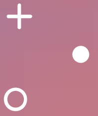
การเขียนโปรแกรมคอมพิวเตอร์ขั้นสูงเพื่อควบคุมอุปกรณ์

Advance Computer Programming

[สัปดาห์ที่ 6]

สอนโดย พงศธร เกียรติเจริญพร (มิว)

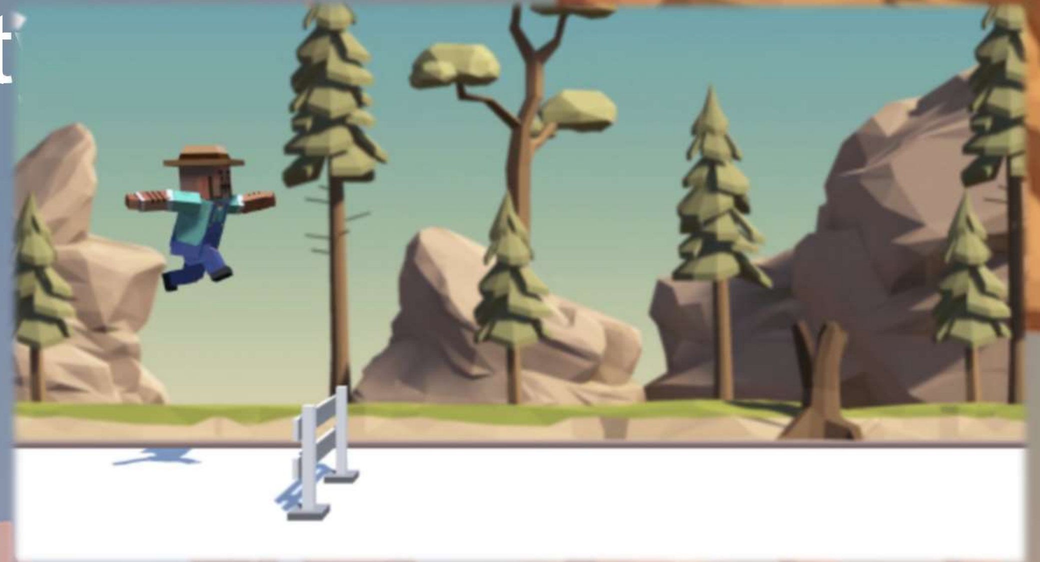


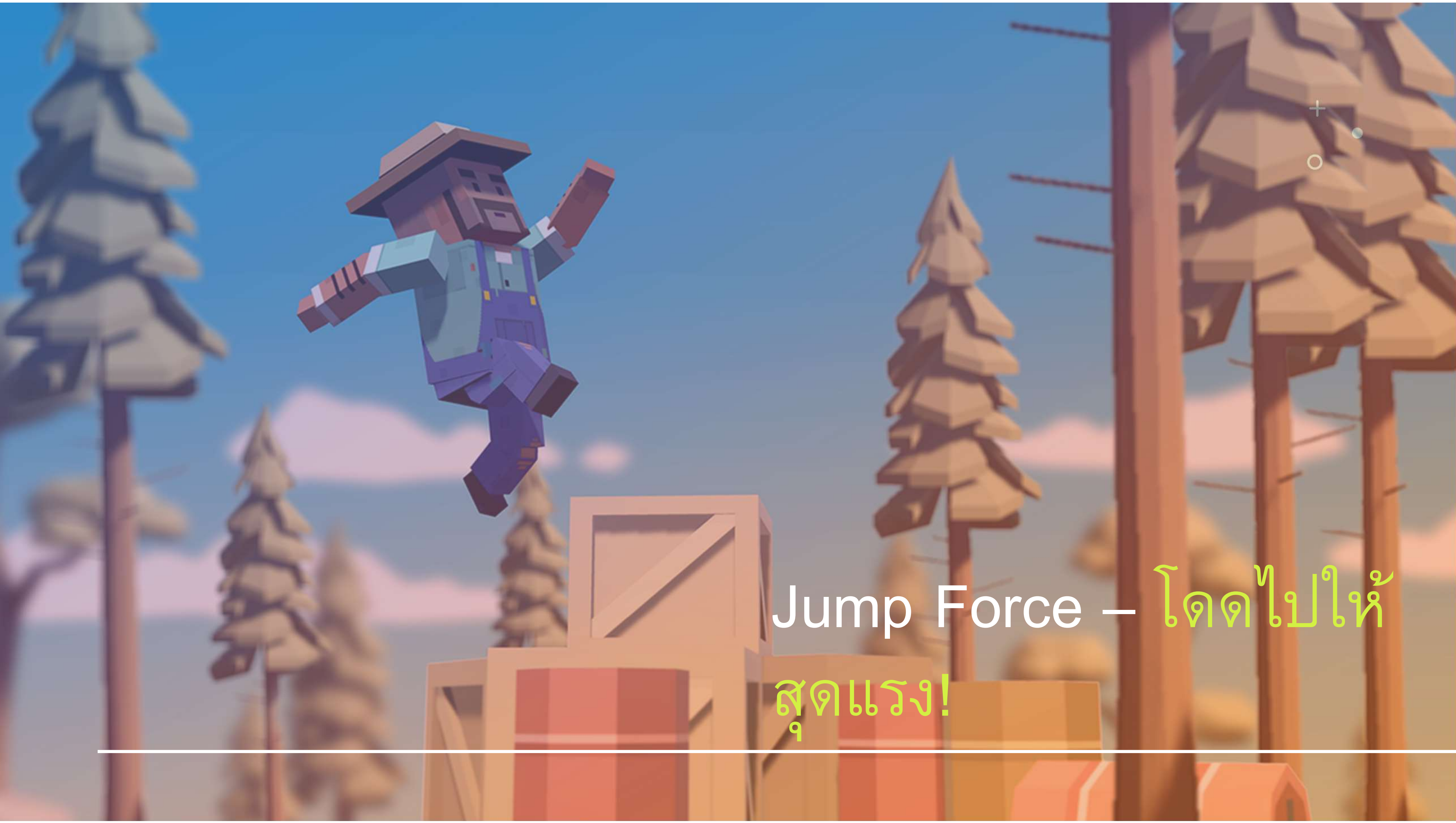
Update Progress - มาโชว์ผลงานกัน

Unit 3 – Sound and Effects (Run and Jump Prototype)

Unit 3 – Animation, Sound and Effects

- Run and Jump Prototype
  - Jump Force
  - Make the World Whiz By
  - Don't Just Stand There
  - Particles and Sound Effects




Jump Force – โดดไปให้
สุดแรง!

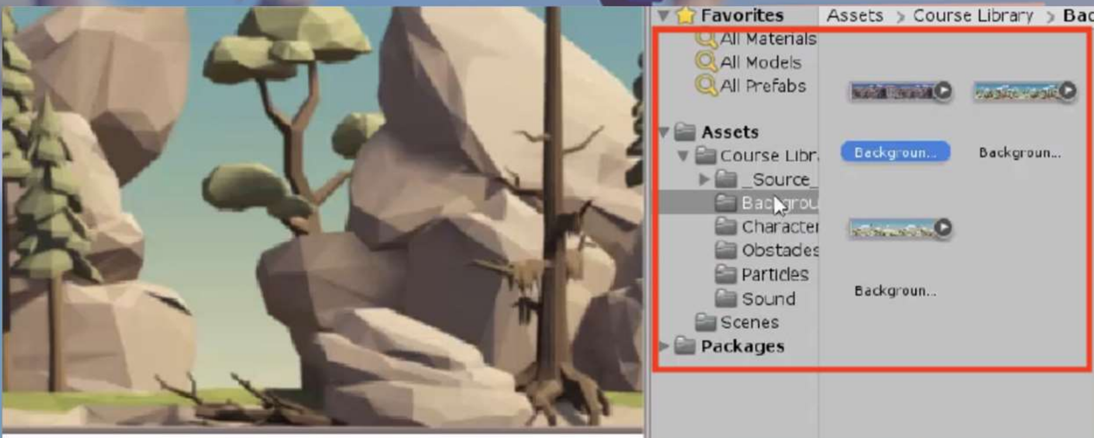
Jump Force

- Step 1 : Open prototype and change background
- Step 2 : Choose and set up a player character
- Step 3 : Make player jump at start
- Step 4 : Make player jump if spacebar pressed
- Step 5 : Tweak the jump force and gravity
- Step 6 : Prevent player from double-jumping
- Step 7 : Make an obstacle and move it left
- Step 8 : Create a spawn manager
- Step 9 : Spawn obstacles at intervals

change background

The first thing we need to do is set up a new project, import the starter files, and choose a background for the game.

1. Open **Unity Hub** and create an empty "Prototype 3" project in your course directory on the correct Unity version.
2. Click to download the Prototype 3 Starter Files, **extract** the compressed folder, and then **import** the .unitypackage into your project.
3. Open the Prototype 3 scene and **delete** the Sample Scene without saving
4. Select the Background object in the hierarchy, then in the Sprite Renderer component > Sprite, select the _City, _Nature, or _Town image



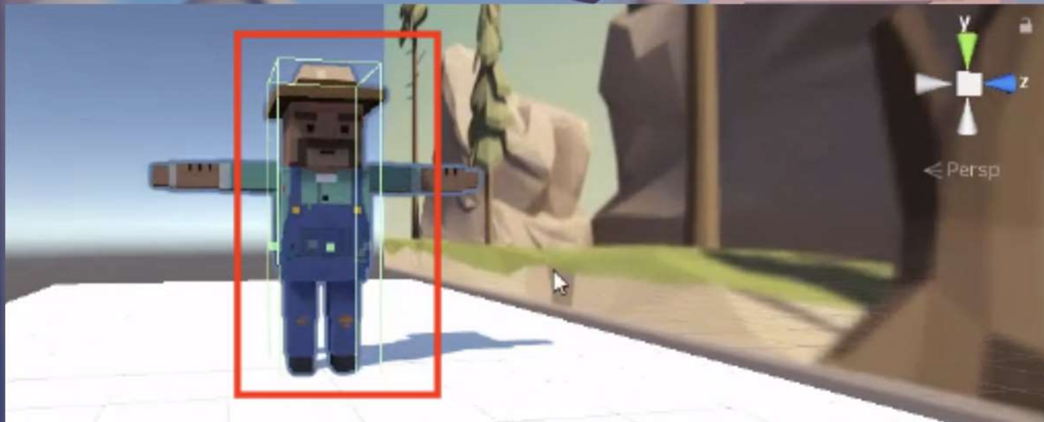
New Concept: Sprites / Sprite Renderer

Tip: Browse all of the Player and Background options before choosing either - some work better with others

player character

Now that we've started the project and chosen a background, we need to set up a character for the player to control.

1. From Course Library > Characters, Drag a character into the hierarchy, rename it "Player",
then rotate it on the Y axis to face to the right
2. Add a Rigid Body component
3. Add a box collider, then edit the collider bounds
4. Create a new "Scripts" folder in Assets, create a "PlayerController" script inside, and attach it to the player



Don't worry: We will get the player and the background moving soon

Tip: Use isometric view and the gizmos to cycle around and edit the collider with a clear perspective

Warning: Keep isTrigger UNCHECKED!

Jump Force – Step 3 : Make player jump at start

Until now, we've only called methods on the entirety of a gameobject or the transform component. If we want more control over the force and gravity of the player, we need to call methods on the player's Rigidbody component, specifically.

1. In PlayerController.cs, declare a new private Rigidbody playerRb; variable
2. In Start(), initialize playerRb = GetComponent<Rigidbody>();
3. In Start(), use the AddForce method to make the player jump at the start of the game

```
private Rigidbody playerRb;

void Start()
{
  playerRb = GetComponent<Rigidbody>();
  playerRb.AddForce(Vector3.up * 1000);
}
```

New Function: GetComponent

Tip: The playerRb variable could apply to anything, which is why we need to specify using GetComponent

spacebar pressed

We don't want the player jumping at start - they should only jump when we tell it to by pressing spacebar.

1. In Update() add an if-then statement checking if the spacebar is pressed
2. Cut and paste the AddForce code from Start() into the if-statement
3. Add the ForceMode.Impulse parameter to the AddForce call, then reduce force multiplier value

```
void Start()
{
  playerRb = GetComponent<Rigidbody>();
  playerRb.AddForce(Vector3.up * 100);
}
```

```
void Update() {
  if (Input.GetKeyDown(KeyCode.Space)) {
    playerRb.AddForce(Vector3.up * 100, ForceMode.Impulse); } }
```

- **Warning:** Don't worry about the slow jump double jump, or lack of animation, we will fix that later
- **Tip:** Look at Unity documentation for method overloads here
- **New Function:** ForceMode.Impulse and optional parameters

gravity

We need to give the player a perfect jump by tweaking the force of the jump, the gravity of the scene, and the mass of the character.

1. Replace the hardcoded value with a new public float jumpForce variable
2. Add a new public float gravityModifier variable and in Start(), add Physics.gravity *= gravityModifier;
3. In the inspector, tweak the gravityModifer, jumpForce, and Rigidbody mass values to make it fun

```
private Rigidbody playerRb;
public float jumpForce;
public float gravityModifier;

void Start() {
  playerRb = GetComponent<Rigidbody>();
  Physics.gravity *= gravityModifier; }

void Update() {
  if (Input.GetKeyDown(KeyCode.Space)) {
    playerRb.AddForce(Vector3.up * 10 jumpForce, ForceMode.Impulse); } }
```

- **New Function:** the students about something
- **Warning:** Don't make gravityModifier too high - the player could get stuck in the ground
- **New Concept:** Times-equals operator *=

double-jumping

The player can spam the spacebar and send the character hurtling into the sky. In order to stop this, we need an if-statement that makes sure the player is grounded before they jump.

1. Add a new public bool isOnGround variable and set it equal to true
2. In the if-statement making the player jump, set isOnGround = false, then test
3. Add a condition && isOnGround to the if-statement
4. Add a new void onCollisionEnter method, set isOnGround = true in that method, then test

```
public bool isOnGround = true

void Update() {
  if (Input.GetKeyDown(KeyCode.Space) && isOnGround) {
    playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
    isOnGround = false; } }

private void OnCollisionEnter(Collision collision) {
  isOnGround = true; }
```

- **New Concept:** Booleans
- **New Concept:** "And" operator (&&)
- **New Function:** OnCollisionEnter
- **Tip:** When assigning values, use one = equal sign. When comparing values, use == two equal signs

move it left

We've got the player jumping in the air, but now they need something to jump over.

We're going to use some familiar code to instantiate obstacles and throw them in the player's path.

1. From Course Library > Obstacles, add an obstacle, rename it "Obstacle", and position it where it should spawn
2. Apply a Rigid Body and Box Collider component, then edit the collider bounds to fit the obstacle
3. Create a new "Prefabs" folder and drag it in to create a new Original Prefab
4. Create a new "MoveLeft" script, apply it to the obstacle, and open it
5. In MoveLeft.cs, write the code to Translate it to the left according to the speed variable
6. Apply the MoveLeft script to the Background

```
private float speed = 30;

void Update() {
  transform.Translate(Vector3.left * Time.deltaTime * speed);
}
```

- **Warning:** Be careful choosing your obstacle in regards to the background. Some obstacles may blend in, making it difficult for the player to see what they're jumping over.
- **Tip:** Notice that when you drag it into hierarchy, it gets placed at the spawn location

Jump Force – Step 8 : Create a spawn manager

Similar to the last project, we need to create an empty object Spawn Manager that will instantiate obstacle prefabs.

1. Create a new "Spawn Manager" empty object, then apply a new SpawnManager.cs script to it
2. In SpawnManager.cs, declare a new public GameObject obstaclePrefab;, then assign your prefab to the new variable in the inspector
3. Declare a new private Vector3 spawnPos at your spawn location
4. In Start(), Instantiate a new obstacle prefab, then delete your prefab from the scene and test

- **Don't worry:** We're just instantiating on Start for now, we will have them repeating later
- **Tip:** You've done this before! Feel free to reference code from the last project

```
public GameObject obstaclePrefab;
private Vector3 spawnPos = new Vector3(25, 0, 0);

void Start() {
  Instantiate(obstaclePrefab, spawnPos, obstaclePrefab.transform.rotation); }
```

intervals

Our spawn manager instantiates prefabs on start, but we must write a new function and utilize InvokeRepeating if it to spawn obstacles on a timer. Lastly, we must modify the character's RigidBody so it can't be knocked over.

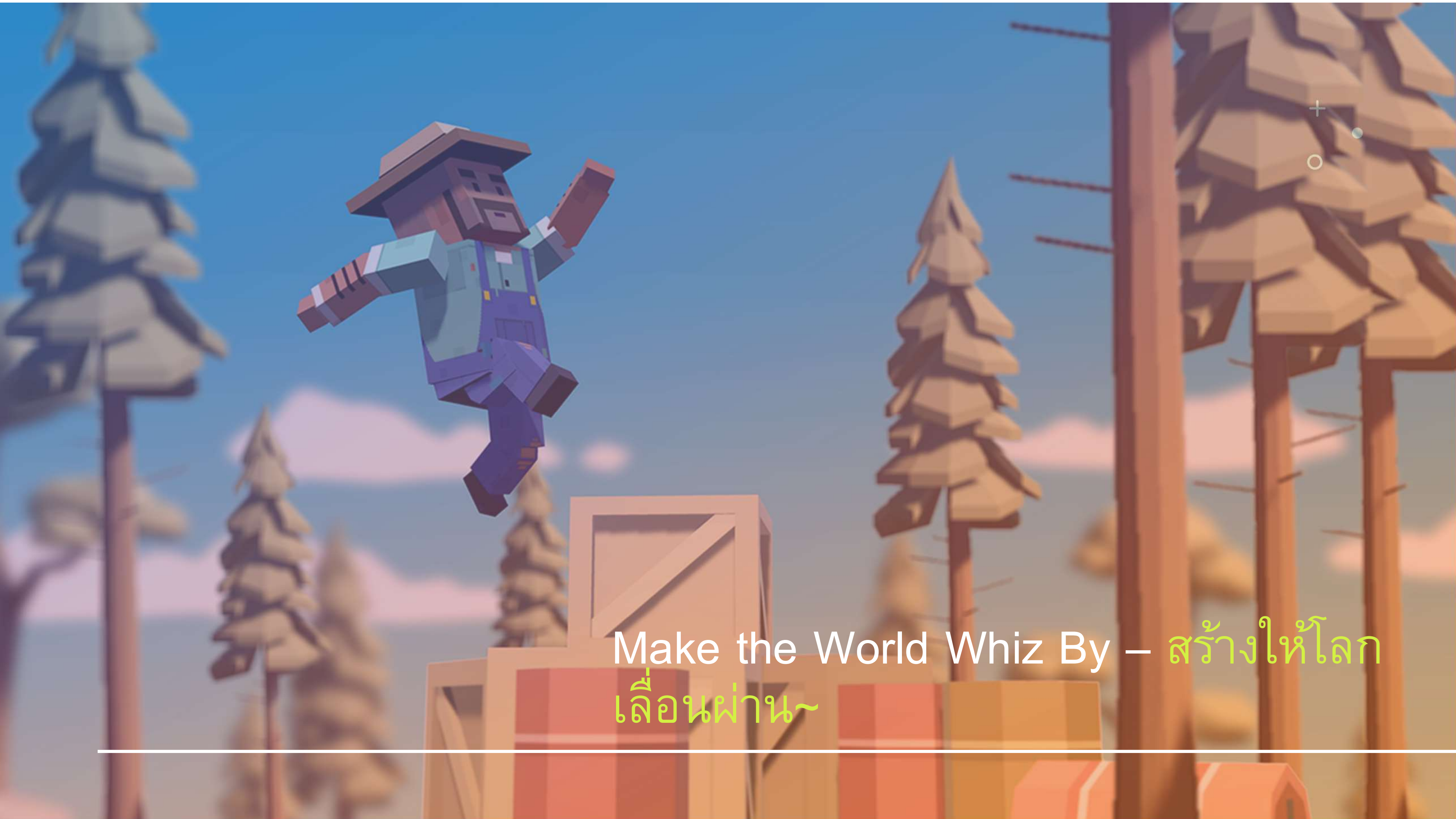
1. Create a new void SpawnObstacle method, then move the Instantiate call inside it
2. Create new float variables for startDelay and repeatRate
3. Have your obstacles spawn on intervals using the InvokeRepeating() method
4. In the Player RigidBody component, expand Constraints, then Freeze all but the Y position

```
private float startDelay = 2;
private float repeatRate = 2;

void Start() {
  InvokeRepeating("SpawnObstacle", startDelay, repeatRate);
  Instantiate(obstaclePrefab, spawnPos, obstaclePrefab.transform.rotation); }

void SpawnObstacle () {
  Instantiate(obstaclePrefab, spawnPos, obstaclePrefab.transform.rotation); }
```

- New Concept: RigidBody constraints


Make the World Whiz By – สร้างให้โลกเลื่อนผ่าน~

Make the World Whiz By

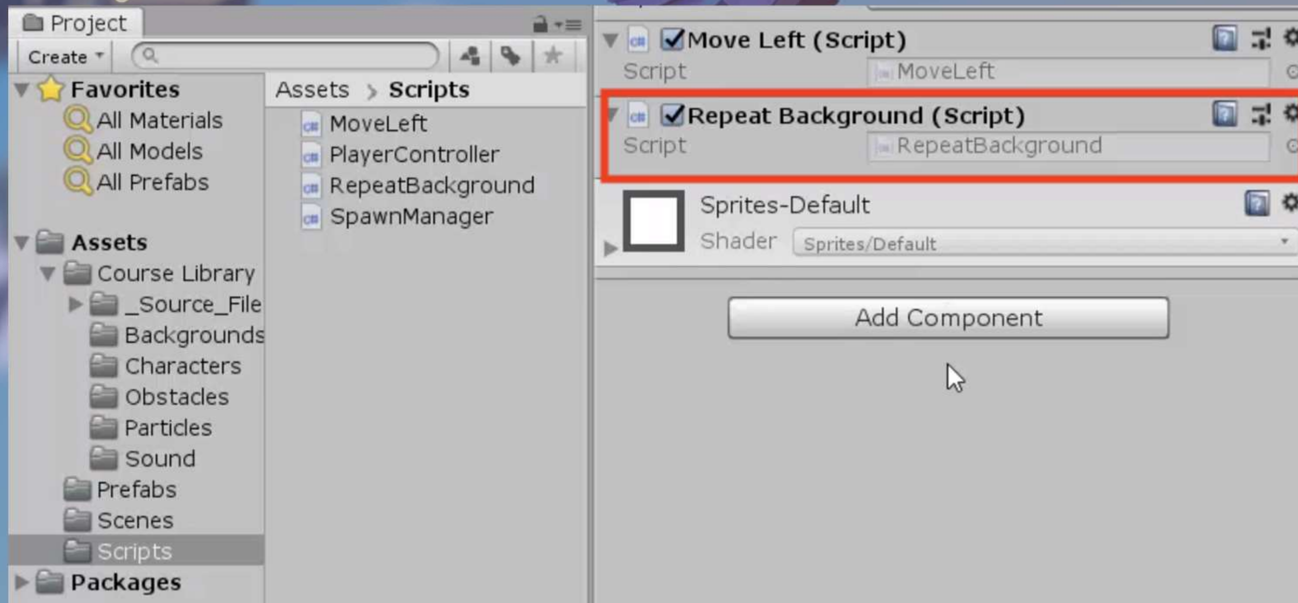
- Step 1 : Create a script to repeat background
- Step 2 : Reset position of background
- Step 3 : Fix background repeat with collider
- Step 4 : Add a new game over trigger
- Step 5 : Stop MoveLeft on gameOver
- Step 6 : Stop obstacle spawning on gameOver
- Step 7 : Destroy obstacles that exit bounds

background

We need to repeat the background and move it left at the same speed as the obstacles, to make it look like the world is rushing by. Thankfully we already have a move left script, but we will need a new script to make it repeat.

1. Create a new script called **RepeatBackground.cs** and attach it to the **Background Object**

- **Tip:** Think through what needs to be done: when the background moves half of its length, move it back that distance



background

In order to repeat the background and provide the illusion of a world rushing by, we need to reset the background object's position so it fits together perfectly.

1. Declare a new variable ***private Vector3 startPos;***
2. In ***Start()***, set the ***startPos*** variable to its actual starting position by assigning it ***= transform.position;***
3. In ***Update()***, write an **if-statement** to reset position if it moves a certain distance

- **Don't worry:** We're setting it at 40 for now, just to test basic functionality. You could probably get it right with trial and error... but what would happen if you changed the size?

```
private Vector3 startPos;

void Start() {
  startPos = transform.position; }

void Update() {
  if (transform.position.x < startPos.x - 55) {
    transform.position = startPos; } }
```

with collider

We've got the background repeating every few seconds, but the transition looks pretty awkward. We need make the background loop perfectly and seamlessly with some new variables.

1. Add a **Box Collider** component to the **Background**
2. Declare a new ***private float repeatWidth*** variable
3. In ***Start()***, get the width of the **box collider**, divided by 2
4. Incorporate the ***repeatWidth*** variable into the **repeat function**

- **Don't worry:** We're only adding a box collider to get the size of the background
- **New Function:** .size.x

```
private Vector3 startPos;
private float repeatWidth;

void Start() {
  startPos = transform.position;
  repeatWidth = GetComponent<BoxCollider>().size.x / 2; }

void Update() {
  if (transform.position.x < startPos.x - ~~50~~ repeatWidth) {
    transform.position = startPos; } }
```

trigger

When the player collides with an obstacle, we want to trigger a "Game Over" state that stops everything In order to do so, we need a way to label and discern all game objects that the player collides with.

1. In the inspector, add a "Ground" tag to the **Ground** and an "Obstacle" tag to the **Obstacle prefab**
2. In PlayerController, declare a new ***public bool gameOver;***
3. In ***OnCollisionEnter***, add the **if-else statement** to test if player collided with the "Ground" or an "Obstacle"
4. If they collided with the "Ground", set ***isOnGround = true***, and if they collide with an "Obstacle", set ***gameOver = true***

- **New Concept**: Tags
- **Warning:** New tags will NOT be automatically added after you create them. Make sure to add them yourself once they are created.
- **Tip:** No need to say gameOver = false, since it is false by default

```
public bool gameOver = false;

private void OnCollisionEnter(Collision collision) {
  isOnGround = true;
  if (collision.gameObject.CompareTag("Ground")) {
    isOnGround = true;
  } else if (collision.gameObject.CompareTag("Obstacle")) {
    gameOver = true;
    Debug.Log("Game Over!"); }
}
```

gameOver

We've added a gameOver bool that seems to work, but the background and the objects continue to move when they collide with an obstacle. We need the MoveLeft script to communicate with the PlayerController, and stop once the player triggers gameOver.

1. In **MoveLeft.cs**, declare a new ***private PlayerController playerControllerScript;***
2. In ***Start()***, initialize it by finding the **Player** and getting the PlayerController component
3. Wrap the **translate method** in an **if-statement** checking if game is not over

- **New Concept:** Script Communication
- **Warning:** Make sure to spell the "Player" tag correctly

```
private float speed = 30;
private PlayerController playerControllerScript;

void Start() {
  playerControllerScript =
  GameObject.Find("Player").GetComponent<PlayerController>(); }

void Update() {
  if (playerControllerScript.gameOver == false) {
    transform.Translate(Vector3.left * Time.deltaTime * speed); } }
```

on gameOver

The background and the obstacles stop moving when gameOver == true, but the Spawn Manager is still raising an army of obstacles! We need to communicate with the Spawn Manager script and tell it to stop when the game is over.

1. In **SpawnManager.cs**, get a reference to the **playerControllerScript** using the same technique you did in MoveLeft.cs
2. Add a condition to only instantiate objects if ***gameOver == false***

```
private PlayerController playerControllerScript;

void Start() {
  InvokeRepeating("SpawnObstacle", startDelay, repeatRate);
  playerControllerScript =
  GameObject.Find("Player").GetComponent<PlayerController>(); }

void SpawnObstacle () {
  if (playerControllerScript.gameOver == false) {
    Instantiate(obstaclePrefab, spawnPos, obstaclePrefab.transform.rotation);
} }
```

exit bounds

Just like the animals in Unit 2, we need to destroy any obstacles that exit boundaries. Otherwise they will slide into the distance… forever!

- **Tip:** Reference your code from MoveLeft

1. In **MoveLeft**, in ***Update()***; write an if-statement to **Destroy** Obstacles if their position is less than a ***leftBound*** variable
2. Add any **comments** you need to make your code more **readable**

```
private float leftBound = -15;

void Update() {
  if (playerControllerScript.gameOver == false) {
    transform.Translate(Vector3.left * Time.deltaTime * speed); }

  if (transform.position.x < leftBound && gameObject.CompareTag("Obstacle")) {
    Destroy(gameObject); } }
```

- Run and Jump Prototype
  - Jump Force
  - Make the World Whiz By
  - Don’t Just Stand There
  - Particles and Sound Effects

Next Week : To Be Continue…