unity

3.1 Jump Force

Steps:

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

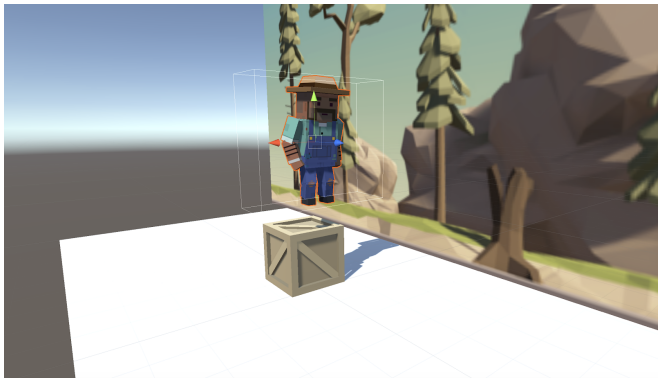
Step 6: Prevent player from double-jumping

Step 7: Make an obstacle and move it left

Step 8: Create a spawn manager

Step 9: Spawn obstacles at intervals

Example of project by end of lesson



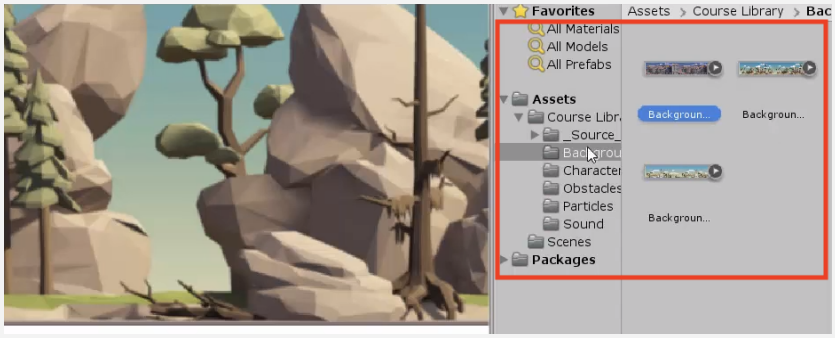
Length:	90 minutes
Overview:	The goal of this lesson is to set up the basic gameplay for this prototype. We will start by creating a new project and importing the starter files. Next we will choose a beautiful background and a character for the player to control, and allow that character to jump with a tap of the spacebar. We will also choose an obstacle for the player, and create a spawn manager that throws them in the player's path at timed intervals.
Project Outcome:	The character, background, and obstacle of your choice will be set up. The player will be able to press spacebar and make the character jump, as obstacles spawn at the edge of the screen and block the player's path.
Learning Objectives:	By the end of this lesson, you will be able to: - Use GetComponent to manipulate the components of GameObjects - Influence physics of game objects with ForceMode.Impulse - Tweak the gravity of your project with Physics.gravity - Utilize new operators and variables like && - Use Bool variables to control the number of times something can be done - Constrain the RigidBody component to halt movement on certain axes

Step 1: Open prototype and 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.
   If you forget how to do this, refer to the instructions in Lesson 1.1 - Step 1
2. Click to download the Prototype 3 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 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

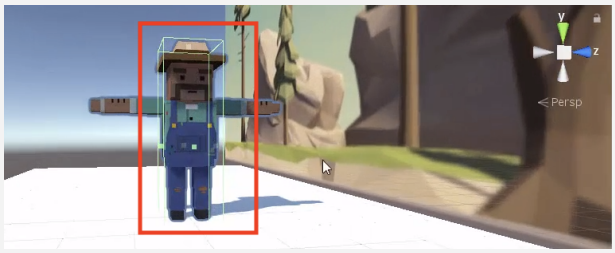


Step 2: Choose and set up a player character

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
- **Warning:** Keep isTrigger UNCHECKED!
- **Tip:** Use isometric view and the gizmos to cycle around and edit the collider with a clear perspective



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

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

```
private Rigidbody playerRb;

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

Step 4: Make player jump if 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

- **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

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

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

Step 5: Tweak the jump force and 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

- **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 *=

```
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); } }
```

Step 6: Prevent player from 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**

- **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

```
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; }
```

Step 7: Make an obstacle and 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**

- **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

```
private float speed = 30;

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

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); }
```

Step 9: Spawn obstacles at 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

- **New Concept:** RigidBody constraints

```
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); }
```

Lesson Recap

New Functionality	• Player jumps on spacebar press • Player cannot double-jump • Obstacles and Background move left • Obstacles spawn on intervals
New Concepts and Skills	• GetComponent • ForceMode.Impulse • Physics.Gravity • Rigidbody constraints • Rigidbody variables • Booleans • Multiply/Assign ("*) Operator • And (&&) Operator • OnCollisionEnter()
Next Lesson	• We're going to fix that weird effect we created by moving the background left by having it actually constantly scroll using code!