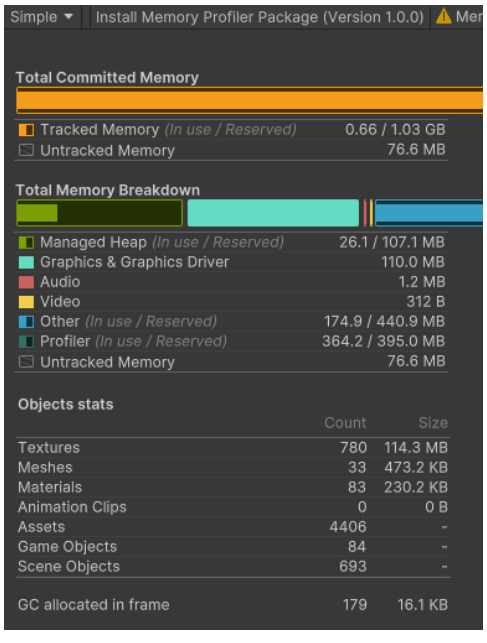
Ejemplo de Flyweight Pattern 3D y Materiales:

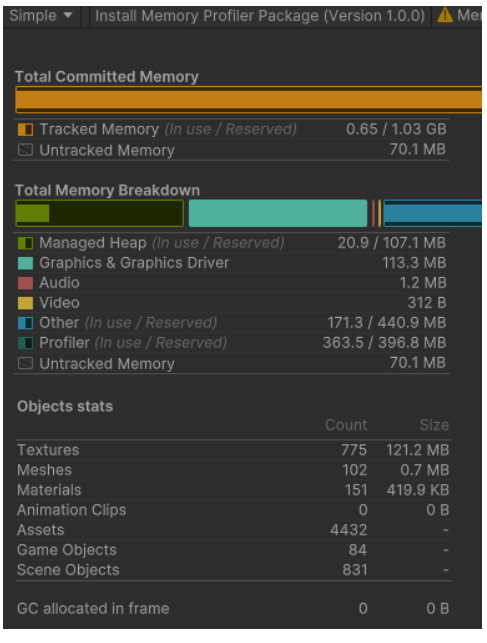
En el video mostrado en clase se puede observar cómo el uso de Flyweight ayuda a optimizar y reducir el uso de memoria a la hora de realizar un proceso en Unity, en este caso la utilización de materiales. En primer lugar nos da un ejemplo de cómo sería una implementación sin utilizar Flyweight, para que el color de un material cambie en todo momento:

```
public class ChangeColors : MonoBehaviour
{
    Renderer _renderer;
    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        _renderer = GetComponent<Renderer>();
        _propBlock = new MaterialPropertyBlock();


    }



    2 referencias
    private Color GetRandomColor()
    {
        return new Color(
            Random.Range(0f, 1f),
            Random.Range(0f, 1f),
            Random.Range(0f, 1f)
            );
    }

    private MaterialPropertyBlock _propBlock;

    Mensaje de Unity | 0 referencias
    void Update()
    {
        //_renderer.GetPropertyBlock(_propBlock);
        //_propBlock.SetColor("Color", GetRandomColor());
        //_renderer.SetPropertyBlock(_propBlock);

        _renderer.material.color = GetRandomColor();
        _renderer.material.SetColor("Color", GetRandomColor());
    }
}
```

Utilizando esto, a la hora de ejecutar el proyecto se puede observar un aumento significativo en el uso de la memoria. Especialmente en indicador de Meshes y Materials.

Antes de Ejecutar // Después de Ejecutar





Esto sucede debido a que se está queriendo acceder al material de dicho objeto, para esto primero se hace referencia del renderer, luego buscamos acceder a dicho material, y por último a su color, después de esto, pasamos a igualarlo con un color aleatorio, por lo que no es muy óptimo, debido a que para querer crear un color aleatorio, lo que se está haciendo en realidad es crear una copia del material con dicho color.

```
_renderer.material.color = GetRandomColor();
_renderer.material.SetColor("Color", GetRandomColor());
```

Por otro lado, para utilizar el Flyweight se está utilizando un MaterialPropertyBlock, aquí accedemos al bloque de propiedades de dicho material, y aquí se cambia el color, evitando crear nuevos materiales para cambiar el anterior. Haciendo referencia del propio material del objeto.

```
_renderer.GetPropertyBlock(_propBlock);
_propBlock.SetColor("_Color", GetRandomColor());
_renderer.SetPropertyBlock(_propBlock);
```

En conclusión, con el uso de Flyweight podemos reducir el uso de memoria, de diversos métodos, con la finalidad de optimizar más el proyecto.