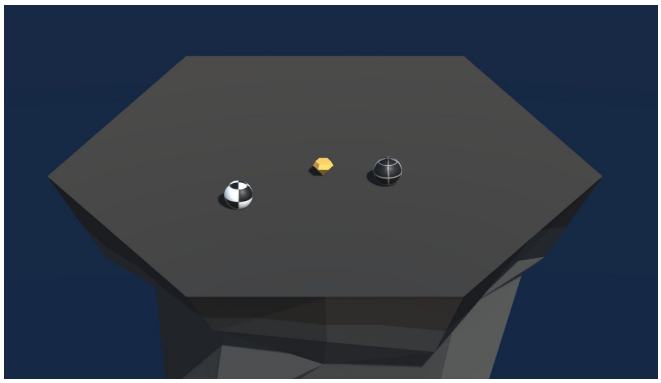
unity

4.3 PowerUp and CountDown

Steps:

Example of project by end of lesson



Length: 60 minutes

Overview: The enemy chases the player around the island, but the player needs a better way to defend themselves... especially if we add more enemies. In this lesson, we're going to create a powerup that gives the player a temporary strength boost, shoving away enemies that come into contact! The powerup will spawn in a random position on the island, and highlight the player with an indicator when it is picked up. The powerup indicator and the powerup itself will be represented by stylish game assets of your choice.

Project Outcome: A powerup will spawn in a random position on the map. Once the player collides with this powerup, the powerup will disappear and the player will be highlighted by an indicator. The powerup will last for a certain number of seconds after pickup, granting the player super strength that blasts away enemies.

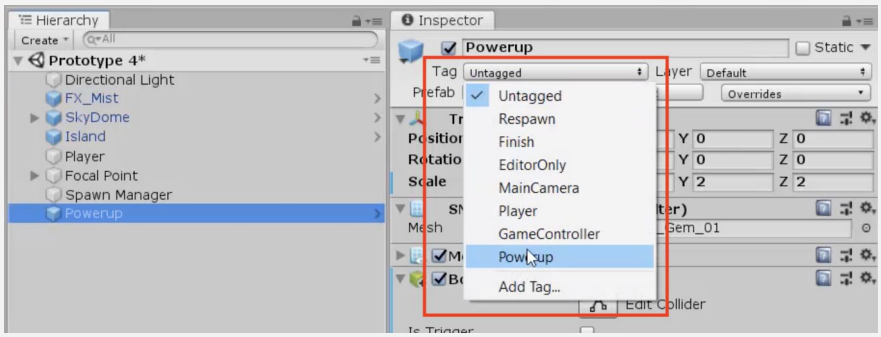
Learning Objectives: By the end of this lesson, you will be able to:
- Write informative debug messages with Concatenation and variables
- Repeat functions with the power of IEnumerator and Coroutines
- Use SetActive to make game objects appear and disappear from the scene

Step 1: Choose and prepare a powerup

In order to add a completely new gameplay mechanic to this project, we will introduce a new powerup object that will give the player temporary superpowers.

1. From the *Library*, drag a **Powerup** object into the scene, rename it "Powerup" and edit its **scale** & **position**
2. Add a **Box Collider** to the powerup, click **Edit Collider** to make sure it fits, then check the "**Is Trigger**" checkbox
3. Create a new "Powerup" **tag** and apply it to the **powerup**
4. Drag the **Powerup** into the **Prefabs folder** to create a new "Original Prefab"

- **Warning:** Remember, you still have to apply the tag after it has been created.



Step 2: Destroy powerup on collision

As a first step to getting the powerup working, we'll make it disappear when the player hits it and set up a new boolean variable to track that the player got it.

1. In **PlayerController.cs**, add a **new *OnTriggerEnter()*** method
2. Add an **if-statement** that destroys ***other.CompareTag("Powerup")*** **powerup** on collision
3. Create a new ***public bool hasPowerup;*** and set ***hasPowerup = true;*** when you **collide** with the Powerup

- **Don't worry:** If this doesn't work, make sure that the Powerup's collider "Is trigger" and player's collider is NOT
- **Tip:** Make sure hasPowerup = true in the inspector when you collide

```
public bool hasPowerup

private void OnTriggerEnter(Collider other) {
  if (other.CompareTag("Powerup")) {
    hasPowerup = true;
    Destroy(other.gameObject); } }
```

Step 3: Test for enemy and powerup

The powerup will only come into play in a very particular circumstance: when the player has a powerup AND they collide with an enemy - so we'll first test for that very specific condition.

1. Create a new "Enemy" tag and apply it to the **Enemy Prefab**
2. In **PlayerController.cs**, add the ***OnCollisionEnter()*** function
3. Create the **if-statement** with the **double-condition** testing for enemy tag and hasPowerup boolean
4. Create a **Debug.Log** to make sure it's working

- **Tip:** OnTriggerEnter is good for stuff like picking up powerups, but you should use OnCollisionEnter when you want something to do with physics
- **New Concept:** Concatenation in Debug messages
- **Tip:** When you concatenate a variable in a debug message, it will returns its VALUE not its name

```
private void OnCollisionEnter(Collision collision) {
  if (collision.gameObject.CompareTag("Enemy") && hasPowerup) {
    Debug.Log("Player collided with " + collision.gameObject
    + " with powerup set to " + hasPowerup); } }
```

Step 4: Apply extra knockback with powerup

With the condition for the powerup set up perfectly, we are now ready to program the actual powerup ability: when the player collides with an enemy, the enemy should go flying!

1. In ***OnCollisionEnter()*** declare a new **local variable** to get the Enemy's **Rigidbody** component
2. Declare a new variable to get the **direction** away from the **player**
3. Add an **impulse force** to the **enemy**, using a new **powerupStrength** variable

- **Tip:** Reference the code in Enemy.cs that makes the enemy follow the player. In a way, we're reversing that code in order to push the enemy away.
- **Don't worry:** No need to use .Normalize, since they're colliding

```
private float powerupStrength = 15.0f;

private void OnCollisionEnter(Collision collision) {
  if (collision.gameObject.CompareTag("Enemy") && hasPowerup) {

    Rigidbody enemyRigidbody = collision.gameObject.GetComponent<Rigidbody>();
    Vector3 awayFromPlayer = (collision.gameObject.transform.position
    - transform.position);

    Debug.Log("Player collided with " + collision.gameObject
    + " with powerup set to " + hasPowerup);
    enemyRigidbody.AddForce(awayFromPlayer * powerupStrength,
    ForceMode.Impulse); } }
```

Step 5: Create Countdown Routine for powerup

It wouldn't be fair to the enemies if the powerup lasted forever - so we'll program a countdown timer that starts when the player collects the powerup, removing the powerup ability when the timer is finished.

1. Add a new ***IEnumerator PowerupCountdownRoutine () {}***
2. Inside the ***PowerupCountdownRoutine***, wait 7 seconds, then **disable** the powerup
3. When player **collides** with powerup, start the **coroutine**

- **New Concept:** IEnumerator
- **New Concept:** Coroutines
- **Tip:** WaitForSeconds()

```
private void OnTriggerEnter(Collider other) {
  if (other.CompareTag("Powerup")) {
    hasPowerup = true;
    Destroy(other.gameObject);
    StartCoroutine(PowerupCountdownRoutine()); } }

IEnumerator PowerupCountdownRoutine() {
  yield return new WaitForSeconds(7); hasPowerup = false; }
```

Step 6: Add a powerup indicator

To make this game a lot more playable, it should be clear when the player does or does not have the powerup, so we'll program a visual indicator to display this to the user.

1. From the *Library*, drag a **Powerup object** into the scene, rename it "Powerup Indicator", and edit its **scale**
2. **Uncheck** the "**Active**" checkbox in the inspector
3. In **PlayerController.cs**, declare a new ***public GameObject powerupIndicator*** variable, then assign the **Powerup Indicator** variable in the inspector
4. When the player collides with the powerup, set the indicator object to **Active**, then set to **Inactive** when the powerup expires
5. In ***Update()***, set the Indicator position to the player's position + an **offset value**

- **New Function:** SetActive
- **Tip:** Make sure the indicator is turning on and off before making it follow the player

```
public GameObject powerupIndicator

void Update() {
  ... powerupIndicator.transform.position = transform.position
  + new Vector3(0, -0.5f, 0); }

private void OnTriggerEnter(Collider other) {
  if (other.CompareTag("Powerup")) {
    ... powerupIndicator.gameObject.SetActive(true); } }

IEnumerator PowerupCountdownRoutine() {
  ... powerupIndicator.gameObject.SetActive(false); }
```

Lesson Recap

New Functionality	• When the player collects a powerup, a visual indicator appears • When the player collides with an enemy while they have the powerup, the enemy goes flying • After a certain amount of time, the powerup ability and indicator disappear
New Concepts and Skills	• *Debug concatenation* • *Local component variables* • *IEnumerators and WaitForSeconds()* • *Coroutines* • SetActive(true/false)
Next Lesson	• We'll start generating waves of enemies for our player to fend off!