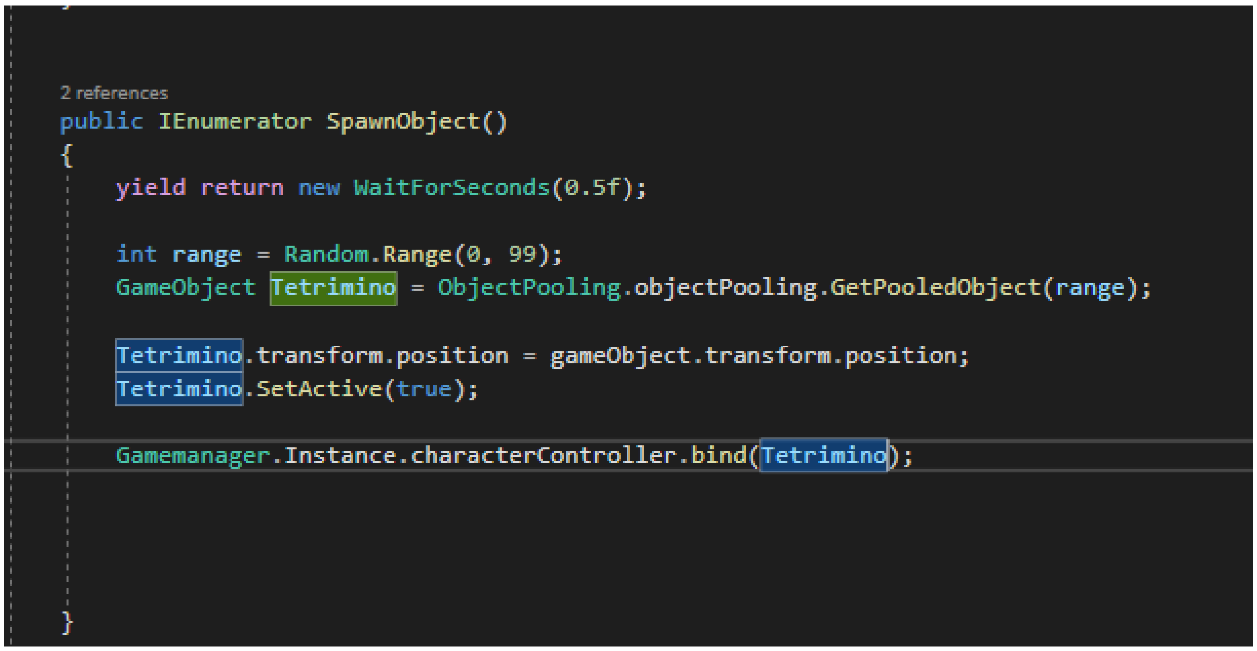
Daniel Barbier
100820942

1 + 8 + 4 + 2 =15

Singleton Pattern

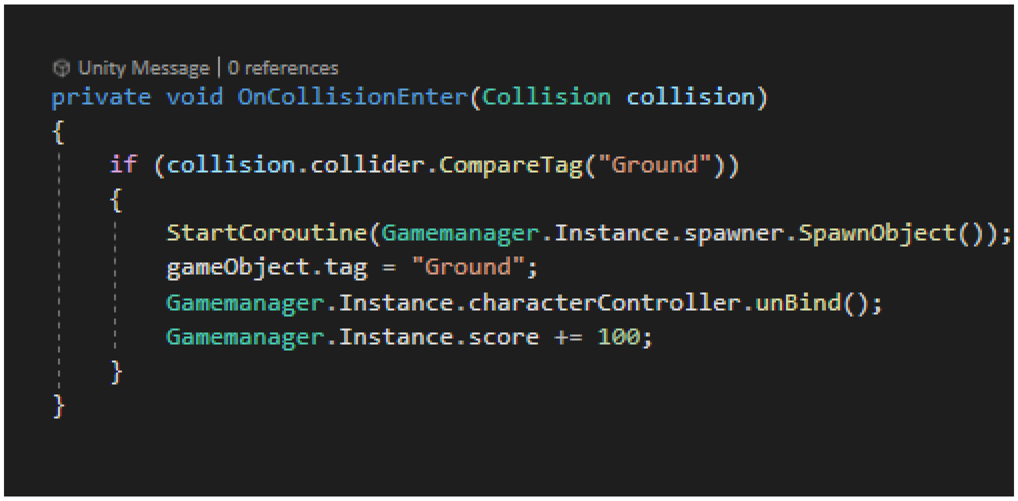
The singleton pattern is used in the game to keep track of several game objects in the scene It tracks the UI manager and the character controller which the piece controller can talk to it through the character controller when it hits the ground.

Binding the tetrimino to the character controller trough “Gamemanager” which is a singleton.



```
2 references
public IEnumerator SpawnObject()
{
    yield return new WaitForSeconds(0.5f);

    int range = Random.Range(0, 99);
    GameObject Tetrimino = ObjectPooling.objectPooling.GetPooledObject(range);

    Tetrimino.transform.position = gameObject.transform.position;
    Tetrimino.SetActive(true);

    Gamemanager.Instance.characterController.bind(Tetrimino);



}
```

Another instance where the “Gamemanager” is being used where it adds up score when a tetrimino hits the ground

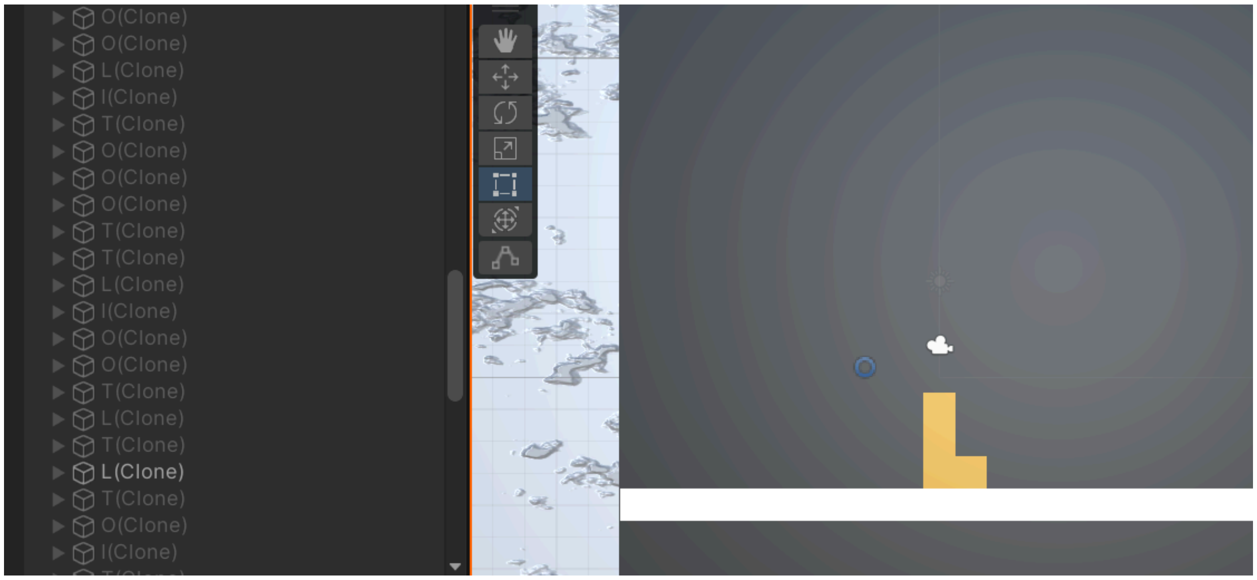


```
Unity Message | 0 references
private void OnCollisionEnter(Collision collision)
{
    if (collision.collider.CompareTag("Ground"))
    {
        StartCoroutine(Gamemanager.Instance.spawner.SpawnObject());
        gameObject.tag = "Ground";
        Gamemanager.Instance.characterController.unBind();
        Gamemanager.Instance.score += 100;
    }
}
```

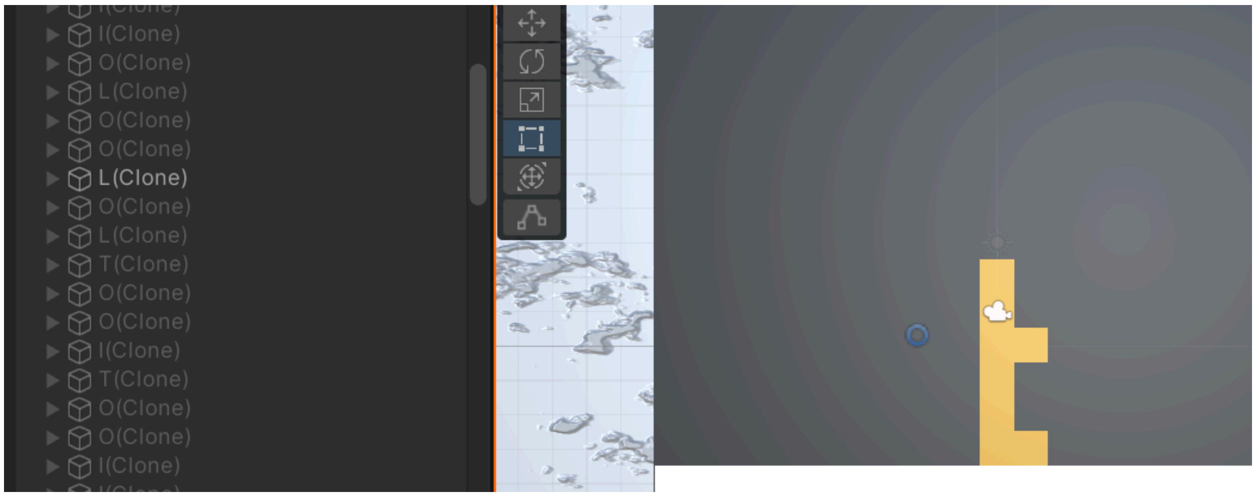
Object Pooling

The object pool takes all available tetriminos to be used in gameplay, it instantiates 100 instances of the tetriminos in a random order, the spawner then takes accesses the object pool and make it so that it takes the spawners location and then sets it to active.

The reason for doing this is to make it so that we don't have to instantiate and destroy every single time as it is costly in unity, and in this instance we don't wan't to destroy the tetriminos



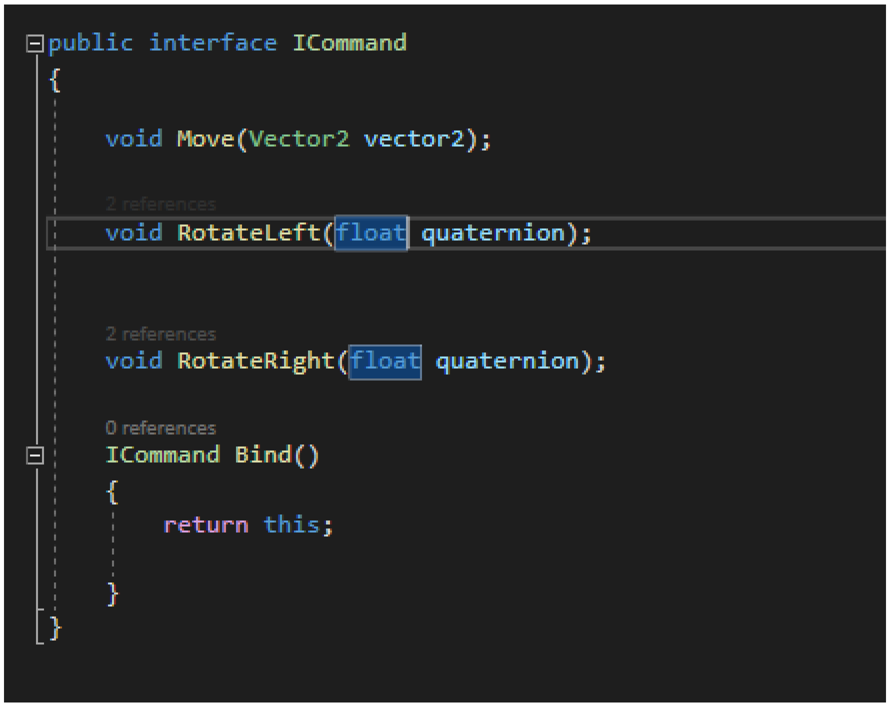
Different Object in the pool being called.



Command Pattern

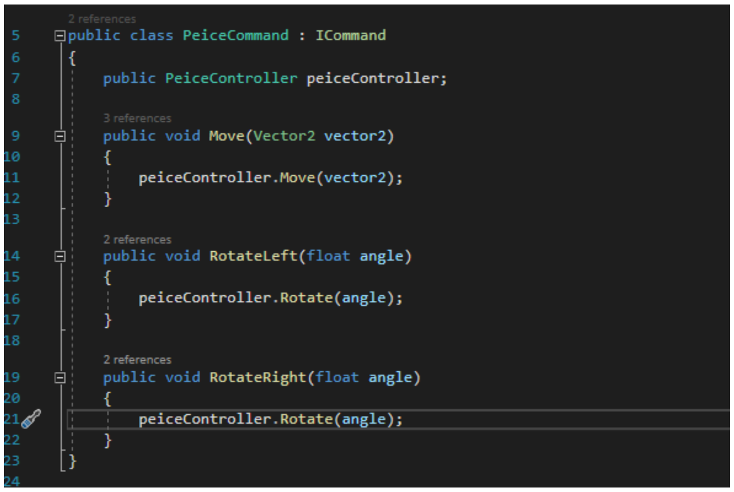
The command pattern is a pattern that makes it so that the commands and the player are separate, for this use case it is made to bind to tetriminos and as soon as it reaches the ground it rebinds to the most current one.

We can make the “Commands” do different things depending on how its set up

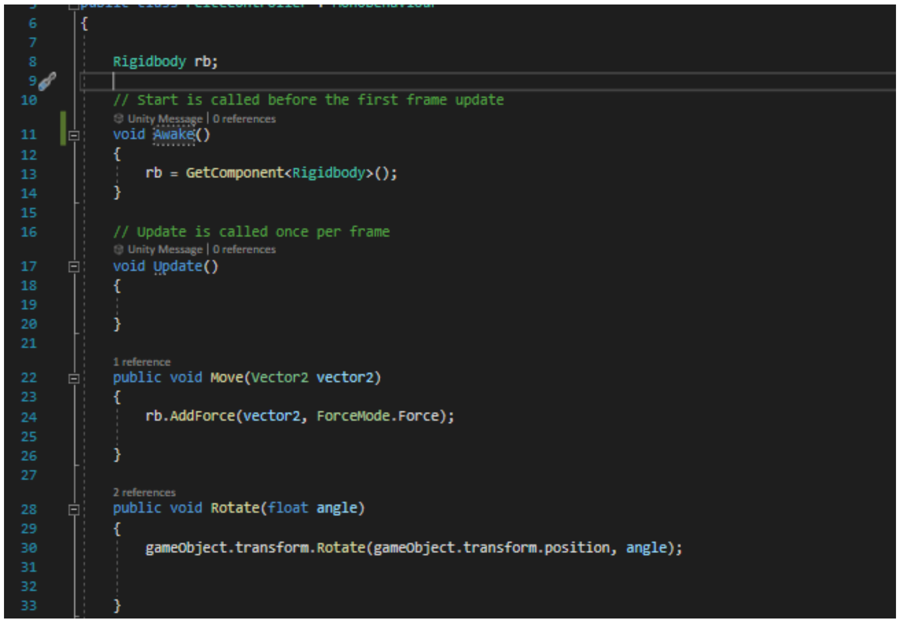


```
public interface ICommand
{

    void Move(Vector2 vector2);

    2 references
    void RotateLeft(float quaternion);


    2 references
    void RotateRight(float quaternion);

    0 references
    ICommand Bind()
    {
        return this;

    }
}
```

This is so that the Character controller can communicate with the command that was made and influence the pieces without the pieces getting a direct reference to the player controller



```
2 references
public class PeiceCommand : ICommand
{
    public PeiceController peiceController;

    3 references
    public void Move(Vector2 vector2)
    {
        peiceController.Move(vector2);
    }

    2 references
    public void RotateLeft(float angle)
    {
        peiceController.Rotate(angle);
    }

    2 references
    public void RotateRight(float angle)
    {
        peiceController.Rotate(angle);
    }
}
```



```
public class FeetController : MonoBehaviour
{

    Rigidbody rb;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Awake()
    {
        rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    1 reference
    public void Move(Vector2 vector2)
    {
        rb.AddForce(vector2, ForceMode.Force);

    }

    2 references
    public void Rotate(float angle)
    {
        gameObject.transform.Rotate(gameObject.transform.position, angle);


    }
```