horizontal line

Grand Traffic Auto

System Validation Document

**24th October 2019**

Project Team

|  |  |
| --- | --- |
| **Name** | **Student Number** |
| Robert Pennefather | 21511164 |
| Daniel Gonzalez | 22024722 |
| Martin Porebski | 21498791 |
| Jacqueline Soon | 21719676 |

[**1.0 Purpose of this document**](#_tah5peo60nr5) **2**

[**2.0 Project Validation Scope**](#_udulaq2rn7sc) **2**

[**3.0 Objectives of Validation**](#_9u9e6ydyjnuu) **3**

[**4.0 Validation Strategies**](#_lrnjazrei935) **3**

[4.1 Task Management](#_ufao4km37ync) 3

[4.2 Game Development](#_xm6o7fvhpf4o) 4

[4.2.1 GUI Usability Tests](#_e4ul6rvfoq2f) 4

[4.2.2 Unit Tests](#_vwdk85k1ikxo) 5

[4.2.3 Unity Test Runner](#_j12azpd8ic8s) 6

[4.3 Web Server Development](#_koa8a6263nch) 8

[4.3.1 GUI Usability Tests](#_6tymlgbjxof2) 8

[4.3.2 Continuous Integration](#_g4mudda2jlwu) 9

[**5.0 Validation Results**](#_4c9qapf4thky) **10**

# 1.0 Purpose of this document

This document describes how software was validated and tested throughout the duration of the project. System validation is divided into three sections:

1. Game Development
2. Web Server Development
3. Continuous Integration

**Game Development** involves the testing and validation of game components within the project which were mostly developed within Unity. The Unity Testing Framework (Unity Test Runner) provided most of the testing functionality required, this framework is based on the .NET library NUnit [[1]](https://docs.unity3d.com/2017.4/Documentation/Manual/testing-editortestsrunner.html). Unity Test Runner allowed the team to create unit tests on the backend and majority of the frontend features. The frontend requirements that could not be validated through unit tests involved testing the design of the GUI..

**Web Server Development** includes the functionality requirements of the server, user and admin log-in, game sessions and the website. The web server is built using .NET and testing was done using GUI navigation. Due to the simplicity of the web server, formal testing frameworks such as Nunit were not used.

**Continuous Integration** involves integrating game development with web server development and ensuring compatibility every time a code update is made. Unity Collaborate was used to run tests on the game, which was then automatically built into WebGL format for use on the server. This meant that the project was compiled and built every time a change was pushed to Unity Collaborate, to ensure that the functionality built in Unity also worked in the WebGL build.

# 2.0 Project Validation Scope

The following components of the project involved software testing and validation:

* Unity Project (Hosted on Unity Collaborate)
* Server Software
* .NET Website
* MongoDB Connections

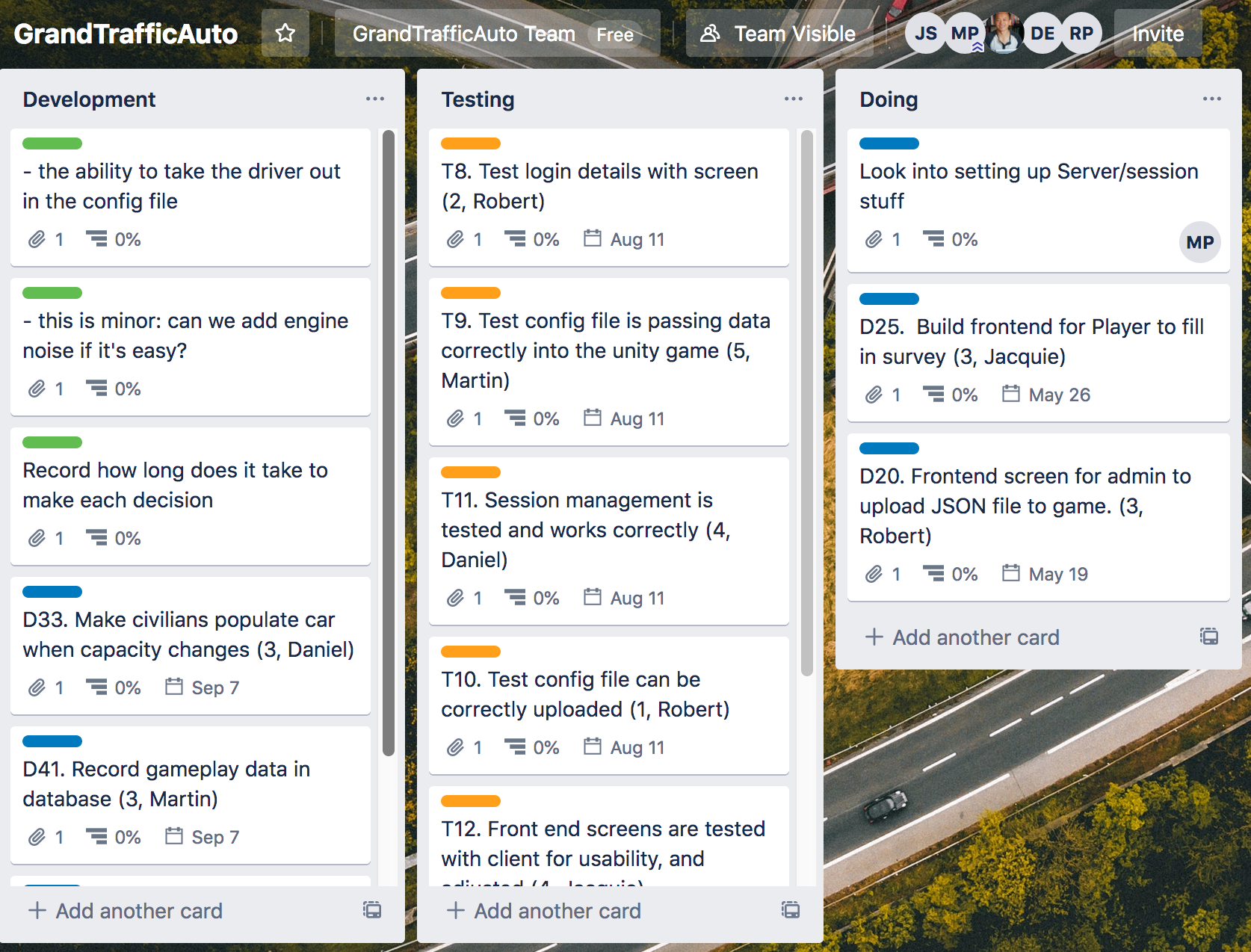
# 3.0 Objectives of Validation

The objective of system validation is to ensure that the software produced meets the pre-defined requirements that were specified by the client. Ongoing testing and quality assurance throughout the project checks the quality of software during development, such that shortcomings can be fixed early on. Validation also ensured that requirements specified in the [‘Functional Specifications’](https://docs.google.com/document/d/16DOOfl_p-Lp0SQT32DF3l0jYS_BuNRxR__oJP6Hx98c/edit#heading=h.qg8dl2pzv6xb) document are met.

# 4.0 Validation Strategies

## 4.1 Task Management

In the early stages of the project, each software requirement was translated into tasks, including tasks specifically for testing. All of these were listed on the teams trello board for project management purposes, where each task was assigned to a group member. The tasks were used as a rough guide indicating what areas and functionality of the software needed testing operations to meet requirements. Cards in Trello labeled in orange indicate they involve ‘Testing’.



## 4.2 Game Development

### 4.2.1 GUI Usability Tests

These tasks were completed by navigating the GUI and using a level of judgement to assess whether the GUI’s response met project requirements. Unity places restrictions on how ‘Monobehaviour’ objects can be tested using Unity Test Runner. Therefore the team found it easier to test the majority of game features through the GUI interface. Remaining functionality that does not depend on scene creation could be tested with unit tests. The following tasks were tested using the GUI.

|  |  |  |
| --- | --- | --- |
| **Overall System Features:**  T9: Graphics are realistic, quality of rendering does not have obvious faults.  T10: Unity build when run on local desktop does not crash  T11: Unity build when run on local desktop does not lag  T19: WebGL runs on chrome & firefox.  T10. Test that config file can be correctly uploaded  T9. Test that config file is passing data correctly into the Unity game.  T33. Test different combinations of csv configurations on the Unity game  T35. Test the sound assets for correct timing | **Drive Functionality:**  T1: When the car moves, wheels rotate  T2: Car steering wheel  turns appropriately with car  T3: When car turns left, wheels also turn left and vice versa.  T5: Collision detection prevents car from colliding with other assets  T6: Player is unable to drive off roads  T7: Player cannot drive off virtual world map  T20: Animation of vehicle driving is smooth and realistic  T34. Test logic of AV routing | **User input:**  T8: When player mouse moves, vision pans accordingly. (180 degree range)  T12: Phone pops up at specified time interval in csv file  T13: User presses button and phone popup goes away  T15: Session code entered connects to relevant csv file configuration in database  T16: Game session id, log of user clicks and user decisions are uploaded to the database  T18: System rejects invalid csv files gracefully i.e. system can recover and request new input. |

### 4.2.2 Unit Tests

The following tasks were tested by creating unit tests:

|  |  |
| --- | --- |
| **Drive Functionality:**  T36: Test that follow script can retrieve correct car coordinates  T37: Test whether minimap car rotation works. | **User Input:**  T4: Car has forward, left, right and reverse movement when selected keyboard input is detected  T9: Only valid session codes and ids can be submitted.  T14: User presses button, money increments by discount rate specified in csv.  T14: User presses button, time increments by time addition rate specified in csv. |

Unit testing was done in *‘PlayMode’* of the Unity Test Runner. *‘PlayMode’* means that the game code is run whilst tests are applied as coroutines [[4]](https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/edit-mode-vs-play-mode-tests.html). There are two types of class behaviours in Unity scripts, regular classes and ‘Monobehaviour Classes’. The latter classes are tied to elements in a game scene and provide basic scene functions such as Update() and Start(), and makes them much more practical for game development, therefore the majority of scripts consist of Monobehaviour Classes.

Testing of the class types differs, Monobehaviour classes must be tied to a game scene when being tested whilst regular classes do not. The team found that testing regular classes was easier because it did not require a game objects to be instantiated. Tests could be run with simple input and output values, much like typically unit tests. An example of a regular test is *‘T36: Test that follow script can retrieve correct car coordinates’,* this unit test is called *‘FollowTests’* in the Unity project.

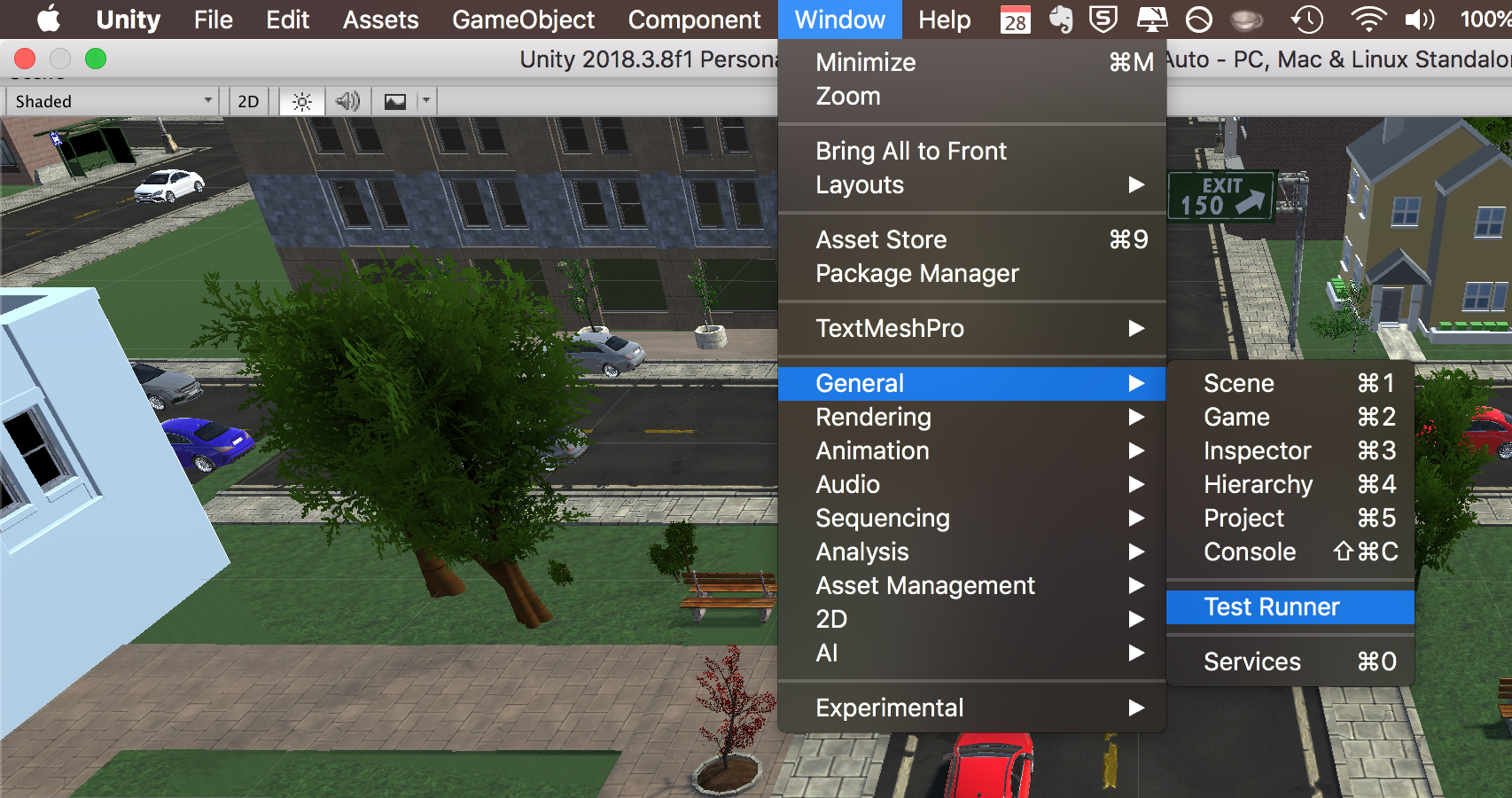
Testing Monobehaviour Classes requires the Monobehaviour object to be initialised, this was done using the Unity Coroutine ‘MonoBehaviourTest’ [[5]](http://monobehaviourtest). This coroutine is able to instantiate the Monobehaviour Class much like it would in a running game scene. Tests involving relationships between multiple scene objects where found to require Monobehaviour test methods. For example, the test *‘T9: Only valid session codes and IDs can be submitted’* checks whether invalid entries in the InputFields of session code and ID modify the errorText object. The figure below shows how the scene should look when test T9 passes.



It is of note that this test T9 can also be done through the GUI. Testing through the GUI is notably easier, particularly for single input/output combinations. In this case, T9 has several input/output values that should be tested such as null cases and invalid game codes, therefore unit tests were written.

### 4.2.3 Unity Test Runner

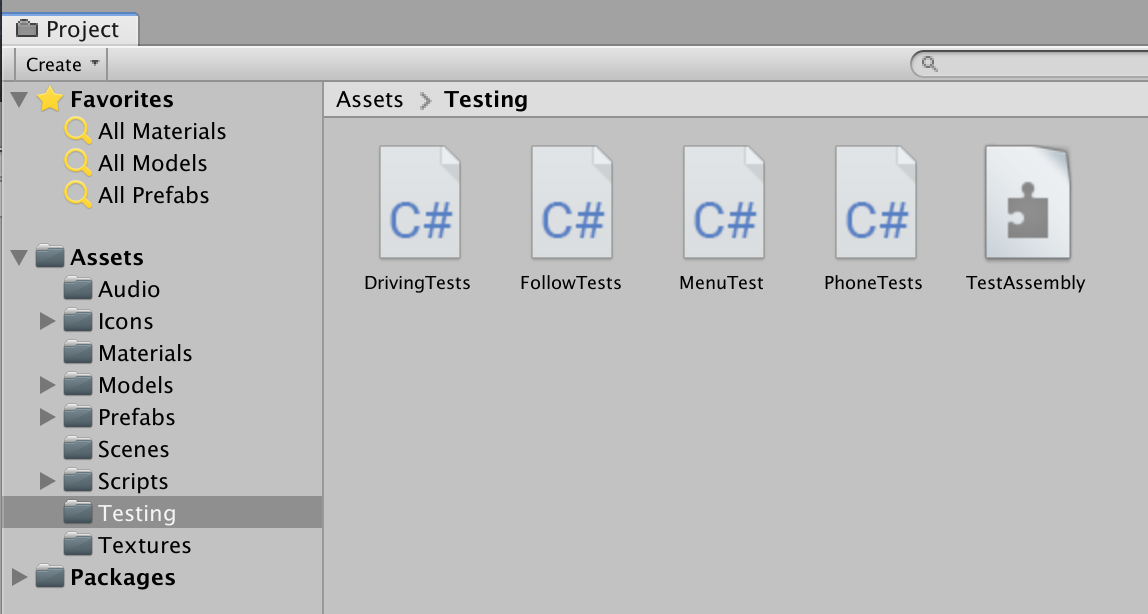
In Unity Desktop, the test runner window can be opened by going to Window → General → Test Runner.



Once opened, the test runner window shows all the tests that have been saved within the *‘Play mode’* test folder. The test runner can be started by selecting ‘Run All’ in the dialog box, it will then display a status, indicated by a tick or cross, followed by the conducted test.



In this project test scripts can be found in ‘Assets/Testing’. Note that an additional file called ‘Test Assembly’ must be created within the test script folder. Unity is unable to find class definitions on its own, this test assembly defines where to look for class scripts i.e. in ‘Assets/Scripts’.



## 

## 4.3 Web Server Development

### 4.3.1 GUI Usability Tests

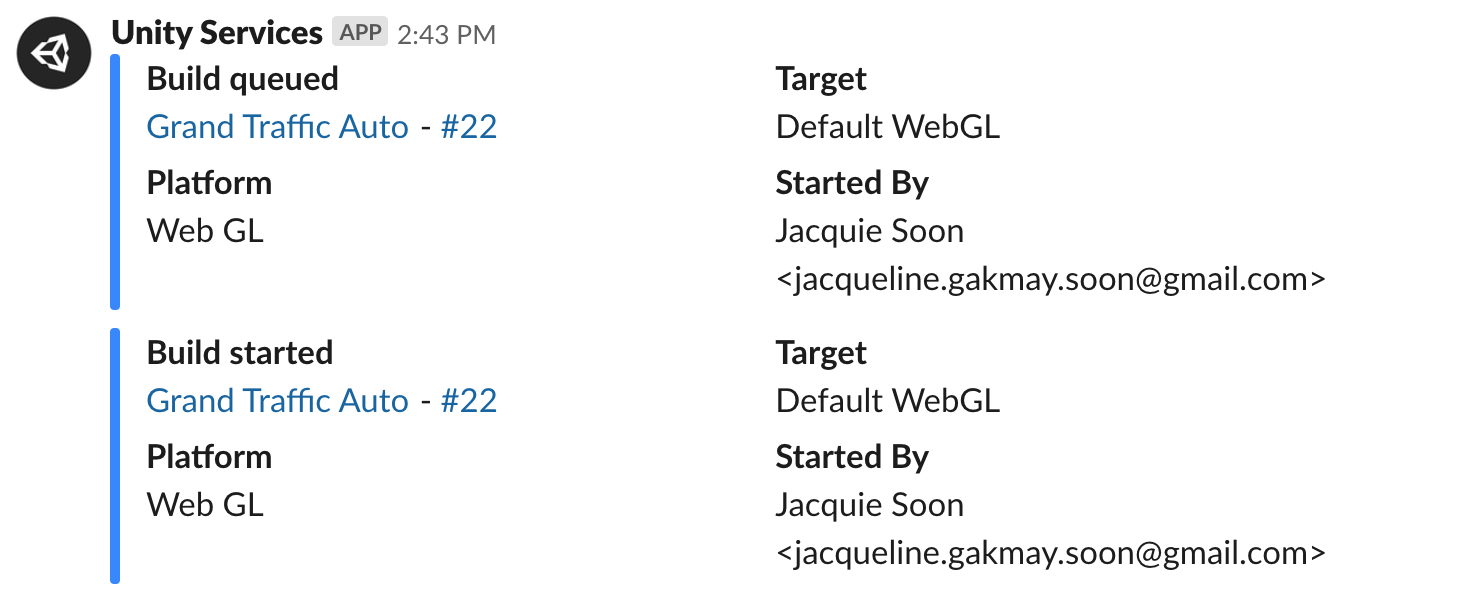
Server tasks were tested by navigating the website GUI and MongoDB web app. The functionality of the web application is simple enough that a testing framework, such as Nunit, is not necessary. Almost all features involve the GUI which makes it easier to test through the apps interface.

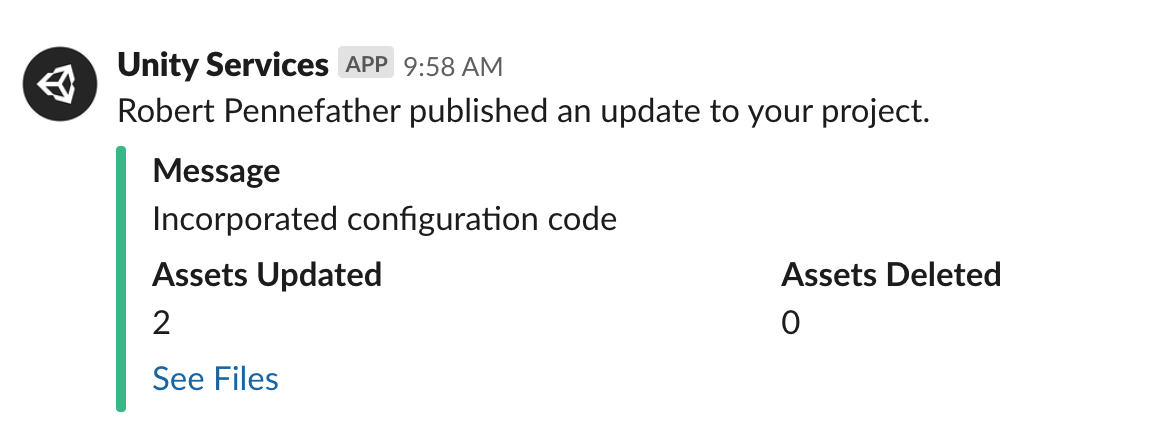
The database (MongoDB) provides an online application where data collections pushed from the game can be viewed. This makes it easy to see whether data has been correctly passed from the server to MongoDB. Therefore the team decided there was no need for unit tests on the database.

|  |  |  |
| --- | --- | --- |
| **Server Performance:**  T17: Server can handle up to 20 users at once  T21: Test Browser compatibility and performance for WebGL  T22: Server allows admin panel to be accessed using secure credentials  T31: Link to game is provided to admin after configurations are set.  T32. Session management is tested and works correctly | **Database:**  T26: Database stores the GameID, SessionID, Player choices and time of each choice.  T27: Database can be connected to from UWAs network  T25: Database can handle 20 players using the web server.  T35. Test different combinations of csv configurations on the Database | **User Input**  T23: Players can register on the website using email name and session code.  T24: System handles invalid email format or session codes gracefully i.e. prompt to enter correct fields occurs and the system does not crash.  T29: Admin can specify the starting amount, time scale, money scale, number of trials, request times, travel times and diversion times.  T30: System handles invalid input gracefully i.e. prompt is displayed that asks for input to be corrected instead of crashing. |

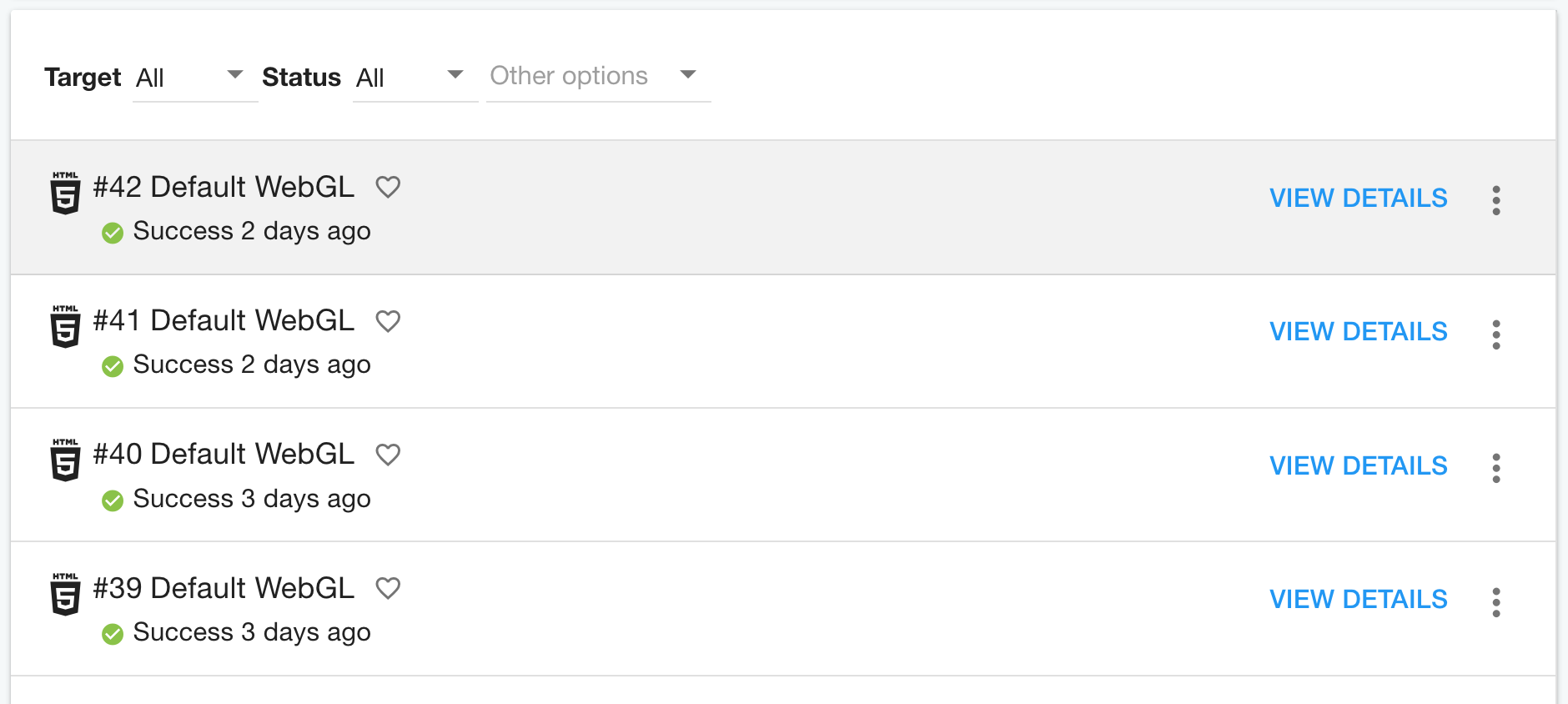
### 4.3.2 Continuous Integration

Continuous Integration involved building the project every time there was a change. Unity Collaborate connects to Unity Cloud Build [[2]](https://docs.unity3d.com/560/Documentation/Manual/UnityCloudBuildContinuousIntegration.html). Every time a team member committed their updates, Unity Cloud Build automatically compiles the project. This allowed the team to know when and in which project version errors occurred. Unity Cloud Build also allows the project to be restored to the last good commit. Every time there is an error on Unity Cloud Build, all team members are notified via Slack using the Unity Add-on. This ensures that a team member can quickly see the error, fix it and push a new update. The images below show the Slack notifications received after Unity updates and builds:





Unity Cloud Build allows the project to be complied using various platforms. In this case, the project could be compiled into WebGL format [[3]](https://docs.unity3d.com/560/Documentation/Manual/UnityCloudBuildSupportedPlatforms.html). Because WebGL is used for the web server, this immediately ensures compatibility between Unity and the web application. This allowed the team to apply continuous integration all the way from game development to web server development. The image below shows how successful WebGL builds were displayed after projet updates.



# 5.0 Validation Results

Most of the tests from the GUI interface and Unit testing passed, the ones that did not meet requirements were discussed with the client. These requirements were modified in agreeance with the client. The table below shows the tasks that required changes:

|  |  |
| --- | --- |
| **Testing Task** | **Reason for change in requirement** |
| T17: Server can handle up to 20 users at once | The team did not test the server with 20 people at once, instead cpu on the server was monitored as 5 people gradually joined. We noticed that adding additional people had a negligible effect on server’s performance. This is because the game is downloaded by the browser and loaded on the client side which means that it runs off the local machine’s cpu. We demonstrated this experiment to the client who was satisfied with these outcomes. |
| T27: Database can be connected to from UWAs network | Database connections from MongoDB work outside of UWA’s network. However, when the tests were done on campus, MongoDB was not able to connect to UWA’s server possibly due to firewall protections. The client was notified of this issue and may need to make a request through UWA’s IT Department to allow MongoDB connections to work. |
| T25: Database can handle 20 players using the web server. | The team was unable to test the database on more than one person due to the issue with T27. This will most likely not be an issue once Database connections are fixed. The free tier (M0 Sandbox) on MongoDB Atlas has the capacity for 100 connections which is more than enough for this project [[6]](https://docs.atlas.mongodb.com/connection-limits/). |