**Test Plan**

The primary goal of a test plan is to identify bugs in your game. A test plan is made up of many different test cases. A test case can be both positive and negative and should show the process taken by the tester to arrive at the desired or undesired conclusion.

**Test Case 1**

**Player functionality**

Perhaps one of the most important things to get right in a game is the user controls and functionality. I tested the player movement for multi button functionality meaning when the player presses the jump key while already holding the move key does he still jump.

Also, as was pointed out by the designer in the design document, it is very important that the user be able to reload the current scene quickly and easily as it easy to become stuck in the game with no means of progressing. I set the ‘R’ key as the reload button, the user can press the key at any point in any level and the scene and all game objects revert to their original state immediately.

public void ReloadScene()

currentScene = SceneManager.GetActiveScene(); // Sets current scene equal to the active scene

if (Input.GetKeyDown(KeyCode.R))

{

SceneManager.LoadScene(currentScene.name); // loads currentScene

Debug.Log(currentScene.name);

}

**Test Case 2**

**Menus**

Menu transition should be seamless. From the main menu you should be able to access the instructions and the options as well as begin the game or quit the application. Every menu should have a route back to the main menu and any changed settings should carry over.

The font of all menus and buttons should be consistent and they should scale to the screen whether in fullscreen or half etc.

**Test Case 3**

**Background music and sound effects**

There was a lot of testing done for the sounds in the game. Firstly, I created a music player as a singleton to ensure only one instance of the soundtrack persisted across all scenes. This stopped the music being played more than once at a time if the user went back to the main menu.

private void Awake()

{

SetupSingleton();

}

private void SetupSingleton()

{

// find object of type musicplayer

// if there is one, use that instance

// destroy the one just created

// FindObjectOfType()

if(FindObjectsOfType<MusicPlayer>().Length > 1)

{

Destroy(gameObject);

}

else

{

DontDestroyOnLoad(gameObject);

}

}

}

I also tested the options menu to ensure that the toggle buttons were functioning properly as well as the volume slider. Each toggle should interacts with a different audio mixer, one for the soundtrack and one for the sound effects. The slider changes the main audiomixers volume.

**Test Case 4**

**Animations**

The player animations should appear smooth and transition well. When stopping or jumping. I struggled to achieve this as once the player stops moving on the horizontal axis he continues to ‘slide’ along the ground for another half a second or so.

else

{

rb.velocity = new Vector2(0f, rb.velocity.y); // Sets player horizontal movement to 0, this prevents sliding

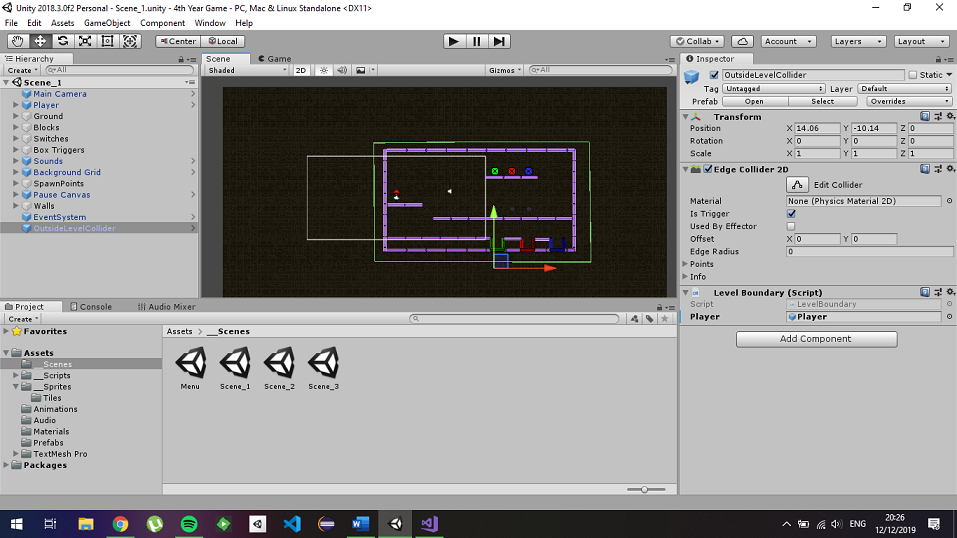
}

I tried to set the velocity of the rb to 0 if there was no movement but for some reason it isn’t immediate in its effect. This causes the animation to seem slow or laggy

**Test Case 5**

**Performance and Game Flow**

Overall game logic and functionality. Can the player somehow get out of the game area? If so, what happens? I put in a box collider that immediately restarts the current scene if the player game object comes into contact with it.



Above you can see the edge collider surrounding the map. This is set as a trigger with a simple levelBoundaries script attached to it. If the player gameObject hits the collider, the SceneManager reloads the scene.