COMP280 Worksheet 3 - Optimization

1904819  
December 4, 2021

# Introduction

For this assignment I will be optimizing the Worksheet 2 of COMP280 (Pacman AI) Found here:  
[*https://github.com/atdeJimmyG/COMP280WS3Project*](https://github.com/atdeJimmyG/COMP280WS3Project)I will be using this project as I have not done any optimization to it prior to this assignment so I believe that there will be some things I will be able to change and make better.

# Benchmark

First, I needed to record a benchmark for the project. This will allow me to see the performance before any optimisations are made and allow me to see what is impacting it the most. For this run, I played the game and took one of the enemies and then stopped the benchmark. This is a good way of gauging the performance as it will be almost the same when capturing each of the other enemies. As the enemies are roaming around, they are collecting pills at random.

As we can see below in figure 1, there are many peaks as we go through the project, with a sudden dip in the middle. While playing, this dip occurred when capturing an enemy; this reflects the profiler as removing an enemy from the game would free up resources. We can also look at the percentages at the bottom of Figure 1, these show the percentage that the game is using up. We can ignore the EditorLoop percentage as this only really refers to resources used in editor, not in the final executable. (I was not able to run the profiler on the executable only for some reason)



Figure A screenshot of the Unity profiler during the benchmark.

# Detect

From the benchmark we are able to find out what parts of the script is taking up the most computational resources. By looking at a random point in the game, we can have a look at Figure 2 and Figure 3, which is showing that Debug.Log Functions are using up 14.3% and 9.8% of the resources on that current frame. The Debug.Log functions are called in the PlayerLoop, more specifically in the Update method for the BtController script. The Debug.Log function occurs when the enemies are both targeting a pill and when they collect a pill. They will call each frame to say which pill they are targeting and if it has collected one. The spikes seen in both Figures 2, and 3 result when a pill has been consumed by the enemy, this is due to a new Debug.Log being called every time this happens.

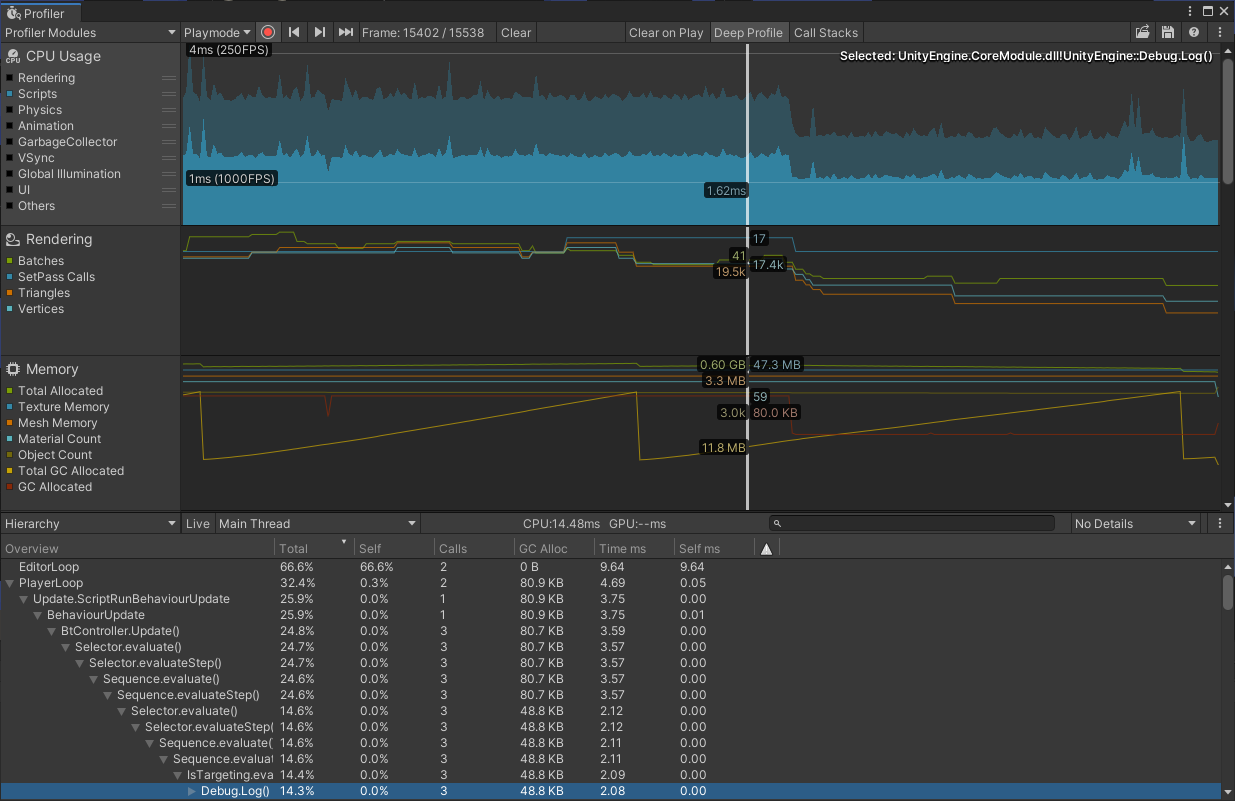


Figure A screenshot showing the Unity profiler highlighting the Debug.Log function.

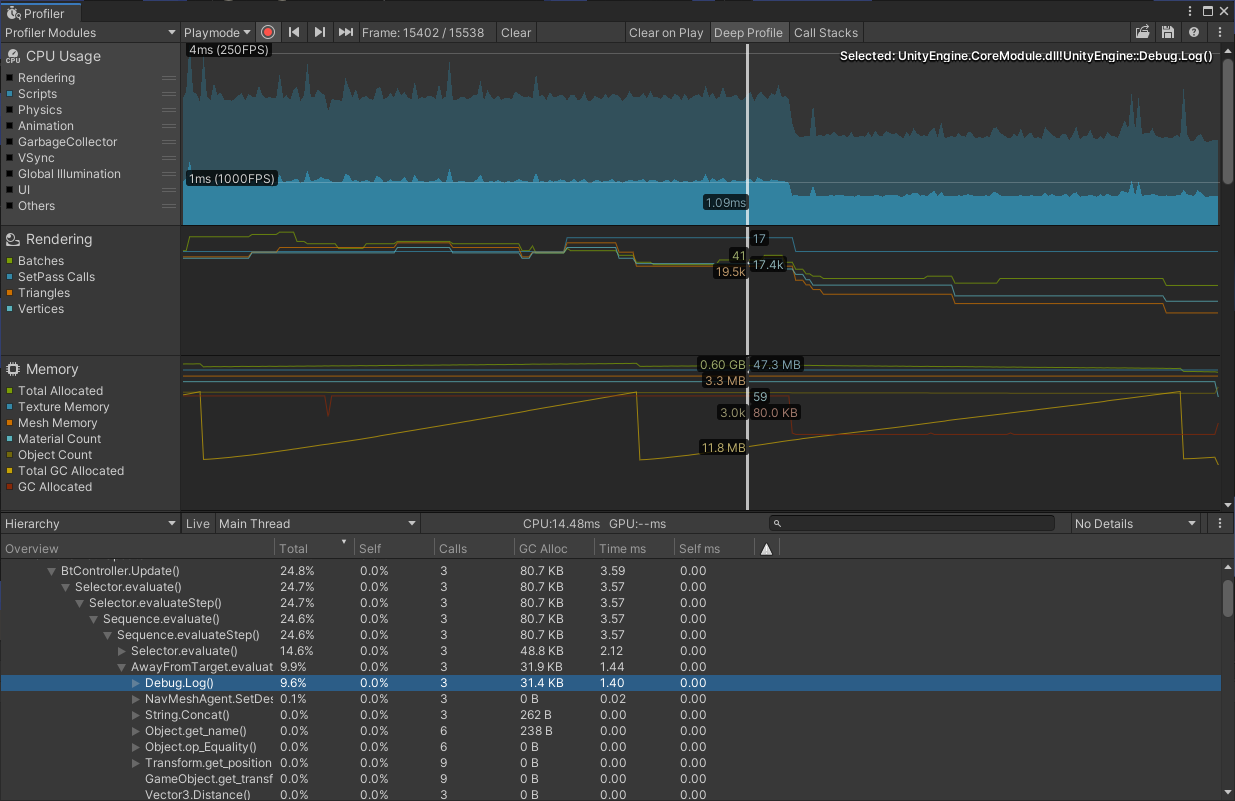


Figure A screenshot showing the Unity profiler highlighting the Debug.Log function.

# Solve

We first need to go to the scripts in question, these are the AwayFromTarget and IsTargeting scripts. From both scripts we need to remove the lines with Debug.Log as this is a simple thing that only relates to the editor it doesn’t matter too much if it isn’t removed, however it will improve performance within the editor which is nice.

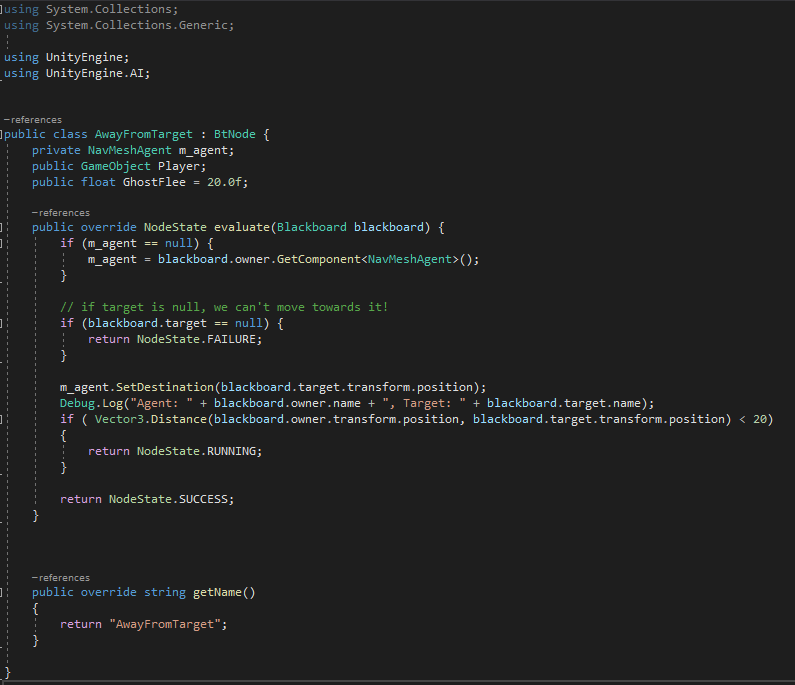
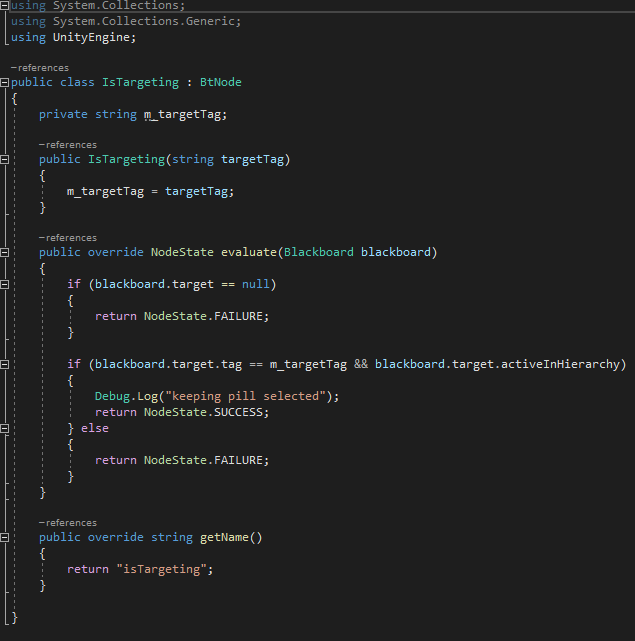
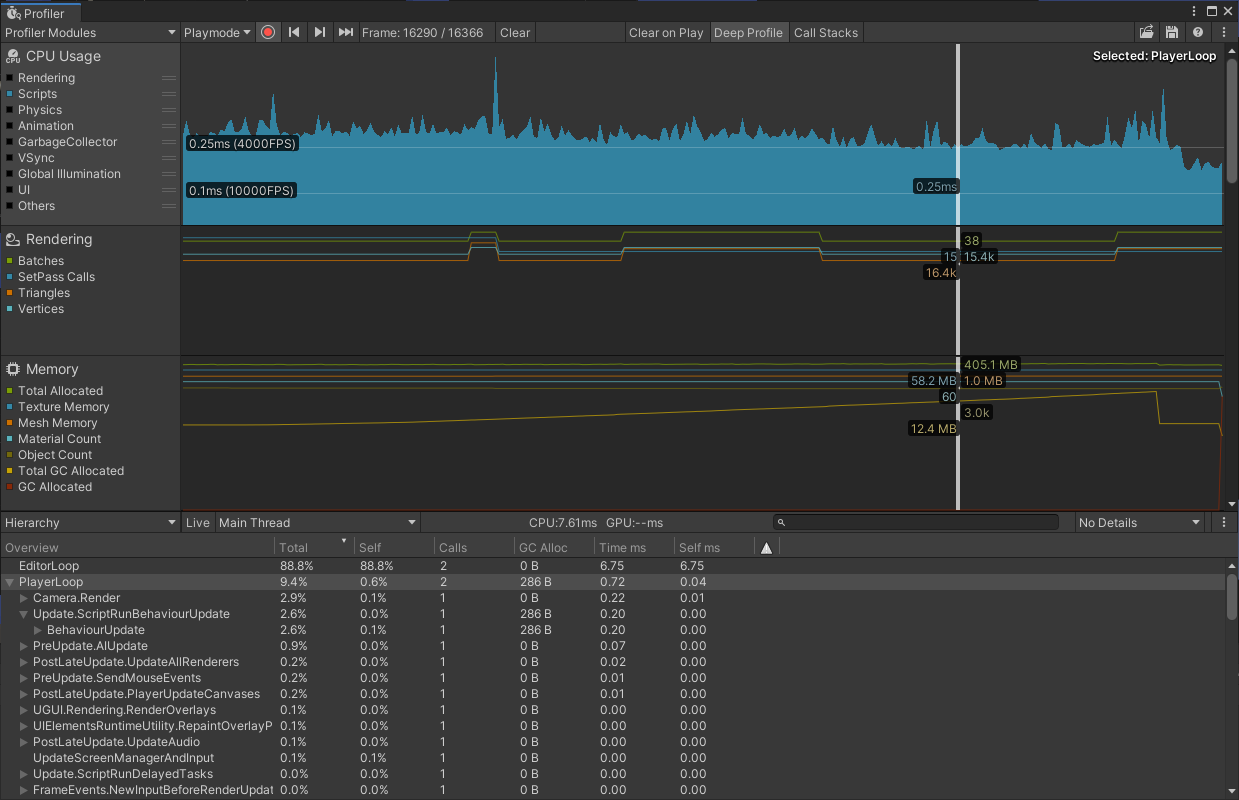


Figure A screenshot showing the code where the resources being consumed is highest



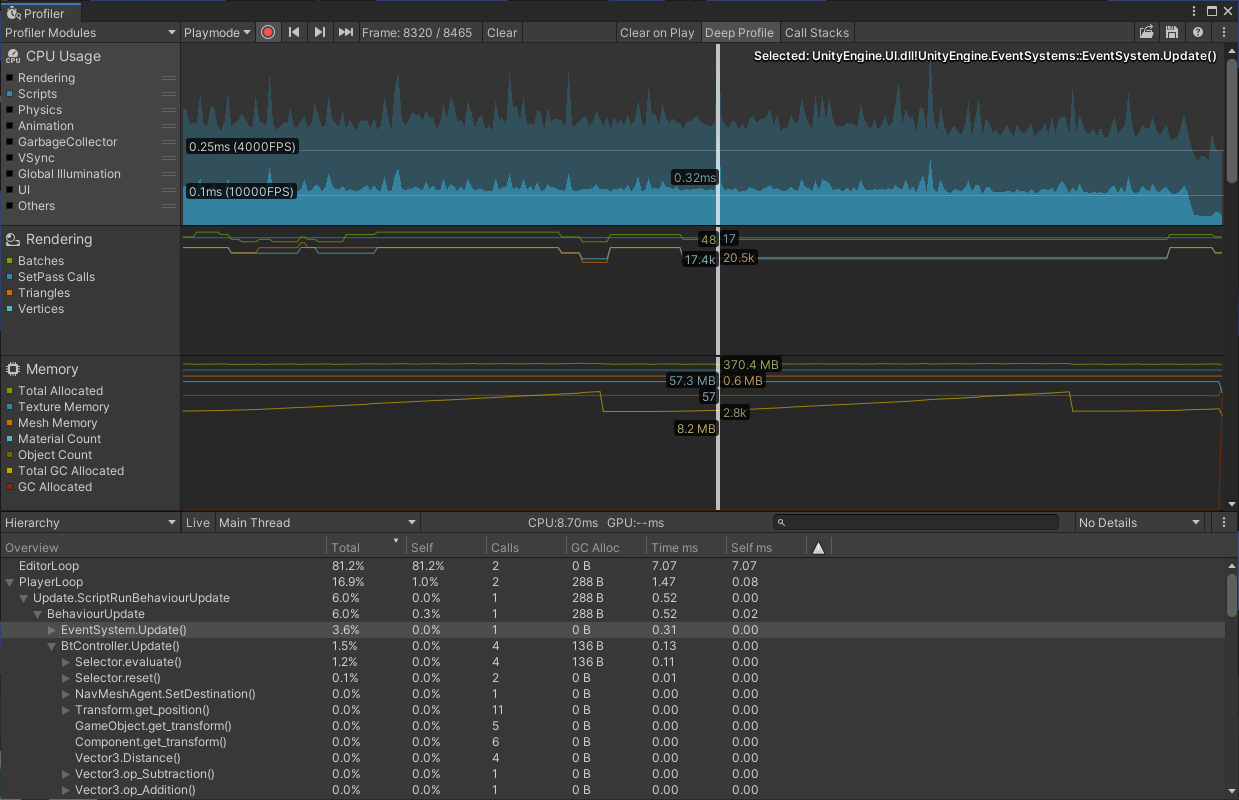
# Check

As we can see form the new profiler below, there are now less spikes overall within the profiler. However now the debug log is now gone, we can now see a bit easier about which new parts will need optimising and we can dig a bit deeper to see what is impacting the performance the most.



# Second Iteration

When I ran my next iteration, it came up with the results below. As we can see that during one of the biggest spikes, the EventSystem was a large cause of loss of performance, with 3.6% of the total performance being used on it. As the EventSystem is used for UI, and this project doesn’t contain any UI, it can simply be deleted.



After deleting the EventSystem from the hierarchy we can run the profiler again to see if there is any change on usage and performance. We can see that overall, there are less sharp peaks on average throughout the project and that the total usage has gone down since the last run.

