CZ3003 - Software System Analysis & Design

# Lab 4 Deliverables

Project Name: Food Wars

Group Name: Team 1

Lab group: TDDP3

Date of Submission: 11 November 2021

| Group Member | Matric No. |
|---|---|
| David Tay Ang Peng | U1910603L |
| Grace Ong Yong Han | U1721575H |
| Jordon Kho Junyang | U1920297F |
| Lim Wei Rong | U1921791D |
| Ryan Tan Yu Xiang | U1922774F |
| Joy Cheng Zhaoyi | U1922716L |
| Tang Hoong Jing | U1721417E |
| Guo Wanyao | U1822530E |
| Ng Wee Hau, Zaphyr | U1822044D |
| Lee Kai Jie, John | U1921862J |
| Chio Ting Kiat | U1720465K |

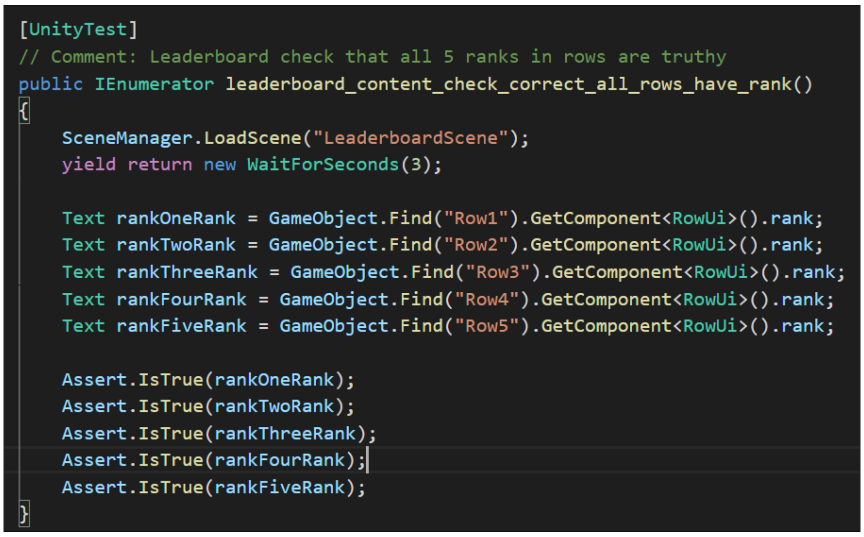
## Table of Contents

## 1 Test stubs and drivers

Stubs and drivers refer to replicas of the modules that act as substitutes to the undeveloped portion or missing module.They are specially developed to meet the necessary requirements of the undeveloped module by identifying the missing behaviour required in the module. This ensures that the right functionalities will be developed. Stubs are used in top-down integration while drivers are used in bottom-up integration testing. Collectively, they ensure that both the upper and lower modules of the system are properly and tested in detail.

In our Food Wars Application, we used Stubs to test the upper level interface Scenes in Unity that have not been developed yet. Specifically, we created mock class Managers which are attached to the scenes to simulate the output for the interfaces. This allows us to verify the location of these outputs on the interface and ensure that the right output and format is being displayed to the user. While the actual Managers are still being developed, the use of Stubs allow the UI team to test their scene outputs and interactions of the user with the interface despite having the missing Manager class.

The following is a sample snippet from our testing scripts which is testing for whether the leaderboard is arranged in descending order by elo:



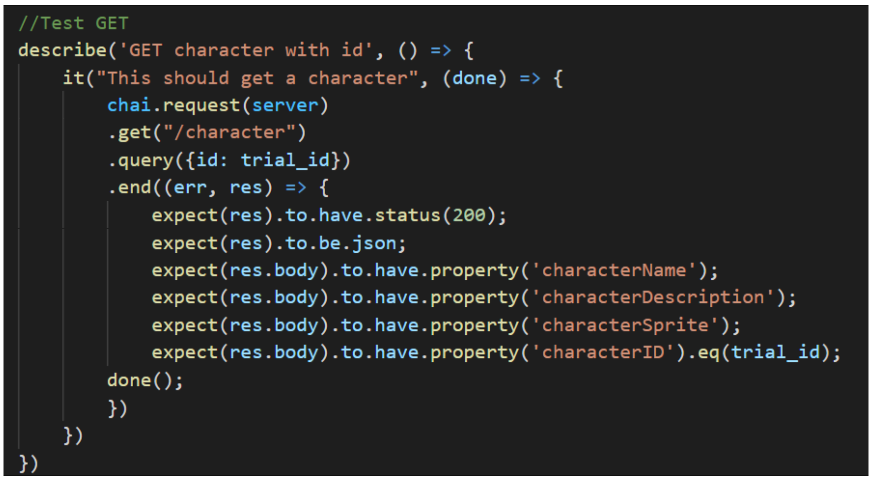
```
[UnityTest]
// Comment: Leaderboard check that all 5 ranks in rows are truthy
public IEnumerator leaderboard_content_check_correct_all_rows_have_rank()
{
    SceneManager.LoadScene("LeaderboardScene");
    yield return new WaitForSeconds(3);

    Text rankOneRank = GameObject.Find("Row1").GetComponent<RowUi>().rank;
    Text rankTwoRank = GameObject.Find("Row2").GetComponent<RowUi>().rank;
    Text rankThreeRank = GameObject.Find("Row3").GetComponent<RowUi>().rank;
    Text rankFourRank = GameObject.Find("Row4").GetComponent<RowUi>().rank;
    Text rankFiveRank = GameObject.Find("Row5").GetComponent<RowUi>().rank;

    Assert.IsTrue(rankOneRank);
    Assert.IsTrue(rankTwoRank);
    Assert.IsTrue(rankThreeRank);
    Assert.IsTrue(rankFourRank);
    Assert.IsTrue(rankFiveRank);
}
```

We also used Drivers to test the backend API that we developed for the database. Since the logic and request for data in the Managers are incomplete, it was useful to use a Driver to test the backend API calls to the database. This ensured that our API calls would return the right information in a valid and acceptable format for the Manager class when it was completed. The approach we used was to print the stack trace error message on the Console to verify the errors encountered by the backend API scripts.

The following is a sample snippet from our testing scripts which is testing for whether the character object retrieved from the server has the necessary properties:



```
//Test GET
describe('GET character with id', () => {
    it("This should get a character", (done) => {
        chai.request(server)
        .get("/character")
        .query({id: trial_id})
        .end((err, res) => {
            expect(res).to.have.status(200);
            expect(res).to.be.json;
            expect(res.body).to.have.property('characterName');
            expect(res.body).to.have.property('characterDescription');
            expect(res.body).to.have.property('characterSprite');
            expect(res.body).to.have.property('characterID').eq(trial_id);
        done();
        })
    })
})
```

To further ensure that our components had the valid and correct outputs, we further incorporated the use of unit testing for both the Unity component and the backend component.

## 2 Unit Testing

Unit Testing is a type of testing whereby the main purpose is to test the individual units or function. A unit is the smallest testable part of an application mainly having one or few inputs and produces a single output.

Unit Testing isolates a section of the code and verifies its correctness. While it has three forms (1) White Box Testing (2) Black Box Testing & (3) Grey Box Testing, our team decided to go with (2) Black Box Testing as our main goal was to test how well our components conforms to the requirements stated in our Software Requirement Document(SRS).

Some of the Different Black Box Testing Techniques are as follows:
(A) Equivalence Partitioning
(B) Boundary value Analysis
(C) Cause-Effect Graphing Techniques
(D) Comparison Testing
(E) Fuzz Testing
(F) Model-based testing.
In our testing, we mainly used (A) Equivalence Partitioning & (F) Model-based testing which will be further elaborated below.

In our project, we mainly use 2 tools to help us with our automated unit testing, Unity Testing Framework & Mocha & Chai Testing which we will be going into more details about.

## 2.1 Unity Testing Framework

As our game is developed in Unity, after evaluating various Unit Testing Tools such as Python UnitTest, Selenium & AltUnityTester, we have chosen to use the Unity Testing Framework as it is a tool that allows us to test our code in both the Edit Mode & Play Mode. It also has the best support & documentations which would allow for accelerated learning & testing.

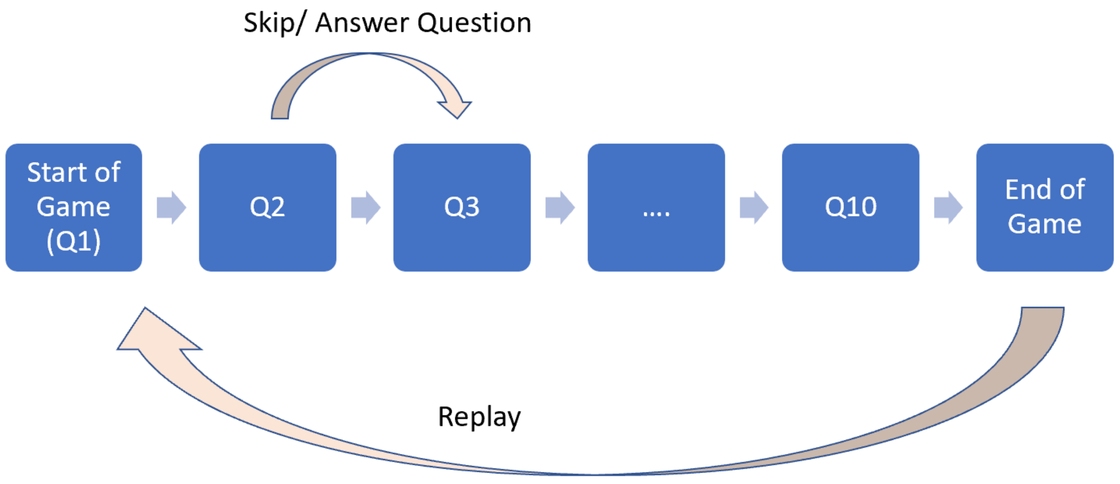
A total of 33 test cases were conducted for the various components.

In the Login & Register Testing Cases, we mainly used Equivalence Partitioning technique. Equivalence Partitioning technique is a technique of software testing in which input data is divided into partitions of valid & invalid values, and that it is mandatory that all partitions should exhibit the same behaviour. For instance in our login test cases,

| Invalid<br>2 test cases | Invalid<br>3 test cases | Valid<br>1 test case |
|---|---|---|
| Missing input fields | Wrong Inputs<br>( not registered email/ wrong password etc) | Valid & registered email address & password |

For the rest of the unit testing, we mainly used Model Based Testing. Model Based Software testing is a software testing technique whereby the runtime of a software under test is checked against predictions made by the model. It describes how a system behaves in response to an action and see if the system responds as per expectations. While there are two times of Model Based Testing, offline & online, we have mainly chosen to use the offline method: i.e. generating test cases before executing it ( in comparison to generation of test suites during execution).

For instance in our single player mode testing, a finite state machine model helps the testers to assess the results which result in a corresponding state of the system.



## 2.2 Mocha And Chai Testing

Mocha & Chai have been used in our project for backend unit testing. Mocha is a JavaScript test framework running on Node.js and in the browser. Mocha allows asynchronous testing, test coverage reports, and use of any assertion library. Chai is a BDD (Behavior Driven Development) / TDD (Test Driven Development) assertion library for NodeJS and the browser that can be paired with any javascript testing framework.

A total of 35 test cases were conducted for the various APIs: Testing Character API, Item API, Question API, Restaurant API, User API. We mainly used the Equivalent Partitioning technique as described above, whereby a set of valid parameters were tested followed by invalid parameters and then missing parameters, with all partitions exhibiting the same behaviour.

For instance in our Question API testing GET question test cases,

| Invalid<br>3 test cases | Valid<br>1 test case |
|---|---|
| Invalid parameters (pri 0, pri 7) , Missing Parameters | All parameters valid & present. |
| Error message | Gets all question for the specified primary level |

## 3 Performance Testing

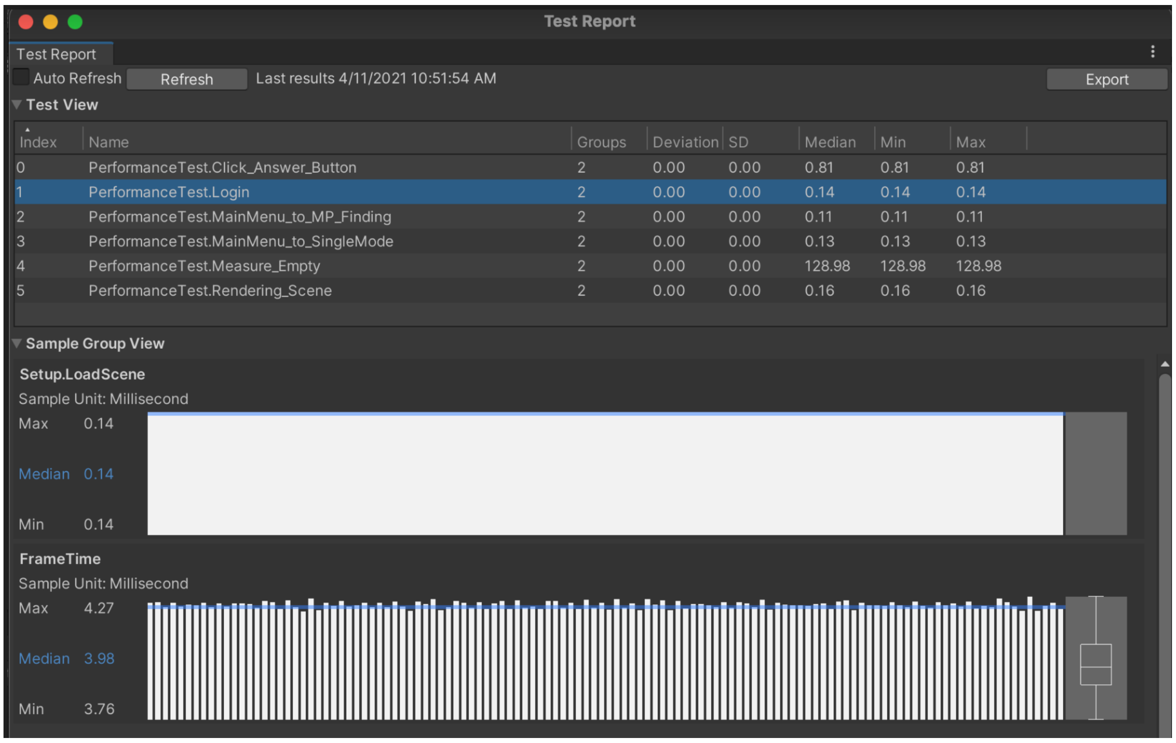
It is important to conduct performance testing to assess how our application will perform under a given load. In our project, we made use of the Performance Testing Extension for Unity Test Runner to conduct 6 different testing scenarios. The Unity Performance Testing Extension is a Unity Editor Package that provides APIs & test case decorators to make it easier to take measurements/ samples of Unity Profile Markers, and other customer metrics outside of the profiler.

### 3.1 Measuring Speed

Different scenarios were tested to measure the average speeds of completing these scenarios as well as the time taken to load each scene. This is an important measurement as a more responsive game would encourage users to stay on the game and have a better user experience when playing our game.

It took an average of 0.11 to 0.81 milliseconds to load scenes on our game. This is relatively responsive attributing to our design decision to build the game in 2D, allowing for our scenes to render at a much faster speed.

Below is an example of a test result on one of the scenarios that we have run. The rest of the scenario output file can be found in our submission folder.



Last results 4/11/2021 10:51:54 AM

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Setup.LoadScene
Sample Unit: Millisecond
Max 0.14
Median 0.14
Min 0.14

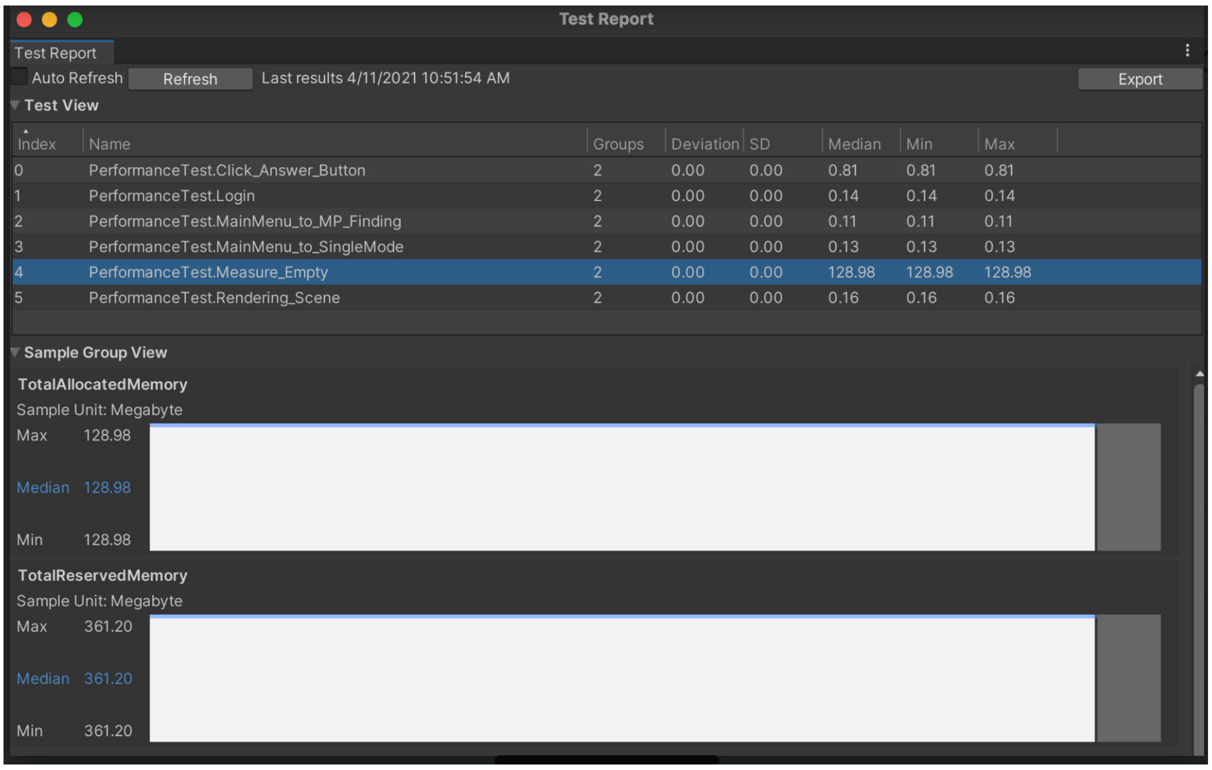
FrameTime
Sample Unit: Millisecond
Max 4.27
Median 3.98
Min 3.76

**Figure 1**: Output Results of Login Scenario

## 3.2 Measuring Memory Usage

We have also tested the allocated memory to be used which turns out to be about 129 MB, which isn't too big to run on normal computers. This would allow our game to be played by almost all students as you wont need an expensive computer to run the game, which hits our target group of audience well because students aren't making any money.

Below is an example of a test result of the total allocated memory for our game.



Last results 4/11/2021 10:51:54 AM

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

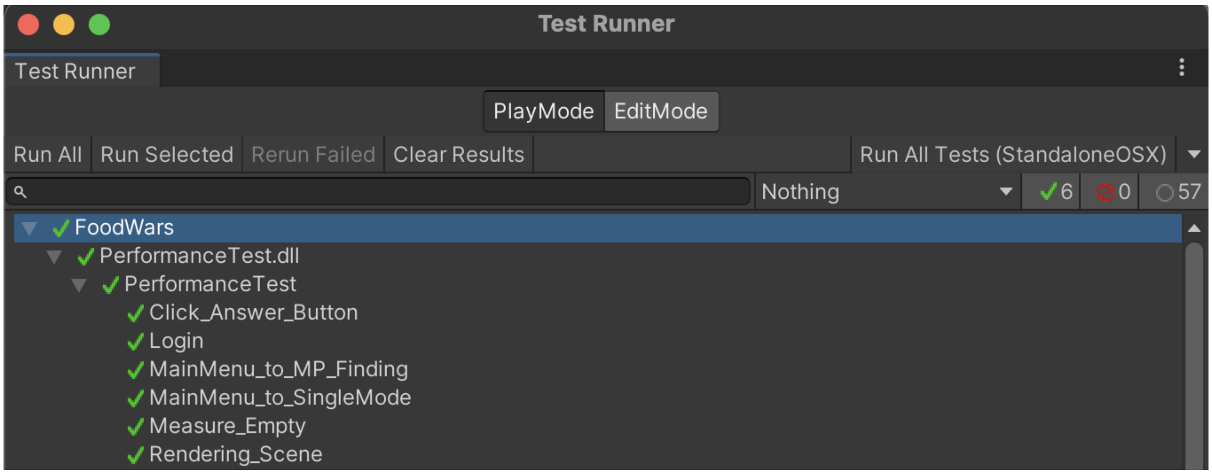
TotalAllocatedMemory
Sample Unit: Megabyte
Max 128.98
Median 128.98
Min 128.98

TotalReservedMemory
Sample Unit: Megabyte
Max 361.20
Median 361.20
Min 361.20

**Figure 2**: Allocated & Reserved Memory

## 3.3 Performance Testing Summary

For our performance testing of the game, we have used the automated performance testing extension, "`Unity.PerformanceTesting`". A total of 6 test cases were conducted and all of them have passed. We will go into the details for each component's testing in the document below.

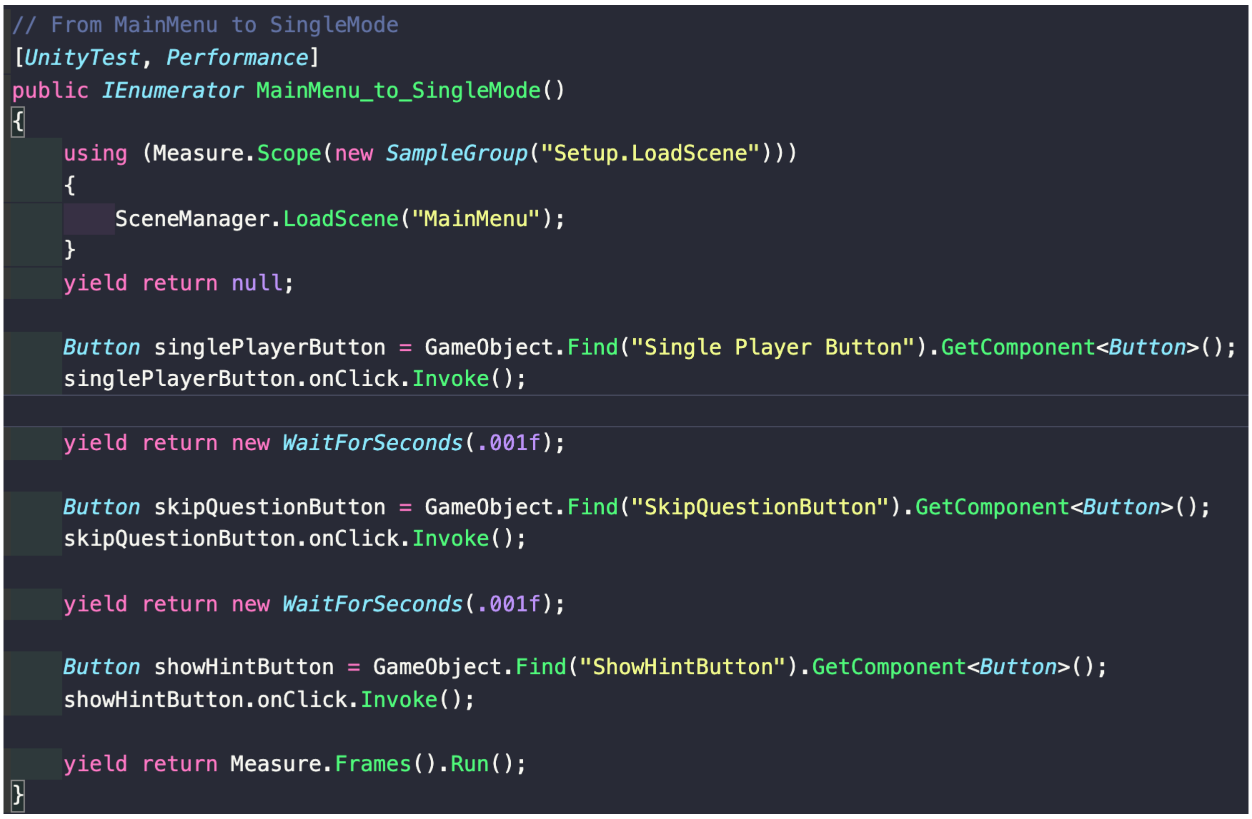


```
FoodWars
  PerformanceTest.dll
    PerformanceTest
      Click_Answer_Button
      Login
      MainMenu_to_MP_Finding
      MainMenu_to_SingleMode
      Measure_Empty
      Rendering_Scene
```

**Figure 3**: All Performance Tests Passed

### 3.3.1 Sample Script

The following is a sample snippet from our performance testing scripts which is to measure how long it takes to transit from the Main Menu Scene to the Single Player Game Scene:



```
// From MainMenu to SingleMode
[UnityTest, Performance]
public IEnumerator MainMenu_to_SingleMode()
{
    using (Measure.Scope(new SampleGroup("Setup.LoadScene")))
    {
        SceneManager.LoadScene("MainMenu");
    }
    yield return null;

    Button singlePlayerButton = GameObject.Find("Single Player Button").GetComponent<Button>();
    singlePlayerButton.onClick.Invoke();

    yield return new WaitForSeconds(.001f);

    Button skipQuestionButton = GameObject.Find("SkipQuestionButton").GetComponent<Button>();
    skipQuestionButton.onClick.Invoke();

    yield return new WaitForSeconds(.001f);

    Button showHintButton = GameObject.Find("ShowHintButton").GetComponent<Button>();
    showHintButton.onClick.Invoke();

    yield return Measure.Frames().Run();
}
```

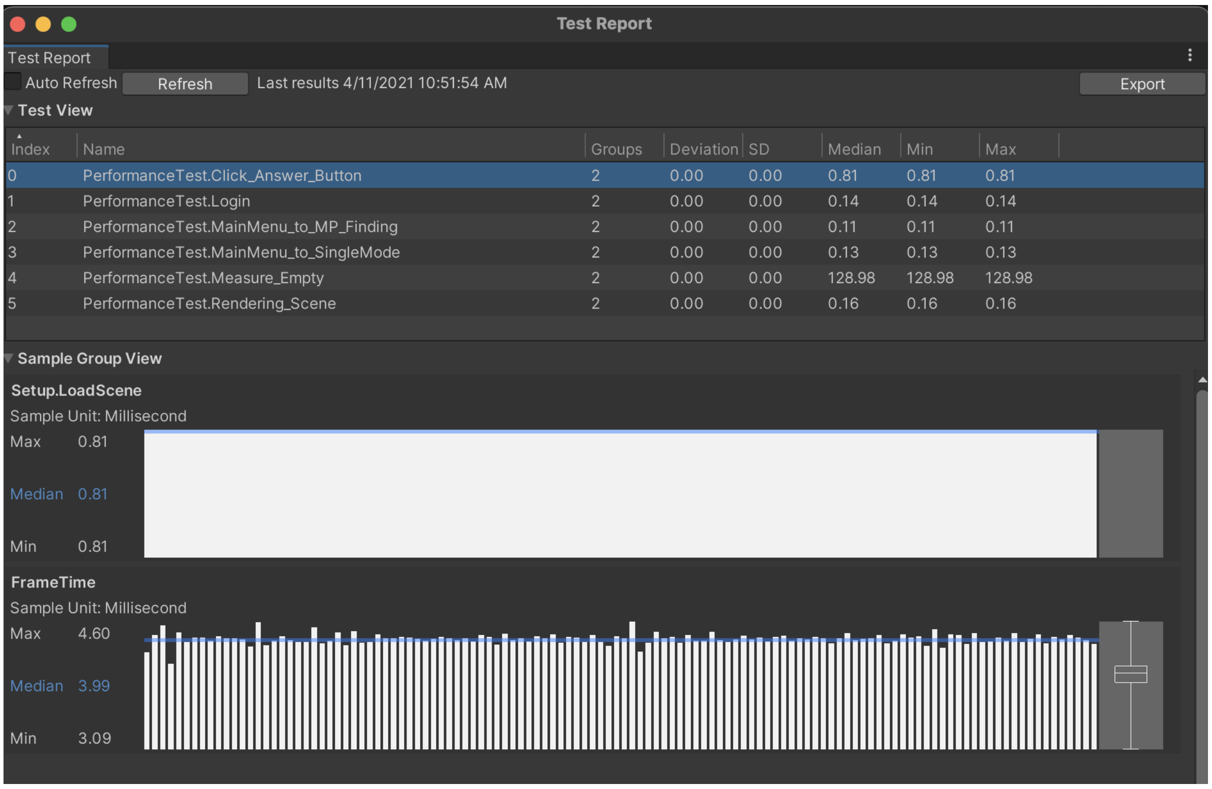
**Figure 4**: Sample Script for Performance Testing

## 3.3.2 Test Cases

| Test # | Test Name | Description | Status |
|---|---|---|---|
| 1 | Login() | Measure time taken for the complete process of user logging in | **pass** |
| 2 | Measure_Empty() | Custom measurement to capture total allocated and reserved memory | **pass** |
| 3 | Rendering_Scene() | Scene measurement | **pass** |
| 4 | MainMenu_to_SingleMode() | Measure time taken for the complete process transitioning from main menu scene to single player gane scene | **pass** |
| 5 | MainMenu_to_MP_Finding() | Measure time taken for the complete process transitioning from main menu scene to matching player scene | **pass** |
| 6 | Click_Answer_Button() | Measure time taken to get the button content updated after it is being clicked | **pass** |

The following figures show the detailed results of each performance test as mentioned above respectively:

**<u>Click_Answer_Button()</u>**



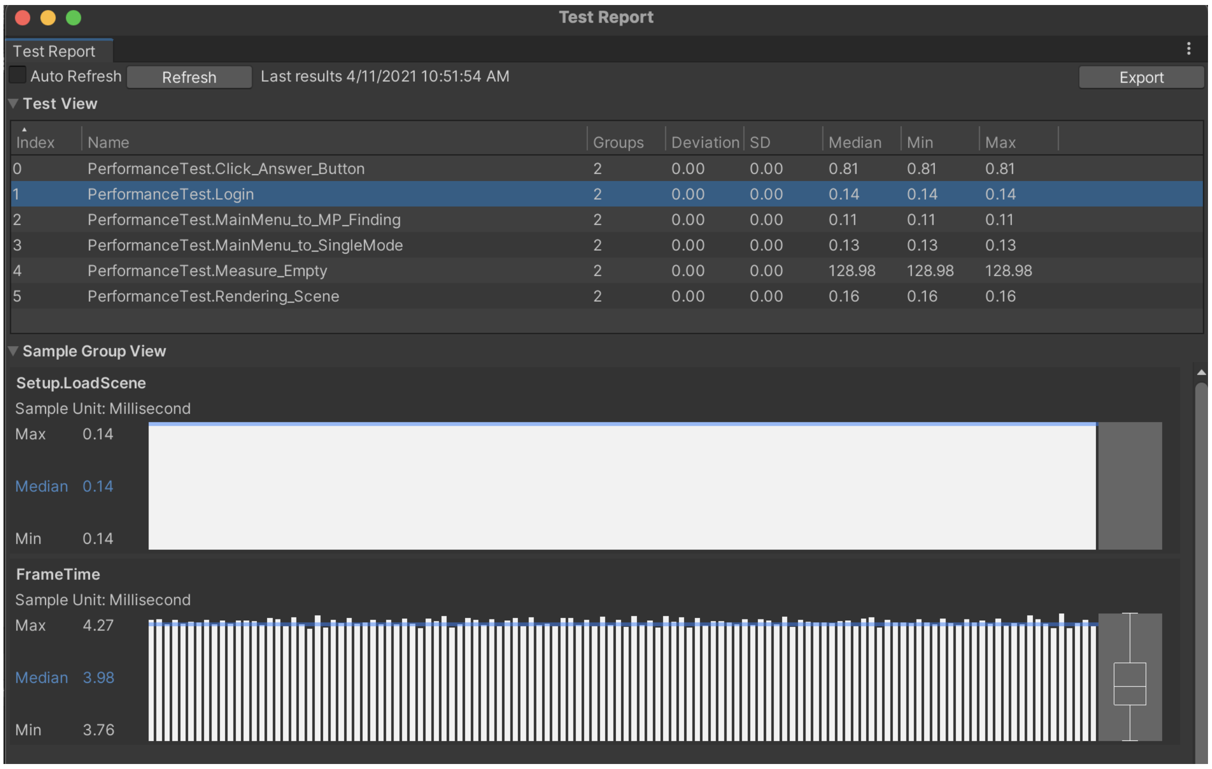
Last results 4/11/2021 10:51:54 AM

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Setup.LoadScene
Sample Unit: Millisecond
Max 0.81
Median 0.81
Min 0.81

FrameTime
Sample Unit: Millisecond
Max 4.60
Median 3.99
Min 3.09

## Login()



Last results 4/11/2021 10:51:54 AM

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Sample Group View

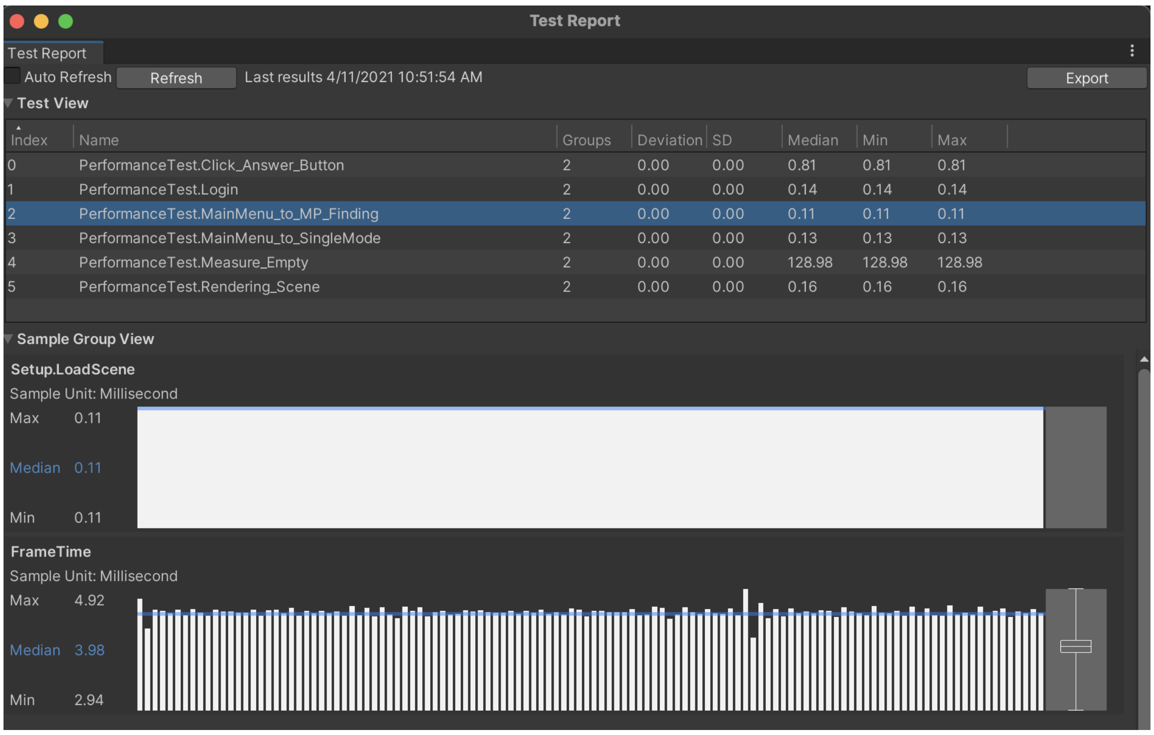
Setup.LoadScene
Sample Unit: Millisecond

| | |
|---|---|
| Max | 0.14 |
| Median | 0.14 |
| Min | 0.14 |

FrameTime
Sample Unit: Millisecond

| | |
|---|---|
| Max | 4.27 |
| Median | 3.98 |
| Min | 3.76 |

## MainMenu_to_MP_Finding()



Last results 4/11/2021 10:51:54 AM

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Sample Group View

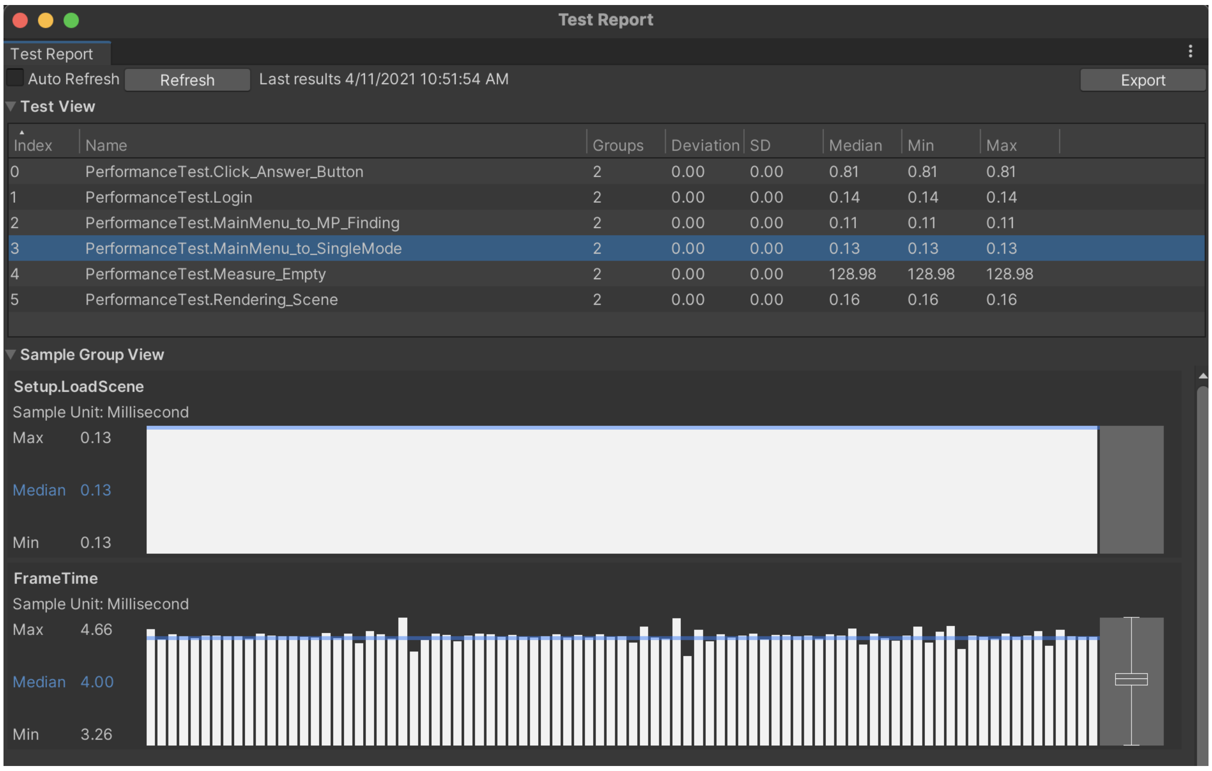
Setup.LoadScene
Sample Unit: Millisecond

| | |
|---|---|
| Max | 0.11 |
| Median | 0.11 |
| Min | 0.11 |

FrameTime
Sample Unit: Millisecond

| | |
|---|---|
| Max | 4.92 |
| Median | 3.98 |
| Min | 2.94 |

## MainMenu_to_SingleMode()



Test Report

Auto Refresh | Refresh | Last results 4/11/2021 10:51:54 AM | Export

Test View

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Sample Group View

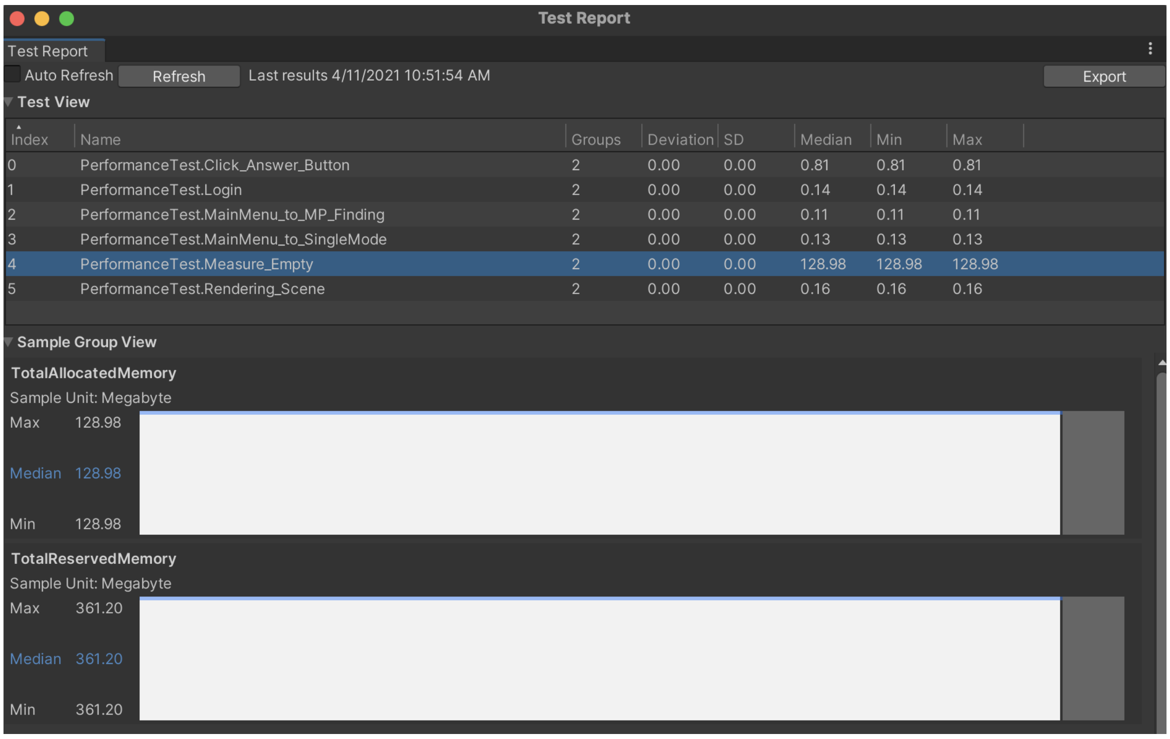
Setup.LoadScene
Sample Unit: Millisecond

| | |
|---|---|
| Max | 0.13 |
| Median | 0.13 |
| Min | 0.13 |

FrameTime
Sample Unit: Millisecond

| | |
|---|---|
| Max | 4.66 |
| Median | 4.00 |
| Min | 3.26 |

## Measure_Empty()



Test Report

Auto Refresh | Refresh | Last results 4/11/2021 10:51:54 AM | Export

Test View

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Sample Group View

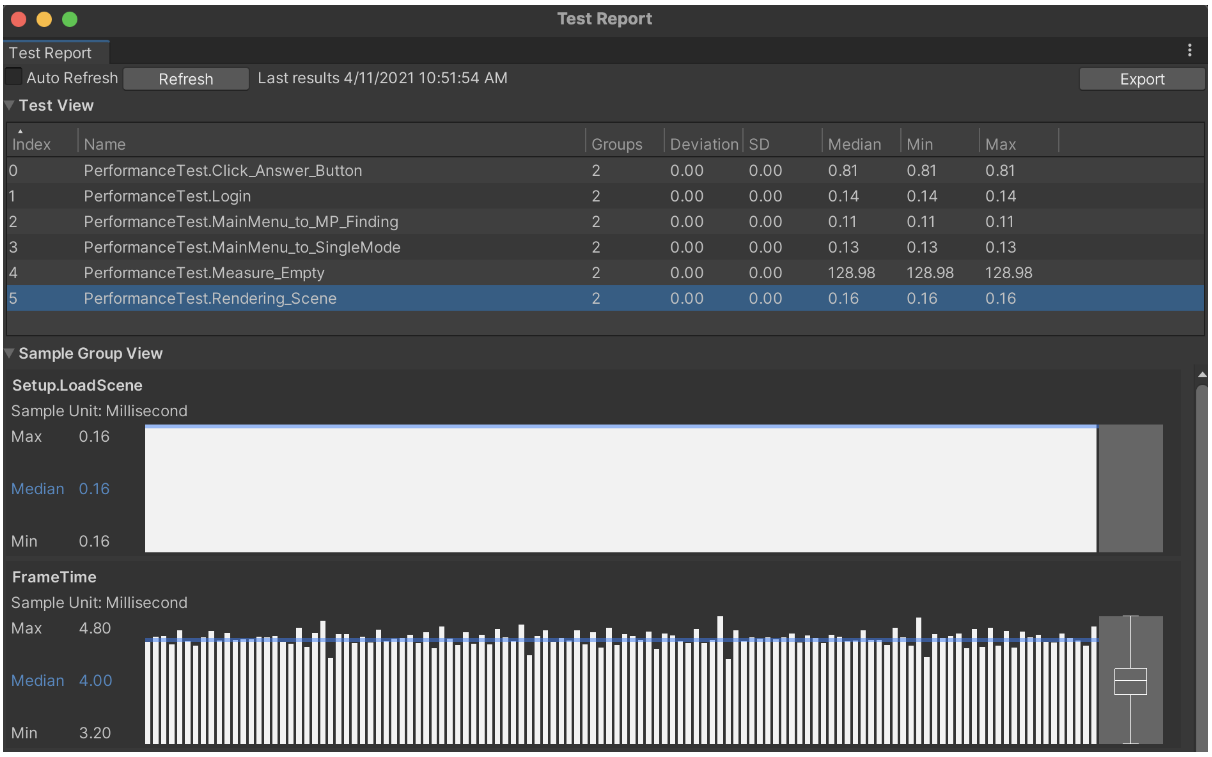
TotalAllocatedMemory
Sample Unit: Megabyte

| | |
|---|---|
| Max | 128.98 |
| Median | 128.98 |
| Min | 128.98 |

TotalReservedMemory
Sample Unit: Megabyte

| | |
|---|---|
| Max | 361.20 |
| Median | 361.20 |
| Min | 361.20 |

**Rendering_Scene()**



Test Report

Test Report

Auto Refresh | Refresh | Last results 4/11/2021 10:51:54 AM | Export

Test View

| Index | Name | Groups | Deviation | SD | Median | Min | Max |
|---|---|---|---|---|---|---|---|
| 0 | PerformanceTest.Click_Answer_Button | 2 | 0.00 | 0.00 | 0.81 | 0.81 | 0.81 |
| 1 | PerformanceTest.Login | 2 | 0.00 | 0.00 | 0.14 | 0.14 | 0.14 |
| 2 | PerformanceTest.MainMenu_to_MP_Finding | 2 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 |
| 3 | PerformanceTest.MainMenu_to_SingleMode | 2 | 0.00 | 0.00 | 0.13 | 0.13 | 0.13 |
| 4 | PerformanceTest.Measure_Empty | 2 | 0.00 | 0.00 | 128.98 | 128.98 | 128.98 |
| 5 | PerformanceTest.Rendering_Scene | 2 | 0.00 | 0.00 | 0.16 | 0.16 | 0.16 |

Sample Group View

Setup.LoadScene
Sample Unit: Millisecond
Max 0.16
Median 0.16
Min 0.16

FrameTime
Sample Unit: Millisecond
Max 4.80
Median 4.00
Min 3.20

## 4 Smoke, Load and Stress Testing

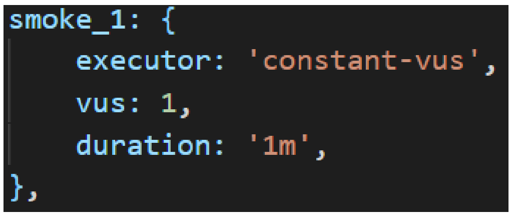
All tests in this section are performed using k6, an open-source load testing tool and cloud service provider for API performance testing. The purpose of load testing is to determine a system's behavior under minimal, normal and peak load conditions. It allows us to determine how many users our game system can actually handle by checking how the system functions under a heavy number of concurrent virtual users performing transactions over a certain period of time.

### 4.1 Smoke test

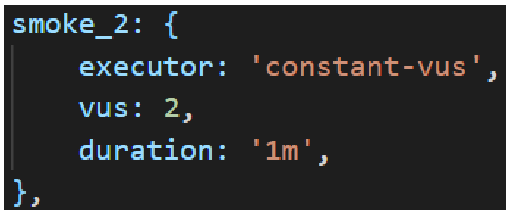
Smoke test is like a regular load test that is configured for minimal load, where it acts as a sanity check to ensure that the system is functioning correctly by being able to run smoothly with just the minimum load (i.e. a single user).

In our case, our smoke test consists of testing the system with a single virtual user, and also a separate test with 2 virtual users to take into account our multiplayer game mode which required at least 2 distinct virtual users. The figures below display the test conditions, the results, as well as graphs plotted as a visual representation of the test.

Test Conditions:

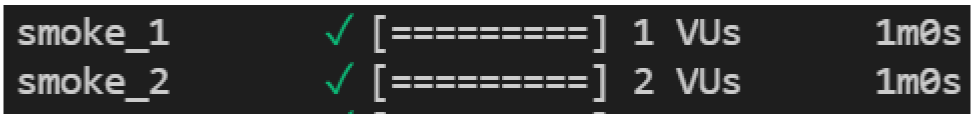


```
smoke_1: {
    executor: 'constant-vus',
    vus: 1,
    duration: '1m',
},
```



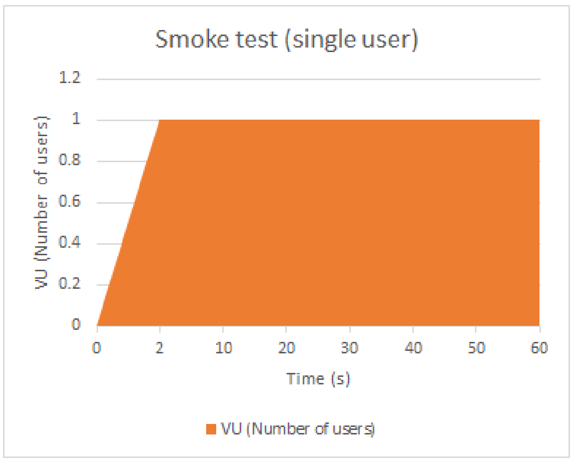
```
smoke_2: {
    executor: 'constant-vus',
    vus: 2,
    duration: '1m',
},
```

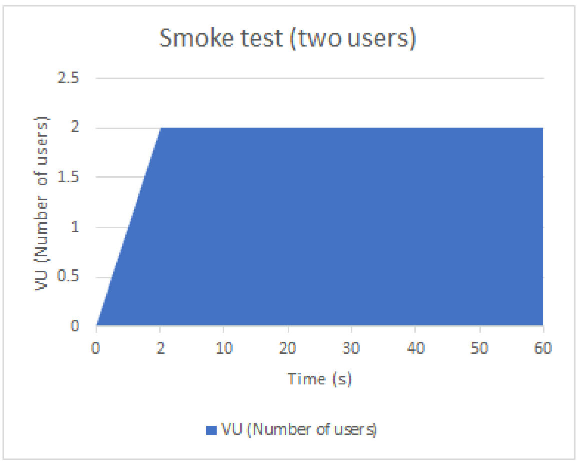
Test Results:



```
smoke_1      ✓ [==========] 1 VUs      1m0s
smoke_2      ✓ [==========] 2 VUs      1m0s
```

Plotted Graphs:



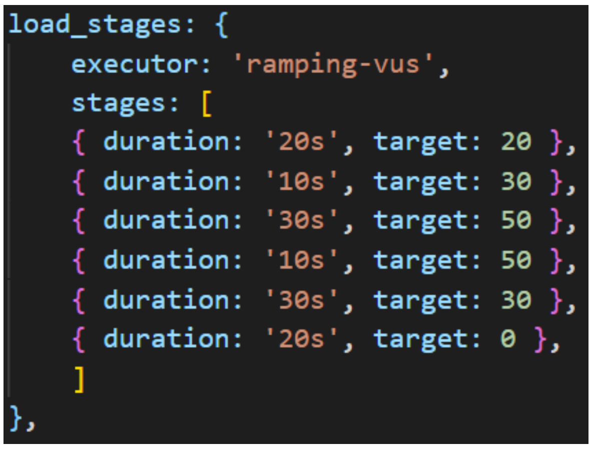


## 4.2 Load test

The primary function of a load test is to assess the current performance of a system in terms of concurrent users or requests per second. It is used to determine if a system is able to function correctly under the expected load conditions, where multiple users will access the system and send requests at the same time.

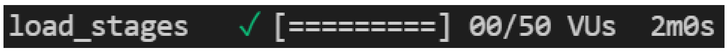
In our case, our expected number of users in the game at each moment would be around 50, and we expect that each of the 50 users would be performing around 10 requests in a minute. Hence, we have performed 2 tests, where the first test consists of stages to ramp up the total users from 0 to 50 within a minute, and then cooling the system back to 0. The second test consists of 50 users each doing 10 request iterations in a minute. The figures below display the test conditions, the results, as well as graphs plotted as a visual representation of the test.

Test 1:



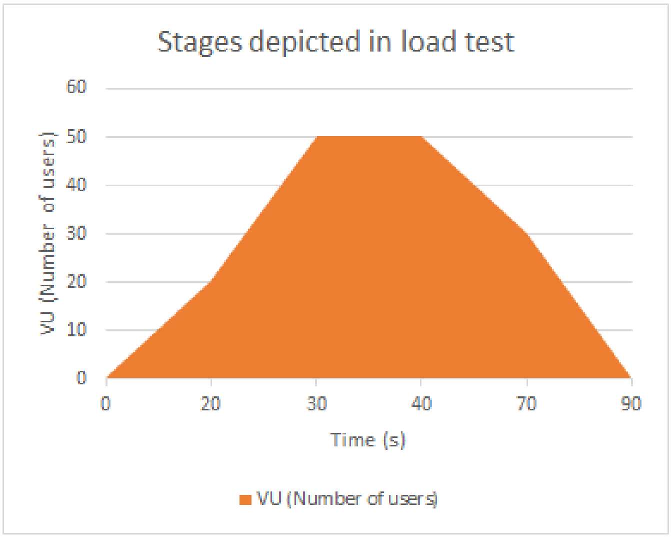
```
load_stages: {
    executor: 'ramping-vus',
    stages: [
    { duration: '20s', target: 20 },
    { duration: '10s', target: 30 },
    { duration: '30s', target: 50 },
    { duration: '10s', target: 50 },
    { duration: '30s', target: 30 },
    { duration: '20s', target: 0 },
    ]
},
```

Results:

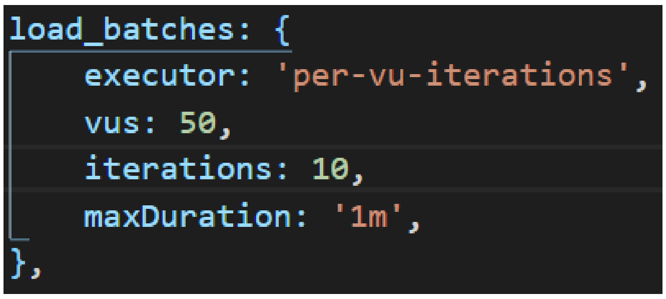


```
load_stages    ✓ [=========] 00/50 VUs   2m0s
```

Plotted Graphs:

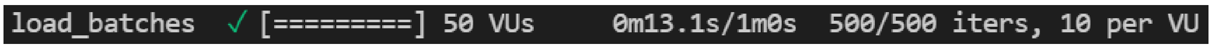


Test 2:



```
load_batches: {
    executor: 'per-vu-iterations',
    vus: 50,
    iterations: 10,
    maxDuration: '1m',
},
```

Results:



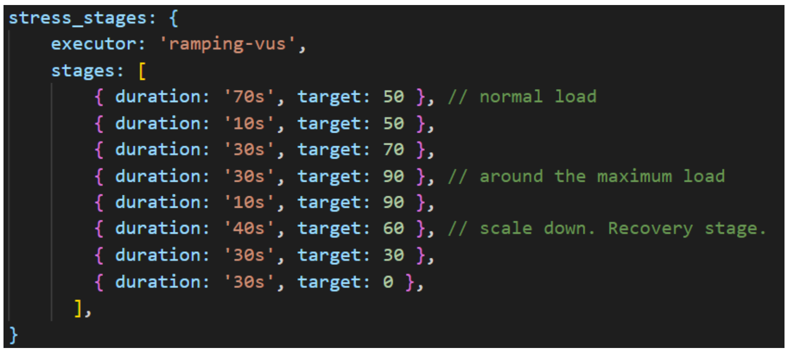
```
load_batches   ✓ [=========] 50 VUs     0m13.1s/1m0s  500/500 iters, 10 per VU
```

## 4.3 Stress test

The goal of a stress test is to assess the availability and stability of the system under heavy load. As such in order to execute the stress test, the system has to be pushed beyond its normal operation load.

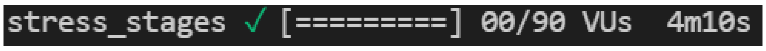
In our case, our normal expected load is to be at around 50 users concurrently, so we used 90 users as the benchmark for our stress test. The figures below display the test conditions, the results, as well as graphs plotted as a visual representation of the test.

Test Conditions:



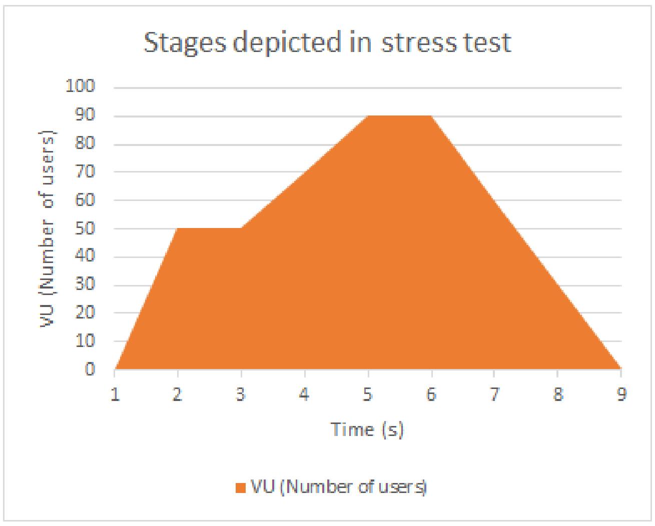
```
stress_stages: {
    executor: 'ramping-vus',
    stages: [
        { duration: '70s', target: 50 }, // normal load
        { duration: '10s', target: 50 },
        { duration: '30s', target: 70 },
        { duration: '30s', target: 90 }, // around the maximum load
        { duration: '10s', target: 90 },
        { duration: '40s', target: 60 }, // scale down. Recovery stage.
        { duration: '30s', target: 30 },
        { duration: '30s', target: 0 },
      ],
}
```

Results:



```
stress_stages ✓ [==========] 00/90 VUs  4m10s
```

Plotted Graphs:



## 5 References

1. Technologies, U. (n.d.). *Unity - Manual: Unit Testing*. Unity Documentation. Retrieved November 11, 2021, from https://docs.unity3d.com/Manual/testing-editortestsrunner.html
2. Cindrea, R. (2021, August 24). *AltUnityTester – Testing Unity games and apps using Appium*. Altom. Retrieved November 11, 2021, from https://altom.com/altunitytester-unity-using-appium/
3. *unittest — Unit testing framework — Python 3.10.0 documentation*. (n.d.). Python Organisation. Retrieved November 11, 2021, from https://docs.python.org/3/library/unittest.html
4. Sawant, A. A., Bari, P. H., & Chawan, P. M. (2012, June 1). *Software Testing Techniques and Strategies*. ResearchGate. Retrieved November 11, 2021, from https://www.researchgate.net/publication/316510706_Software_Testing_Techniques_and_Strategies
5. *Introduction to Testing with Mocha and Chai*. (n.d.). Codecademy. Retrieved November 11, 2021, from https://www.codecademy.com/articles/bapi-testing-intro
6. Tsang, T. (2019, July 5). *Unity Performance Test - Terence Tsang*. Medium. Retrieved November 11, 2021, from https://medium.com/@terence410/unity-performance-test-e80ee3cd64a7
7. Performance testing vs. Load Testing vs. stress testing. BlazeMeter. (n.d.). Retrieved November 13, 2021, from https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing.