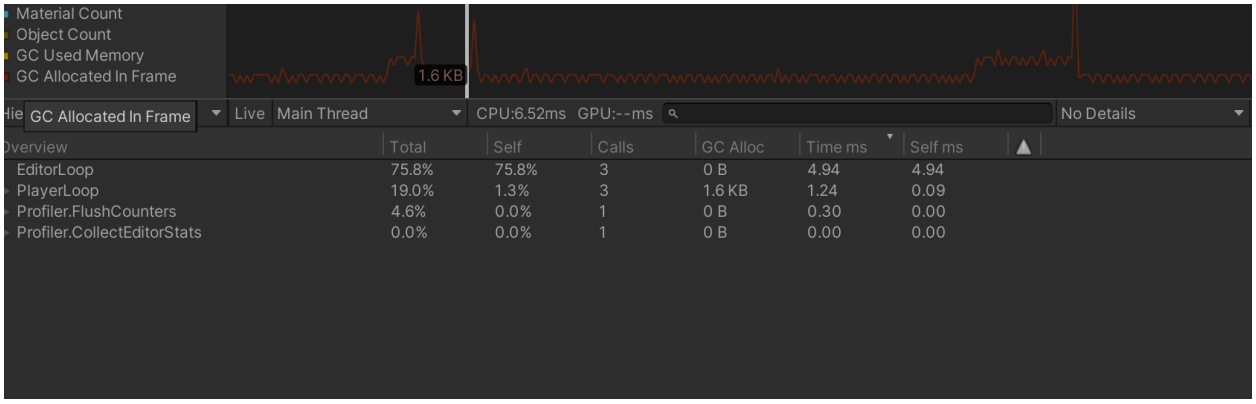
Command Pattern

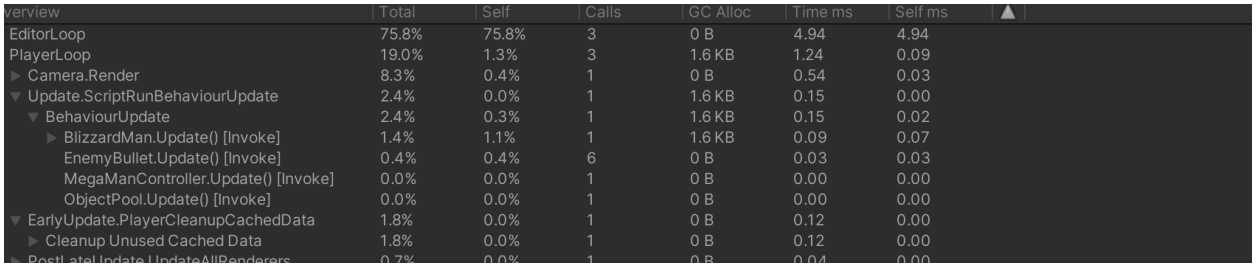
I Implemented the command pattern using a singleton script called command manager, so I can easily control the amount of commands being implemented all from one place, as well as control the command stack through one global script, then I created an ICommand script, with an execute and an Undo, then used this to create three commands, jump, move, and reverseMove, I then used these to create the movement system. I used this to try and switch between normal movement and the inverted movement, but this did not work for switching between normal and inverted, though it did work for stopping the player from jumping when hit

For the objectpool, I created an Objectpool class which holds a list of poolable objects, and I used this to hold the enemy's snowflakes, and if the player is in range, it would attack with flakes from this pool. I limited the initialization amount to just four as that is the main amount that the enemy would attack with.

Here are some of the readings from my analysis, this shows how improved the code was by using the objectpool, it could have easily been going towards 90% percent CPU usage without it, but with it it stays around 76.5% when objects are brought in because it is activating rather than creating.

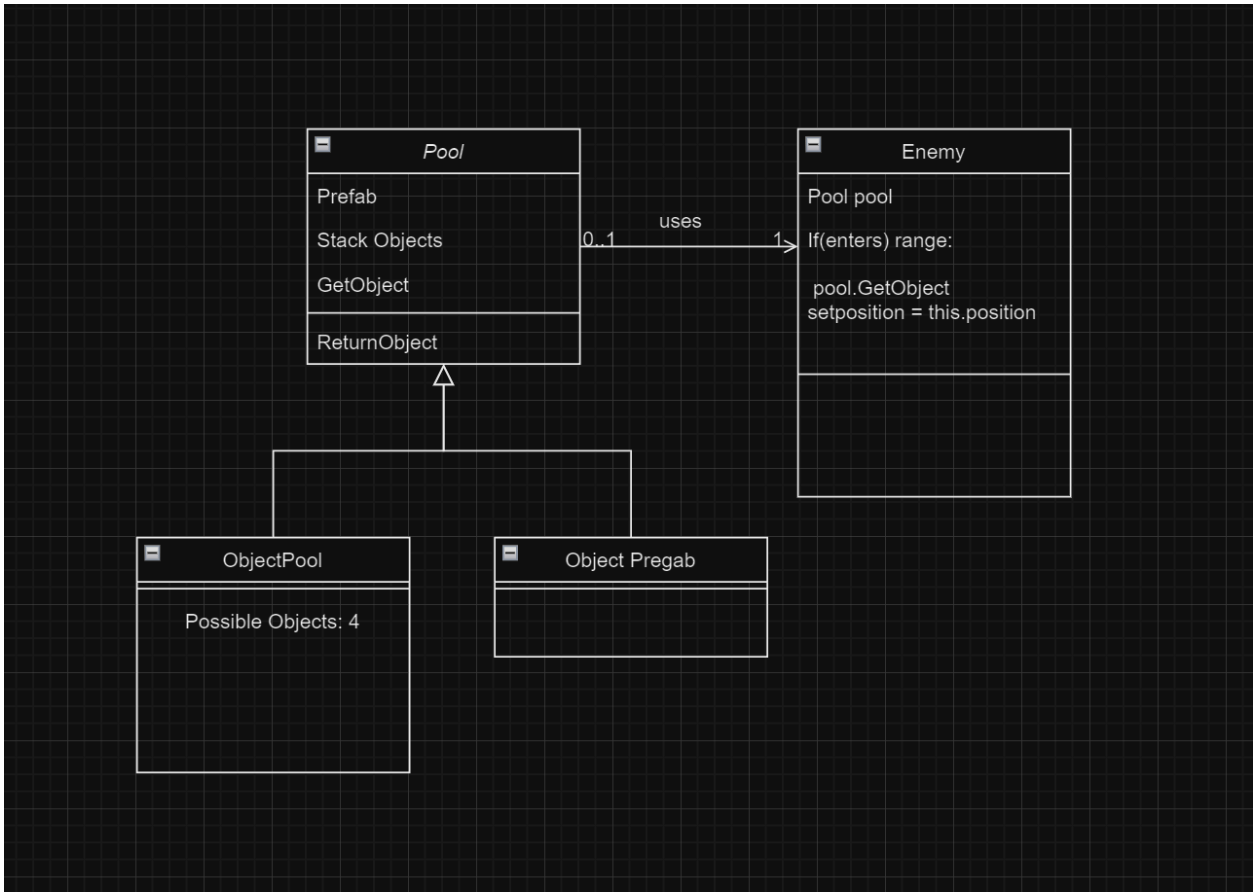
With object pooling





The spikes without object pooling

Here is my UML for the object pool



For the Additional design patterns I implemented both the observer for the switching of control, and a dll so that when the player is within range, and when the enemy is firing, time slows down, using my pauser dll. I did this to add some air of suspense.

```
using UnityEngine;

namespace Pauser
{
    Unity Script | 0 references
    public class Pauser : MonoBehaviour
    {
        [SerializeField] private float slow_down = 0.5f;
        0 references
        public void OnPause()
        {
            Time.timeScale = 0f;
        }

        0 references
        public void OnPlay()
        {
            Time.timeScale = 1f;
        }

        0 references
        public void OnSlowDown()
        {
            Time.timeScale = slow_down;
        }
    }
}
```