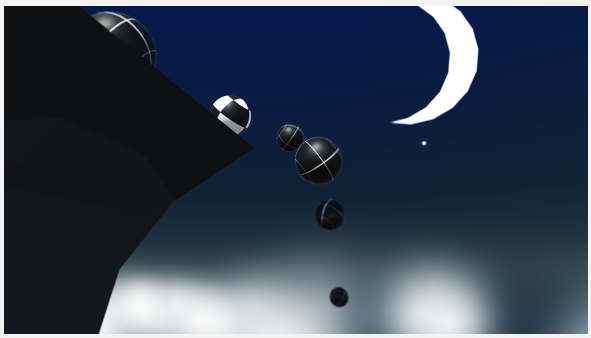
Create with Code

Unit 4 Lesson Plans



unity

4.1 Watch Where You're Going

Steps:

Step 1: Create project and open scene

Step 2: Set up the player and add a texture

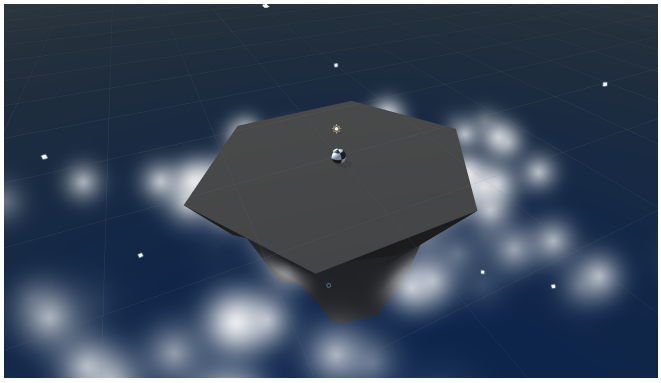
Step 3: Create a focal point for the camera

Step 4: Rotate the focal point by user input

Step 5: Add forward force to the player

Step 6: Move in direction of focal point

Example of project by end of lesson



Length: 60 minutes

Overview: First thing's first, we will create a new prototype and download the starter files! You'll notice a beautiful island, sky, and particle effect... all of which can be customized! Next you will allow the player to rotate the camera around the island in a perfect radius, providing a glorious view of the scene. The player will be represented by a sphere, wrapped in a detailed texture of your choice. Finally you will add force to the player, allowing them to move forwards or backwards in the direction of the camera.

Project Outcome: The camera will evenly rotate around a focal point in the center of the island, provided a horizontal input from the player. The player will control a textured sphere, and move them forwards or backwards in the direction of the camera's focal point.

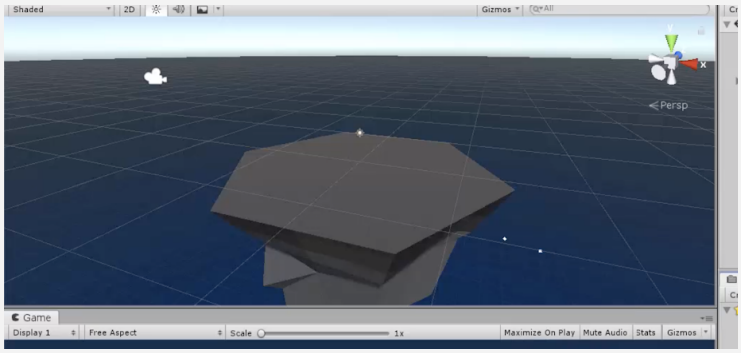
Learning Objectives: By the end of this lesson, you will be able to:
- Apply Texture wraps to objects
- Attach a camera to its focal point using parent-child relationships
- Transform objects based on local XYZ values

Step 1: Create project and open scene

You've done it before, and it's time to do it again... we must start a new project and import the starter files.

1. Open **Unity Hub** and create an empty "Prototype 4" project in your course directory on the correct Unity version.
   If you forget how to do this, refer to the instructions in Lesson 1.1 - Step 1
2. Click to download the Prototype 4 Starter Files, **extract** the compressed folder, and then **import** the .unitypackage into your project.
   If you forget how to do this, refer to the instructions in Lesson 1.1 - Step 2
3. Open the **Prototype 4 scene** and delete the **Sample Scene** without saving
4. Click **Run** to see the **particle effects**

- **Don't worry:** You can change texture of floating island and the color of the sky later
- **Don't worry:** We're in isometric/orthographic view for a reason: It just looks nicer when we rotate around the island

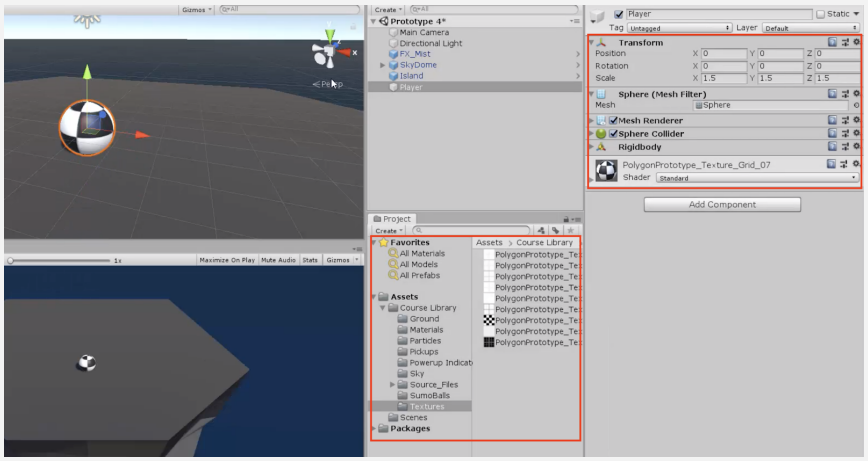


Step 2: Set up the player and add a texture

We've got an island for the game to take place on, and now we need a sphere for the player to control and roll around.

1. In the **Hierarchy**, *create 3D Object* > ***Sphere***
2. Rename it "Player", reset its **position** and increase its XYZ **scale** to 1.5
3. Add a **RigidBody** component to the **Player**
4. From the *Library* > *Textures*, drag a **texture** onto the **sphere**

- **New Concept:** Texture wraps

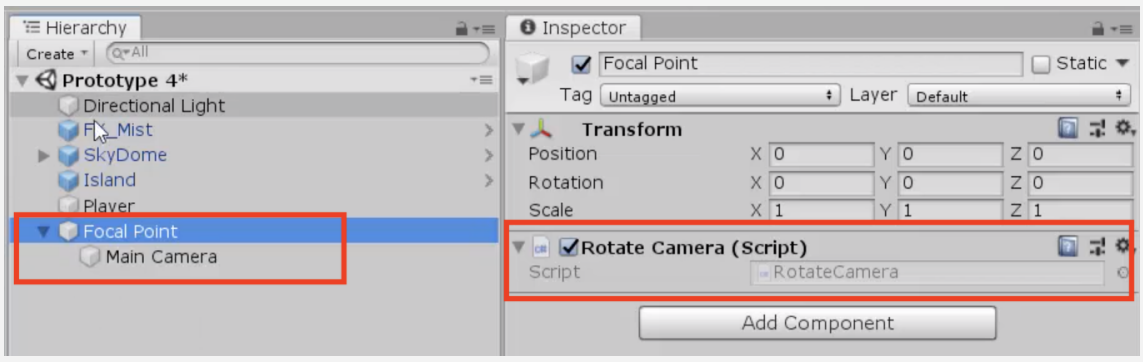


Step 3: Create a focal point for the camera

If we want the camera to rotate around the game in a smooth and cinematic fashion, we need to pin it to the center of the island with a focal point.

1. Create a new **Empty Object** and rename it "Focal Point",
2. Reset its position to the origin (0, 0, 0), and make the Camera a **child object** of it
3. Create a new "Scripts" folder, and a new "RotateCamera" script inside it
4. **Attach** the "RotateCamera" script to the **Focal Point**

- **Don't worry:** This whole "focal point" business may be confusing at first, but it will make sense once you see it in action
- **Tip:** Try rotating the Focal point around the Y axis and see the camera rotate in scene view



Step 4: Rotate the focal point by user input

Now that the camera is attached to the focal point, the player must be able to rotate it - and the camera child object - around the island with horizontal input.

1. Create the code to rotate the camera based on ***rotationSpeed*** and ***horizontalInput***
2. Tweak the ***rotation speed*** value to get the speed you want

- **Tip:** Horizontal input should be familiar, we used it all the way back in Unit 1! Feel free to reference your old code for guidance.

```
public float rotationSpeed;

void Update()
{
  float horizontalInput = Input.GetAxis("Horizontal");
  transform.Rotate(Vector3.up, horizontalInput * rotationSpeed * Time.deltaTime);
}
```

Step 5: Add forward force to the player

The camera is rotating perfectly around the island, but now we need to move the player.

1. Create a new "PlayerController" script, apply it to the **Player**, and open it
2. Declare a new ***public float speed*** variable and initialize it
3. Declare a new ***private Rigidbody playerRb*** and initialize it in ***Start()***
4. In ***Update()***, declare a new ***forwardInput*** variable based on **"Vertical"** input
5. Call the **AddForce()** method to move the player forward based ***forwardInput***

- **Tip:** Moving objects with RigidBody and Addforce should be familiar, we did it back in Unit 3! Feel free to reference old code.
- **Don't worry:** We don't have control over its direction yet - we'll get to that next

```
private Rigidbody playerRb;
public float speed = 5.0f;

void Start() {
  playerRb = GetComponent<Rigidbody>(); }

void Update() {
  float forwardInput = Input.GetAxis("Vertical");
  playerRb.AddForce(Vector3.forward * speed * forwardInput); }
```

Step 6: Move in direction of focal point

We've got the ball rolling, but it only goes forwards and backwards in a single direction! It should instead move in the direction the camera (and focal point) are facing.

1. Declare a new ***private GameObject focalPoint;*** and initialize it in ***Start()***: *focalPoint = GameObject.Find("Focal Point");*
2. In the ***AddForce*** call, Replace ***Vector3.forward*** with ***focalPoint.transform.forward***

- **New Concept:** Global vs Local XYZ
- **Tip:** Global XYZ directions relate to the entire scene, whereas local XYZ directions relate to the object in question

```
private GameObject focalPoint;

void Start() {
  playerRb = GetComponent<Rigidbody>();
  focalPoint = GameObject.Find("Focal Point"); }

void Update() {
  float forwardInput = Input.GetAxis("Vertical");
  playerRb.AddForce(~~Vector3.forward~~ focalPoint.transform.forward
  * speed * forwardInput); }
```

Lesson Recap

New Functionality	• Camera rotates around the island based on horizontal input • Player rolls in direction of camera based on vertical input
New Concepts and Skills	• Texture Wraps • Camera as child object • Global vs Local coordinates • Get direction of other object
Next Lesson	• In the next lesson, we'll add more challenge to the player, by creating enemies that chase them in the game.

unity

4.2 Follow the Player

Steps:

Step 1: Add an enemy and a physics material

Step 2: Create enemy script to follow player

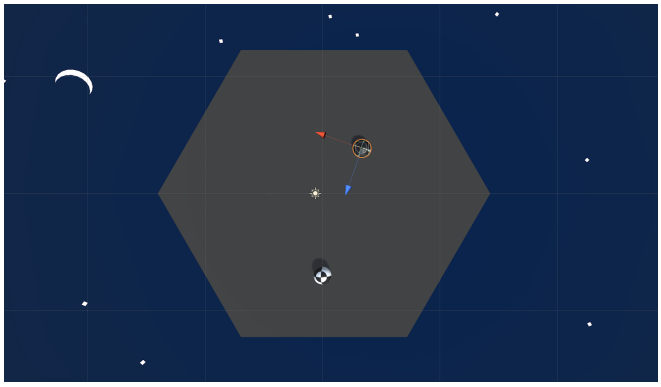
Step 3: Create a lookDirection variable

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.

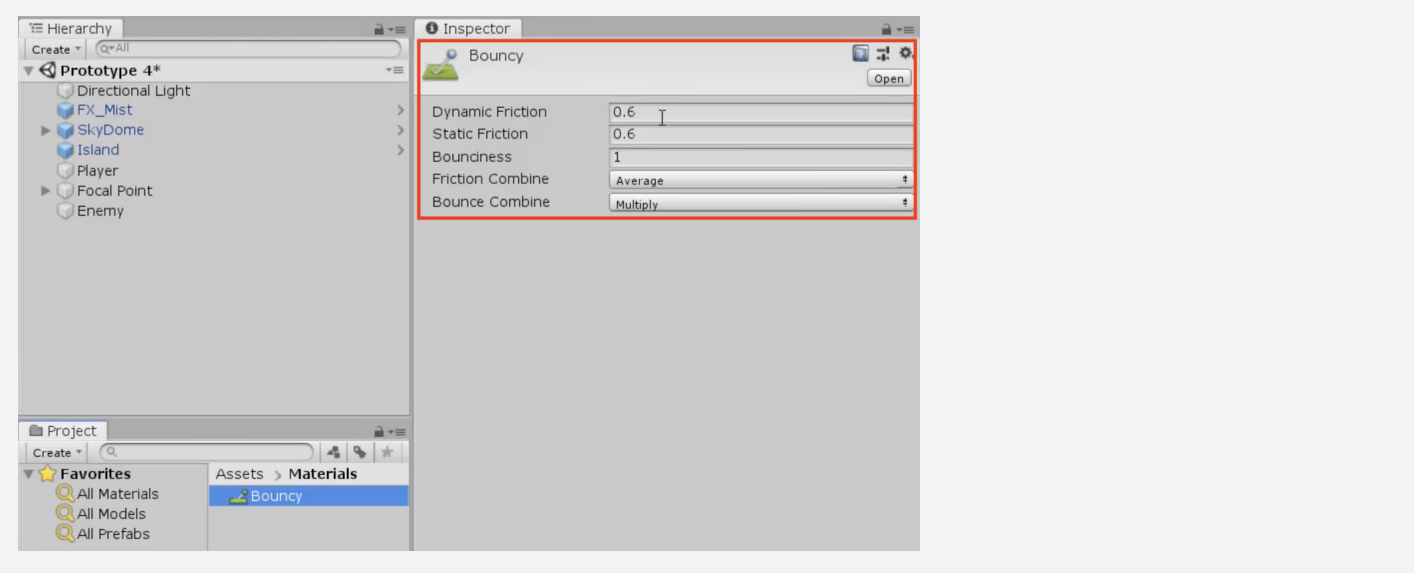
Learning Objectives: By the end of this lesson, you will be able to:
- Apply Physics Materials to make game objects bouncy
- Normalize vectors to point the enemy in the direction of the player
- Randomly spawn with Random.Range on two axes
- Write more advanced custom functions and variables to make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



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!

unity

4.3 PowerUp and CountDown

Steps:

Step 1: Choose and prepare a powerup

Step 2: Destroy powerup on collision

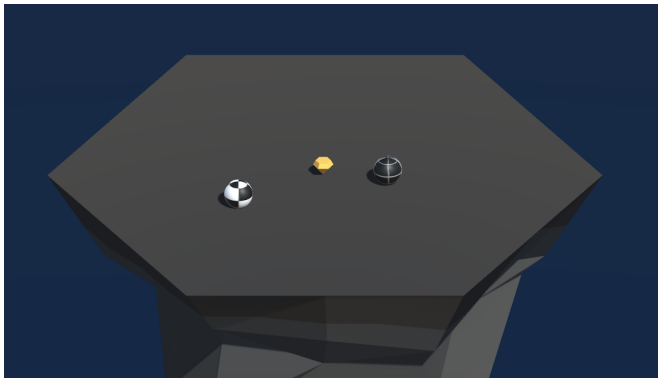
Step 3: Test for collision with a powerup

Step 4: Apply extra knockback with powerup

Step 5: Create Countdown Routine for powerup

Step 6: Add a powerup indicator

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.

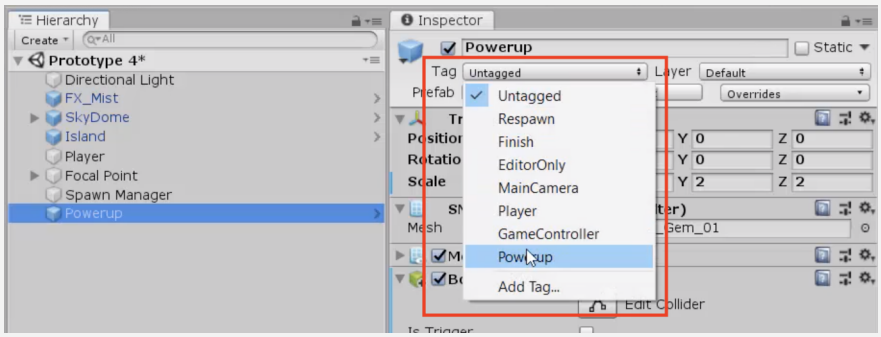
Learning Objectives: By the end of this lesson, you will be able to:
- Write informative debug messages with Concatenation and variables
- Repeat functions with the power of IEnumerator and Coroutines
- Use SetActive to make game objects appear and disappear from the scene

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("Collided with " + collision.gameObject.name
      + " 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!

unity

4.4 For-Loops For Waves

Steps:

Step 1: Write a for-loop to spawn 3 enemies

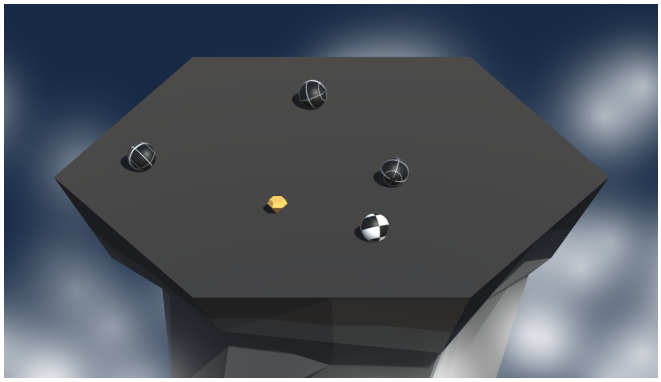
Step 2: Give the for-loop a parameter

Step 3: Destroy enemies if they fall off

Step 4: Increase enemyCount with waves

Step 5: Spawn Powerups with new waves

Example of project by end of lesson



Length: 60 minutes

Overview: We have all the makings of a great game; A player that rolls around and rotates the camera, a powerup that grants super strength, and an enemy that chases the player until the bitter end. In this lesson we will wrap things up by putting these pieces together!
First we will enhance the enemy spawn manager, allowing it to spawn multiple enemies and increase their number every time a wave is defeated. Lastly we will spawn the powerup with every wave, giving the player a chance to fight back against the ever-increasing horde of enemies.

Project Outcome: The Spawn Manager will operate in waves, spawning multiple enemies and a new powerup with each iteration. Every time the enemies drop to zero, a new wave is spawned and the enemy count increases.

Learning Objectives: By the end of this lesson, you will be able to:
- Repeat functions with For-loops
- Increment integer values in a loop with the ++ operator
- Target objects in a scene with FindObjectsOfType
- Return the length of an array as an integer with .Length

Step 1: Write a for-loop to spawn 3 enemies

We should challenge the player by spawning more than one enemy. In order to do so, we will repeat enemy instantiation with a loop.

1. In SpawnManager.cs, in ***Start()***, replace single **Instantiation** with a **for-loop** that spawns 3 enemies
2. Move the for-loop to a new ***void SpawnEnemyWave()*** function, then call that function from ***Start()***

- **New Concept:** For-loops
- **Don't worry:** Loops are a bit confusing at first, but they make sense eventually. Loops are powerful tools that programmers use often
- **New Concept:** ++ Increment Operator

```
void Start() {
  SpawnEnemyWave();
  for (int i = 0; i < 3; i++) {
    Instantiate(enemyPrefab, GenerateSpawnPosition(),
    enemyPrefab.transform.rotation); } }

void SpawnEnemyWave() {
  for (int i = 0; i < 3; i++) {
    Instantiate(enemyPrefab, GenerateSpawnPosition(),
    enemyPrefab.transform.rotation); } }
```

Step 2: Give the for-loop a parameter

Right now, SpawnEnemyWave spawns exactly 3 enemies, but if we're going to dynamically increase the number of enemies that spawn during gameplay, we need to be able to pass information to that method.

1. Add a parameter ***int enemiesToSpawn*** to the ***SpawnEnemyWave*** function
2. Replace ***i < __*** with ***i < enemiesToSpawn***
3. Add this new variable to the function call in ***Start()***: ***SpawnEnemyWave(___);***

- **New Concept:** Custom methods with parameters
- **Tip:** GenerateSpawnPosition returns a value, SpawnEnemyWave does not. SpawnEnemyWave takes a parameter, GenerateSpawnPosition does not.

```
void Start() {
  SpawnEnemyWave(3); }

void SpawnEnemyWave(int enemiesToSpawn) {
  for (int i = 0; i < 3 enemiesToSpawn; i++) {
    Instantiate(enemyPrefab, GenerateSpawnPosition(),
    enemyPrefab.transform.rotation); } }
```

Step 3: Destroy enemies if they fall off

Once the player gets rid of all the enemies, they're left feeling a bit lonely. We need to destroy enemies that fall, and spawn a new enemy wave once the last one is vanquished!

1. In Enemy.cs, **destroy** the enemies if their position is less than a **-Y value**
2. In SpawnManager.cs, declare a new ***public int enemyCount*** variable
3. In ***Update()***, set ***enemyCount = FindObjectsOfType<Enemy>().Length;***
4. Write the **if-statement** that if ***enemyCount == 0*** then ***SpawnEnemyWave***

- **New Function:** FindObjectsOfType

```
void Update() {
  ... if (transform.position.y < -10) { Destroy(gameObject); } }

<------>
public int enemyCount

void Update() {
  enemyCount = FindObjectsOfType<Enemy>().Length;
  if (enemyCount == 0) { SpawnEnemyWave(1); } }
```

Step 4: Increase enemyCount with waves

Now that we control the amount of enemies that spawn, we should increase their number in waves. Every time the player defeats a wave of enemies, more should rise to take their place.

1. Declare a new ***public int waveNumber = 1;***, then implement it in ***SpawnEnemyWave(waveNumber);***
2. In the if-statement that tests if there are 0 enemies left, **increment waveNumber** by 1

- **Tip:** Incrementing with the ++ operator is very handy, you may find yourself using it in the future

```
public int waveNumber = 1;

void Start() {
  SpawnEnemyWave(3 waveNumber); }

void Update() {
  enemyCount = FindObjectsOfType<Enemy>().Length;
  if (enemyCount == 0) { waveNumber++;  SpawnEnemyWave(1 waveNumber); } }
```

Step 5: Spawn Powerups with new waves

Our game is almost complete, but we're missing something. Enemies continue to spawn with every wave, but the powerup gets used once and disappears forever, leaving the player vulnerable. We need to spawn the powerup in a random position with every wave, so the player has a chance to fight back.

1. In SpawnManager.cs, declare a new ***public GameObject powerupPrefab*** variable, assign the **prefab** in the inspector and **delete** it from the scene
2. In **Start()**, **Instantiate** a new Powerup
3. Before the ***SpawnEnemyWave()*** call, **Instantiate** a new Powerup

- **Tip:** Now that we have a very playable game, let's test and tweak values

```
public GameObject powerupPrefab;

void Start() {
  ... Instantiate(powerupPrefab, GenerateSpawnPosition(),
  powerupPrefab.transform.rotation); }

void Update() {
  ... if (enemyCount == 0) { ... Instantiate(powerupPrefab,
    GenerateSpawnPosition(), powerupPrefab.transform.rotation); } }
```

Lesson Recap

New Functionality	• Enemies spawn in waves • The number of enemies spawned increases after every wave is defeated • A new power up spawns with every wave
New Concepts and Skills	• For-loops • Increment (++) operator • Custom methods with parameters • FindObjectsOfType

unity

Challenge 4

Soccer Scripting

Challenge Overview:

Use the skills you learned in the Sumo Battle prototype in a completely different context: the soccer field. Just like in the prototype, you will control a ball by rotating the camera around it and applying a forward force, but instead of knocking them off the edge, your goal is to knock them into the opposing net while they try to get into your net. Just like in the Sumo Battle, after every round a new wave will spawn with more enemy balls, putting your defense to the test. However, almost nothing in this project is functioning! It's your job to get it working correctly.

Challenge Outcome:

- Enemies move towards your net, but you can hit them to deflect them away
- Powerups apply a temporary strength boost, then disappear after 5 seconds
- When there are no more enemy balls, a new wave spawns with 1 more enemy

Challenge Objectives:

In this challenge, you will reinforce the following skills/concepts:

- Defining Vectors by subtracting one location in 3D space from another
- Track the number of objects of a certain type in a scene to trigger certain events
- Using Coroutines to perform actions based on a timed interval
- Using for-loops and dynamic variables to run code a particular number of times
- Resolving errors related to null references of unassigned variables

Challenge Instructions:

- Open your **Prototype 4** project
- **Download** the "Challenge 4 Starter Files" from the Tutorial Materials section, then double-click on it to **Import**
- In the *Project Window > Assets > Challenge 4 >* ***Instructions*** folder, use the resources as a guide to complete this challenge

Challenge		Task	Hint
1	Hitting an enemy sends it back towards you	When you hit an enemy, it should send it *away* from the player	In PlayerControllerX.cs, to get a Vector *away* from the player, you should subtract the [enemy position] minus the [player's position] - not the reverse
2	A new wave spawns when the player gets a powerup	A new wave should spawn when all enemy balls have been removed	In SpawnManagerX.cs, check that the enemyCount variable is being set correctly
3	The powerup never goes away	The powerup should only last for a certain duration, then disappear	In PlayerControllerX.cs, the PowerupCoolDown Coroutine code looks good, but this coroutine is never actually called with the StartCoroutine() method
4	2 enemies are spawned in every wave	One enemy should be spawned in wave 1, two in wave 2, three in wave 3, etc	In SpawnManagerX.cs, the for-loop that spawns enemy should make use of the enemiesToSpawn parameter
5	The enemy balls are not moving anywhere	The enemy balls should go towards the "Player Goal" object	There is an error in EnemyX.cs: "NullReferenceException: Object reference not set to an instance of an object". It looks like the playerGoal object is never assigned.

Bonus Challenge		Task	Hint
X	The player needs a turbo boost	The player should get a speed boost whenever the player presses spacebar - and a particle effect should appear when they use it	In PlayerController, add a simple if-statement that adds an "impulse" force if spacebar is pressed. To add a particle effect, first attach it as a child object of the Focal Point.
Y	The enemies never get more difficult	The enemies' speed should increase in speed by a small amount with every new wave	You'll need to track and increase the enemy speed in SpawnManagerX.cs. Then in EnemyX.cs, reference that speed variable and set it in Start().

Challenge Solution

1 In PlayerControllerX.cs, in OnCollisionEnter(), the ***awayFromPlayer*** Vector3 is in the opposite direction it should be.

```
Vector3 awayFromPlayer = transform.position -
other.gameObject.transform.position;
                        = other.gameObject.transform.position -
transform.position;
```

2 In SpawnManagerX.cs, the ***enemyCount*** variable is counting the number of objects with a “Powerup” tag - it should be counting the number of objects with an “Enemy” tag

```
void Update() {
  enemyCount = GameObject.FindGameObjectsWithTag("Powerup Enemy").Length;
  ...
}
```

3 In PlayerControllerX.cs, in the OnTriggerEnter() method, you need to initiate the ***PowerupCooldown*** Coroutine in order to begin the countdown process

```
private void OnTriggerEnter(Collider other) {
  if (other.gameObject.CompareTag("Powerup"))  {
    ...
    StartCoroutine(PowerupCooldown());
  }
}
```

4 In SpawnManagerX.cs, the for-loop that spawns enemy should make use of the ***enemiesToSpawn*** parameter

```
for (int i = 0; i < 2 enemiesToSpawn; i++) {
  Instantiate(enemyPrefab, GenerateSpawnPosition(), ...
}
```

5 In EnemyX.cs, the ***playerGoal*** variable is not initialized - initialize it in the Start() method

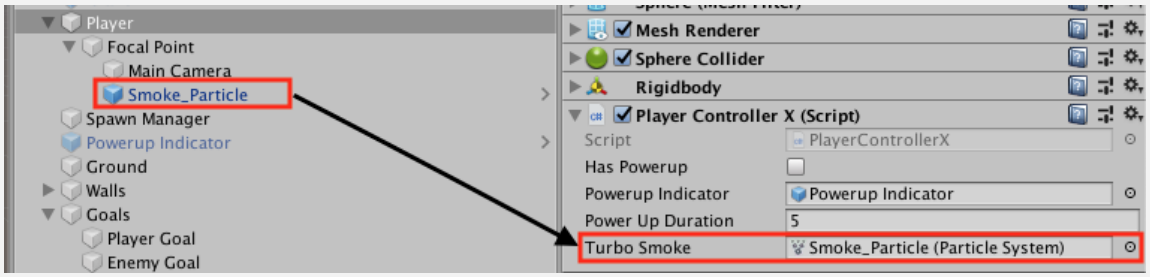
```
void Start() {
  enemyRb = GetComponent<Rigidbody>();
  playerGoal = GameObject.Find("Player Goal");
 }
```

Bonus Challenge Solution

X1 To add a turbo boost, In PlayerControllerX.cs, declare a new ***turboBoost*** float variable, then in Update(), add a simple if-statement that adds an "impulse" force in the direction of the focal point if spacebar is pressed:

```
private float turboBoost = 10;

void Update() {
  ...
  if (Input.GetKeyDown(KeyCode.Space)) {
   playerRb.AddForce(focalPoint.transform.forward * turboBoost, ForceMode.Impulse);
  }
}
```

X2 Add the Smoke_Particle prefab as a child object of the focal point (next to the camera), then in PlayerControllerX.cs, declare a new turboSmoke particle variable and assign it in the inspector



X3 In PlayerControllerX.cs, in the if-statement checking if the player presses spacebar, play the particle

```
if (Input.GetKeyDown(KeyCode.Space)) {
  playerRb.AddForce(focalPoint.transform.forward * turboBoost, ForceMode.Impulse);
  turboSmoke.Play();
}
```

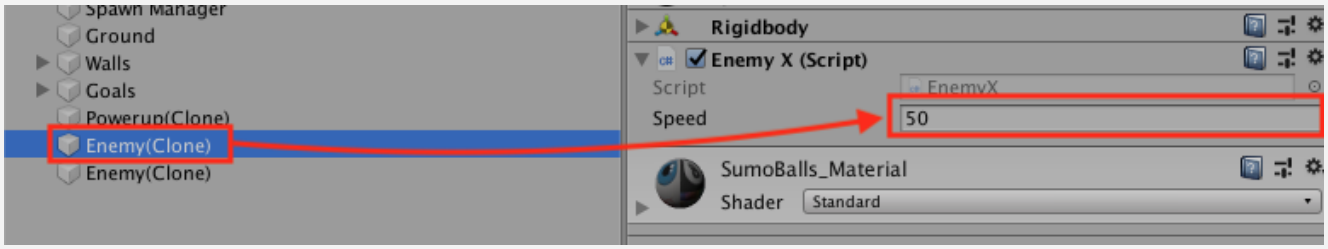
Y1 In SpawnManagerX.cs, declare and initialize a new public ***enemySpeed*** variable, then increase it by a certain amount every time a wave is spawned:

```
public int enemyCount;
public float enemySpeed = 50;

void SpawnEnemyWave(int enemiesToSpawn) {
  ...
  waveCount++;
  enemyCount += 25;
}
```

Y2 In EnemyX.cs, declare a new spawnManagerXScript variable, get a reference to it in Start(), then set the enemy's ***speed*** variable to your new ***enemySpeed*** variable

```
private GameObject playerGoal;
private SpawnManagerX spawnManagerXScript;

void Start() {
  enemyRb = GetComponent<Rigidbody>();
  playerGoal = GameObject.Find("Player Goal");
  spawnManagerXScript = GameObject.Find("Spawn Manager").GetComponent<SpawnManagerX>();
  speed = spawnManagerXScript.enemySpeed;
}
```

Y3 To test, make the ***speed*** variable in EnemyX.cs public and check the enemies' speed when they are spawned in different waves



Unit 4 Lab

Basic Gameplay

Steps:

Step 1: Give objects basic movement

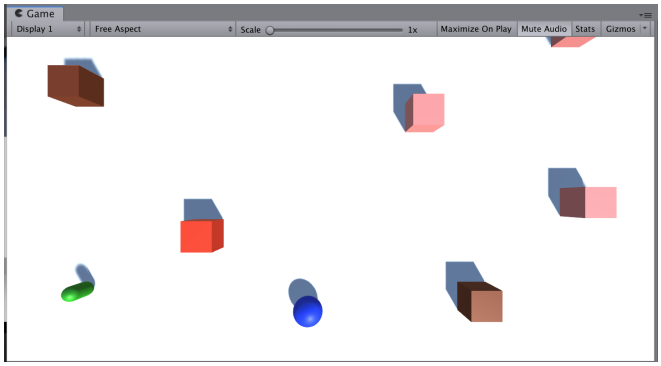
Step 2: Destroy objects off-screen

Step 3: Handle object collisions

Step 4: Make objects into prefabs

Step 5: Make SpawnManager spawn Prefabs

Example of progress by end of lab



Length: 60 minutes

Overview: In this lab, you will work with all of your non-player objects in order to bring your project to life with its basic gameplay. You will give your projectiles, pickups, or enemies their basic movement and collision detection, make them into prefabs, and have them spawned randomly by a spawn manager. By the end of this lab, you should have a glimpse into the core functionality of your game.

Project Outcome: Non-player objects are spawned at appropriate locations in the scene with basic movement. When objects collide with each other, they react as intended, by either bouncing or being destroyed.

Learning Objectives: By the end of this lab, you will be able to:
- More comfortably program basic movement
- More comfortably handle object collisions
- More comfortably spawn object prefabs on timed intervals

Step 1: Give objects basic movement

Before you spawn objects into your scene, they should move the way you want.

1. If relevant, add **Rigidbody** components to your non-player objects
2. Create a **new script**(s) for the objects that will be instantiated during gameplay and attach them to their respective objects (including projectiles or pickups)
3. Program the **basic movement** for your objects and test that they work

- **Tip:** Make sure you uncheck "use gravity" if you don't want them to fall
- **Tip:** In the collider component, check "is trigger" if you don't want actual collisions

By the end of this step, all objects should basically move the way they should in the game.

Step 2: Destroy objects off-screen

To make sure our hierarchy doesn't get too cluttered, let's make sure these objects get destroyed when they leave the screen.

1. Either create a new script or add code to your existing script to make sure objects are destroyed when they leave the screen

- **Tip:** Move your objects in scene view to determine the xyz positions objects should be destroyed

By the end of this step, objects should be removed from the hierarchy when they are no longer in play.

Step 3: Handle object collisions

Now that you have all these moving objects, they're bound to start colliding with each other - we need to program what should happen when everything collides.

1. If relevant, edit the **Rigidbody mass** of your objects
2. If relevant, to change the way your objects collide, create a new **Physics material** for your objects
3. Add **tags** to your objects so you can accurately test for which objects are colliding with which
4. Use ***OnCollisionEnter()*** (for Rigidbody collisions) or ***OnTriggerEnter()*** (for trigger-based collisions) to **Destroy** or **Log** messages to the console what should happen when certain collisions occur

- **Don't worry:** If you collide with a powerup or pickup, the actual functionality does *not* need to be programmed, just the effect
- **Tip:** Should use OnTriggerEnter if objects are being destroyed - but remember that "Is Trigger" must be checked for this to work!

By the end of this step, objects should destroy, bounce, or do nothing based on collisions.

Step 4: Make objects into prefabs

Now that the objects are basically behaving the way they should, if they're going to be instantiated during gameplay, they need to be prefabs

1. In the Assets directory, create a new **Folder** called "Prefabs"
2. **Drag** in each object to create a **new prefab** for it
3. After all objects have been turned into prefabs, **delete** them from the scene
4. **Test** the objects' behavior by dragging them from the Prefabs folder into the scene while the game is running

- **Tip:** When creating new prefabs, you have to drag them one at a time
- **Tip:** Notice that their icons turn blue when they are prefabs

By the end of this step, all objects that will be spawned during gameplay should be prefabs and should no longer be in your scene.

Step 5: Make SpawnManager spawn Prefabs

Now that we have all of our prefabs set up, we can create a spawn manager to spawn them at intervals and, if we want, in random locations.

1. Create an Empty "Spawn Manager" object and attach a new SpawnManager.cs script to it
2. Create individual **GameObject or GameObject array** variables for your prefabs, then **assign** them in the inspector
3. Use the ***Instantiate()***, ***Random.Range()***, and the ***InvokeRepeating()*** methods to spawn objects at intervals (random objects, random locations, or both)
4. Right-click on your *Assets folder* > ***Export Package*** then save a new version in your **Backups** folder

- **Tip:** Name your variables "____Prefab" so you know it requires a prefab value
- **Don't worry:** If it's not perfect yet of if there are some minor bugs - just get the general idea working

By the end of this step, objects should be spawned automatically from the appropriate location.

Lesson Recap

New Progress	• Non-player objects prefabs have basic movement • Objects are destroyed when they leave the screen • Collisions between objects are handled appropriately • Objects are spawned at the appropriate locations on time-based intervals
New Concepts and Skills	• Creating basic gameplay for a project independently

unity

Quiz Unit 4

QUESTION	CHOICES
1 You're trying to write some code that creates a random age between 1 and 100 and prints that age, but there is an error. What would fix the error?	a. Change line 1 to "private float age" b. Add the word "int" to line 8, so it says "int age = …" c. On line 7, change the word "private" to "void" d. Add a new line after line 8 that says "return age;"

```
1. private int age;
2.
3. void Start() {
4.   Debug.Log(GenerateRandomAge());
5. }
6.
7. private int GenerateRandomAge() {
8.   age = Random.Range(1, 101);
9. }
```

2 The following message was displayed in the console: "Monica has 20 dollars". Which of the line options in the PrintNames function produced it?	a. Option A b. Option B c. Option C d. Option D

```
string[] names = new string[] { "Steve", "Monica", "Eric" };
int money = 5;

void Start() {
  money *= 2;
  PrintNames();
}

void PrintNames () {
  A. Debug.Log("Monica has " + money/2 + " dollars");
  B. Debug.Log(names[1] + " has " + money*2 + " dollars");
  C. Debug.Log(names[2] + " has " + money*2 + " dollars");
  D. Debug.Log(names[Monica] + " has " + money/2 + " dollars");
}
```

3 The code below produces “error CS0029: Cannot implicitly convert type 'float' to 'UnityEngine.Vector3'”. Which of the following would remove the error?

```
1. private Vector3 startingVelocity;
2. void Start() {
3.   startingVelocity = 2.0f;
4. }
```

a. On line 1, change “Vector3” to “float”
b. On line 3, change “=” to “+”
c. Either A or B
d. None of the above

4 Which of the following follows Unity’s naming conventions (especially as it relates to capitalization)?

```
A. float forwardInput = Input.GetAxis("Vertical");
B. float ForwardInput = input.GetAxis("Vertical");
C. Float forwardInput = Input.getAxis("Vertical");
D. float forwardInput = input.getAxis("vertical");
```

a. Line A
b. Line B
c. Line C
d. Line D

5 You are trying to assign the powerup variable in the inspector, but it is not showing up in the Player Controller component. What is the problem?

```
public class PlayerController : MonoBehaviour
{
  private GameObject powerup;
}
```

a. You cannot declare a powerup variable in the Player Controller Script
b. You cannot assign GameObject type variables in the inspector
c. The powerup variable should be public instead of private
d. The PlayerController class should be private instead of public

6 Your game has just started and you see the error, “UnassignedReferenceException: The variable playerIndicator of PlayerController has not been assigned.” What is likely the solution to the problem?

```
public class PlayerController : MonoBehaviour
{
  public GameObject playerIndicator;
  void Update() {
    playerIndicator.transform.position.y = 10;
  }
}
```

a. PlayerController variable in the playerIndicator script needs to be declared
b. The playerIndicator variable needs to be made private
c. The PlayerController script must be assigned to the player object
d. An object needs to be dragged onto the playerIndicator variable in the inspector

7 You are trying to create a new method that takes a number and multiplies it by two. Which method would do that?

a. Method A
b. Method B
c. Method C
d. Method D

```
A. private float DoubleNumber() {
     return number *= 2;
   }

B. private float DoubleNumber(float number) {
     return number *= 2;
   }

C. private void DoubleNumber(float number) {
     return number *= 2;
   }

D. private void DoubleNumber() {
     return number *= 2;
   }
```

8 Which comment best describes the code below?

```
public class Enemy : MonoBehaviour
{
  // Comment
  private void OnTriggerEnter(Collider other) {
    if(other.CompareTag("Spike")) {
      Destroy(other.gameObject);
    }
  }
}
```

a. // If the player collides with an enemy, destroy the enemy
b. // If the enemy collides with a spike, destroy the spike
c. // If the enemy collides with a spike, destroy the enemy
d. // If the player collides with a spike, destroy the spike

9 The code below produces the error, “error CS0029: Cannot implicitly convert type 'UnityEngine.GameObject' to 'UnityEngine.Rigidbody'”. What could be done to fix this issue?

```
1. void OnCollisionEnter(Collision collision) {
2.   if(collision.gameObject.CompareTag("Enemy")) {
3.     Rigidbody enemyRb = collision.gameObject;
4.   }
5. }
```

a. On line 1, change “collision” to “Rigidbody”
b. On line 2, change “gameObject” to “Rigidbody”
c. On line 3, delete “.gameObject”
d. On line 3, add “.GetComponent<Rigidbody>()” before the semicolon

10 Which of the following statements about functions/methods are correct:

A. Functions/methods must be passed at least one parameter
B. Functions/methods with a “void” return type cannot be passed parameters
C. A Function/method with an “int” return type could include the code, “return 0.5f;”
D. If there was a function/method declared as “private void RenameObject(string newName)”, you could call that method with “RenameObject();”

a. A and B are correct
b. Only B is correct
c. B and C are correct
d. Only D is correct
e. None are correct

Quiz Answer Key

#	ANSWER	EXPLANATION
1	D	Since the method has an “int” return type “private ***int*** GenerateRandomAge()”, it must *return* an int.
2	B	Debug.Log(names[1] + " has " + money*2 + " dollars"); is correct. Arrays start with index 0, so “Monica” has the index value of “1” (names[1]). In start, money is multiplied by 2, making it 10, so “money*2” would give you the value of 20.
3	A	Changing “Vector3” to “float” would work because you would just be multiplying a flat by another float. Changing “=” to “+” would not work because you can’t add a float to a Vector3.
4	A	Lowercase “float”, camelCase variables, Capitalized class & method names
5	C	Making a variable public will make it appear in the inspector.
6	D	If the consoles says a variable is not assigned, you most likely forgot to assign that variable by dragging on object onto it in the inspector.
7	B	Since it needs to “return” a value, it should have a return type of “private ***float***” as opposed to “private ***void.***” Since it needs to take a number, it needs a float parameter (“float number”).
8	B	Since this is the “Enemy” class, we are testing for the enemy colliding with something. Since it destroys “***other.****gameObject”,* it will destroy the spike.
9	D	The code cannot convert a RigidBody type variable to a GameObject type variable, so you have to get the RigidBody component from the gameObject
10	E	A. Functions/methods do not necessarily require parameters B. Functions/methods with a “void” return type *can* be passed parameters C. A Function/method with an “int” return type could not include the code, “return 0.5f;”, since 0.5f is a float D. If there was a function/method declared as “private void RenameObject(string newName)”, you would have to pass it a string parameter, such as RenameObject(“Steve”);

unity

Bonus Features 4 - Share your Work

Steps:

Length: 60 minutes

Overview: In this tutorial, you can go way above and beyond what you learned in this Unit and share what you've made with your fellow creators.

There are four bonus features presented in this tutorial marked as Easy, Medium, Hard, and Expert. You can attempt any number of these, put your own spin on them, and then share your work!

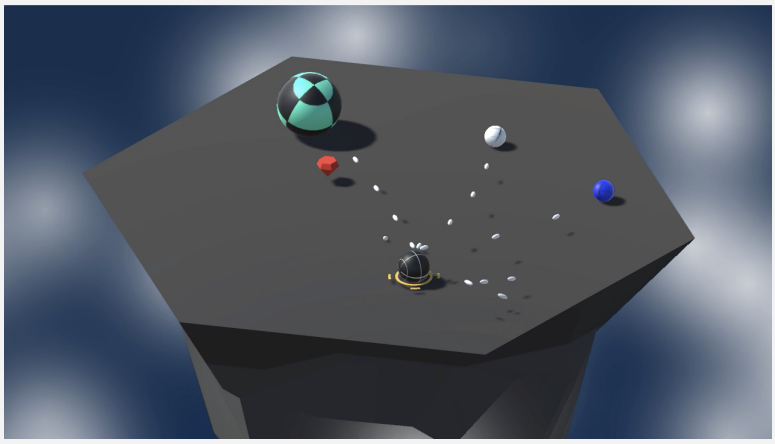
This tutorial is entirely optional, but highly recommended for anyone wishing to take their skills to a new level.

Step 1: Overview

This tutorial outlines four potential bonus features for the Sumo Battle Prototype at varying levels of difficulty:

- **Easy**: Harder enemy
- **Medium**: Homing rockets
- **Hard**: Smashingly good
- **Expert**: Boss battle

Here's what the prototype could look like if you complete all four features:



The Easy and Medium features can probably be completed entirely with skills from this course, but the Hard and Expert features will require some additional research.

Since this is optional, you can attempt none of them, all of them, or any combination in between. You can come up with your own original bonus features as well!

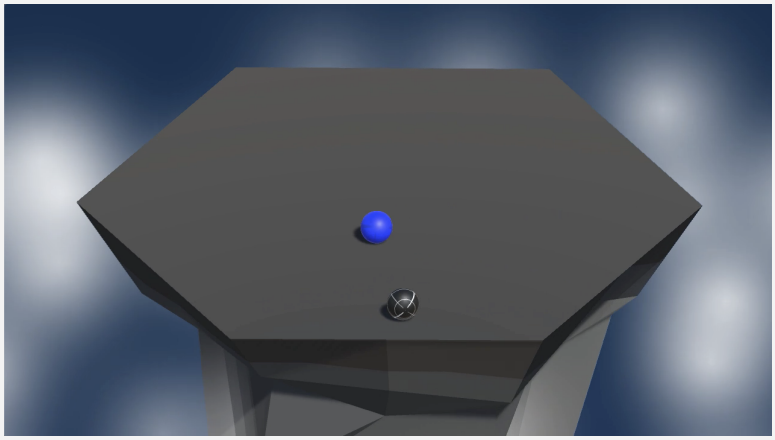
Then, at the end of this tutorial, there is an opportunity to share your work.

We highly recommend that you attempt these using relentless Googling and troubleshooting, but if you do get completely stuck, there are hints and step-by-step solutions available below.

Good luck!

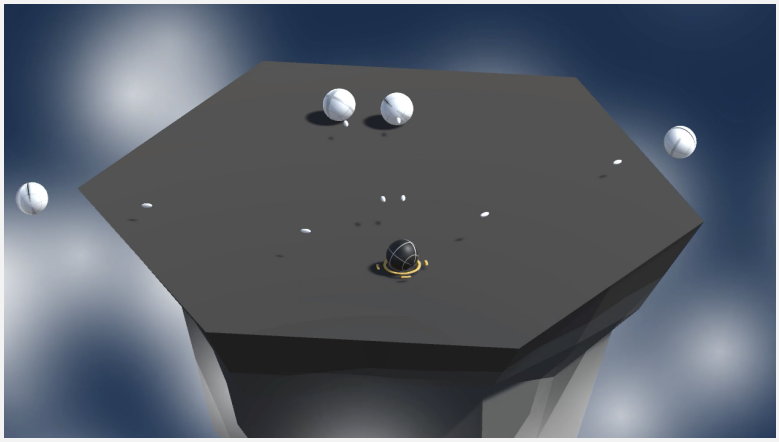
Step 2: Easy: Harder enemy

Add a new more difficult type of enemy and randomly select which is spawned.



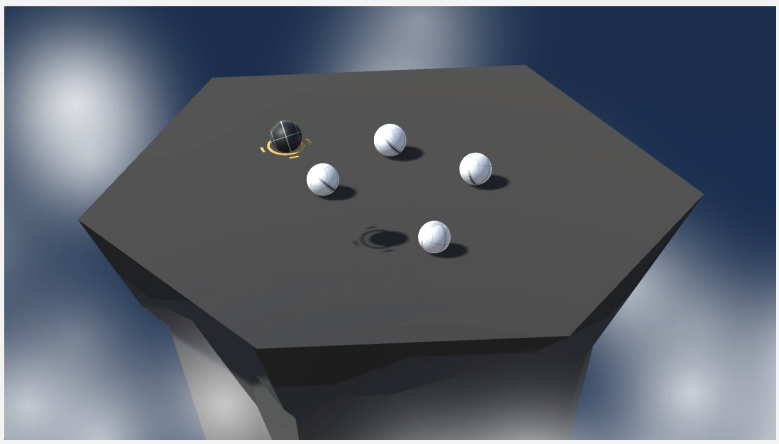
Step 3: Medium: Homing rockets

Create a new powerup that gives the player the ability to launch projectiles at enemies to knock them off (or something that automatically fires projectiles in all directions when the powerup is enabled).



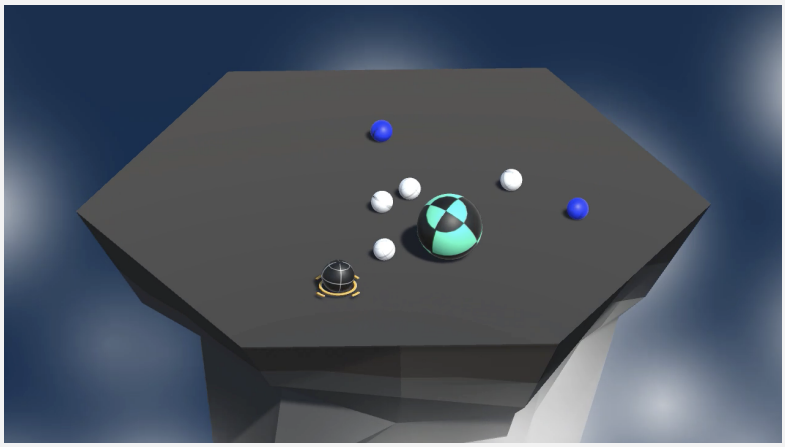
Step 5: Hard: Smash attack

Create a new powerup that allows the player to hop up into the air and smash down onto the ground, sending any enemies nearby flying away from the player. Ideally, the closer an enemy is, the more it should be impacted by the smash.



Step 6: Expert: Boss battle

After a certain number of waves, program a mini “boss battle,” where the boss has some completely new abilities. For example, maybe the boss can fire projectiles at you, maybe it is extremely agile, or maybe it occasionally generates little minions that come after you.



Step 7: Hints and solution walkthrough

Hints:

- Easy: Harder enemy
  - Try using an array for the enemy prefabs.
- Medium: Homing rockets
  - Try using an enum to differentiate the power ups
- Hard: Smashingly good
  - Extend the enum you created in the previous challenge
- Expert: Boss battle
  - Create a new SpawnBossWave function that only runs if the wave number is a multiple of a particular value.

Solution walkthrough
If you are really stuck, download the step-by-step solution walkthrough.
Note that there are likely many ways to implement these features - this is only one suggestion.

Step 8: Share your work

Have you implemented any of these bonus features? Have you added any new, unique features? Have you applied these new features to another project?

We would love to see what you've created!

Please take a screenshot of your project or do a screen-recording walking us through it, then post it here to share what you've made.

We highly recommend that you comment on at least one other creator's submission. What do you like about the project? What would be a cool new feature they might consider adding?