

University of Nevada, Reno

**College of Engineering, Department of Computer Science and
Engineering**

Project Gemini

Acceptance Criteria and Testing Strategy and Plan

Team 01 - Lauren Davis - Nathan Evans - Allen Ma - Chris Trimble - Eric Valdez

CS 426 Instructors - Dr. David Feil-Seifer, Devrin Lee

External Advisor - Nathan Navarro Griffin (Adjunct Professor/Game Developer for Squanch Games)

March 3, 2023

Table of Contents

Table of Contents	1
1. Abstract	2
2. Project Updates and Changes	2
3. User Stories and Acceptance Criteria	2
<i>Dialogue System:</i>	2
<i>Movement and Animations:</i>	3
<i>Save and Load System:</i>	3
<i>In-Game Menu System:</i>	4
<i>Light Abilities:</i>	4
4. Testing Workflow	5
<i>Happy Path Test Workflow</i>	5
<i>Unhappy Path Test Workflow</i>	7
5. Testing Strategy	8
<i>Test Plan</i>	10
6. Unit Testing	11
<i>Player Respawn Test (Written by: Lauren Davis)</i>	11
<i>JSON Validator Test (Written by: Nathan Evans)</i>	12
<i>Player Movement and Animation (Written by Allen Ma)</i>	15
<i>Boss Moveset Unit Test (Written by Chris Trimble)</i>	16
<i>Light Ability Unit Test (Written by Eric Valdez)</i>	17
7. Contributions of Team Members	19
<i>Lauren Davis</i>	19
<i>Nathan Evans</i>	19
<i>Allen Ma</i>	19
<i>Christopher Trimble</i>	19
<i>Eric Valdez</i>	19

1. Abstract

Project Gemini is a 3D puzzle-adventure game developed in the Unity game engine. This game follows the player as they return to their childhood village which has fallen into disrepair due to seemingly supernatural forces. The player can speak and interact with the residents while attempting to rebuild the town. The player can then progress toward removing the evil plaguing the land by delving into the nearby dungeon. The game consists of two primary areas: the village and a desert themed puzzle dungeon. The primary goal for this project is to create a fun and entertaining game that will further develop the skills that each of the team members possess.

2. Project Updates and Changes

Since the previous assignment, the team did not make many major changes to the design of the game or any of the specifications. Continuing with our current project design, the team has fully developed the different boss attacks for the boss room of the dungeon, debugged and improved the functionality of the light abilities, and further developed dungeon levels to include multiple obstacles. The team is currently working on debugging various features, expanding the dialogue tree to include more paths, adding custom UI icons to materials and quests, and tutorializing a dungeon level to help with player friction. Overall, the team hopes to be content complete within the next week without removing extra content like one dungeon level, dialogue paths, etc. As a team, there is an extreme focus on the game flow for the next two weeks so players are able to fully game without bugs or errors with all the content available. After the next two weeks, the team will shift focus from content creation and development to polishing the project with lighting, music, particle effects, etc to give the game a more complete feel. This will also be the time for playtesting between ourselves and others in order to find any bugs or errors throughout the game.

3. User Stories and Acceptance Criteria

Dialogue System:

As a player I want to be able to interact with the story through NPC conversations.

- A player can initiate conversations with the “E” key when in range.
- A player can skip dialogue animations with the “LMB”
- A player can select dialogue choices with the mouse.
- A player can get different conversations based on world state.
- A player can finish a quest goal by turning in required quest materials.

As a player I want to be able to interact with the quest system through NPC conversations.

- A player can receive a quest through NPC Conversation
- A player can receive a request though NPC conversation
- The world state changes as a result of NPC Conversation
- A player can finish a quest though NPC Conversation
- A player can finish a request through NPC Conversation.

Movement and Animations:

As a player, I want to be able to move my character around the world to explore different things and progress the story.

- A player can press "WASD" keys to move.
- A player can move their mouse around to look around.
- A player can press "Space" to jump.
- A player can press "Left Shift" to sprint.

As a player, I want to see my character's movement corresponding to the animations I am inputting.

- The animation for walking plays when a player presses "WASD".
- The animation for jumping plays when a player presses "Space".
- The animation for running plays when a player hold's "Left shift" and uses "WASD" to run.

Save and Load System:

As a player, I want to create a new character so I can play the new game without any previous gameplay experience.

- A player is in the main menu
- The player has already selected "Play" from the UI.
- The player presses "New Game" from the UI to create a new character.
- The player types in the name they would like to use for the new game.
- The player presses "Play".
- After the player presses "Play" for the second time, a new player is developed and saved to JSON files.

As a player, I want to continue playing the game so that I can play the game with my previous character's information and experience.

- A player is in the main menu.
- The has already selected "Play" from the UI.
- The player presses "Saved Game" from the UI to view a list of all previously saved characters.
- The player selects the character name of which we would like to continue playing by pressing the name and then pressing "Play" from the UI.
- After pressing "Play", the specified character information is located within the JSON files and reads all previous information into the player object.

As a player, I want to save my character's progress so that I can continue playing the game later.

- The player has already begun playing the game.
- The player presses “P” on the keyboard to bring up the pause menu.
- The player presses “Save Game” from the UI.
- After pressing “Save Game”, the current player information (e.g position, amount of materials, etc) is saved to the JSON file corresponding to the character’s name by replacing the previous information.
- The player may now close the game or press “Quit” from the UI.

In-Game Menu System:

As a player, I want to check what quest items I need for a certain quest

- A player is loaded into a level of the game.
- The player presses “P” on the keyboard to bring up the pause menu.
- The player clicks on the quest icon tab using the mouse left button.
- The player then clicks on the specified quest names tab using the mouse left button.
- The player will now see what quest items they need for the quest and which ones they have already gathered on the left page.

As a player, I want to change the sound level of the music.

- A player is loaded into a level of the game.
- The player presses “P” on the keyboard to bring up the pause menu.
- The player clicks on the settings “cog” icon using the mouse left button.
- The player then clicks on the music levels bar and drags to where they want the volume.

As a player, I want to check which NPC requests I have left.

- A player is loaded into a level of the game.
- The player presses “P” on the keyboard to bring up the pause menu.
- The player clicks on the resources tab “bag” icon using the mouse left button.
- The player is now able to identify which NPC requests are incomplete, as they will not have a strikethrough indicating completion status.

Light Abilities:

As a player, I want to create boxes so that I can activate pressure plates to open doors and activate traps.

- A player can press a button to create a crate.
- The crate is created based on the position of the player character and the camera’s rotation.
- After a crate is created it can be pushed around by the player.
- If a crate is created or pushed onto a pressure plate a door or trap will activate.
- Creating a crate while one exists will cause the other crate to fade out and be destroyed.

As a player, I want to create an elevator so that I can reach areas that would otherwise be out of reach.

- A player can press a button to create an elevator.
- The elevator is created based on the position of the player character and the camera's rotation.
- After an elevator is created, the object will raise to a specified height.
- Once the elevator has reached its peak, it will be destroyed after a short pause.

As a player, I want to throw a hook so that I can interact with objects that are far away and pull objects towards me.

- A player can press a button to instantiate a hook that will progress until it hits an object.
- If the object that is hit by the hook is "grabbable," the object will be brought back to the player.
- If the object that is hit by the hook is a lever, the lever will be flipped which can activate doors, traps, or reveal hidden objects.
- If the hook hits any other objects the hook will return to the player without any other interaction.

4. Testing Workflow

Happy Path Test Workflow

Create Player Save -

1. Open game
2. Navigate to create a new game.
3. Enter character name
4. Press create character save
5. Character save created

Validation Test - Test date 12/2/2022: User tests where the player was able to create and play a new character save.

Game Boundaries -

1. Load up level
2. Attempt to run off map.
3. Player character cannot run off the map.
4. Attempt to get stuck in an obstacle.
5. Player character cannot get stuck.
6. Attempt to use an elevator to surpass a height limit.
7. Player character cannot exceed the height limit.

Validation Test - Test date 2/27/2023: During our testing process, we attempted to perform various actions such as running out of bounds, exceeding the height limit using the elevator ability, and getting stuck within the town. These tests all proved to be successful in stopping the player, preventing any unintended gameplay or glitches. However, when we tried to perform these same tests within the dungeon, we ran into some issues that need to be further addressed. Therefore, we will be retesting the dungeon on 3/10/2023 to ensure that none of the issues we encountered persist.

Graphics -

1. Load up game
2. Look at animations
3. See that animations work correctly and don't lag
4. See that visual effects run smoothly

Validation Test - Test date 3/17/2023: To ensure the proper functionality of this system, we will begin by loading a player save or creating a new one within the game. We will then move the character around and observe that all animations are operating smoothly and as intended. Next, we will navigate to areas within the game that utilize visual effects, such as particle effects or lighting, and ensure that these systems do not impede gameplay or cause any technical issues. By conducting these tests, we can guarantee that the game's animations and visual effects are optimized for an enjoyable player experience, without any hindrances to gameplay or technical difficulties.

Player Abilities -

1. Load up game
2. Player presses a key that creates a crate using their ability and it spawns.
3. Player presses a key that creates an elevator using their ability it spawns and elevates.
4. Player presses a key that creates a light rope using their ability it spawns and shoots out a hook.
5. Subsequent button presses will result in the respective object being created and the old version being destroyed.

Validation Test - Test date 2/24/2023: Tested each ability using the corresponding keys and the abilities both spawned each object and ran the correct functionality. Also tested using the unit test Eric created.

Dialogue System -

1. Load a player save or create new player
2. Player walks up to an NPC.
3. Player interacts with a NPC and receives a quest.
4. The player finishes the quest and talks to NPC again.
5. The dialogue starts at that particular part in the dialogue tree.

Validation Test - Test date 3/5/2023: To test the functionality of this system, we will begin by creating a new player save and loading into the level. We will then approach an NPC and accept a quest and request. After completing the quest and request, we will return to the NPC and verify that the appropriate dialogue is displayed. This testing process will ensure that the dialogue system is functioning properly and that the player can successfully progress through the game.

Unhappy Path Test Workflow

Create Player Save -

1. Open Game
2. Navigate to player creation menu
3. Enter character name
4. Click create character
5. Character is not created
6. Save storage is out of space since there are only 100 saves allowed.

Validation Test - Test Date 12/2/22. I created 100 previous saves and then tried to make a new character and it didn't let me.

Game Boundaries -

1. Load up level.
2. Attempt to run off the map.
3. Player is able to run off the map.
4. Attempt to get stuck in an obstacle.
5. Player is stuck and the game is now softlocked.
6. Attempt to surpass the height limit.
7. Player can climb over structures using abilities and skip gameplay.

Validation test - Test date 02/20/23: Some boundaries were not set up yet that allowed for the player to run off the entire map. The river in the main town was too high for the player to jump out of so they got stuck there. The elevator ability allows you to keep surpassing the height limit by jumping on to newly spawned elevators.

Graphics -

1. Load up game
2. Attempt to move character
3. Animations are broken
4. Load into the boss room.
5. Boss does specific visual effect movement
6. Visual effects bog down gameplay and make it hard for the game to run smooth

Validation test - Test date 1/07/2023: When moving the player, the animations do not do what they are supposed to and are out of sync with the character movement. When fighting the boss, the particle effects are too demanding and cause unsmooth gameplay.

Player Abilities -

1. Player presses a key to spawn a crate and nothing happens or doesn't function correctly.
2. Player presses a key to spawn the elevator and nothing happens or doesn't function correctly.
3. Player presses a key to spawn a light rope and does nothing or doesn't function correctly.
4. Pressing a key to spawn an object that is already created does not destroy older objects.

Validation test - Test date 2/17/23 : A new character controller was made and the key bindings to activate all of the abilities did not function correctly.

Dialogue System -

1. Load into the game, create a character or load player save file.
2. Attempt to interact with an NPC.
3. Player interacts with an NPC and receives a quest.
4. Player finishes the quest then talks to the NPC again.
5. Dialogue starts from the beginning and does not remember the quest.

Validation test - Test date 2/28/23 : Tried talking to an NPC and getting quests, but after talking to them again I could receive the same quest even though I accepted it.

5. Testing Strategy

Project Gemini will be utilizing several testing techniques in order to ensure that the final product is functioning properly and at an acceptable level. Most notably, Project Gemini will be making use of user tests throughout the current stage of development. These tests will be conducted not only by the group developing the project, but our advisor, project manager as well as various friends, family and colleagues of the group. These generalized user tests will help to identify any issues in the software such as missing colliders, errors, and possible oversights on functionality. Already several people have tested the current build and helped to identify some game breaking issues that have been addressed. Beyond user tests, Project Gemini will also utilize several unit tests including those listed in the section below. These tests will primarily help to ensure that the smaller components are working properly. Any issues with these components can be identified and addressed as part of these tests being completed. As each member of the team has been working diligently on their own areas of this project, it is important that each member tests a

component that they have not worked directly with. The team has distributed several testing responsibilities amongst each other. Lauren will be testing the dialogue trees that were created by Nathan and Nathan will be testing the character controller. The tests conducted by Lauren and Nathan will be completed by March 7th. Allen will be testing the save and load features as well as various levels of the dungeon by March 8th. Chris will be testing the light abilities and Eric will be testing the menus and UI. Each of these tests will be completed by March 6th. The tests conducted by the team will largely be user tests, as each of these elements heavily affects the gameplay of Project Gemini. Any defects that are found by the user tests conducted by those outside the group will be divulged to the entire development team and whoever's component it was that broke will be in charge of fixing it. As for the tests conducted by the team, all members will still be notified. The member who found the defect will then work closely with the member whose component produced the unintended functionality in order to better understand exactly what went wrong and how. This information will largely be spread among the development team's discord as well as in weekly meetings as soon as an issue is discovered. Overall, Project Gemini will be deemed "content complete" once all of the first and second level functional requirements written in the previous document have been integrated and passed any and all acceptance tests. Once each of these main components are working properly, the team can move forward to achieving project completion. Project completion will be denoted by all first and second level non-functional requirements being met and the team completing all necessary art, polish and gameplay tests. This will result in the game being fun to play without any major issues, as well as overall looking, sounding, and feeling great.

Test Plan

Test No.	Test Type	Target File or Screen	Test Name	Purpose of Test	Test Data or Situation	Expected Result	Actual Result	Outcome and Required Actions
1	Saving	SaveSystem.cs	Create Player Save	Tests if player saves are correctly generated	1. No existing saved data 2. existing saved data	Data should be correctly generated and stored in the users appdata	1. As expected 2. As expected	Saved data are generated correctly. No action required.
2	Player	Killz.cs	Game Boundaries	To ensure player cannot escape intended play area	1. Village 2. Dungeon	Player should never be able to escape the intended play area	1. As expected 2. Player can escape	The player is able to escape the intended play area in certain situations. We need to address these edge cases to ensure the player is unable to escape
3	Player	PlayerController.cs	Animations	Verify that player animations are playing as intended	1. In Walk State 2. In any other state	Player animations and their transitions should be smooth and match player intent. During non-walk state no animations other than idle should be playing.	1. Jump animation can break 2. As expected	There are certain situation where the playercontroller does not handle the jump animation correctly We need to add further logic watching for rapid inputs and other edge cases.
4	Player	ObjectCreation.cs	Light Abilities	To verify all player light ability spawn function and fade as intended.	1. Hook 2. Elevator 3. Box	All light abilities should spawn with their proper button press. The ability should carry out its specific function. The ability should fade out.	1. As expected 2. As expected 3. As expected	Light abilities are currently working as designed no action required
5	Conversation	Interactable NPC.cs	Dialogue	Verify all dialogue functions, save state, and allow response correctly.	1. JSON Load 2. Dialogue Interaction	The dialogue tree should load successfully and be able to be traversed by the player without issue.	1. As expected 2. Unable to completely traverse dialogue	We need to change how the player interacts with the dialogue system as the current control scheme does not work on small keyboards.

6. Unit Testing

Player Respawn Test (Written by: Lauren Davis)

This unit test was developed for players to respawn at the beginning of levels or checkpoints. For example, if the player hits an obstacle like spikes or arrows then the player should return to the latest checkpoint or the beginning of the level. The unit test shown below grabs the latest player position before moving, the respawn point position, and the player position after the respawn function is called. The test then outputs to the console these three positions to visually confirm the player has moved to the respawn location using the debug tool. The assert tool is then used to double check that the respawn function is working properly by comparing the old position and the new position of the player and that the new player position matches the respawn location. The image after the code, shows the console displaying the debugging message proving the test is operating correctly and the assert lines never threw exceptions or stopped executing the code while it runs.

```
public void RespawnTest()
{
    //Finding player in scene and recording its position
    var player = GameObject.FindGameObjectWithTag("Player");
    var oldPos = player.transform.position;

    //Finding the respawn point in scene and its location
    var respawnPoint = GameObject.FindGameObjectWithTag("RespawnPoint");

    //Calling function to test
    player.gameObject.GetComponent<PlayerScripts.PlayerStats>().Respawn();

    //Recording the new position of the player
    var newPos = player.transform.position;

    //Testing the positions to confirm player moved to the new position
    Debug.Log("Respawn position: " + respawnPoint.transform.position);
    Debug.Log("Old player position: " + oldPos);
    Debug.Log("New player position: " + newPos);
    Assert.AreNotEqual(oldPos, newPos);
    Assert.AreEqual(respawnPoint.transform.position, newPos);
}
```

Figure 6.1 : Unit test code for player respawning.

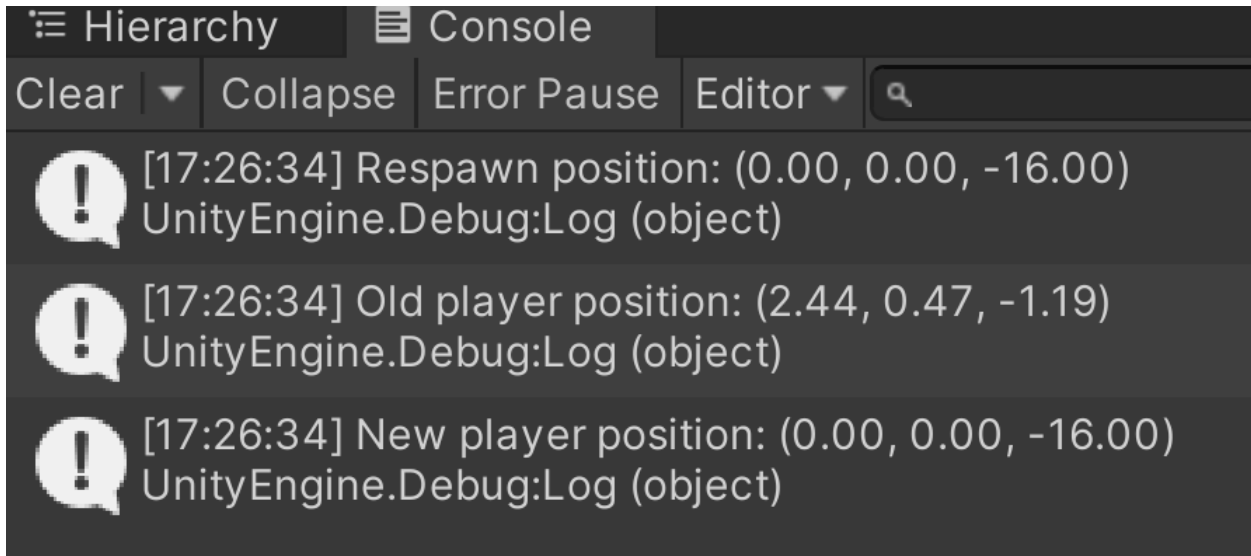


Figure 6.2: Unity console displaying a successful test with the position of the respawn point, old player position, and new player position.

JSON Validator Test (Written by: Nathan Evans)

We created a unit test to check if a JSON file holding a dialogue tree contains all the correct tags. This test can parse through the JSON file and verify that every dialogue entry includes all the necessary tags, such as speaker, id, and isRepeatable. By doing so, the test can ensure that the JSON file is well-formed and adheres to the expected schema. This is important for development because it helps catch errors early on in the development cycle, before they become bigger issues. It also helps ensure that the dialogue system is functioning correctly and can parse through the JSON file without encountering unexpected errors or missing data. By using Unit tests to verify the correctness of JSON files, we can save time and reduce the risk of introducing errors into the dialogue system.

```

public bool isJSONValidTest(TextAsset testJson)
{
    conversation = JsonUtility.FromJson<Conversation>(testJson.text);

    foreach (var dialogue in conversation.dialogues)
    {
        if (!dialogue.AreAllTagsValid())
        {
            Debug.Log(message: "JSON is not valid");
            return false;
        }
    }

    Debug.Log(message: "JSON is valid");
    return true;
}

```

Figure 6.3: Unity test code for JSON Validator

Valid Input:

This input contains all the expected tags that are required by the Dialogue class. While it may not have all possible tags that could be included in the Dialogue class, it is still considered a valid entry as there are no invalid tags present. Ensuring that all required tags are present and that no invalid tags are included is crucial for the proper functioning of the Dialogue class. This helps to prevent unexpected errors and allows for accurate parsing of dialogue data. By adhering to a well-defined schema, developers can ensure consistency across dialogue entries, leading to a more robust and reliable system.

```

"dialogues":
[
  {
    "id": 1,
    "speaker": "Charles",
    "text": "Hi, I am Charles",
    "isTerminal": true,
    "hasOptions": false,
    "isRepeatable": true
  }
]

```

Figure 6.4: Valid JSON file example

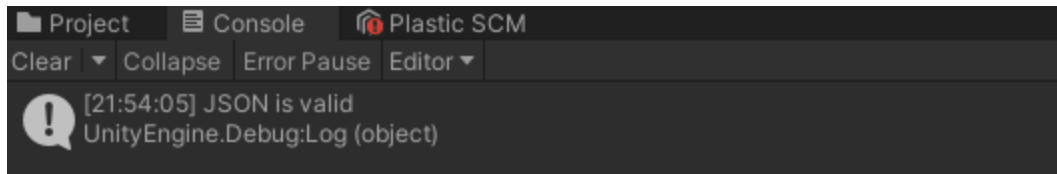


Figure 6.5: Accepted test result

Invalid Input:

This input is considered invalid as it includes tags with unexpected names. For example, the "speaker" tag has been changed to "name". Since this does not adhere to the expected schema, it will be rejected by the JSON validator test. Ensuring that the tags are named correctly is crucial for the proper functioning of the dialogue system, as it allows for accurate parsing of dialogue data. By using a JSON validator test to identify and reject invalid inputs, developers can catch errors early in the development process and prevent unexpected behavior in the system. This ultimately leads to a more robust and reliable dialogue system.

```
"dialogues":  
[  
  {  
    "id": 1,  
    "name": "Charles",  
    "message": "Hi, I am Charles",  
    "end": true,  
    "Options": false,  
    "Repeatable": true  
  }  
]
```

Figure 6.6: Invalid JSON file example

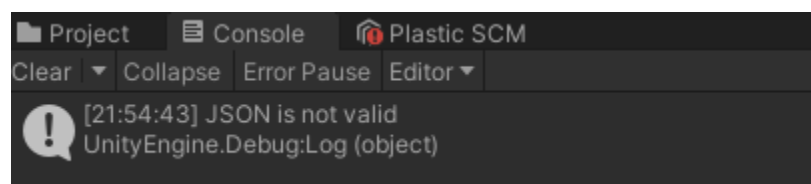


Figure 6.7: Rejected test result

Player Movement and Animation (Written by Allen Ma)

The unit test was designed to verify that each of the players movement and animation mechanics are working properly and each input is being taken in. This test also determines the player's relative position and velocity in the world to check if the player is actually moving when inputs are being taken in.

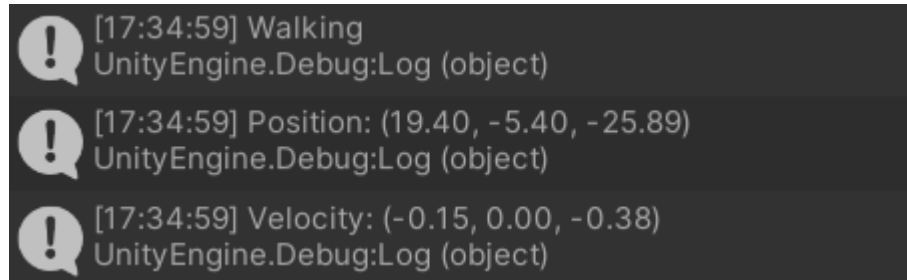


Figure 6.8: Successful player movement, position, and velocity.

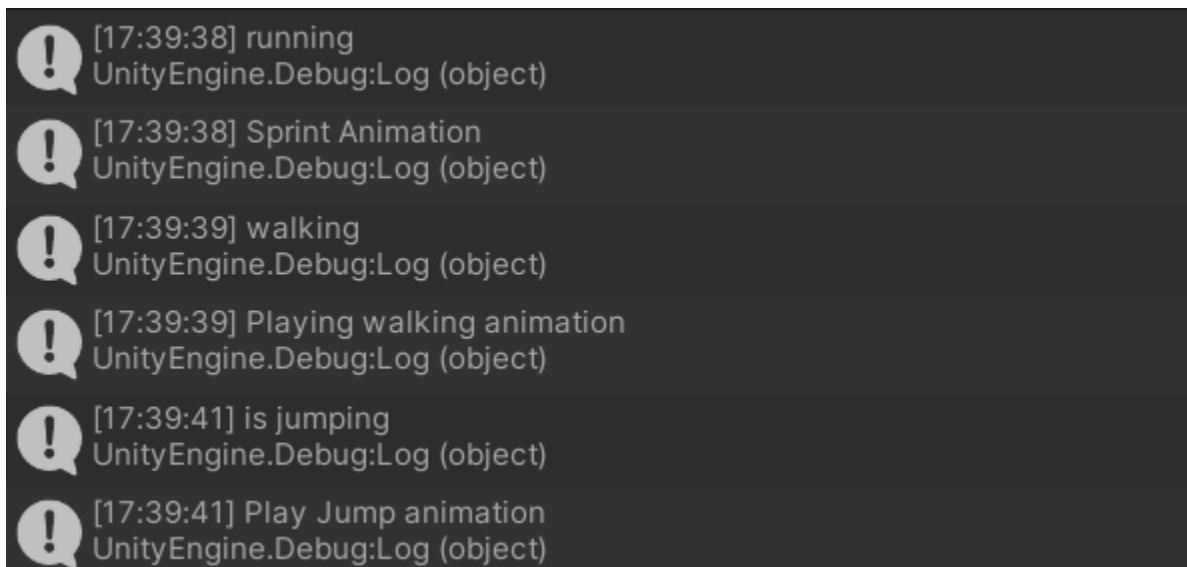


Figure 6.9: Successful player inputs and checks for animations

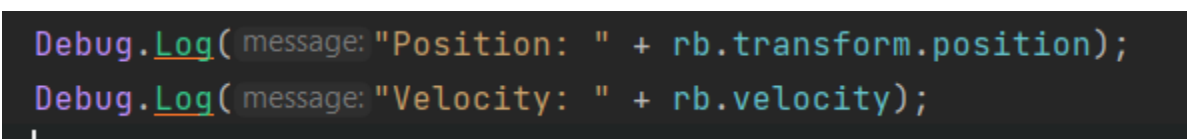


Figure 6.10: Source code for player position and velocity.


```

if (Input.GetKeyDown(KeyCode.Space))
{
    Debug.Log(message: "Play Jump animation");
}
else if (Input.GetKeyDown(KeyCode.W) || (Input.GetKeyDown(KeyCode.S)) || (Input.GetKeyDown(KeyCode.D)) || (Input.GetKeyDown(KeyCode.A)))
{
    Debug.Log(message: "walking");
    Debug.Log(message: "Playing walking animation");
}
else if (Input.GetKeyDown(KeyCode.LeftShift))
{
    Debug.Log(message: "running");

    Debug.Log(message: "Sprint Animation");
}

```

Figure 6.11: Source code to check for the player inputs and animations

Boss Moveset Unit Test (Written by Chris Trimble)

The unit test was designed to verify that each of the boss's mechanics can be invoked with a specific number. During gameplay, the boss will randomly generate numbers to activate these movesets. To enable the specified unit test within the game, I created a hotkey that can be accessed by pressing the "U" key (as shown in Figure 6.12). To ensure that each moveset is called correctly, the unit test passes in the required numbers needed to generate each moveset (as shown in Figure 6.13). Debug statements in Figure 6.14 confirm that each moveset was indeed called accurately.

```

if (Input.GetKeyDown(KeyCode.U))
    StartCoroutine(routine: TestMoveSet());

```

Figure 6.12: Source code for calling TestMoveSet

```

// Test case for the boss movesets.
Frequently called  1 usage
private IEnumerator TestMoveSet()
{
    // First test for Outerjump Move
    yield return new WaitForSeconds(ChooseAMoveSet(rand: 100));
    // Second test for StompAttack Move
    yield return new WaitForSeconds(ChooseAMoveSet(rand: 205));
    // Third test for Ball Attack Move
    yield return new WaitForSeconds(ChooseAMoveSet(rand: 805));
}

```

Figure 6.13: Source code of test for the boss's movesets



Figure 6.14: Successful Boss Moveset Statement Outputs

Light Ability Unit Test (Written by Eric Valdez)

The following unit test was conducted on the creation and destruction of the light crate and the light elevator for the player. This test instantiates the objects in relation to the player and camera rotation and checks to ensure that position is correct. After a short time the elevator will begin to fade and eventually be destroyed. Prior to destruction the test will check that the alpha value, or transparency value, falls by 0.888 before destroying. Next the test will run a similar scenario for the crate wherein the crate will be expected to drop its alpha value by about 0.8 prior to destruction. The following images demonstrate the successful test.

```

//Unit Test of Light Crate and Elevator
1 reference
private void LightTest()
{
    //Gather Player's Position and add in the forward direction of the camera.
    Vector3 tempVec = player.transform.position;
    tempVec += (playerCam.transform.forward*3f);
    tempVec.y = player.transform.position.y;

    //Prevent rotation along x and z axis for spawning.
    var currentRotation = playerCam.transform.rotation;
    currentRotation.x = 0;
    currentRotation.z = 0;

    //Spawn Elevator and ensure proper spawn position.
    isElevator = true;
    elevator = Instantiate(elevatorPrefab, tempVec - (player.transform.up*0.9f), currentRotation);
    Debug.Log(elevator.transform.position == tempVec - (player.transform.up*0.9f));

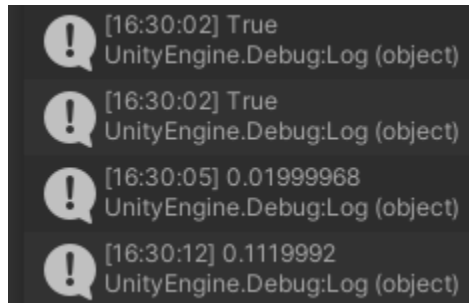
    //Fade and Destroy Elevator after set time.
    model2 = elevator.GetComponentInChildren<Renderer>();
    StartCoroutine(DebugWait());
    lastFade = StartCoroutine(FadeElevator());
    lastDestroy = StartCoroutine(DestroyElevator());

    //Spawn Crate and ensure proper spawn position.
    isBlock = true;
    block = Instantiate(blockPrefab, tempVec + (player.transform.up * 0.5f), currentRotation);
    Debug.Log(block.transform.position == tempVec + (player.transform.up * 0.5f));

    //Fade and Destroy Block after set time.
    model1 = block.GetComponentInChildren<Renderer>();
    StartCoroutine(DebugWaitTwo());
    StartCoroutine(FadeBlock(model1));
    StartCoroutine(DestroyBlock(block));
}

```

Figure 6.15: Source Code of Light Unit Test



```

[16:30:02] True
UnityEngine.Debug:Log (object)

[16:30:02] True
UnityEngine.Debug:Log (object)

[16:30:05] 0.01999968
UnityEngine.Debug:Log (object)

[16:30:12] 0.1119992
UnityEngine.Debug:Log (object)

```

Figure 6.16: Successful Test Output

The order of these debug outputs begin with the elevator and crate's position checks which can both be seen working due to the output of "true." Then the crate is the first to be destroyed and as such the final transparency is sent to the debug log. This transparency can be seen to have dropped below 0.8 as expected. Finally, the elevator's final transparency value is sent to the debugger which can be seen to have properly dropped by about 0.888. As such this test was completed successfully.

7. Contributions of Team Members

Lauren Davis

2.5 hours - Project Updates and Changes, Save and Load Users Stories, Player Respawn Unit Test

Nathan Evans

2 hours - Dialogue user story, Test Plan, JSON Validator Test

Allen Ma

2 hours - Movement and Animation user stories, Movement and animation unit test, Unhappy workflow

Christopher Trimble

2.5 hours - In-Game Menu System User Stories, Boss Moveset unit test, Happy workflow

Eric Valdez

2.5 hours - Abstract, Light Ability User Stories, Testing Strategy, Light Ability Unit Test, and General Formatting