Zombie Shooter Project 4d

Task 2. Give the player health

Explanation

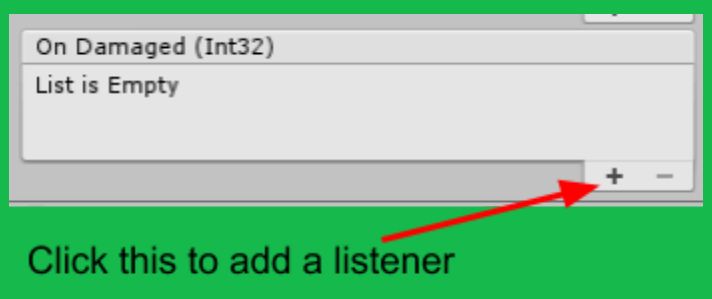
- The player needs some health, we can use the **HealthSystem** script!
- We want the Game UI to get our current health, so we connect the **HealthSystem** event to the Code we just added to the player
- This will update our **HealthBar** in the UI based on the **health** value in our Heros **HealthSystem**

Do this

- Add the **HealthSystem** script to the **Hero** GameObject in the **Hierarchy**
- Set the **initialHealth** to 100 on the **HealthSystem** component

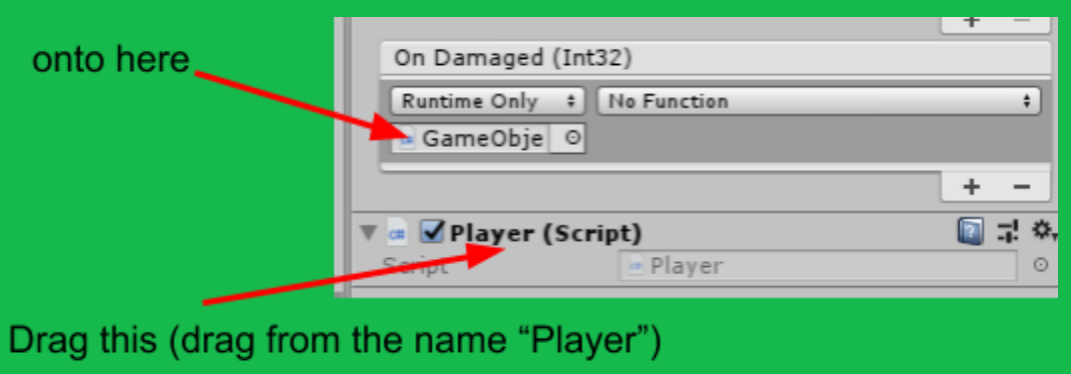
Do this

- Add a listener to the **On Damaged** event on the **HealthSystem** component



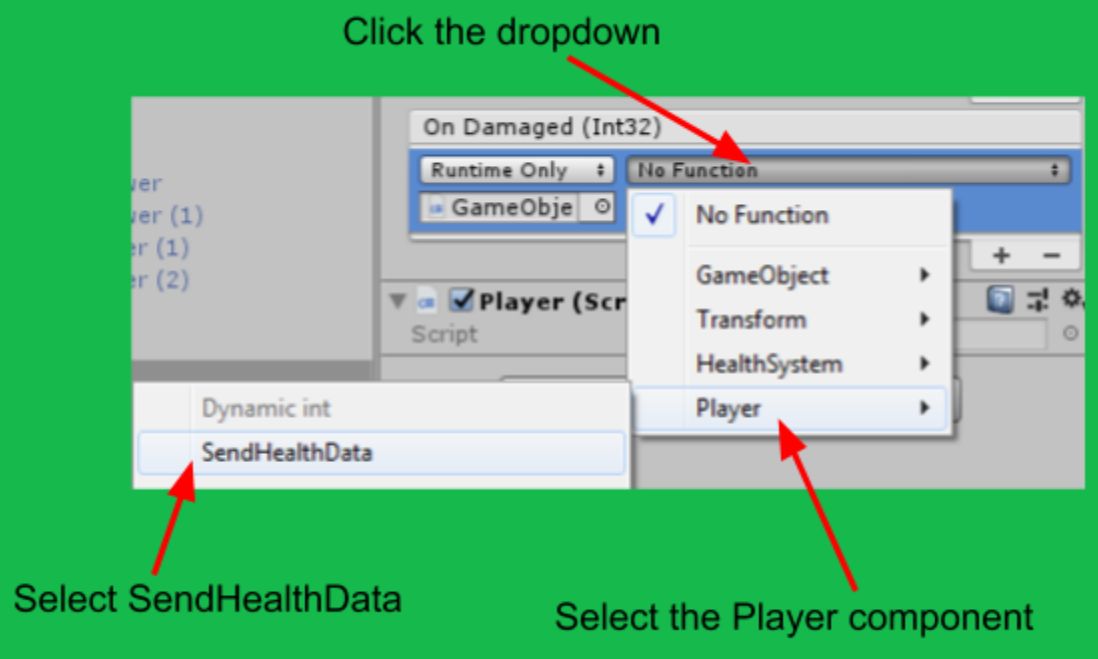
Do this

- Drag the **Player** component onto the inlet on the **On Damaged** event



Do this

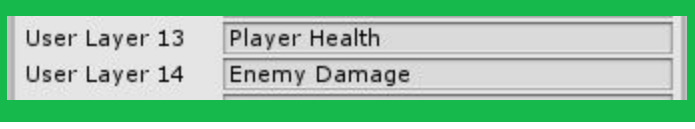
- Select the **SendHealthData** method from the **Player** component in the dropdown



Task 2. Setup Layers for damaging the player and zombies to attack

Do this

- Select the **Hero** GameObject in the **Hierarchy**
- Click the **Layer** dropdown at the top of the **Inspector**
- Select **Add Layer** from the dropdown
- Add the following layers



Do this

- Open the Physics2D settings
- Setup the new layers like the diagram shown

Do this

- Select the **Hero** GameObject in the **Hierarchy**
- Click the **Layer** dropdown at the top of the **Inspector**
- Select **Player Health**

Do this

- Select the **Zombie** GameObject in the **Hierarchy**
- Click the **Create** Button in the **Hierarchy**
- Select **Create Empty Child**
- Name the New GameObject **Hurt Trigger**

Do this

- Select the **Hurt Trigger** GameObject in the **Hierarchy**
- Click the **Layer** dropdown at the top of the **Inspector**
- Select **Enemy Damage**

Do this

- Select the **Zombie** GameObject in the **Hierarchy**
- Click the **Apply** button at the top of the **Inspector**
- NOTE: this will apply our changes to the **Zombie** prefab

Task 3. Make the zombie hurt the player

Explanation

- The zombie currently has no way of damaging the player
- The zombie needs to attack the player when in range, once every second

Do this

- In the **Project view**, create a new **C# Script** in the **Scripts Folder**
- Name the Script **GameUI**

Do this

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

```
using UnityEngine;

public class HurtTrigger : MonoBehaviour {
    public int damage;
    public float resetTime = 0.25f;

    private void OnTriggerEnter2D(Collider2D collision) {
        collision.transform.SendMessage("TakeDamage", damage, SendMessageOptions.DontRequireReceiver);
        GetComponent<Collider2D>().enabled = false;
        Invoke("ResetTrigger", resetTime);
    }

    private void ResetTrigger() {
        GetComponent<Collider2D>().enabled = true;
    }
}
```

Explanation - damage property

- This is the **damage** to be applied to the player

```
public int damage;
```

Explanation - resetTime property

- This is the time between the collider being disabled and enabled
- It will be used to reset the collider to hurt the player every time the collider is triggered

```
public float resetTime = 0.25f;
```

Explanation - OnTriggerEnter2D method

- The **OnTriggerEnter2D** method is a MonoBehaviour method
- **OnTriggerEnter2D** is called every time a GameObject **overlaps** another GameObject
    - Both GameObjects require Collider2D components

```
private void OnTriggerEnter2D(Collider2D other) {

}
```

Useful links

- More information about **OnTriggerEnter2D** — OnTriggerEnter2D - Scripting Reference
- More information about **Collider2D** — Collider2D - Scripting Reference

Explanation - Line 1

- We want to tell the **Hero** it has been damaged and give it our **HurtTriggers damage**
- The **Hero** will have a **TakeDamage** method that takes an int variable for **damage**
- We will use the SendMessage method (part of the Monobehaviour class) to run the **TakeDamage** method on the **Hero** and give it our damage variable
- We can access the **Hero** transform using the "other" parameter in the **OnTriggerEnter** method
- The **Hero** transform will give us access to it's SendMessage method
- NOTE: SendMessage can give an error if it can't run the method - like if we don't have a **TakeDamage** method on the **Hero**!
- NOTE: we can "ignore" the error using **SendMessageOptions** - it has a setting called "DontRequireReceiver" for this!

- SendMessage requires 3 parameters - a method name (as a string), an optional parameter to send and an optional "options"
- Our method name is "TakeDamage"
- Our optional parameter is damage (our public variable **damage** on the bullet)
- Our "options" is DontRequireReciever, which ignores errors if the "TakeDamage" method is not found on the **Hero** we hit

```
private void OnTriggerEnter2D(Collider2D collision) {
    collision.transform.SendMessage("TakeDamage", damage, SendMessageOptions.DontRequireReceiver);
    GetComponent<Collider2D>().enabled = false;
    Invoke("ResetTrigger", resetTime);
}
```

Useful links

- More information about **SendMessage** — SendMessage - Scripting Reference
- More information about **SendMessageOptions** — SendMessageOptions - Scripting Reference

Explanation - Line 2

- We want to use **OnTriggerEnter2D** again every second so the **Hero** takes damage
- We can disable the **Collider2D** component and enable it again after a second to apply damage again if the hero is still in range
- First we disable the **Collider2D** by using **GetComponent** and setting the **Collider2D** component's **enabled** property to **false**

```
private void OnTriggerEnter2D(Collider2D collision) {
    collision.transform.SendMessage("TakeDamage", damage, SendMessageOptions.DontRequireReceiver);
    GetComponent<Collider2D>().enabled = false;
    Invoke("ResetTrigger", resetTime);
}
```

Explanation - Line 3

- Here we set a timer to enable our Collider2D again
- The **Invoke** method will call the **ResetTrigger** method after the time specified in the **resetTime** variable
- resetTime is a public variable, so we can set it later in the editor
- The **ResetTrigger** is a custom method in our **HurtTrigger** class
- NOTE: Invoke requires the method name to be a string, "ResetTrigger"

```
private void OnTriggerEnter2D(Collider2D collision) {
    collision.transform.SendMessage("TakeDamage", damage, SendMessageOptions.DontRequireReceiver);
    GetComponent<Collider2D>().enabled = false;
    Invoke("ResetTrigger", resetTime);
}
```

Explanation - ResetTrigger method

- This custom method will reset our Collider2D's enabled property to true
- This will let the OnTriggerEnter2D method run again if the hero is in range of the zombie

```
private void ResetTrigger() {
    GetComponent<Collider2D>().enabled = true;
}
```

Explanation - Line 1

- Use Getcomponent to get the Collider2D, then set it's enabled property to true

```
private void ResetTrigger() {
    GetComponent<Collider2D>().enabled = true;
}
```

Do this

- Select the **Zombie** prefab in the **Project view**
- Open the **Zombie** GameObject and select the “**Hurt Trigger**” child
- Add the **HurtTrigger** script to the **HurtTrigger** GameObject in the **Inspector**
- Set the **damage** on the **HurtTrigger** component to 1