Unit 09b: Turret Health

Introduction

We've got a turret that senses the player, and reacts by shooting at it. But it's unstoppable! Let's make it take damage and be destroyed.

Goal

To add health and death to the turret.

Process

> With our `Health` script, we planned on being able to drop it onto something and have it work. Let's see how we did.

1. Find the `Health` script in your Project, and drag it onto the `Turret` gameObject.

> Test it in the game. Make sure the `Turret` is selected, so we can watch if the health successfully goes down.
>
> OH NOES! Chances are good it *doesn't* work. Let's see why not.
>
> If you look at your `Turret`, the gameObject hierarchy should look something like:
>
> - Turret (the root gameObject, with the scripts and a trigger)
>   - Cylinder (the body of the turret, with a collider)
>   - Cube (the barrel, with a collider)
>   - Nozzle (an empty gameObject)
>
> The issue with our health script is that our PlayerBullet objects are technically hitting the Cylinder and Cube – the gameObjects with the **Colliders**. Which is how it *should* work, as we don't want things to collide that do not have a Collider. The problem is that in our `PlayerBullet` code, we use this line to check for an attached `Health`:
>
> `C# if (other.gameObject.TryGetComponent(out Health health))`

> So when the bullet collides with the Cylinder, it checks if the *Cylinder* has a `Health` script – which it doesn't! The Cylinder's *parent* does! So that `if` statement fails. Bummer.
>
> Let's fix it. Instead of checking the object we collide with, we want to check it's parent. But wait! What if we hit the Cube? It's parent is the Cylinder. Neither have the `Health` script. So *just checking the parent won't work*. We want to check the top-most gameObject in this particular entity – the Turret object. Luckily, Unity gives us a way to do it.

1. In your `PlayerBullet` code, change this line:

```
if (other.gameObject.TryGetComponent(out Health health))
```

to this:

```
if (other.transform.root.gameObject.TryGetComponent(out Health health))
```

> The `transform.root` addition traverses up this object's hierarchy until it reaches the top, which is exactly what we're after.
>
> Test again, and we should be able to do some damage now!
>
> This kind of edit/refactor is quite common in game dev, where you initially make it work for one situation, and then refactor to make it work for multiplie situations.
>
> Our next issue is that when its health drops below zero, nothing happens. Let's make a new `Death` script for the Turret.

1. Create a new `TurretDeath` script on the top level of the `Turret` object, and open it in the editor.
2. Change the class declaration:

```
public class TurretDeath : Death
```

> If you rememeber from creating our `PlayerDeath` script, the one method we *need* to create is the `HandleDeath` method.

1. Add the following method:

```
    public override void HandleDeath()
    {
        Destroy(gameObject);
    }
```

> Now when you attack the turret, you can destroy it!
>
> When we get to special effects, we'll be able to add an explosion. Fun!

Complete Code

```
public class TurretDeath : Death
{
    public override void HandleDeath()
    {
        Destroy(gameObject);
    }
}
```

Wrap-Up

Further Material