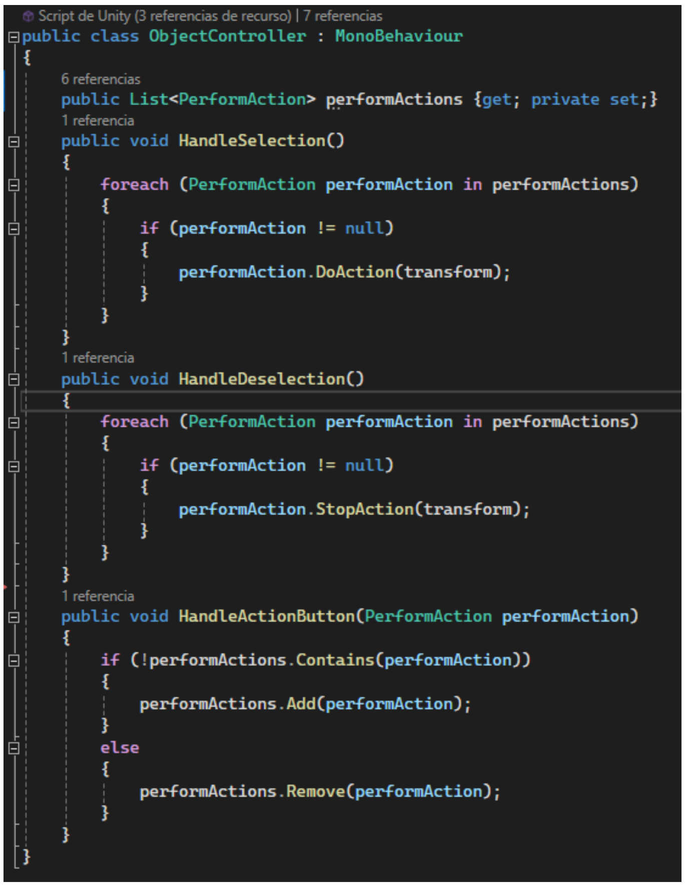
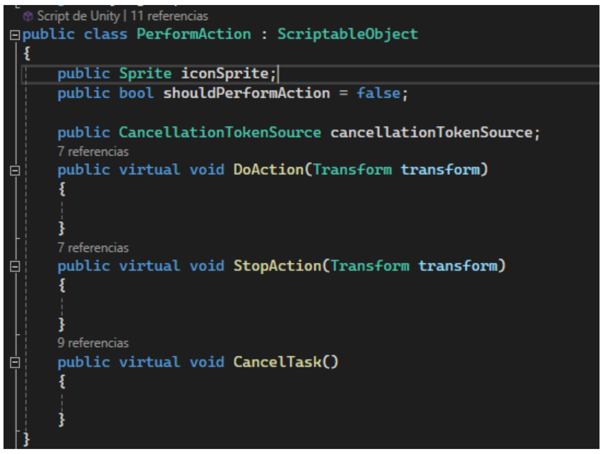
Everyrealm coding Test - Documentation

Following the Strategy Pattern, objects have generic methods to handle different situations, they are no aware of what behaviors are being set inside them:



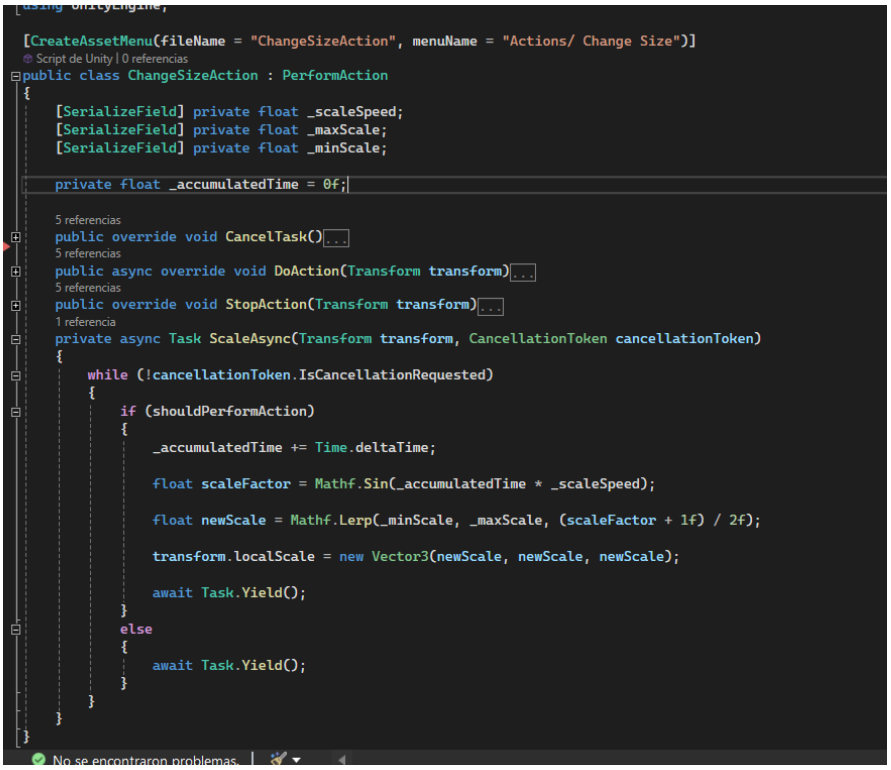
```
Script de Unity (3 referencias de recurso) | 7 referencias
public class ObjectController : MonoBehaviour
{
    6 referencias
    public List<PerformAction> performActions {get; private set;}
    1 referencia
    public void HandleSelection()
    {
        foreach (PerformAction performAction in performActions)
        {
            if (performAction != null)
            {
                performAction.DoAction(transform);
            }
        }
    }
    1 referencia
    public void HandleDeselection()
    {
        foreach (PerformAction performAction in performActions)
        {
            if (performAction != null)
            {
                performAction.StopAction(transform);
            }
        }
    }
    1 referencia
    public void HandleActionButton(PerformAction performAction)
    {
        if (!performActions.Contains(performAction))
        {
            performActions.Add(performAction);
        }
        else
        {
            performActions.Remove(performAction);
        }
    }
}
```

Behaviors are simple scriptable objects that inherit from the PerformAction class, so they also inherit from the virtual methods inside it:



```
Script de Unity | 11 referencias
public class PerformAction : ScriptableObject
{
    public Sprite iconSprite;
    public bool shouldPerformAction = false;

    public CancellationTokenSource cancellationTokenSource;
    7 referencias
    public virtual void DoAction(Transform transform)
    {

    }
    7 referencias
    public virtual void StopAction(Transform transform)
    {

    }
    9 referencias
    public virtual void CancelTask()
    {

    }
}
```

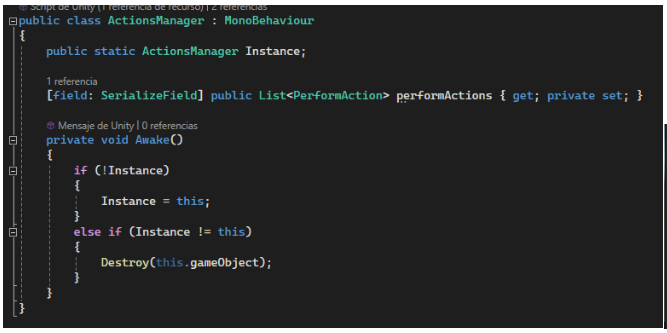
This ScriptableObjects hold all their data so that it can be accessed and easily modified in the inspector. Actions use Tasks to care for performance, instead of executing its methods on Update



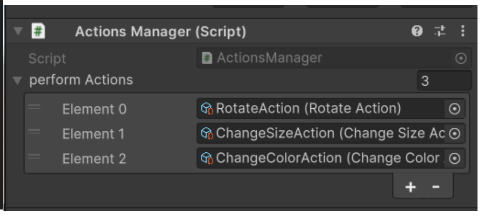
```csharp
using UnityEngine;

[CreateAssetMenu(fileName = "ChangeSizeAction", menuName = "Actions/ Change Size")]
Script de Unity | 0 referencias
public class ChangeSizeAction : PerformAction
{
    [SerializeField] private float _scaleSpeed;
    [SerializeField] private float _maxScale;
    [SerializeField] private float _minScale;

    private float _accumulatedTime = 0f;

    5 referencias
    public override void CancelTask()...
    5 referencias
    public async override void DoAction(Transform transform)...
    5 referencias
    public override void StopAction(Transform transform)...
    1 referencia
    private async Task ScaleAsync(Transform transform, CancellationToken cancellationToken)
    {
        while (!cancellationToken.IsCancellationRequested)
        {
            if (shouldPerformAction)
            {
                _accumulatedTime += Time.deltaTime;

                float scaleFactor = Mathf.Sin(_accumulatedTime * _scaleSpeed);

                float newScale = Mathf.Lerp(_minScale, _maxScale, (scaleFactor + 1f) / 2f);

                transform.localScale = new Vector3(newScale, newScale, newScale);

                await Task.Yield();
            }
            else
            {
                await Task.Yield();
            }
        }
    }
}
```

No se encontraron problemas.

A ActionsManager singleton keeps the information for the actions that we want to implement:

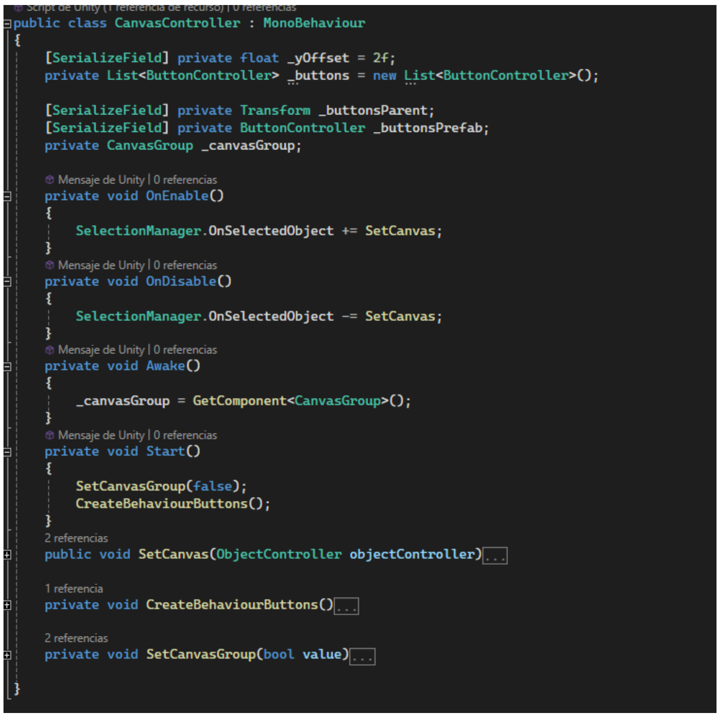


```csharp
Script de Unity (1 referencia de recurso) | 2 referencias
public class ActionsManager : MonoBehaviour
{
    public static ActionsManager Instance;

    1 referencia
    [field: SerializeField] public List<PerformAction> performActions { get; private set; }

    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        if (!Instance)
        {
            Instance = this;
        }
        else if (Instance != this)
        {
            Destroy(this.gameObject);
        }
    }
}
```



```
Actions Manager (Script)
Script            ActionsManager
perform Actions                          3
    Element 0     RotateAction (Rotate Action)
    Element 1     ChangeSizeAction (Change Size Ac
    Element 2     ChangeColorAction (Change Color
                                        +  -
```

A CanvasController class creates the necessary buttons for each action set in the ActionsManager, and sets the canvas in the event that is invoked when an object has been selected:



```
Script de Unity (1 referencia de recurso) | 0 referencias
public class CanvasController : MonoBehaviour
{
    [SerializeField] private float _yOffset = 2f;
    private List<ButtonController> _buttons = new List<ButtonController>();

    [SerializeField] private Transform _buttonsParent;
    [SerializeField] private ButtonController _buttonsPrefab;
    private CanvasGroup _canvasGroup;

    Mensaje de Unity | 0 referencias
    private void OnEnable()
    {
        SelectionManager.OnSelectedObject += SetCanvas;
    }
    Mensaje de Unity | 0 referencias
    private void OnDisable()
    {
        SelectionManager.OnSelectedObject -= SetCanvas;
    }
    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        _canvasGroup = GetComponent<CanvasGroup>();
    }
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        SetCanvasGroup(false);
        CreateBehaviourButtons();
    }
    2 referencias
    public void SetCanvas(ObjectController objectController)...

    1 referencia
    private void CreateBehaviourButtons()...

    2 referencias
    private void SetCanvasGroup(bool value)...

}
```

German Salas – Game Developer