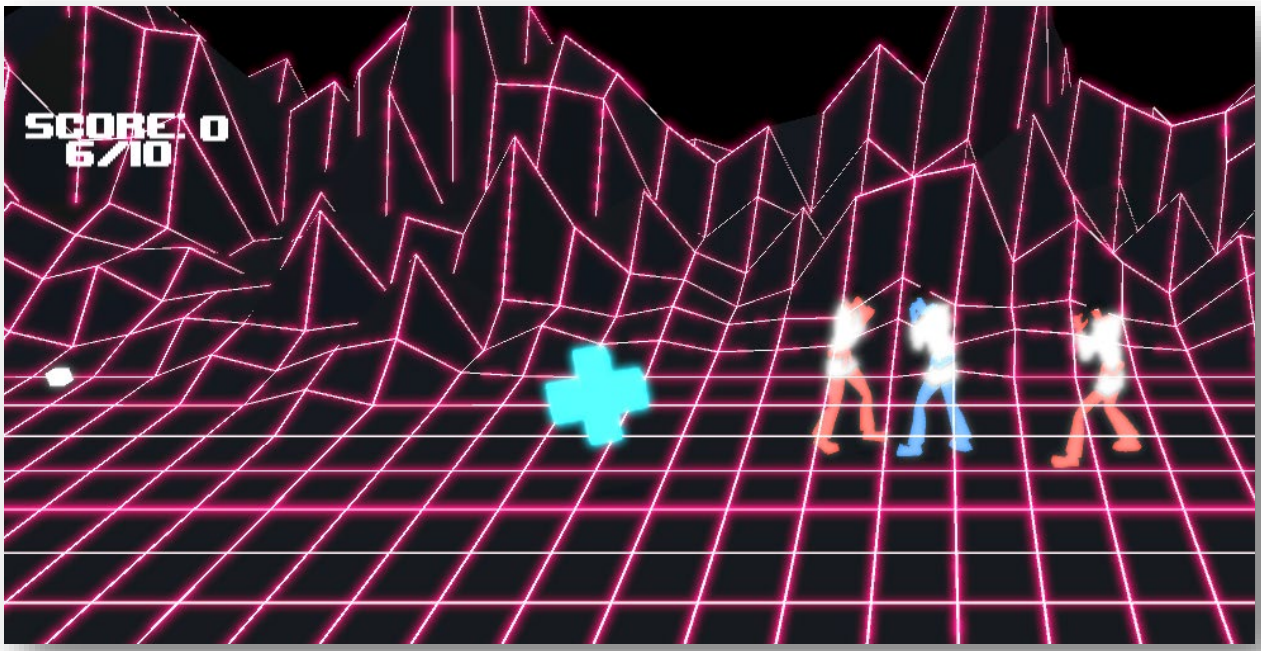
CODE NINJAS®

# Silver Belt Ninja Guide
# Activity 10: CyberFu Part 2

Activity 10

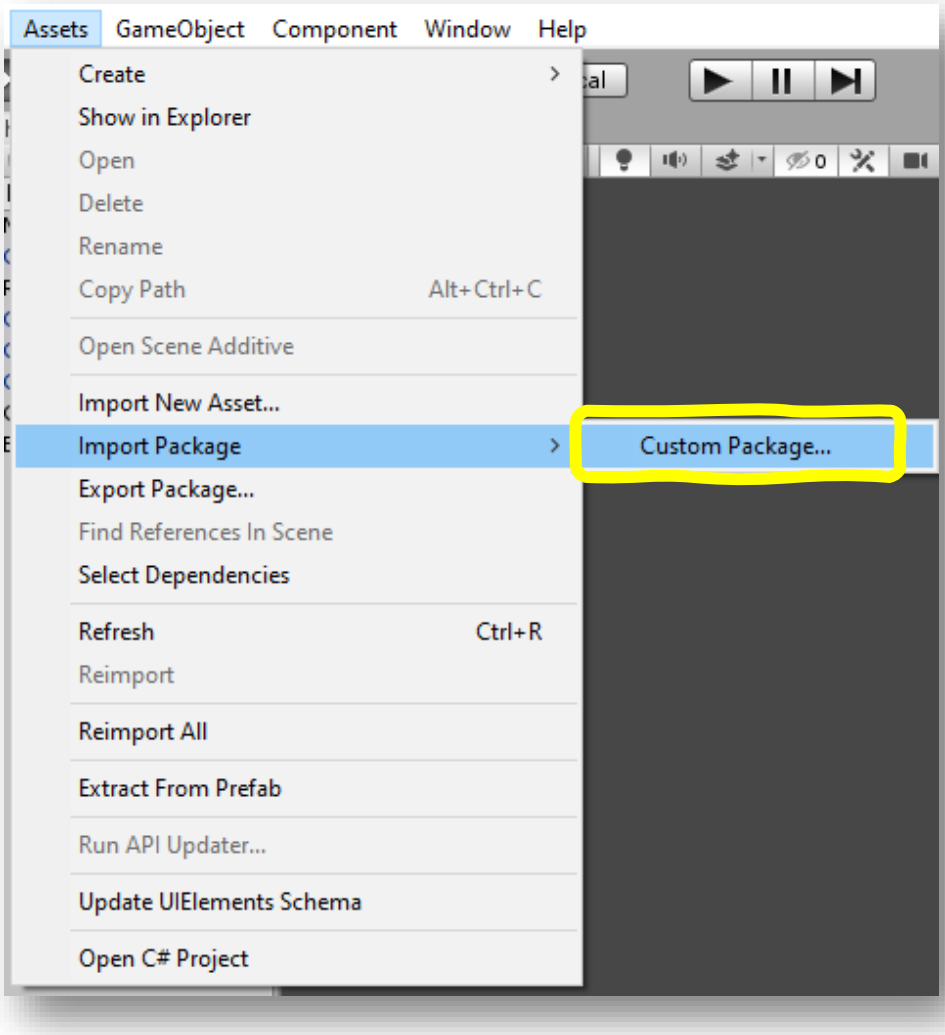
# CyberFu - Part 2

In this lesson, you will add health to both the player and the enemy, then program what happens when they run out of health.

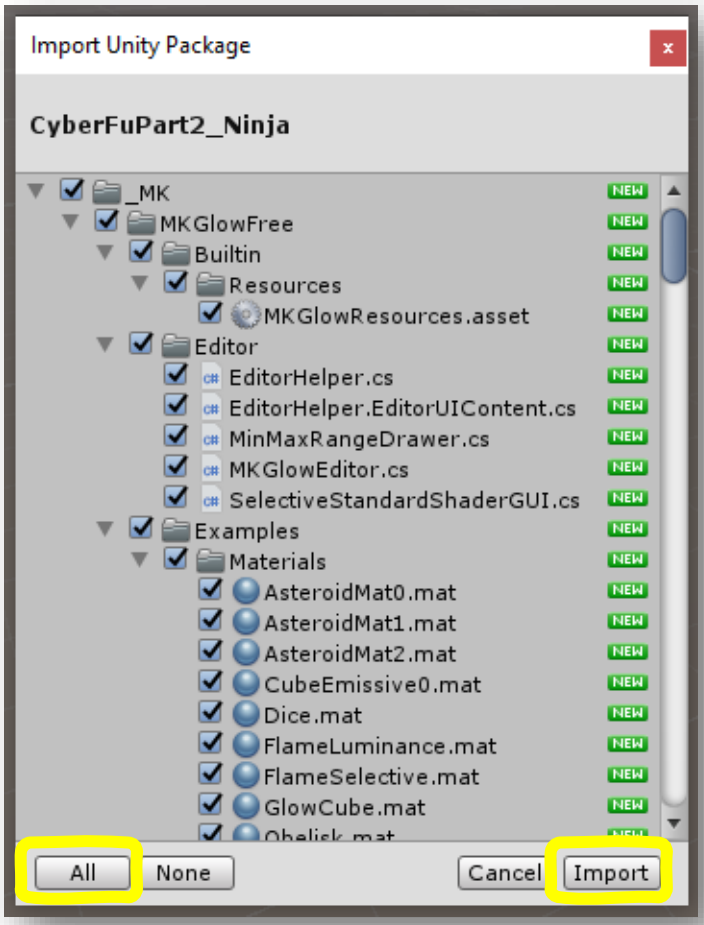


1 Start a new **Unity** Project and name it *YOUR INITIALS – CyberFu Part 2*. Select the **3D core** and place the project it in the correct folder. Click the "**Create**" button.

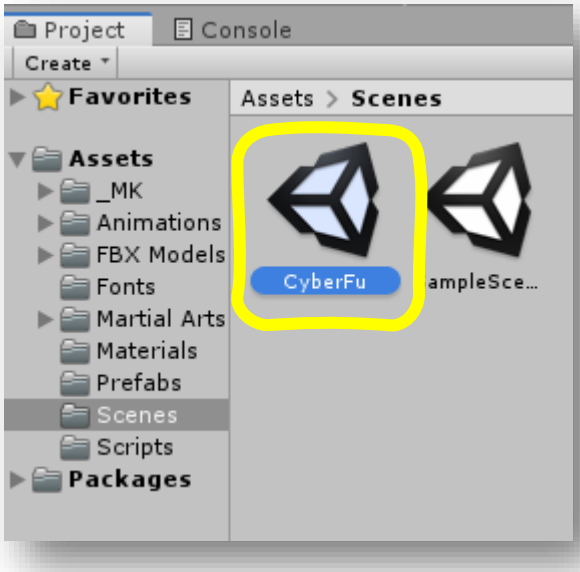
2 After **Unity** loads your new project, **import** the **Activity 10 - CyberFu P2** Ninja Unity Package by clicking on **Assets** -> **Import Package** -> **Custom Package**.



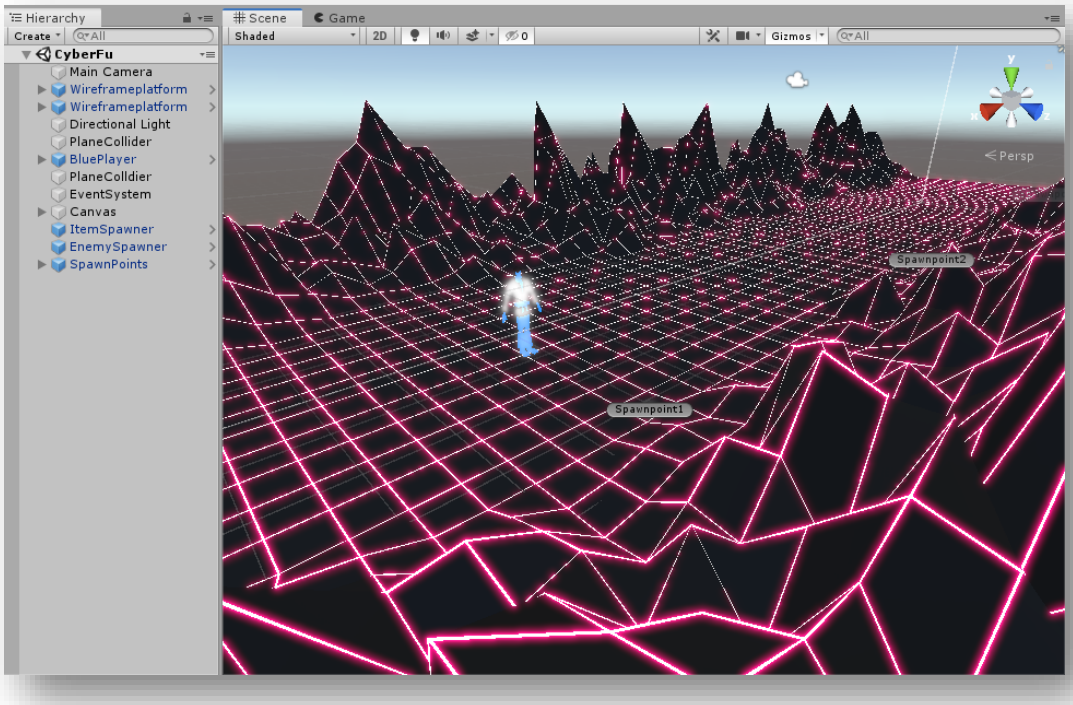
3 On the **Import Unity Package** window, click "**All**" and then "**Import**".



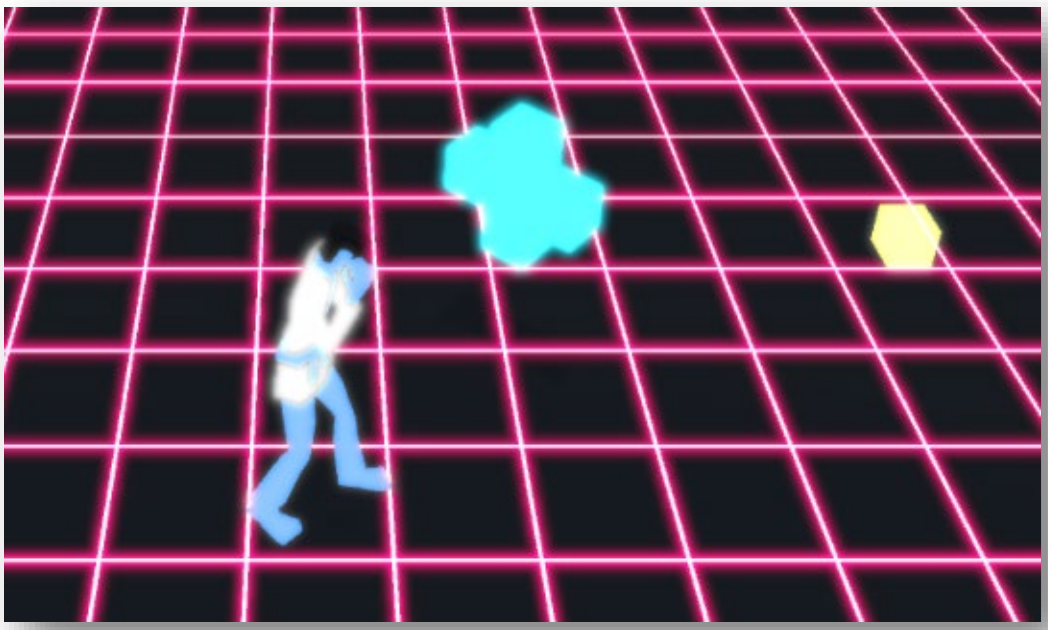

4 After it is done importing, double click the **CyberFu** scene located in **Project** -> **Scenes** -> **Assets**.



5 This will load the **scene** with all our game objects! Make sure can see all the objects and the assets in the **Hierarchy**.

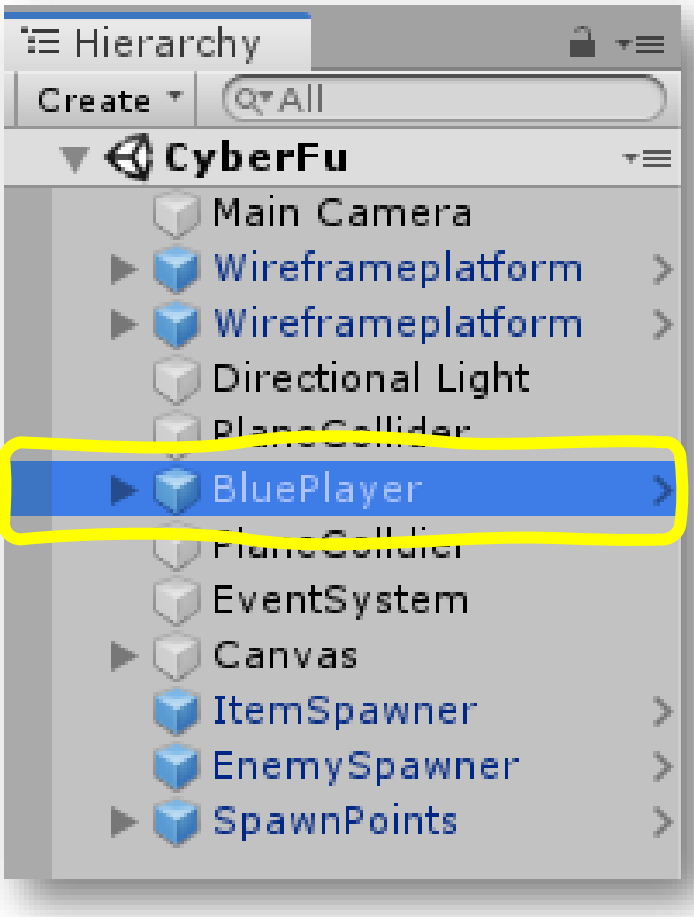



**Play** your game. What happens **touches** a blue or yellow **pickup**? What happens if an **enemy touches** a **pickup**?

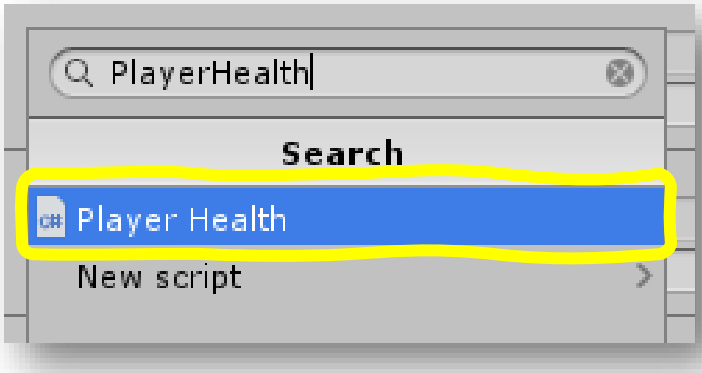


6 One of the **assets** that we **imported** was the **PlayerHealth** script. We need to add this to the **BluePlayer** game object.

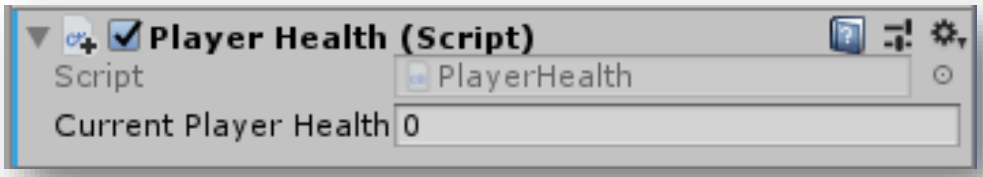
In the **Hierarchy**, click on **BluePlayer**.




7 Click the Add Component button. Search for **"PlayerHealth"** and select the **Player Health** result.



**8** The **BluePlayer** game object now has a **Player Health (Script)** component.




**9** Double click the **script** to open it in **Visual Studio**.

**10** We need to create **public variables** to define the **properties** of the player's health.

There is already a variable to store the max health and current health of the player. Its initial value is set to `10`.

```
public class PlayerHealth : MonoBehaviour
{
    public int maxPlayerHealth = 10;
    public int currentPlayerHealth;

    Unity Message | 0 references
    void Start()
    {
        currentPlayerHealth = maxPlayerHealth;
    }
```

**11** After the `public int currentPlayerHealth;` line, type `public int enemyDamage = 2;` to set how much damage each enemy hit causes.

```
public class PlayerHealth : MonoBehaviour
{
    public int maxPlayerHealth = 10;
    public int enemyDamage = 2;
```

You can adjust this value to make the game easier or harder!

**12** Now that we have the **variables** that store the player's health and the enemy's damage, we need to create a **function** that will help us change the **values** of the **variables**.

After the **Start** function, create a new **function** called **HurtPlayer** by typing `public void HurtPlayer() { }` making sure to leave an empty line between the curly brackets.

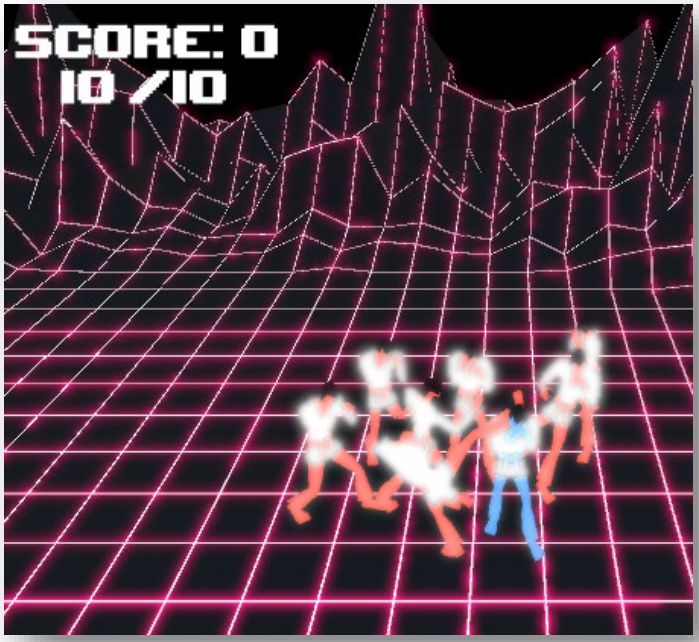
```
public void HurtPlayer()
{

}
```

**13** Every time we call the **HurtPlayer** function we want to **subtract** the **enemy's damage** from the **player's current health**.

Type `currentPlayerHealth -= enemyDamage;` inside of the **body** of the function.

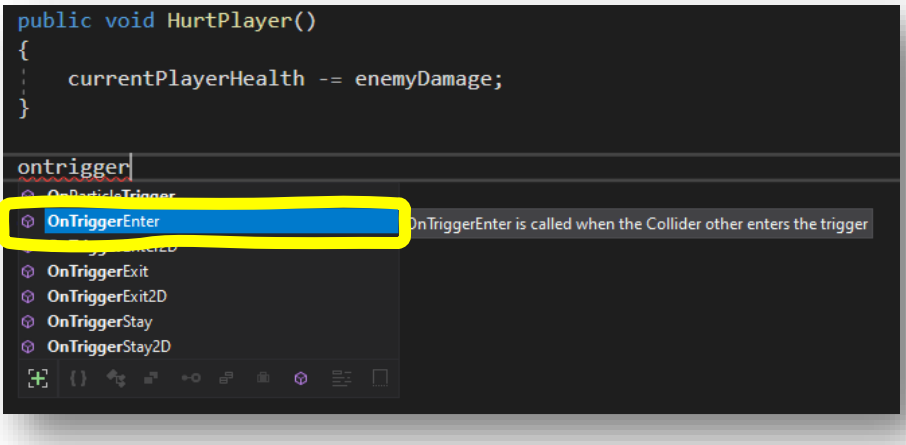
```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
}
```

**14** **Play** your game. The player's health never goes down, no matter how many enemies are attacking! Talk with a **Code Sensei** about why the player is not responding to enemy attacks.

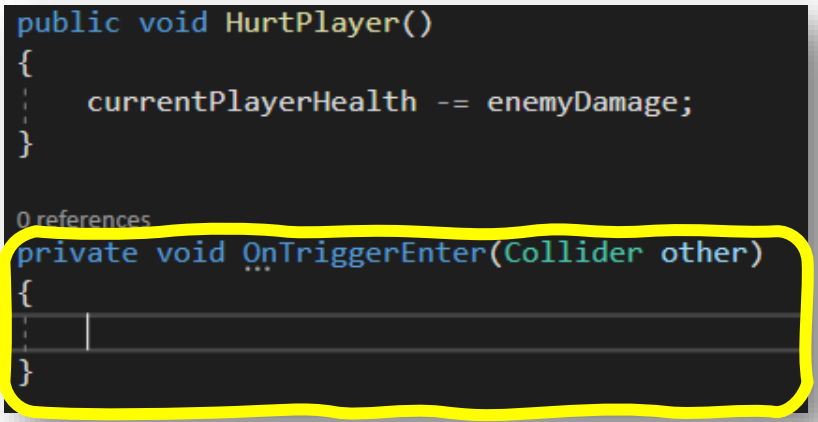



**15** Our player knows how to respond to an enemy attack, it just doesn't know when it should!

After the **HurtPlayer function**, start typing "**ontrigger**" then **Visual Studio** might suggest a list of functions. You can select "**OnTriggerEnter**" and have **Visual Studio** auto-complete the function for you.

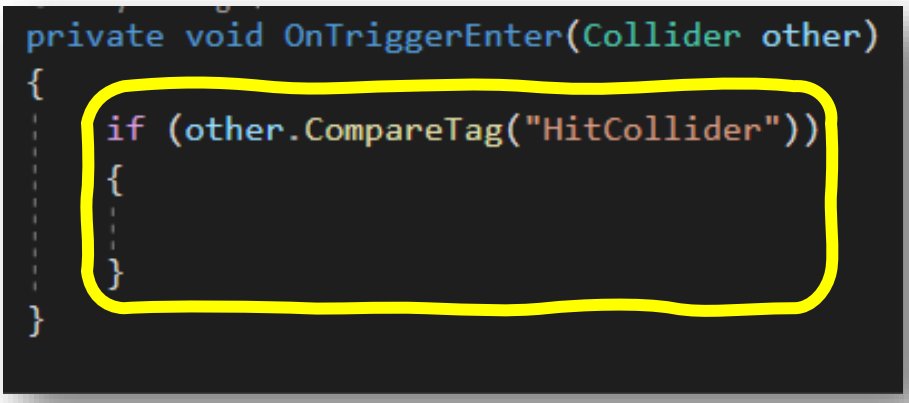


**16** Alternatively, you can type the entire function out yourself. Type `private void OnTriggerEnter(Collider other) { }` making sure to leave an empty line between the two curly brackets.




Unity will run the code inside of the **OnTriggerEnter** function **every time** the player **collides** with a **trigger collider**.

**17** We want to run our **HurtPlayer** function **if** the player collides with another trigger with the special "**HitCollider**" **tag**. Inside the **OnTriggerEnter** function, check to see of the **other** object has a **tag equal** to "**HitCollider**" by typing `if (other.CompareTag("HitCollider")) { }` making sure to leave a line in between the curly brackets.



## 18

Inside of our **if** statement, we want to run our **HurtPlayer** function. Type `HurtPlayer();` to have the player run the function **every time** the enemy attacks and **collides** with the player.

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("HitCollider"))
    {
        HurtPlayer();
    }
}
```

## 19

**Play** your game. The player's health now correctly responds when the enemies attack.


Talk to a **Code Sensei** and **explain** how the player knows if the enemy's attack was successful or not.

## 20

We can add an animation for when the player gets hit. We need to control when the player **animates**, so we need to get a reference to its **Animator** component.

After the `public int enemyDamage = 2;` line, type `private Animator playerAnimator;` to set up the animator variable that only this script knows about.

```
public class PlayerHealth : MonoBehaviour
{
    public int maxPlayerHealth = 10;
    public int currentPlayerHealth;
    public int enemyDamage = 2;

    private Animator playerAnimator;
```

## 21

In the Start function after the `currentPlayerHealth = 10;` line, type `playerAnimator = GetComponent<Animator>();` to find the **BluePlayer**'s **Animator** component and store it in the `playerAnimator` variable.

```
void Start()
{
    playerAnimator = GetComponent<Animator>();
}
```

## 22

Now that we can use the player's **animator**, we need to tell it to play the "**Hit**" animation whenever the player receives an enemy's attack.

Talk with a **Code Sensei** about **where** this code should go. It should be wherever the player **responds** to being hit by an enemy.

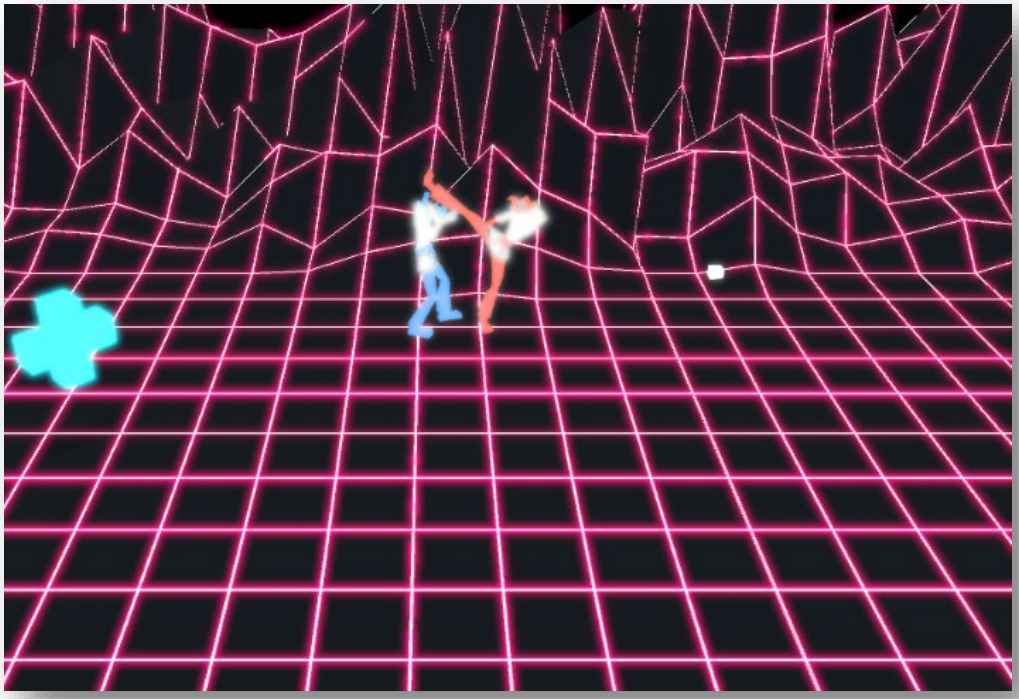
## 23

Inside the **HurtPlayer** function and after the `currentPlayerHealth -= enemyDamage;` line, type `playerAnimator.SetTrigger("Hit");` to tell the player's **animator** to **play** the "**Hit**" animation.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");
}
```

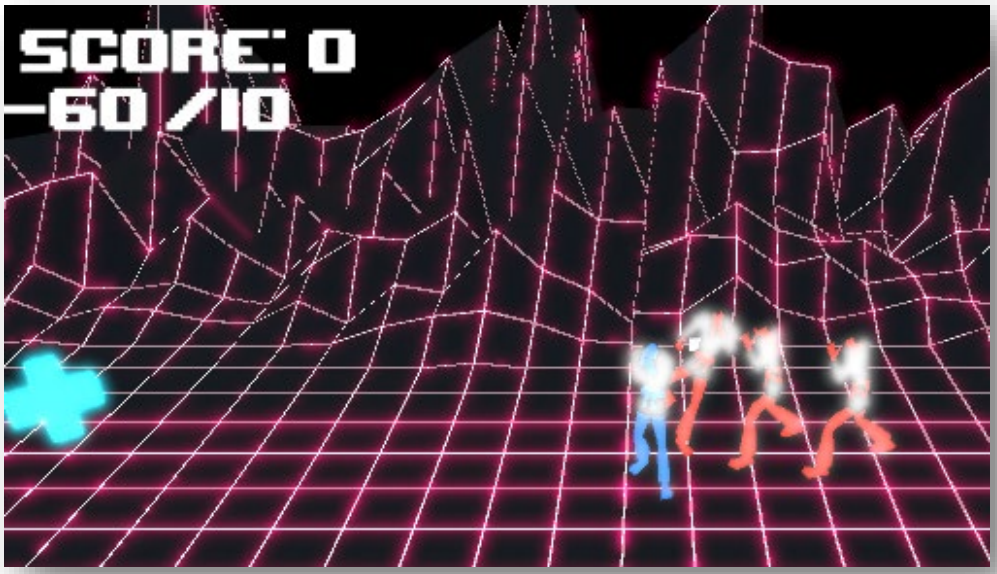
## 24

**Play** your game. The **player** is now **animating every time** he gets **hit**!



## 25

In your **playtests**, did you ever notice what happened when the player's **health** reached **zero**? The game keeps playing and the **value** of **health** becomes **negative**!




## 26

We need to code what happens when the player runs out of health. At the end of the **HurtPlayer** function, we need to check to see if the **value** of the player's health **is less than or equal to zero** and then disable the player.

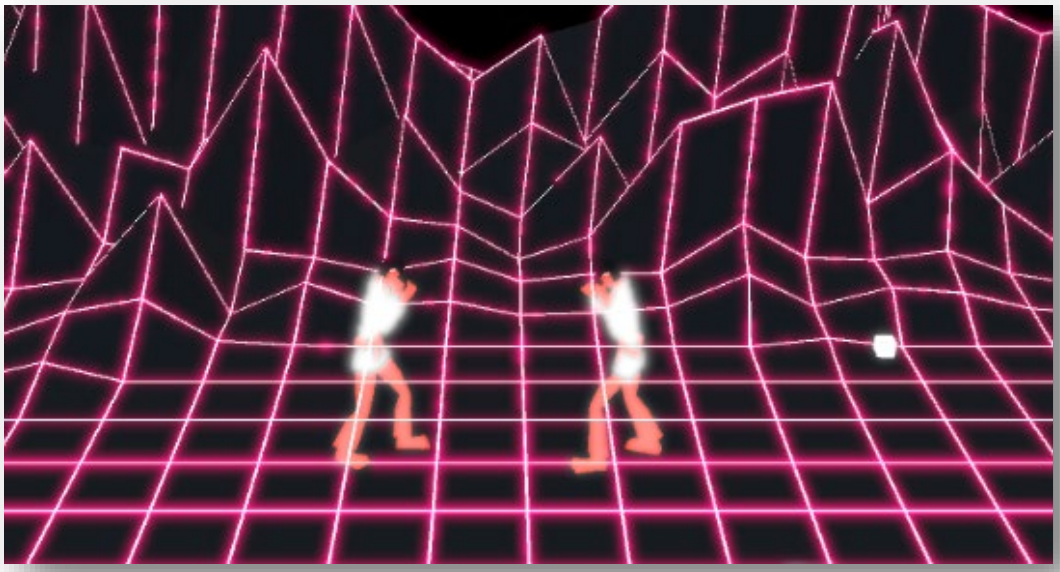
After the `playerAnimator.SetTrigger("Hit");` line, type `if (currentPlayerHealth <= 0) { }` making sure to leave an empty line between the curly brackets.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {

    }
}
```

**27** We want to **disable** the player if his life is **zero or less**, so inside the if statement type `gameObject.SetActive(false);`

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {
        gameObject.SetActive(false);
    }
}
```

**28** Play your game. What happens now when the player is defeated?



**29** We want to give the player another chance by **reloading** the **scene** if they lose. Create a new function after the **HurtPlayer** function called **ReloadScene** by typing `private void ReloadScene() { }` making sure to leave a blank line between the curly brackets.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {
        gameObject.SetActive(false);
    }
}

0 references
private void ReloadScene()
{

}
```

**30** In this function we want to ask Unity's **SceneManager** to reload the CyberFu scene by typing `SceneManager.LoadScene("CyberFu");` inside the body of the function.

```
private void ReloadScene()
{
    SceneManager.LoadScene("CyberFu");
}
```

**31** We need to run this **function** when the player runs out of health. In the `HurtPlayer` function after the `gameObject.SetActive(false);` line, type `ReloadScene();` to call the **function** and reload the **scene**.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {
        gameObject.SetActive(false);
        ReloadScene();
    }
}
```

**32** Play your game and test what happens when the player runs out of health. The game instantly reloads!

**33** We can use the special Unity **function** named `Invoke` to **delay** running our `ReloadScene` function.

Delete the `ReloadScene();` line and replace it with `Invoke("ReloadScene", 5);` to ask Unity to call the `ReloadScene` function `5` five seconds after the **player** runs out of **health**.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {
        gameObject.SetActive(false);
        Invoke("ReloadScene", 5);
    }
}
```

**34** **Play** your game.

The game should now properly reload when the player is defeated.

**35** We can make the player's defeat a little more dramatic! After the `public int enemyDamage = 2;` line, type `public PlayerExplosionParticles particles;` to create a **variable** for the explosion script.

```
public int currentPlayerHealth = 10;
public int enemyDamage = 2;

public PlayerExplosionParticles particles;

private Animator playerAnimator;
```

**36** In the Start function type `particles = GetComponent<PlayerExplosionParticles>();` to get the player's **PlayerExplosionParticles component** and store it in the `particles` **variable**.

```
Unity Message | 0 references
void Start()
{
    currentPlayerHealth = maxPlayerHealth;
    playerAnimator = GetComponent<Animator>();
    particles = GetComponent<PlayerExplosionParticles>();
}
```

**37** In the **HurtPlayer**'s **if statement**, remove the `gameObject.SetActive(false);` line and replace it with `particles.Explode();`.

```
public void HurtPlayer()
{
    currentPlayerHealth -= enemyDamage;
    playerAnimator.SetTrigger("Hit");

    if (currentPlayerHealth <= 0)
    {
        particles.Explode();
        Invoke("ReloadScene", 5);
    }
}
```

**38** **Play** your game. Now when the player runs out of **health**, there's an explosion of particles!