University of Nevada, Reno

College of Engineering

Department of Computer Science

Dragonlord Chronicles
Revised Specification and Design

Team 18

Sean Stevens

Jonathan Meade

Ryan Lieu

Christine Vaughan

Instructors

Sergiu Dascalu

Devrin Lee

Advisor

Eelke Folmer

CSE Department Chair

February 22, 2019

Table of Contents

Abstract

Dragonlord Chronicles is designed to be an interactive Role-Playing Game (RPG) that can be enjoyed by average players. The primary focus of the game is fighting, capturing, and training dragons in a medieval fantasy setting, taking inspiration from Nintendo’s *Pokémon* series. The game offers players with an immersive and engaging narrative experience as well as complex strategy required for many of the combat encounters. The game will be developed using Unity and programmed in C#. It will feature 2D pixel art aesthetics to make it feel reminiscent of classic SNES RPGs.

Introduction

For our senior project, we will be developing an interactive Role Playing Game where players will be able to explore an immersive, virtual world. This game is designed to allow people to relieve stress and have fun playing with the mechanics that the game will offer.

The theme of the game is a fantasy setting, with technologies from the bronze age, in conjunction with magical forces. The people of the world worship dragons as their divine creators and they believe that dragons are responsible for the magical elements of nature (earth, water, fire, and air).

As the player explores the world, they will talk to NPCs, complete quests, and gain insight to the characters who inhabit the world. The player will explore different biomes in search of the divine dragons to obtain the power needed to seal away a great evil and save the world.

There will be a turn based combat system where the player will have a limited amount of time to decide the turn they will make. They will be able to fight with a weapon, use magic, or use a potion during battle to defeat enemies. There will also be a stats system where each characters stats will have an affect in the outcome of battle.

There will also be an inventory system where the player has to manage their weapons and armor. In addition, when the player explores “cold” and “hot” sections of the world, they may need to change their armor or use a potion that would enable them to traverse the environment.

The game will be controllable with a keyboard and mouse or with an Xbox controller. It will be developed using Unity and written in C#.

The changes in the project between now and the proposal from PA1 is mainly that we have fleshed out the game's setting, story, and gameplay. These changes were made because we needed a more specific idea for the game's setting and gameplay. Now that we have a better idea for the direction of our final project, we will be able to create a more refined and fun game.

Recent Project Changes

Since project P1, there has only been one change to the game. The decision was made to change the overall layout of the world from a single, continuous space to a series of areas separated by loading screens. This change was made primarily for performance reasons, but also to more easily add location-based constraints to the random dragon generation algorithm.

Summary of Changes in Project Specification

There have been several updates to the project specifications since Fall 2018. The high level business requirements have been updated to more concisely define the project's goals. The technical requirements have also been updated to better reflect the features we will implement, or in the case of L3 would like to implement. The Use Case diagram has been fixed, and some of the detailed use cases have been rewritten for length and clarity.

High Level Business Requirements

The goal of this project is to fulfill the CS 425/426 Senior Project requirements by creating a digital RPG.

1. The game should be an RPG and feature a main character and several dragon companions.
2. The main character should be able to capture and train new dragon companions.
3. The game should feature a list of all the different types of dragons, with information on each.
4. The game should be marketable towards gamers of every age group, with content that various audiences can find enjoyable. The plot of the game should be engaging enough to keep an adult player interested, but not too complex for a younger player.
5. The game should contain minimal profanity and no content that is inappropriate for children to ensure that it remains marketable to that demographic.
6. The game should be released for PC through digital downloads.

Technical Requirements Specification

[L1] Denotes requirements we plan to implement by the end of Spring 2019.
[L2] Denotes requirements we might implement by the end of Spring 2019.
[L3] Denotes requirements we would like to implement, but most likely will not be completed by the end of Spring 2019.

Functional Requirements:

FR01	[L1]	The game will have a main menu display.
FR02	[L1]	The game will have character dialogue boxes.
FR03	[L1]	The player will be able to encounter, fight, and capture various randomly generated dragons.
FR04	[L1]	The player will be able to select their captured dragons and train them to increase their combat skills.
FR05	[L1]	The game will have an in-game menu with different submenus.
FR06	[L1]	The game will have options to adjust volume and input devices.
FR07	[L1]	The world map will be comprised of several areas, separated by loading screens.
FR08	[L2]	The game will have music audio.
FR09	[L2]	The player will be able to visually customize their character.
FR10	[L2]	The player will have different means of capturing dragons, some of which may be more effective than others.
FR11	[L2]	The player will gain information on a specific type of dragon the more it is encountered/captured.
FR12	[L3]	All sprites will be fully animated.
FR13	[L3]	The game will feature a morality system (ie. good vs. evil), which is impacted by the player’s actions.

FR14	[L3]	The game will have visual options for colorblindness modes (black & white and inverted colors).

Non-functional Requirements:

NFR01	[L1]	The game will be playable on Windows platforms.
NFR02	[L1]	The game will be controllable with both a keyboard & mouse setup and a gamepad.
NFR03	[L1]	The game will have pixel art reminiscent of Super Nintendo RPGs.
NFR04	[L1]	The game will be developed using Unity.
NFR05	[L1]	The game will not have a loading time of more than two seconds when transitioning to a new scene.
NFR06	[L1]	The game's assets will be loaded from disk through the "streamingAssets" folder.
NFR07	[L1]	The game will be saved using JSON Serialization.
NFR08	[L2]	The game will support multiple monitor resolutions and aspect ratios.
NFR09	[L2]	The game will be able to automatically recover from unexpected bugs and crashes.
NFR10	[L3]	The game may release on the Nintendo Switch.

Use Case Modeling

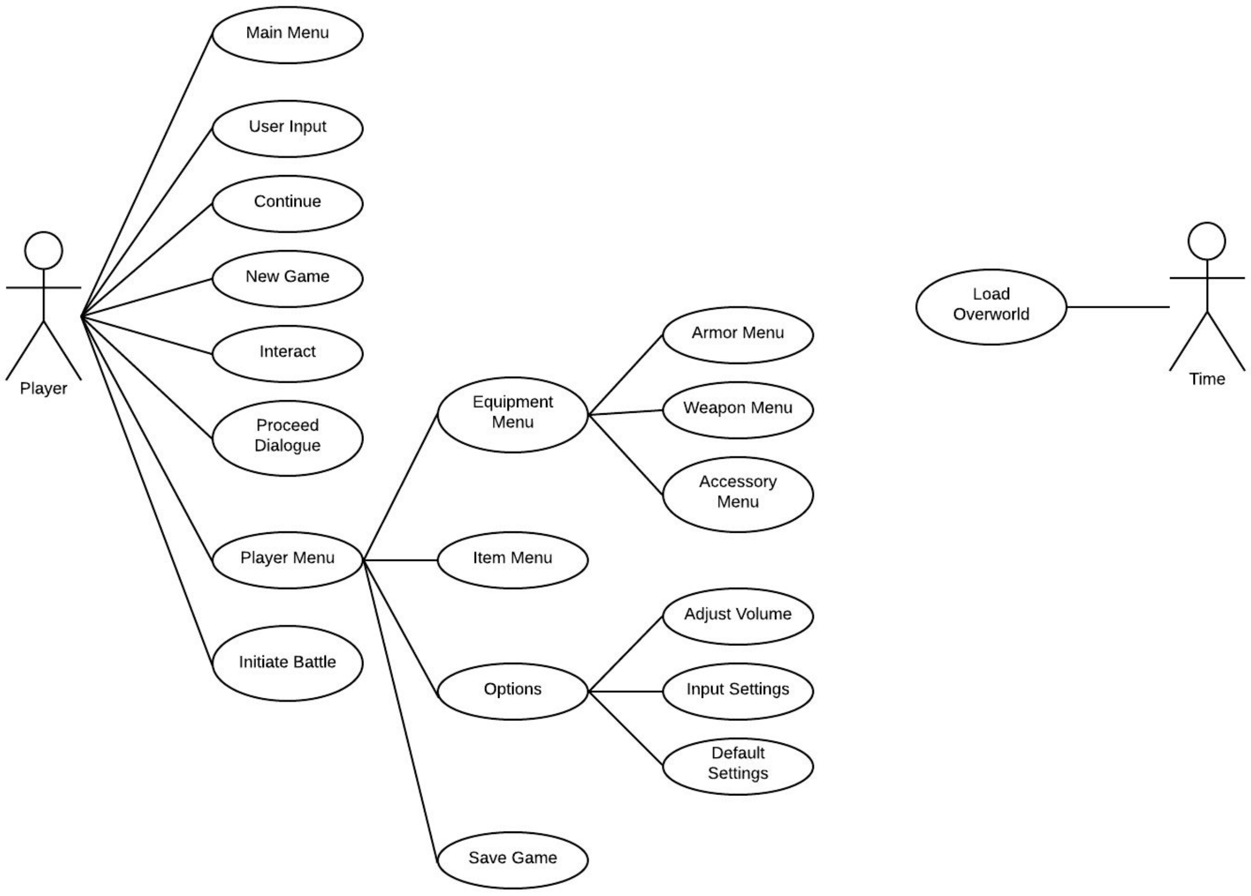


Figure 1: A use case diagram for Dragonlord Chronicles.

Use Case Descriptions

Identifier	Name	Description
UC01	User Input	The game will detect input from keyboard or other game controller. This will be used to navigate menus and control the main character during gameplay.
UC02	Continue	Finds and loads a previously saved game from a file and allows the player to continue playing their game from the state it was in when it was saved.
UC03	New Game	If a save file exists, it will be deleted. A new file that starts from the beginning of the game will be created.
UC04	Interact	The player may interact with different virtual objects and characters within the world. The details of the interaction depend on the object or character.
UC05	Proceed Dialogue	When the interact button is pressed near certain NPCs, a dialogue box will appear. Pressing the same button will cause the next lines to appear until that dialogue script is exhausted..
UC06	Player Menu	The player can pause the game and open a menu with several options, allowing the player to see their equipment, items, or dragons, save their game, or change their options.
UC07	Equipment Menu	This menu shows all of the player's equipable items with submenus for each type of equipable item. It also provides the stats for the players equipped items.
UC08	Armor Menu	This menu displays each armor that the player is carrying. Armor is used to protect the player from attacks. The player may get a description, drop, or equip any of the armors in this menu.
UC09	Weapon Menu	This menu displays each weapon that the player is carrying. Weapons are used for fighting enemies. The player may get a description, drop, or equip any of the weapons in this menu.

UC10	Accessory Menu	This menu displays each accessory that the player is carrying. Accessories can augment the player's abilities. The player may get a description, drop, or equip any accessories in this menu.
UC11	Item Menu	This menu has all items that the player is carrying. Unlike equipment, these cannot be equipped but they can still be used in battle or in the overworld.
UC12	Options	This menu allows the user to adjust settings for the game's input controls, audio levels, and window size.
UC13	Adjust Volume	Allows the user to change the volume of the game's sound effects and music.
UC14	Input Settings	Allows the user to choose if they want to play the game with a keyboard device or some other game controller.
UC15	Default Settings	This resets all of the audio and input settings to their default configurations. These match the settings as they are when a new game is first loaded.
UC16	Save Game	The player's current progress and state are saved to a file in the system's memory and can be loaded at a later time.
UC17	Main Menu	This is the initial scene in the game. It displays buttons that allow the player to go to the options menu, continue a previously saved game, or start a new game.
UC18	Load Overworld	Upon loading a saved game, the section of the overworld that the player saved in and the player character's coordinates will be loaded. In the case of a new game, the starting section and coordinates will be used.
UC19	Initiate Battle	When the player encounters an enemy in the overworld, a new scene will load and the player will have a turn-based battle.

Extended Descriptions

Use Case: User Input
Tag: UC01
Actors: Player
Preconditions: 1. The player has their preferred input device (keyboard or controller). 2. The input device setting in the options menu is set to the player’s preferred input device. 3. The game has been loaded.
Events: 1. The player presses a key or button recognized by the game. 2. The game responds based on the button pushed. a. If a directional key or button is pushed, the character or cursor (depending on the game’s state) will move in the indicated direction. b. If the “select” key or button is pushed, the character will interact with whatever is in front of it, or the element the cursor is pointing to will be selected, depending on the game’s state. c. If the “menu” key or button is selected, the player menu will open. d. If the “exit” key or button is pushed, the current menu will be closed.
Postconditions: 1. The player character’s or cursor’s position changes, an interaction with an object or character in the game world occurs, a selection is made, or a menu closes.

Use Case: Proceed Dialogue
Tag: UC05
Actors: Player
Preconditions: 1. The player has interacted with an object or a NPC or triggered a cutscene. 2. The current game screen shows a dialogue box.
Events: 1. The current line of dialogue appears in the dialogue box. 2. The player presses the “select” key or button.

Postconditions 1. The current line of dialogue disappears and the next line appears in the dialogue box, or the dialogue box closes if the current line is the last line.

Use Case: Player Menu
Tag: UC06
Actors: Player
Preconditions: 1. The game state is in the overworld. 2. The game state is not in a battle, shop, dialogue screen, or cutscene.
Events: 1. The player pushes the “menu” key or button. 2. The game world pauses; nothing in the overworld will move or act in any way, including the player character. 3. The player menu appears on the screen, with several different options that allow the player to see information about their character and inventories and modify them in various ways, open the options menu, or save their game.
Postconditions: 1. The game world is paused except for the menu. 2. The player menu is open. 3. The player can traverse the player menu with a cursor by pushing the directional keys or buttons. 4. The player can make a selection in the menu by pushing the “select” key or button. 5. The player can exit the menu and unpause the game world by pushing the “exit” button.

Requirement Traceability Matrix

	FR01	FR02	FR03	FR04	FR05	FR06	FR07	FR08	FR09	FR10	FR11	FR12	FR13	FR14	FR15
UC01	x	x	x	x		x	x								
UC02	x														
UC03	x														
UC04		x	x	x	x									x	x
UC05		x										x		x	x
UC06						x					x				
UC07						x					x				
UC08						x					x				
UC09						x					x				
UC10						x					x				
UC11						x									
UC12						x	x		x						
UC13						x	x			x					
UC14	x	x				x									
UC15						x	x								
UC16						x									
UC17	x														
UC18								x						x	
UC19		x	x										x	x	x

Changes in Project Design

We created a public static class called “Global Flags”, which will make transitioning between scenes easier. Instead of pushing data to the StateManager so the correct state can retrieve the data, data will be sent to the GlobalFlags class and then pulled by the appropriate system when needed. We also added a CombatManager class, which will handle the interactions between the player and the in-game enemies.

Updated High and Medium Level Design

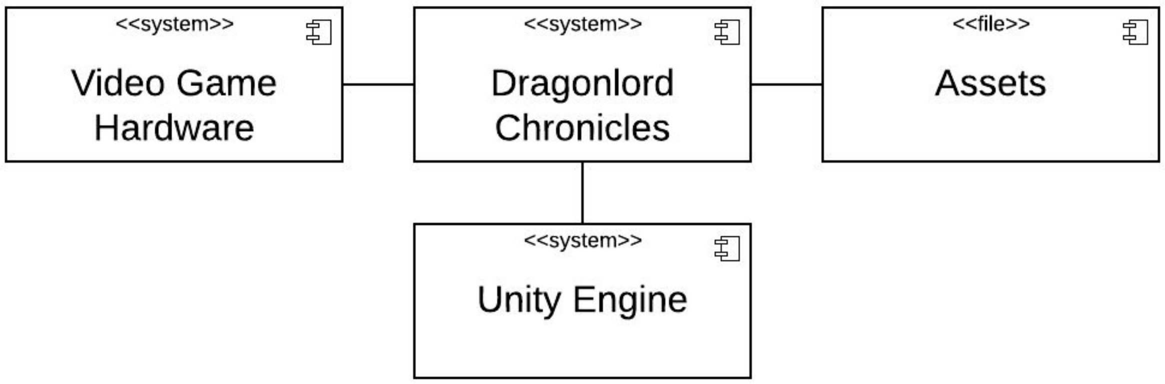


Figure 2: A component-based system level diagram of Dragonlord Chronicles’ structure.

Figure 2 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 MonoBehavior 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.

Class	GlobalFlags
Method	GetCurrentOverworldScene
Visibility	public
Return	string
Parameters	void
Description	Returns the name of the current overworld scene that should be loaded so the correct overworld scene can be loaded.

Class	GlobalFlags
Method	SetCurrentOverworldScene

Visibility	public
Return	void
Parameters	string
Description	Sets the name of the current overworld scene, so the game can load the correct scene during transitions (i.e from battle to overworld or overworld section to another section).

Class	GlobalFlags
Method	GetCurrentBattleScene
Visibility	public
Return	string
Parameters	void
Description	Returns the name of the current battle scene that should be loaded.

Class	GlobalFlags
Method	SetCurrentBattleScene
Visibility	public
Return	void
Parameters	string
Description	Sets the name of the current battle scene that should be loaded so the correct battle scene can be loaded during transitions.

Class	GlobalFlags

Method	SetCombatManagerFlags
Visibility	public
Return	void
Parameters	EntityData, EntityData, EntityData
Description	Sets the player, enemy, and dragon data in a combat scene.

Class	GlobalFlags
Method	GetCombatManagerFlags
Visibility	public
Return	void
Parameters	out EntityData, out EntityData, out EntityData
Description	Retrieves the player, enemy, and dragon data so it can be used by the CombatManager.

Class	CombatManager
Method	TakeRound
Visibility	public
Return	void
Parameters	void
Description	This function will act out the actions input by the player and the enemy classes.

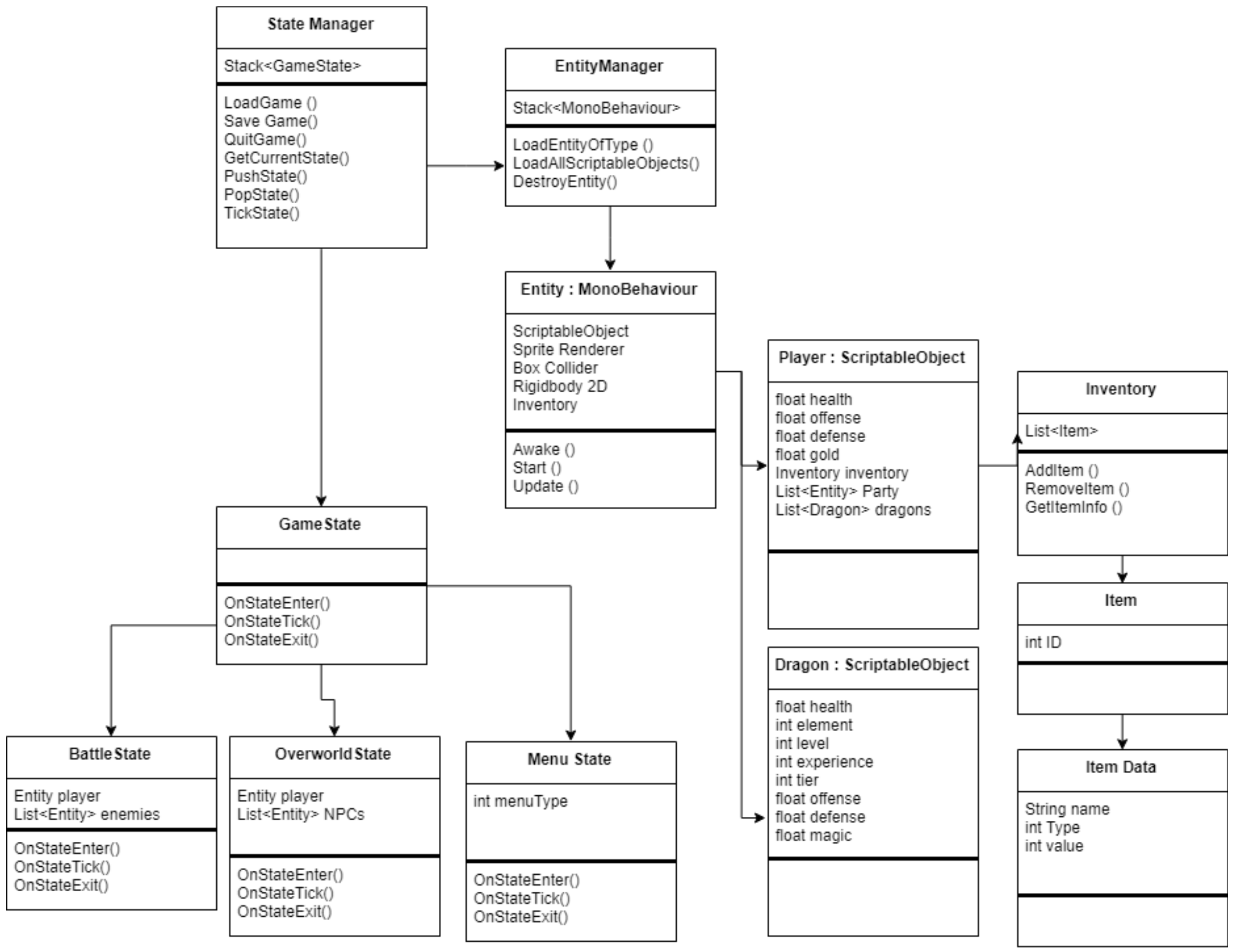


Figure 3: Class Diagram. This shows how the custom data structures are related to each other.

Detailed Design

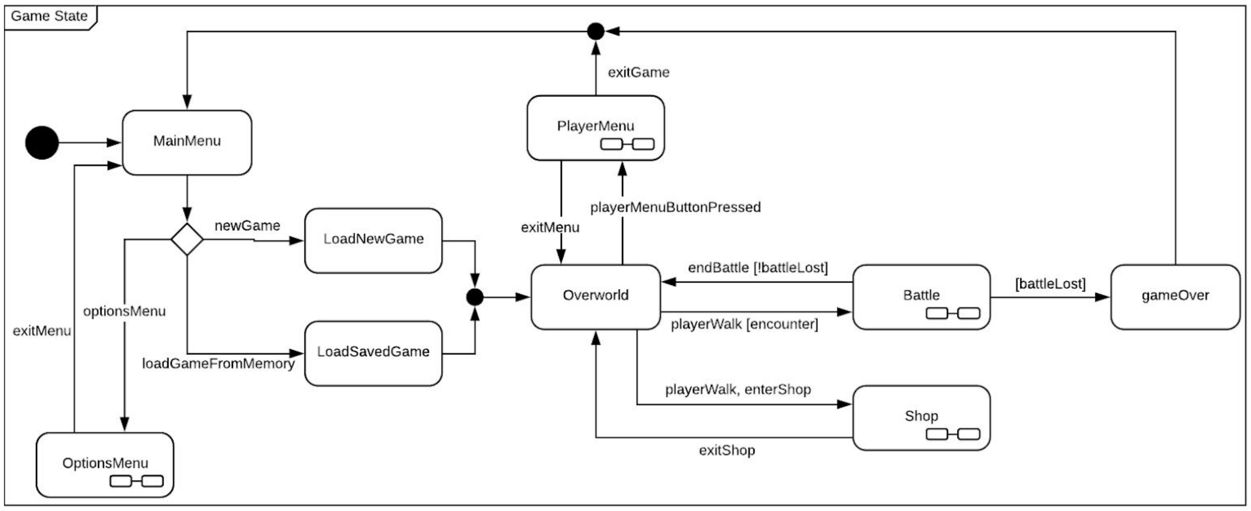


Figure 4: A state chart showing the flow of game states in Dragonlord Chronicles. A player begins at the main menu and can choose to start a new game or load a saved game. Then they can explore the overworld, use their menu, shop, and enter battles. If they choose to quit the game from the player menu or if they lose a battle, the game will go back to the main menu.

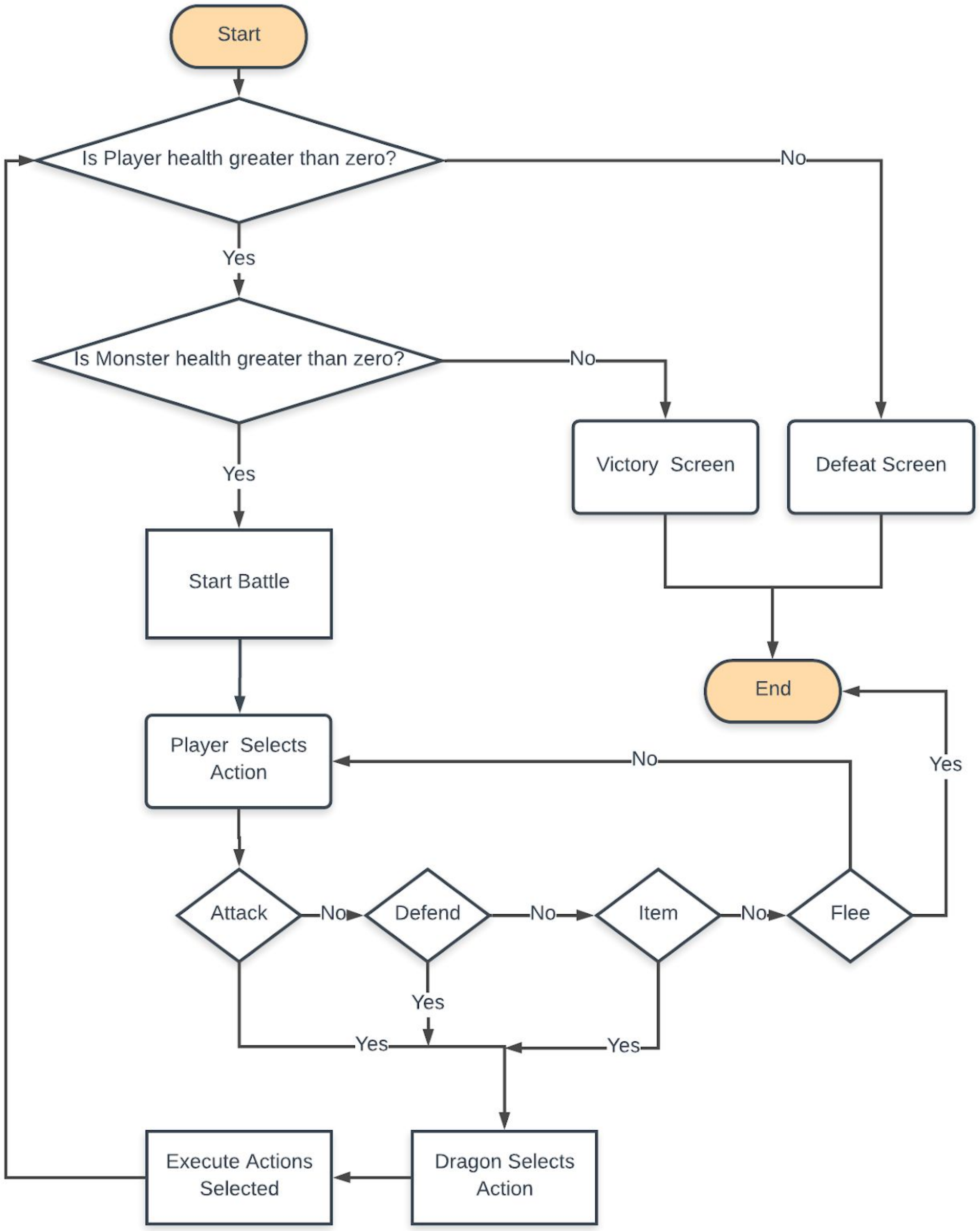


Figure 5: A flowchart giving an overview of the game's battle system.

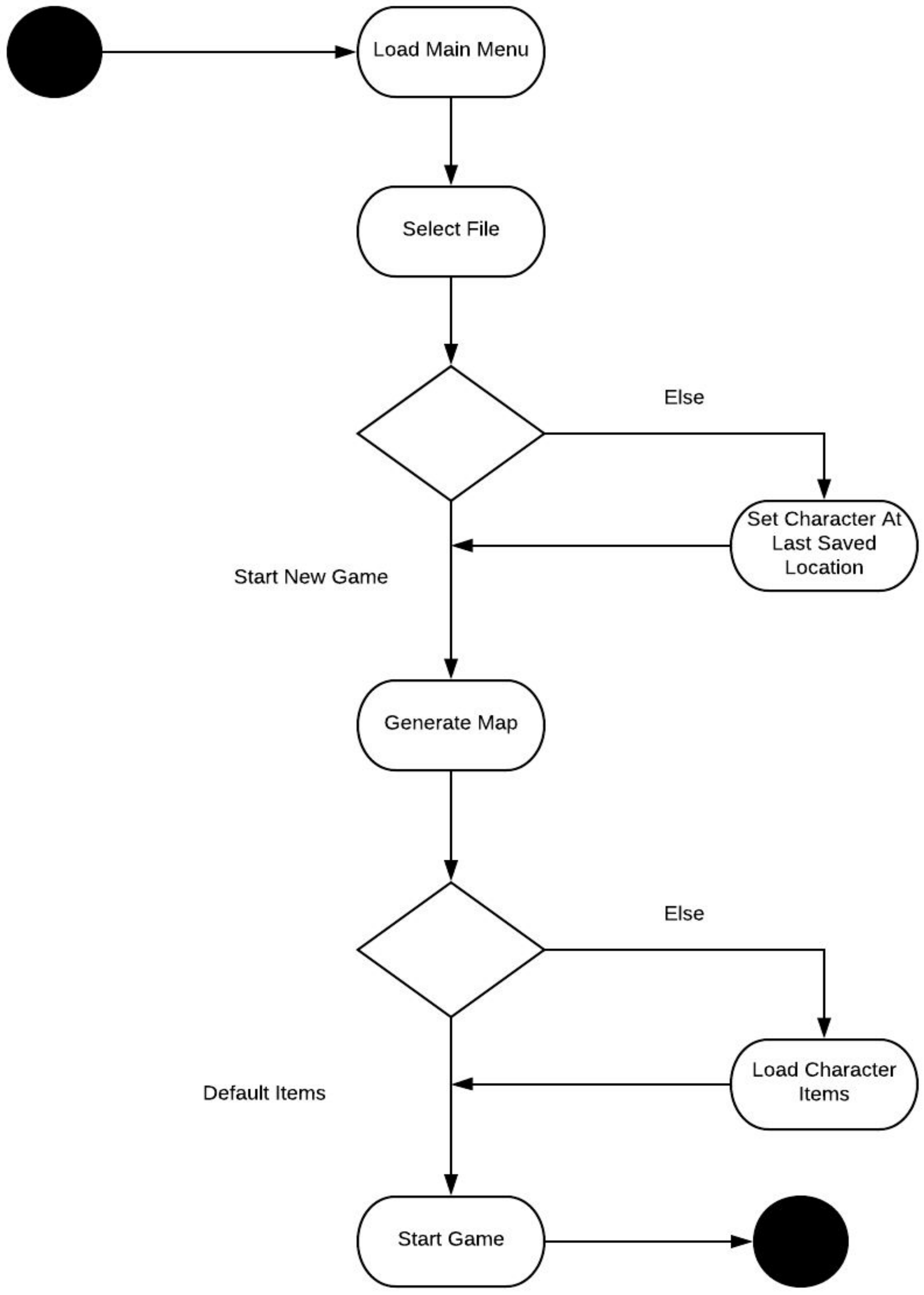


Figure 6: An activity diagram that shows the process of starting and loading a game.

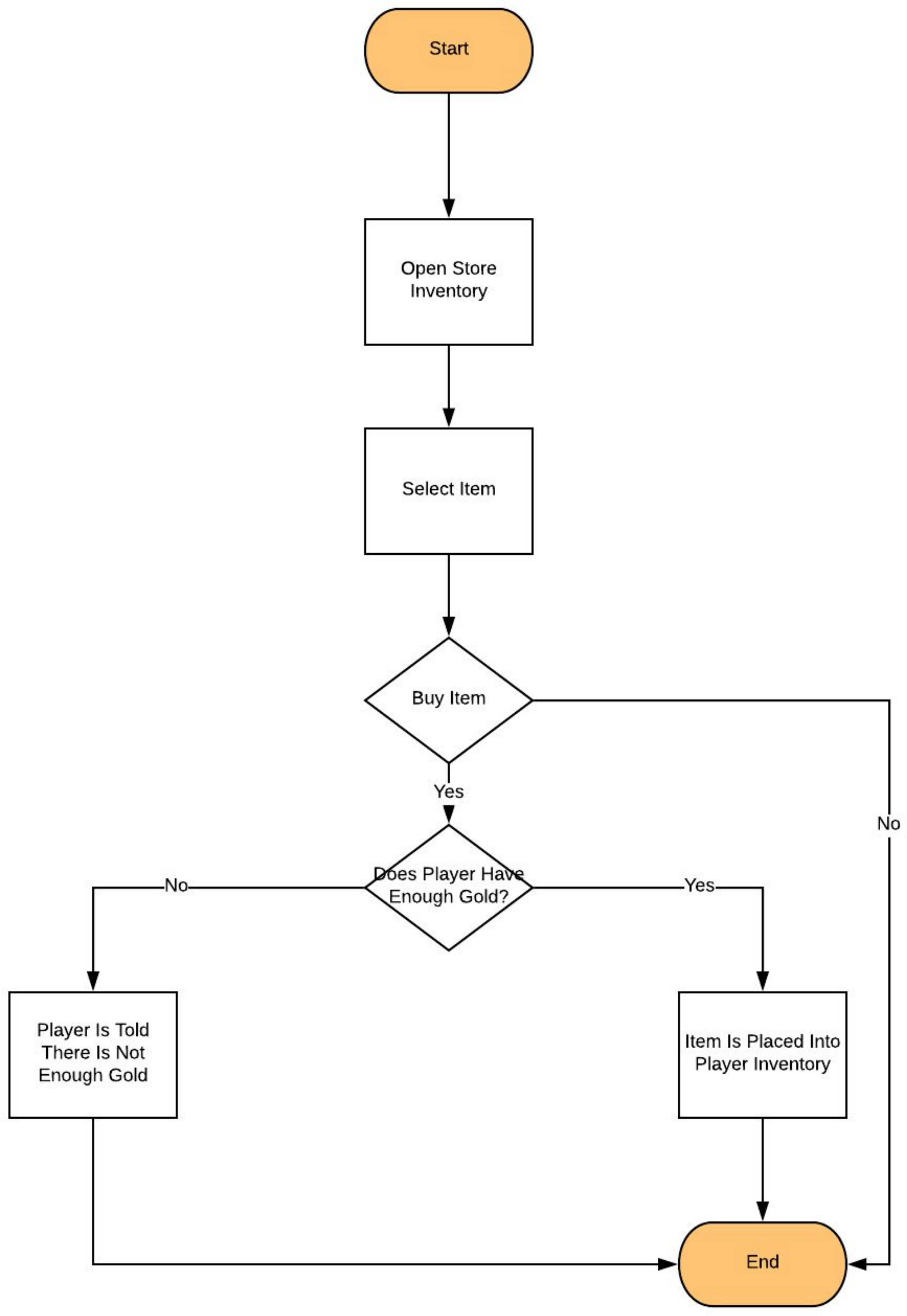

Start
Open Store Inventory
Select Item
Buy Item
Yes
No
Does Player Have Enough Gold?
No
Yes
Player Is Told There Is Not Enough Gold
Item Is Placed Into Player Inventory
End

Figure 7: A flowchart giving an overview of the game's shop system.

Updated UI Design

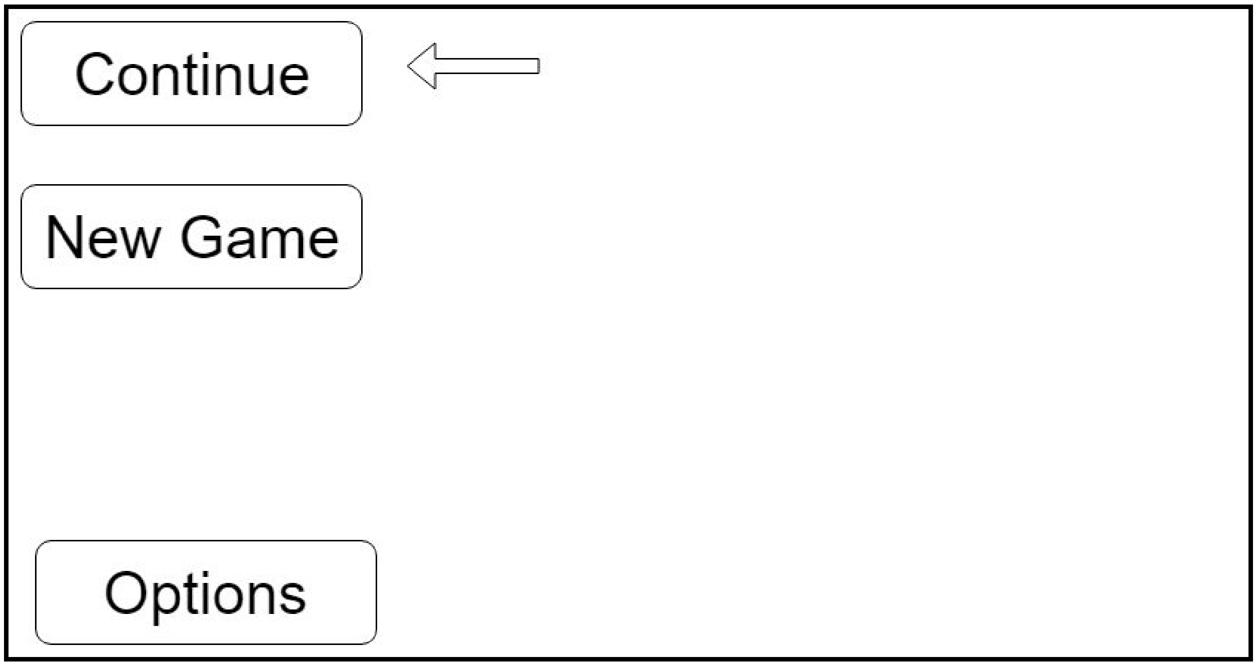


Figure 8: Main Menu. The screen shown when the game starts.

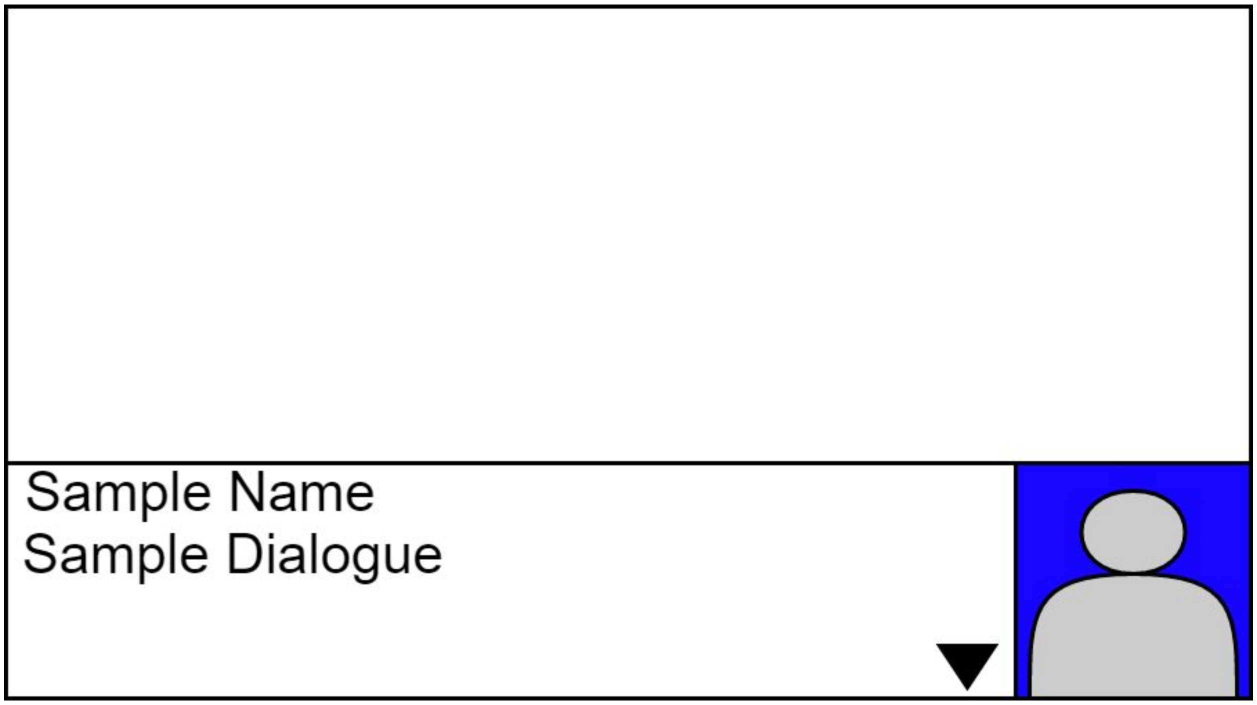


Figure 9: NPC Dialogue Menu. When the player presses the action button near an NPC, they will speak to the player.

.

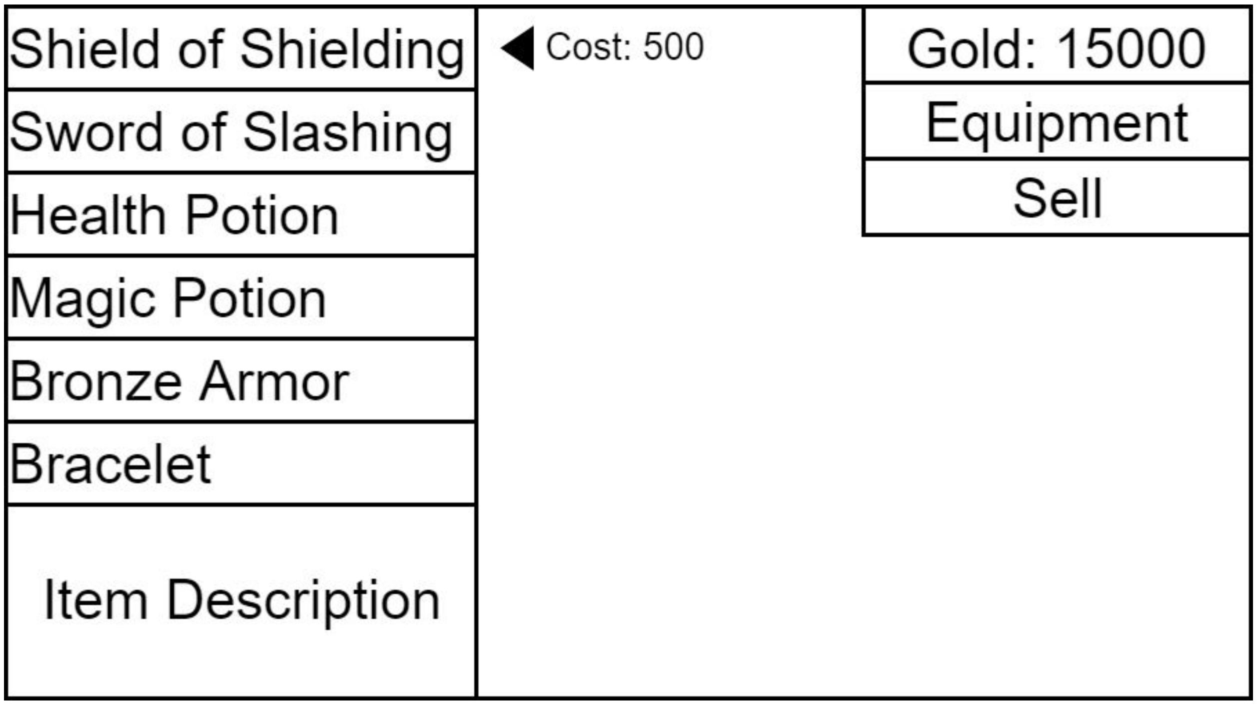


Figure 10: Shop Menu. When the player talks to a shopkeeper, they may purchase items for their quest.

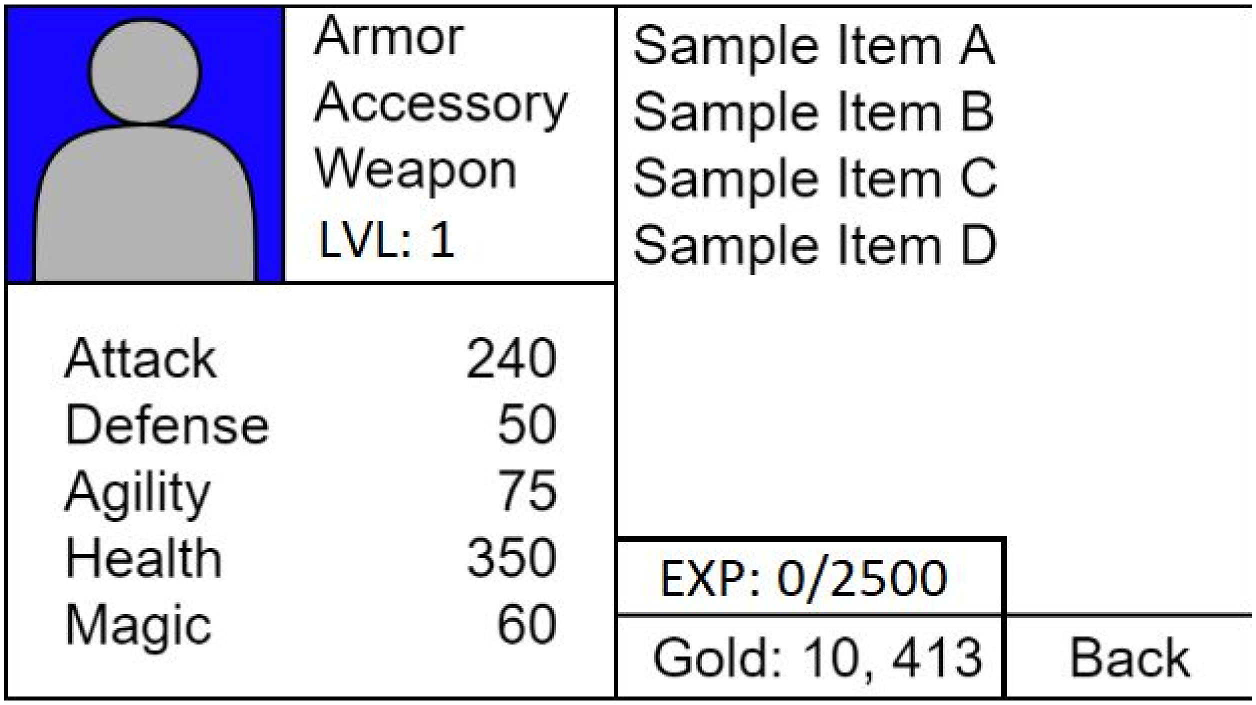


Figure 11: Equipment Menu for when the player is managing their weapons/armor, taking potions.

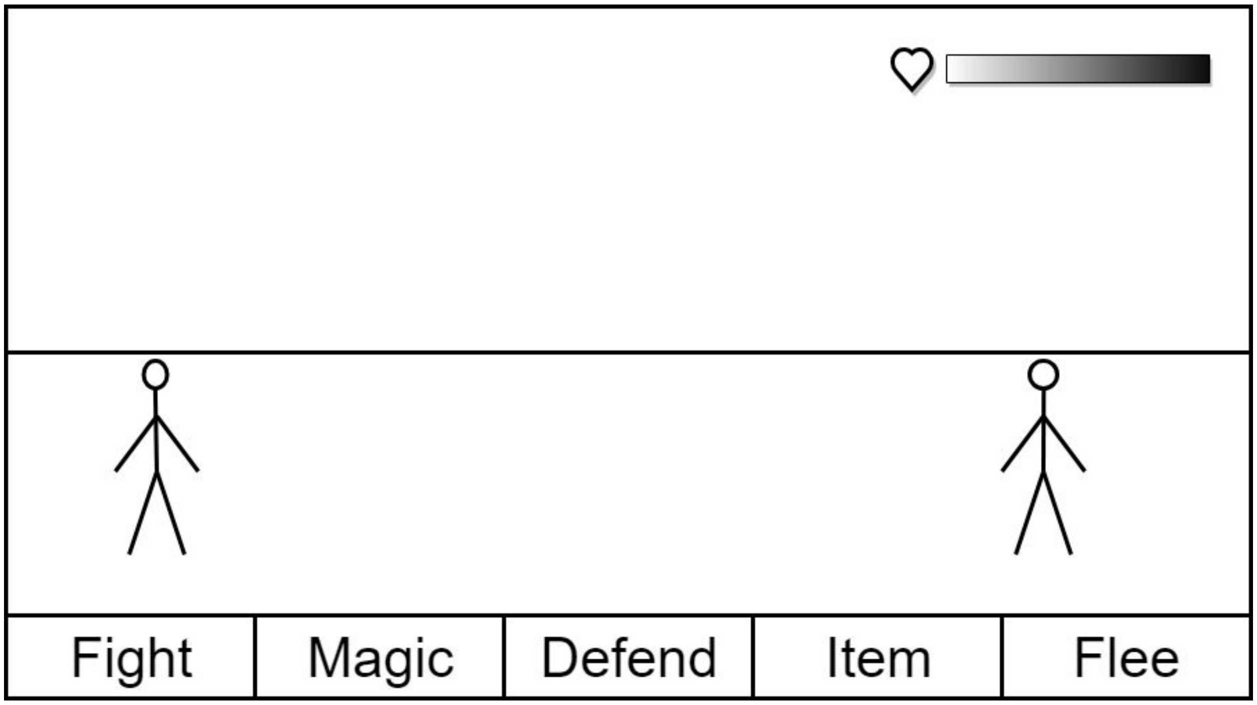


Figure 12: Battle Menu. When the player encounters an enemy, they will enter a scene where they can make turn-based decisions for battle.

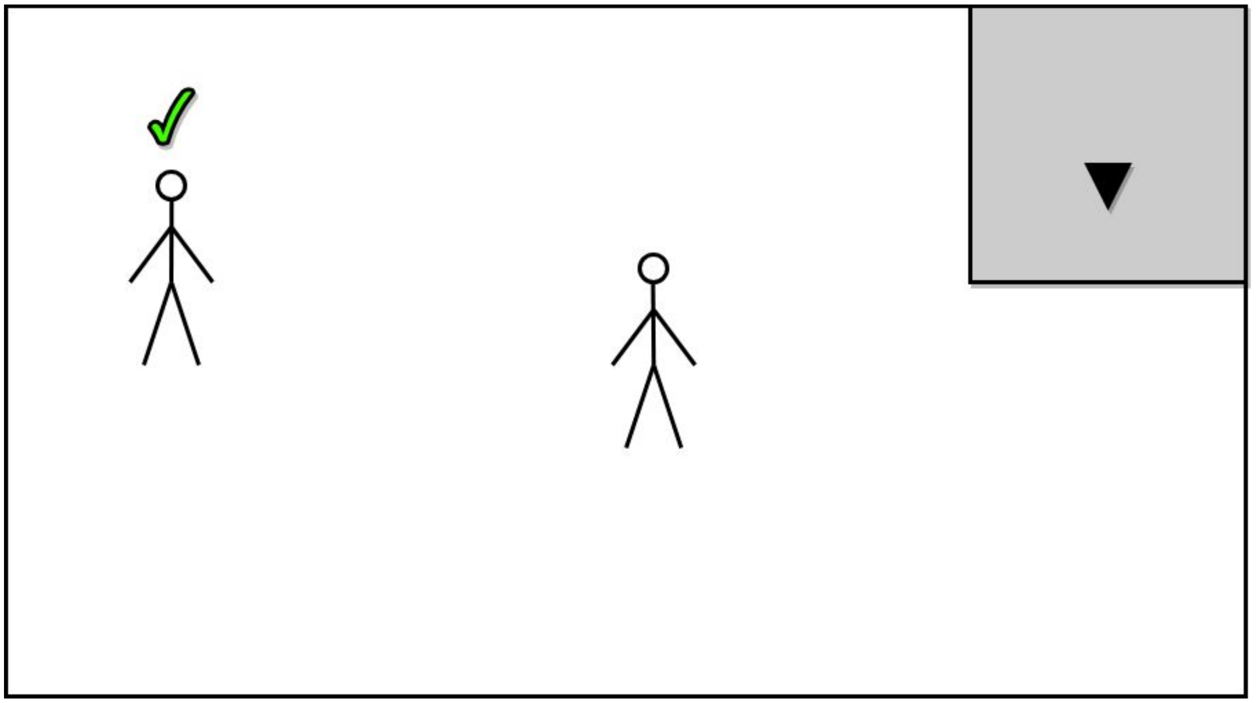
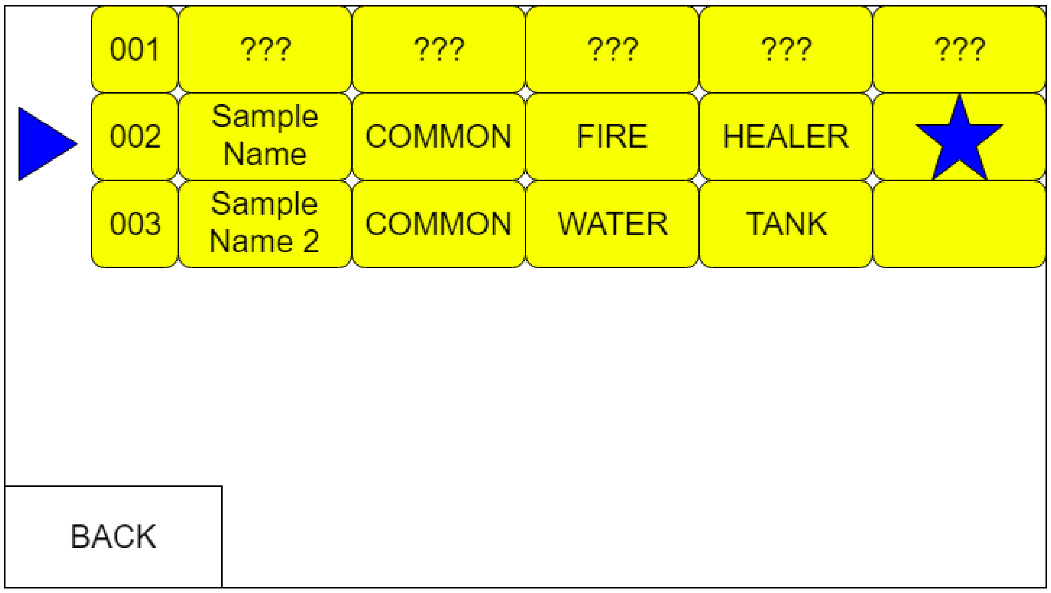


Figure 13: Overworld Screen. During overworld exploration, the UI will be very minimal only displaying visual cues for NPCs with quests and a minimap.



001	???	???	???	???	???
002	Sample Name	COMMON	FIRE	HEALER	★
003	Sample Name 2	COMMON	WATER	TANK	

Figure 14: Dragon Encyclopedia. This will detail the different types of dragons encountered over the course of the game. Captured dragons will be shown with a star, so the player can differentiate encountered dragons and captured dragons.

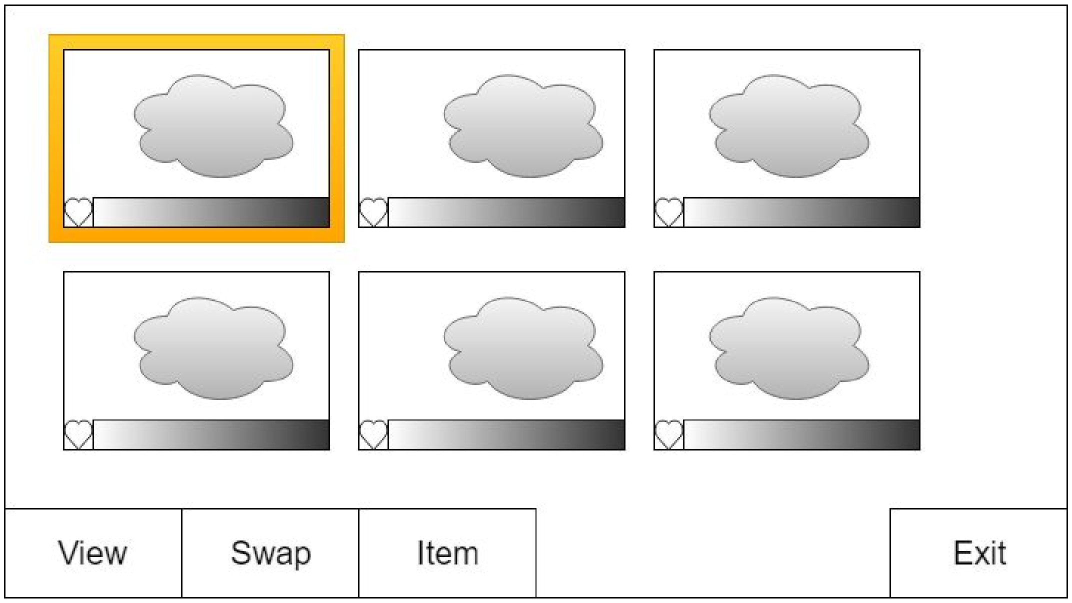


Figure 15: Dragon Party. This screen shows general information about the current set of dragon's in the player's party, with options to use an item on the dragons and to swap them with a different character.

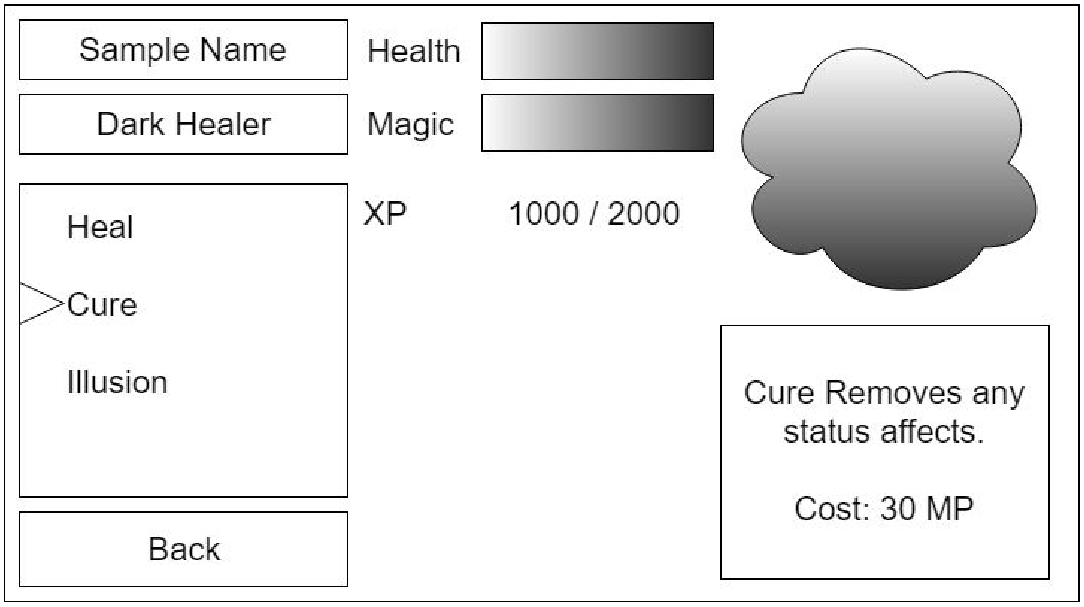


Figure 16: Dragon Stats. When the player selects one of their dragons to view, they will enter this menu, which shows detailed information about the current dragon.

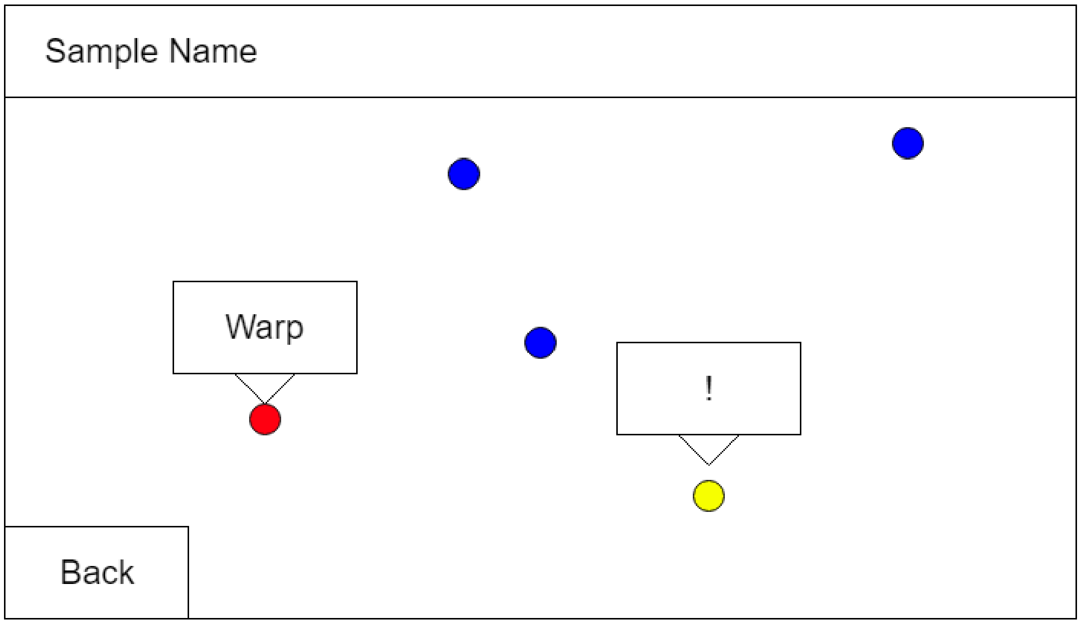


Figure 17: World Map. As the player discovers locations, they will unveil more of the world map. The player may skip travel to different locations by clicking on the map.

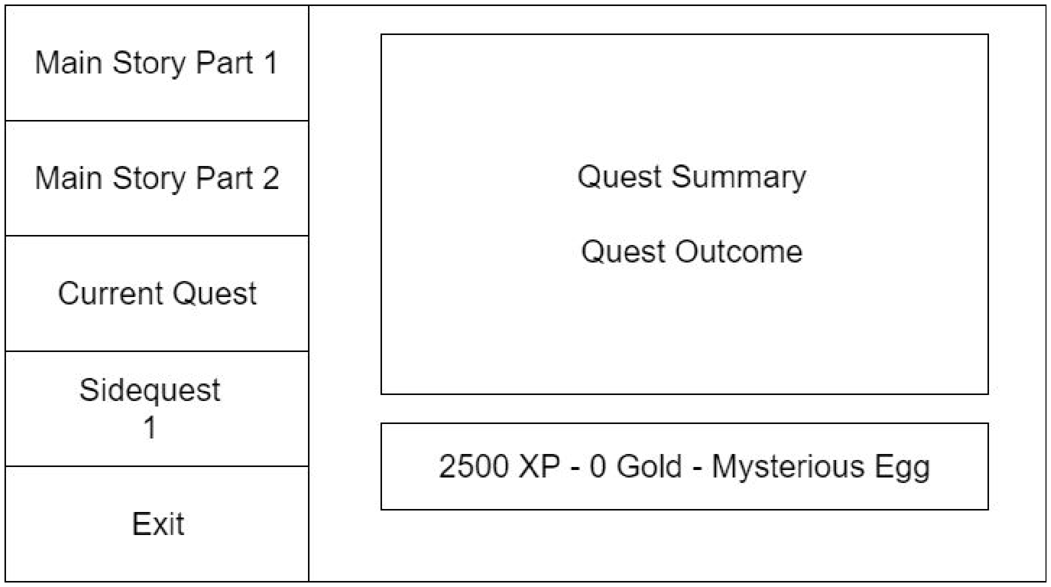


Figure 18: Quest Journal. As the player completes the main story, major events will be logged into this journal. The data shows will be the quest reward, quest status (completed, failed, in-task), a summary of the quest, and its outcome.

Glossary

Component	The specific attribute(s) that may be attached to GameObjects.
Equipment	Virtual items that can be picked up by characters.
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 the 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 scriptable component in Unity. This may be used to elicit game-specific behavior on a GameObject.
NPC	Non-Player Character. Any character that cannot be directly controlled by the player.
Overworld	The area in the game that connects all of the locations.
Player	The end user of the game.
Pixel Art	Minimalistic artwork where the image is comprised of a small pixel resolution and a few colors per image.
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
Tilemap	A 2D grid of images that are the same distance apart.
Top-Down	A game where the player’s perspective 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, developed by UnityTechnologies.
UI	A User Interface (UI) allows the player to interact with the software.
Video Game	A game played by manipulating images displayed on a monitor or television.

Engineering Standards & Technologies

Standards:

ISO/IEEE 12207: Systems and Software Engineering: Software Lifecycle Processes. This standard establishes a set of processes for managing the lifecycle of software. We will use this to satisfy our target audience by creating a game that meets their expectations.

ISO 9000: Series of Standards refers to the seven quality management principles. We will use this standard to effectively manage our project.

UML: Unified Modeling System is the method for designing the architecture of software. This will be used to design and document the models that will be used in our game’s systems.

Technologies:

Pyxel Edit. This software is designed to create minimalistic pixel art. It will be used to create the sprites, tiles, and animations for our game.

UnityEngine. This software is a game engine which will be used to create the game.

Visual Studio. This software is an IDE that will be used with Unity to create C# scripts.

C#. This is a programming language, designed to be a variant of C but with a garbage collector like Java. This will be used to create scripts.

Unity Collaboration. This is a service provided by Unity that allows teams to work together. We will use this to track progress and keep our work synchronized with each other.

List of References

Book:

Introduction to Game Design, Prototyping, and Development Second Edition (Jeremy Gibson Bond)

Jeremy Gibson worked as both a professor of game design and a game developer. He wrote this book to teach others how to prototype a game and develop it into a fleshed-out, playable product. The book emphasizes using Unity and C#, but the skills can be applied to making a game in any language and any engine.

Reference Articles:

Game Development “Tales of Mamochi” with Role Playing Game Concept Based on Android

This journal details the development of the Role-Playing Game (RPG) “Tales of Mamochi”. This game features a variety of different mechanics such as Person vs. Environment combat, crafting, farming, cooking, and questing. This game is designed to be fun, so players can enjoy it at their leisure.

Critical Success Factors to Improve the Game Development Process from a Developer's Perspective

This journal details the factors that contribute to developing quality software. This study was created after observing that there is a lot of pressure on upcoming developers to build a game that meets expectations. The study also discusses how developers can remain competitive in a growing industry and handle the pressure that comes with being a software developer.

Game Development Software Engineering Process Life Cycle: A Systematic Review

This journal discusses how game development requires synthesis between sound, art, input systems, artificial intelligence, and other factors. This study assesses the state-of-the-art research on the game development process and highlights areas that need to be researched and refined further, so the process may be improved.

Practices and Technologies in Computer Game Software Engineering

This journal discusses the techniques and technologies that are used when developing video games. It also mentions how game developers, regardless of their focus on entertainment or non-entertainment applications, each share an interest in the best methodology to engineer software.

Websites:

GDC 2017 - Hitchhiker's Guide to Rapid Prototypes
https://www.youtube.com/watch?v=sYWkiv1hTPM

Game Developers Conference is the gaming industry's largest global event where game developers discuss very helpful tips to both novice and expert developers to improve their games. This talk discussed how to quickly build a prototype that tests the core functionality of a game. Some of the helpful tips were: focusing on urgent goals and creating the minimum viable interaction.

Unity User Manual: https://docs.unity3d.com/Manual/index.html

The official Unity manual contains descriptions and examples on all the features that Unity provides. This includes tutorials on creating 2D games, scripting in C#, creating audio, using the UI system, and using the physics system. It also has documentation on how to use the Unity API. Unity's manual is well-organized, so it will be beneficial for learning how to implement certain mechanics into our game.

Brackeys: http://brackeys.com/

The Brackeys contains many tutorials for creating a variety of different types of games in Unity. He has tutorials on platformers, Pong, and a 3D RPG. He has also done tutorials on more specific subjects such as using ScriptableObjects. The Brackeys website contains free assets that could be used for prototyping our game. The only drawback is that, since Brackeys has been doing tutorials since 2012, some of the tutorials use Unity functions that have since been deprecated.

quill18creates YouTube Channel: https://www.youtube.com/user/quill18creates

This YouTube channel contains many helpful tutorials for creating fleshed out games in Unity opposed to creating one-off games. For example, his Base-Building Game tutorial goes in-depth to creating tile systems, loading files from disk, AI Pathfinding, and other features to create a base-building game. He has also done tutorials for creating a Civilization-style game in a hexagonal grid. These tutorials will help us understand how to create a game with a larger scope than the other tutorial sites provided.

Contributions of Team Members

Sean spent about three hours updating the List of References, adding a few terms to the glossary, and explaining a few of the classes defined in the High Level Design section. He also wrote the Engineering Standards and Technologies section.

Jonathan spent approximately an hour and a half rewriting the high level business requirements, updating functional and non-functional requirements, and added the recent project changes section.

Christine spent about an hour and a half summarizing the specification changes, updating the Use Case Diagram and the detailed use case descriptions, and making various formatting fixes.

Ryan spent about an hour updating the UI Design portion and editing the paper.