Stories of the Gorbapork Nebula

Nick Bucher, Wood McGowan, and Kaden Goodell
9/22/16

Document History

Name	**Reason**	**Date**
Nick, Kaden, Wood	Initial Draft	9-16-16

Table of Contents

Part 1. Introduction

1.1: Motivation

Our motivation behind choosing to develop a video game arose from a collective interest in video games, programming, and a desire to increase the visibility of video game developers at The University of Alabama. In following this motivation, we plan to produce a viable product that is fun to play and present to others. Furthermore, we intend for our project to showcase the Unity engine as an empowering tool for small teams developing games. We hope that our project will provide a concrete example of a fun, complete video game produced by students at The University of Alabama.

1.2: Purpose

The purpose of this project is to create a two dimensional role-playing video game that delivers an immersive and engaging narrative experience for players.

1.3: Scope

The completed game will have a start point and an end point, while also having optional content for the player to explore and experience. The game will have key role-playing elements such as an explorable environment, realistic and interactive characters, items which affect the player's stats in battle sequences, and items which play different roles in the story. The scope of this project will be limited in that the goal will be to squeeze as much fun gameplay and content into a reasonably small amount of playtime. In this way, the game should be able to be experienced and understood fairly quickly, and our project should be completed on schedule.

1.4: Goals

The goals of this game development project include producing a high functioning interactive graphical environment to represent the game world, and creating a highly reliable user control scheme that allows the user to seamlessly explore this environment. In creating an interactive world, our goal is to develop several unique and entertaining non-playable characters for the user to engage with, while also providing enjoyable dialogues and story arcs for the player to experience. The finished game should also provide a fun and well-designed battle system that functions with coherent and effective game mechanics, and that allows the user to be challenged and improve their skill.

1.5: Glossary

- **Game Engine:** A software development framework for creating video games.
- **Unity:** A game engine that supports development across many platforms with the C# programming language.
- **NPC:** A non-playable character in a video game
- **Assets:** Art models, audio effects, animations, or any element that is created outside of the game development environment in order to be utilized during development.
- **FPS:** The number of frames rendered every second.
- **Sprite:** A two dimensional graphical image.
- **Game State:** The state of all the game's elements and parameters at any given time during gameplay.
- **Colliders:** An area of space on the game screen that represents the shape and form of a game object in order to organize physical collisions on the screen.
- **Map:** The entire graphical representation of objects and space within the loaded scene.
- **GameObject:** A generic object in the Unity game engine.
- **The Main Game Loop:** The looped code sequence which updates the game each frame. In Unity, each GameObject calls its Update function every frame.
- **Coroutines:** A type of function in the Unity framework which acts as a new thread and can run asynchronously alongside the main game loop.
- **Prefab:** A GameObject which has been constructed in the Unity editor and saved to a file. This prefab object can then be referenced later in code.
- **Top-Down Perspective:** A graphical paradigm in which the the player object and the immediate surroundings are portrayed from above.
- **Shooter:** A broad genre of gameplay where the primary player activity is shooting projectiles at enemies.
- **360° Shooter:** A sub-genre of video games where the player can rotate 360° in order to shoot projectiles at enemies that approach from all directions. Usually implementing top-down perspective.
- **RPG:** A genre of video games characterized by narrative arcs, character dialogue, and world exploration in which the player assumes a fictional character's role in a fictional universe.
- **Battle System:** The algorithms, rules, and mechanics that determine the statistical system for a game's combat gameplay.
- **Play-Through:** A single completion of a video game's main story from start to finish.
- **Spawn:** The initialization of a graphical game object on the current map.
- **Replayability:** The degree to which a game is enjoyable to play through more than once.
- **Critical Path:** The sum of all the mandatory actions and events that must take place for the player to reach the end of the game from the initial starting point.

Part 2. Description

2.1: General Description

This project is a science fiction story based two dimensional role-playing video game. It is composed of a main storyline and multiple optional side quests. The main storyline begins with the main character, a space cadet in training, waking up in their room in a space station. The main character then does optional side quests by talking to different characters and exploring the space station, and then eventually goes to the space cadet training room to continue the main storyline. After doing the space cadet turret battle training, the main character then returns to their room and goes to sleep. Upon waking up, the main character realizes that their dog is missing. In searching for their dog on the space station, the main character may do more optional side quests. When the player finally finds their dog, the main storyline concludes with the space station being attacked by aliens. If there is sufficient development time, the Player will experience different endings based on which side quests they complete.

2.2: Gameplay Description

The gameplay entails exploring the current map, interacting with NPCs, discovering game items, and battling enemies. The battle system gameplay is both designed as a turret based shooter, as well as a top-down, 360-degree shooter where the player aims and shoots projectiles as enemies approach from all directions. Another component of the gameplay is interacting with the player statistical system in order to customize and improve a character's attributes, which is connected to the basic reward system for engaging in the game's battle system. Some portion of gameplay which involve item discovery and character exploration are intentionally designated as optional game scenarios, which the player is free to experience during their play-through of the main storyline.

2.3: User Control Description

The project implements a two-dimensional, top-down perspective, in which the user moves the player object around the map and navigates game menus using a joystick touchscreen GUI. The user interacts with game objects in the environment and selects player options in menus using a touchscreen button GUI. These ergonomic GUIs allow the gameplay to be controlled on a smartphone seamlessly without any need for the user to gain practice or develop motor skills.

2.4: Device Requirements

The device which the game will be running on will need to be running a version of Android 2.3.1 (Gingerbread) or higher to function. The pixel dependent user interface of the game will be scaled relative to the pixel density of the device it is on during runtime, so the screen size is independent from the functionality of the game. However, since the game is being developed for a screen ratio of 16:9, any screen ratios which are significantly different than 16:9 may have scaling issues with the user interface. The device which the game will be playing on will also need to have a touch screen for user input, and a large enough pixel resolution for the text displayed on the screen to be readable.

Part 3. Functional Requirements

3.1: System Startup Requirements

1. The system will load the title menu upon game launch.
2. The system will allow the user to start a new game from the title menu.
3. The system will allow the user to load a saved game state from the title menu and the pause menu.
4. The system will initialize the start state of the gameplay sequence upon the user creating a new game.
5. The system will properly load the player's saved game state upon the user loading a game.
6. The system will display the functional joystick and button interfaces on the screen during gameplay.

3.2: User Control Requirements

1. The system will allow the user to move the player object with the on-screen joystick interface during gameplay sequences which involve exploration.
2. The system will allow the user to interact with interactive game objects by utilizing the touch screen button interface.
3. The system will allow the player to select text options from selection menus during the proper game states using the touch screen button interface.
4. The system will allow the user to shoot the selected player weapon with the touch screen button interface during the game's battle sequences.
5. The system shall allow the user to aim the player weapon using the touch screen while in the game's battle state.

3.3: Graphical Map Requirements

1. The system shall load all the game objects that make up the graphical environments in the proper locations upon the user starting a new game or a saved game.
2. The system shall render the game object colliders in the proper spatial positions throughout the user's gameplay.

3. The system will allow the user to access the proper map locations dependant on the user's game state.
4. The system shall render the separate map sprites properly such that sprites that move will interact properly with the correct layers, and multiple sprites do not overlap on the same layer.

3.4: Game State Requirements

1. The system will load the correct menus as the user interacts with the specific game objects corresponding to those menus.
2. The system shall transition to the game's correct battle state at the proper times as the user engages enemies.
3. The system shall spawn and eliminate enemy game objects properly as the user triggers the battle state.
4. The system shall calculate health and damage parameters properly for all items and enemies as the user engages the battle system.
5. The system will allow the user to pause the game using the graphical pause button.
6. The system will freeze the current game state upon the user pausing.
7. The system shall allow the user to load a game state, save the current game state, quit the game, or resume the game state from the pause menu.
8. The system will execute the player death sequence when user reaches zero health.
9. The system will execute the endgame sequence when the user reaches the proper game state.

Part 4. Non-Functional Requirements

4.1: Optimization Requirements

- There will be a limit to the number of sprites on the screen.
- Input will be guaranteed to be responsive on most machines through extensive testing and script optimization.
- Script optimization will include moving cpu heavy code out of the game loop to coroutines where possible, so that the cpu heavy code will run at fixed time intervals as opposed to running every frame update.

4.2: Documentation Requirements

- Project documentation will be created using Visual Studio's XML syntax. This syntax allows for instant generation of intellisense, which will speed up the development of the game significantly.
- Documentation will be written for every function where the name of the function does not entirely describe what the function does.

- Code will be commented significantly, at least at every point where the developer has to research what a code segment does.

4.3: Coding Standards and Reusability Requirements

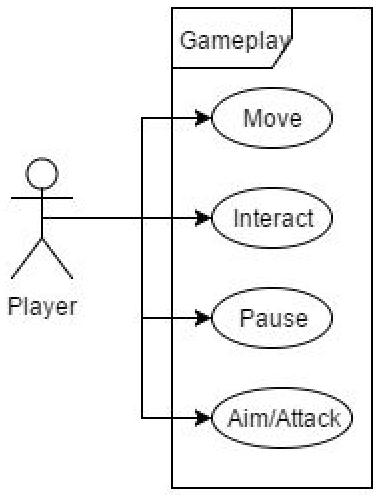
- The inversion of control design pattern will be applied by acquiring frequently used dependencies in a GameSceneController class in each scene, and referencing them through a single GameSceneController instance in each scene.
- Prefab GameObjects will be used to instantiate UI objects and other complicated objects, as suggested in the Unity documentation.
- If modules are identified as more code is written, they will be extracted to their own package as development continues. These extracted modules will be designed to be reusable in future projects.

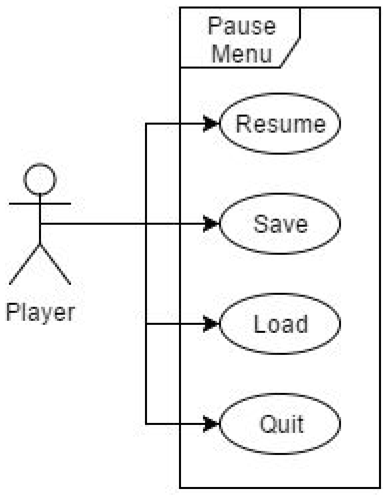
4.4: Project Backup Requirements

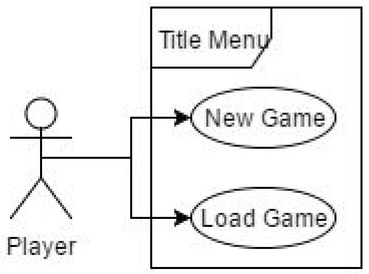
- This project is in a git repository hosted on bitbucket.org.
- Each group member has their own branch.
- Any changes to the repository will be committed frequently.
- If an error is made which makes the game unplayable, the game will be able to be rolled back to its last playable commit.

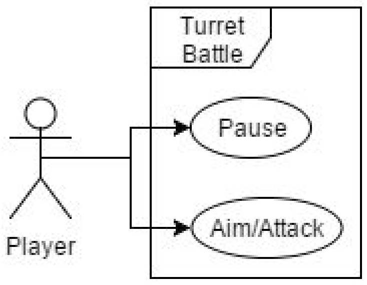
Part 5. Diagrams

5.1: Use Case Diagrams





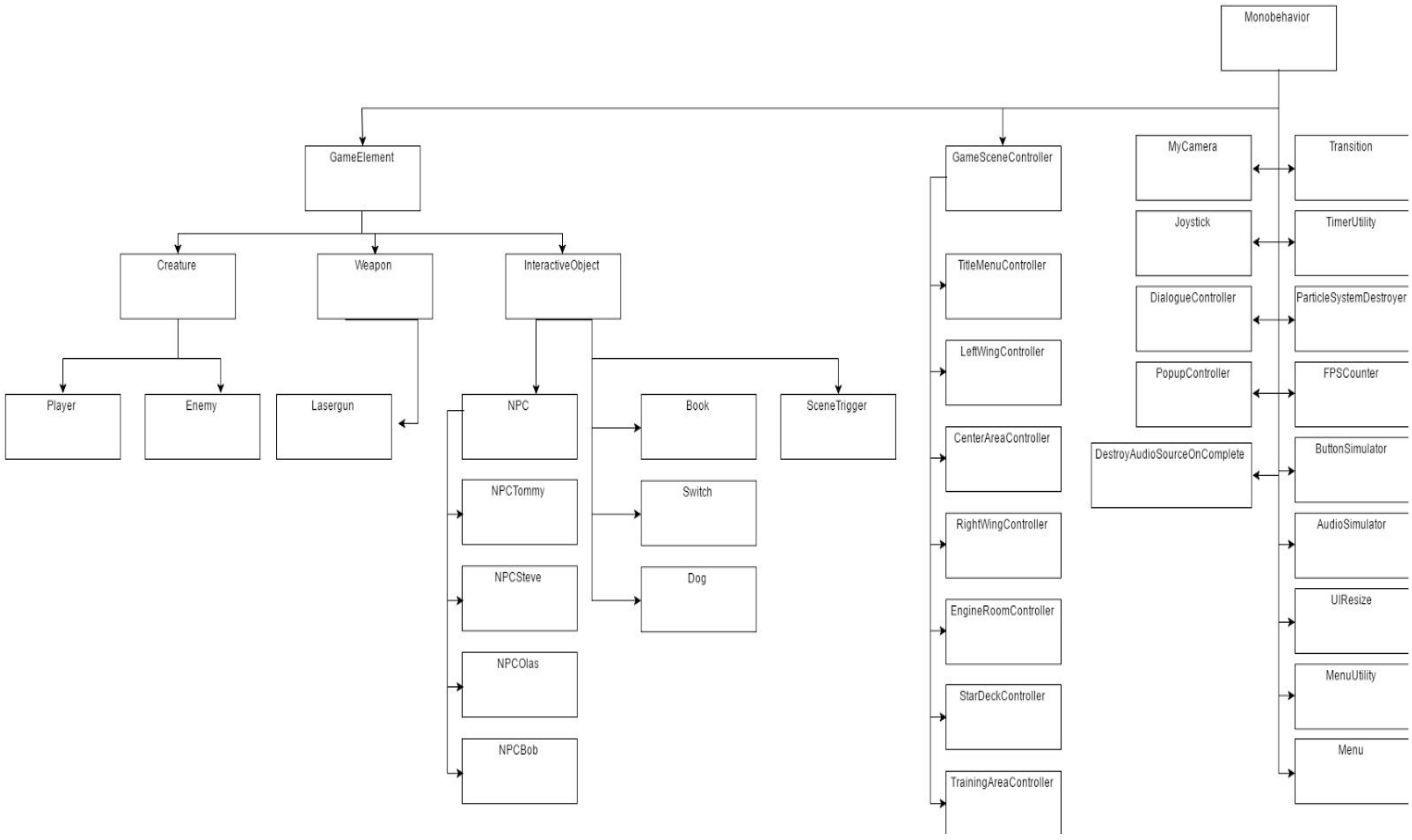


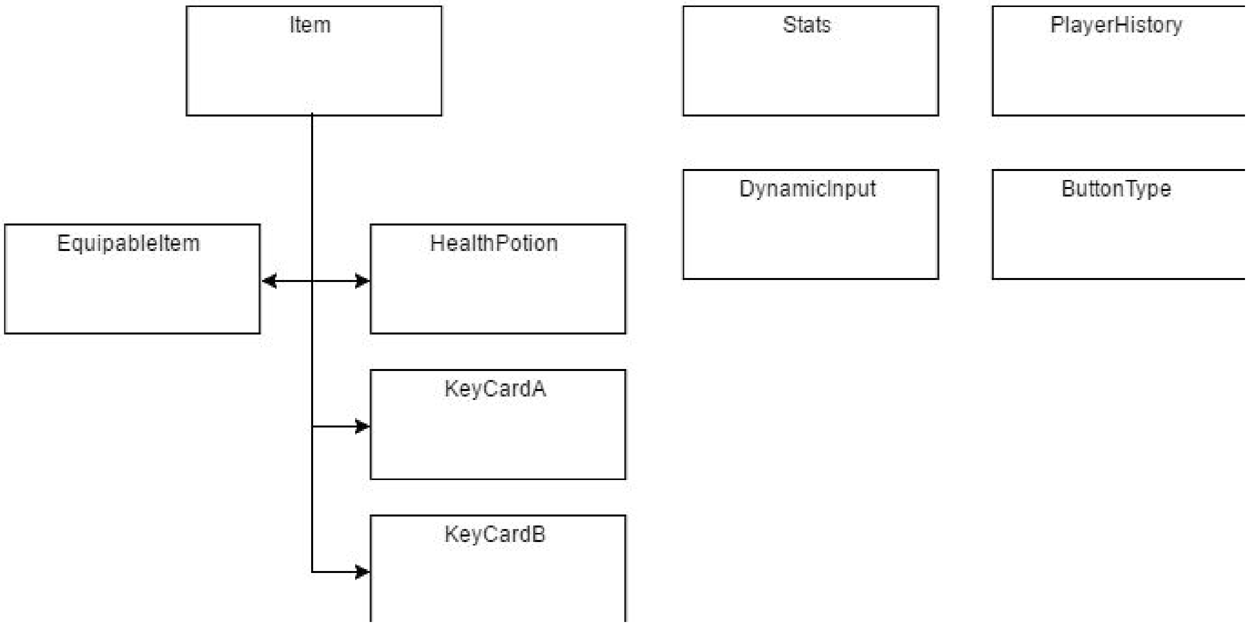


There are four use case scenarios in this project.

- The Gameplay scenario involves moving, interacting, pausing, and aiming/attacking. This scenario exists when the player is in full, direct control of the player GameObject.
- The Player can resume, save, load, or quit the game in the Pause Menu scenario. This scenario is loaded when the Player pauses the game.
- The Player can start a new game or load a game from the Title Menu scenario. This scenario is loaded whenever the game is opened.
- The Player can pause the game or aim/attack in the Turret Battle scenario. This scenario is loaded when the Player enters a turret battle.

5.2: High Level Class Diagrams





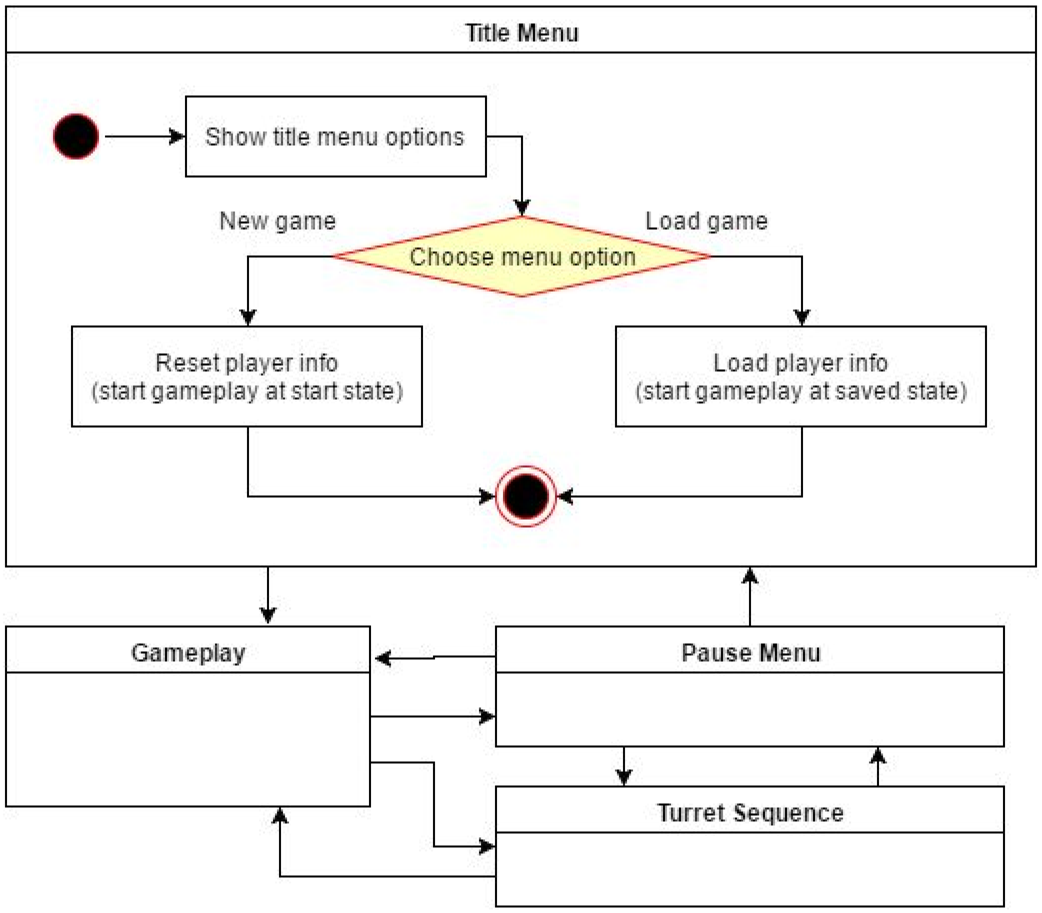
Class Sections

The classes in this project can be clearly divided into two sections. One section includes classes which inherit from Monobehaviour, and the other section includes classes which do not inherit from Monobehaviour. Monobehaviour is a class in the Unity api which is necessary for a class to inherit from if the class is to be a component of a GameObject. As a component of a GameObject, these classes may have fields exposed which can be changed and filled from the Unity inspector. This ability is useful in many cases. For example when instantiating menus in the MenuUtility class, GameObjects are created in the Unity editor and passed in as fields to classes which can instantiate them as needed. In this way, classes which inherit from Monobehaviour often deal directly with gameplay mechanics, and classes which do not inherit from Monobehaviour often represent other data.

Important Classes

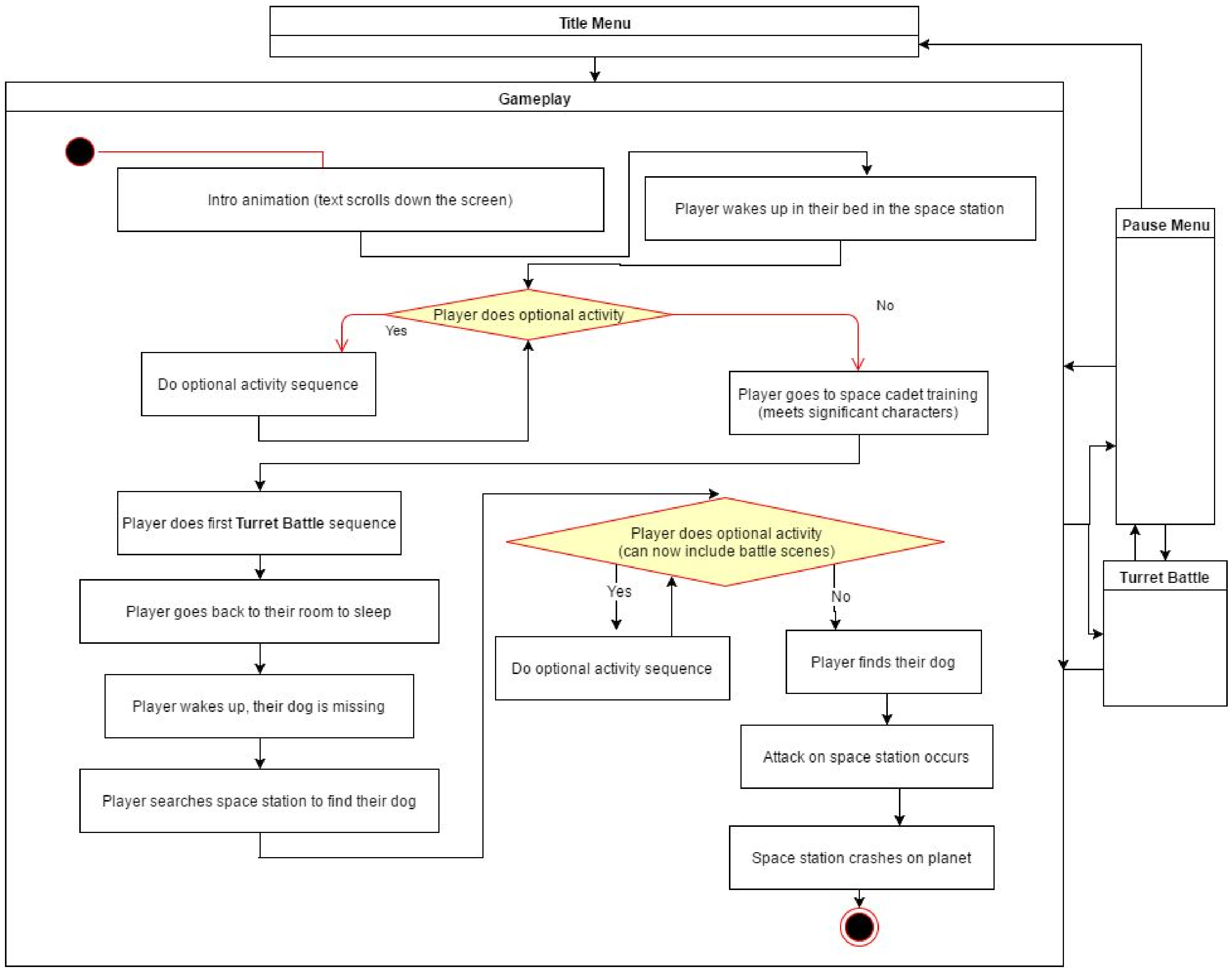
- **GameElement:** An abstract class which requires every subclass to have an elementName, elementIcon, and elementDescription.
- **Item:** An abstract class which represents an item, and requires an itemName, itemDescription, and itemIcon. This class is distinct from the GameElement class since Item objects do not have a physical representation in Unity as a GameObject (this is because Item does not inherit from Monobehaviour).
- **DynamicInput:** A wrapper class for Unity's Input class. This class is used instead of Unity's Input class to allow for different types of input for the same button.
- **GameSceneController:** An abstract class which contains references to other GameObjects (such as the Player GameObject). Each scene has a class that inherits from GameSceneController which contains the unique gameplay sequence for the scene.
- **InteractiveObject:** An abstract class which requires each subclass to implement the Interact function. This function is called whenever the Player is within a predefined distance from the InteractiveObject and presses a predefined button.

5.3: Activity Diagrams



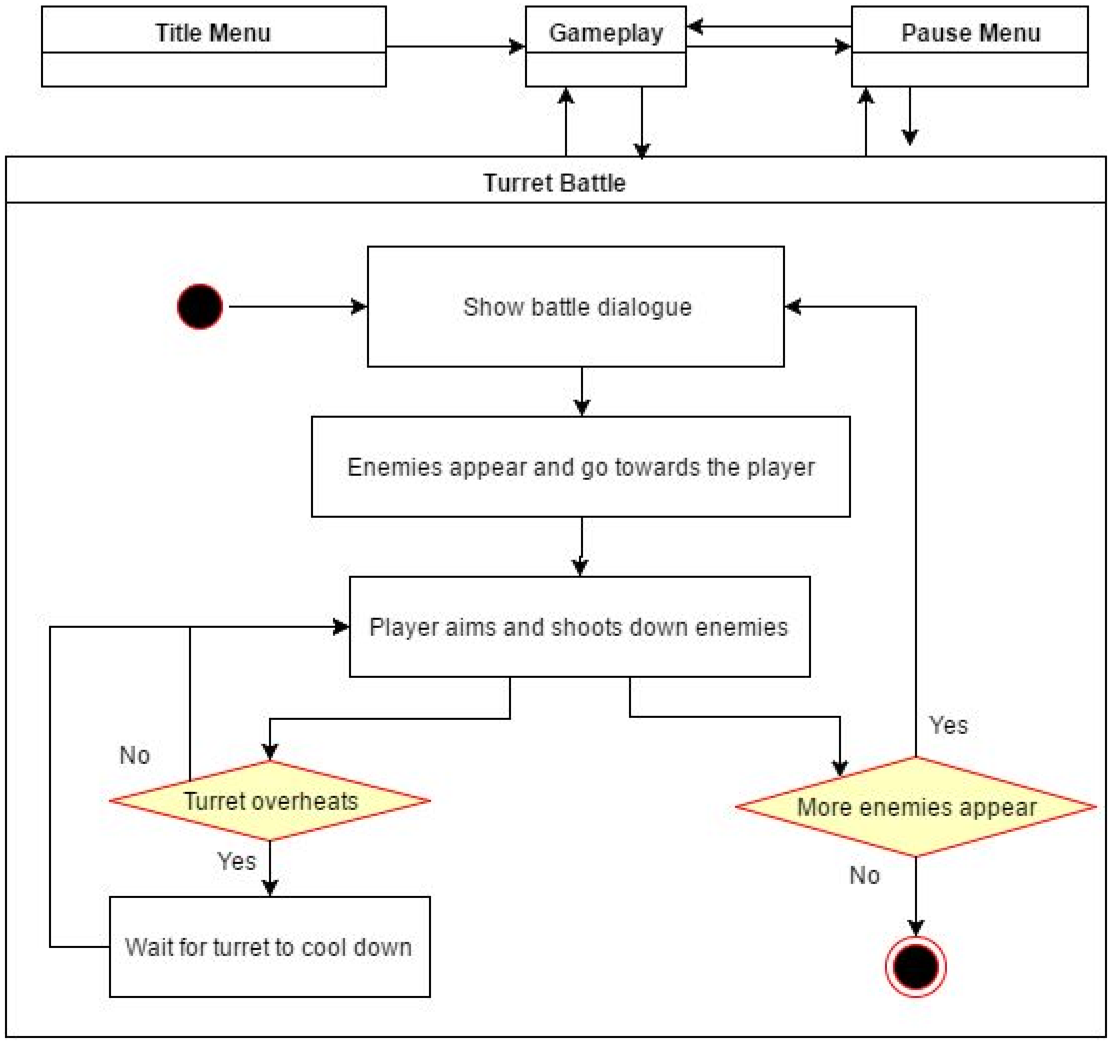
Title Menu

The Title Menu is loaded whenever the game is opened. When the Title Menu is loaded, the New Game and Load Game buttons are shown. If there are no saved game states, the Load Game button is disabled.


Title Menu
Gameplay
Intro animation (text scrolls down the screen)
Player wakes up in their bed in the space station
Player does optional activity
Yes
No
Do optional activity sequence
Player goes to space cadet training (meets significant characters)
Player does first Turret Battle sequence
Player goes back to their room to sleep
Player wakes up, their dog is missing
Player searches space station to find their dog
Player does optional activity (can now include battle scenes)
Yes
No
Do optional activity sequence
Player finds their dog
Attack on space station occurs
Space station crashes on planet
Pause Menu
Turret Battle

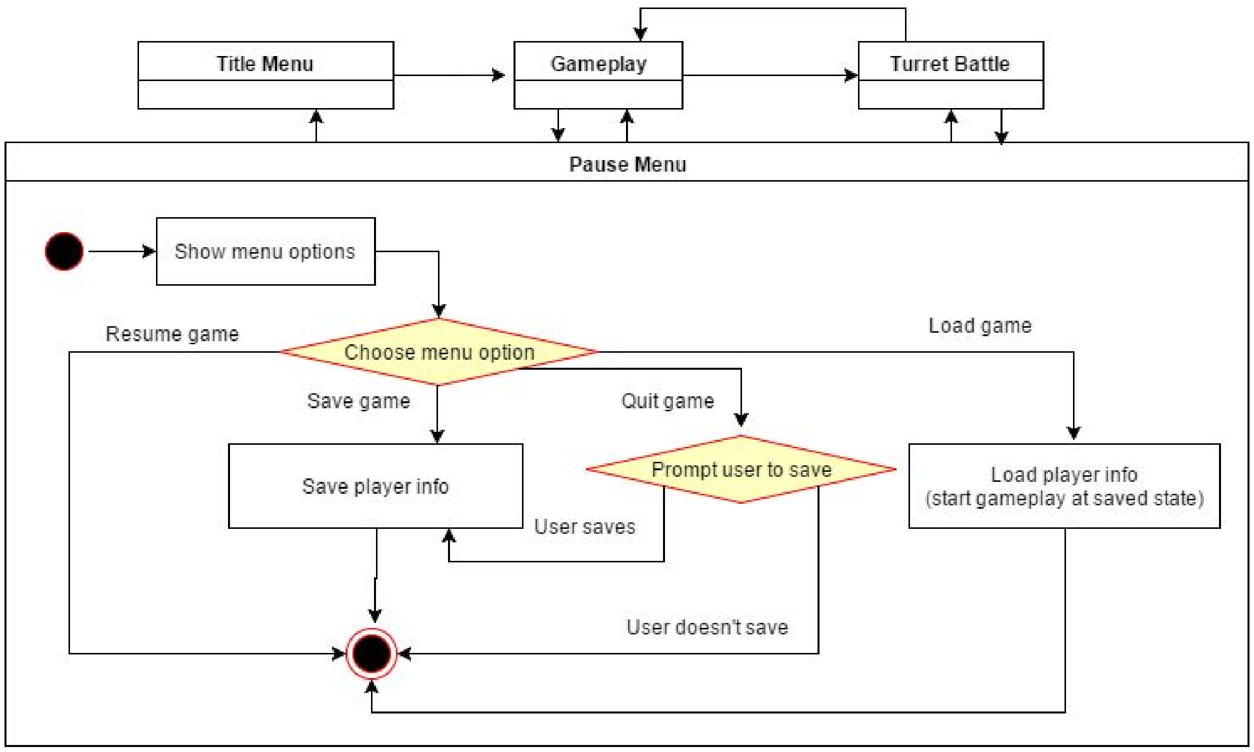
Gameplay

The initial state of the Gameplay sequence is loaded on the start of a new game. If a saved game state is loaded, the Gameplay sequence begins at that game state. The Player progresses through the Gameplay states as the story progresses. When the pause button is pressed, the Pause Menu sequence is loaded, and when the Player goes to space cadet training, the Turret Battle sequence is loaded.



Turret Battle

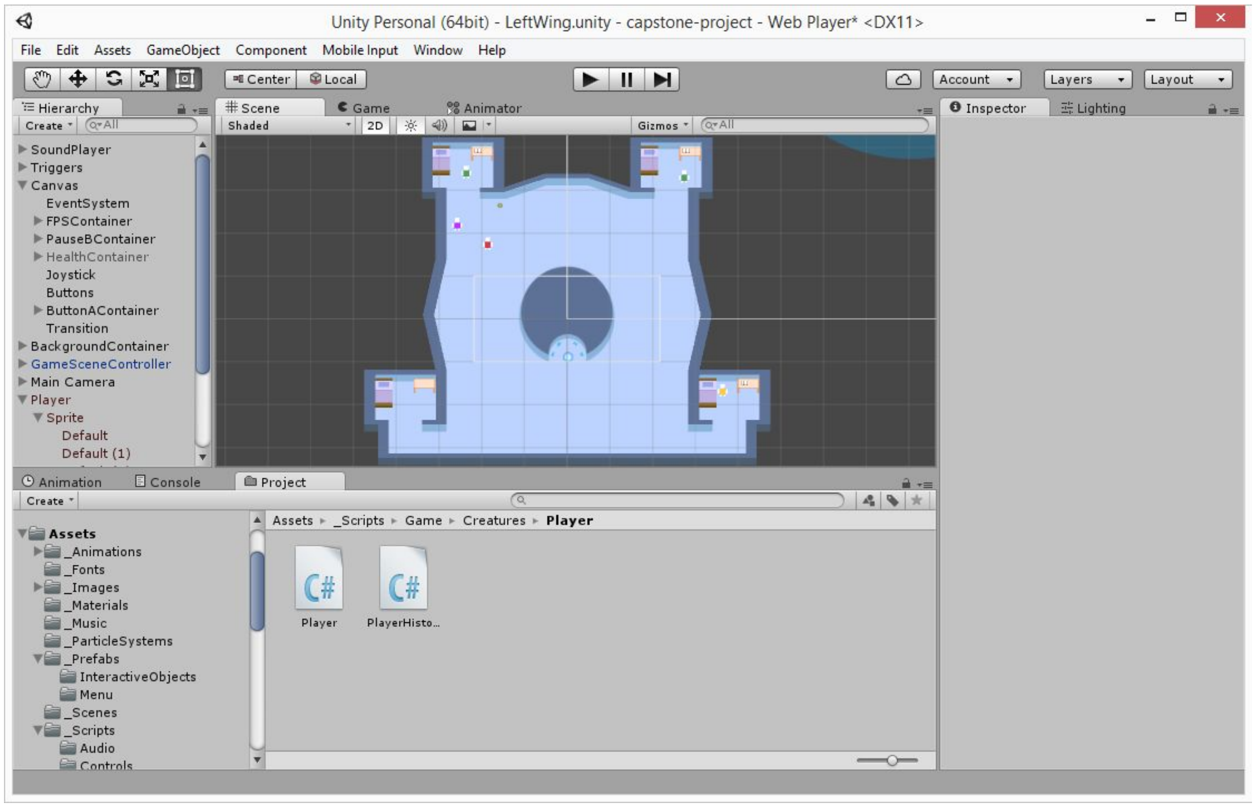
The Turret Battle sequence is loaded when the player gets to certain points in the Gameplay sequence. If the turret overheats while the Player is aiming and shooting down enemies, the Player must wait for the turret to cool down before continuing to shoot down enemies. When the pause button is pressed, the Pause Menu sequence is loaded. Also, when the Turret Battle sequence is complete, the Gameplay sequence is reloaded and the Player resumes the game at the state where the Turret Battle sequence was initialized.



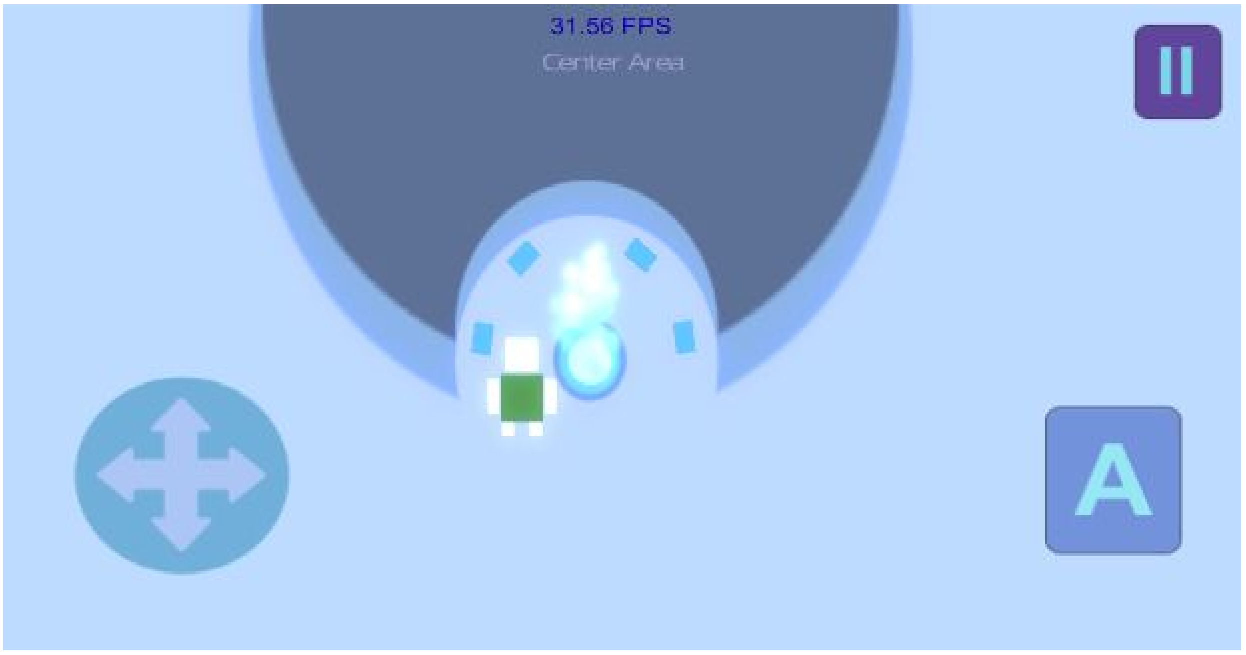
Pause Menu

The Pause Menu is loaded whenever the pause button is pressed during the Gameplay sequence or the Turret Battle sequence. When the Pause Menu is loaded, the Resume Game, Save Game, Quit Game, and Load Game buttons are shown. If there are no saved game states, the Load Game button is disabled. The Quit Game button prompts the user to save, then loads the Title Menu sequence.

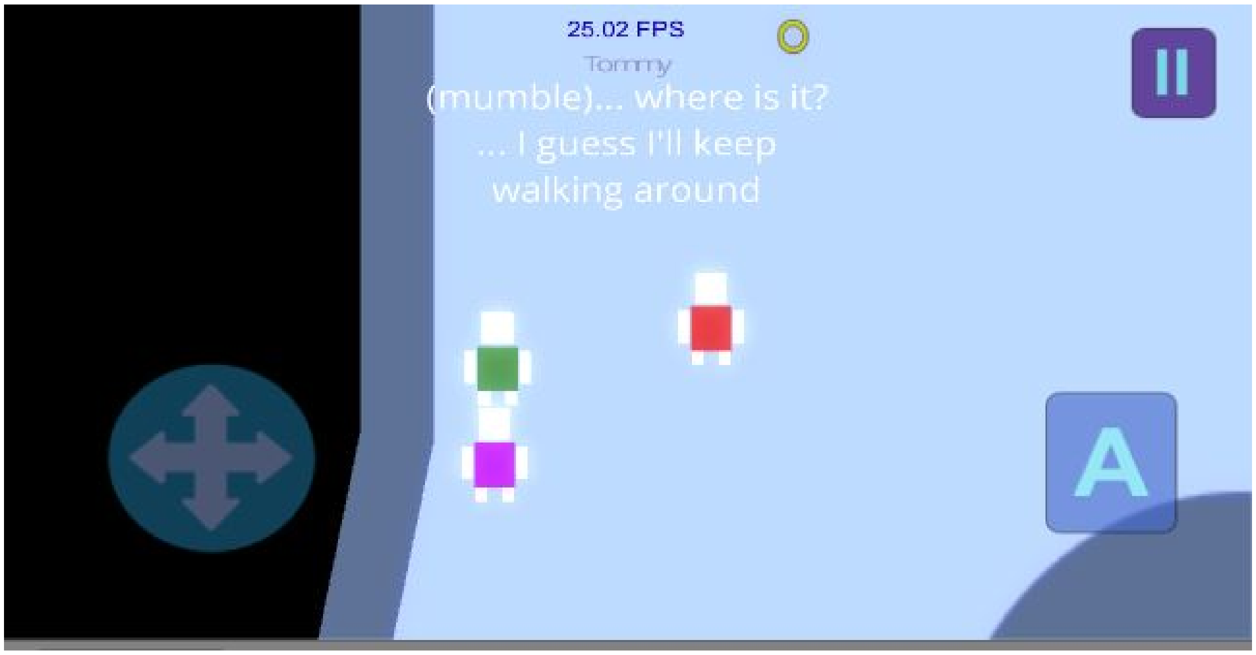
5.4: Project Screenshots



Creating a Map in Unity and populating the map with sprites for game objects



Gameplay screenshot of a player and a teleporter GameObject



Early Dialogue Example of player interacting with an NPC GameObject