during a battle		
Attributes	word	String	The word the Card represents
Operations	SetLabel(text: String)	Update the word to text	
Relationships	Owned by MovePane	Owned by a MovePane, which constrains the location of the card. The card signals to the move pane when its buttons are pushed	

Class Name	HealthBar	
Description	Displays the health of the associated Character as a percent	
Operations	SetHealth(value: double)	Updates the health value of the bar
Relationships	Owned by BattleUI	The BattleUI owns a HealthBar for the player and opponent. The BattleUI updates the Health as needed

Class Name	LogPane		
Description	The log of a battle for a given Character		
Attributes	log	Queue<object>	All the items that are to be displayed
Operations	LogMove(move:string, type:MoveType)	Add the move to the log	
	Log(item:T)	Adds the provided item to the log	
	Clear()	Clears the log	
Rekationships	Owned by BattleUI	A BattleUI owns two LogPanes, for the player and opponent. The BattleUI updates the log as event occur	

Class Name	ExitPane	
Description	The UI Element to exit the game to the menu	
Relationships	Contained by HubWorld	A HubWorld contains one ExitPane. The HubWorld receives signals from the ExitPane
	Saves and Quits	Triggers the HubWorld to Save the game state, and or Quit to the main menu

Class Name	MainMenu	
Description	The Game's main menu	
Relationships	Loads the HubWorld	Loads the HubWorld when the start button is pressed

Class Name	Target	
Description	An enum to represent the target of an Action	
Attributes	Player	Targeting the player
	Enemy	Targeting the enemy

Class Name	BattleUIState	
Description	An enum to represent the state of the BattleUI	
Attributes	SelectWord	The player can select a word
	ViewDefinition	The player is viewing the definition of a word
	SelectMove	The player is selecting the Action to perform with a word
	EnemyMove	The enemy is making a move, so block all UI interactions

Class Name	MoveType
Description	An enum to represent the type of an Action

Attributes	Attack	The Action is an attack
	Defend	The Action is a defence

Class Name	GameState	
Description	Any enum to represent the state of the HubWorld	
Attributes	Hub	The player can walk around the map
	Battle	A battle is active
	Info	The player is in the inventory
	Exit	The player has the ExitPane open

Class Name	Relation	
Description	An enum to represent the relation between words or a word and a type	
Attributes	None	There is no special relation
	Synonym	There is a synonym relationship
	Antonym	There is an antonym relationship

Class Name	AttackStatus	
Description	An enum representing the status of an attack	
Attributes	Effective	The attack was effective
	Normal	The attack was normal
	Ineffective	The attack was ineffective

4.3 Sequence Diagrams

These are sequence diagrams that are representitive of two different processes that are a part of Leximon. The first is the process for a enemy turn during a battle and the other is the process for a player turn during the battle. The subsections themselves will also contain the sequence diagram itself as well as a description describing the process.

4.3.1 Enemy Turn

This sequence diagram shows the process of an enemy's turn during a battle. First the player enters a battle by interacting with an enemy. Then a random action for the enemy is selected. This move is compared against the player to determine the damage to apply. This damage is then apply to the player. If the player's HP reaches 0, the game ends with the player showing a loss and both combatants being returned to the hub world with their HP reset.

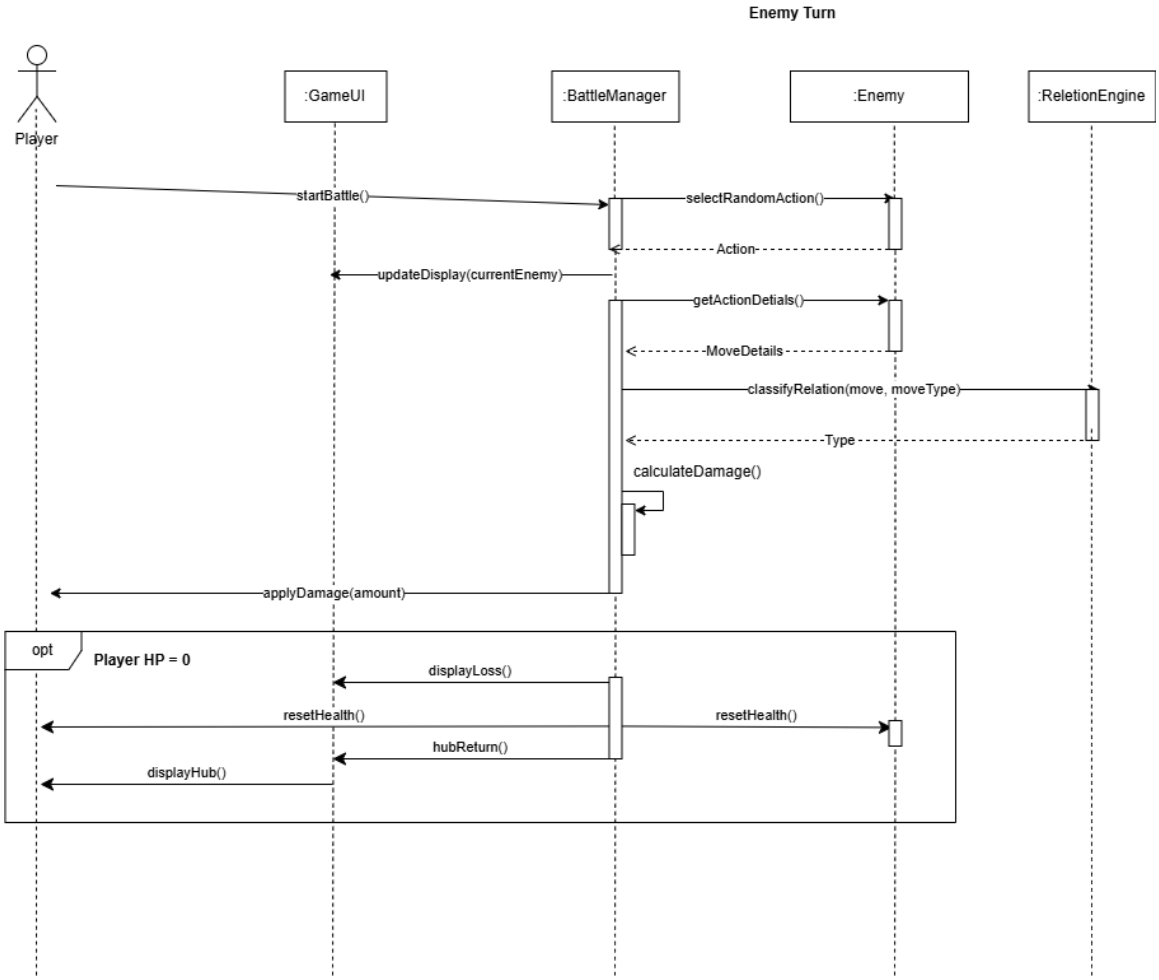

Figure 5: Enemy Turn Sequence Diagram

4.3.2 Player Turn

This sequence shows the the process the player goes through on their turn in a battle. First they select what action they want to use. Then the system determines the types of the player's move and the enemy. These are compared and a damage multiplier is derived from their relationship. Then the ddamage is applied to the enemy. After the damage is applied the UI is updated. Lastly, if the enemy's HP dropped to 0, the battle ends, and the player is granted new moves and returned to the hub world.

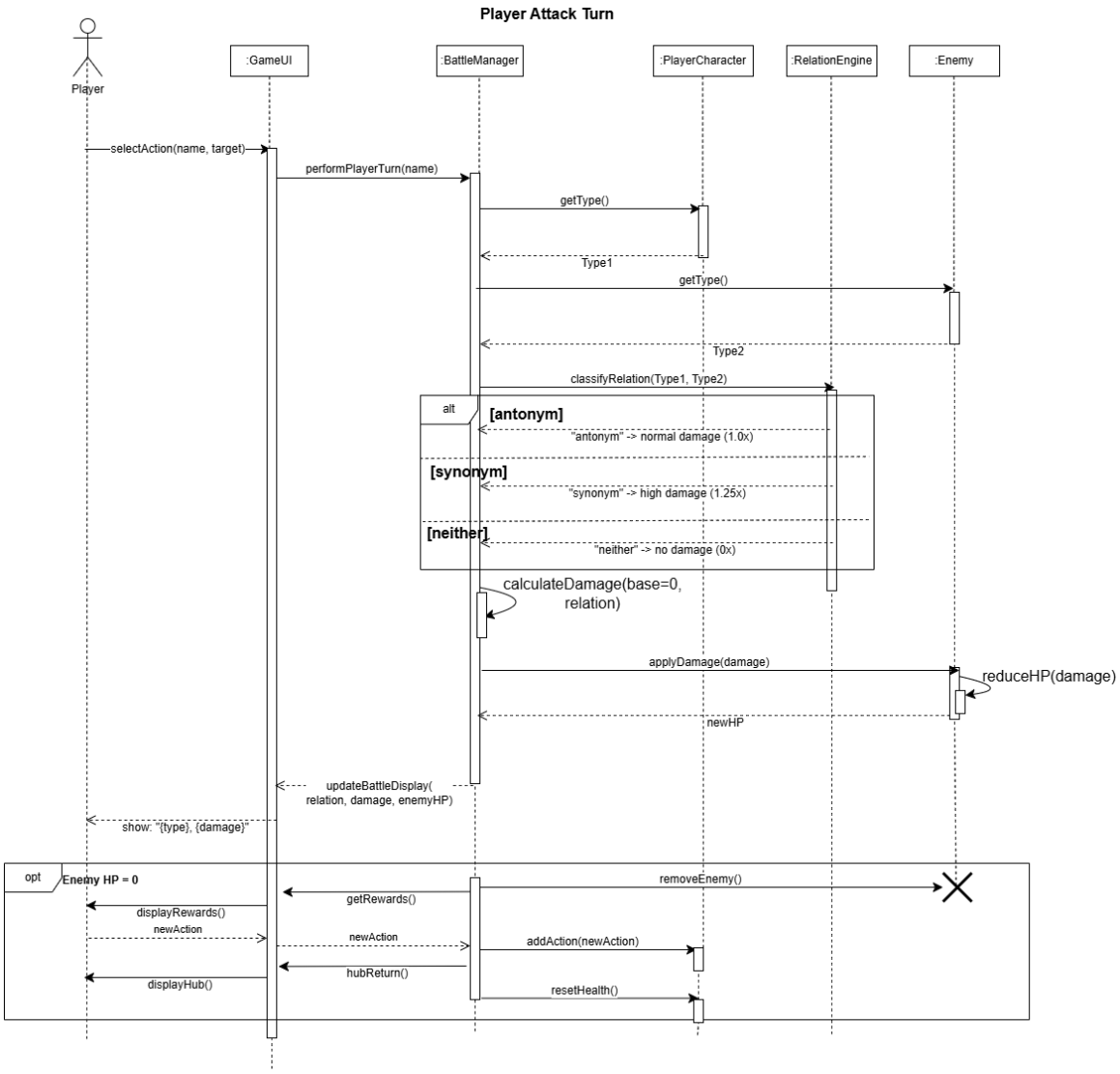

Figure 6: Player Turn Sequence Diagram

4.4 State Diagram

Below is a state diagram that outlines the different states the player may come across throughout the gameplay. The player begins in the Main Menu. This is where they

are given the choice to either begin a new game, load a previous game, or exit the game. Once the game is started or loaded, the player transitions into the Hub World. This acts as the central point of navigation and exploration. In the Hub World, the player has several options. The main task is to progress through the game by encountering enemies. This transitions the player into the Battle state. The battle sequence determines whether the player wins or loses. A win will return the player to the Hub World to continue exploring/preparing for future encounters. Meanwhile a loss will prompt the player to retry or return to the main menu depending on their choice. The player may also interact with menus and UI screens. These are things such as viewing character information or accessing other gameplay interfaces. These screens briefly interrupt movement before returning the player back to the Hub World once closed. On top of that the player may choose to pause or quit the game at various points. If the player opens the exit screen, they may save and return to the main menu, just save, or press escape to cancel.

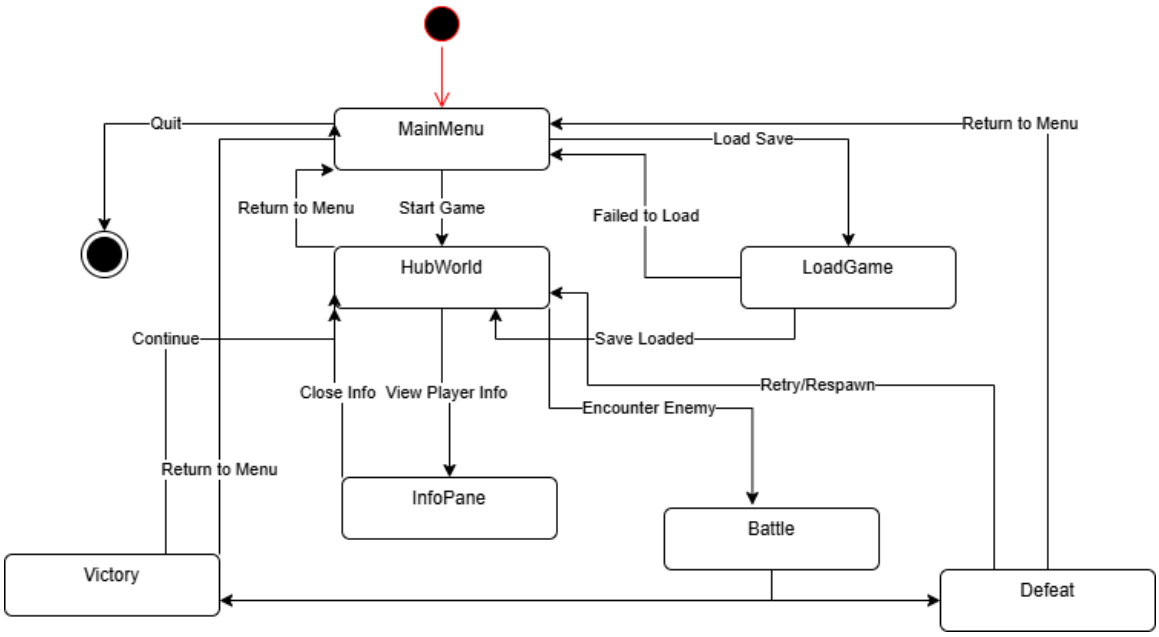

Figure 7: State Diagram

5 Prototype

The prototype of Leximon will include a main menu. It will also have a small portion of the game. The player will have the ability to move around a basic overworld. This overworld will contain limited environmental features and interactable elements. Core gameplay aspects such as encountering enemies, initiating battles, and managing a basic inventory will be present. The battle system will allow the player to perform Actions. It will also let the player see the results of combat. However balancing, enemy variety, and full visual polish will not yet be finalized. This prototype demon-

strates the foundation of the battle system, allowing players to perform attacks and defenses by identifying synonyms and antonyms. Exploration of the world exists in this prototype only so far as it enables the player to engage with the battle system, and progression exists only in the player's expanding vocabulary.

5.1 How to Run Prototype

To run the prototype you must be using a Windows computer. Download the most recent release from the project GitHub at `https://github.com/Leximon-SWE-2025/Leximon/releases`. Unzip the file and run the leximon.exe executable. If there is a popup that has the title "Windows protected your PC", click "More info" at the bottom of the explanation, and then click the "Run anyway" button at the bottom of the popup. If you want to move the executable, you must also have all the files in the zip in the same directory as the executable.

The controls for the game is a follows

Movement: Use the *WASD* or arrow keys to move.

Exit Game: Use the <escape> key to bring up the exit menu, and click the button to exit the game

View Inventory: Use the *f* key to open in Inventory

Interact with enemy: Use the *e* key when close to an enemy to battle them

Interact with UI: Use the mouse to place the cursor over a UI element and then click it with the left mouse button

5.2 Sample Scenarios

Figure 8: The main menu of the game. Shown when the game is started. From here the player can enter the game, view the controls, visit the game's website or quit

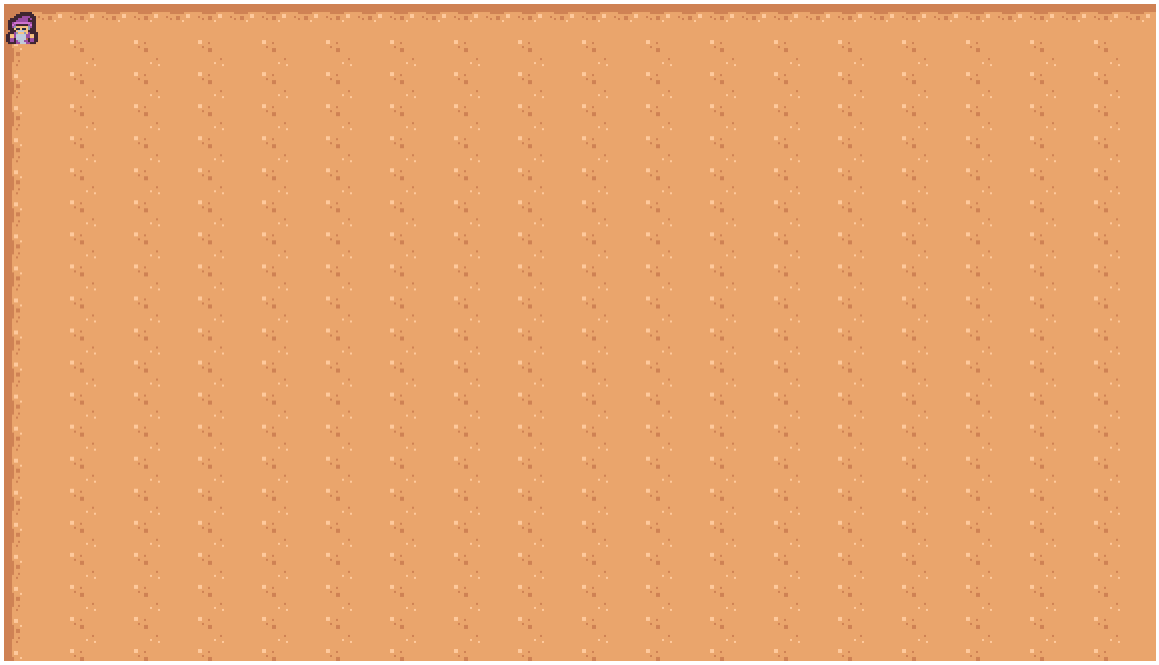

Figure 9: The player starts a new game in the top left corner of the map. They might have to walk around to find enemies

Figure 10: The player can view the dictionary of words they can use, including definitions and their assoicated types. The player can also view their type and health in the menu

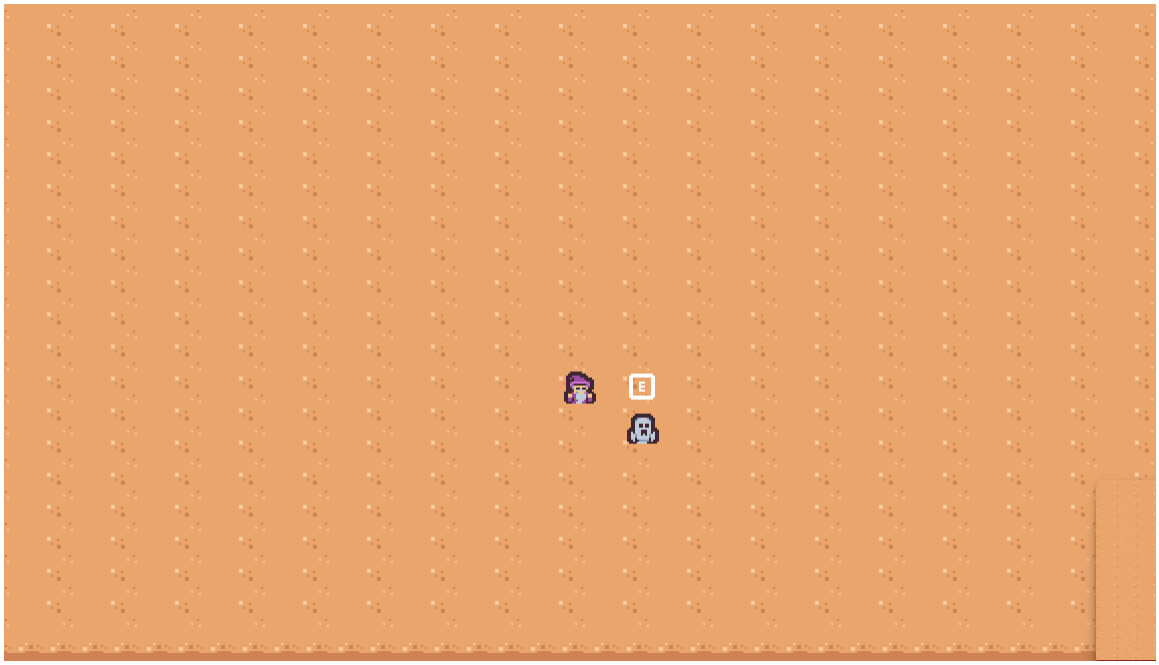

Figure 11: When the player finds an enemy, they can approach it and interact to enter a battle

Figure 12: In a battle, the player can select a word, and choose to attack or defend with it

Figure 13: After the player's turn, the enemy gets one

Figure 14: If the player does not know what a word means, they can view its definition

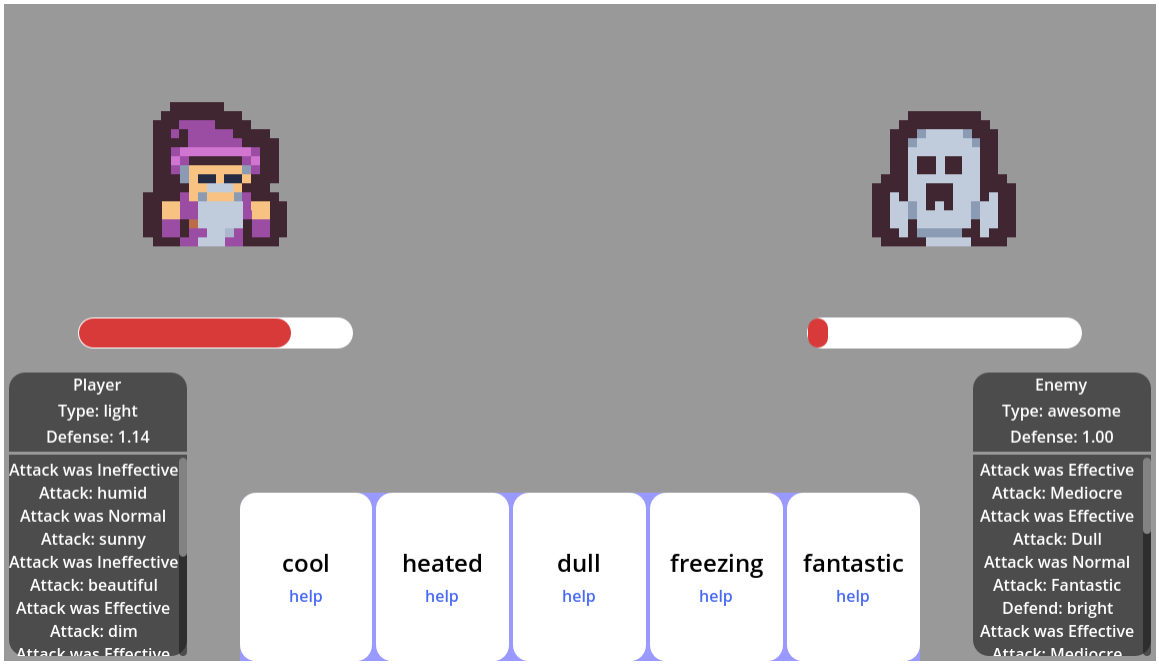

Figure 15: The battle ends when either combatant runs out of health

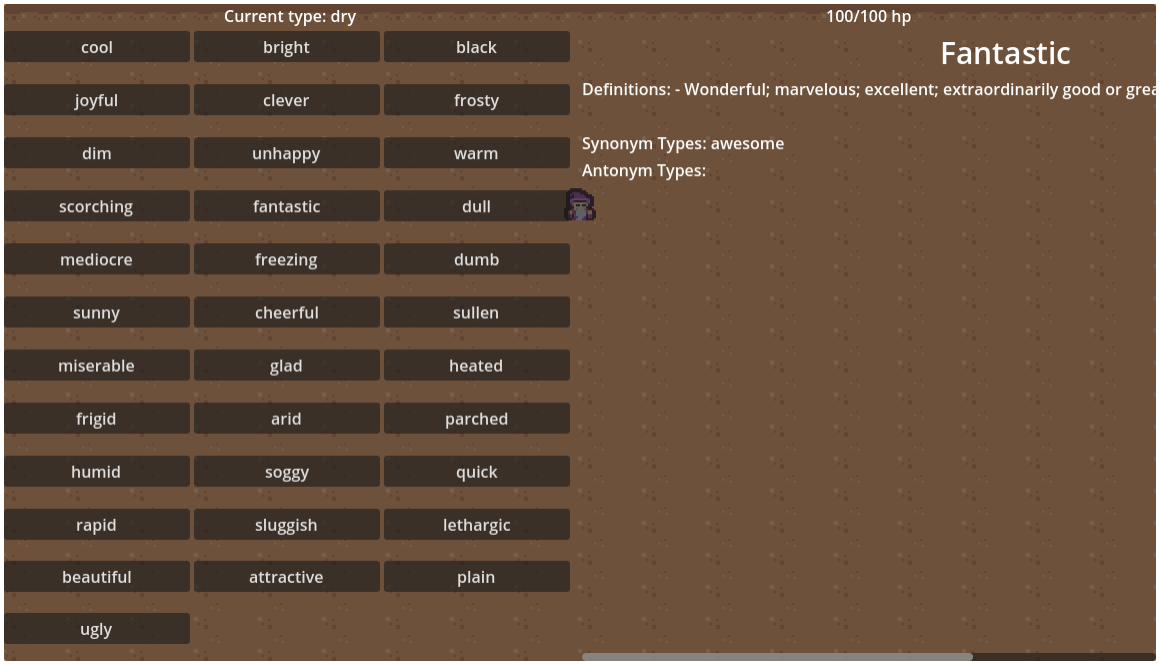

Figure 16: After the player wins 3 battles, their type changes

6 References

[1] Piotr Czapla. *Quick way to create a list of values in C#?* https://stackoverflow.com/questions/723211/quick-way-to-create-a-list-of-values-in-c. 2009.

[2] Jader Dias. *Is there any way to call the parent version of an overridden method? (C# .NET).* https://stackoverflow.com/questions/438939/is-there-any-way-to-call-the-parent-version-of-an-overridden-method-c-net. 2009.

[3] C# Docs. *How to read JSON as .NET objects (deserialize).* https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/deserialization.

[4] C# Docs. *Records (C# Reference).* https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record.

[5] C# Docs. *User-defined explicit and implicit conversion operators.* https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/user-defined-conversion-operators.

[6] C# Docs. *Using the StringBuilder Class in .NET.* https://learn.microsoft.com/en-us/dotnet/standard/base-types/stringbuilder.

[7] Godot Engine Documentation. *FileAccess.* https://docs.godotengine.org/en/stable/classes/class_fileaccess.html.

[8] Godot Engine Documentation. *Handling quit requests.* https://docs.godotengine.org/en/stable/tutorials/inputs/handling_quit_requests.html.

[9] Godot Engine Documentation. *Input examples - Godot Engine documentation.* https://docs.godotengine.org/en/stable/tutorials/inputs/input_examples.html.

[10] Godot Engine Documentation. *Saving games.* https://docs.godotengine.org/en/stable/tutorials/io/saving_games.html.

[11] Massachusetts Department of Elementary and Secondary Education. *English Language Arts and Literacy Massachusetts Curriculum Framework.* https://www.doe.mass.edu/frameworks/ela/2017-06.pdf. 2017.

[12] GDQuest. *Setting up pixel art graphics in Godot 4.* https://www.gdquest.com/library/pixel_art_setup_godot4/.

[13] kingzonas. *How do I fix error: "No export template found at the expected path: [path]"?* https://forum.godotengine.org/t/how-do-i-fix-error-no-export-template-found-at-the-expected-path-path/1982. 2023.

[14] Kron. *How to Open Web Pages from Godot.* https://www.youtube.com/watch?v=3oWiAF_UbEA. 2021.

[15] *Overview of Renderers.* https://docs.godotengine.org/en/stable/tutorials/rendering/renderers.html.

[16] Raul Santos. *Current state of C# platform support in Godot 4.2*. Tech. rep. The Godot Foundation, 2024. URL: https://godotengine.org/article/platform-state-in-csharp-for-godot-4-2/#web.

[17] Platon Supranovich et al. *Leximon*. https://leximon-swe-2025.github.io/Website/. 2025.

[18] u/_P0PCAT_012345. *Grid Container with Scroll*. https://www.reddit.com/r/godot/comments/mhnpgl/grid_container_with_scroll/.

[19] u/Gulferamus. *BoxContainers seem to squish child nodes to nothing*. https://www.reddit.com/r/godot/comments/1ag5agj/boxcontainers_seem_to_squish_child_nodes_to/.

[20] user192472. *Is there a standard "never returns" attribute for C# functions?* https://stackoverflow.com/questions/1999181/is-there-a-standard-never-returns-attribute-for-c-sharp-functions. 2010.

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.