

**University of Nevada, Reno**  
**College of Engineering**  
**Department of Computer Science**

**Dragonlord Chronicles**

**Team 18**

Sean Stevens  
Jonathan Meade  
Ryan Lieu  
Christine Vaughan

**Instructors**

Sergiu Dascalu  
Devrin Lee

**Advisor**

Eelke Folmer

November 19, 2018

## Table of Contents

<b>Abstract</b>	<b>3</b>
<b>Introduction</b>	<b>3</b>
<b>High-Level Design</b>	<b>5</b>
<b>Detailed Design</b>	<b>14</b>
<b>User Interface Design</b>	<b>18</b>
<b>Glossary</b>	<b>24</b>
<b>Contributions</b>	<b>26</b>

# Abstract

The goal of this project is to create a digital interactive roleplaying game (RPG) that an average person can use for personal entertainment. The primary focus of the game will be fighting, capturing, and training dragons in a medieval fantasy setting, with mechanics not unlike other RPGs such as Nintendo's *Pokémon* series. The game will offer its players an immersive and engaging narrative experience, as well as complex strategy required for many of the combat encounters. The game will be developed for Microsoft's Windows platform.

# Introduction

Dragonlord Chronicles aims to create a fantasy RPG where the player fights, captures, and controls dragons. The game will be a top-down 2D game where the player can explore a large open world and complete a main quest.

The most significant progress made since the last report is the development of the main story and combat system of the game. The player will awake in an abandoned temple in a strange and unfamiliar world. In this temple, the player will discover an ancient prophecy depicting a hero that would rise up and defeat the greatest of dragons: the Dragonlord. Armed with the knowledge of this prophecy, and some other miscellaneous gear, the player steps out into a strange world ruled by dragons. The player will encounter their first dragon, which the player must battle and defeat in order to survive. Upon emerging victorious from this battle, the player discovers they are able to magically break the dragon's will and enlist their assistance in the world.

From here, the player discovers they are the one told of in the prophecy and sets out to build their dragon army to overthrow the Dragonlord. The player will travel across the continent and gain enough power to overthrow each of the Dragonlord's generals and convert them to their cause before finally overthrowing the Dragonlord himself.

The combat system will consist of the player as the primary participant in combat, with up to one companion dragon by their side to complement their skillset. The player and dragon are each able to perform one action per turn, such as attacking, casting a spell, using an item, etc. Only one dragon may be active at a time, but different dragons can be swapped in and out during combat.

Unlike the player, dragons are magical creatures capable of casting spells, and every dragon have different magical capabilities. Each dragon the player is able to capture will have an element associated with it, a class, experience, and a tier.

Each dragon will also have an element associated with it, and will gain bonuses or penalties when facing enemies of other elements. For example, and fire dragon will receive a bonus to damage against ice dragons, but receive a penalty against earth dragons.

The classes will consist of generalized roles, such as Tank for drawing enemy attention and damage away from the player, Healer for restoring health, Brawler for dealing damage, and Sorcerer for specialization in magic. Dragons of a particular class will have spells and magical abilities to complement their roles.

Experience will determine the level and physical stats of a dragon, while tiers will determine the rarity and magical prowess of a dragon. Experience can be gained whenever the dragon participants in a successful combat encounter, but tiers can only be gained by performing magical rituals with rare and costly components. Dragons will begin with a randomized experience level and tier appropriate to the stage in the game they are encountered.

# High-Level Design

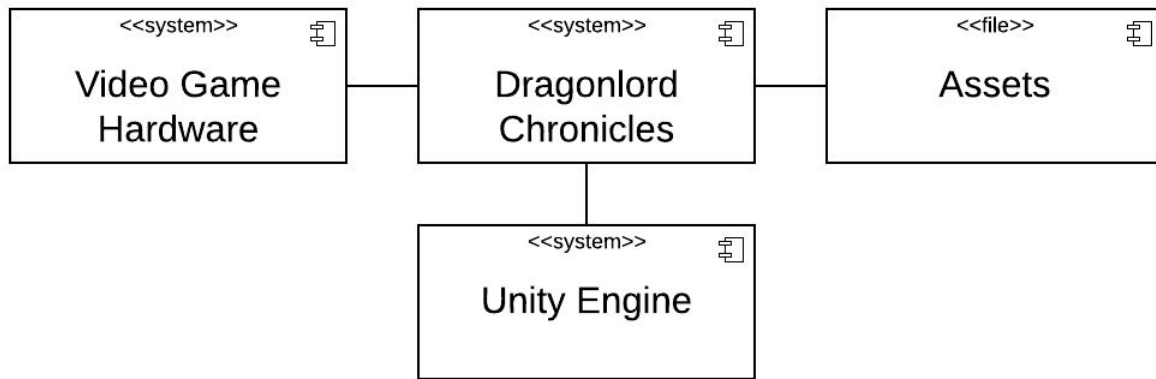


Figure 1: A component-based system level diagram of Dragonlord Chronicles' structure.

Figure 1 shows a model of the component structure of the game. The video game hardware will be a Windows machine or a gaming console (time permitting). It will accept player input via a keyboard or game controller and will provide the visual and auditory outputs of the game through a screen and speakers. Unity's game engine will handle the execution of the game, including visuals, music and sound effects, and all C# scripts that dictate how the game will respond to player input. Assets is a directory which will hold all scripts, sprites and other art, audio files, and any other data the game may need to retrieve.

For the data structures, Unity offers a special object called a scriptable object. A scriptable object is a data container that hold relevant information used to instantiate objects within a scene in Unity. In this game, the player will have its own unique scriptable object, and all dragons will have their own scriptable object associated with them.

The player's scriptable object will have the following attributes:

- **Health:** A representation of the amount of damage taken. When health reaches zero, the player dies.
- **Offense:** A numeric representation of how well the player is able to inflict damage on enemies.
- **Defense:** A numeric representation of how well the player is able to mitigate damage from enemies.
- **Gold:** A numeric value representing the currency in the player's possession.
- **Inventory:** A list of all items in the possession of the player.
- **Party:** A list of the dragons currently accompanying the player, to a maximum of four.

- **Default Dragon:** The default dragon that will assist a player in combat. The Default Dragon must be in the player's Party and may be changed outside of combat.
- **Active Dragon:** The dragon currently assisting the player in combat. The Active Dragon is always the Default Dragon at the beginning of combat, but may be swapped for another dragon in the player's Party during combat.

The dragons will have a generalized scriptable object. Each attribute will be assigned when the dragon is first created. If the dragon is captured by the player, the object will be saved and attached to the dragon permanently, else it will be deleted once the combat has concluded.

- **Health:** A representation of the amount of damage taken.
- **Element:** The element the dragon is associated with. A dragon may have exactly one element, which may not be changed. Possible elements include but are not limited to: Fire, Ice, Water, Earth, Lightning, Air, Flora, Fauna, Light, and Dark.
- **Level:** A numerical value that determines the strength of the dragon's Offense and Defense stats. Can be increased by gaining experience.
- **Experience:** A numeric value that increases a dragon's level whenever a threshold is reached. Experience is gained following a combat, and the amount gained is proportional to the level of the enemy defeated.
- **Tier:** A numeric value that represents a dragon's magical capabilities and available spells. Unlike experience, tiers can only be gained by performing special rituals which require rare and expensive components.
- **Offense:** A numeric representation of how well the dragon is able to inflict damage on enemies.
- **Defense:** A numeric representation of how well the dragon is able to mitigate damage from enemies.
- **Magic:** A list of all the spells a dragon is capable of casting.

The following tables describe main data structures that will be used in the project:

Class	StateManager
Method	LoadGame
Visibility	public
Return	bool
Parameters	string
Description	This function loads the latest save. The parameter is the persistent data path. It returns true if save data exists and it is readable. It returns false if no data can be loaded.

Class	StateManager
Method	SaveGame
Visibility	public
Return	bool
Parameters	void
Description	This function saves the game in its current state so the player may resume their adventure from the same location at a later time. It returns true if it successfully overwrote the game. It returns false if it cannot save in the current game state.

Class	StateManager
Method	QuitGame
Visibility	public
Return	void
Parameters	void
Description	This function pops game states until it is at the Main Menu state (which is always at the bottom of the state stack).

Class	StateManager
Method	GetCurrentState
Visibility	public
Return	GameState
Parameters	void
Description	This function returns the current state of the game.

Class	StateManager
Method	PushState
Visibility	public
Return	bool
Parameters	GameState
Description	This function pushes a new game state onto the state stack and then calls OnStateEnter on the new state. The parameter is the new GameState. The function returns false if the parameter is the same state as the current state. It returns true otherwise.

Class	StateManager
Method	PopState
Visibility	public
Return	GameState
Parameters	void
Description	This function calls OnStateExit on the current state and then pops the GameState that is on top of the state stack and then returns the popped value.



Class	StateManager
Method	TickState
Visibility	public
Return	void
Parameters	float
Description	This function runs logic on the GameState that is on top of the state stack by calling OnStateTick. The parameter is used to track the delta time, and it is passed into OnStateTick.

Class	GameState
Method	OnStateEnter
Visibility	public
Return	void
Parameters	params object[]
Description	This function handles any initialization when this state is entered. For example, when a battle state is entered, the parameters would be the player's party and the enemy's party along with their respective stats.

Class	GameState
Method	OnStateExit
Visibility	public
Return	void
Parameters	out params object[]
Description	This function handles anything that occurs when the state is exited. For example, if a battle state is exited, the data passed through the parameter would be based on the outcome of the battle (experience, new inventory items, etc.)

Class	GameState
Method	OnStateTick
Visibility	public
Return	void
Parameters	float
Description	This function implements the state behavior.

Class	EntityManager
Method	LoadEntityOfType
Visibility	public
Return	GameObject
Parameters	Int, Vector3,
Description	This function loads an entity based on the entity ID and the position of the entity. The returned value is the instantiated GameObject. The behavior of the instantiated entity will be defined by a MonoBehaviour script attached to the GameObject.

Class	EntityManager
Method	LoadAllScriptableObjects
Visibility	public
Return	List<ScriptableObject>
Parameters	string
Description	This function loads all scriptable objects, which is contains data for the different types of entities (enemies, dragons, NPCs, player, etc.). It returns each ScriptableObject as a list, which is used by the EntityManager to load entities.

Class	EntityManager
Method	DestroyEntity
Visibility	public
Return	void
Parameters	Entity
Description	Uses Unity's Object.Destroy method to remove the GameObject, and it removes the entity from the list of entities.

Class	Entity : MonoBehaviour
Method	Awake
Visibility	public
Return	void
Parameters	void
Description	Overrides Unity's MonoBehaviour.Awake function. This can be used to initialize data on the frame that the GameObject was instantiated.

Class	Entity : MonoBehaviour
Method	Start
Visibility	public
Return	void
Parameters	void
Description	Overrides Unity's MonoBehaviour.Start function. This function is called one frame after start and can be used for additional initialization.

Class	Entity : Monobehaviour
Method	Update
Visibility	public
Return	void
Parameters	void
Description	Overrides Unity's Monobehaviour.Update function. This can be used to implement the entity's behavior at runtime.

Class	Inventory
Method	AddItem
Visibility	public
Return	void
Parameters	Item
Description	Adds an item to the internal item list.

Class	Inventory
Method	RemoveItem
Visibility	public
Return	bool
Parameters	Item
Description	Removes an item from the internal item list if it matches the parameter. Returns false if the character does not have the item. Returns true if the character does have the item.

Class	Inventory
Method	GetItemInformation
Visibility	public
Return	string[]
Parameters	Item
Description	Returns a string array that contains the item's name, resell value, item type, and other important information for the item.

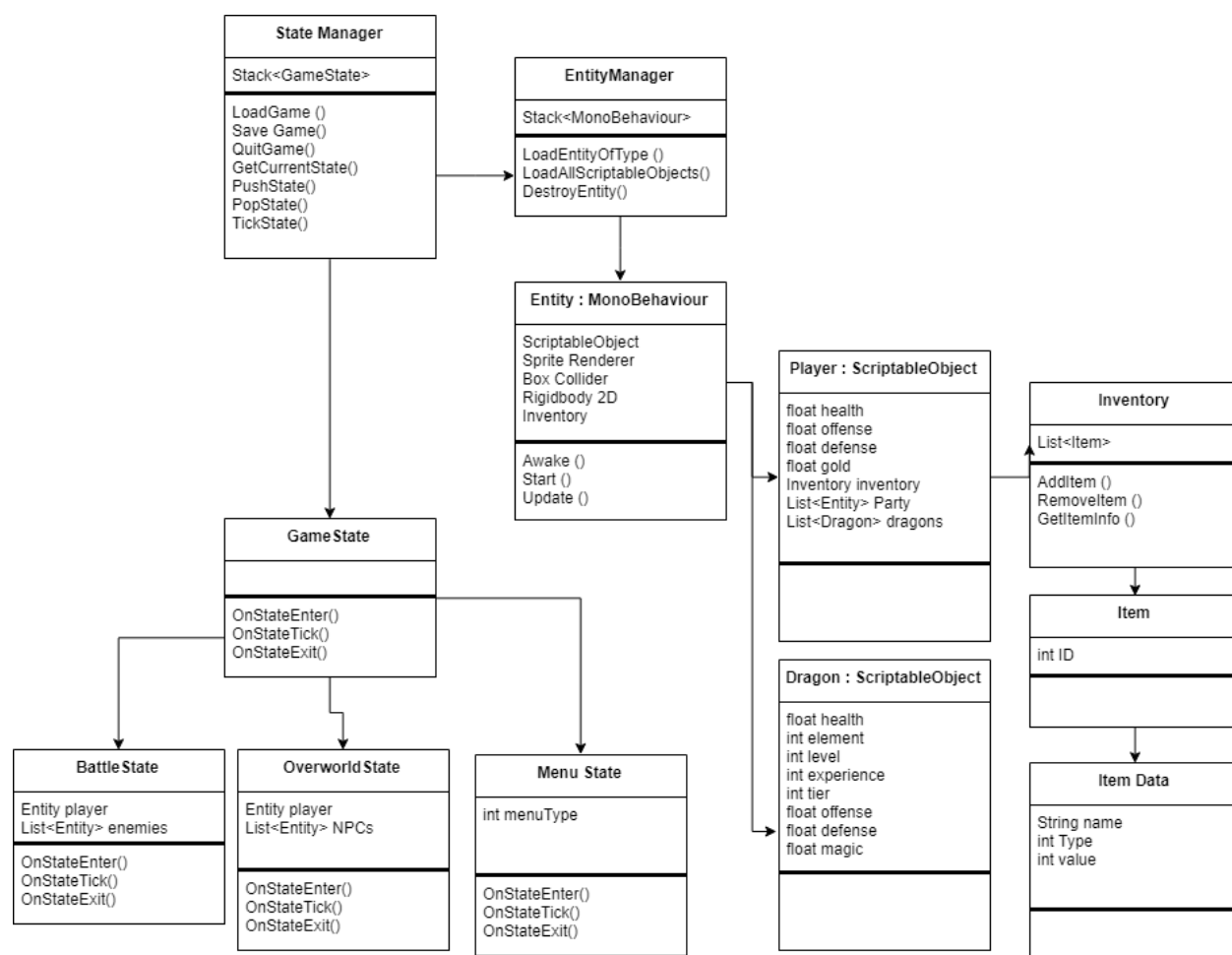


Figure 2: Class Diagram. This shows how the custom data structures are related to each other.

## Detailed Design

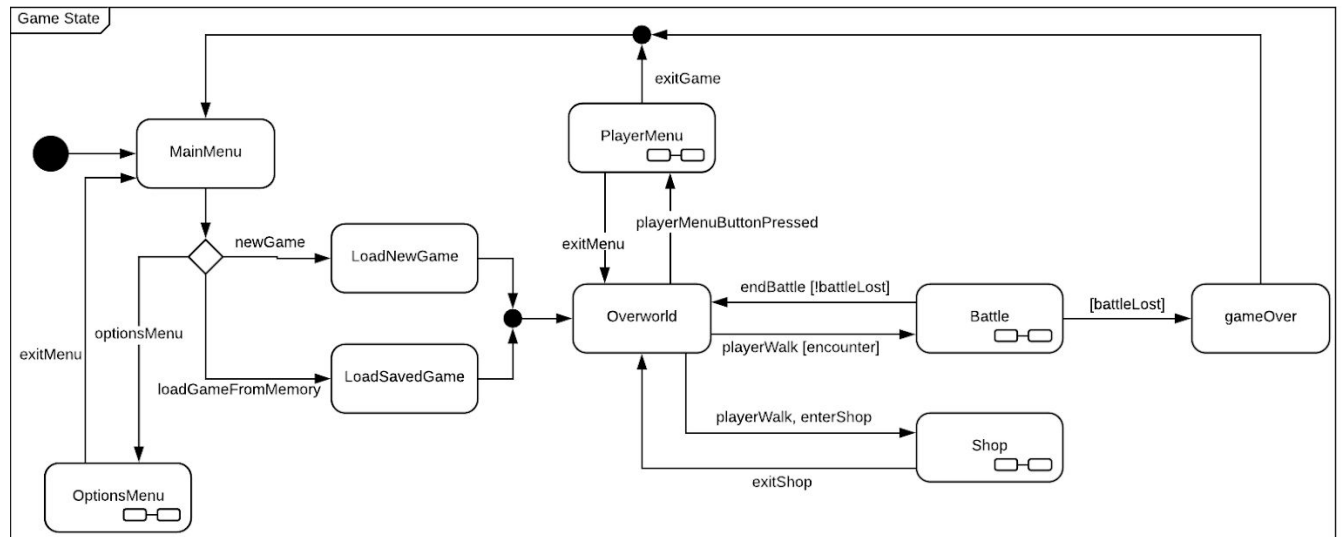


Figure 3: A state chart showing the flow of game states in Dragonlord Chronicles. A player begins at the main menu and can choose to start a new game or load a saved game. Then they can explore the overworld, use their menu, shop, and enter battles. If they choose to quit the game from the player menu or if they lose a battle, the game will go back to the main menu.

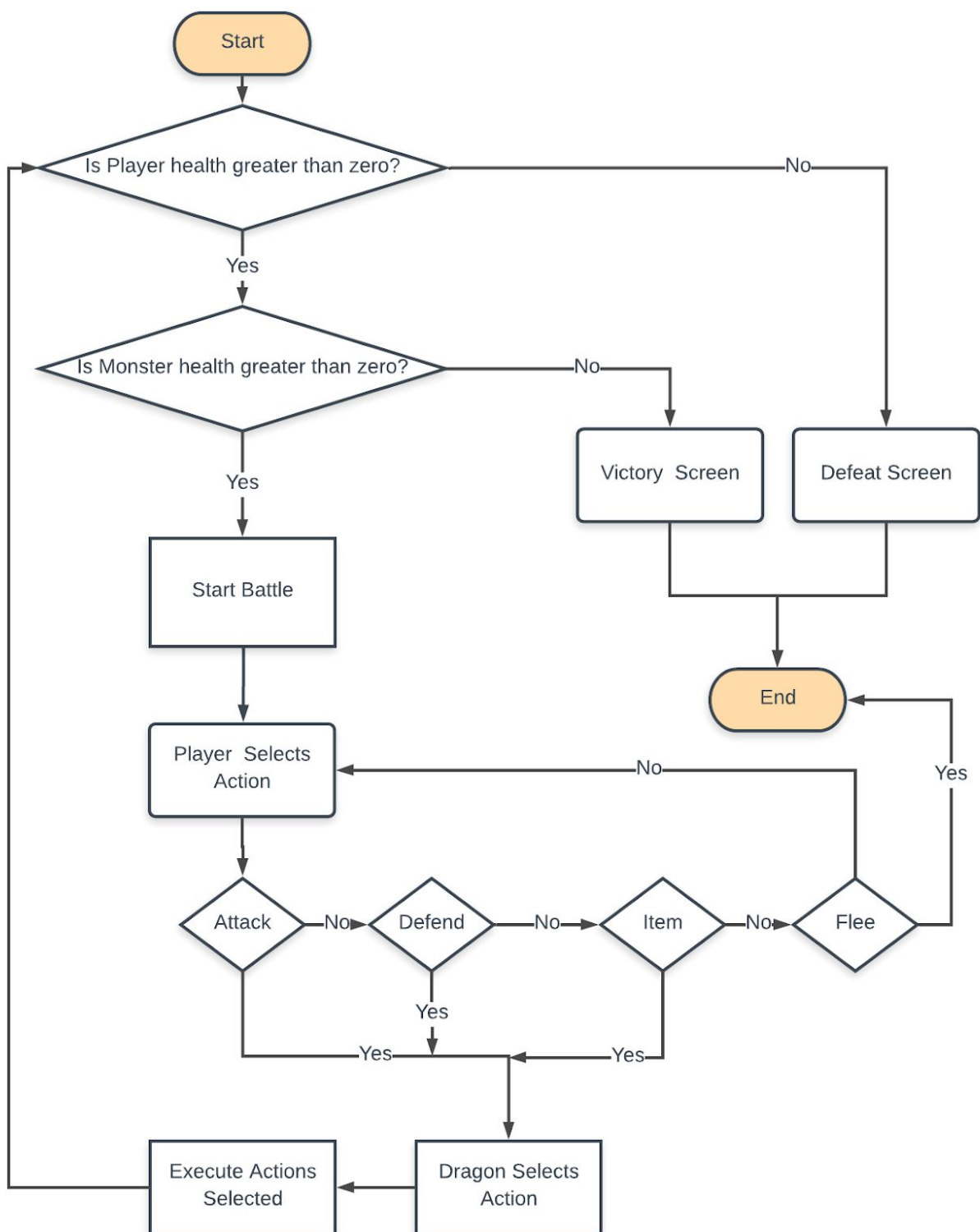


Figure 4: A flowchart giving an overview of the game's battle system.

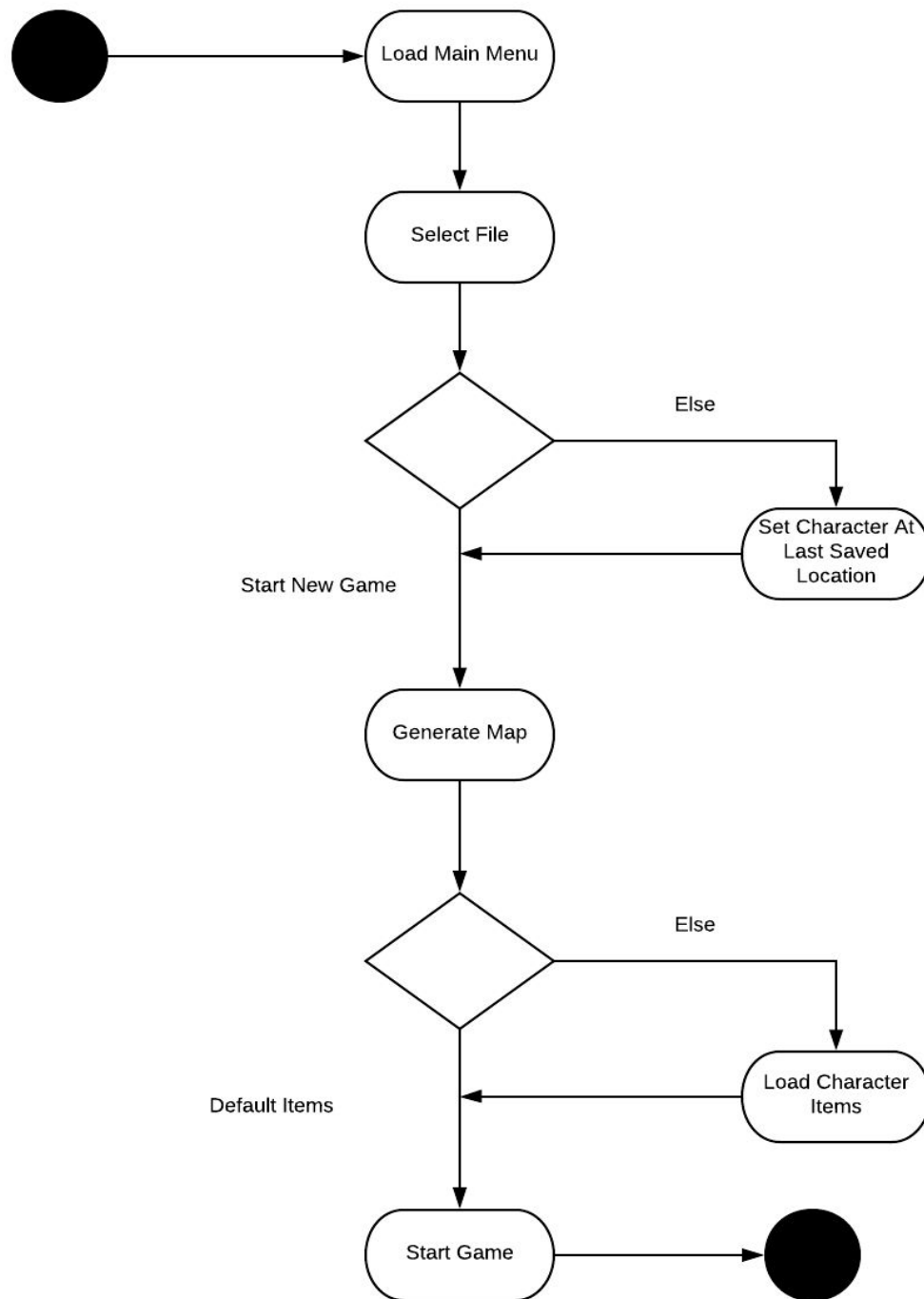


Figure 5: An activity diagram that shows the process of starting and loading a game.



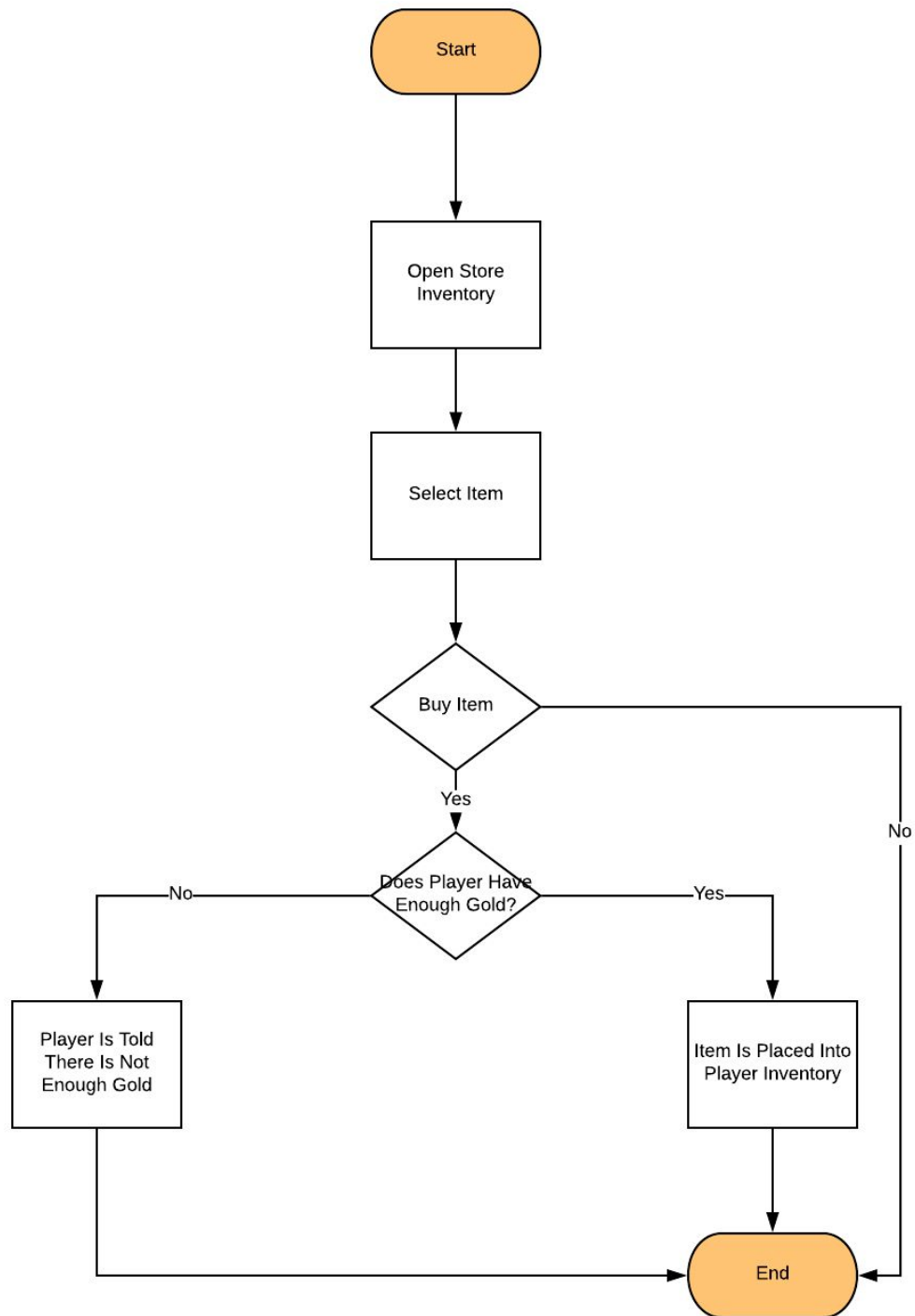


Figure 6: A flowchart giving an overview of the game's shop system.

## User Interface Design

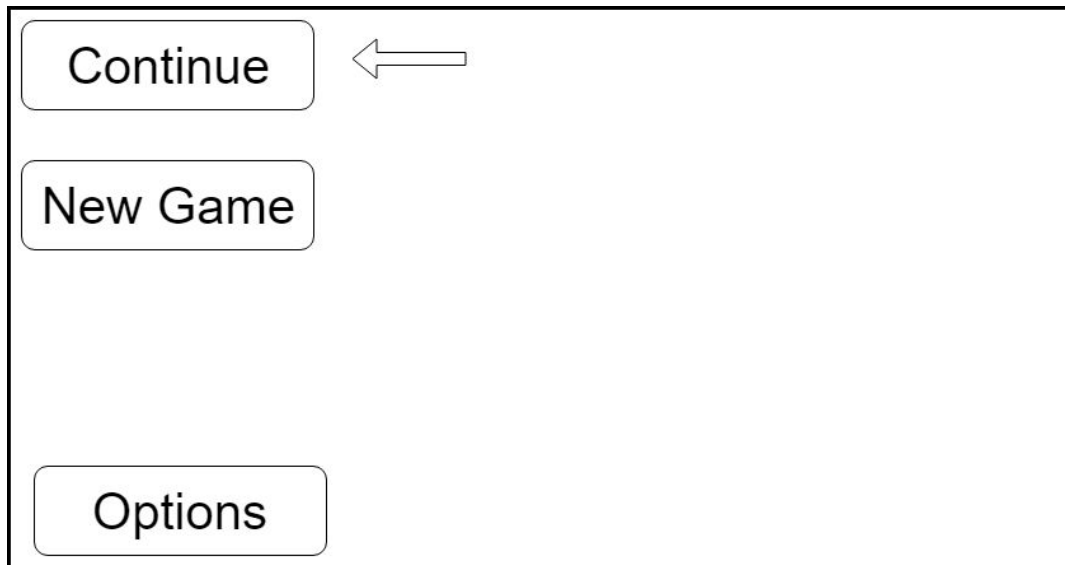


Figure 7: Main Menu. The screen shown when the game starts. Pressing “Continue” will load the previous save. Pressing “New Game” will delete any save data and start the player from the beginning of the story. Pressing “Options” will allow the player to configure settings for volume and input devices.

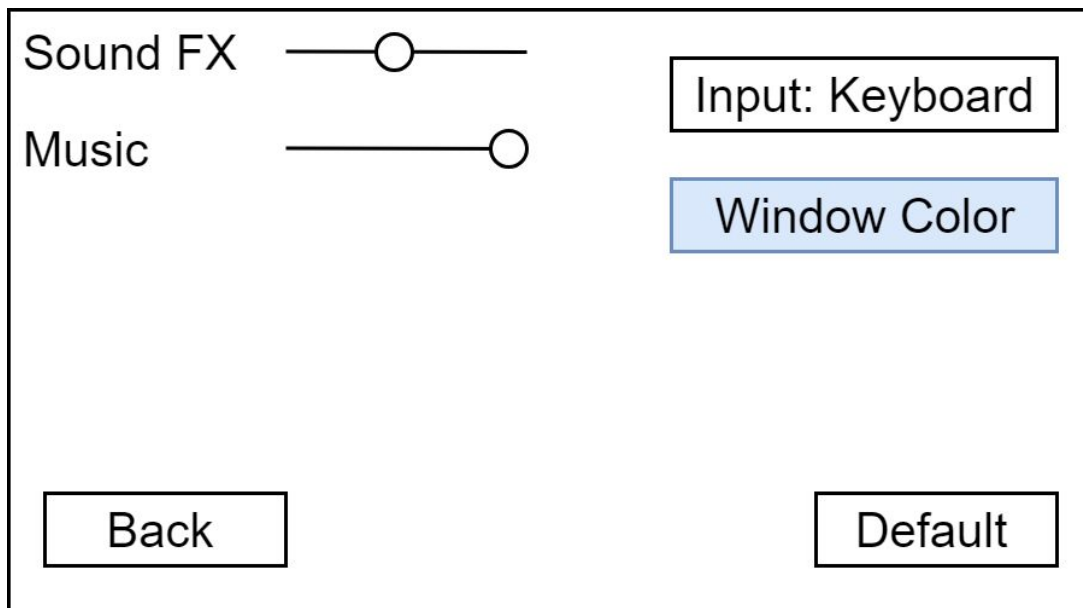


Figure 8: Options Menu to modify different sound and input settings or reset them to the default settings. The SoundFX slider will change the volume of sound effects. The Music slider will change the volume of the background music.

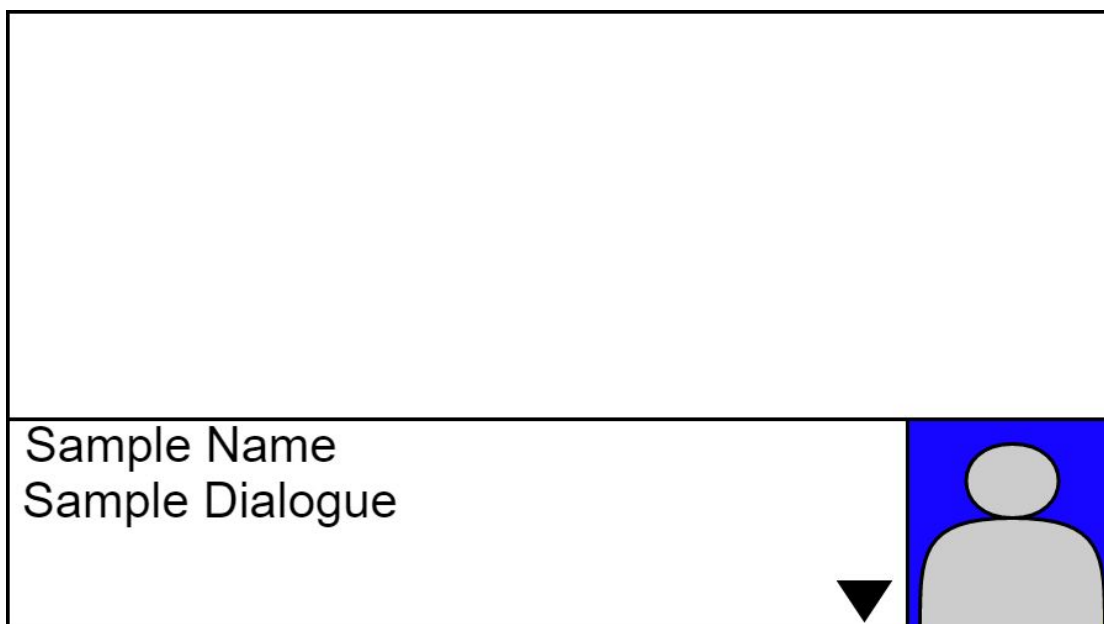


Figure 9: NPC Dialogue Menu. When the player presses the action button near an NPC, they will be able to speak to the player. The NPC will have an icon for their face and their name will be visible in the text box. There will be an on-screen indicator for when the player may advance to the next text box.

Shield of Shielding	◀ Cost: 500	Gold: 15000
Sword of Slashing		Equipment
Health Potion		Sell
Magic Potion		
Bronze Armor		
Bracelet		
Item Description		

Figure 10: Shop Menu. When the player talks to a shopkeeper, they may purchase items for their quest. Items will be sorted based on their type (weapons, armor, support, and potions). The player may get a description of each item or sell items from their own inventory.

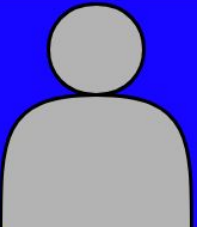
	Armor Accessory Weapon	Sample Item A Sample Item B Sample Item C Sample Item D	
	Attack 240 Defense 50 Agility 75 Health 350 Magic 60	Gold: 10, 413	Back

Figure 11: Equipment Menu shows the player's stats and their inventory items. The player may equip weapons, armor, and accessories to improve their stats, and the player may use items to heal, restore magic, and to support them in battle.




 				
 				
Fight	Magic	Defend	Item	Flee

Figure 12: Battle Menu. When the player encounters an enemy, they will enter a scene where they can make turn-based decisions for battle. Fight will have the player use physical weapons. Magic will have the player call their current dragon to use a spell. Defend will have the player enter a guard stance. Item will have the player search their inventory for a support item. Flee will have the player escape the battle.

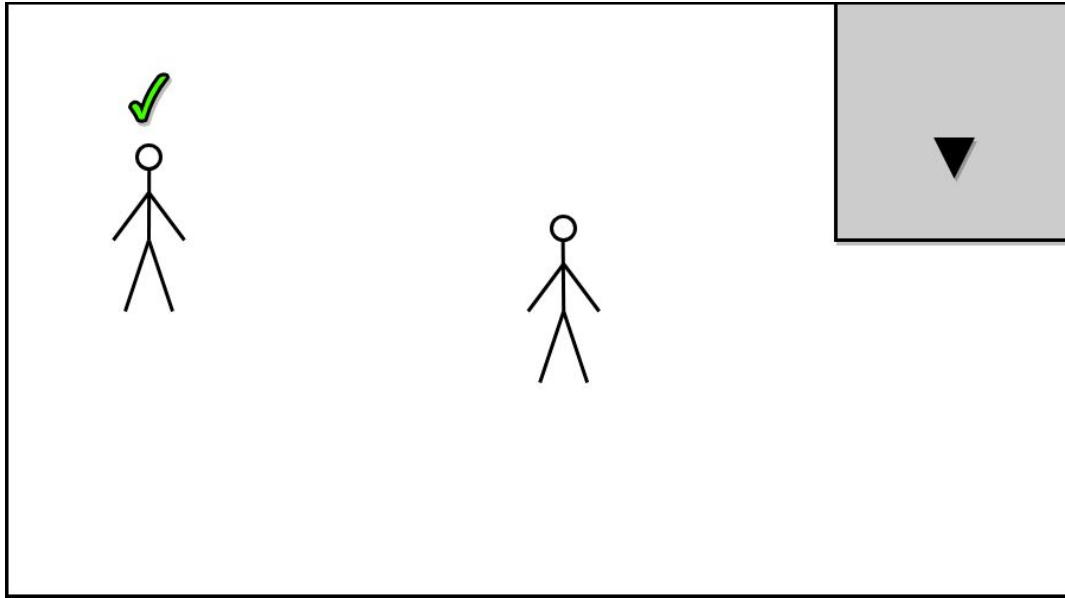


Figure 13: Overworld Screen. During overworld exploration, the UI will be very minimal only displaying visual cues for NPCs with quests and a minimap. An NPC with a quest will be denoted with a check mark for when it is completed and a question mark for when there is a new quest.

	001	???	???	???	???	???
▶	002	Sample Name	COMMON	FIRE	HEALER	★
	003	Sample Name 2	COMMON	WATER	TANK	
<div>BACK</div>						

Figure 14: Dragon Encyclopedia. As the player encounters and captures dragons, their stats (number, name, rarity, element, class) will be displayed. Otherwise, that information will be denoted with “???”. If the player both encountered and captured the dragon, a star will be in the final checkbox, allowing players to keep track of which dragons they encountered and which ones they captured.

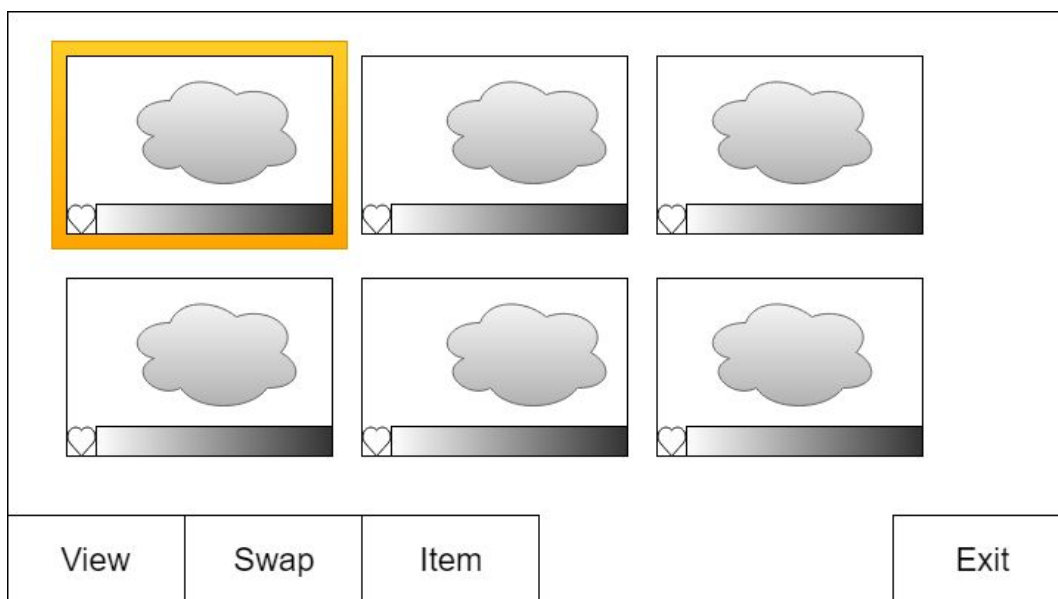


Figure 15: Dragon Party. This screen shows the six dragons that the player may use during battle. The basic information showed is the dragon's health and a sprite showing the dragon's design. More detailed information may be found by selecting the dragon and clicking "view". To change to a different dragon, the player may press "swap". To use an item on a dragon outside of battle, the player may press "swap".

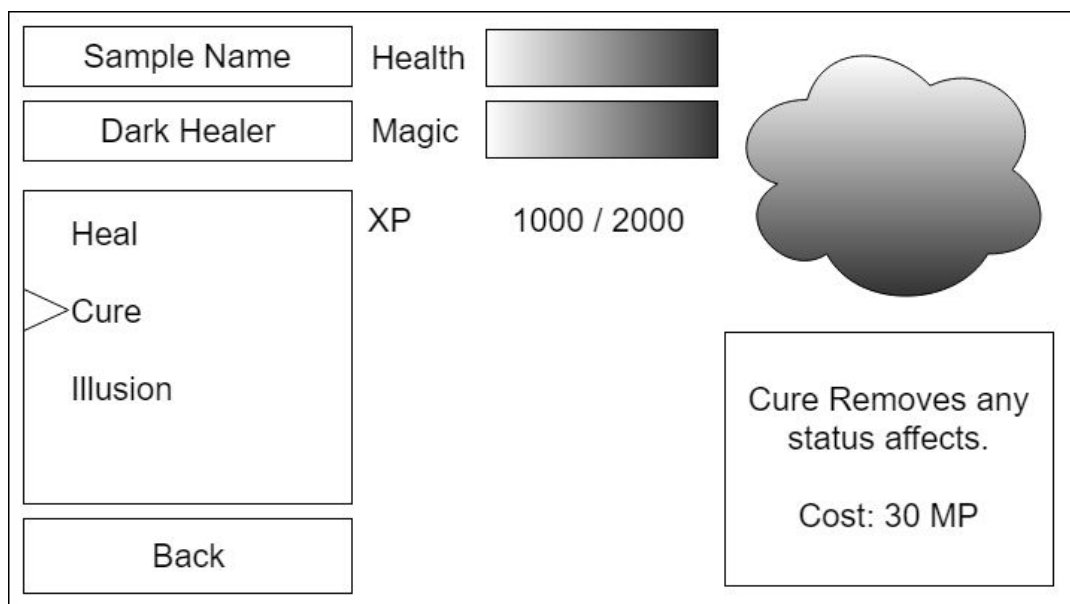


Figure 16: Dragon Stats. When the player selects one of their dragons to view, they will enter this menu, which shows detailed information about how much XP points a dragon has gained, the elemental type, the abilities, and the class type. The player may also get a description of each spell's abilities.

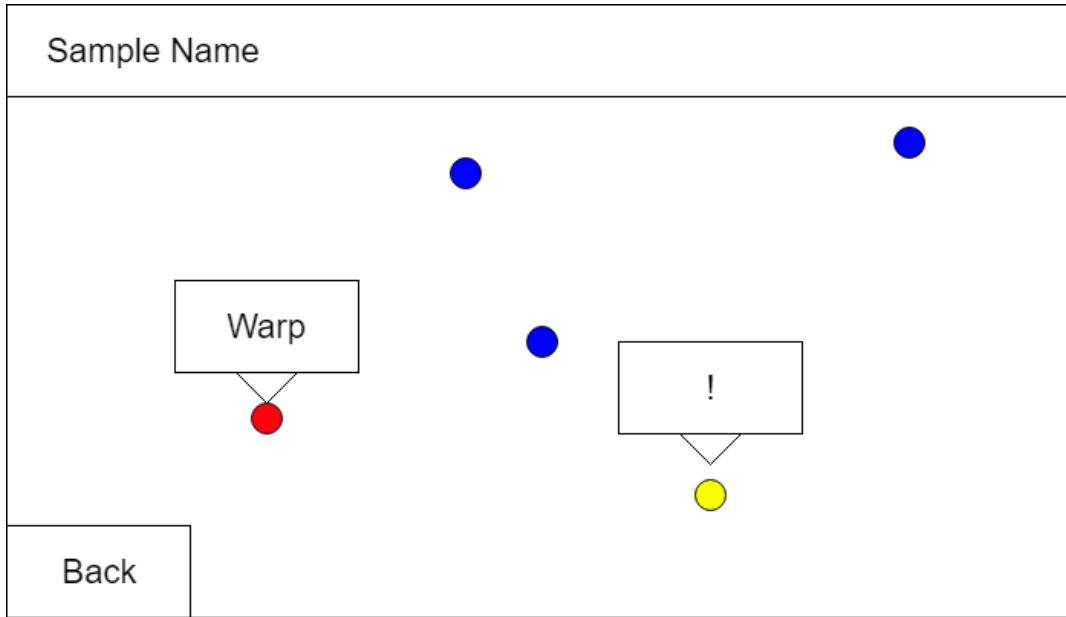


Figure 17: World Map. As the player explores the world, discovered locations will be pinned onto the map. The player may skip travel to different locations by selecting the pin and then clicking warp. The name of the selected region will be shown on the top of the screen. The region for the next main quest will be denoted with an exclamation mark.

Main Story Part 1	<div> <div>Quest Summary</div> <div>Quest Outcome</div> </div> <div>2500 XP - 0 Gold - Mysterious Egg</div>
Main Story Part 2	
Current Quest	
Sidequest 1	
Exit	

Figure 18: Quest Journal. As the player completes the main story, major events of the main quest will be logged onto the quest journal. The data logged will be the quest reward, the name of the quest, a summary of the quest, and its outcome.

## Glossary

Component	A component defines specific attributes that may be attached to GameObjects. The scriptable components are MonoBehaviors while non-scriptable components are renderers, colliders, rigidbodies, etc.
Equipment	Virtual items that can be picked up by the game's main character.
Frame Rate	The number of visual updates per second
Game Engine	Software that can be used to develop a video game
GameObject	The fundamental objects in Unity that may represent graphics, physics, and behaviors, depending on its attached components.
Gamepad	A handheld controller for video games.
Gameplay	The actions that the player takes while playing the game.
Game State	A particular condition or behavior exhibited by the game.
Inventory	A list of virtual items that the player is carrying.
MonoBehaviour	The base class for every script component in Unity. This may be attached to a GameObject to elicit game-specific behavior.
NPC	Acronym for Non-Player Character, which refers to any in-game character that cannot be controlled by the player
Overworld	The area in the game that connects all of the locations
Player	The end user of the game
Plot	The main sequence of events in the game.
Prototype	A preliminary model design to test the functionality or the design of a product.
Quest	A mission that the player may complete
Role Playing Game (RPG)	A genre where players assume the role of a fictional character who will have an adventure in their world.
Scene	A distinct environment of the game.



ScriptableObject	A special container class that can represent data without a GameObject.
Sprite	A 2D image
Super Nintendo Pixel Art	Minimalistic artwork where the image is comprised of a small pixel resolution and a few colors per image.
Tilemap	A 2D grid of images that are the same distance apart.
Top-Down	A game where the player's perspective of the world is from an elevated viewpoint.
Turn-based Combat	A battle system in which the player takes their turn and then the enemy takes their turn.
Unity	A game engine designed to create 2D and 3D games
UI	User Interface for the player to interact with the software
Video Game	A game played by manipulating images displayed on a monitor or television.

## Contributions

Jonathan worked for approximately two and a half hours writing the title page, abstract, introduction, and the Player and Dragon data structure descriptions.

Sean worked for four hours on the User Interface Design section, adding class diagram and the method descriptions in the High Level Design section, and alphabetizing the glossary terms.

Christine worked for about three hours creating the system-level component based diagram and the state chart for the game state, and writing descriptions for both.

Ryan worked for about three hours on the Detailed Design portion, creating the flowcharts and activity diagram.

In addition, all team members spent approximately one and a half hours in a group meeting deciding on elements of the game, such as story and mechanics, and planning out how they would create this document.