Software Requirements Specification (SRS)

Formula Fighters

Team:	**Kyran Chan, Tameron Honnellio, Anson Lu, Will Shahbazian, Forrest Boyer**
Authors:	**Kyran Chan, Tameron Honnellio, Anson Lu, Will Shahbazian, Forrest Boyer**
Customer:	**5th Graders**
Instructor:	**Professors James Daly**

1 Introduction

The purpose of this document is to outline the Software Requirements Specifications (SRS) for “Formula Fighters”, a mathematics-based deck building game designed for 4th graders to enhance their learning development.

The following is an outline of the topics that will be covered in this document: the product’s perspective and function, user characteristics and project constraints, assumptions and dependencies, project requirements, prototypes, how to operate the prototype, sample scenarios, and references.

1.1 Purpose

The purpose of a SRS document is to completely describe and explain the purpose, features and functions of this proposed project video game.

This document is intended for developers, project managers, and any who may be of concern to this project.

1.2 Scope

This project will produce the following software product, a video game called “Formula Fighters”.

The applicable domain for this software application is an intersection between education and entertainment. The intended target audience will be students in the 4th grade, who are learning the aspects of basic math operations: addition, subtraction, multiplication, and division with whole number integers. The game will incorporate an educational component, aligning with the Massachusetts Department of Education and Common Core standards. The game must also be entertaining, in order to capture the attention and interest of students. This domain balances considerations for target audience’s age, educational standards, and the need for an engaging and effective learning experience.

This software product will consist of an entertaining video game. It will also feature an educational component as the main purpose. This product will allow for human interaction through a user interface to facilitate the educational component. This software product will not feature any overly complex mechanics, whether in mechanical skill or educational concepts, that will exceed the cognitive or physical abilities of the intended audience.

1.3 Definitions, acronyms, and abbreviations

Math operators - Addition, multiplication, subtraction, division.

Token - Individual integer numbers/math operators/items.

Deck - A collection of all of the player's current playable tokens.

Hand - Set of tokens that the player has access to on their turn.

Equation - Set of tokens built by the player to attack or defend.

Item - Tokens which alter the game state when played from hand (e.g. multiply player attack value *2). This persists until the end of combat or until the next offense phase.

Passive - A permanent effect on the player which enhances certain actions (e.g. +5 to player defensive value). This persists between levels.

Level - A room containing an enemy encounter, or item shop.

Entity - Player or enemy game object.

Hitpoints - Amount of health an entity has. Each successful attack depletes hit points, if this value reaches zero the player or enemy dies.

Offensive phase - A state where the player will build an equation to attack the enemy. Attack values will be integers.

Defensive phase - A state where the player will build an equation to defend from an enemy attack. Defense values will be integers.

Combat - An enemy encounter consisting of the player and enemy switching between offense and defensive phases until the player wins (depletes enemy's hit points), or loses (loses all hit points).

Discard Pile - A separate deck of tokens used during combat. After a token is used, it is sent to the discard pile where it cannot be played.

Damage Differential - The resulting integer damage value of an attack after it has been subtracted by the defense value.

Inventory - The total card storage, including every token, in the player’s possession. The player’s deck is built from their inventory.

Valid Equation - A valid math equation follows the standard rules and conventions of mathematical notation and syntax.

1.4 Organization

The next chapter, Overall Description, will speak about the general context and overview for the product. It will discuss the product's perspective, its function, user characteristics, constraints, assumptions and dependencies.

Chapters 3 and 4 are written primarily for the developers, they describe the technical details of the product and its features and functions. Chapter 3 addresses the specific requirements. That is, a numbered list of requirements in a hierarchical setup scheme, from general at the top to more specifics toward the sublevels. Chapter 4 addresses the modeling requirements. A Use Case diagram, data dictionary, and sequence diagrams for the product are constructed. When combined, both sections describe the software product in its entirety, and serve to bridge the gap between the application and machine domain.

Chapter 5 describes a prototype of the product we will be creating. A general outline of the system's functionality. A short "how to run the prototype" explanation and sample scenarios are also given.

Lastly, Chapter 6 lists all documents referenced in this SRS.

2 Overall Description

This section of the SRS will focus on the high-level view of the product. This includes context for its creation, major functionality, our intended audience, constraints, assumptions/dependencies, and stretch goals.

2.1 Product Perspective

Formula Fighters is a game designed to bring the worlds of classic deck building games, such as Dominion or Slay the Spire, together with an educational component. The idea is to be both educational and entertaining at the same time, giving users a fun experience while also having them learn. The game aims to be playable with or without prior math skills, with users hopefully picking some up along the way. Due to the level of math involved in the game, the general target audience is late elementary school students, but the game should be enjoyable for all ages.

The interface will be kept simple, so our intended audience will have an easy time learning and interacting with it. User interfaces will primarily consist of keyboard and mouse input. Formula Fighters will be constrained via the Godot game engine, which is the foundational engine we plan on using to create our 2D game.

2.2 Product Functions

The major functionality of the software will be as follows. A main menu to allow players to start or quit the game, requirement details for this functionality can be seen in section 3, 1.1. A random map will be generated when the game is started and displayed to the user. The map consists of a horizontal series of nodes connected by dotted lines. The user will be able to linearly progress through the map as they select complete levels via a level select menu. Requirement details for the map feature can be seen in section 3, 1.2. A combat system will begin once the user selects a level. This will involve attack and defense phases that will repeat until either the user or the enemy have depleted their health. The user will build equations from cards in their hand to reach a high attack or defense number depending on the phase. At the beginning of each offense phase, the software will draw a new hand of cards for the player to use. More details for the combat requirements can be found in section 3, 1.3. Finally, if the user wins the combat encounter, by depleting the health of the enemy first, they will be prompted to choose one of three new cards to add to their deck. Requirements for the reward functionality can be seen in section 3, 2.

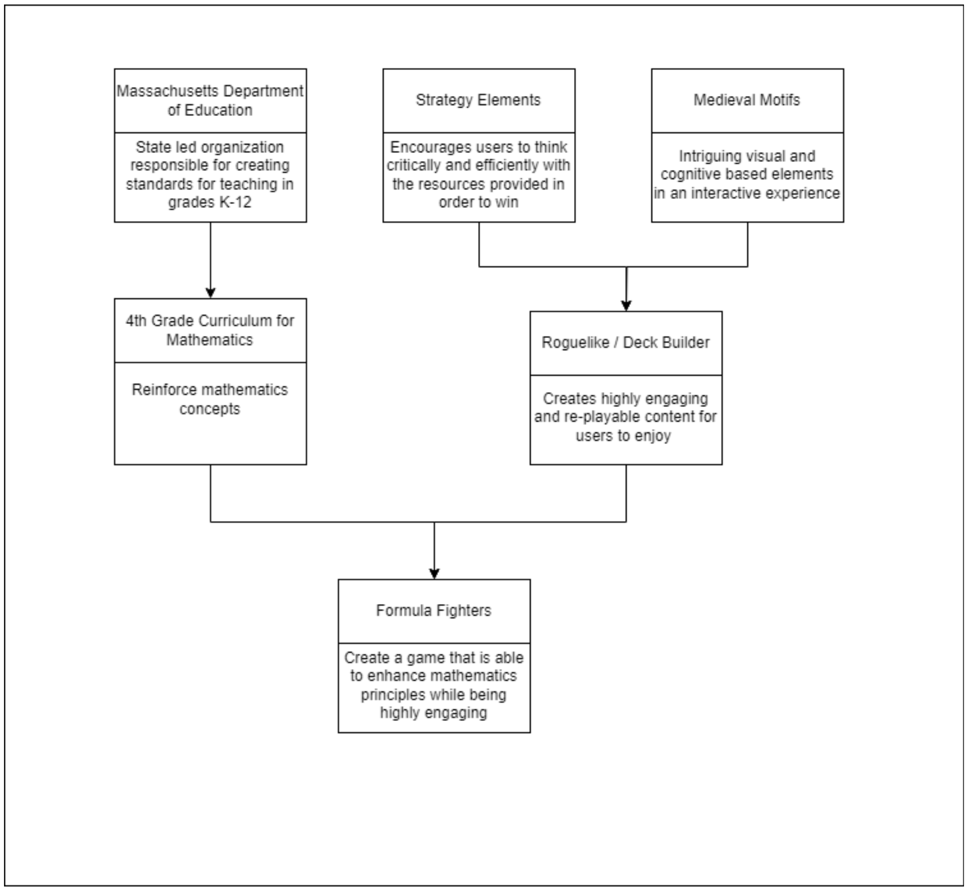
Massachusetts Department of Education
State led organization responsible for creating standards for teaching in grades K-12
Strategy Elements
Encourages users to think critically and efficiently with the resources provided in order to win
Medieval Motifs
Intriguing visual and cognitive based elements in an interactive experience
4th Grade Curriculum for Mathematics
Reinforce mathematics concepts
Roguelike / Deck Builder
Creates highly engaging and re-playable content for users to enjoy
Formula Fighters
Create a game that is able to enhance mathematics principles while being highly engaging

Fig. 1, Goal Diagram

Figure 1, pictured above outlines our high-level goals with this product. We intend to build a bridge between mathematics concepts for 4th graders set by the MA Department Of Education, and highly engaging, re-playable roguelike deck builder games.

2.3 User Characteristics

The intended audience for this game consists of late elementary children in 4th grade who are familiar with the four basic operations (addition, subtraction, multiplication, and division) and their ordering. Ideally, the users should be somewhat experienced with interactions between the four operations and building equations with them so the game can deepen their understanding.

Users should also be familiar with computers, card games, and computer games. Formula Fighters requires some understanding of the concept of hands, decks, and level progression. Additionally, a mouse or trackpad is required to play Formula Fighters. Users also require some reading skills to understand the tutorial, but with simple wording, the requirements should be minimal.

2.4 Constraints

1. Users may not be able to run the Vulkan renderer
2. Users may not be able to download and extract the project folder
3. Project files may be corrupted or tampered with leading to incorrect functionality
4. The Godot game engine may have buggy implementation of certain functionality
5. The Godot game engine does not currently support .NET exports of a project
    - 5.1. This means that the product can not run in a webpage
6. The internal data representing the user's inventory may exceed the memory limit on certain machines
7. Users may select cards which result in an invalid equation

2.5 Assumptions and Dependencies

We are developing this software with the assumption that users will have Windows, macOS, or Linux based machines. It is necessary that users are able to download the zipped project folder, extract it, and run the executable for their respective machine. Hardware requirements for this project are that the machine must be able to run the Vulkan renderer. The user must also have access to some form of mouse interaction in order to interact with the game environment.

2.6 Apportioning of Requirements

1. Based on negotiations with customers, requirements that are determined to be beyond the scope of the current project and may be addressed in future versions/releases.

    - 1.1. Remainder Representation in division
        - 1.1.1. To avoid confusion
    - 1.2. Higher Difficulties that involve more complex math
        - 1.2.1. Broaden audience
    - 1.3. Passive effects like buffs and debuffs
        - 1.3.1. More nuance and entertainment factor
        - 1.3.2. e.g. Even numbers deal half damage but multiples of 3 deal triple
    - 1.4. Additional Levels
        - 1.4.1. Extends runs so reruns are more enjoyable
        - 1.4.2. Multiple floors
    - 1.5. Additional Enemies
        - 1.5.1. Bosses

        1.5.2. Enemies with their own passive effects
        1.5.3. Increases replayability
    1.6. Enemies playing cards instead of having set values
        1.6.1. Makes game more variable and entertaining
        1.6.2. Makes enemies feel more alive
    1.7. Saving Progress
        1.7.1. Requires too much resources

3 Specific Requirements

Requirements:

1. Gameplay
    1.1. There will be a start menu and quit option
        1.1.1. The option to start the game will be present on start menu
        1.1.2. The quit option shall be present on the screen at all times
    1.2. Game play will consist of a series separate levels to be beaten by the player
        1.2.1. There must be a level traversal map consisting of branching paths, comprised of selectable levels to play
            1.2.1.1. The levels shall be arranged via a randomly generated map
        1.2.2. The player will have the ability to choose when to enter the first level, allowing themself to familiarize themself with their tokens and the map
    1.3. Combat must function as follows
        1.3.1. Token Management must function as follows
            1.3.1.1. Player's have a passive operator that they are able to use every turn that persists through turns
            1.3.1.2. At the start of each offense phase the player must discard the remaining tokens in their hand and draw a new hand
                1.3.1.2.1. The player must have some max hand size
            1.3.1.3. All cards draws must come from the top of the deck
                1.3.1.3.1. The deck will be generated at the start of combat based on the player's collected cards
            1.3.1.4. After a token is used, it must be sent to the discard pile
                1.3.1.4.1. The discard pile shall be used to refill the deck if the deck becomes empty
            1.3.1.5. Each equation shall display a maximum token count, and will only accept equations with less than or equal to the correct token count
            1.3.1.6. There must be a method of returning all tokens from an equation to the hand before submitting the attack
            1.3.1.7. Item tokens played shall alter the game state
        1.3.2. Enemies shall have default
            1.3.2.1. Hitpoint values
            1.3.2.2. Attack values

1.3.2.3. Defense values

1.3.3. The player and enemy must switch off between offense phase and defense phase until the player or enemy's hp has depleted

1.3.3.1. Offense Phase

1.3.3.1.1. The player will draw a new hand

1.3.3.1.2. The enemy will display a defense value

1.3.3.1.3. Player must have the opportunity to build an equation from tokens in hand

1.3.3.1.4. Player may use item tokens to alter the equation/game state

1.3.3.1.5. Player's equation must evaluate to a single number

1.3.3.1.6. The number will always evaluate to an integer to maintain an elementary education level

1.3.3.1.7. The resulting damage differential, after the solved defense equation is subtracted from the enemy's projected attack value, will be dealt to the enemy

1.3.3.2. Defense Phase

1.3.3.2.1. The enemy will display an attack value

1.3.3.2.2. Player must have the opportunity to build an equation from tokens in hand

1.3.3.2.3. The resulting damage differential, after the solved defense equation is subtracted from the enemy's projected attack value, will be dealt to the player

1.3.3.3. The player will have the ability to end their turn at any point during either phase

2. Progression

2.1. After the player beats a level there must be some form of strength progression

2.1.1. The player will be able to grow their deck size

2.1.1.1. The player's deck outside of combat will be tracked through their inventory

2.1.1.1.1. The inventory will be used to generate a deck at the beginning of combat

2.1.1.2. Both number and operator tokens will be offered to be added to the player's deck throughout the game

2.1.1.3. Reward options will be generated after the player beats a level

2.1.2. The player will occasionally be granted items to obtain passives

2.1.3. The player will occasionally gain extra max hitpoints

2.2. Player shall be able to choose the next level after receiving their rewards

2.2.1. The player must be able to move an indicator to navigate the level selection

2.3. As levels progress, enemies stats increase:

2.3.1. Hitpoints

2.3.2. Attack value

2.3.3. Defense value

3. Backend
    3.1. During the attack and defense phases of combat, the software must perform a comparison between the player's attack value and the enemy's defense value and vice versa
        3.1.1. If the damage differential is <= 0, no damage is taken
        3.1.2. If the damage differential is > 0, the entity defending takes that amount of damage
    3.2. Software shall allow the game to switch states from menu, offense, defense, rewards, and level selection

4 Modeling Requirements

Use Case Diagram

This use case diagram shows the general formula of the game, moving down chronologically.

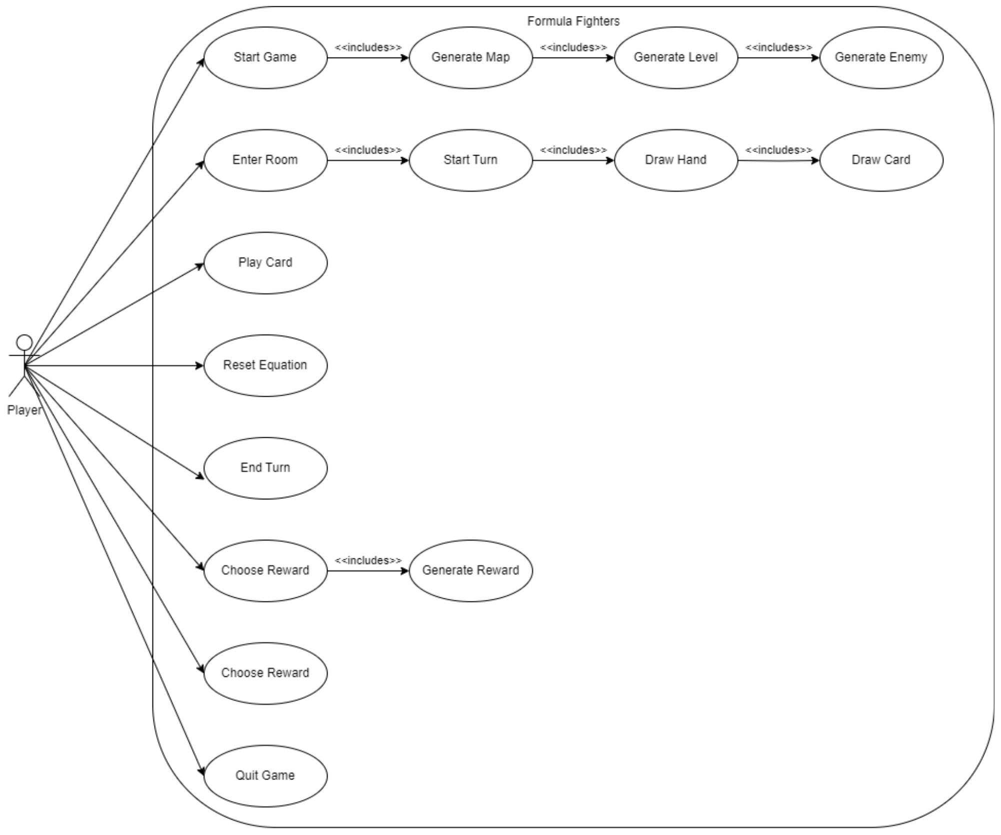

Fig. 2, Use Case Diagram

Use Case Name:	Start Game
Actors:	Player
Description:	The Player chooses to begin a Run
Type:	Primary + Essential
Includes:	Generate Map
Extends:	None
Cross-refs:	Requirement(s): 1.1, 1.1.1
Uses cases:	The player pressed the button to start the game

Use Case Name:	Generate Map
Actors:	None
Description:	The map is randomly generated
Type:	Secondary
Includes:	Generate Level
Extends:	None
Cross-refs:	Requirement(s): 1.2.1, 1.2.1.1
Uses cases:	Done automatically on run start

Use Case Name:	Generate Level
Actors:	None
Description:	Individual levels within the map are generated
Type:	Secondary
Includes:	Generate Enemy
Extends:	None
Cross-refs:	Requirement(s): 1.2
Uses cases:	Done as map generates

Use Case Name:	Generate Enemy
Actors:	None
Description:	Enemies for the player to fight are generated
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.2, 1.3.2.1, 1.3.2.2, 1.3.2.3
Uses cases:	Done when levels are generated

Use Case Name:	Enter Room
Actors:	Player
Description:	The player enters a level
Type:	Primary + Essential
Includes:	Start Turn
Extends:	None
Cross-refs:	Requirement(s): 1.2.2
Uses cases:	The player chooses to enter the next level

Use Case Name:	Start Turn
Actors:	None
Description:	A turn of combat is started
Type:	Primary
Includes:	Draw Hand
Extends:	None
Cross-refs:	Requirement(s): 1.3.1.2, 1.3.3
Uses cases:	The player chooses to start combat

Use Case Name:	Draw Hand
Actors:	None
Description:	The player’s hand is filled from the deck at the start of the turn
Type:	Primary
Includes:	Draw Card
Extends:	None
Cross-refs:	Requirement(s): 1.3.1.2, 1.3.3.1.1
Uses cases:	Done a new offense turn has started

Use Case Name:	Draw Card
Actors:	None
Description:	The player’s hand gains a card from the deck
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.1.3
Uses cases:	Done a new offense turn has started

Use Case Name:	Play Card
Actors:	Player
Description:	A card moves from the player’s hand to the equation
Type:	Primary + Essential
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.3.1.3, 1.3.3.1.4, 1.3.3.2.2
Uses cases:	The player plays a card into the equation

Use Case Name:	Reset Equation
Actors:	Player
Description:	Clears the equation and moves cards back to the hand
Type:	Primary + Essential
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.1.6
Uses cases:	The player presses the button to reset the equation

Use Case Name:	End Turn
Actors:	Player
Description:	Resolves the turn and does damage
Type:	Primary + Essential
Includes:	Solve Equation, Deal Damage
Extends:	None
Cross-refs:	Requirement(s): 1.3.3.3
Uses cases:	The player presses the end turn button

Use Case Name:	Solve Equation
Actors:	None
Description:	Calculates the total of the equation
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.3.1.5, 1.3.3.1.6
Uses cases:	Done when the turn is ended

Use Case Name:	Deal Damage
Actors:	None
Description:	Deals damage to the entity that is currently defending
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.3.3.1.7, 1.3.3.2.3
Uses cases:	Done when the turn is ended

Use Case Name:	Choose Reward
Actors:	Player
Description:	Adds the player's choice of card to their deck from some generated card options
Type:	Primary
Includes:	Generate Reward
Extends:	None
Cross-refs:	Requirement(s): 2.1.1, 2.1.1.2
Uses cases:	Done when a level is beaten and the player chooses a card

Use Case Name:	Generate Reward
Actors:	None
Description:	Generates cards for the player to choose from as rewards
Type:	Secondary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 2.1.1.3
Uses cases:	Done when a level is beaten

Use Case Name:	Choose Path
Actors:	Player
Description:	Selects the next level
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 2.2, 2.2.1
Uses cases:	Done when a level is beaten

Use Case Name:	Quit Game
Actors:	None
Description:	Exits and closes the game
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirement(s): 1.1, 1.1.2
Uses cases:	Done when a level is beaten

Class Diagram:

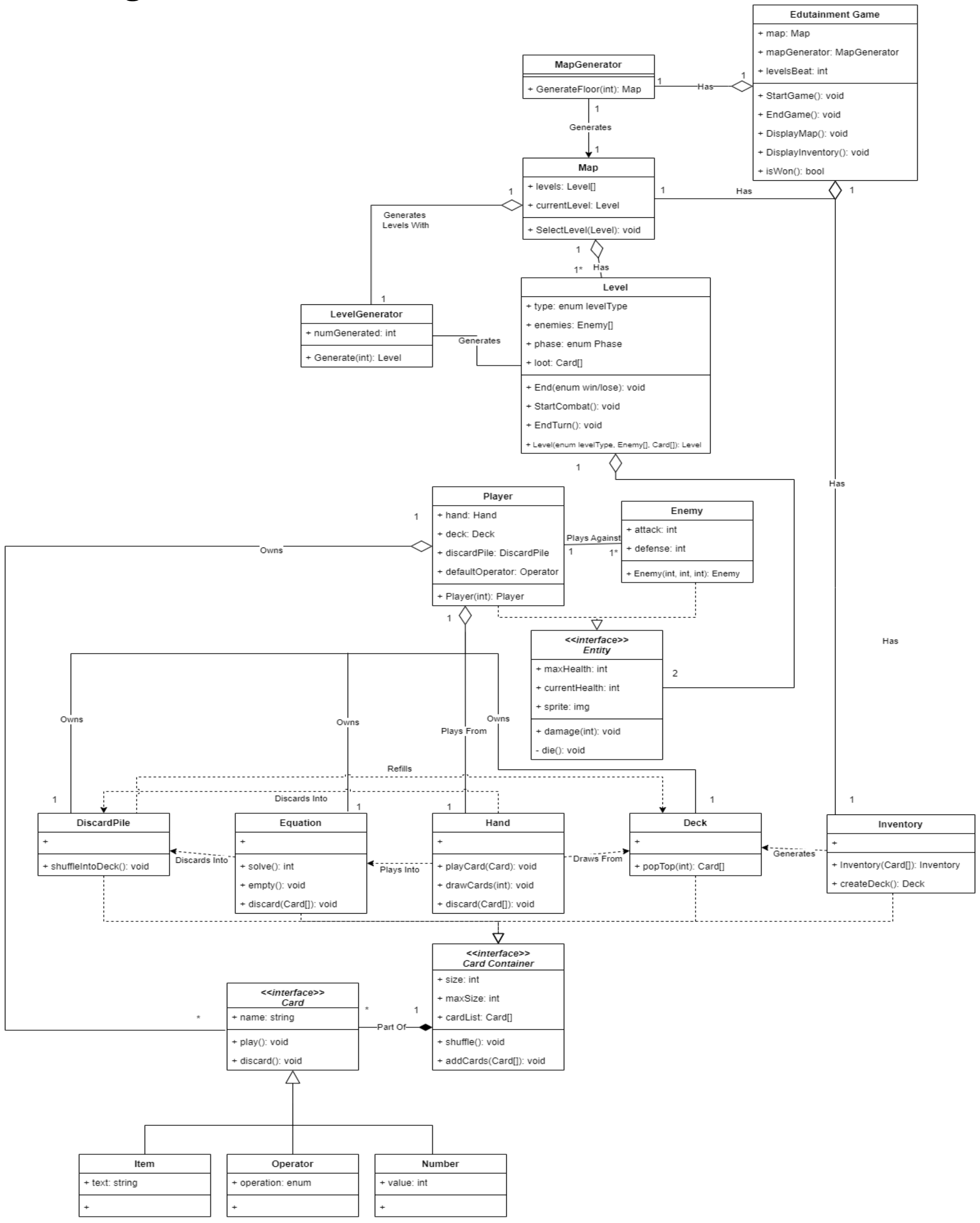

Fig. 3, Class Diagram

Data Dictionary:

Element Name		**Description**
Card		The cards players use to interact with the game through. Cards are either items, operators, or numbers.
Attributes		
	name: string	The name of the card.
Operations		
	play(): void	The operation of playing a card. Cards can only be played when in the hand and move to either the Equation or DiscardPile afterwards.
	discard(): void	The operation of discarding a card. Cards move from the hand directly to the DiscardPile.
Relationships	CardContainer - Any number of Card objects may be held within a CardContainer object. Item, Operator, and Number - Item, Operator, and Number objects are extensions of the Card object Player - The Player interacts through Cards	
UML Extensions	Has <<interface>> stereotype as any item interacting with Item, Operator, or Number objects does so through the Card object.	

Element Name		**Description**
CardContainer		An object to hold Card objects, saving their order.
Attributes		
	size: int	The current number of cards held in a CardContainer object
	maxSize: int	The maximum number of cards allowed to be held in a CardContainer object
	cardList: Card[]	The Card objects currently held in a CardContainer object
Operations		
	shuffle(): void	Randomizes the order of cards in a CardContainer object
	addCards(Card[]): void	Adds a number of cars to the end of a CardContainer object
Relationships	Card - CardContainer objects hold Card objects Deck, DiscardPile, Equation, Hand, and Inventory - Deck, DiscardPile, Equation, Hand, and Inventory objects are all extensions of the CardHolder object	
UML Extensions	Has <<interface>> stereotype as it is how Card objects interface with Deck, DiscardPile, Equation, Hand, and Inventory objects	

Element Name		**Description**
Deck		The object the player will draw from to add cards to their hand
Attributes		
Operations		
	popTop(count: int): Card[]	Removes count cards from the top of the Deck and returns the cards that were removed.
Relationships	CardContainer - Inherits from the CardContainer class DiscardPile - Once the deck is exhausted, the cards in the discard pile are shuffled into the deck Hand - The cards drawn into the hand are removed from the deck Inventory - The deck is generated using the cards contained in the inventory Player - Owned and drawn from by player	

Element Name		**Description**
DiscardPile		Where Card objects are stored once they are discarded
Attributes		
Operations		
	shuffleIntoDeck(): void	Transfers every card in the DiscardPile into the Deck
Relationships	CardContainer - Inherits from the CardContainer class Deck - Once the deck is exhausted, the cards in the discard pile are shuffled into the deck Equation - After the player ends their turn, all of the cards used in the equation are transferred to the discard pile Hand - At the end of each defense phase turn, all of the cards in the hand are transferred into the discard pile Player - Owned and discarded into by player	

Element Name		**Description**
Enemy		The player’s opponents in levels
Attributes		
	attack: int	The number an enemy will attack with during defense turns
	defense: int	The number an enemy will block with during attack turns
Operations		
	Enemy(hp: int, atk: int, def: int): Enemy	Constructor that creates an Enemy with hp health, atk attack, and def defense
Relationships	Entity - Inherits from the entity class Player - Battles against the player during combat	

Element Name		**Description**
Entity		Objects with health that are meant to be interacted with through combat
Attributes		
	maxHealth: int	The highest possible health number for an entity
	currentHealth: int	The current remaining health for an entity
	sprite: img	The image meant to visually represent an entity
Operations		
	damage(dmg: int): void	Reduces the health of an entity by dmg. Calls die() if health reaches zero or less.
	die(): void	Informs the level that an entity has died
Relationships	Enemy - The Enemy class inherits from Entity Level - Entities are owned by Level objects Player - The Player class inherits from Entity	

Element Name		**Description**
Equation		Brief description (e.g., purpose and scope).
Attributes		
Operations		
	solve(): int	Returns the result of the equation created by the player. Returns an invalid int if the player tries to submit an invalid equation
	empty(): void	Returns all cards in the equation to the player's hand.
	discard(cards: Card[]): void	Transfers cards to the discard pile
Relationships	CardContainer - Equation inherits from CardContainer DiscardPile - Discards into DiscardPile when discard() is called Hand - Cards played from hand go into equation Player - Owned by Player. Player goal is to build the highest possible number using the Equation.	

Element Name		**Description**
Game		An object to represent the entire game state
Attributes		
	map: Map	The layout of the game
	mapGenerator: MapGenerator	An object used to create a new map
	levelsBeat: int	The number of levels the player has beaten on their current run. Used to increase difficulty as level progress
Operations		
	startGame(): void	Begins a new run
	endGame(): void	Quits a run
	displayMap(): void	Shows the player the map
	displayInventory(): void	Shows the player their inventory
	isWon(): bool	Returns whether or not the game has been beaten yet
Relationships	Inventory - Owns the Inventory of a run Map - Owns the Map of a run MapGenerator - Owns the Map Generator	

Element Name		**Description**
Hand		The container for the playable cards in a turn
Attributes		
Operations		
	playCard(card: Card): void	Puts a card into the Equation
	drawCards(count: int): void	Places the top count cards of the Deck into the Hand
	discard(cards: Card[]): void	Discards cards and places them in the DiscardPile
Relationships	CardContainer - Hand inherits from CardContainer Deck - Cards drawn come from Deck DiscardPile - Cards discarded go to DiscardPile Equation - Cards played go to Equation Player - Player owns and uses Hand in combat	
UML Extensions	Include whether any of the UML extensions are used (e.g., stereotype (e.g., for software product lines), tags, constraints).	

Element Name		**Description**
Inventory		A container with all of the cards collected by the player in the current run
Attributes		
Operations		
	Inventory(cards: Card[]): Inventory	Constructor for Inventory that creates an inventory with all cards in the array
	createDeck(): Deck	Generates a deck containing all the same cards as Inventory
Relationships	CardContainer - Inventory inherits from CardContainer Deck - Generates new Decks Game - Owned by Game object	

Element Name		Description
Item		A card that grants a passive effect in exchange for an equation slot
Attributes		
	text: string	The description on the card shown to the player
Operations		
Relationships	Card - Item inherits from the Card class	

Element Name		Description
Level		The representation of one encounter in a run.
Attributes		
	type: enum levelType	The type of level
	enemies: Enemy[]	The enemies contained in a level
	phase: enum Phase	The current phase of combat
	loot: Card[]	The cards that will be offered as a reward for beating the level
Operations		
	end(enum Victory): void	Ends combat. Takes an enum that shows if the player won or lost
	startCombat(): void	Begins combat
	endTurn(): void	Ends the current turn. Calculates damage. Changes phases.
	Level(typeInit: enum LevelType, enemiesInit: Enemy[], lootInit: Card[])	Constructs a new level using parameters for attributes
Relationships	Entity - Owns entities LevelGenerator - Generated by LevelGenerator Map - Part of Map	

Element Name		**Description**
LevelGenerator		An object meant to create new Levels
Attributes		
	numGenerated: int	The number of levels that have been generated in this run
Operations		
	generate()	Generates a new level based on many levels have already been created
Relationships	Level - LevelGenerator generates Levels Map - The Map uses the LevelGenerator to generate Levels	

Element Name		**Description**
Map		The layout of the current run
Attributes		
	levels: Level[]	All of the levels
	currentLevel: Level	The level that the player is currently on
Operations		
	selectLevel(next: Level): void	Changes the currentLevel
Relationships	Game - Owned by Game Level - Holds all of the Levels LevelGenerator - Uses LevelGenerator to create new Levels MapGenerator - Generated by MapGenerator	

Element Name		Description
MapGenerator		An object to generate map layouts for runs.
Attributes		
Operations		
	generateFloor(size: int): Map	Generates Map based on a seed
Relationships	Game - Owned by Game Map - Generates Map	

Element Name		Description
Number		Cards that add a number to the equation
Attributes		
	value: int	The numeric value of the card
Operations		
Relationships	Card - Inherits from the Card class	

Element Name		Description
Operations		Cards that add an operator to the equation
Attributes		
	operation: enum Operations	The operation the card adds
Operations		
Relationships	Card - Inherits from the Card class	

Element Name		**Description**
Player		The user’s avatar and character the user needs to keep alive in order to win
Attributes		
	hand: Hand	The Player’s Hand
	deck: Deck	The Player’s Deck
	discardPile: DiscardPile	The Player’s DiscardPile
	defaultOperator: Operator	An Operator that will always be in the Player’s hand
Operations		
	Player(hp: int): Player	Constructor for the player, starting them with hp maxHealth.
Relationships	Card - The Player damages Enemies using Cards Deck - The Player owns and draws from the deck DiscardPile - The Player owns and discards into the DiscardPile Enemy - The Player and Enemies both try to damage each other Entity - Inherits from the Entity class Equations - The Player builds and owns Equations to battle Enemies Hand - The Player owns and plays card from the Hand	

Sequence Diagrams:

Sequence 1:

The general gameplay loop for the User. The scenario begins with the User starting the Edutainment Game, and continues until the player wins or loses.

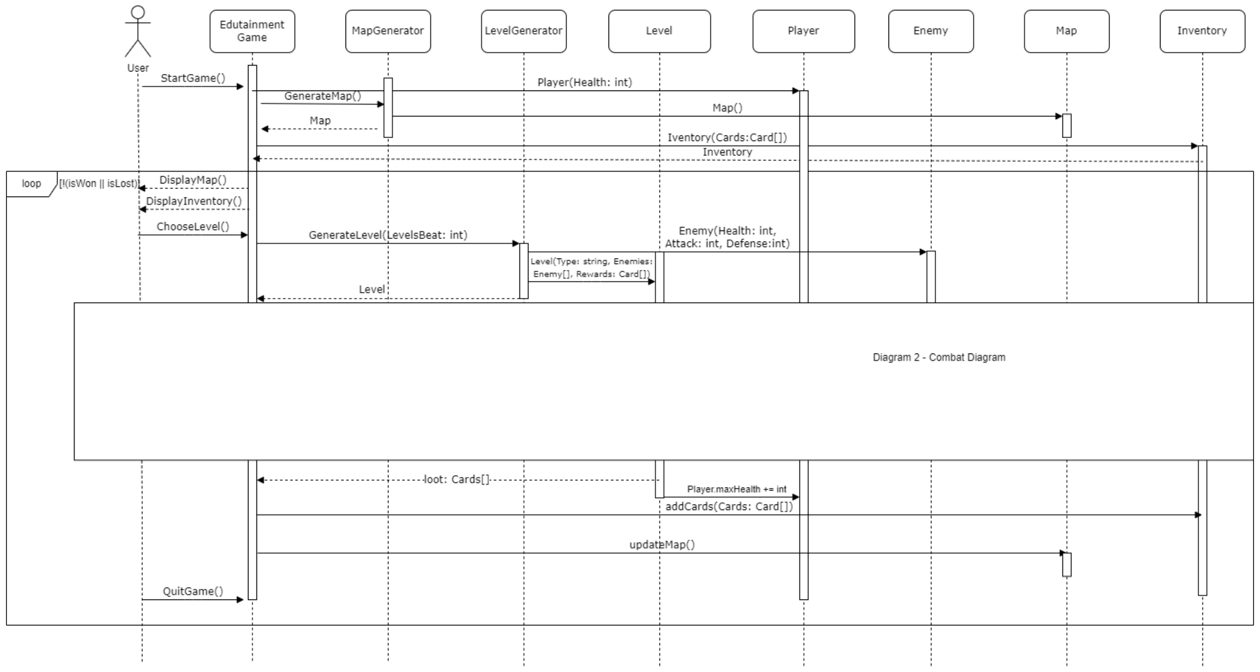

Fig. 4, Sequence Diagram 1

Sequence 2:

The entire Level sequence. The Level generates all of the Level-based objects then combat turns switch between offense and defense until the Player or Enemy is defeated.

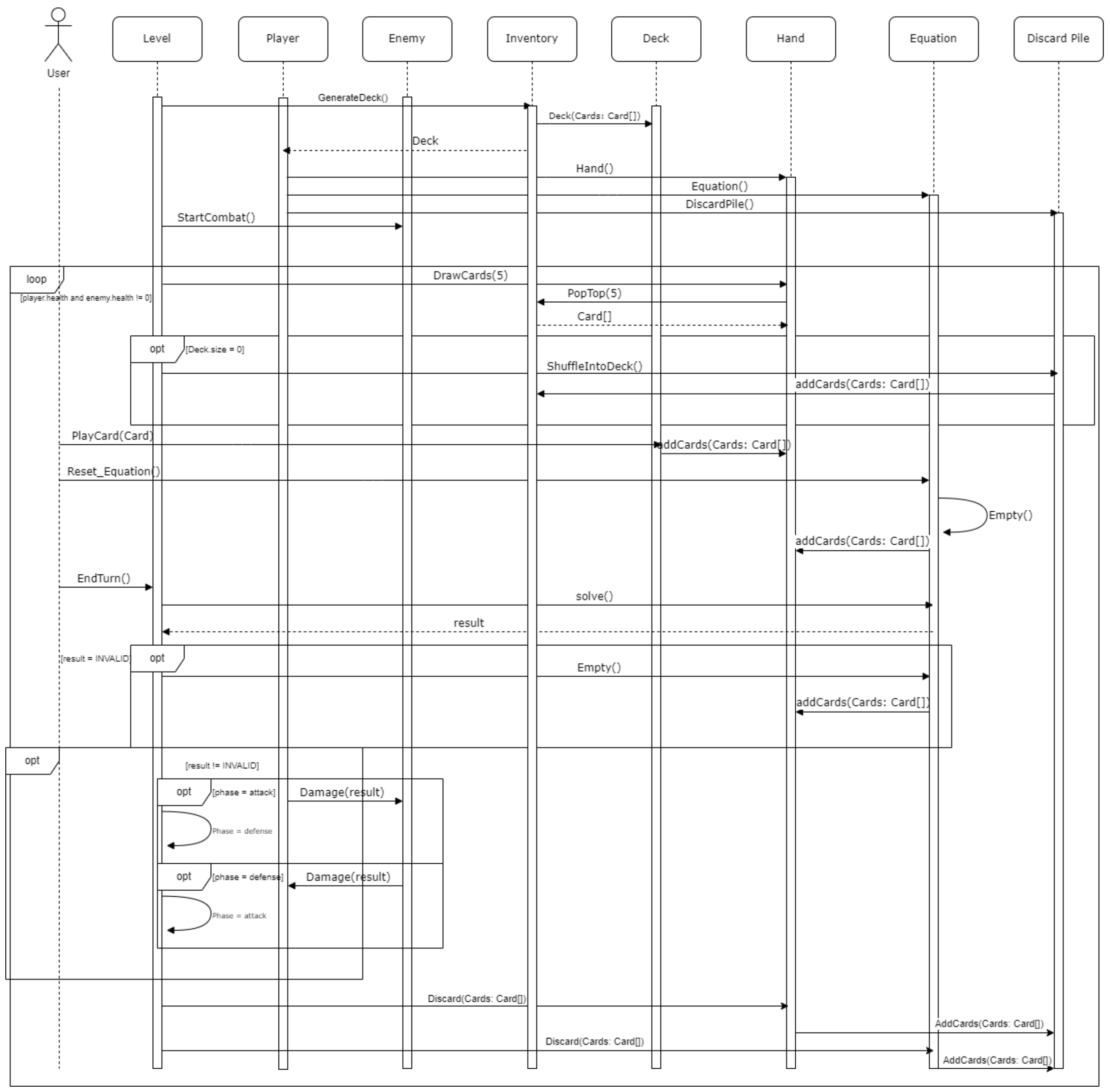

Fig. 5, Sequence Diagram 2

State Diagram:

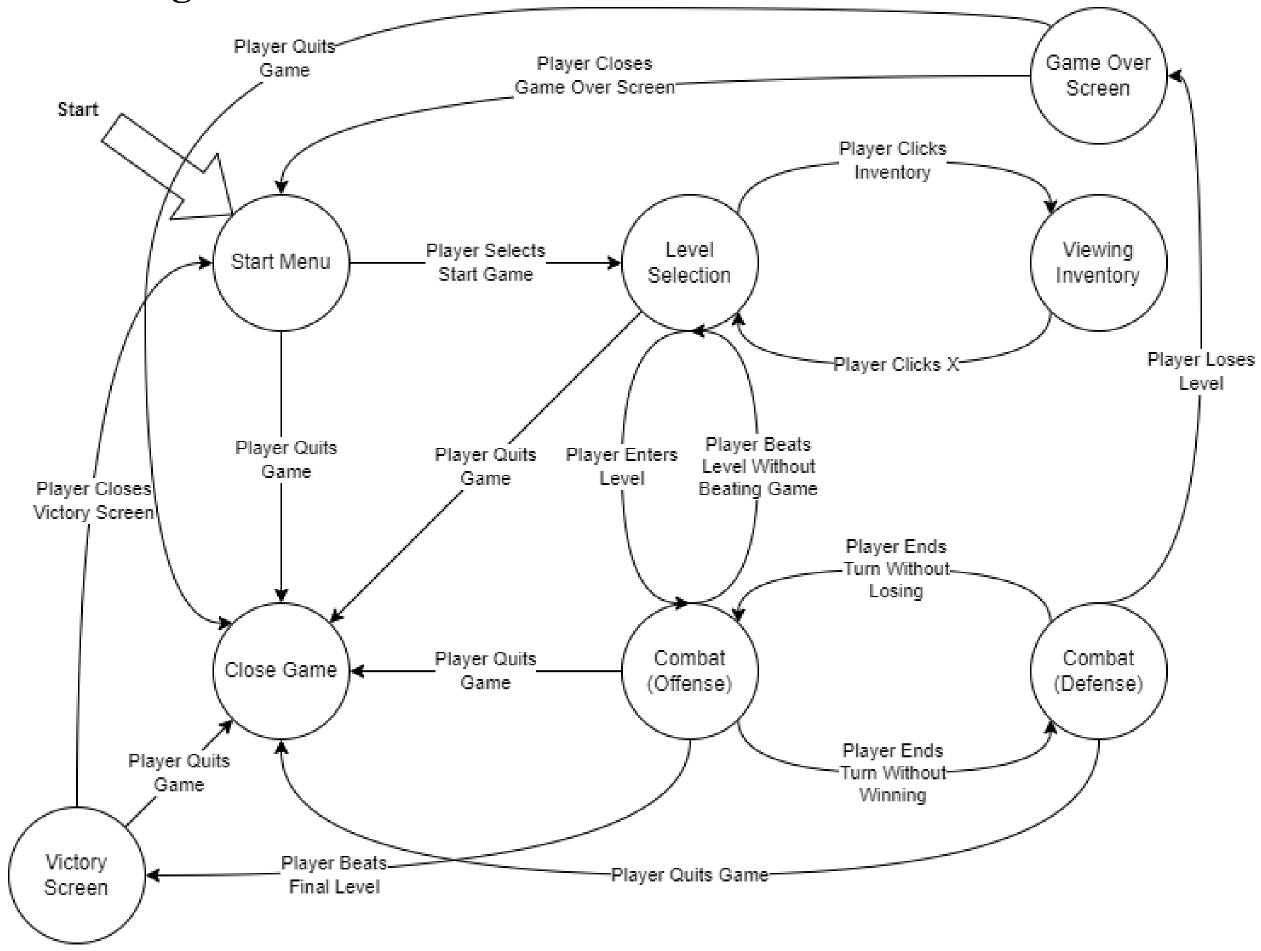

Fig. 6, State Diagram

5 Prototype

The UI prototype for this project shows the following system functionality. Menus for starting / quitting the game, level selection, and viewing the inventory. The prototype also contains a screen to simulate UI elements for combat during a level. This includes sprites and health bars for the player and enemy, menus to view the deck, and discard pile when clicked, a hand consisting of several draggable cards that can be placed into an equation bar, a button to simulate ending respective phases during combat, and buttons to display win/loss screens. Finally, after selecting the option to win combat, there is a menu where the user may choose one of three cards to simulate the progression system after the completion of a level.

5.1 How to Run Prototype

The prototype will be available via the project folder obtainable here: https://github.com/ForrestBoyer/SWEProjectTeam7/tree/UI-Prototype

For Windows users, after downloading the folder, running the FormulaFightersUI.exe file will run the prototype. The prototype may also be accessed through Godot 4.1.3 .Net for macOS, Windows, and Linux by downloading the project folder, importing the project.godot file as a new project, and running it inside of the Godot editor.

5.2 Sample Scenarios

Users can start a new game or quit via a start menu *fig. 7*. If the start game button is selected, the system will then display the level select screen as well as the option to view the inventory to the user *fig. 8*. After clicking on the inventory area, a new screen will be displayed to the user showing several cards separated by type *fig. 9*. The user may click the back button from this screen to return to the level select. The user may then select any visible level from this menu.

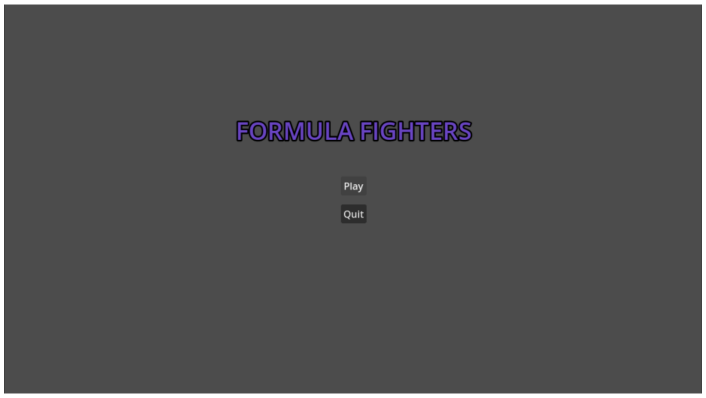

Fig. 7, Start Menu

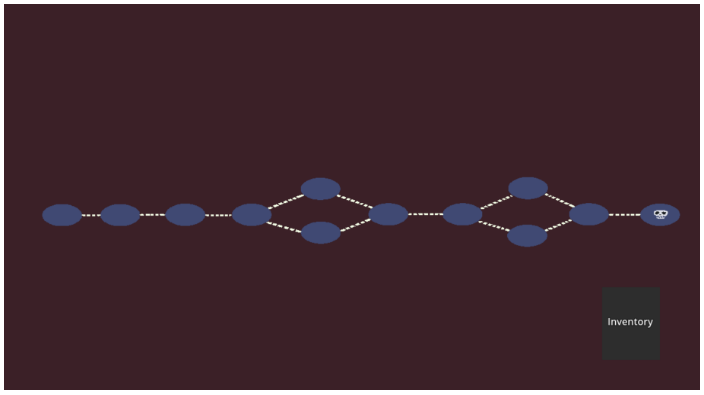

Fig. 8, Level Select

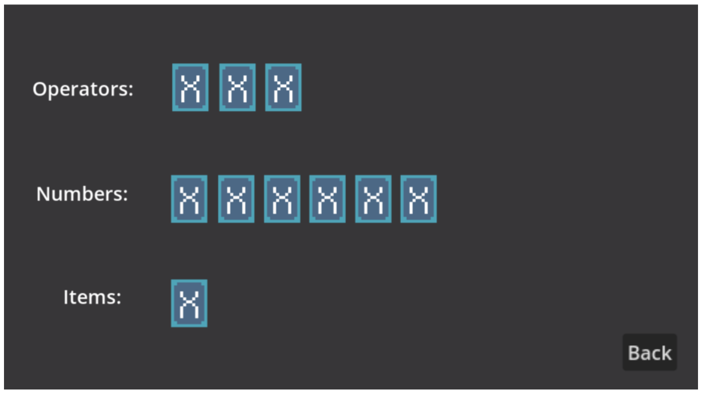

Fig. 9, Inventory View

Following this, the user will load into a level and be shown the combat screen *fig. 10*. In this screen, the user can view their deck, and discard pile in the same manner as their inventory during the level select screen. The user may also move the cards from their hand to the equation bar to build an equation *fig. 11*. The user can select the attack / defend button next to the equation bar to simulate ending the respective combat phase. In the combat screen, there are buttons to simulate the end of combat. If lose combat is selected, a game over screen will be displayed and the user will be returned to the start menu *fig. 12*.

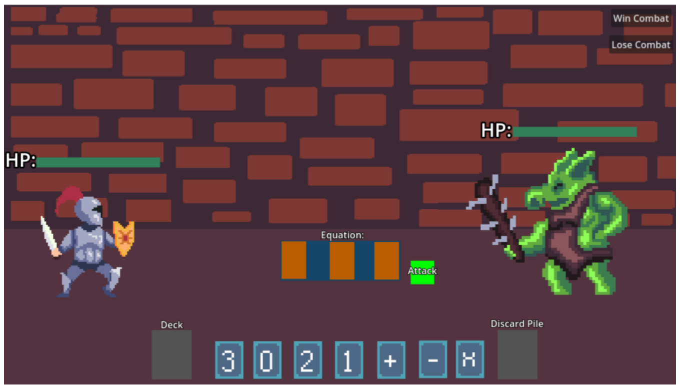

Fig. 10, Combat Screen

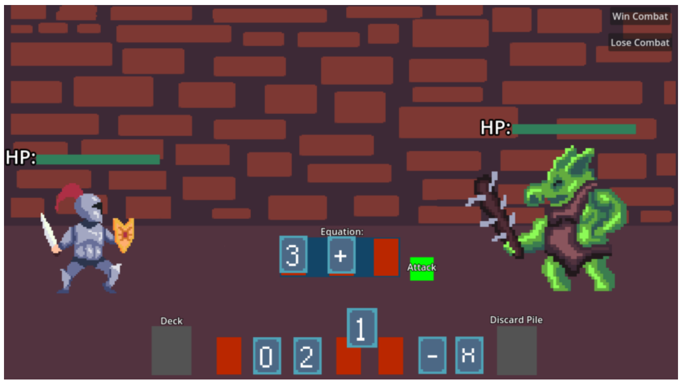

Fig 11, Moving Cards

Fig. 12, Game Over

If win combat is selected, a victory screen will be displayed to the user before the system prompts them with an option to choose one of three new cards to simulate the reward screen after the player completes the level *fig. 13*. Finally, the system will return the player to the level select screen.

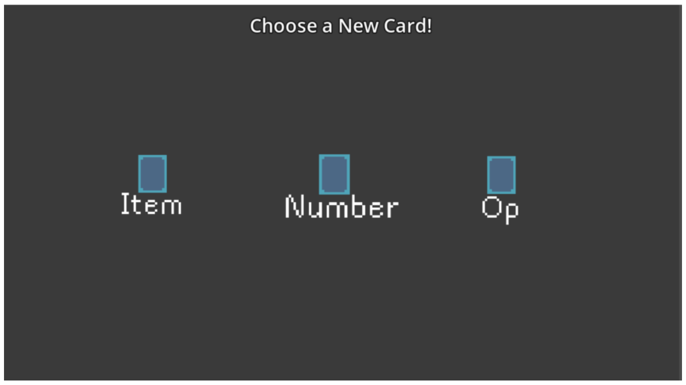

Fig. 13, Reward Screen

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. https://www.cse.msu.edu/~cse435/Handouts/SRSExample-webapp.doc.

[2] J. Linietsky and A. Manzure, “Godot Docs – 4.1 Branch.” *Godot Engine Documentation*, 2023. docs.godotengine.org/en/stable/index.html.

[3] J. Linietsky and A. Manzure, “Download Godot 4 for Windows.” *Godot Engine*, 1 Nov. 2023. godotengine.org/download/windows/.

[4] *Massachusetts Curriculum Framework for Mathematics - 2017*, Massachusetts Dept. of Elementary and Secondary Education, 2017.

[5] A. Lu, F. Boyer, K. Chan, T. Honnellio, and W. Shahbazian, “Formula Fighters” 17 Nov. 2023. https://forrestboyer.github.io/SWEProjectTeam7/.