2 | Evaluation and Testing Report

2.1 | Approach to Evaluation and Testing

We have implemented multiple changes and new features during this production cycle such that the game will fulfil the updated requirements and brief. This brief was laid out at the beginning of the assessment and this evaluation report will examine how well the initial brief has been fulfilled by the game.

After the code was finished we performed unit tests on all of our code. This seemed like the best choice as they allow for reliable testing and evaluation of code. To evaluate whether the end-product meets the client brief, we read through the updated requirements and customer specification and play tested the game to determine whether it met the deliverables to a satisfactory extent. These methods were followed by play-testing from outsiders to obtain fair and unbiased data. To determine if our code was good quality we considered several factors to do with maintainability. These include how easy the code is to read/understand, correctly named variables, correct use of specific programming techniques and creating efficient code where applicable. It was also important that the code works as intended at all times. Throughout play-testing we evaluated how well the game met its brief and fixed any issues with the game that arose from play-testing.

Play-testing- https://ojp508.github.io/Mambo-Studios/TestTable4.pdf

After completing a first edition of the final game we decided the best way to conduct unbiased testing of game quality, enjoyment and successfulness was to conduct play tests. This method of testing was not done in previous iterations but we feel that as this is the final version, personal input is as important as functionality to make the game enjoyable.

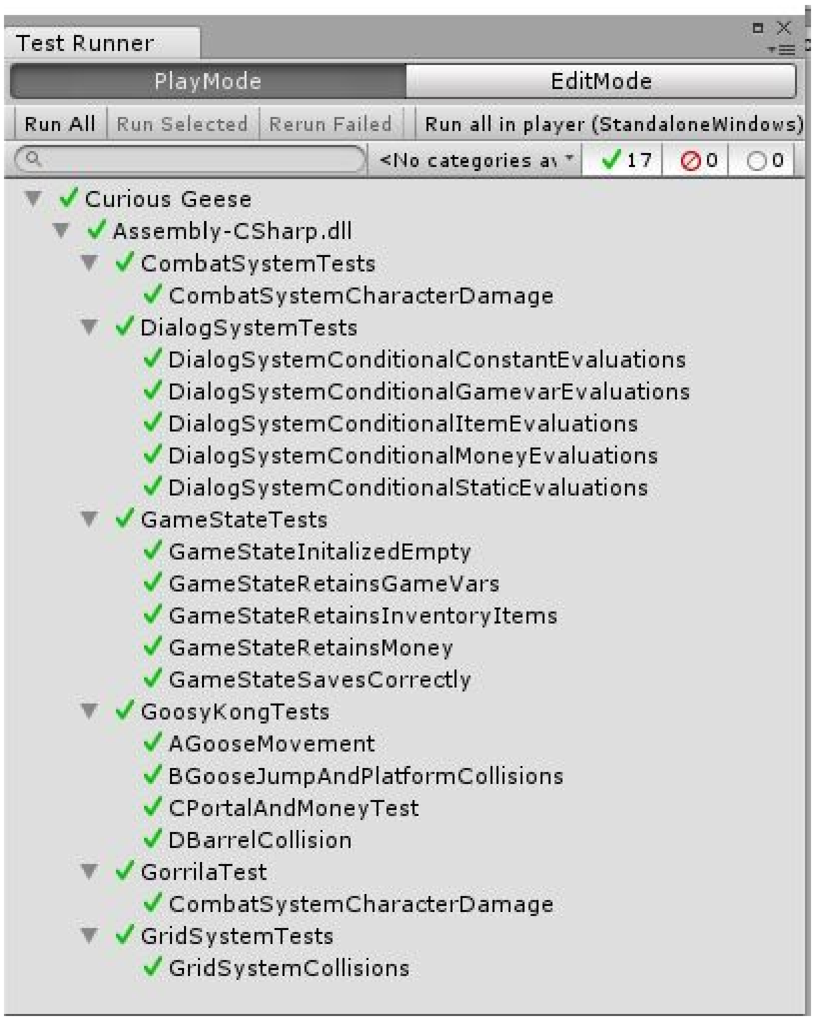
The advantage of play tests is that each participant is brand new to the game. This means they will make mistakes, both in playthrough and in time, whereas the game designers know the exact path to take and the optimal solution to every battle. This process is thus more to identify bugs and flaws, over testing by ourselves.

To conduct this testing, we set out a table to record the time, deaths and level reached as well as any main comments from the testers. The comments section was often the most useful as the raw opinions of a small group of testers can be very informative of how well the game works in smoothness and playability. This allows us to get a picture of exactly what a player would experience when playing the game. During testing our participants played the game uninterrupted with no help from the tester. They were given a loaded version of the game and told they could access the manual (which was loaded in a browser).

The results of the testing were very informative. Firstly they provided a list of bugs which had not been identified through our own play throughs, giving us the ability to fix and improve on the broken features. The second gain was an insight into the quality of our game. Quality is a very difficult value to measure as its somewhat subjective; for example this is an amateur creation with limited variance and very set features that must be implemented so it can't possibly compare to professional titles. This means that when dissecting comments from the tests we must take only what is fixable and not get caught up trying to implement something beyond our capabilities or too beyond the client's requirements.

After reviewing the test data it was decided we needed to focus more on teaching players to interact with the game's systems. This was due to the low proportion of players actually finishing the game. It is worth noting that this was not due to the game being too hard. Instead, it was due to the fact that players weren't using all available upgrades and characters and the final boss had been purposefully made more difficult so that players would need to upgrade and change their characters to defeat him (fulfilling F6 and F4) Otherwise, there would be no use for items and improvement if the game was beatable without them. To fix this issue we implemented an introduction at the beginning of the game to introduce the commands and keys, in the hope that with this, the player will be more knowledgeable about the available game systems and able to progress further within the game.

2.2 | Unit Testing



```
▼ ✓ Curious Geese
  ▼ ✓ Assembly-CSharp.dll
    ▼ ✓ CombatSystemTests
        ✓ CombatSystemCharacterDamage
    ▼ ✓ DialogSystemTests
        ✓ DialogSystemConditionalConstantEvaluations
        ✓ DialogSystemConditionalGamevarEvaluations
        ✓ DialogSystemConditionalItemEvaluations
        ✓ DialogSystemConditionalMoneyEvaluations
        ✓ DialogSystemConditionalStaticEvaluations
    ▼ ✓ GameStateTests
        ✓ GameStateInitalizedEmpty
        ✓ GameStateRetainsGameVars
        ✓ GameStateRetainsInventoryItems
        ✓ GameStateRetainsMoney
        ✓ GameStateSavesCorrectly
    ▼ ✓ GoosyKongTests
        ✓ AGooseMovement
        ✓ BGooseJumpAndPlatformCollisions
        ✓ CPortalAndMoneyTest
        ✓ DBarrelCollision
    ▼ ✓ GorrilaTest
        ✓ CombatSystemCharacterDamage
    ▼ ✓ GridSystemTests
        ✓ GridSystemCollisions
```

Our Unit testing regime is shown above, as you can see all tests that we inherited from the previous team passed and our new GorrilaTest (testing the Gorilla's attack correctly functions) also passed. This test is based on the CombatSystemCharacterDamage test which we inherited from Coffeeless but is held in a separate testing category for clarity.

Our system for Quests and Quest rewards is tested by the inherited GameStateTests as it uses the inherited Saving and GameState Variable functionality for its implementation, meaning the inherited tests of those systems, in addition to playtesting, allows us to be confident in the Quest's functionality.

As part of minimising changes we aimed to ensure that all inherited unit tests passed. This also ensures that any inherited functionality of the game is not lost, reducing the likelihood of compromising inherited Requirements.

2.3 | Meeting the Requirements

In terms of meeting the requirements in the final version of the product, Coffeeless used slightly different requirements structuring, wherein they brokedown NF requirements using the VOLERE template. This was a different approach to how we initially created our requirements, but presented each requirement in more detail, making them easier to implement and understand.

The requirements document shows the majority of the requirements they identified were already implemented (highlighted in green). Upon reviewing these, and playing through the game ourselves, we concluded that all the requirements listed had been met to a satisfactory standard, meaning we had a lot more time to focus on the new requirements we were given in this part of the assessment. The URL for the final requirements which are all met within the game can be found here: https://ojp508.github.io/Mambo-Studios/Req4.pdf

In terms of the functional requirements implemented, those which focus on the basic structure of the game (such as having 6 characters and 10 map locations) are clearly implemented and are obvious immediately after playing the game. Those which are more subtly implemented (such as character improvement throughout the game) are less obvious upon initial play through and could be improved by adding tutorials in the game to make the player more weary of the their impact. However these were still met to an extent in the game.

Coffeeless' usability requirements were met as it was clear from play-testing that the game menu allows for controls to be adjusted to meet specific user needs, as well as testing with windows accessibility settings to assist users when needed whilst playing the game. In terms of the performance requirements, it is clear that using Unity to build the game makes its performance more than adequate for use in all lab computers at a playable level as the requirement states. The demo mode also meets the operational requirements of the game, as it is suitable for demonstration, and also meets the criteria of containing solely PG-13 material. Moreover, the game has been tested and is playable completely offline, storing all data locally, meaning the security requirements are fully implemented.

In terms of new requirements, the gorilla character has been implemented such that it will randomly attack the player party (F17). In order to meet the requirement that the Gorilla is still an asset to the player party, the Gorilla attacks enemies the majority of the time and when the Gorilla does attack the player characters the player has a choice of who they attack. The Gorilla also has an ability that deals reduced damage, if a player wants to use the Gorilla with reduced risks. The requirement to implement quests is met within F18, as the user can select 2 quests at the main menu which add narrative, enemies, and change the main questline of the game. They each contain 3 objectives (F18) and give the player in-game currency (F19), meaning the requirement is satisfied by allowing quests to be picked before the game begins, and also by making the quests rewarding to the player.