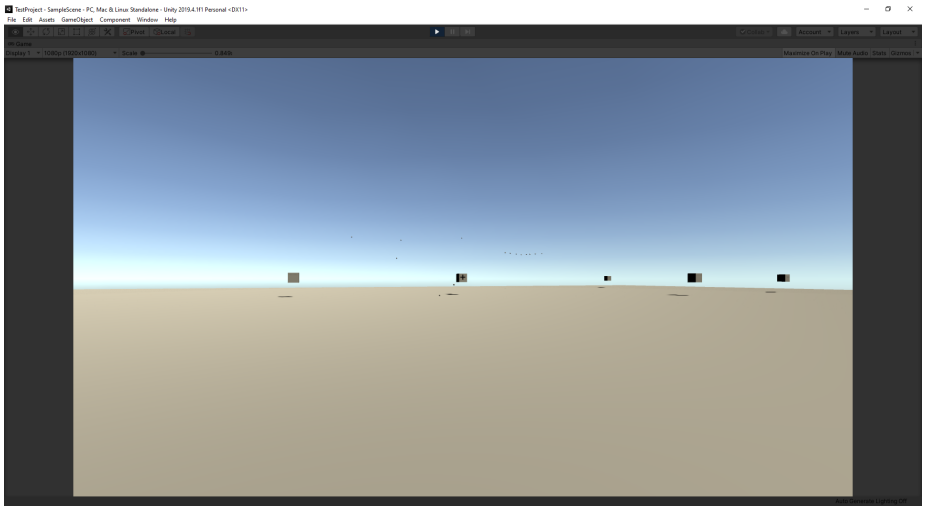
Comp280 Worksheet 3

Matt Shaw - ms228668

January 5, 2021

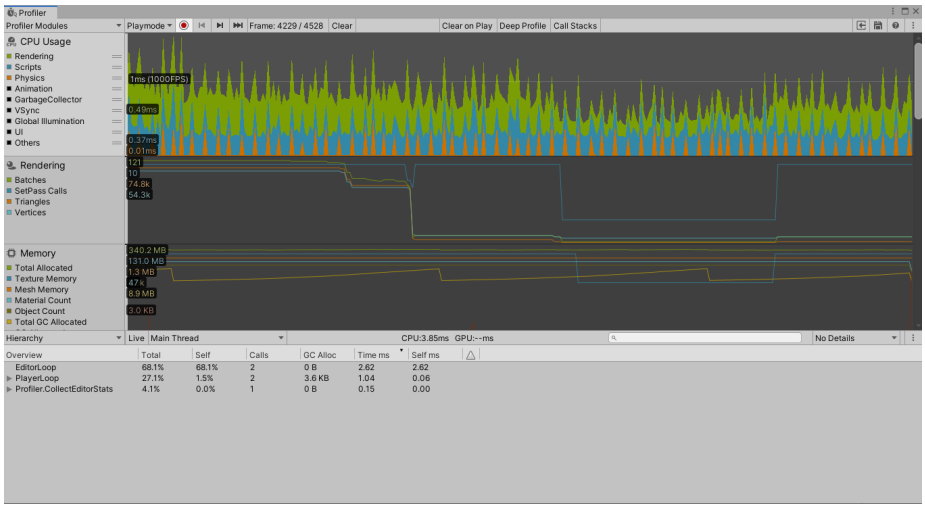
1 Introduction

For this assignment, I'm going to be using a project I worked on, which can be found here: `https://github.com/ThatVoidUpdate/TestProject`.
It is a simple effect, where you can fire out a load of bullets, then right click on an object to summon all the bullets to that object. I believe that it is pretty unoptimised, and while the frames per second is still above 60, it was written in a relatively unoptimised way, and as such is a good candidate for this task.



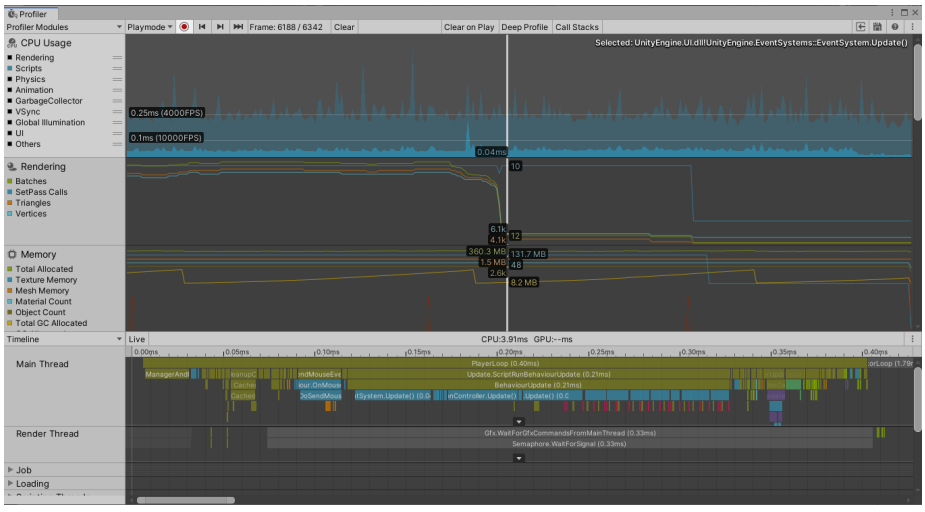
An example of the effect in action, though it may be hard to see

Much of the information relating to inefficient code practices that is used in this comes from `https://learn.unity.com/tutorial/fixing-performance-problems-2019-3`.
The first thing to do is to get a first look at the performance of the project using the profiler.



The first profiler view

In this view, I have turned off a load of the cpu views, leaving just the main 3 contributors, those being rendering, scripts and physics. The rendering stage deals with displaying the game on screen. Slow performance there could be caused by badly optimised culling, drawing objects when they dont need to be drawn, and potentially repeated changes in materials. Scripts is all of the scripts ive added to objects. Reduced performance there is usually down to poor programming descisions, like calling expensive functions every frame. Finally, physics is all of the rigidbody and collider interactions. If there are many physics objects in the scene, this would contribute to decreased speed.
The first thing to focus on, as it is usually a large contributor to slow performance, is scripts.



A view of an average frame

Looking at the Timeline view at the bottom, we can see that the main thing happening each frame is the Update method calls. If we look first at my code specifically, the first method is called on the PlayerController, and it is taking a comparatively long time (0.04ms). This seems like a good candidate to look at first.

Here is a listing of the FirstPersonController script

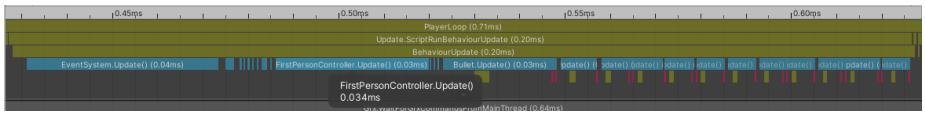
```
1     public class FirstPersonController : MonoBehaviour
2     {
3         [Header("Speeds")]
4         public float ForwardSpeed;
5         public float SidewaysSpeed;
6
7         [Header("Mouse Sensitivity")]
8         public float MouseSensitivityX;
9         public float MouseSensitivityY;
10
11        [Space]
12        public bool AirControl = true;
13        public float JumpForce;
14
15        private Vector3 MovementVector;
16        private float HeadAngle = 0;
17        // Start is called before the first frame update
18        void Start()
19        {
20            Cursor.lockState = CursorLockMode.Locked;
21            Cursor.visible = false;
22        }
23
24        // Update is called once per frame
25        void Update()
26        {
27            CharacterController controller = GetComponent<CharacterController>();
28            MovementVector.x = Input.GetAxis("Horizontal") * SidewaysSpeed;
29            MovementVector.z = Input.GetAxis("Vertical") * ForwardSpeed;
30            if (Input.GetButtonDown("Jump"))
31            {
32                MovementVector.y = JumpForce;
33            }
34
35            MovementVector.y -= -Physics.gravity.y * Time.deltaTime;
36
37            if (controller.isGrounded || (!controller.isGrounded && AirControl))
38            {
39                controller.Move(transform.TransformDirection(MovementVector * Time.deltaTime));
40            }
41
42
43
44            transform.Rotate(new Vector3(0, Input.GetAxis("Mouse X") * MouseSensitivityX, 0));
45            HeadAngle = Mathf.Clamp(HeadAngle + (Input.GetAxis("Mouse Y") * MouseSensitivityY), -90, 90);
46
47            Camera.main.transform.localEulerAngles = new Vector3(HeadAngle, 0, 0);
48
49            if (Input.GetAxis("Cancel") == 1)
50            {
51                Cursor.lockState = CursorLockMode.None;
52                Cursor.visible = true;
53            }
54
55        }
56    }
```

The main things that are important are lines 27 and 47. Line 27 runs an expensive GetComponent<>() method. These should ideally only be run in the Start or Awake methods, as they can take a very logn time to run. The other problem is in line 47, with a call to Camera.main. This checks every GameObject in the scene to try and find the camera tagged with MainCamera. This should also be moved to the start method if possible.
Making these changes results in this code:

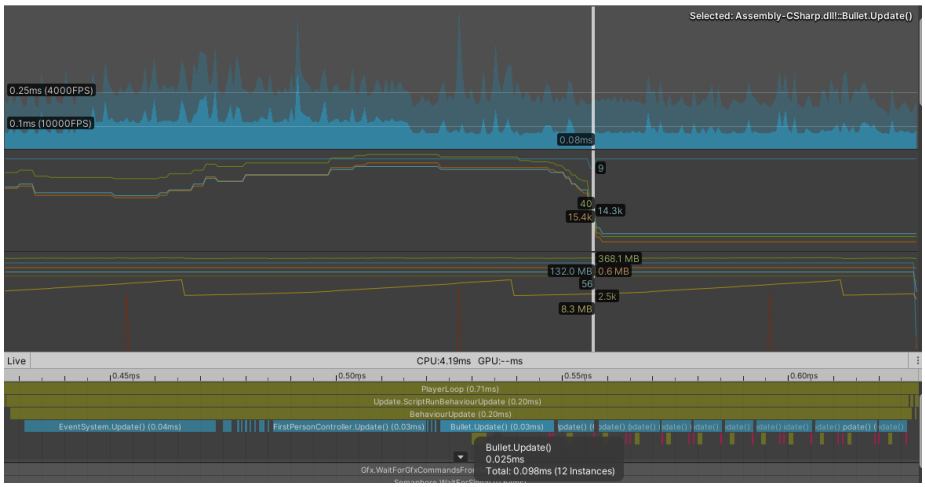
```
1  public class FirstPersonController : MonoBehaviour
2  {
3      [Header("Speeds")]
4      public float ForwardSpeed;
5      public float SidewaysSpeed;
6
```

```
7       [Header("Mouse Sensitivity")]
8       public float MouseSensitivityX;
9       public float MouseSensitivityY;
10
11      [Space]
12      public bool AirControl = true;
13      public float JumpForce;
14
15      private Vector3 MovementVector;
16      private float HeadAngle = 0;
17
18      private Camera camera;
19      private CharacterController controller;
20      // Start is called before the first frame update
21      void Start()
22      {
23          Cursor.lockState = CursorLockMode.Locked;
24          Cursor.visible = false;
25
26          camera = Camera.main;
27          controller = GetComponent<CharacterController>();
28      }
29
30      // Update is called once per frame
31      void Update()
32      {
33
34          MovementVector.x = Input.GetAxis("Horizontal") * SidewaysSpeed;
35          MovementVector.z = Input.GetAxis("Vertical") * ForwardSpeed;
36          if (Input.GetButtonDown("Jump"))
37          {
38              MovementVector.y = JumpForce;
39          }
40
41          MovementVector.y -= -Physics.gravity.y * Time.deltaTime;
42
43          if (controller.isGrounded || (!controller.isGrounded && AirControl))
44          {
45              controller.Move(transform.TransformDirection(MovementVector * Time.deltaTime));
46          }
47
48          transform.Rotate(new Vector3(0, Input.GetAxis("Mouse X") * MouseSensitivityX, 0));
49          HeadAngle = Mathf.Clamp(HeadAngle + (Input.GetAxis("Mouse Y") * MouseSensitivityY), -90, 90);
50
51          camera.transform.localEulerAngles = new Vector3(HeadAngle, 0, 0);
52
53          if (Input.GetAxis("Cancel") == 1)
54          {
55              Cursor.lockState = CursorLockMode.None;
56              Cursor.visible = true;
57          }
58
59      }
60  }
```

Making these changes reduces the time takes to 0.034ms, 85% of the original speed. While this is not a huge saving, if there were more components on the character, the GetComponent<>() call would take much longer, and if there are more objects in the scene, the Camera.main call would take longer. It is always best to move these expensive calls to the Start method where possible.



The next thing to look at is the bullet script.



Looking at the histogram at the top with the bullet update method selected, we can see that it makes up a large proportion of the frame budget. We can also see that it makes a call to a lower function, and provokes a memory allocation. This is the code for the bullets:

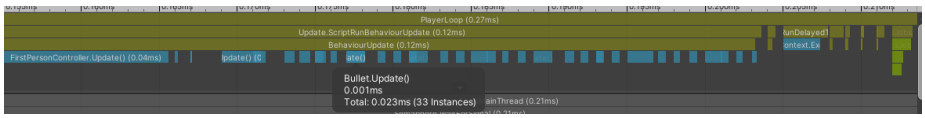
```
1  public class Bullet : MonoBehaviour
2  {
3      public float speed;
4
5      Vector3 Direction;
6
7      // Start is called before the first frame update
8      void Start()
9      {
10         Direction = transform.forward;
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16         GetComponent<Rigidbody>().velocity = Direction * speed;
17
18         if (FindObjectsOfType<Gun>()[0].IsAttracting)
19         {
20             Direction = (FindObjectsOfType<Gun>()[0].AttractingObject.transform.position - transform.position).normalized;
21         }
22     }
23 }
```

Here the main culprits are lines 16, 18 and 20. These are all calls to expensive methods, and 18 and 20 especially since FindObjectsOfType<>() will iterate over every component of every gameobject, which is extremely inefficient in larger projects. Making changes to the code to move those expensive calls to the Start method gives this improved code.

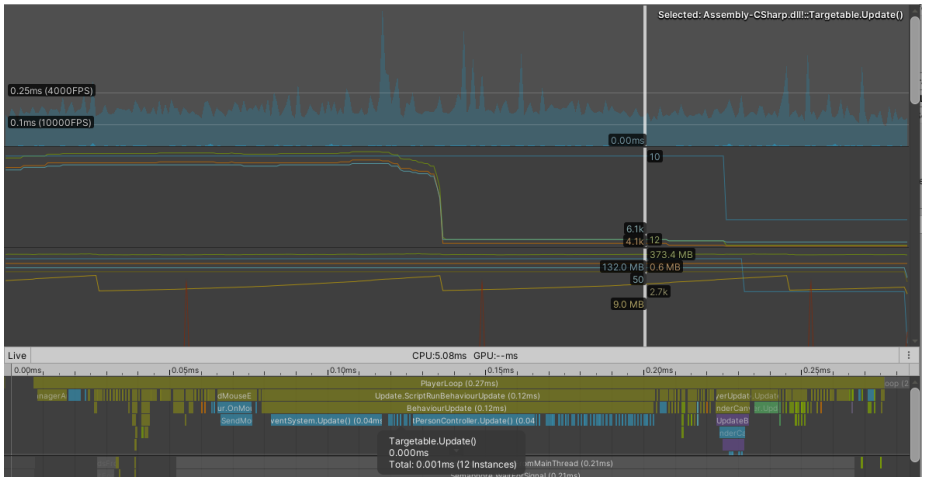
```
1      public class Bullet : MonoBehaviour
2  {
3      public float speed;
4
5      Vector3 Direction;
6
7      private Rigidbody rb;
8      private Gun gun;
9
10     // Start is called before the first frame update
11     void Start()
12     {
13         Direction = transform.forward;
14         rb = GetComponent<Rigidbody>();
15         gun = FindObjectOfType<Gun>();
```

```
16      }
17
18      // Update is called once per frame
19      void Update()
20      {
21          rb.velocity = Direction * speed;
22
23          if (gun.IsAttracting)
24          {
25              Direction = (gun.AttractingObject.transform.position - transform.position).normalized;
26          }
27      }
28  }
```

Running the profiler again shows a roughly 25x increase in speed. This shows just how intensive the FindObjectsOfType<>() method can be.



Continuing to look at the profiler again, we can see many small slices in the update section.



Clicking on them shows that they are from the Targetable script, which is odd since they dont do anything each frame. Looking at their code quickly reveals the problem.

```
1   public class Targetable : MonoBehaviour
2   {
3       // Start is called before the first frame update
4       void Start()
5       {
6
7       }
8
9       // Update is called once per frame
10      void Update()
11      {
12
13      }
14
15      private void OnMouseEnter()
16      {
17          FindObjectsOfType<Gun>()[0].AttractingObject = this.gameObject;
```

```
18      }
19
20      private void OnMouseExit()
21      {
22          FindObjectsOfType<Gun>()[0].AttractingObject = null;
23      }
24  }
```

The problem here is not obvious at first glace, as usually the C# compiler would optimise away empty functions. However, Unity will keep them around during compilation, and they are still called, due to how the engine works internally. Empty Unity functions are still called whenever they would usually be called, which can cause a performance hit if it is done a lot. These functions can be removed for a boost, like this:

```
1  public class Targetable : MonoBehaviour
2  {
3      private void OnMouseEnter()
4      {
5          FindObjectsOfType<Gun>()[0].AttractingObject = this.gameObject;
6      }
7
8      private void OnMouseExit()
9      {
10         FindObjectsOfType<Gun>()[0].AttractingObject = null;
11     }
12 }
```

After running the profiler yet again, we can see that Targetable is no longer involved in the Update method. Again, this did not cause a particularly large improvement in this project, but it could become a problem in larger projects.