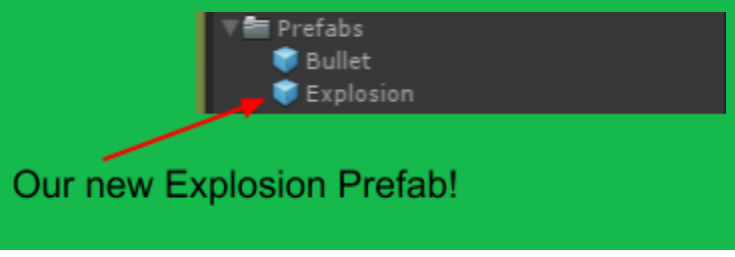
Zombie Shooter Project 3b

Task 1. Prefab the Explosion

Do this

- Select the **Explosion** GameObject in the **Hierarchy**
- Drag the **Explosion** GameObject into the **Prefabs folder** in the **Project view** to create an **Explosion Prefab**



Useful links

- Learn more about Prefabs [Prefabs - Manual]

Task 2. Make the explosion destroy itself when the animation is finished

Explanation

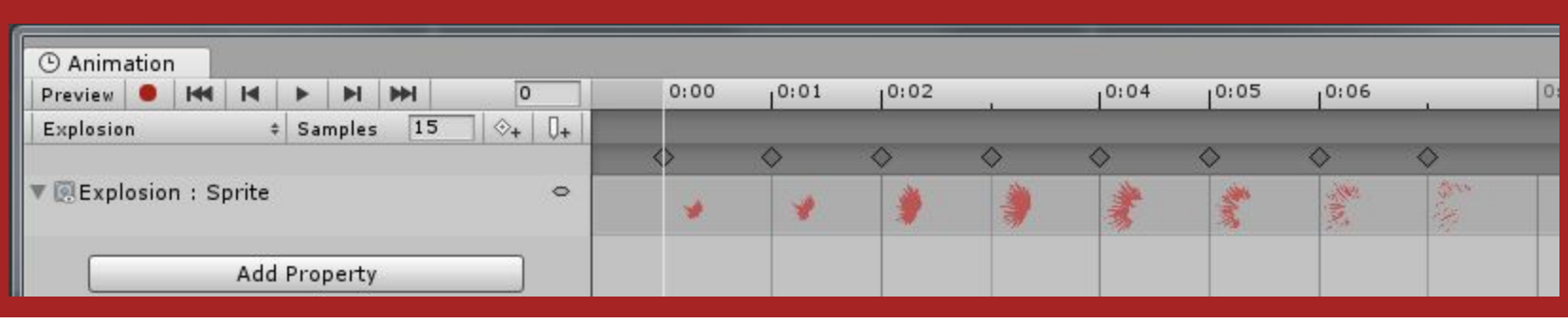
- Get the animation to destroy the Explosion GameObject when it finishes

Do this

- Select the **Explosion** GameObject in the **Scene view**
- Drag the **DestroyOnDie** script onto the prefab

Check this

- Check you can see the animation we just created for the Explosion in the Animation view

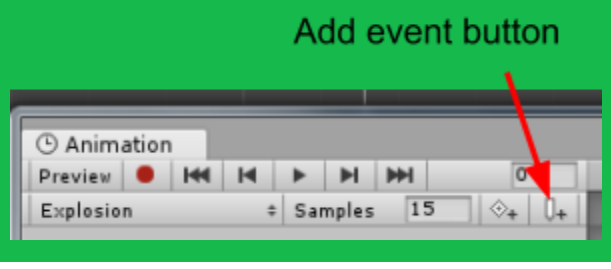


Explanation

- We can add an **Animation Event** in the **Animation view** to run a method at a specific point in our animation
- The method we run will need to be available to the same GameObject we have the animation on
- We want the **Explosion** to Destroy itself when it is finished animating
- We can use the **Die** method **DestroyOnDie** script for this!

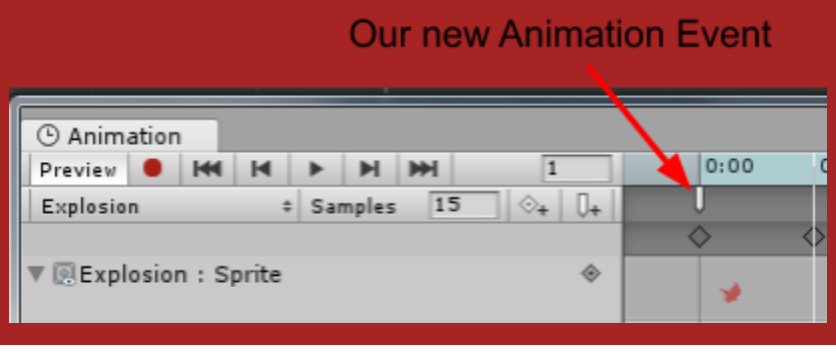
Do this

- In the **Animation view**, click the **Add Event** button



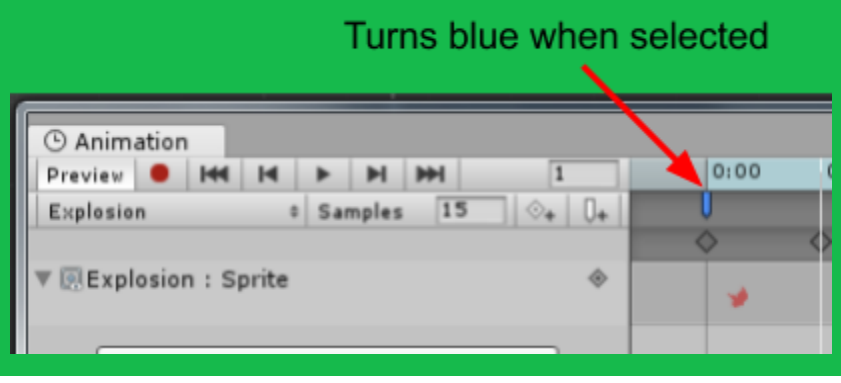
Check this

- Check you can see the small white event box in the **Animation view**



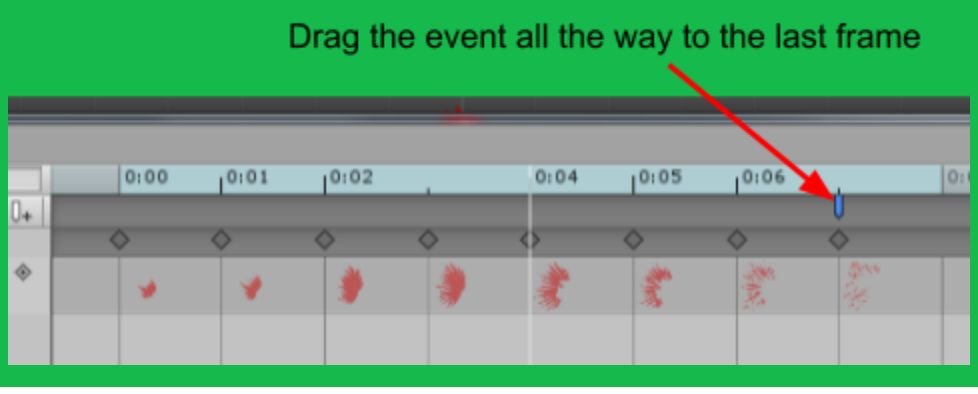
Do this

- In the **Animation view**, select the **Animation Event** we just created - it should turn blue when selected



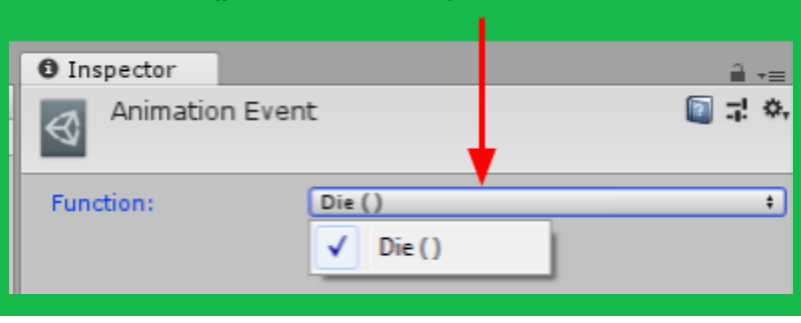
Do this

- In the **Animation view**, drag the Animation Event to the last frame of the animation



Do this

- Select the **Animation Event** if it isn't already (it will be blue if selected)
- In the **Inspector** select the **Die ()** option from the dropdown



Explanation

- We have connected our **Animation Event** to the **Die** method of the **DestroyOnDie** Component we added to the **Explosion**
- This will run the **Die** method when the explosion animation hits the last frame, destroying it

Useful links

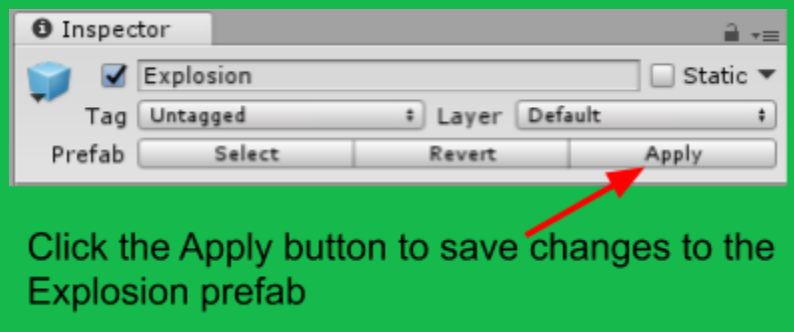
- More information about **Animation Events** — Animation Events - documentation
- More information about **Animation Events** — Animation Events - scripting reference

Explanation

- We want to save our changes to the **Explosion** GameObject in the **Scene view** to its prefab in the **Project view**
- We cannot save these until we close the **Animation view** first

Do this

- Close the **Animation view** using the small “x” button on the top right corner of **Animation view** window
- Select the **Explosion** GameObject in the **Scene view**
- In the **Inspector** click the **Apply** button to apply our changes to the **Explosion** prefab



Click the Apply button to save changes to the Explosion prefab

Task 3. Spawn an explosion when the zombie dies

Explanation

- We now have a blood splat **Explosion**
- We can use this on a **Zombie**
- When a **Zombie dies**, we can **spawn** an **Explosion** at the moment the Zombie gets destroyed
- We will create a simple **Spawner** script to do this
- We can use this **Spawner** script in many other places!

Do this

- In the **Project view**, create a new C# script in the **Scripts** folder
- Name the script **Spawner**
- Double click the the **Spawner** script to open for editing

Do this

- Type out this code into your script file
- Make sure your code is ***EXACTLY*** the same!

```
using UnityEngine;

public class Spawner : MonoBehaviour {
    public GameObject prefabToSpawn;
    public float adjustmentAngle = 0;

    public void Spawn(){
        Vector3 rotationInDegrees = transform.eulerAngles;
        rotationInDegrees.z += adjustmentAngle;

        Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

        Instantiate(prefabToSpawn, transform.position, rotationInRadians);
    }
}
```

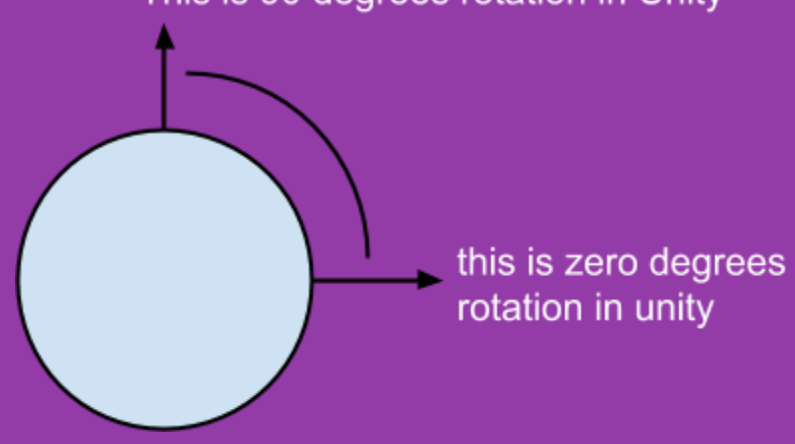
Explanation - prefabToSpawn property

- The **prefab** in the **Project view** we wish to spawn

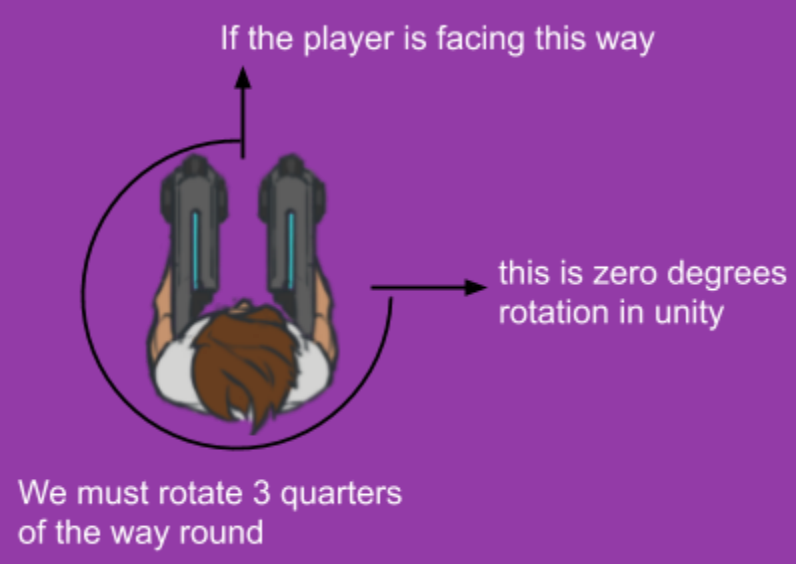
```
public GameObject prefabToSpawn;
```

Explanation - adjustmentAngle property

- We need to be able to **compensate** for the **artwork not facing the right way.**
- Unity sees angles as starting from the right:



If our artwork were facing up (or 90 degrees according to Unity) we need a way of offsetting this so our artwork faces the right way
Artwork facing up would need to be turned 3/4 of the way round until it faces to the right




```
public float adjustmentAngle = 0.0f;
```

Explanation - Our custom Spawn method

Get our current rotation in degrees

Add our adjustmentAngle to the z axis

```
public void Spawn ()
{
    Vector3 rotationInDegrees = transform.eulerAngles;
    rotationInDegrees.z += adjustmentAngle;

    Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

    Instantiate(prefabToSpawn, transform.position, rotationInRadians);
}
```

Spawn our thing into the scene

Convert our rotation in degrees to radians

Explanation - Line 1

- Create a **Vector3** variable to store our rotation in degrees
- **eulerAngles** is our rotation on the X,Y, and Z axes in degrees

```
public void Spawn(){
    Vector3 rotationInDegrees = transform.eulerAngles;
    rotationInDegrees.z += adjustmentAngle;

    Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

    Instantiate(prefabToSpawn, transform.position, rotationInRadians);
}
```

Useful links

- More information about **transform.eulerAngles** — transform.eulerAngles - scripting reference

Explanation - Line 2

- Add our public variable, **adjustmentAngle** to the Z axis of our **rotationInDegrees** variable
- We use the **+=** operator to add the **adjustmentAngle**

```
public void Spawn(){
    Vector3 rotationInDegrees = transform.eulerAngles;
    rotationInDegrees.z += adjustmentAngle;

    Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

    Instantiate(prefabToSpawn, transform.position, rotationInRadians);
}
```

Useful links

- More information about **+= operator** — += operator - scripting reference

Explanation - Line 3

- We want to convert our degrees into radians again.
- We can do this using **Quaternion.Euler**
- We give **Quaternion.Euler** a **Vector3** and it will convert the values in there to radians
- We create a **Quaternion** variable **rotationInRadians** to store our conversion

```
public void Spawn(){
    Vector3 rotationInDegrees = transform.eulerAngles;
    rotationInDegrees.z += adjustmentAngle;

    Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

    Instantiate(prefabToSpawn, transform.position, rotationInRadians);
}
```

Useful links

- More information about **Quaternion.Euler** — Quaternion.Euler - scripting reference

Explanation - Line 4

- We spawn our item here, using **Instantiate**
- We want to place our item and set its rotation, so we give **Instantiate** 3 parameters:
    - The prefab to spawn
    - A position to spawn it at
    - A rotation to spawn it at
- We can give **Instantiate** our **rotationInRadians** so it spawns facing the way set by **adjustmentAngle**

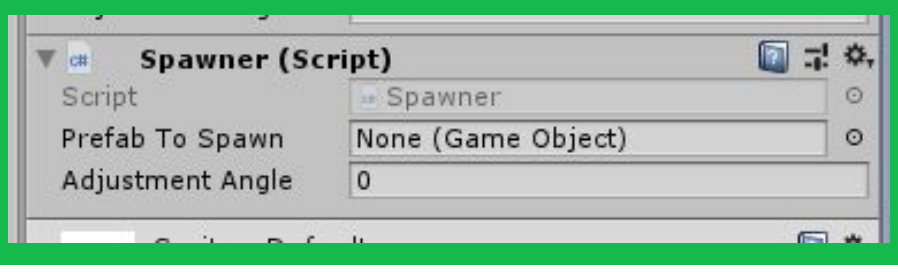
```
public void Spawn(){
    Vector3 rotationInDegrees = transform.eulerAngles;
    rotationInDegrees.z += adjustmentAngle;

    Quaternion rotationInRadians = Quaternion.Euler(rotationInDegrees);

    Instantiate(prefabToSpawn, transform.position, rotationInRadians);
}
```

Useful links

- More information about **Instantiate** — Instantiate - scripting reference

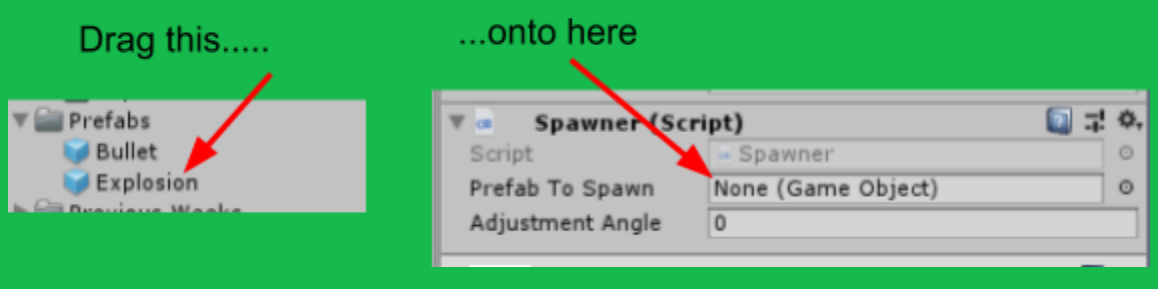
Do this

- In the **Scene view**, select the **Zombie** GameObject
- Drag the **Spawner** script onto the **Zombie** in the Inspector



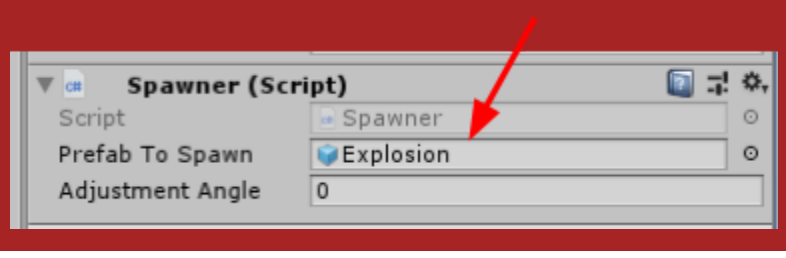
Do this

- In the **Project view**, drag the **Explosion** prefab onto the **Prefab To Spawn** inlet of the **Spawner** in the **Inspector**



Check this

- Check you have the **Explosion** prefab on the **Zombie** GameObject's **Spawner** component in the **Prefab To Spawn** inlet

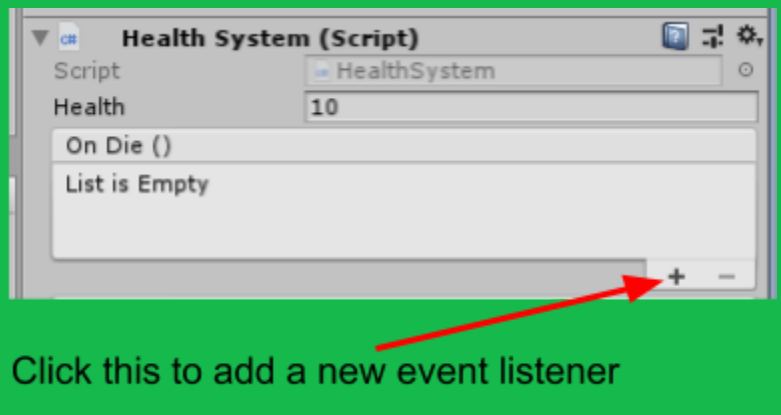


Explanation

- We want to the **Explosion** to spawn when our zombie dies
- The **Zombie** has a **HealthSystem** component, which has an event, **onDie**
- Our **Spawner** can listen for the **onDie** event and spawn the **Explosion** when the event runs
- We can add listeners to the **onDie** event in the Unity Editor

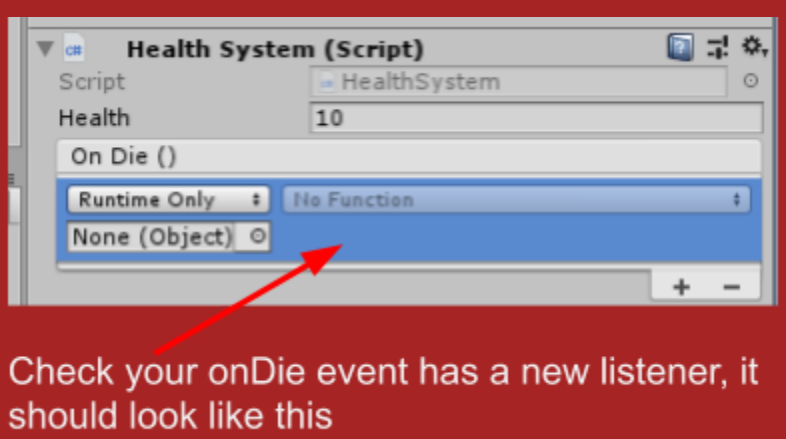
Do this

- In the **Scene view**, select the **Zombie** GameObject
- On the **HealthSystem** Component in the Inspector, click the **+** button underneath the **onDie** event to add an event listener



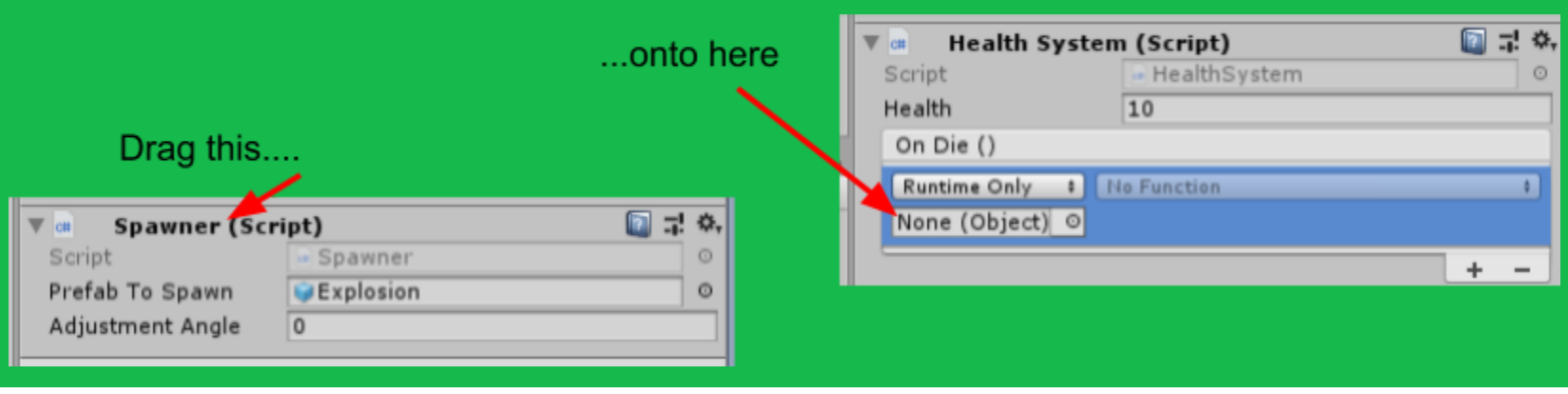
Check this

- Check you have a new event listener for the onDie event



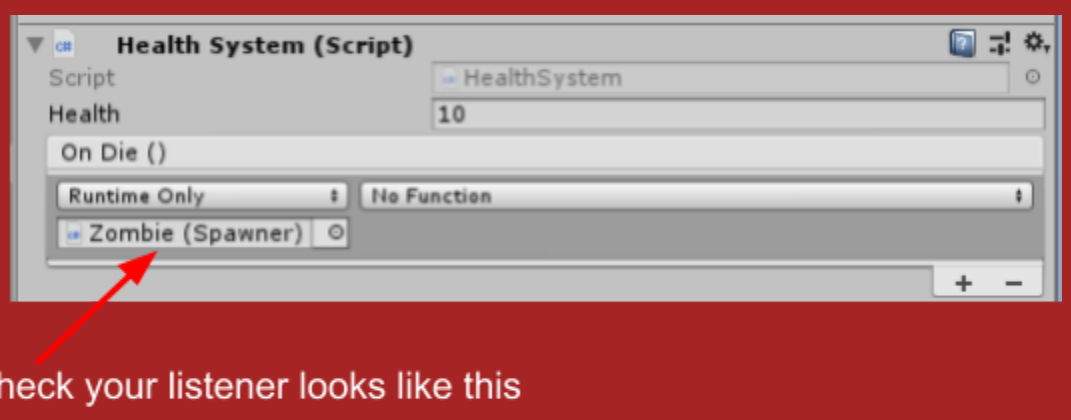
Do this

- In the **Inspector**, drag the **Spawner** component onto the inlet on the new **onDie** listener
- NOTE: drag the **Spawner** from the title at the top of the component, where it says “Spawner (Script)”



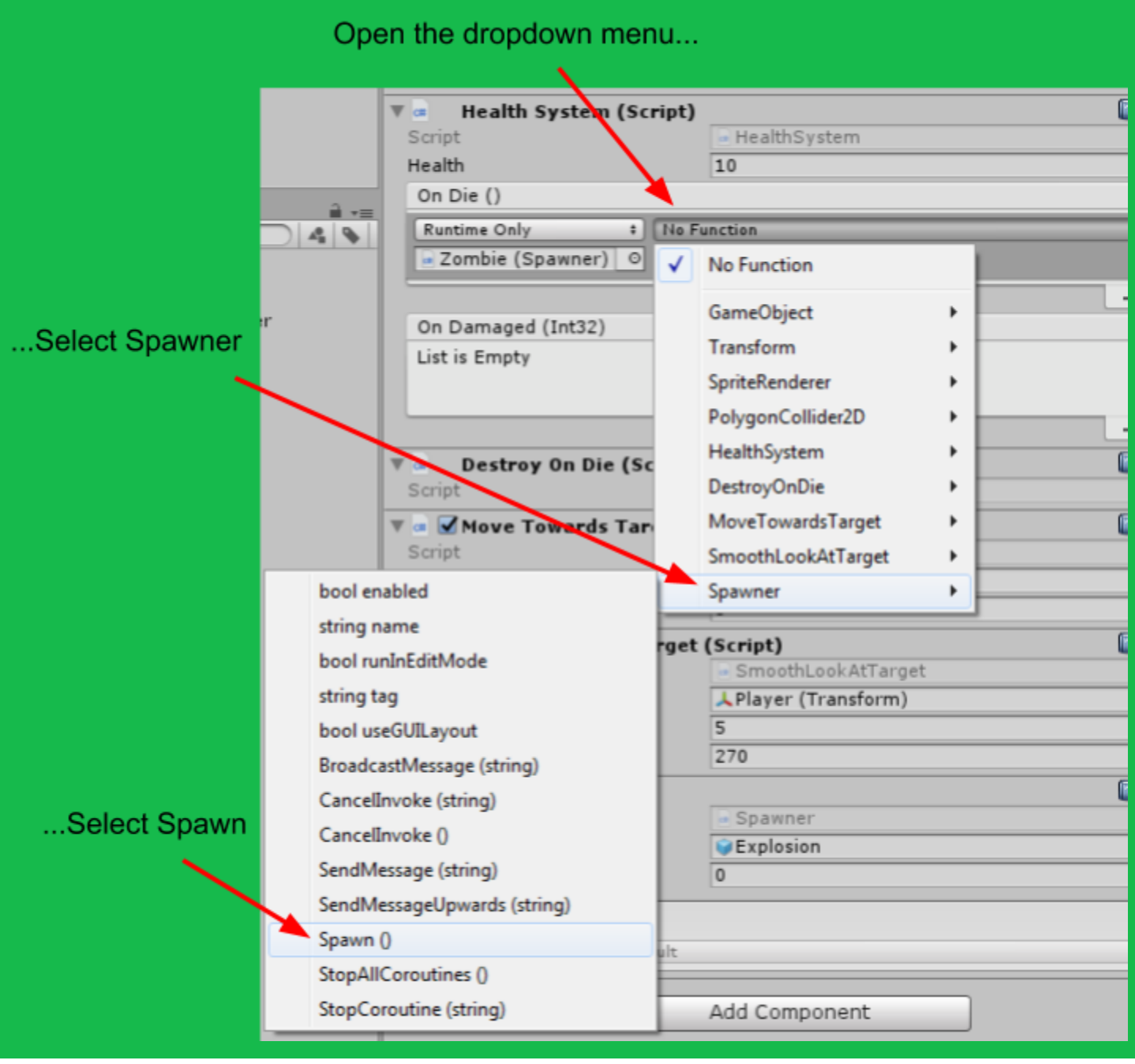
Check this

- Check you have the inlet filled for the listener
- It should say “Zombie (Spawner)”



Do this

- In the **onDie** event, select following dropdown menu options:
- Select **Spawner**, then select **Spawn**



Check this

- Check you have the **Spawner.Spawn** method selected on the dropdown menu for the **onDie** event listener