# Attribute Declarations

If you need a reference to an object, component or script which is in the same object hierarchy as this script, use:

[SerializeField] private GameObject \_myObject;

And drag the corresponding object from the hierarchy to the box in the inspector in Unity Editor.

If you need a reference to a different object in the scene, use:

private GameObject \_aForeignObject;

Declare any state as private:

private float \_speed;

private Dictionary<int, PlayerState> \_playerState;

Only use public attributes when it really makes sense:

public Constants.Team Team;

Declare any events that this script will trigger:

public static event Action<GameController> gameActive;

public static event Action gameStarted;

# Awake()

This is the first method that gets called.

Destroy this script if it’s not needed. For example, we destroy *CollectingCrystals.cs* if it’s attached to a Guardian and we destroy *PlayerNameTag.cs* if it’s attached to your player.

Destroy(this);

Instantiate all your personal attributes here:

\_speed = 0.5f;

\_playerState = new Dictionary<int, PlayerState>();

# OnEnable()

This is the next method that gets called just after Awake() and when enabled/set active.

Subscribe your functions to events here:

GameObject.gameActive += OnGameActive;

# Start()

This gets called after Awake() and OnEnable() have been called for all scripts.

Since we know all other scripts have executed their own Awake() function and initialised themselves, it’s now safe to access other scripts:

\_preGame = GameObject.FindWithTag(“PreGameController”);

Perform any actions that need to happen when the script starts:

transform.position = GetRandomSpawnPoint();

# Update()

This gets called every frame.

Handle key presses here:

if (Input.GetKeyDown(KeyCode.Escape)) \_paused = !\_paused;

Perform any actions that need to happen every frame:

UpdatePosition();

SavePlayerState();

# OnDisable()

This gets called just before the associated object is disabled, set inactive or destroyed.

Unsubscribe from events (to avoid null reference errors):

GameObject.gameActive -= OnGameActive;

# Event Subscribe Methods

If something needs to happen once, use a method that gets called when an event is triggered.

private void OnGameActive()

{

Destroy(\_tutorial);

}

private void OnPaused(bool paused)

{

\_lockMovement = paused;

\_lockRotation = paused;

}

# Private Methods

You should be able to use private methods most of the time. The more self-contained your script is, the better. If you do need to access something from outside, consider if an event trigger can be used.

# Public Methods

Only use a public method if it really makes sense for one script to command another or if it’s not possible to use an event.

Here, TimeConn needs to command ParticleController to trigger a dissolve and be notified when the dissolve is finished. Since we only want the TimeConn and ParticleController attached to the same Player object to interact with each other, we have to use public functions. This is because an event would trigger the function on ALL active player objects in the scene.

public void StartDissolving(Constants.JumpDirection jd, bool dissolveOut)

{

…

}

public void NotifyStoppedDissolving(bool dissolveOut)

{

…

}