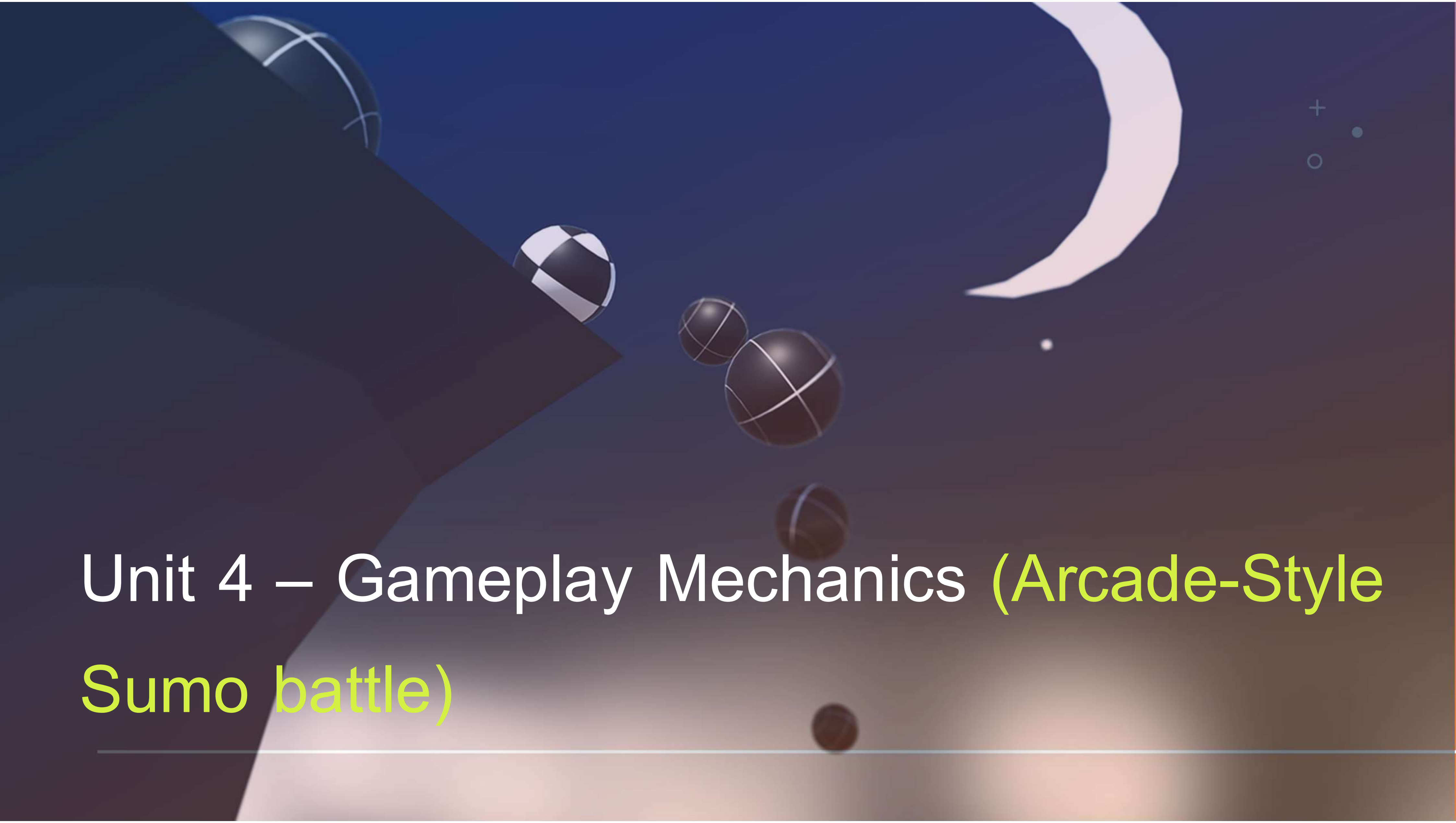
การเขียนโปรแกรมคอมพิวเตอร์ขั้นสูงเพื่อควบคุมอุปกรณ์

Advance Computer Programming

[สัปดาห์ที่ 8]

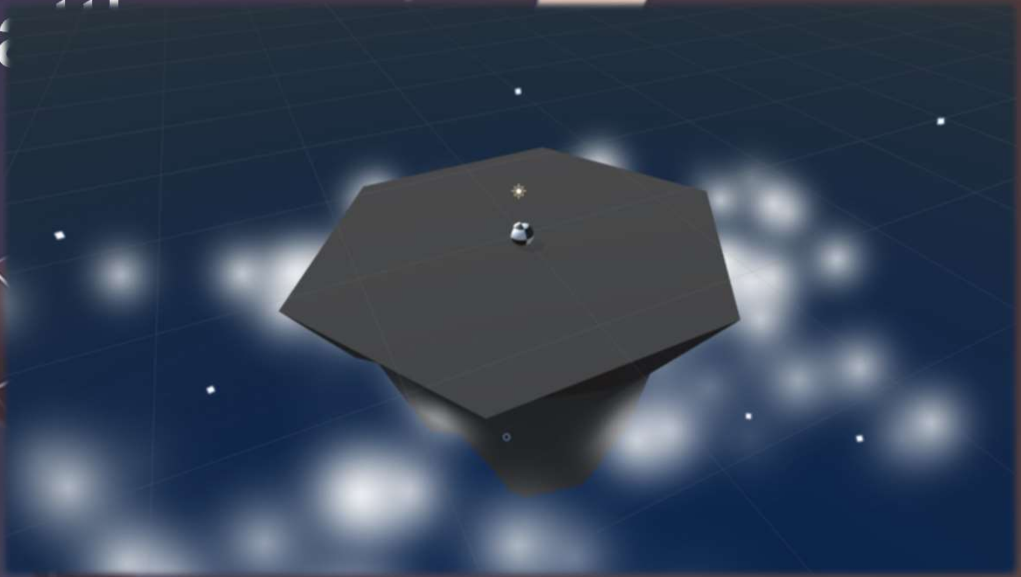
สอนโดย พงศธร เกียรติเจริญพร (มิว)

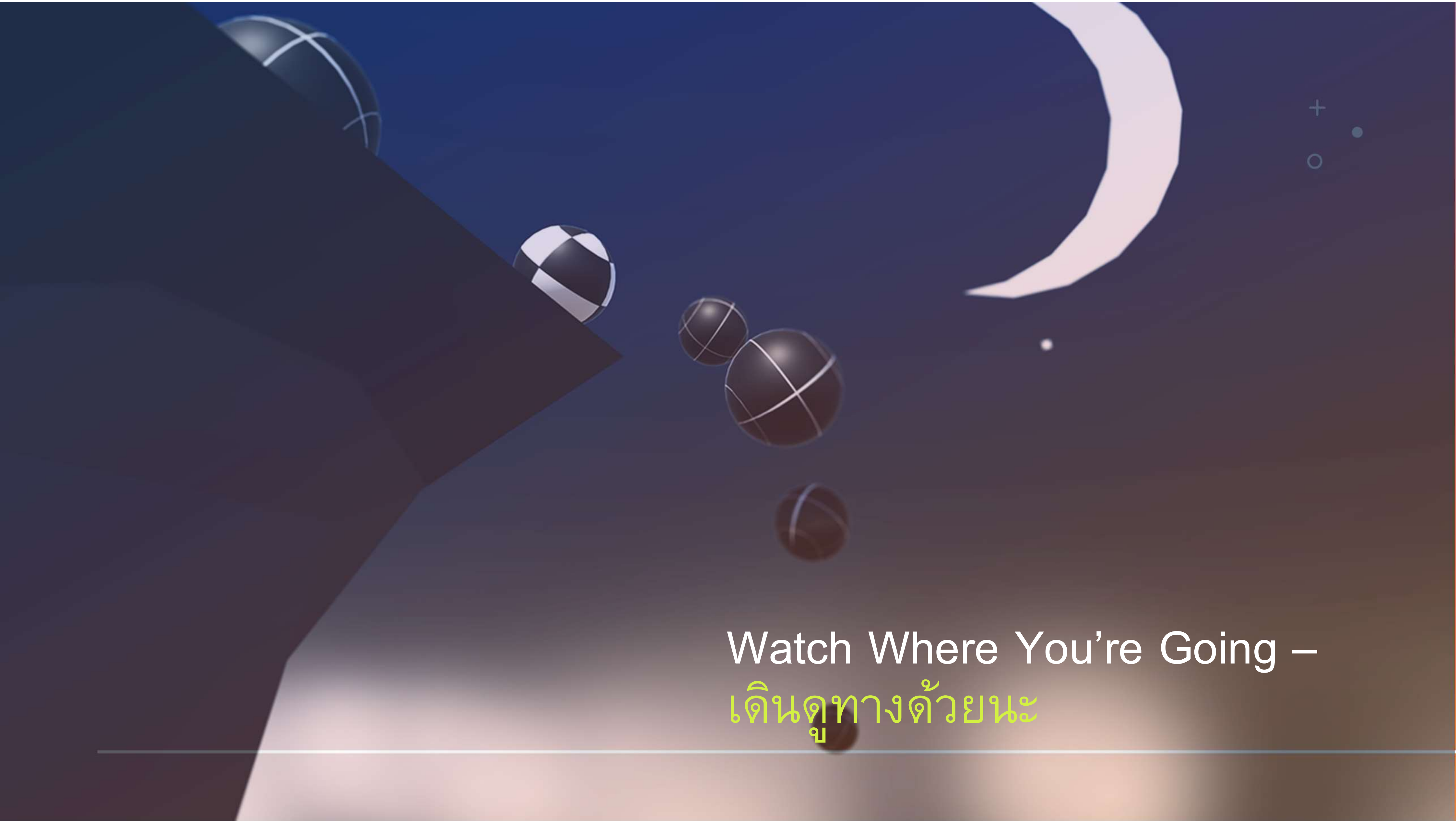


Unit 4 – Gameplay Mechanics (Arcade-Style Sumo battle)

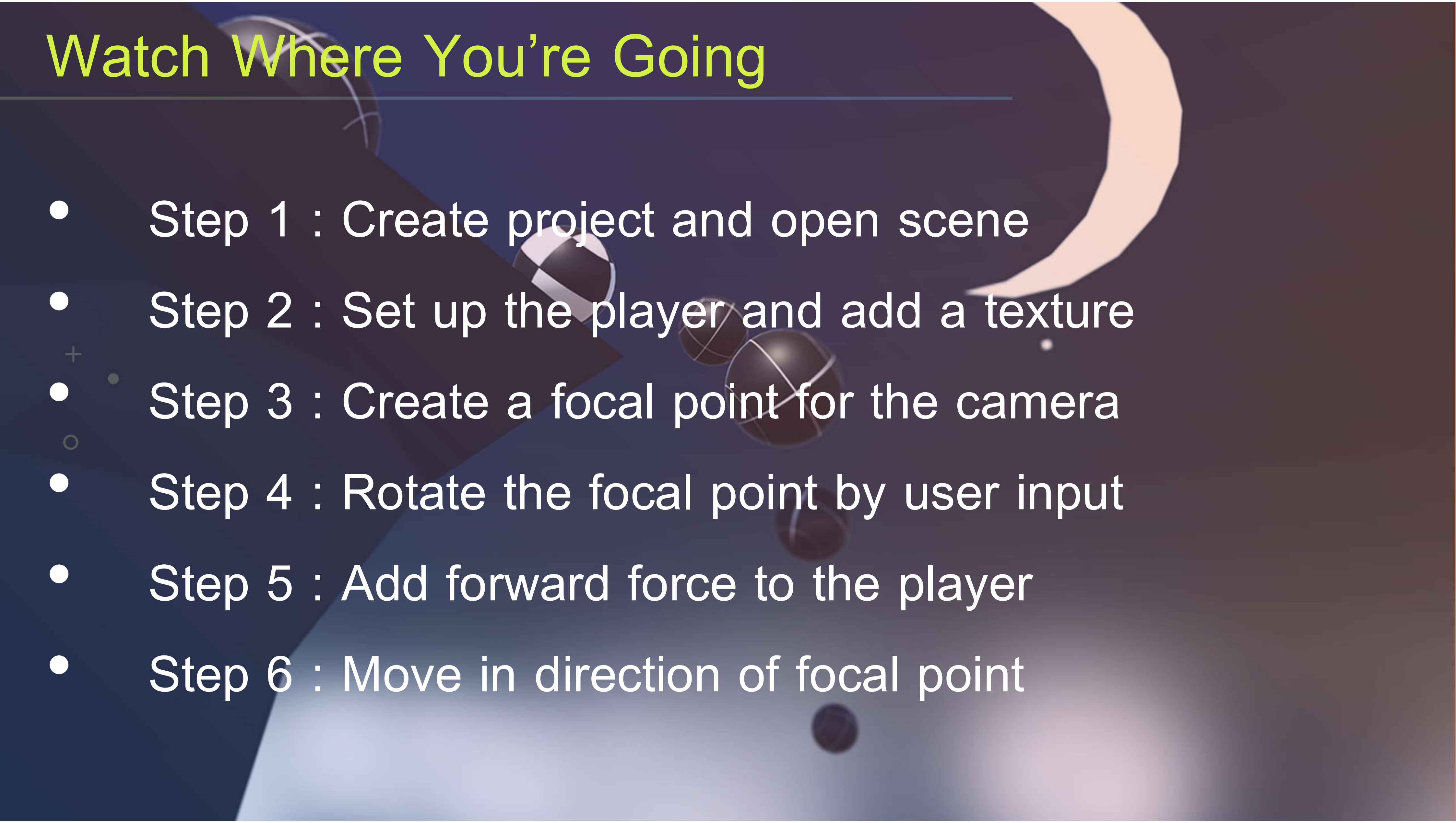
Unit 4 – Gameplay Mechanics

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





Watch Where You’re Going – เดินดูทางด้วยนะ



Watch Where You’re Going

- 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

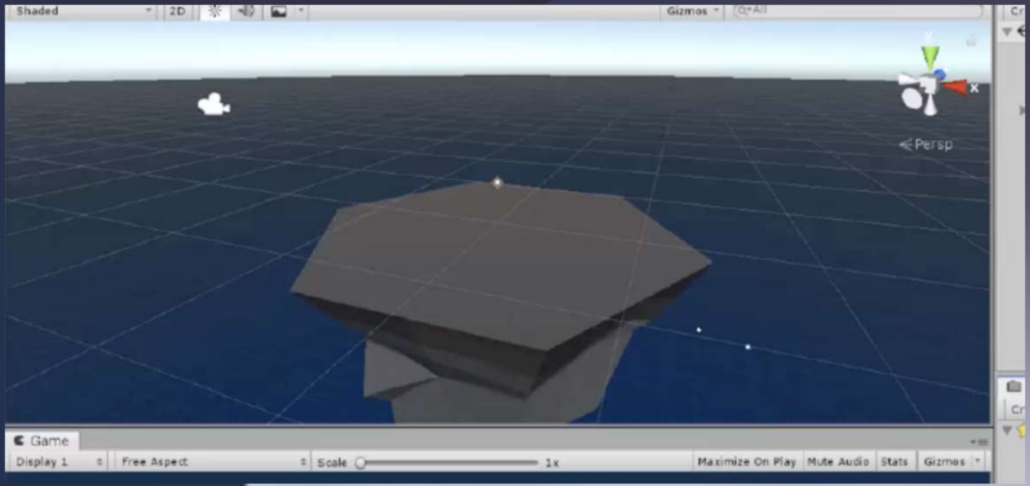
and open scene

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.
2. download the Prototype 4 Starter Files, **extract** the compressed folder, and then import the .unitypackage into your project.
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

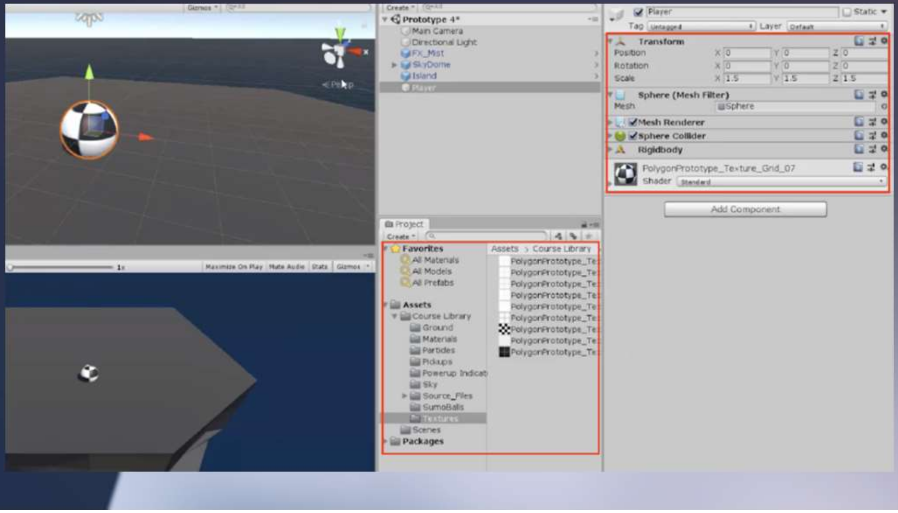


player and add a texture

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 wra



☑ Player □ Static

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0

Rotation X 0 Y 0 Z 0

Scale X 1.5 Y 1.5 Z 1.5

Sphere (Mesh Filter)

Mesh Sphere

☑ Mesh Renderer

☑ Sphere Collider

Rigidbody

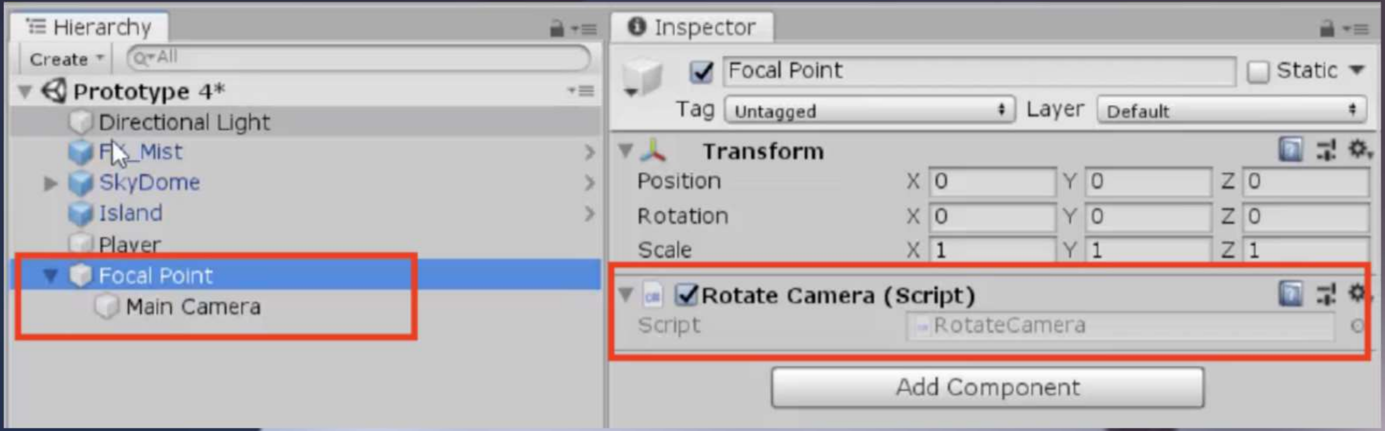
PolygonPrototype_Texture_Grid_07

Shader Standard

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



Prototype 4*
Directional Light
FX_Mist
SkyDome
Island
Player
Focal Point
Main Camera

☑ Focal Point □ Static

Tag Untagged Layer Default

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

☑ Rotate Camera (Script)

Script RotateCamera

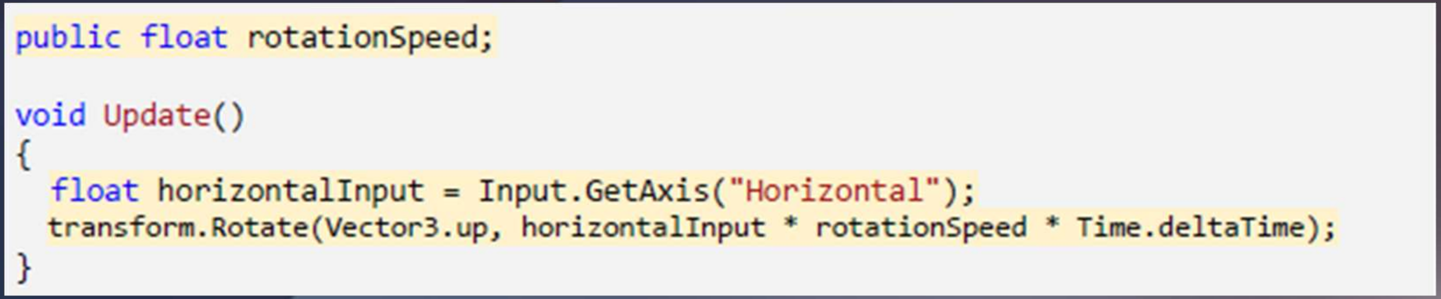
- Don’t worry : This whole “focal point” business may be confusing at first, but it will make sense once you see it in action
- Tip : Try rotating the Focal point around the Y axis and see the camera rotate in scene view

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);
}
```

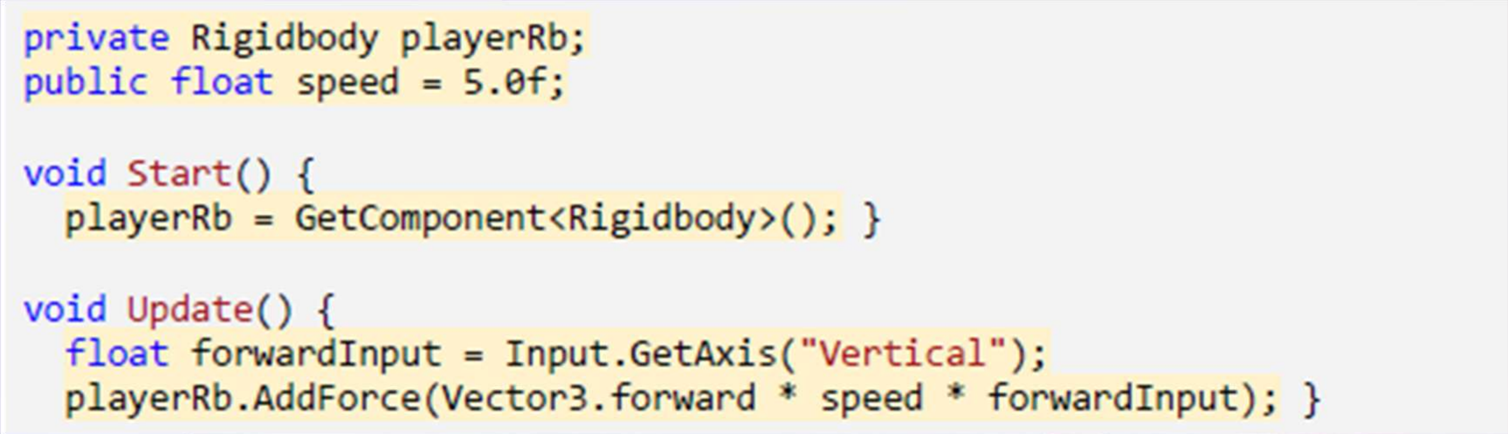
force to the player

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); }
```

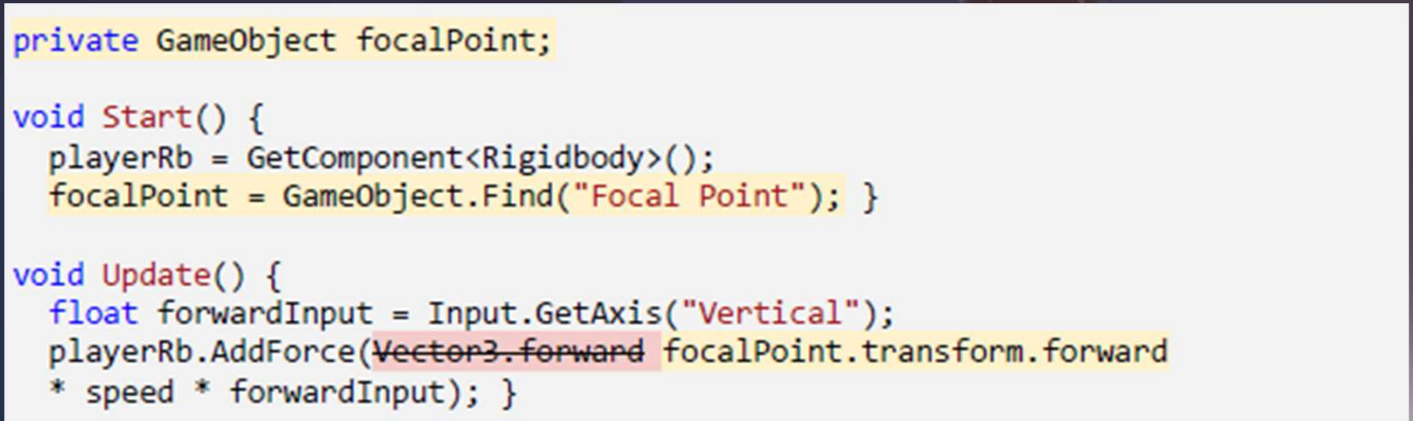
direction of focal point

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.

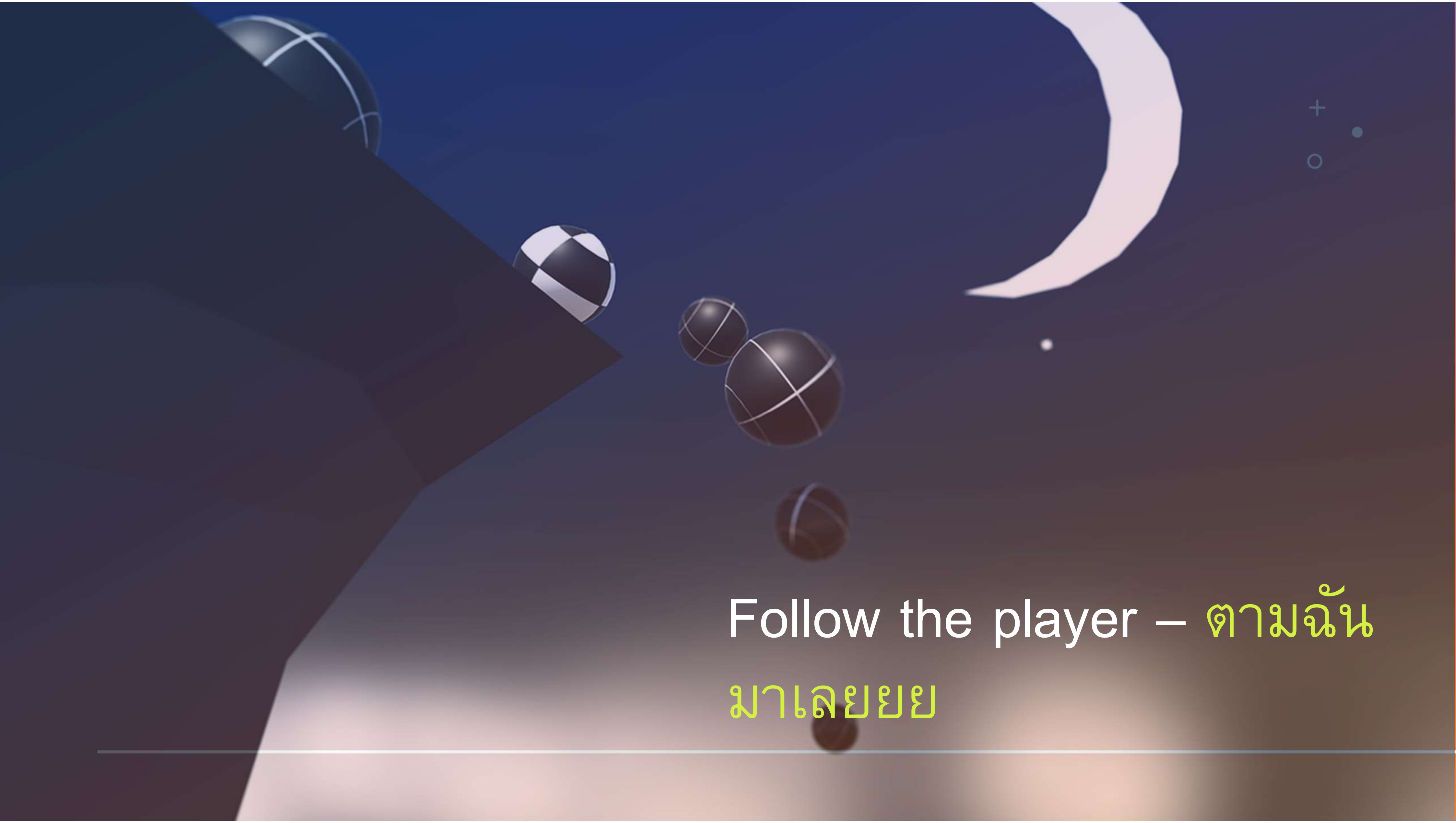
1. Declare a new private GameObject focalPoint; and initialize it in Start(): focalPoint = GameObject.Find("Focal Point");
2. In the AddForce call, Replace Vector3.forward with focalPoint.transform.forward

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); }
```

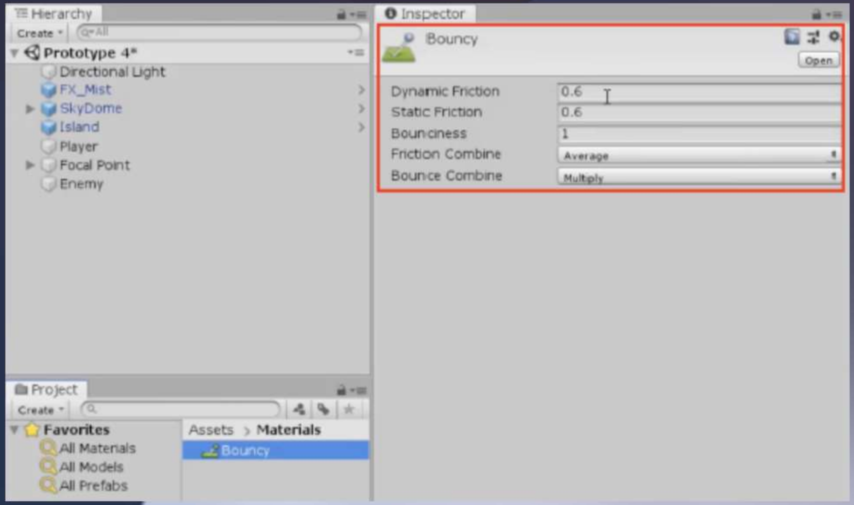


Follow the player – ตามฉันมาเลยยย

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



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

Don't worry: If your game is lagging, uncheck the "Active" checkbox for your clouds

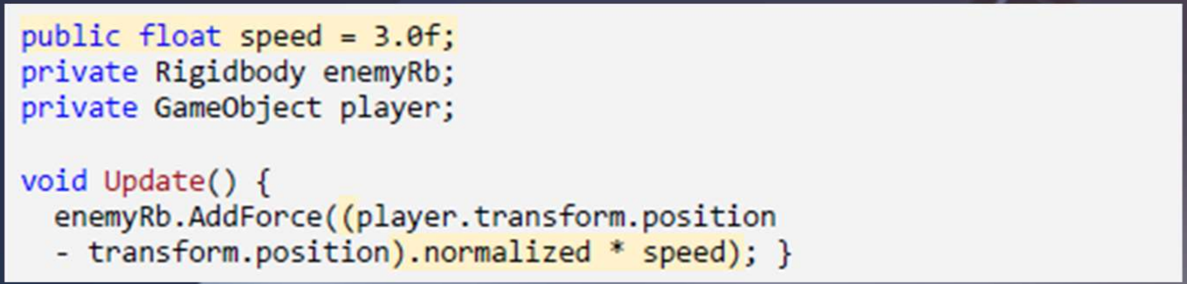
New Concept : Physics Materials

New Concept : Bounciness property and Bounce Combine

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



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

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

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?

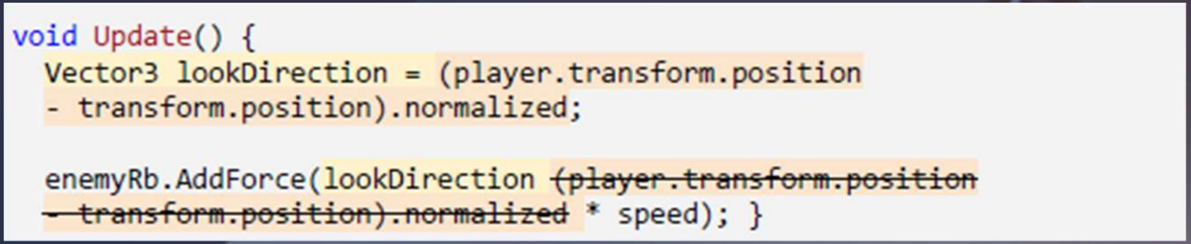
Tip : When normalized, a vector keeps the same direction but its length is 1.0, forcing the enemy to try and keep up

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

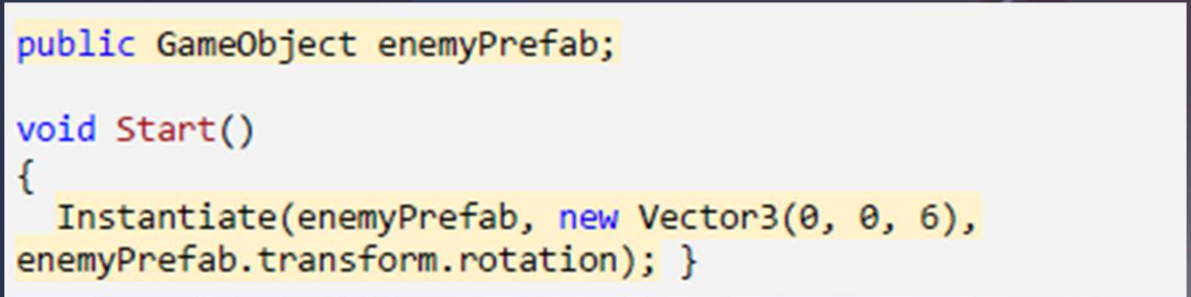


```
void Update() {
  Vector3 lookDirection = (player.transform.position
  - transform.position).normalized;

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

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



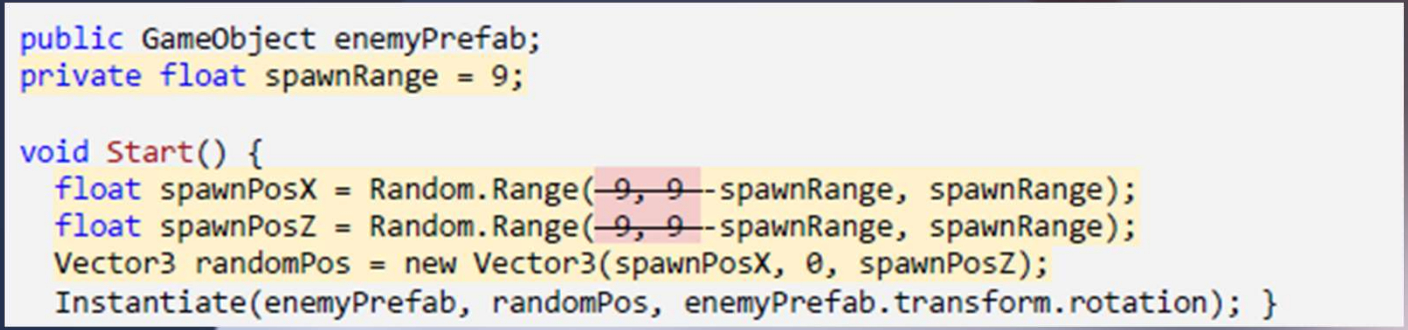
```
public GameObject enemyPrefab;

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

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); }
```

spawn point

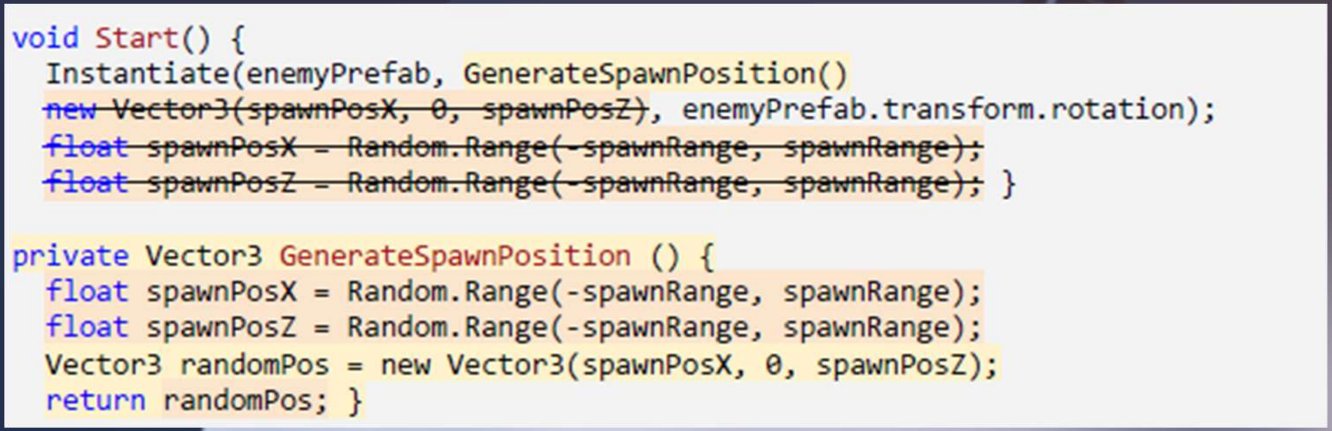
The code we use to generate a random spawn position is perfect, and we're going to be using it a lot. If we want to clean the script and use this code later down the road, we should store it in a custom function.

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; }
```

- Gameplay Mechanics
  - Watch Where You’re Going
  - Follow the Player
  - PowerUp and CountDown
  - For-Loops For Waves

To Be Continue..