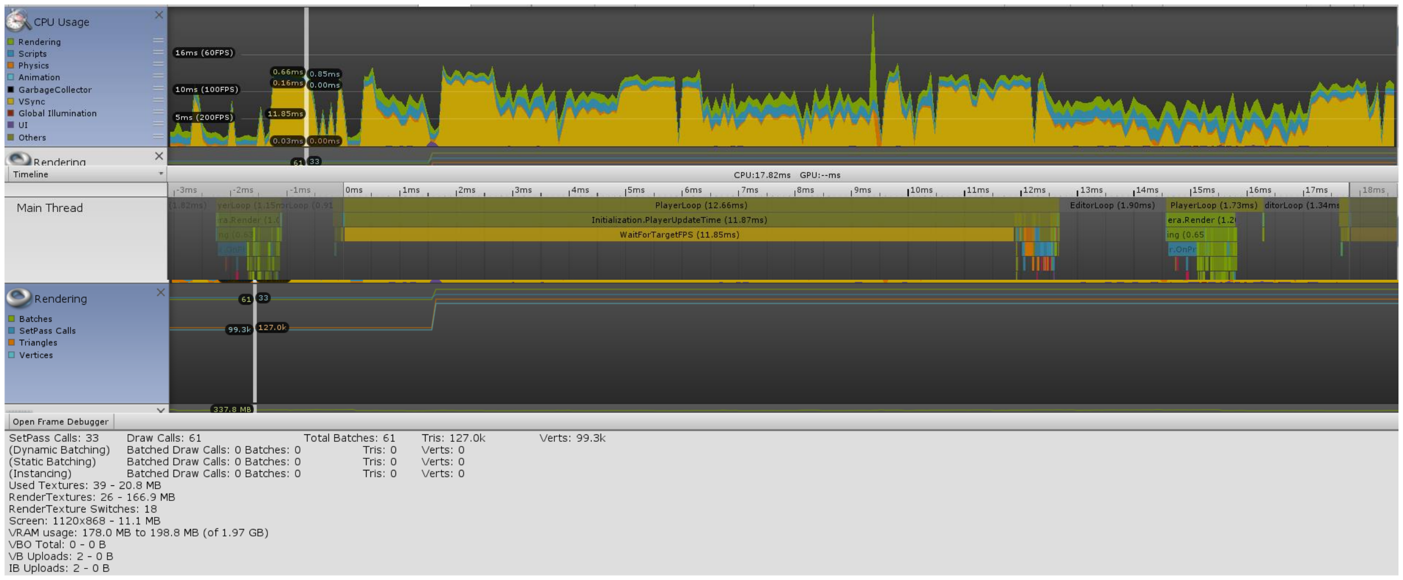
In Game Tutorial

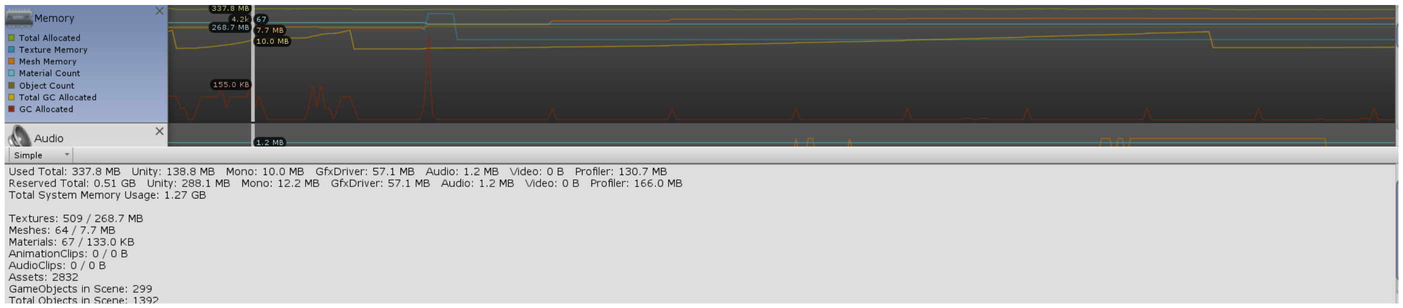
Gil Robern – 100651824 **Jessica Le – 100555079** **Zach Allen – 100650188**

Table of Contents

Unity Profiling………… …………………………………………….. 3
Jessica Le
Observer Pattern ……………………..……………………………….. 4
Jessica Le
Object Pooling…………………………………………………………. 4
Gil Robern
Quest Management System……………..……………………………...4
Jessica Le, Zach Allen
Metrics Logger ……..……………………………………………………4
Jessica Le, Gil Robern
References ……………………………………………………………4-5
Gil Robern, Jessica Le



```
SetPass Calls: 33      Draw Calls: 61                    Total Batches: 61    Tris: 127.0k     Verts: 99.3k
(Dynamic Batching)     Batched Draw Calls: 0 Batches: 0            Tris: 0    Verts: 0
(Static Batching)      Batched Draw Calls: 0 Batches: 0            Tris: 0    Verts: 0
(Instancing)           Batched Draw Calls: 0 Batches: 0            Tris: 0    Verts: 0
Used Textures: 39 - 20.8 MB
RenderTextures: 26 - 166.9 MB
RenderTexture Switches: 18
Screen: 1120x868 - 11.1 MB
VRAM usage: 178.0 MB to 198.8 MB (of 1.97 GB)
VBO Total: 0 - 0 B
VB Uploads: 2 - 0 B
IB Uploads: 2 - 0 B
```



```
Used Total: 337.8 MB   Unity: 138.8 MB   Mono: 10.0 MB   GfxDriver: 57.1 MB   Audio: 1.2 MB   Video: 0 B   Profiler: 130.7 MB
Reserved Total: 0.51 GB   Unity: 288.1 MB   Mono: 12.2 MB   GfxDriver: 57.1 MB   Audio: 1.2 MB   Video: 0 B   Profiler: 166.0 MB
Total System Memory Usage: 1.27 GB

Textures: 509 / 268.7 MB
Meshes: 64 / 7.7 MB
Materials: 67 / 133.0 KB
AnimationClips: 0 / 0 B
AudioClips: 0 / 0 B
Assets: 2832
GameObjects in Scene: 299
Total Objects in Scene: 1392
```

In Game Tutorial

Jessica Le
Game Development and Entrepreneurship
Ontario Tech University
Oshawa, Ontario
jessica.le@ontariotechu.net

Gil Robern
Game Development and Entrepreneurship
Ontario Tech University
Oshawa, Ontario
gil.robern@ontariotechu.net

***Abstract*—As students of the Ontario Tech University Game Development and Entrepreneurship program, to add to our toolbox of game development, we were assigned to create a tutorial for our game in development "Dodgket Brawlers".**

I. Obsever Pattern

To implement observer pattern in our tutorial we incorporated it in our quest system. Our subject, `GiveQuest`, maintains the different "quests" or achievements the player can achieve. If the THROW quest is active and the player throws a ball then `GiveQuest` is notified that they have completed the quest by calling the function `LoadQuest`. Where it will load the text display of the next text and mark the previous quest to completed.

II. Object Pooling

First, we create a Queue of dodgeball GameObjects by instantiating each one and deactivating it and Enqueuing it. There is a `SpawnBall` function that is used to get the first item in the queue by dequeuing it and setting it active and setting it to a temporary GameObject. This GameObject is returned at the end of the function call so it can be set to a GameObject in another script. There is also the `DespawnBall` function which takes in the most recent dodgeball and sets it to inactive and adds it back to the queue.

There is an issue with this object pool where it stops spawning more balls after a certain point. To test that it is actually working with all the functionality you will have to go into the unity hierarchy and activate another dodgeball clone that is in there and you will see after a short bit it will deactivate again, proving that the code works.

III. Quest Management System

Each "quest" in the tutorial is for the player to learn the mechanics of the game. In a class called `Quest` there is a variable `isActive`. In a class called `GiveQuest` there is a function called `LoadQuest` that will be called when the player completes a quest so the next quest can be loaded. Each quest has a number associated with it, 1 2 3 4 for "MOVE", "THROW", "CLIMB" and "FINISH" respectively. For example, once a player completes the "THROW" quest that involves them picking up a dodgeball and throwing it, we call `LoadQuest` with the int parameter (3), to load the next quest to complete, and mark the previous quest as completed.

IV. Metrics Logger

To create a metrics logger with the use of a DLL C++ plugin, we decided to save the time it takes for the player to complete the tutorial. This could be later to used to test the effectiveness of our tutorial. If players found it more difficult to learn the mechanics of the game, then they would have a significantly higher time to complete the tutorial.

To do this we store the time using unity `Time.time` in a variable `timePassed`. Once the player completes the tutorial by passing the last trigger at the end. We check to see if they have passed the trigger before, so the time won't be recorded multiple times in one tutorial run. If they have not passed the finish trigger before then we call the function `SaveTime` that is from our C++ DLL. We then mark with a bool that they have finished the tutorial.

We then have a function `LoadTime` from our DLL. When the L key is pressed it will load the time it took the player to complete the level in the debug log.

In the DLL there is a WriteFile function that is used to write the time value from the game and store it in a text file. There is a `ReadFile` function used to read the text file and return the time value stored in it. We use this function and output it to the debug during testing to see if it is working properly. We have `SaveTime` and `LoadTime` functions to save values in C++ for different uses if needed as well as to load them if needing to recall previous saved values. We also have simple getter and setter functions called `SetTime` and `GetTime`.

References

[1] Brackeys. "Questing System in Unity" Online video clip. *Youtube*.Youtube, 11 November. 2019

[2] Jason Weimann. "Observer Pattern – Game Programming Patterns" Online video clip. *Youtube*.Youtube, 11 November. 2019

[3] Rob and Tom. Tutorial (1) Starter code
[4] Rob and Tom Individual Assignment 2 Starter Code