COMP280 - Optimisation

Callum Metcalfe
1900133

January 5, 2021

1 Introduction

During the Plymouth game jam I worked with 3 others to create a game we called Tumble Town. This was a simple game where you play as a tumble weed trying to avoid being shot by cacti whilst escaping their towns.

Due to this being a simple game for a game jam it meant that it would likely have many inefficiencies that I could try and improve upon.

Link to the repository: `https://github.com/TragicDragoon20/PGJ-Untouchable`

2 Performance Profiling

In order to figure out where I need to prioritise optimisation I started the game and played through the second level as it had the majority of mechanics in.

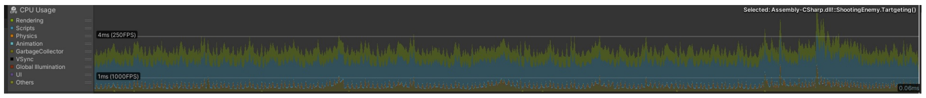


Figure 1: Profiler Base line

This profiling shows that there isn't a specific area that drastically needs to be focused for optimising. However, since my main involvement with the game was AI I will be focusing on that area mostly. The only area I would like to point out is the small spike on the profiler which was Semaphore.WaitForSignal within Gfx.WaitForPresentOnGfxThread which indicates that the CPU is waiting on the GPU before moving onto the next frame meaning the game is slightly GPU bound at times. The only other spike that I recognised were on scene loads which would obviously happen due to everything being rendered at once.

3 Performance Improvements

3.1 Object Pooling

The first area I wanted to look into was object pooling as I know that a lot of bullets are spawned throughout each level. Since Object pooling means that the game objects are instantiated on level load it should theoretically improve the performance whilst the game is running. Since no more bullets have to be spawned during run time. An example of how it works can be seen in figure 2

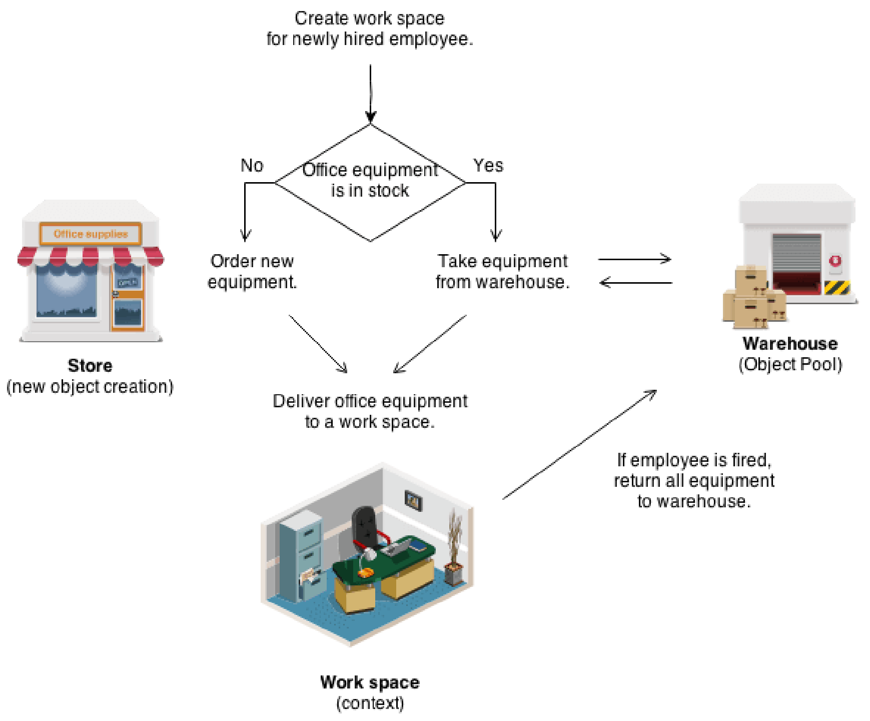
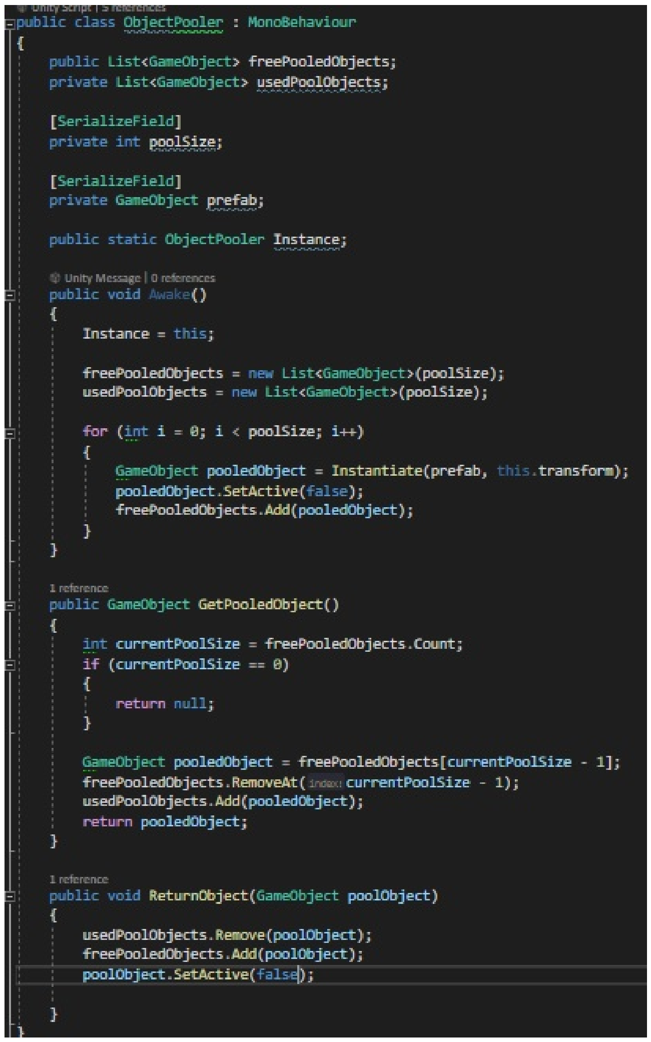


Figure 2: Object Pool Example Kowalski [2018]

You can see my implementation of object pooling in figure 3

The only issue I can think of with the current implementation is if there are more bullets fired than the amount in the lists. Whilst this seem unlikely with my game an area this can be changed is by making the list expandable so that when all the bullets are active it would start to instantiate more.

After running the profiler after making this change it seems that overall there was not much of change I suspect this to be because In the grand scheme of things I'm not spawning a lot of bullets at once but instead over the duration of



```
Unity Script | 5 references
public class ObjectPooler : MonoBehaviour
{
    public List<GameObject> freePooledObjects;
    private List<GameObject> usedPoolObjects;

    [SerializeField]
    private int poolSize;

    [SerializeField]
    private GameObject prefab;

    public static ObjectPooler Instance;

    Unity Message | 0 references
    public void Awake()
    {
        Instance = this;

        freePooledObjects = new List<GameObject>(poolSize);
        usedPoolObjects = new List<GameObject>(poolSize);

        for (int i = 0; i < poolSize; i++)
        {
            GameObject pooledObject = Instantiate(prefab, this.transform);
            pooledObject.SetActive(false);
            freePooledObjects.Add(pooledObject);
        }
    }

    1 reference
    public GameObject GetPooledObject()
    {
        int currentPoolSize = freePooledObjects.Count;
        if (currentPoolSize == 0)
        {
            return null;
        }

        GameObject pooledObject = freePooledObjects[currentPoolSize - 1];
        freePooledObjects.RemoveAt( index: currentPoolSize - 1);
        usedPoolObjects.Add(pooledObject);
        return pooledObject;
    }

    1 reference
    public void ReturnObject(GameObject poolObject)
    {
        usedPoolObjects.Remove(poolObject);
        freePooledObjects.Add(poolObject);
        poolObject.SetActive(false);

    }
}
```

Figure 3: Object Pooling

the game. An area I find it does help though is close to the end as more bullets are being fired at once compared to elsewhere in the level.I therefore suspect that if I were to either increase the fire rate or have multiple bullets per shot than this would have a larger affect at reducing the time per frame.

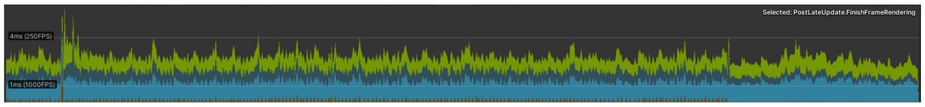
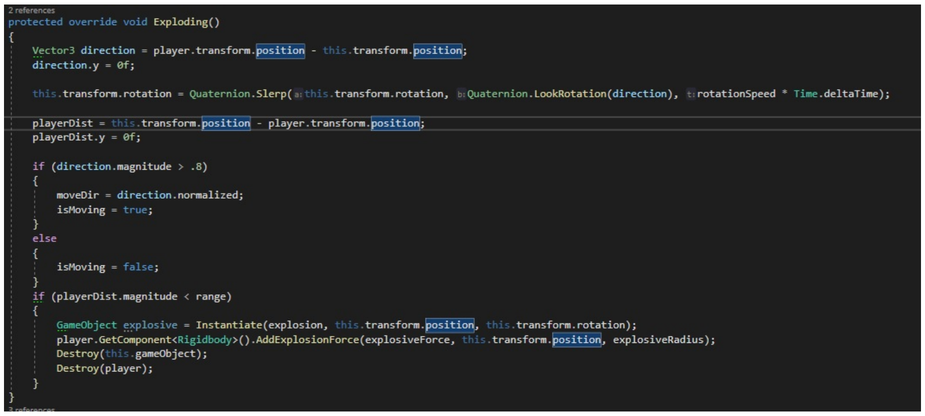


Figure 4: Object Pool Profiler

3.2 Caching

An area I realised I could make a small difference is caching some values to reduce how much unity calls those functions.

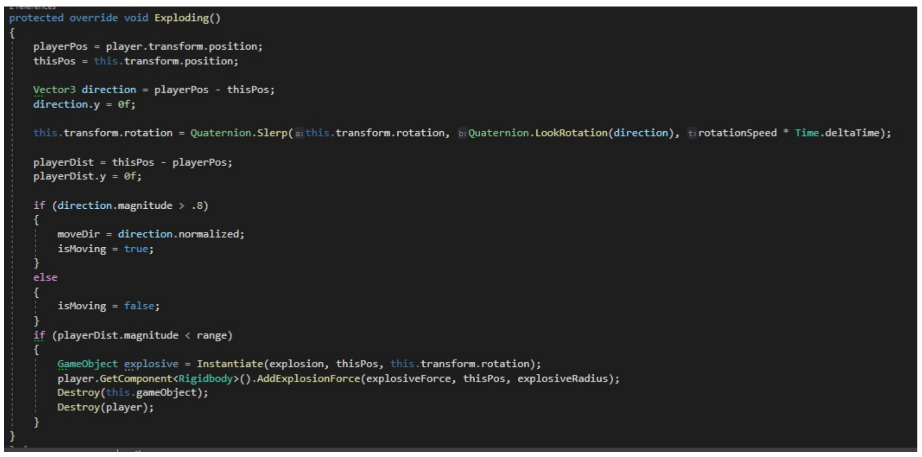
The values I'm wanting to cache are the transform positions since they are being called 2 - 4 times each update as shown in figure 5. In figure 6 you can see that I changed it so that the script only calls for the player and its location once per run rather than multiple.



```
2 references
protected override void Exploding()
{
    Vector3 direction = player.transform.position - this.transform.position;
    direction.y = 0f;

    this.transform.rotation = Quaternion.Slerp(a:this.transform.rotation, b:Quaternion.LookRotation(direction), t:rotationSpeed * Time.deltaTime);

    playerDist = this.transform.position - player.transform.position;
    playerDist.y = 0f;

    if (direction.magnitude > .8)
    {
        moveDir = direction.normalized;
        isMoving = true;
    }
    else
    {
        isMoving = false;
    }
    if (playerDist.magnitude < range)
    {
        GameObject explosive = Instantiate(explosion, this.transform.position, this.transform.rotation);
        player.GetComponent<Rigidbody>().AddExplosionForce(explosiveForce, this.transform.position, explosiveRadius);
        Destroy(this.gameObject);
        Destroy(player);
    }
}
```

Figure 5: Caching Values

After making the change it once again doesn't show a massive improvement in the time taken per frame. However, when looking at the the number of calls that get the transform and positions of the player and active game object it has halved the number of calls as seen in figures 8 and 9



```
protected override void Exploding()
{
    playerPos = player.transform.position;
    thisPos = this.transform.position;

    Vector3 direction = playerPos - thisPos;
    direction.y = 0f;

    this.transform.rotation = Quaternion.Slerp(a:this.transform.rotation, b:Quaternion.LookRotation(direction), t:rotationSpeed * Time.deltaTime);

    playerDist = thisPos - playerPos;
    playerDist.y = 0f;

    if (direction.magnitude > .8)
    {
        moveDir = direction.normalized;
        isMoving = true;
    }
    else
    {
        isMoving = false;
    }
    if (playerDist.magnitude < range)
    {
        GameObject explosive = Instantiate(explosion, thisPos, this.transform.rotation);
        player.GetComponent<Rigidbody>().AddExplosionForce(explosiveForce, thisPos, explosiveRadius);
        Destroy(this.gameObject);
        Destroy(player);
    }
}
```

Figure 6: After Caching

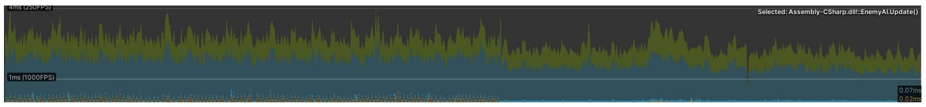
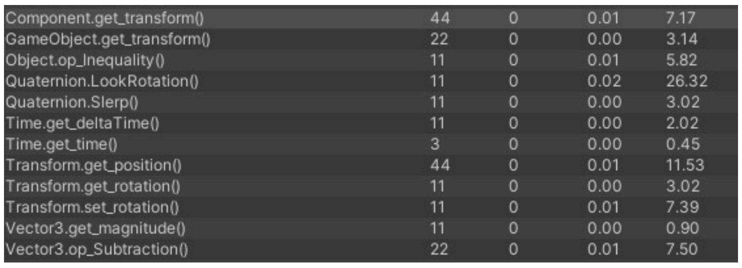
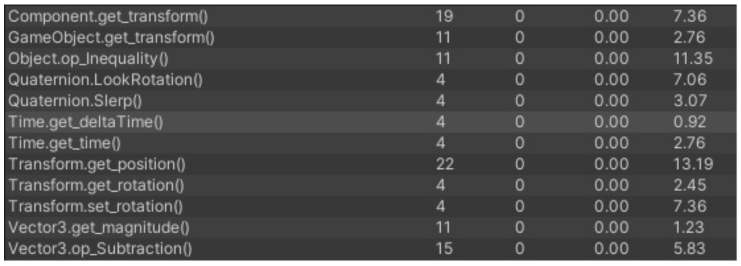


Figure 7: Caching Profile



Component.get_transform()	44	0	0.01	7.17
GameObject.get_transform()	22	0	0.00	3.14
Object.op_Inequality()	11	0	0.01	5.82
Quaternion.LookRotation()	11	0	0.02	26.32
Quaternion.Slerp()	11	0	0.00	3.02
Time.get_deltaTime()	11	0	0.00	2.02
Time.get_time()	3	0	0.00	0.45
Transform.get_position()	44	0	0.01	11.53
Transform.get_rotation()	11	0	0.00	3.02
Transform.set_rotation()	11	0	0.01	7.39
Vector3.get_magnitude()	11	0	0.00	0.90
Vector3.op_Subtraction()	22	0	0.01	7.50

Figure 8: Before Caching Changes



Component.get_transform()	19	0	0.00	7.36
GameObject.get_transform()	11	0	0.00	2.76
Object.op_Inequality()	11	0	0.00	11.35
Quaternion.LookRotation()	4	0	0.00	7.06
Quaternion.Slerp()	4	0	0.00	3.07
Time.get_deltaTime()	4	0	0.00	0.92
Time.get_time()	4	0	0.00	2.76
Transform.get_position()	22	0	0.00	13.19
Transform.get_rotation()	4	0	0.00	2.45
Transform.set_rotation()	4	0	0.00	7.36
Vector3.get_magnitude()	11	0	0.00	1.23
Vector3.op_Subtraction()	15	0	0.00	5.83

Figure 9: After Caching Changes

4 Conclusion

Based on the changes I have made I would say for the size of the project it has not improved the efficiency massively. However, if the game were to be expanded upon I believe the changes would help out in the long run.

References

Sławomir Kowalski. Design patterns: Object pool, 2018. URL `https://medium.com/@sawomirkowalski/design-patterns-object-pool-e8269fd45e10`. [accessed January 4th 2021].