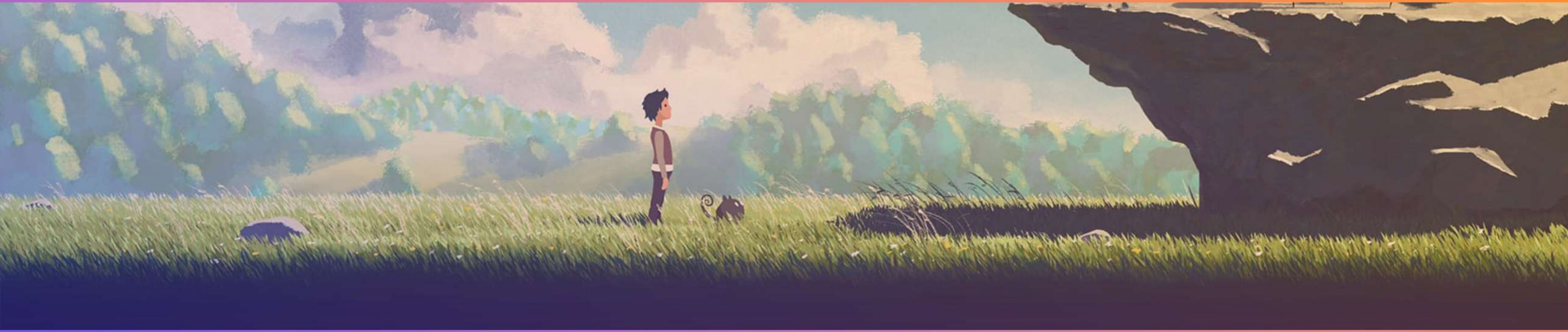


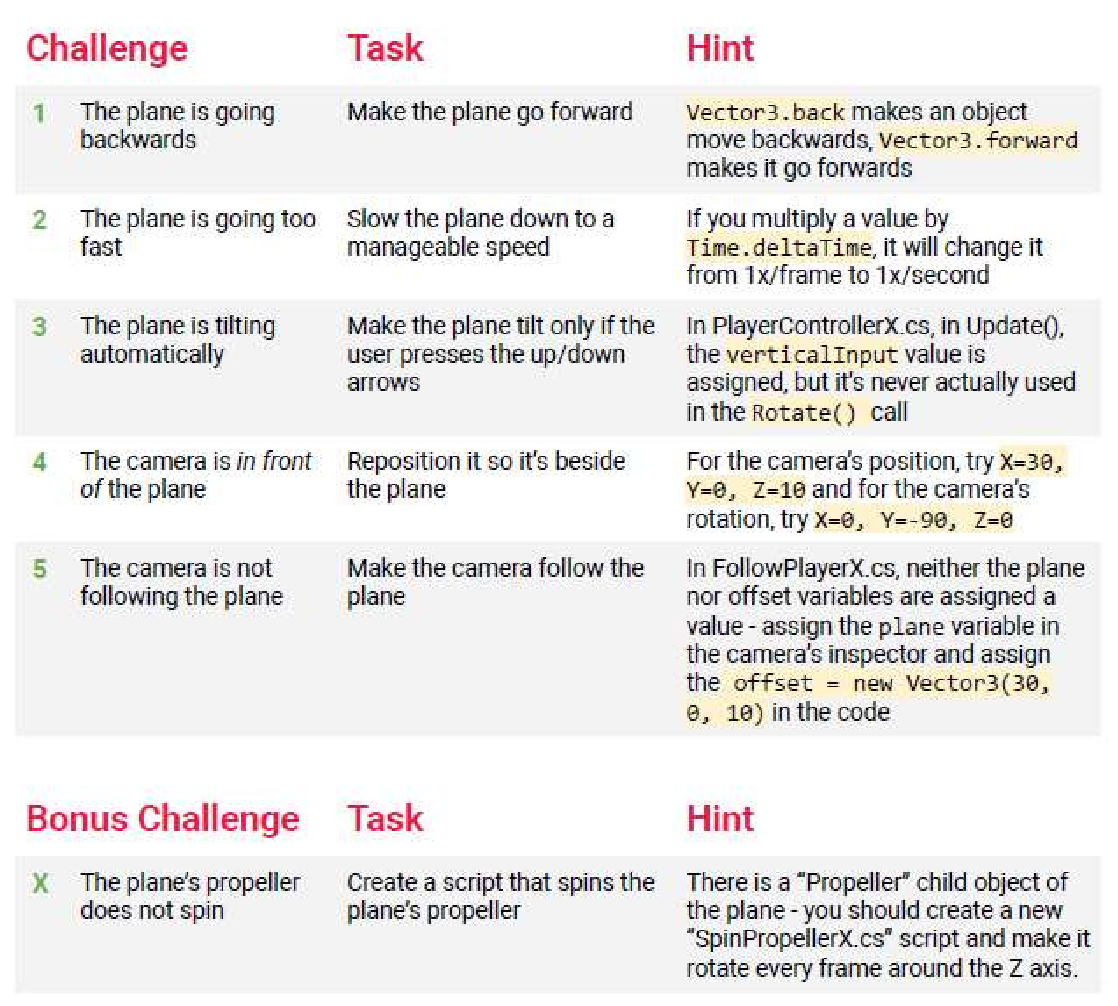
การเขียนโปรแกรมคอมพิวเตอร์ขั้นสูงเพื่อควบคุมอุปกรณ์

ADVANCE COMPUTER PROGRAMMING

สอนโดย พงศธร เกียรติเจริญพร (มิว)

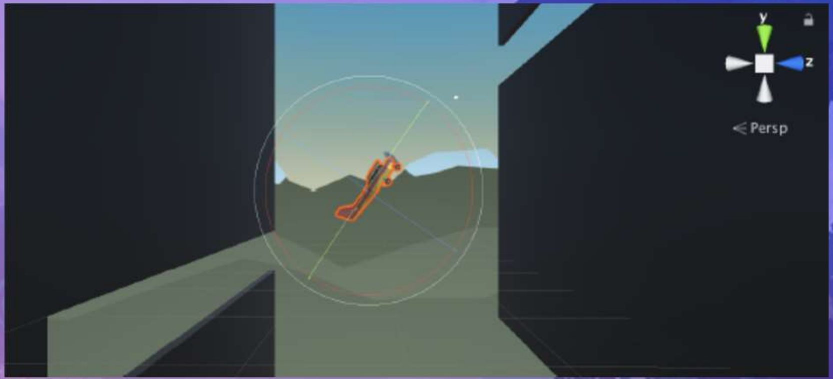
15/09/2021

CHALLENGE 01

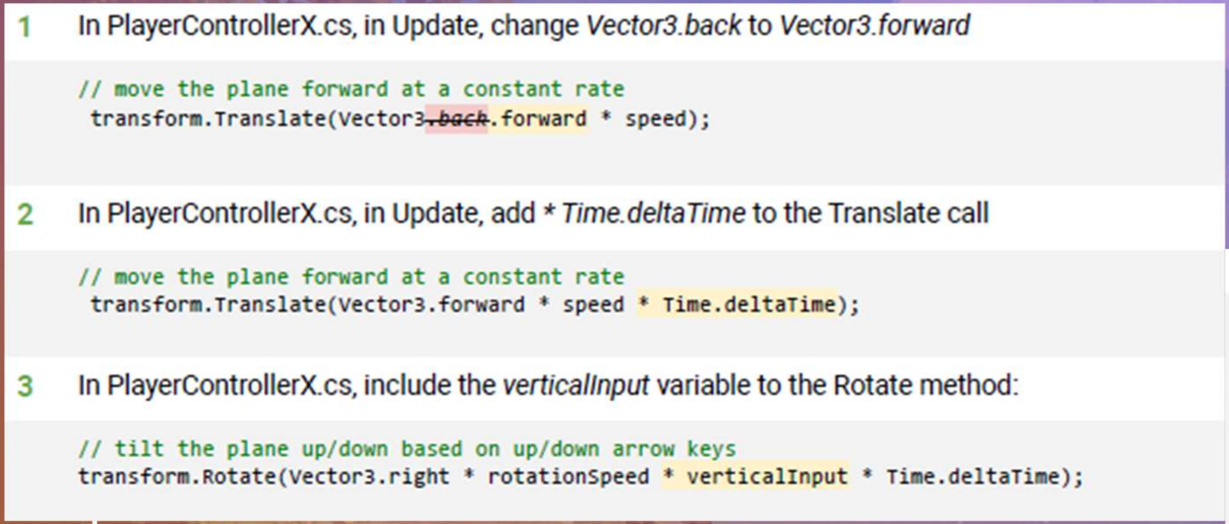


	Challenge	Task	Hint
1	The plane is going backwards	Make the plane go forward	`Vector3.back` makes an object move backwards, `Vector3.forward` makes it go forwards
2	The plane is going too fast	Slow the plane down to a manageable speed	If you multiply a value by `Time.deltaTime`, it will change it from 1x/frame to 1x/second
3	The plane is tilting automatically	Make the plane tilt only if the user presses the up/down arrows	In PlayerControllerX.cs, in Update(), the `verticalInput` value is assigned, but it's never actually used in the `Rotate()` call
4	The camera is *in front of* the plane	Reposition it so it's beside the plane	For the camera's position, try `X=30, Y=0, Z=10` and for the camera's rotation, try `X=0, Y=-90, Z=0`
5	The camera is not following the plane	Make the camera follow the plane	In FollowPlayerX.cs, neither the plane nor offset variables are assigned a value - assign the `plane` variable in the camera's inspector and assign the `offset = new Vector3(30, 0, 10)` in the code

	Bonus Challenge	Task	Hint
X	The plane's propeller does not spin	Create a script that spins the plane's propeller	There is a "Propeller" child object of the plane - you should create a new "SpinPropellerX.cs" script and make it rotate every frame around the Z axis.



SOLUTION



1 In PlayerControllerX.cs, in Update, change *Vector3.back* to *Vector3.forward*

```
// move the plane forward at a constant rate
 transform.Translate(Vector3~~.back~~.forward * speed);
```

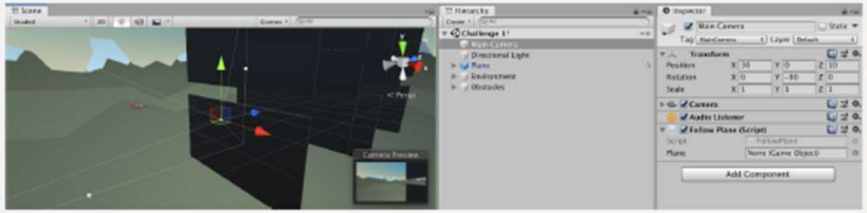
2 In PlayerControllerX.cs, in Update, add ** Time.deltaTime* to the Translate call

```
// move the plane forward at a constant rate
 transform.Translate(Vector3.forward * speed * Time.deltaTime);
```

3 In PlayerControllerX.cs, include the *verticalInput* variable to the Rotate method:

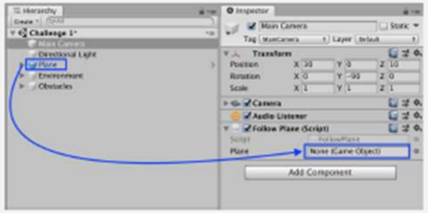
```
// tilt the plane up/down based on up/down arrow keys
transform.Rotate(Vector3.right * rotationSpeed * verticalInput * Time.deltaTime);
```

4 Change the camera's position to (30, 0, 10) and its rotation, to (0, -90, 0)



Main Camera
Transform
Position X 30 Y 0 Z 10
Rotation X 0 Y -90 Z 0
Scale X 1 Y 1 Z 1
Camera
Audio Listener
Follow Plane (Script)
Script FollowPlane
Plane None (Game Object)

5 To assign the *plane* variable, select **Main Camera** in the hierarchy, then drag the **Plane** object onto the "Plane" variable in the inspector

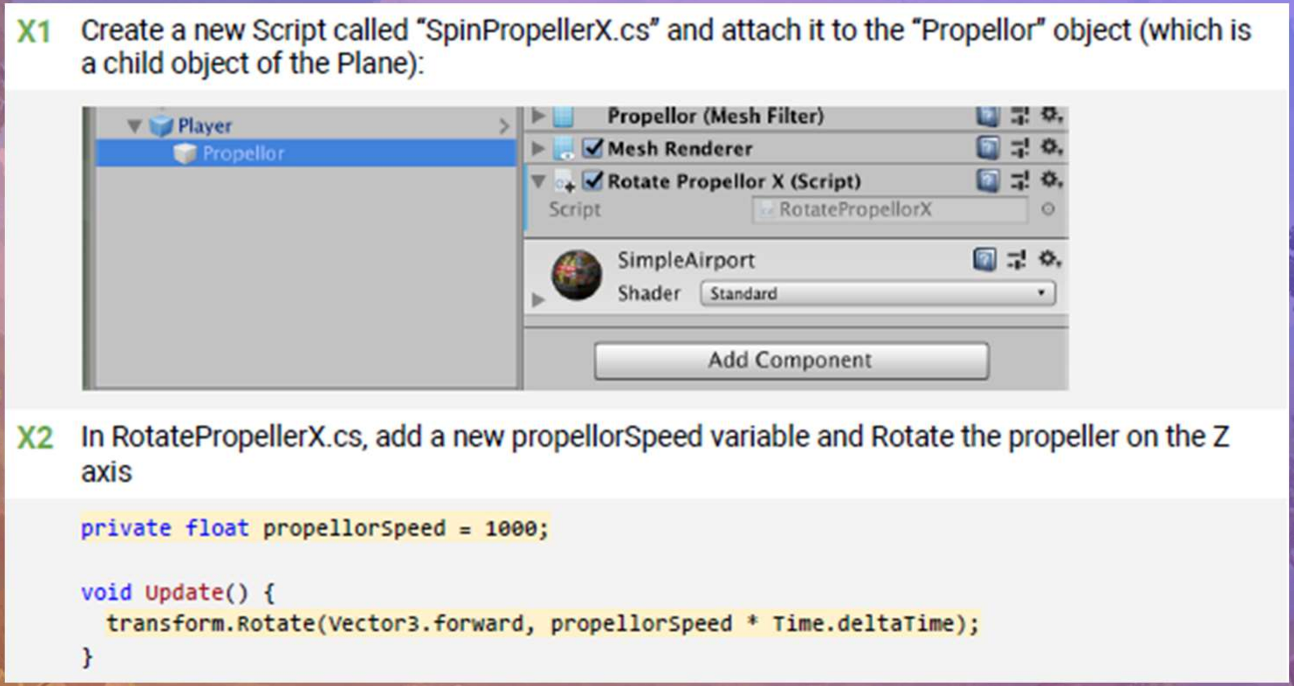


Main Camera
Transform
Position X 30 Y 0 Z 10
Rotation X 0 Y -90 Z 0
Scale X 1 Y 1 Z 1
Camera
Audio Listener
Follow Plane (Script)
Script FollowPlane
Plane None (Game Object)

To assign the *offset variable*, add the value as a new Vector3 at the top of FollowPlane.cs:

```
private Vector3 offset = new Vector3(30,
0, 10);
```

SOLUTION



X1 Create a new Script called “SpinPropellerX.cs” and attach it to the “Propellor” object (which is a child object of the Plane):

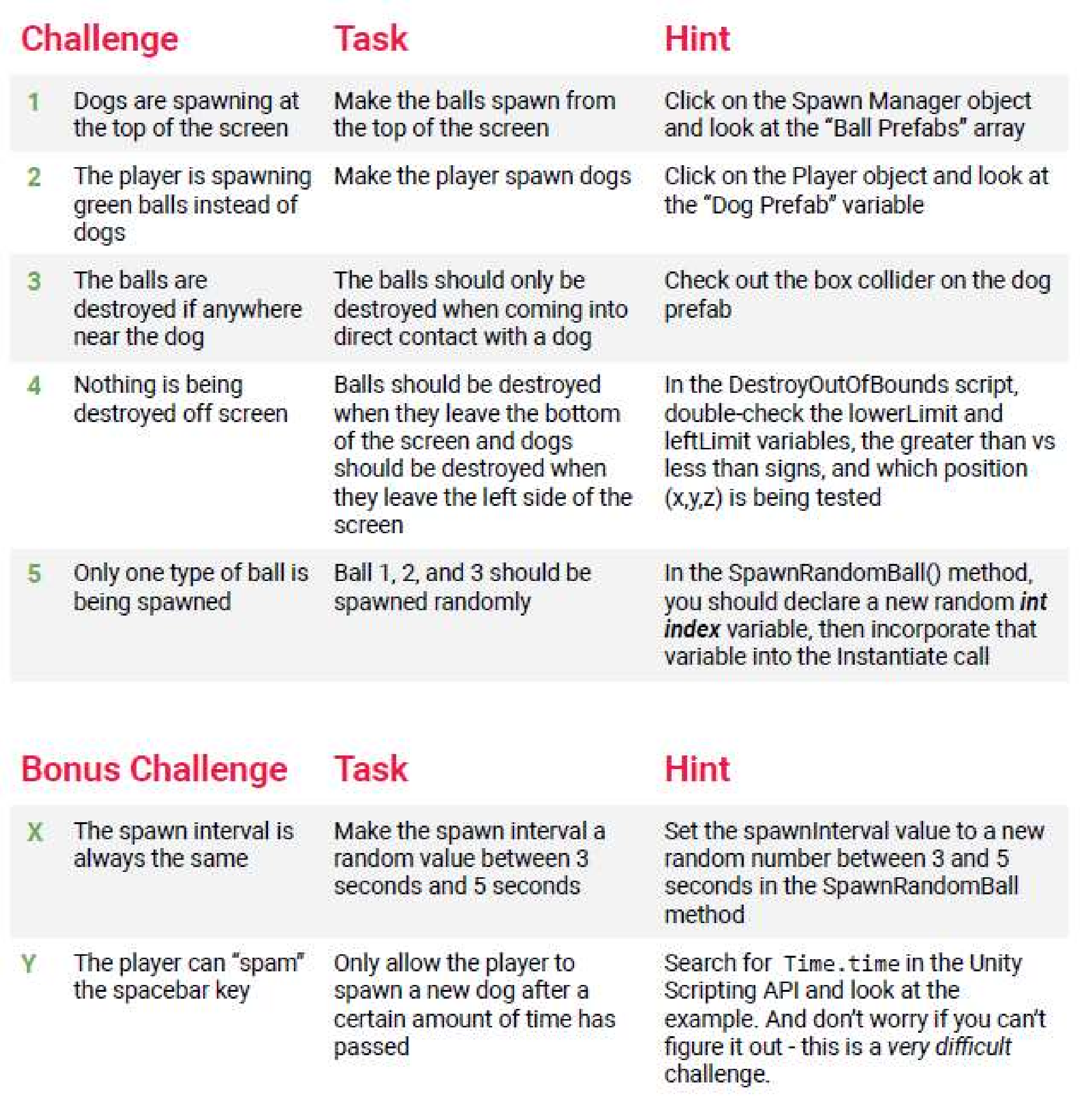
Script RotatePropellorX

Shader Standard

X2 In RotatePropellerX.cs, add a new propellorSpeed variable and Rotate the propeller on the Z axis

```
private float propellorSpeed = 1000;

void Update() {
  transform.Rotate(Vector3.forward, propellorSpeed * Time.deltaTime);
}
```

02



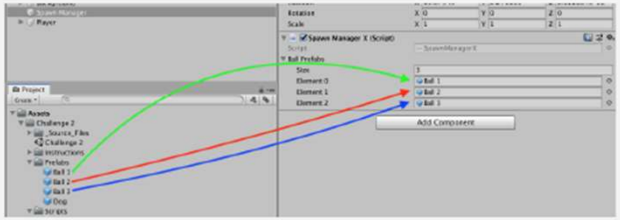
	Challenge	Task	Hint
1	Dogs are spawning at the top of the screen	Make the balls spawn from the top of the screen	Click on the Spawn Manager object and look at the "Ball Prefabs" array
2	The player is spawning green balls instead of dogs	Make the player spawn dogs	Click on the Player object and look at the "Dog Prefab" variable
3	The balls are destroyed if anywhere near the dog	The balls should only be destroyed when coming into direct contact with a dog	Check out the box collider on the dog prefab
4	Nothing is being destroyed off screen	Balls should be destroyed when they leave the bottom of the screen and dogs should be destroyed when they leave the left side of the screen	In the DestroyOutOfBounds script, double-check the lowerLimit and leftLimit variables, the greater than vs less than signs, and which position (x,y,z) is being tested
5	Only one type of ball is being spawned	Ball 1, 2, and 3 should be spawned randomly	In the SpawnRandomBall() method, you should declare a new random ***int index*** variable, then incorporate that variable into the Instantiate call

	Bonus Challenge	Task	Hint
X	The spawn interval is always the same	Make the spawn interval a random value between 3 seconds and 5 seconds	Set the spawnInterval value to a new random number between 3 and 5 seconds in the SpawnRandomBall method
Y	The player can "spam" the spacebar key	Only allow the player to spawn a new dog after a certain amount of time has passed	Search for `Time.time` in the Unity Scripting API and look at the example. And don't worry if you can't figure it out - this is a *very difficult* challenge.

15/09/2021

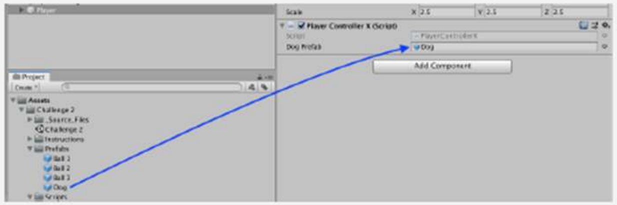
SOLUTION

1 Select the Spawn Manager object and expand the “Ball Prefabs” array, then drag the **Ball 1, 2, 3** prefabs from *Assets > Challenge 2 > Prefabs* onto **Element 0, 1, 2**



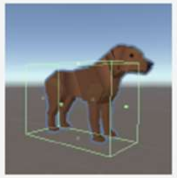
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
☑ Spawn Manager X (Script)
Script: SpawnManagerX
Ball Prefabs
Size: 3
Element 0: Ball 1
Element 1: Ball 2
Element 2: Ball 3

2 Select the Player object and drag the **Dog** prefab from *Assets > Challenge 2 > Prefabs* onto the “Dog Prefab” variable

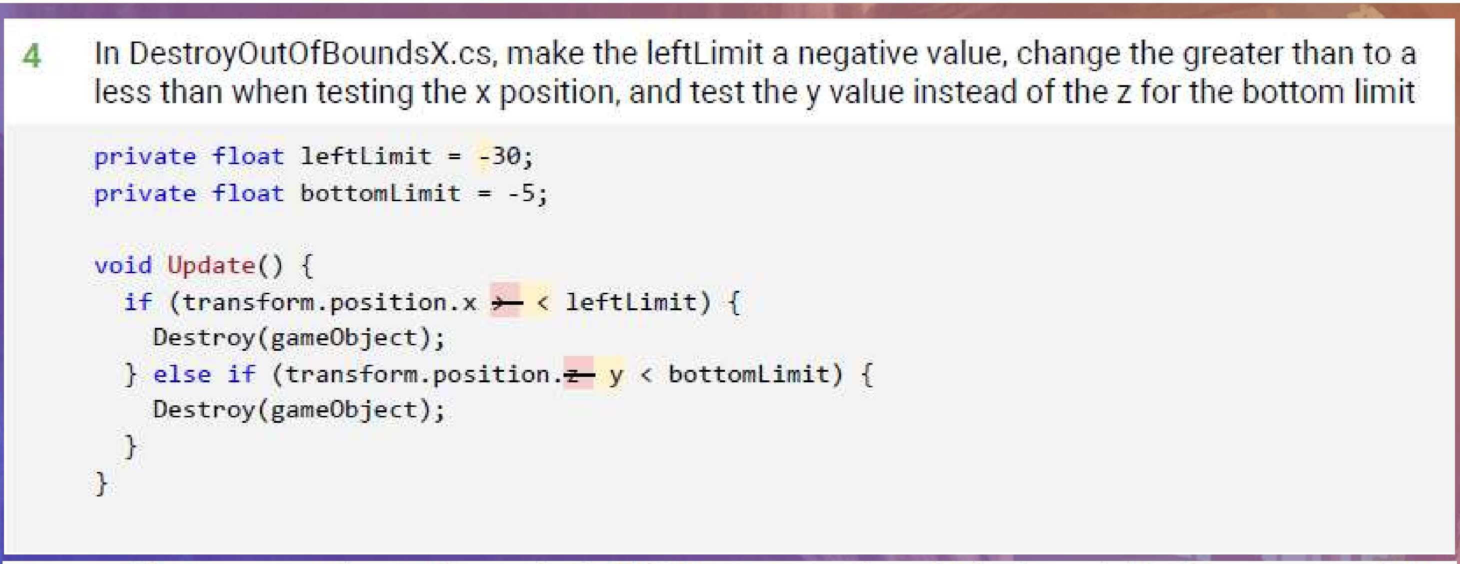


Scale X 2.5 Y 2.5 Z 2.5
☑ Player Controller X (Script)
Script: PlayerControllerX
Dog Prefab: Dog

3 Double-click on the Dog prefab, then in the Box Collider component, click **Edit Collider**, and reduce the collider to be the same size as the dog

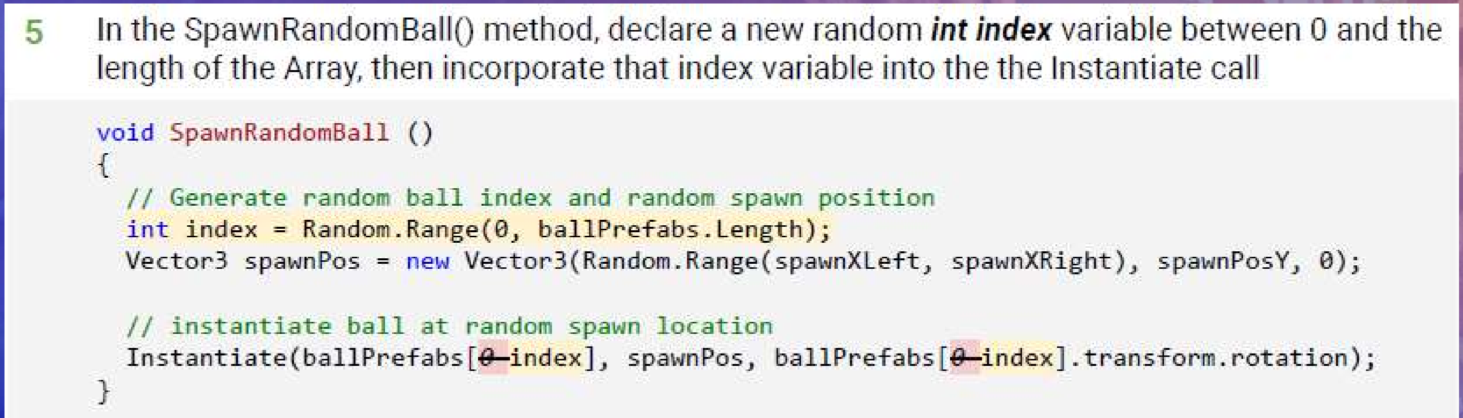


SOLUTION



4 In DestroyOutOfBoundsX.cs, make the leftLimit a negative value, change the greater than to a less than when testing the x position, and test the y value instead of the z for the bottom limit

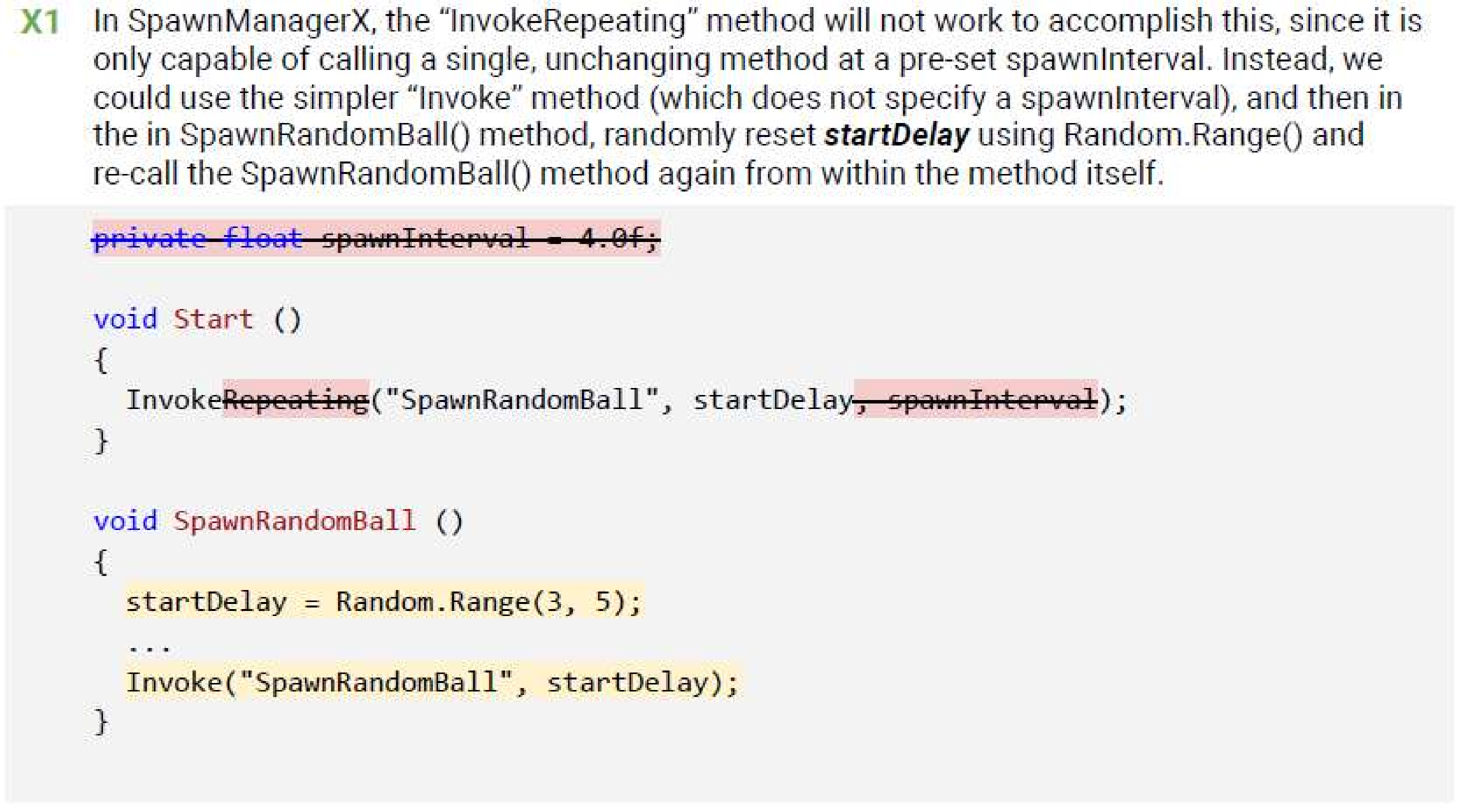
```
private float leftLimit = -30;
private float bottomLimit = -5;

void Update() {
  if (transform.position.x ~~>~~ < leftLimit) {
    Destroy(gameObject);
  } else if (transform.position.~~z~~ y < bottomLimit) {
    Destroy(gameObject);
  }
}
```



5 In the SpawnRandomBall() method, declare a new random ***int index*** variable between 0 and the length of the Array, then incorporate that index variable into the the Instantiate call

```
void SpawnRandomBall ()
{
  // Generate random ball index and random spawn position
  int index = Random.Range(0, ballPrefabs.Length);
  Vector3 spawnPos = new Vector3(Random.Range(spawnXLeft, spawnXRight), spawnPosY, 0);

  // instantiate ball at random spawn location
  Instantiate(ballPrefabs[~~0~~index], spawnPos, ballPrefabs[~~0~~index].transform.rotation);
}
```

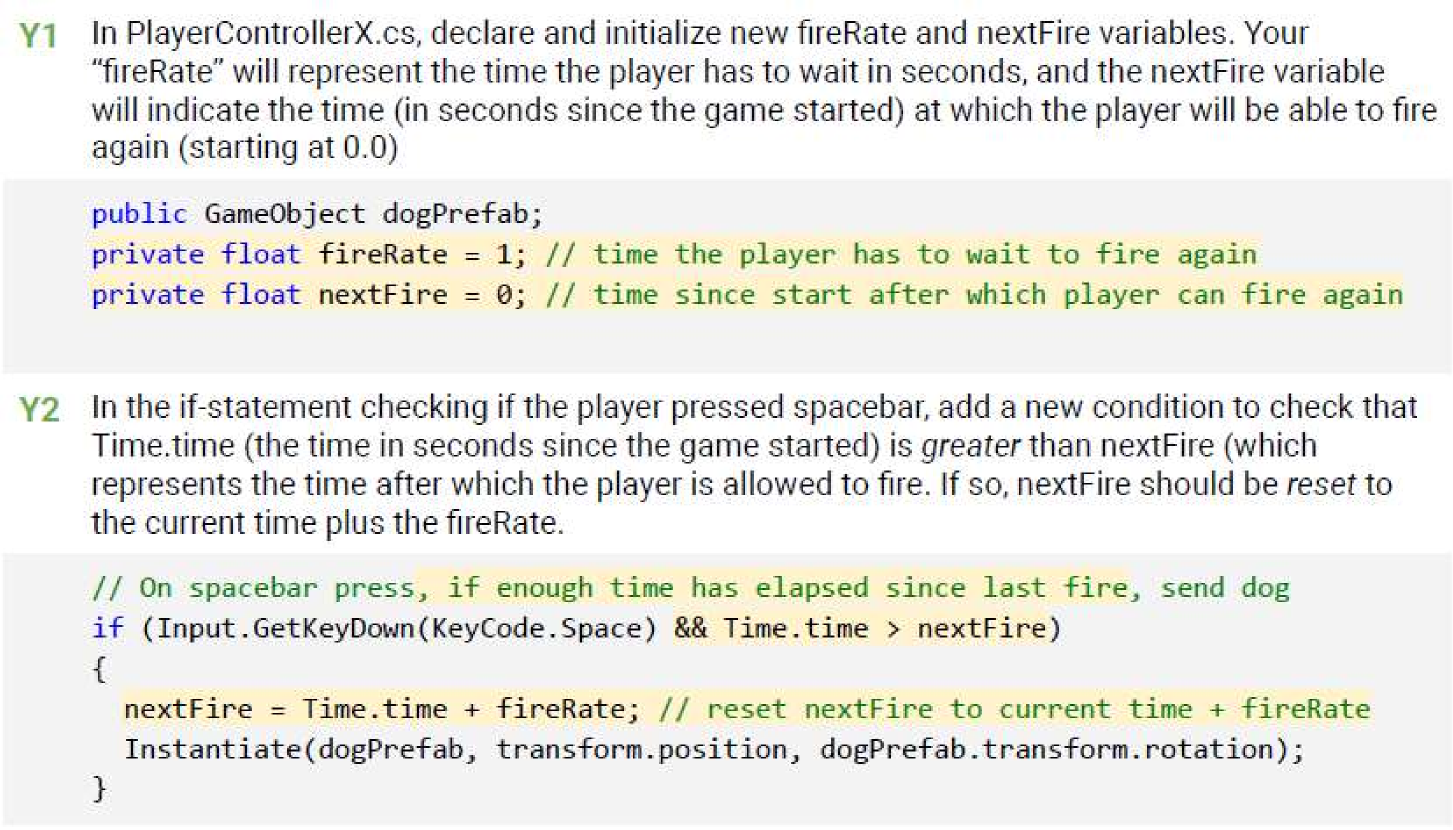
SOLUTION



X1 In SpawnManagerX, the “InvokeRepeating” method will not work to accomplish this, since it is only capable of calling a single, unchanging method at a pre-set spawnInterval. Instead, we could use the simpler “Invoke” method (which does not specify a spawnInterval), and then in the in SpawnRandomBall() method, randomly reset ***startDelay*** using Random.Range() and re-call the SpawnRandomBall() method again from within the method itself.

```
~~private float spawnInterval = 4.0f;~~

void Start ()
{
  Invoke~~Repeating~~("SpawnRandomBall", startDelay~~, spawnInterval~~);
}

void SpawnRandomBall ()
{
  startDelay = Random.Range(3, 5);
  ...
  Invoke("SpawnRandomBall", startDelay);
}
```

SOLUTION



Y1 In PlayerControllerX.cs, declare and initialize new fireRate and nextFire variables. Your "fireRate" will represent the time the player has to wait in seconds, and the nextFire variable will indicate the time (in seconds since the game started) at which the player will be able to fire again (starting at 0.0)

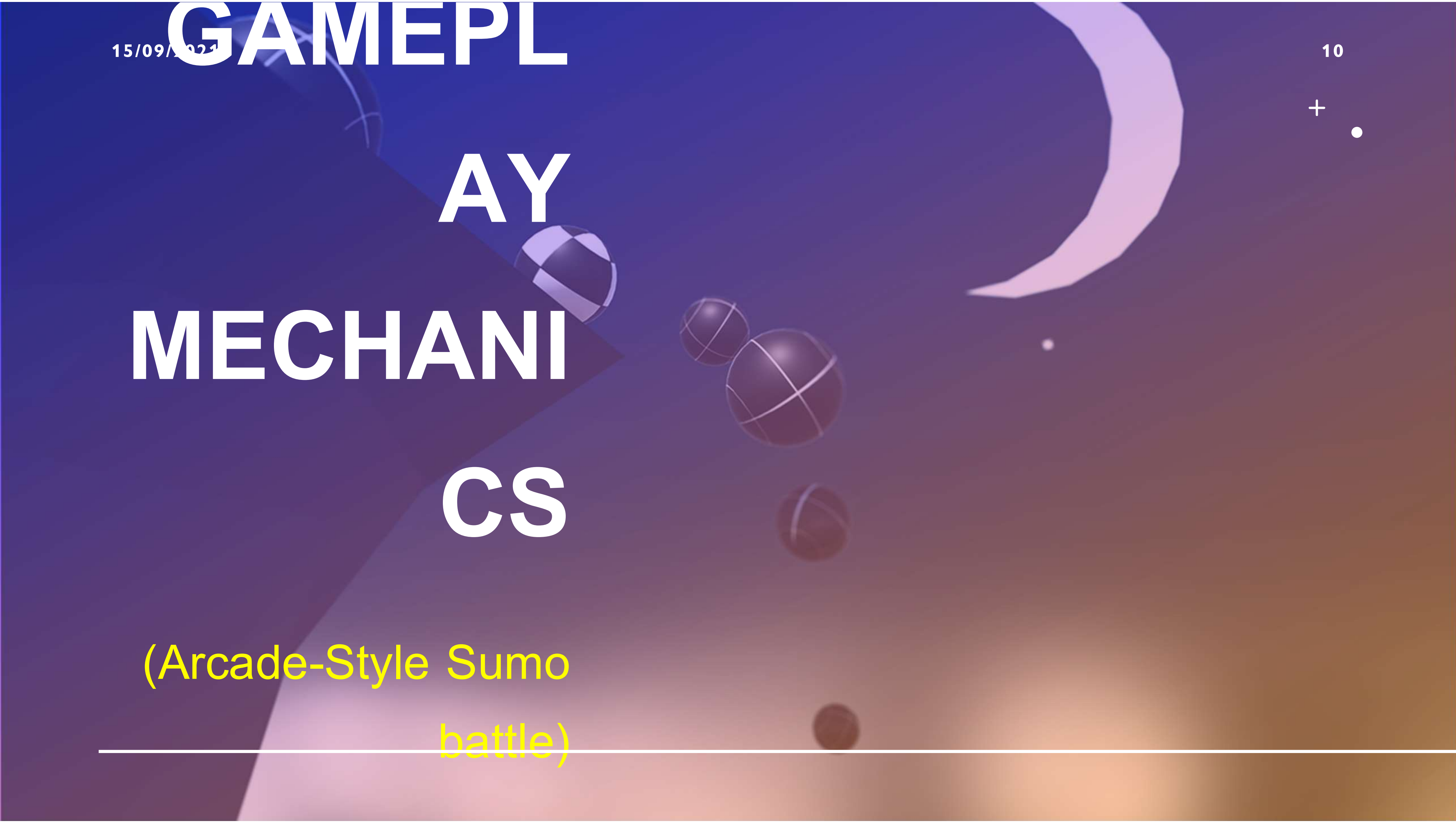
```
public GameObject dogPrefab;
private float fireRate = 1; // time the player has to wait to fire again
private float nextFire = 0; // time since start after which player can fire again
```

Y2 In the if-statement checking if the player pressed spacebar, add a new condition to check that Time.time (the time in seconds since the game started) is *greater* than nextFire (which represents the time after which the player is allowed to fire. If so, nextFire should be *reset* to the current time plus the fireRate.

```
// On spacebar press, if enough time has elapsed since last fire, send dog
if (Input.GetKeyDown(KeyCode.Space) && Time.time > nextFire)
{
  nextFire = Time.time + fireRate; // reset nextFire to current time + fireRate
  Instantiate(dogPrefab, transform.position, dogPrefab.transform.rotation);
}
```

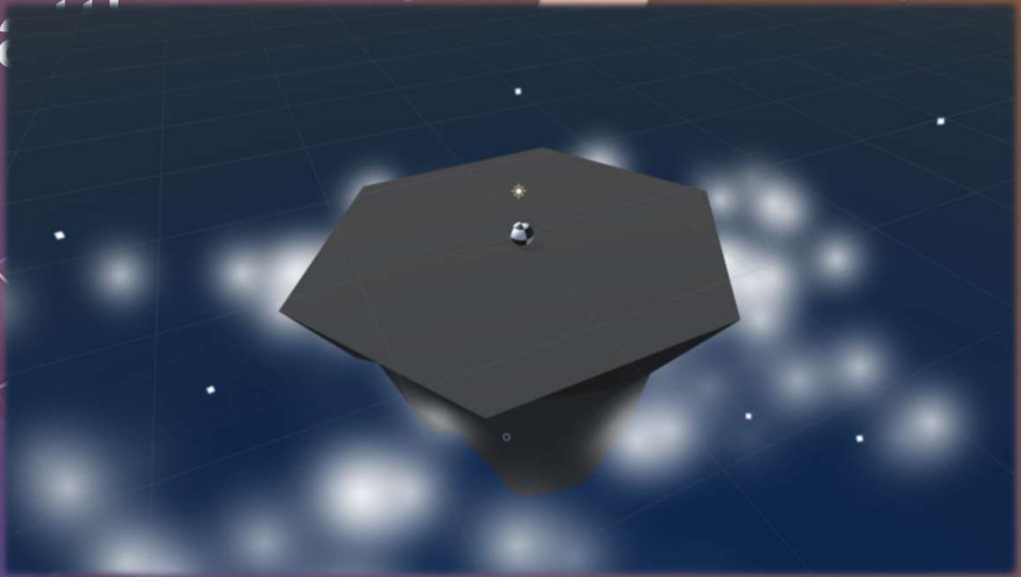
GAMEPLAY MECHANICS

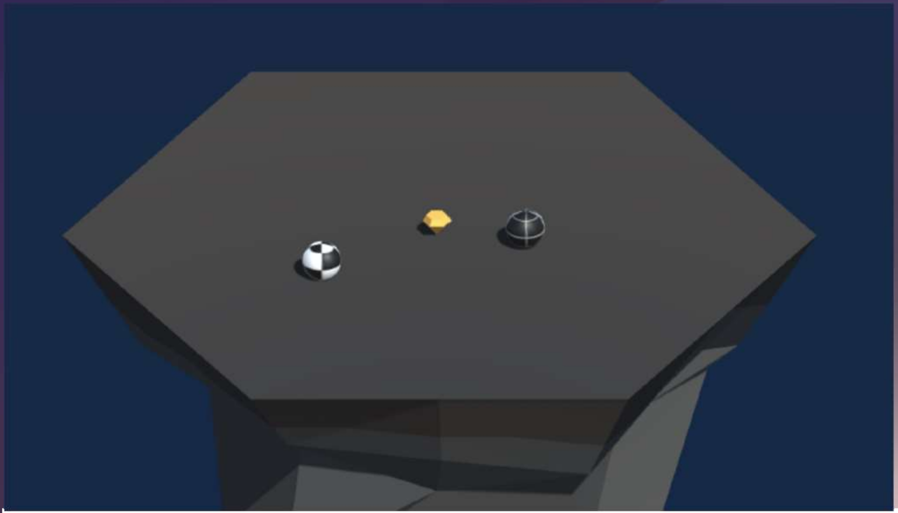
(Arcade-Style Sumo battle)



Unit 4 – Gameplay Mechanics

- Arcade-Style Sumo ba
  - Watch Where You're Going
  - Follow the Player
  - PowerUp and CountDown
  - For-Loops For Waves





PowerUp and CountDown – เพิ่มพลังและนับถอยหลัง

PowerUp and CountDown

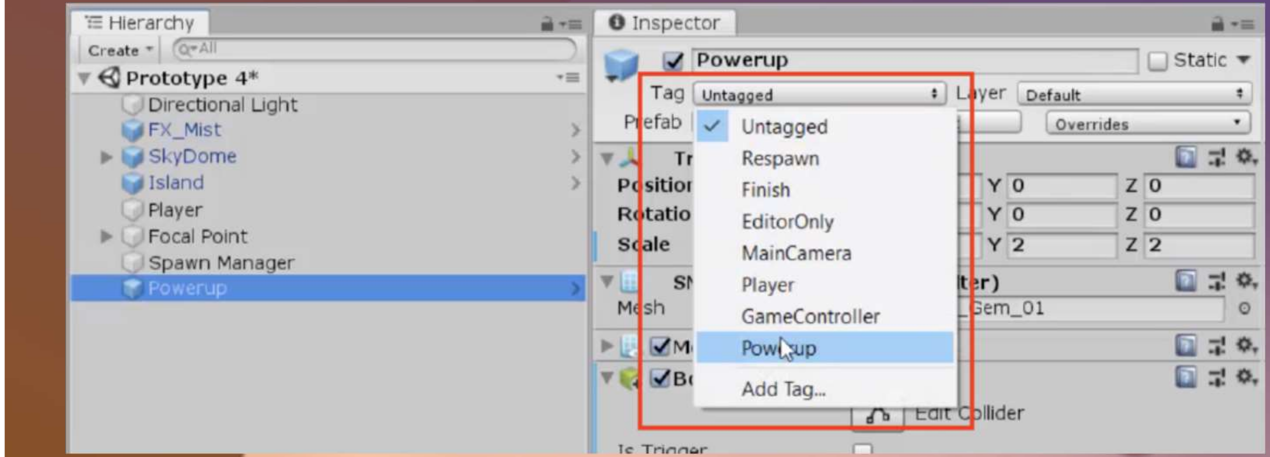
- Step 1 : Choose and prepare a powerup
- Step 2 : Destroy powerup on collision
- Step 3 : Test for collision with a powerup
- Step 4 : Apply extra knockback with powerup
- Step 5 : Create Countdown Routine for powerup
- Step 6 : Add a powerup indicator

Powerup and CountDown – 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”

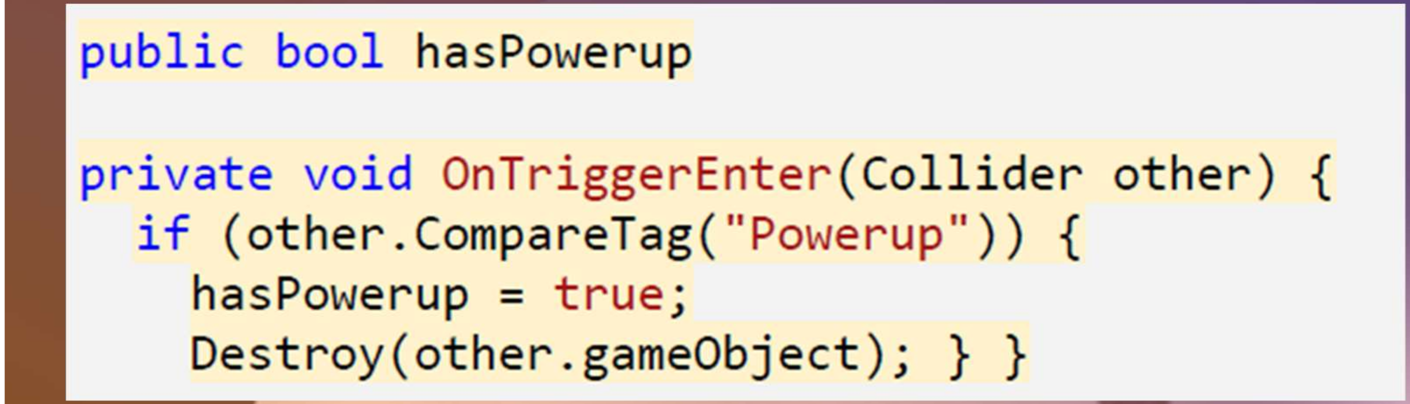
Warning : Remember, you still have to apply the tag after it has been created.



Powerup and CountDown – 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



```
public bool hasPowerup

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

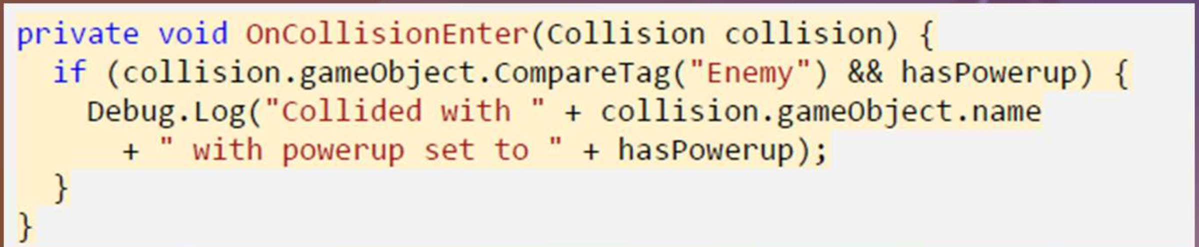
Don't worry : If this doesn't work, make sure that the Powerup's collider "Is trigger" and player's collider is NOT

Tip : Make sure hasPowerup = true in the inspector when you collide

and powerup

The powerup will only come in to 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



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

Tip : OnTriggerEnter is good for stuff like picking up powerups, but you should use OnCollisionEnter when you want something to do with physics

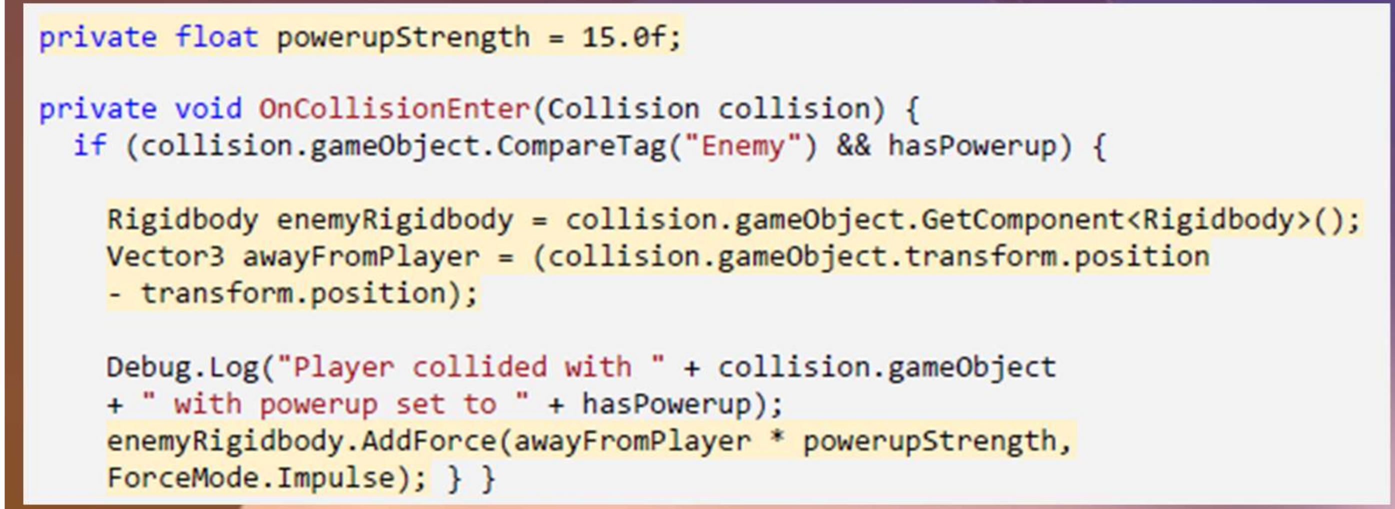
New Concept : Concatenation in Debug
Messages

Tip : When you concatenate a variable in a debug message, it will returns its VALUE not its name

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



```
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); } }
```

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

PowerUp and CountDown - Step 5 : Create Countdown Routine for powerup

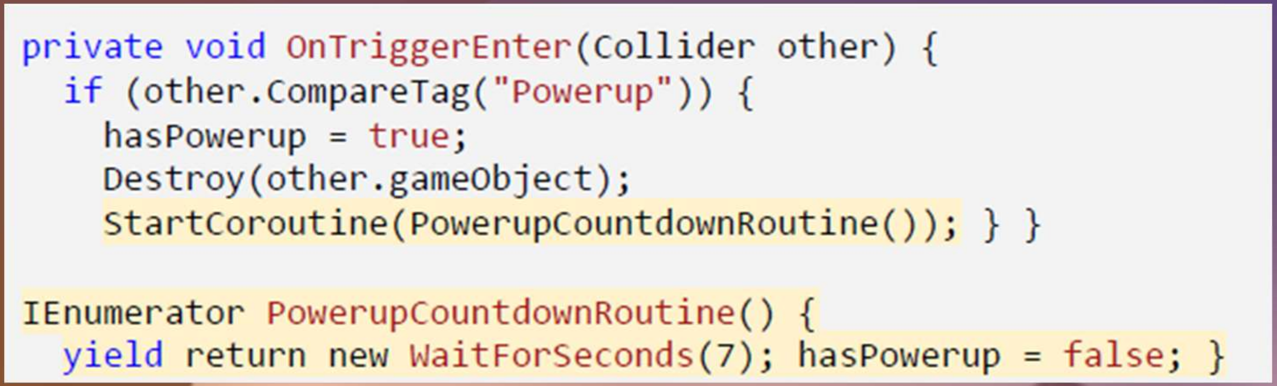
It wouldn't be fair to the enemies if the powerup lasted forever - so we'll program a countdown timer that starts when the player collects the powerup, removing the powerup ability when the timer is finished.

New Concept: IEnumerator

New Concept: Coroutines

Tip: WaitForSeconds()

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



```
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; }
```

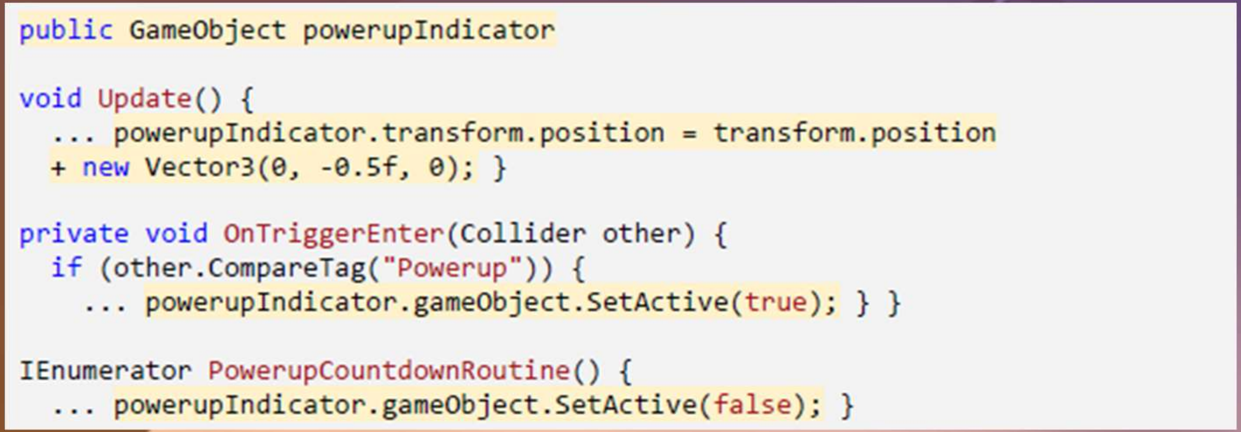
Powerup and CountDown – Step 6 : Add a powerup indicator

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.

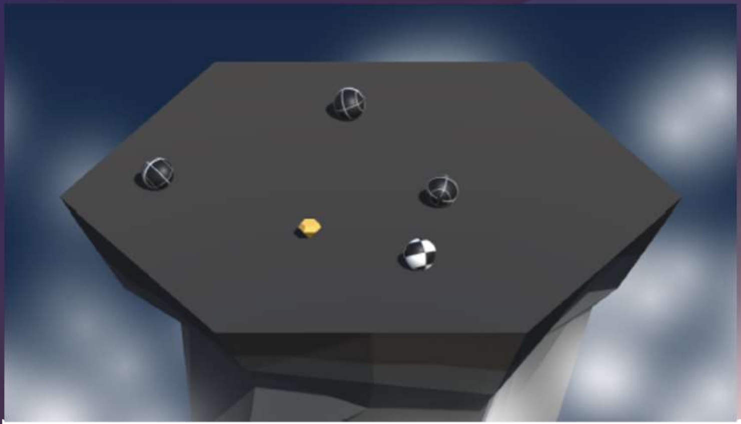
1. From the Library, drag a Powerup object into the scene, rename it "Powerup Indicator", and edit its scale
2. Uncheck the "Active" checkbox in the inspector
3. In PlayerController.cs, declare a new public GameObject powerupIndicator variable, then assign the Powerup Indicator variable in the inspector
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); }
```



For-Loops For Waves – วนเวียนเรียงหน้ากันเข้ามาได้เลย

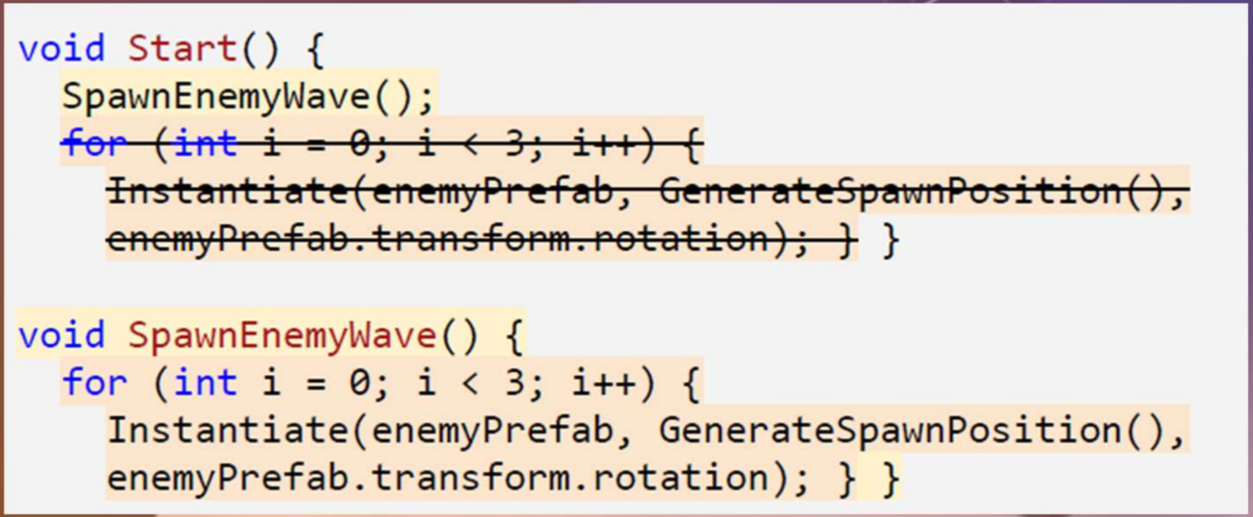
PowerUp and CountDown

- 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

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()



```
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); } }
```

New Concept : For-loops

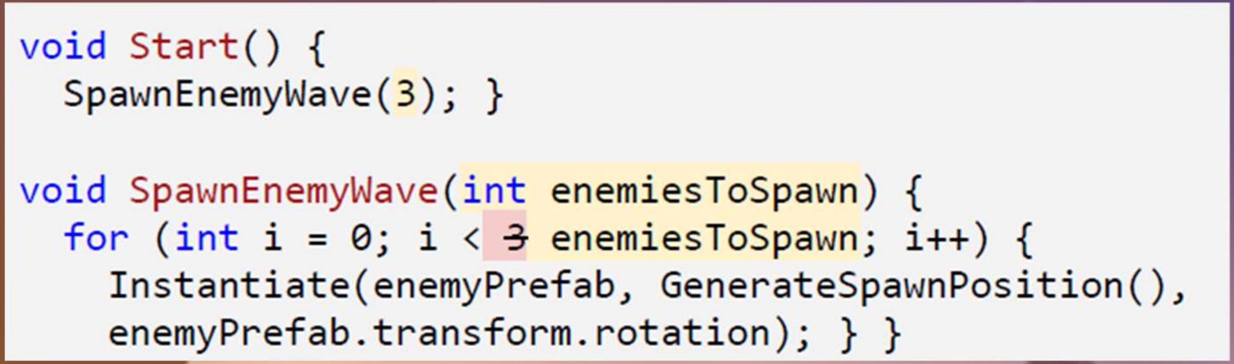
Don't worry : Loops are a bit confusing at first, but they make sense eventually. Loops are powerful tools that programmers use often

New Concept : ++ Increment Operator

PowerUp and CountDown – 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(___);



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

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

New Concept : Custom methods with parameters

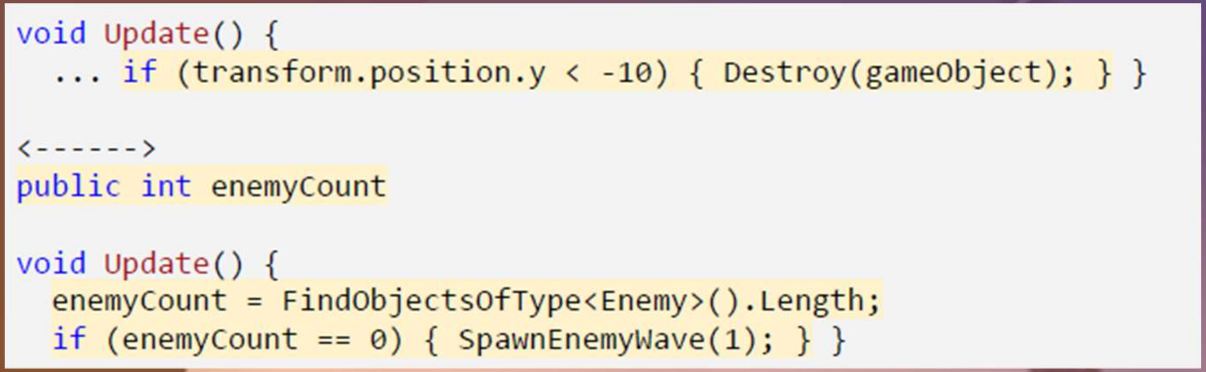
Tip : GenerateSpawnPosition returns a value, SpawnEnemyWave does not. SpawnEnemyWave takes a parameter, GenerateSpawnPosition does not.

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



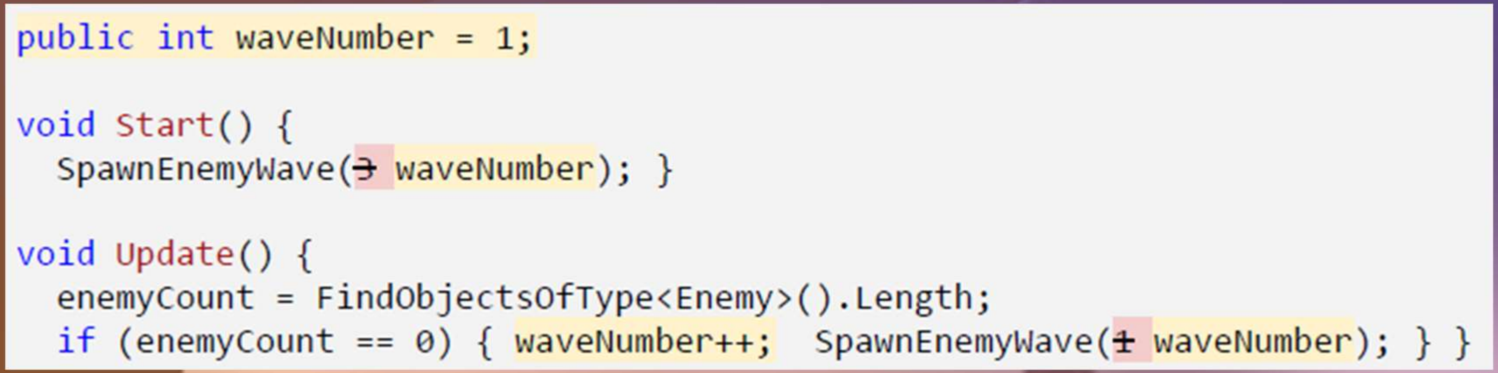
```
void Update() {
  ... if (transform.position.y < -10) { Destroy(gameObject); } }

<------>
public int enemyCount

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

PowerUp and CountDown – Step 7 : 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.

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

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



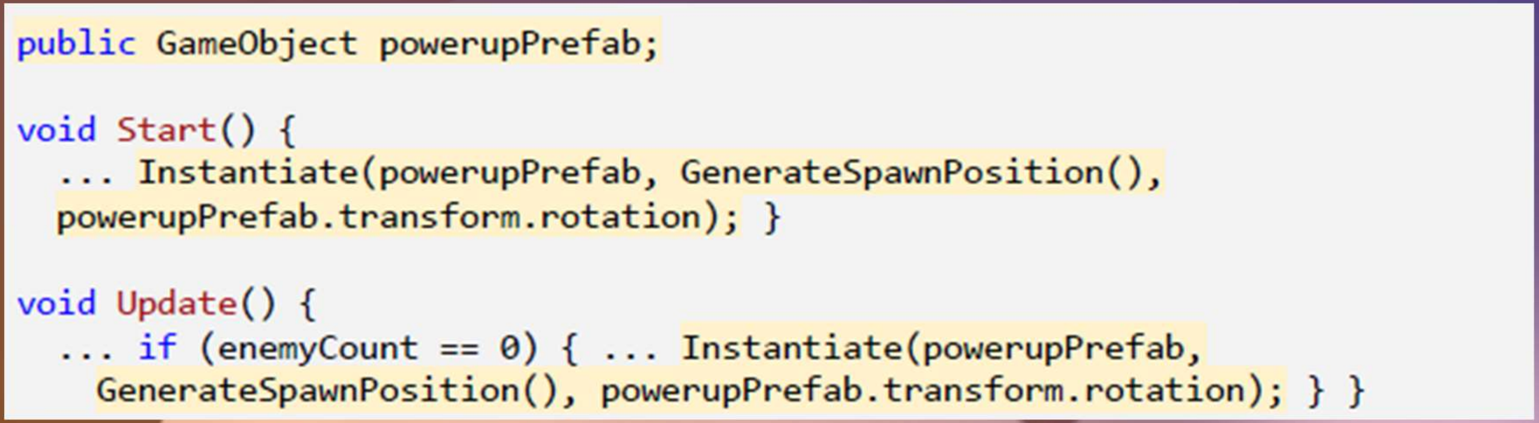
```
public int waveNumber = 1;

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

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

Powerup and CountDown – Step 6 : 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



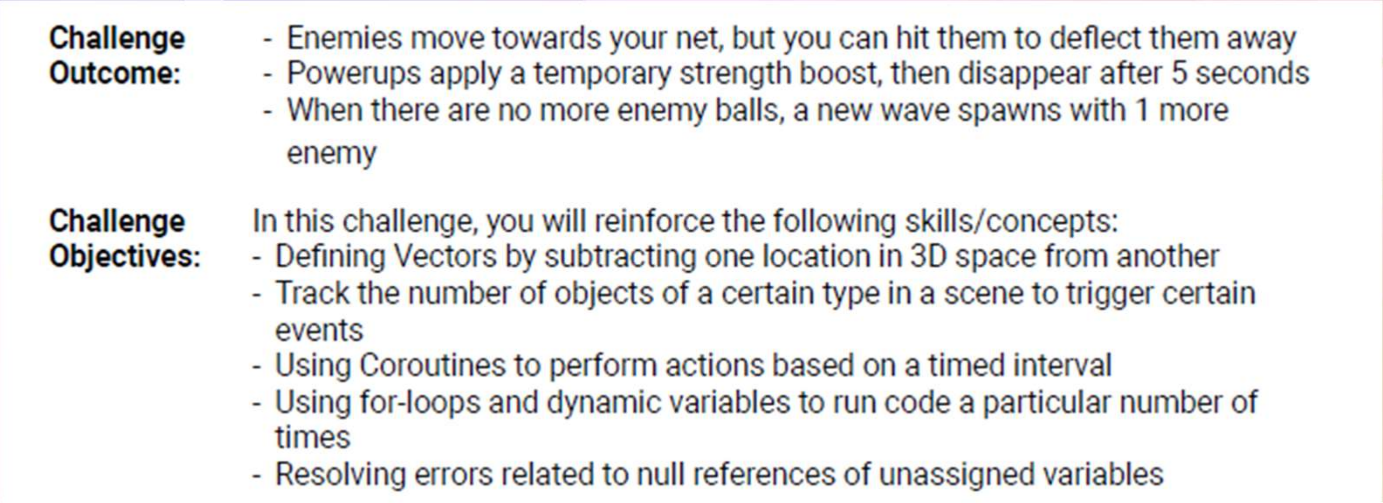
```
public GameObject powerupPrefab;

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

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

CHALLENGE 4

Soccer Scripting

CHALLENGE 4

Soccer Scripting



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

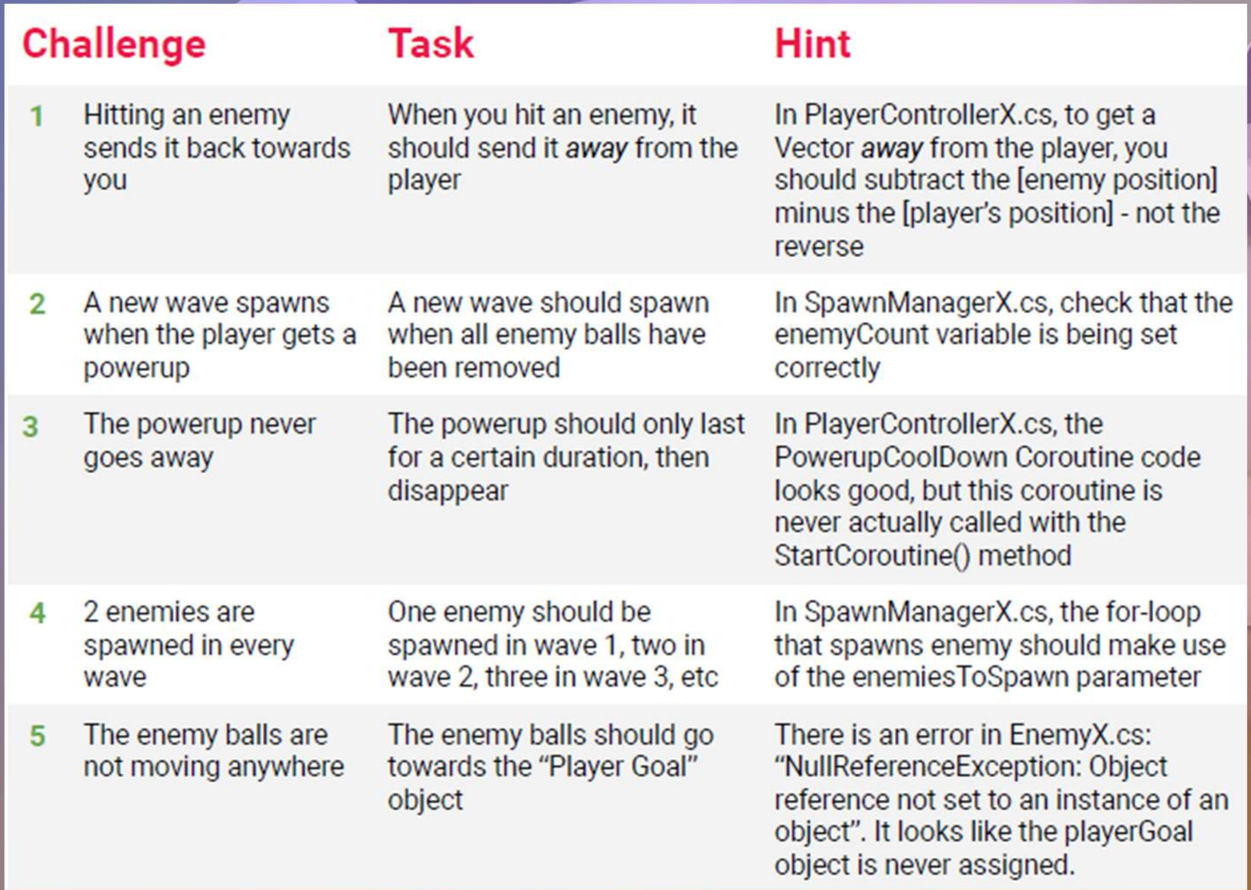
Challenge Objectives:

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 4

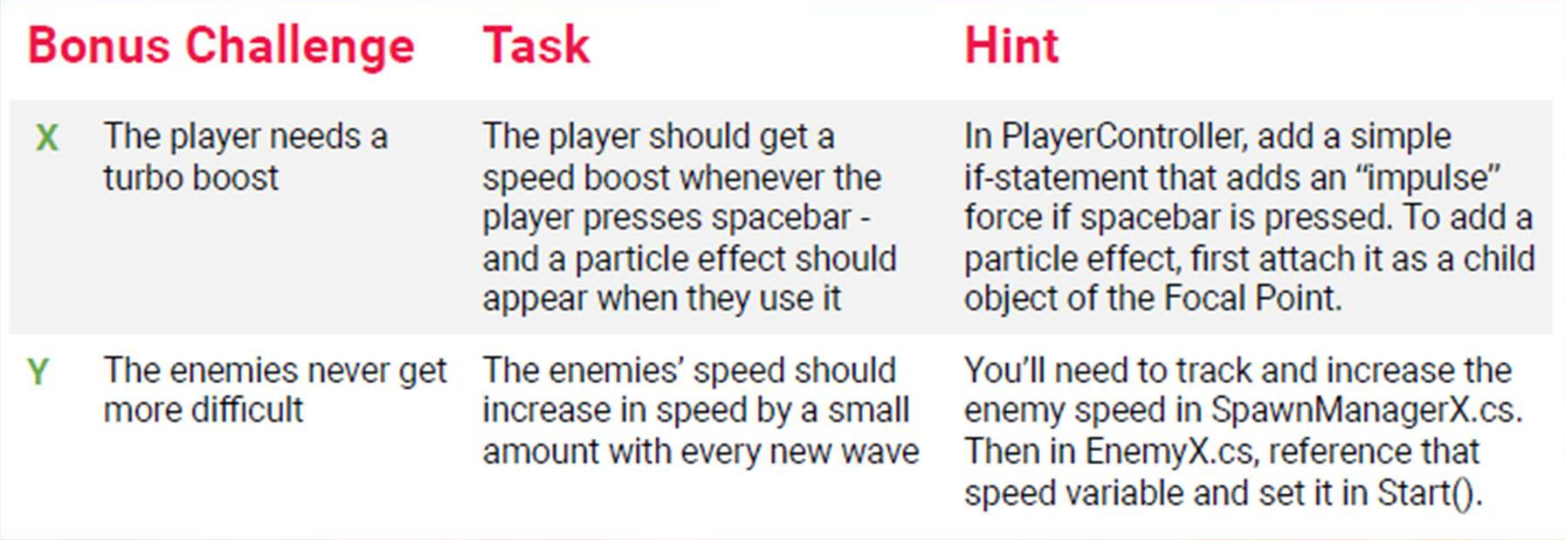
Soccer Scripting



	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.

CHALLENGE 4

Soccer Scripting



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().