**Project Testing**

The purpose of this document is to track the tests performed on my Platformer Game project. Due to the nature of many of my scripts being tied to GameObjects in Unity, and me not having a significant amount of experience writing tests to be run in Play Mode within Unity’s Editor, I have decided to primarily run manual testing on my game. There will be unit tests written for the DatabaseManager class, as it is entirely a static utility class written in C#, allowing me to take a more conventional unit testing approach in that class. The rest of my scripts will be tested manually according to relevant requirements, and the results of that testing will be documented here. Some scripts may have additional testing outside of the original requirements, as while I kept scope creep or changes to requirements to a minimum during this project, some additional factors or features required consideration and implementation, despite not being listed in the requirements originally. Such additional or modified requirements will be marked with *italicised* requirement text.

Below is a complete list of each C# script written for the game, in the same structure as the files present in the Scripts folder of my project. I will perform tests on each script, against each relevant requirement.

|  |  |
| --- | --- |
| Gameplay:   * Player:   + PlayerMovement   + PlayerLook   + MoveCamera   + GrappleHook   + HUDController * Level:   + Interactables     - Door     - TimedButton     - ToggleButton   + Level Completion     - LevelCompleteScreen     - LevelCompleteBtns     - NextLevel   + Level Details     - Collectible     - LevelData     - Timer | Menus:   * Input Menu **[CUT FEATURE]**   + InputMenu * Leaderboard   + Leaderboard   + LeaderboardBack * Settings   + ResolutionDropdown   + SettingsMenu * StartGame   + LeaderboardBtn   + LevelSelect   + LevelSelectBack   + StartGame   Utility:   * DatabaseManager * LevelManager * PlayerTime * SaveManager |

I will also discuss the non-functional testing I performed at the end of this document.

**Functional Testing**

Gameplay: Player

PlayerMovement:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| The player should be able to move in all four cardinal directions from a first-person perspective | Enter the game, and attempt to move in all four directions, using WASD keys and a gamepad’s left stick | The player character moves around according to the player’s input | PASS |
| The player should have access to a basic jump ability that adheres to physics | Enter the game, and attempt to jump using the spacebar key and the south face button on a gamepad | The player character performs a jump action, propelling them into the air, and falling back to the ground | PASS |
| *The player should be able to stand on, or connect their grappling hook to, any relevant object in the level* | Enter the game, and use the movement controls to stand on a platform. Then, move to stand on a grappleable object | The player should be able to stand and move around on regular ground with correct physics.  The player’s movement should become slippery and their jump will not be reset if they step on a grappleable object | PASS |

PlayerLook & MoveCamera:

These two scripts are intertwined, with PlayerLook taking the player’s input, and MoveCamera updating the camera’s orientation. Having the camera directly attached to a RigidBody object for the player resulted in some disorienting effects under certain circumstances, so I decided to split the functionality into two scripts that control separate GameObjects

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| The player should be able to freely look around, but be unable to turn the camera upside down | Enter the game, and attempt to look around using the mouse or the right stick on a gamepad. | The player’s camera turns in the direction of the input | PASS |
| *After a camera movement, the player’s movement input should continue to reflect their camera angle* | Enter the game, and move the camera to a new orientation. Then, attempt to move in a cardinal direction after changing the camera angle | The player continues to move in the same direction relative to the camera | PASS |

GrappleHook:

Some changes were made during development regarding which objects should be grappleable or not. I decided to only make a specific layer of GameObjects grappleable, allowing more control when creating levels.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to engage and disengage the grappling hook using the left mouse button or right trigger*  *The grappling hook should connect to any grappleable object within a set range* | Aim the camera at a grappleable object within range, and click the left mouse button, or pull the right trigger on a gamepad | The player fires a grappling hook at the object, and a swing begins | PASS |
| Upon attachment, the player should be tethered to the point selected, restricting their movement | Fire the grappling hook at a grappleable object in range, and attempt to move using WASD or the left stick on a controller | The player’s input does **not** have an impact on their direction or position | PASS |
| While attached, the player should be able to swing using their character’s weight as the mass of a pendulum | Fire the grappling hook at a grappleable object in range, and hold the input | The player character starts swinging like a pendulum, adhering to regular physics.  The player’s momentum should be gradually decreased by damping. | PASS |
| *The player should be able to freely look around while swinging* | Fire the grappling hook at a grappleable object in range, and hold the input. Then, attempt to look around using the mouse or the right gamepad stick | The player character and camera can rotate freely, with no impact on the direction of the swing | PASS |
| *The player should be able to stand on, or connect their grappling hook to, any relevant object in the level* | Enter the game, and attempt to grapple onto a regular platform. Then, attempt to grapple onto a grappleable object | The player should not be able to grapple to any regular platforms or other objects, and they can only grapple onto a grappleable object | PASS |

HUDController:

As a note on the final requirement, I decided to change course on showing if the collectible had been gathered on any previous attempt. I felt that having its historical status on-screen for every attempt would clutter the UI and prove distracting. The collectible’s status can be viewed on the level selection screen instead.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| The player’s UI should indicate if their current target point can be grappled to or not | Aim at a non-grappleable object. Then aim at a grappleable object but outside of the grapple range. Then move into range of the object. | The circle in the centre of the HUD should remain white until the player is aiming at a grappleable object *and* is in range to grapple onto it | PASS |
| The UI should show the player’s time spent in the current level, as well as their number of restarts on this attempt | Enter a level, and observe the HUD for the timer and restart count. Then, jump into a Pit and observe the timer and restart count. | The timer should tick up as long as the player is in the level.  Once the player falls into a Pit, the timer should be reset and the restart count should be incremented | PARTIAL PASS  The timer functions as expected, but my implementation of resetting the level did not allow for a reset count to be shown |
| *The UI should show if the player has collected the bonus collectible item on the current attempt* | Enter a level, and observe the HUD. Then, collect the collectible item in the current level. | The HUD should not show any text regarding the collectible until it is collected | PASS |

Gameplay: Levels

Interactables:

Door:

This was a new feature I chose to implement in the time I had remaining after deciding to cut the Input Menu - a decision I will expand upon in other documents and my presentation. As a result, its requirements are wholly new, and are listed within each table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The door should be impassable when it is closed* | Enter a level with a door present, and attempt to move through it | The player should become stuck on the door, and unable to pass through | PASS |
| *Touching a door’s relevant button should cause it to slide open, allowing the player to move through* | Enter a level with a door present, and walk into its corresponding button object. Then, attempt to move through the space the door originally occupied | The door should smoothly slide open, and the player can freely move through the area it blocked before | PASS |

TimedButton:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *A timed button should open the door for a specific duration, then close it.* | Enter a level with a door connected to a timed button. Walk into the timed button. | The door should slide open, remain open for a number of seconds, and then slide closed again | PASS |
| *Once activated, a timed button should become disabled until its operation is complete* | Enter a level with a door connected to a timed button. Walk into the timed button and observe the button. Also, attempt to reactivate the button before the door closes again. | The door should slide open, and the button will be disabled until it closes. This is represented by the button turning red. | PASS |

ToggleButton:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *A toggle button should only either open or close a door.* | Enter a level with a toggle button and walk into it. Wait for some time, and then walk into it again | The button should open the door if it is closed, and close it if it is open | PASS |
| *Once activated, a toggle button should become disabled until its operation is complete* | Enter a level with a door connected to a toggle button. Walk into the toggle button and observe the button. Also, attempt to reactivate the button before the door closes again. | The door should slide open, and the button will be disabled until it has been completely opened. This is represented by the button turning red. | PASS |

Level Completion:

LevelCompleteScreen:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| Upon completing a level, the player should be shown their time and whether they collected the level’s collectible. | Enter a level, and complete it without collecting the collectible, and observe the completion screen. Then, replay the level and collect the collectible, and observe the completion screen | The Level Complete screen should display the level’s name, the player’s time for completion, and whether they collected the collectible on this run. | PASS |

LevelCompleteBtns:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| Players should be provided the option to continue to the next level or replay the previous one | Enter a level, and complete it. Observe the Level Complete screen. Click a button and observe its effect. Repeat until all buttons have been tested. | Each button should perform its relevant action: Continuing to the next level, replaying the last one, or returning to the main menu | PASS |

NextLevel:

NextLevel has already been tested as part of testing LevelCompleteBtns, as it is only used when clicking the Next Level button on the Level Completion screen.

Level Details:

Collectible:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| Levels should contain a bonus collectible item to encourage experimentation | Enter each level, and explore it until a floating purple orb is found | Each level should contain one such collectible | PASS |
| *The UI should show if the player has collected the bonus collectible item on the current attempt* | Enter a level, and collect the collectible item and observe the HUD. Then, fail the level by falling into a Pit, and again observe the HUD. Then, collect the collectible and complete the level. | Collecting the collectible should cause it to disappear and show a message on the HUD. Resetting the level should cause the message to disappear, and the collectible to reappear. Completing the level with the collectible should show a message stating that it was collected. | PASS |

LevelData and Timer:

LevelData is a script used only for utility purposes. It is used to pass information between the level scene and the level complete screen. Specifically, it holds the level’s name, player’s final time and the collectible’s status. Other tests have already covered the outcomes of this script’s functionality.

Timer is also a utility script, used to keep track of the player’s time on the current attempt of the level to pass to the HUD, and their final time to pass to the Level Complete screen. Its functionality has also already been tested by testing other scripts.

Menus: Input Menu

InputMenu:

This feature had to be cut due to time constraints on this project. Multiple attempts were made to implement the features this menu would have contained, and I will cover more details on this decision in other documentation. For the purposes of this document, I will still log testing against the requirements, in the interest of completeness.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| The options submenu should provide the player options for camera sensitivity and inversion, as well as key bindings for various actions | Enter the Options menu from the main menu, and navigate to the Input settings. | Options should be presented regarding camera sensitivity and inversion, as well as key rebinding for each action in the game. | FAIL |
| *The player should be able to exit the input menu* | Enter the Options menu, and navigate to Input settings. Then, click the Back button | The player should be navigated back to the Options menu | PASS |

Leaderboard:

I deviated from my original design for the leaderboard somewhat significantly. My original plan was to present the leaderboard on the Level Complete screen only, however my attempts at implementing this made the UI on this screen feel very cluttered. I decided to move this functionality to the Level Select screen, allowing for access to the leaderboard from there, which also required additional functionality to prevent users from viewing leaderboards of levels they have not unlocked. This will be tested in the LevelSelect script.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| The leaderboard should display multiple players’ names and their best recorded times for each level | Populate the leaderboard for a given level with data. Then, navigate to the level’s leaderboard page through Level Select. Observe the leaderboard page | The top 100 players’ usernames and times on the selected level should be visible in a scrollable view, without the entries overflowing | PARTIAL PASS  Database operation appear to fail when the game is run as a built product, but not in the editor |

LeaderboardBack:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to click a Back button to exit the leaderboard page, and return to the Level Select page* | Navigate to a level’s leaderboard page, then click the Back button | The player should be navigated back to Level Select | PASS |

Settings:

ResolutionDropdown:

I did not originally log specific requirements for this page, but I felt that it was necessary to include to meet other non-functional requirements.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to view available screen resolutions in a dropdown menu* | Navigate to the Settings page, and click on the resolution dropdown. Observe the list of available resolutions | The list should dynamically populate based on the connected display, and present a range of resolutions at different aspect ratios and refresh rates | PASS |
| *Selecting a resolution should change the application’s resolution* | Navigate to the settings page, and select a different resolution from the dropdown | The game’s window resolution should change to the one selected | PASS |

SettingsMenu:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to toggle whether the application window is in fullscreen mode using a button* | Navigate to the Settings page, and click on the Toggle Fullscreen button multiple times | If the application was in fullscreen, the application should change to windowed mode on clicking the button, and vice-versa | PASS |
| *The user should be able to navigate to the Input menu and back to the Main menu* | Navigate to the settings page,and click the Input Settings button. Then, return to the settings page and click Back | The player should be able to navigate from this screen to whichever screen they selected using buttons. | PASS |

StartGame:

LeaderboardBtn:

As mentioned in the Leaderboard section, the button to access each level’s leaderboard was moved to the level select screen. The intent with this was to only display the level’s leaderboard button if the level had been unlocked. However, doing so would also prompt a rework of the LevelSelect script, which I was unable to complete in the time I had left.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The leaderboard button for a given level should only be visible if the level is unlocked* | On a new save, navigate to Level Select, and observe the available buttons | The Leaderboard button for each level should only be visible if its corresponding level’s select button is also visible | FAIL |

LevelSelect:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| Level selection should indicate the player’s best time on that level, and whether they have collected its bonus collectible | Navigate to the Level Select screen, and observe the options there | Each level’s best time and collectible status should be shown | PARTIAL PASS  Level collectible status is shown, but not their best time. This would have required database queries for every level, and would introduce lag and instability  Additionally, the player’s progress is only reflected when playing through Unity’s Editor, and not in a built product. |
| *The user should only be able to select levels they have unlocked* | On a new save, navigate to Level Select, and observe the available buttons | The player should only see Level1-1 | PASS |

LevelSelectBack:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to click a Back button to exit the level select page, and return to the Start Game page* | Navigate to level select, then click the Back button | The player should be navigated back to the Start Game page | PASS |

StartGame:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| *The player should be able to click New Game button to start from the first level* | Navigate to Start Game, and click New Game | The player should be loaded into Level1-1 | PASS |

Utility:

DatabaseManager:

This class is already tested by unit tests, and its functionality has already been proven in testing functionality of the leaderboard.

LevelManager:

This script is used for utility purposes, as part of multiple other scripts, providing a conversion from a level’s numeric ID to its scene name in Unity.

PlayerTime:

This class is simply used to give structure to results from DatabaseManager - it has no intrinsic functionality to manually test.

SaveManager:

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Test Action** | **Expected Result** | **Status** |
| Player progress should be saved automatically upon completion of a level | Complete a level that has not been completed yet, and inspect playerProg.json | The file should be updated, such that the levelsCompleted boolean value at the completed level’s ID becomes True | PASS |
| *Collectible status should also be saved to the file* | Complete a level while collecting the collectible, and inspect playerProg.json | The file should be updated, such that the collectibles boolean value at the completed level’s ID becomes True | PASS |

**Non-Functional Testing:**

This section will consist of discussion of the testing approaches I took for each of the non-functional requirements I listed in my Project Requirements document.

**During gameplay, the game should maintain a minimum frame rate of at least 60 frames per second on modern gaming-capable Windows platforms**

* **The game should also maintain its frame rates consistently during gameplay, avoiding stuttering**

My testing approach for this requirement primarily consisted of running the game both in the editor and in a compiled form. In the editor, it became clear that under some circumstances, despite having high frame rates, the editor itself would stutter in Play Mode. I then moved to only testing performance in a built version of the game. I was unable to verify performance on a wide range of platforms, but on my personal system the game ran at a locked 144FPS, the selected refresh rate in the Display Settings, with 1% low FPS (a measure of frame rate consistency - the higher the better) of approximately 100FPS, indicating good stability.

**The game should be compatible with a range of Windows-based computers**

My initial plan to test this involved using a secondary PC I have access to, in order to check the game’s compatibility on a different Windows version. However, external circumstances forced me to swap components from that PC into my own, meaning that I was unable to test on that machine’s installation of Windows.

**Load times should be minimised, taking no longer than 15 seconds to load a new level**

This requirement was tested in a similar way to the first, by simply playing the game in a built state. Levels loaded near-instantly, to the point where I was unable to time the process myself. However, I found that if the database Docker container was offline, the game would hang for some time while it waited for the database query to time out. This process would not take longer than the time set out in the initial requirement, and did not affect the loading of levels, but it is still more time than I felt comfortable waiting on a stuck game.

**The game should be stable and avoid crashes to the desktop**

This requirement cannot be entirely guaranteed, but during my testing there were no crashes. Unity handles errors quite cleanly, and will output non-fatal errors to a log file without crashing the game.

**The user interface should be intuitive and easy to navigate**

* **It should be legible at a range of screen resolutions**
* **It should also scale appropriately with common display aspect ratios**

I tested this by asking individuals unfamiliar with modern games, but familiar with computer systems, to try a number of tasks. I found that my positioning of the Leaderboard button in Level select proved confusing for people, but otherwise my UI was considered easy to navigate. I also tested screen resolutions and aspect ratios by setting the game to windowed mode and selecting a number of resolutions, each at different aspect ratios. The game mostly scaled well, however there were exceptions. On select aspect ratios, particularly ultrawide ones like 21:9, some buttons would fall off the bottom of the screen, with their text becoming illegible. Additionally, the player’s HUD would not scale with resolution, an issue that I had missed until the testing process.

**The game should be designed such that new levels or features can be easily added**

This requirement has been met in some ways, but is lacking in others. When I decided to cut the Input menu, and implement doors, I decided it would be a good point during the project to test this requirement. The implementation of this feature was very quick, with issues only stemming from my misunderstanding of how asynchronous processes work in Unity. However, I did run into some issues with adding the level to level selection. My early decision to base loading levels off of the scene name meant that my system for adding levels was quite fragile in the event that a new level was added with even a small typo. It also required manual addition of the level to the database for the leaderboard, which I feel I should have done at the beginning of the project instead.

**The player should be unable to directly tamper with the database of player times, or send it invalid entries**

Because the player never gets to input data for themselves in this version of the game, there is no risk of invalid data. All processing and validation occurs within the game itself. I must note, however, that because of the database connection implementation I ended up keeping, the integrity of the database is not highly secure. There are connection strings visible in plain text in the code, and they are hardcoded to require a specific port for a Docker Microsoft SQL server. If I were to expand to a real product, I would need some other approach to database hosting, creation and security.

**The game should include a brief guide for how to play, to inform the player of the options they have during gameplay**

I decided to not directly implement this requirement in the game, as I felt that it fell out of the scope of the project. However, I will add documentation for how to play the game.