Unit 05g: Spawning

Introduction

In the Health unit, we introduced the idea of a health variable, and a death script for objects that just need to be destroyed. This unit works through a respawn script for the player.

Goal

The goal of this unit is to make a respawn function for the player.

Process

1. First thing, let's make a zone that causes damage to the player. In your scene, create a new empty GameObject and call it "PainZone". Move it to `-4, 0, 0`.
2. Add a BoxCollider to the PainZone. Set the center to `0, 0.5, 0`, and the size to `2, 1, 2`.
3. Set the BoxCollider to be `IsTrigger`.

> Setting a collider to Trigger makes it so it doesn't stop anything, but still fires off a notification.

1. On the PainZone, create a new script called `PainZone`. Open it in the editor. We're going to start with this, much like the bullet. Note, however, that because it's a Trigger (and not a Collider), we need to use a slightly different method:

```
public class PainZone : MonoBehaviour
{
    public int damage = 7;

    public void OnTriggerEnter(Collider other)
    {
        Debug.Log("Pain!");
    }
}
```

> When you test, you should get the debug message when you drive through the zone.

1. And then we can tie in the Health script:

```
    public void OnTriggerEnter(Collider other)
```

```
    {
        // Debug.Log("Pain!");
        if (other.attachedRigidbody.gameObject.TryGetComponent(out Health health))
        {
            health.Damage(damage);
        }
    }
```

> Note the extra `attachedRigidBody` call. This is because the Collider is on the `Cube` object, but the `Health` script is on the `Player` object. We need to traverse the object hierarchy to get to the Player object, and using `attachedRigidBody` is an easy way of getting there.
>
> Now when you test, you should take damage when you enter the PainZone.

1. Next, we're going to set up the Player's `PlayerDeath` script. Create a new script on the Player, and add this code:

```
public class PlayerDeath : Death
{
    public override void HandleDeath()
    {
        Debug.Log("Player has died!");
    }
}
```

> Do a quick test to make sure you're getting the debug after entering your PainZone.
>
> The functionality we're looking for when the player dies is for the player to return to the spawn point.

1. To do this, we need to set up a couple of things. Add the following class variables:

```
public class PlayerDeath : Death
{
    public Vector3 spawnLocation;
    public Quaternion spawnRotation;
```

This will be the point we return to after dying. You can specify this location in the editor, or you can preset it with the player's initial location using the following `Start` callback:

```
void Start()
    {
        spawnLocation = transform.position;
        spawnRotation = transform.rotation;
    }
```

Which sets those variables when the game starts.

1. Next, we need to return the player to that point on dying:

```
    public override void HandleDeath()
```

```
    {
        Debug.Log("Player has died!");
        GetComponent<Rigidbody>().position = spawnLocation;
        GetComponent<Rigidbody>().rotation = spawnRotation;
    }
```

> Easy as, right? Not quite! If you're using a rigidbody, just setting the position and location doesn't remove any impulse or inertia the player currently has. So if it dies moving quickly in one direction, upon respawn it'll keep moving in that direction.

1. To remove any existing movement, we can use the following:

```
public override void HandleDeath()
{
    // Debug.Log("Player has died!");
    GetComponent<Rigidbody>().position = spawnLocation;
    GetComponent<Rigidbody>().rotation = spawnRotation;

    GetComponent<Rigidbody>().velocity = Vector3.zero;
    GetComponent<Rigidbody>().angularVelocity = Vector3.zero;
}
```

> Next steps would be to set up a SpawnPoint gameObject, that replaces this value when the player steps into it. You can hack together concepts from the PainZone and the PlayerBullet to figure out how you might do that...

Wrap-Up

Further Material