4.2 Follow the Player

Steps:

Step 1: Add an enemy and a physics material

Step 2: Create enemy script to follow player

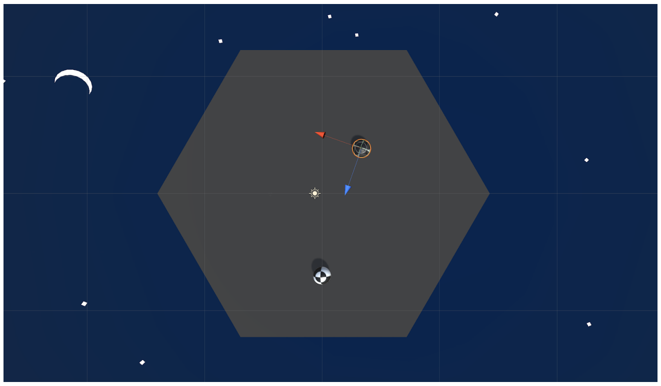
Step 3: Create a lookDirection variable

Step 4: Create a Spawn Manager for the enemy

Step 5: Randomly generate spawn position

Step 6: Make a method return a spawn point

Example of project by end of lesson



Length: 60 minutes

Overview: The player can roll around to its heart's content… but it has no purpose. In this lesson, we fill that purpose by creating an enemy to challenge the player! First we will give the enemy a texture of your choice, then give it the ability to bounce the player away... potentially knocking them off the cliff. Lastly, we will let the enemy chase the player around the island and spawn in random positions.

Project Outcome: A textured and spherical enemy will spawn on the island at start, in a random location determined by a custom function. It will chase the player around the island, bouncing them off the edge if they get too close.

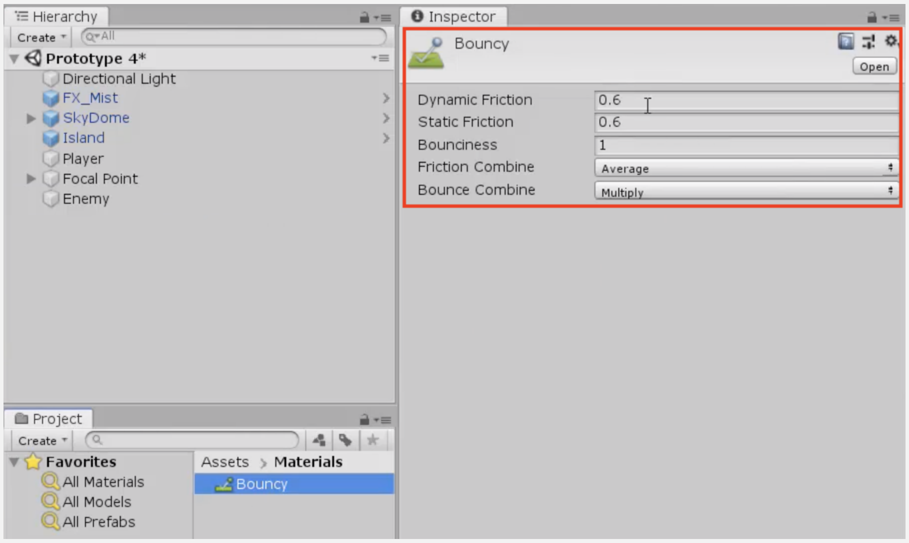
Learning Objectives: By the end of this lesson, you will be able to:
- Apply Physics Materials to make game objects bouncy
- Normalize vectors to point the enemy in the direction of the player
- Randomly spawn with Random.Range on two axes
- Write more advanced custom functions and variables to make your code clean and professional

Step 1: Add an enemy and a physics material

Our camera rotation and player movement are working like a charm. Next we're going to set up an enemy and give them them some special new physics to bounce the player away!

1. Create a new **Sphere**, rename it "Enemy" reposition it, and drag a **texture** onto it
2. Add a new **RigidBody** component and adjust its XYZ **scale**, then test
3. In a new "Physics Materials**"** folder, *Create > Physics Material,* then name it "Bouncy"
4. Increase the **Bounciness** to "1", change **Bounce Combine** to "Multiply", apply it to your player and enemy, then **test**

- **Don't worry:** If your game is lagging, uncheck the "Active" checkbox for your clouds
- **New Concept:** Physics Materials
- **New Concept:** Bounciness property and Bounce Combine



Bouncy

Dynamic Friction	0.6
Static Friction	0.6
Bounciness	1
Friction Combine	Average
Bounce Combine	Multiply

Step 2: Create enemy script to follow player

The enemy has the power to bounce the player away, but only if the player approaches it. We must tell the enemy to follow the player's position, chasing them around the island.

1. Make a new “Enemy**”** script and attach it to the **Enemy**
2. Declare 3 new variables for ***Rigidbody enemyRb;***, ***GameObject player;***, and ***public float speed;***
3. Initialize ***enemyRb = GetComponent Rigidbody>();*** and ***player = GameObject.Find("Player");***
4. In ***Update()***, AddForce towards in the direction between the Player and the Enemy

- **Tip:** Imagine we're generating this new vector by drawing an arrow from the enemy to the player.
- **Tip:** We should start thinking ahead and writing our variables in advance. Think… what are you going to need?
- **Tip:** When normalized, a vector keeps the same direction but its length is 1.0, forcing the enemy to try and keep up

```
public float speed = 3.0f;
private Rigidbody enemyRb;
private GameObject player;

void Update() {
  enemyRb.AddForce((player.transform.position
  - transform.position).normalized * speed); }
```

Step 3: Create a lookDirection variable

The enemy is now rolling towards the player, but our code is a bit messy. Let's clean up by adding a variable for the new vector.

1. In ***Update()***, declare a new ***Vector3 lookDirection*** variable
2. Set ***Vector3 lookDirection = (player.transform.position - transform.position).normalized;***
3. Implement the ***lookDirection*** variable in the ***AddForce*** call

- **Tip:** As always, adding variables makes the code more readable

```
void Update() {
  Vector3 lookDirection = (player.transform.position
  - transform.position).normalized;

  enemyRb.AddForce(lookDirection ~~(player.transform.position
  - transform.position).normalized~~ * speed); }
```

Step 4: Create a Spawn Manager for the enemy

Now that the enemy is acting exactly how we want, we're going to turn it into a prefab so it can be instantiated by a Spawn Manager.

1. Drag **Enemy** into the Prefabs folder to create a new **Prefab**, then delete **Enemy** from scene
2. Create a new "Spawn Manager" **object**, attach a new "SpawnManager" **script**, and open it
3. Declare a new ***public GameObject enemyPrefab*** variable then assign the prefab in the **inspector**
4. In ***Start()***, instantiate a new ***enemyPrefab*** at a predetermined location

```
public GameObject enemyPrefab;

void Start()
{
  Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation); }
```

Step 5: Randomly generate spawn position

The enemy spawns at start, but it always appears in the same spot. Using the familiar Random class, we can spawn the enemy in a random position.

1. In SpawnManager.cs, in ***Start()***, create new **randomly generated** X and Z
2. Create a new ***Vector3 randomPos*** variable with those random X and Z positions
3. Incorporate the new ***randomPos*** variable into the ***Instantiate*** call
4. Replace the hard-coded **values** with a ***spawnRange*** variable
5. **Start** and **Restart** your project to make sure it's working

- **Tip:** Remember, we used Random.Range all the way back in Unit 2! Feel free to reference old code.

```
public GameObject enemyPrefab;
private float spawnRange = 9;

void Start() {
  float spawnPosX = Random.Range(~~-9, 9~~ -spawnRange, spawnRange);
  float spawnPosZ = Random.Range(~~-9, 9~~ -spawnRange, spawnRange);
  Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
  Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation); }
```

Step 6: Make a method return a spawn point

The code we use to generate a random spawn position is perfect, and we're going to be using it a lot. If we want to clean the script and use this code later down the road, we should store it in a custom function.

1. Create a new function ***Vector3 GenerateSpawnPosition() { }***
2. Copy and Paste the ***spawnPosX*** and ***spawnPosZ*** variables into the new method
3. Add the line to ***return randomPos;*** in your new method
4. Replace the code in your **Instantiate call** with your new function name: ***GenerateSpawnPosition()***

- **Tip:** This function will come in handy later, once we randomize a spawn position for the powerup
- **New Concept:** Functions that return a value
- **Tip:** This function is different from "void" calls, which do not return a value. Look at "GetAxis" in PlayerController for example - it returns a float

```
void Start() {
  Instantiate(enemyPrefab, GenerateSpawnPosition()
  ~~new Vector3(spawnPosX, 0, spawnPosZ)~~, enemyPrefab.transform.rotation);
  ~~float spawnPosX = Random.Range(-spawnRange, spawnRange);~~
  ~~float spawnPosZ = Random.Range(-spawnRange, spawnRange);~~ }

private Vector3 GenerateSpawnPosition () {
  float spawnPosX = Random.Range(-spawnRange, spawnRange);
  float spawnPosZ = Random.Range(-spawnRange, spawnRange);
  Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
  return randomPos; }
```

Lesson Recap

New Functionality	● Enemy spawns at random location on the island ● Enemy follows the player around ● Spheres bounce off of each other
New Concepts and Skills	● Physics Materials ● Defining vectors in 3D space ● Normalizing values ● Methods with return values
Next Lesson	● In our next lesson, we'll create ways to fight back against these enemies using Powerups!