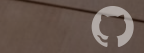
Welcome to 2023 Computer Science Summer Camp

Start →



- C# basics
- Unity - 1 Player and Platforms
- Unity - 2 PlayerController script
- Unity - 3 PlatformsMovement Script
- Unity - 4 ObjectFalling Script
- Unity - 5 UI & UX
- Unity - 6 - Restart
- Resources
- Learn More

C# basics

C# 基礎教學

Write a Hello World program

```
using System;
Console.WriteLine("Hello World!");
```

Basic Programming Concepts 基礎程式觀念

- Basic Programming Concepts 基礎程式觀念
  - Comments 註解
  - Variables & Data types 變數與資料型態
    - int, float & double
    - char & string
    - boolean

Comments 註解

- Doesn’t affect the functionality of programs
- Written for the purpose of explaining codes to developers

```
using System;
Console.WriteLine("Hello World!");
// This line outputs "Hello World!" to screen
```

Variables & Data types 變數與資料型態

type	function	
int	32-bit signed integer	整數
double	64-bit floating point type	浮點數
float	32-bit floating point type	浮點數
char	16-bit single Unicode character	字元
string	A sequence of Unicode characters	字串
bool	8-bit logical true/false value	布林值

int, float & double

- int 整數
- float, double 浮點數

```
using System;

int integer = 5;
Console.WriteLine(integer);

float f_num = 1.5f;
Console.WriteLine(f_num);

double d_num = 3.14;
Console.WriteLine(d_num);
```

output

```
5
1.5
3.14
```

char & string

- char 字元
- string 字串

```
using System;

char gender = 'm';
Console.WriteLine(gender);

string name = "Alex";
Console.WriteLine(name);
```

output

```
m
Alex
```

boolean

- bool 布林值

```
using System;

bool is_student = true;
if(is_student)
    Console.WriteLine("Yes");
else
    Console.WriteLine("No");
```

output

```
Yes
```

Array 陣列

What is an array?

- An array is a collection of items of same data type stored at contiguous memory locations

What is the purpose of using arrays?

- To store multiple pieces of data of the same type together

Array

Declaration

```
type[] arrayName;
// all elements are initialized with the default value
```

Array

Example

```
// Declare a single-dimensional array of 5 integers.
int[] arr = new int[5];
// arr[0] = 0, arr[1] = 0, arr[2] = 0, ..., arr[4] = 0

// Declare and set array element values
int[] arr = new int[] {1, 2, 3, 4, 5};

// You can also do in this way :
int[] arr = {1, 2, 3, 4, 5};

// Declare a two dimensional array
int[,] 2DArr = new int[2, 3];
//[0, 0] [0, 1] [0, 2]
//[1, 0] [1, 1] [1, 2]

// Declare and set array element values
int[,] 2DArr = {{1, 2, 3}, {4, 5, 6}};
```

Loops 迴圈

- Loops 迴圈
  - while
  - do while
  - for

while

```
while(condition){
    statement
}
```

example

```
using System;

int i = 0;
while(i < 3){
    Console.WriteLine(i);
    i++;
}
```

output

```
0
1
2
```

do while

```
do{
        statement
} while(condition);
```

example

```
using System;

int i = 0;
do{
    Console.WriteLine(i);
    i++;
} while(i < 3);
```

output

```
0
1
2
```

for

```
for(init-state; condition; expression){
    statement
}
```

example

```
using System;

for(int i = 0; i < 3; i++){
    Console.WriteLine(i);
}
```

output

```
0
1
2
```

A Number Guessing Game

- Given an unknown number. Whenever the player makes a guess, the program should tell if it is correct or wrong.
- More specifically, the program should tell the player if the answer is higher or lower.

A Number Guessing Game

Let’s start from game message and input

```
Console.WriteLine("Enter your guess: ");
```

we have to convert the input to INT datatype

```
int guess;
guess = Console.Convert.ToInt32(Console.ReadLine());
```

A Number Guessing Game

Now we check if the number is correct

```
if(guess > answer)
    Console.WriteLine("Smaller");
else if(guess < answer)
    Console.WriteLine("Bigger");
else
    Console.WriteLine("BINGO!!!");
```

A Number Guessing Game

Lastly, we put all the components in a do while loop

```
using System;

int answer = 20;
int guess = answer + 1;
while(guess != answer){
    Console.WriteLine("Enter your guess: "); // user input
    guess = Convert.ToInt32(Console.ReadLine()); // convert string to int

    if(guess > answer){
        Console.WriteLine("Smaller");
    }
    else if(guess < answer){
        Console.WriteLine("Bigger");
    }
    else{
        Console.WriteLine("BINGO!!!");
    }
}
```

A Number Guessing Game

Advanced Gameplay

- Set limit on guessing tries
- Choose the answer randomly

Function

What is a function?

- A technique of wrapping code to perform a certain task

Why do we use functions?

- Same code can be reused over and over
- Enables reusability and reduces redundancy
- The program becomes easy to understand and manage

Function

- How to declare a function?

```
<Access Specifiers> <return type> <name of the function>(< function parameters>)
{
    <function code>
    return;
}
```

Function

How to use it?

```
using System;
void square(int num)
{
    int sq = num * num;
    Console.WriteLine(sq);
}

square(5);
// Calling the method
```

output

```
25
```

Function

- return type : defines and constrains the data type of the value returned from a function

Function

- return type : void
- doesn’t return any value

```
using System;
void square(int num)
{
    int sq = num * num;
    Console.WriteLine(sq);
    // Doesn’t provide any return statement
}

square(5);
```

output

```
25
```

Function

- return type : int(or any datatype you want)
- returns a value

```
using System;
int square(int num)
{
    return num * num;
    // return statement
}

Console.WriteLine(square(5));
```

output

```
25
```

Function

- parameters參數 : the data you give to a function

Function

- example

```
//addition function : 2 parameters
int plus(int num1, int num2){
    return num1 + num2;
}

//subtraction function : 2 parameters
int minus(int num1, int num2){
    return num1 - num2;
}

//square function : 1 parameters
int square(int num){
    return num * num;
}

//print function : 1 parameters
void print(int times){
    for(int i = 0; i < times; ++i){
        Console.WriteLine("Hi");
    }
}
```

Class (類別)

- What is a class?

```
using System;
public class Cat
{
    int age;
    int health;
    int speed;

    public Cat(int age)
    {
        this.age = age;
        this.health = 100;
        this.speed = age + 20;
    }

    public void Meow()
    {
        Console.WriteLine("Cat with age:" + age +" said meow~");
    }
}
```

Class (類別)

- Classes are like Data Types, but with more functionality

```
class Program
{
    static void Main()
    {

        Cat cat1 = new Cat(10);
        Cat cat2 = new Cat(20);
        cat1.Meow();
    }
}
```

Class (類別)

- Properties

```
using System;
public class Cat
{
    int age;
    int health;
    int speed;

    public Cat(int age)
    {
        this.age = age;
        this.health = 100;
        this.speed = age + 20;
    }

    public void Meow()
    {
        Console.WriteLine("Cat with age:" + age +" said meow~");
    }
}
```

Class(類別)

- Class Methods

```
using System;
public class Cat
{
    int age;
    int health;
    int speed;

    public Cat(int age)
    {
        this.age = age;
        this.health = 100;
        this.speed = age + 20;
    }

    public void Meow()
    {
        Console.WriteLine("Cat with age:" + age +" said meow~");
    }
}
```

Class(類別)

- Class Constructor (構造器)

```
using System;
public class Cat
{
    int age;
    int health;
    int speed;

    public Cat(int age)
    {
        this.age = age;
        this.health = 100;
        this.speed = age + 20;
    }

    public void Meow()
    {
        Console.WriteLine("Cat with age:" + age +" said meow~");
    }
}

class Program
{
    static void Main()
    {
```

Class(類別)

- public private modifiers

1. Public 會使 Property 跟 Method 變成任何人都可觀看與修改
2. Private 會使 Property 跟 Method 變成只有Class內可觀看與修改

```
class User{
    public string name;
    private string password;

    public User(string name, string password){
        this.name = name;
        this.password = password;
    }

    public void SetPassword(string password){
        this.password = password;
    }

    private string GetPassword(){
        return password;
    }
}
class Program
{
    static void Main()
    {
```

Lab

- Make the following code work.

```
class Program
{
    static void Main()
    {
        string type = "strawberry";
        Food food = new Food(type); //創建一個type為strawberry的食物
        string name = "Jeffery";
        Person person = new Person(name); //創建一個名字叫Jeffery的人
        person.Feed(food); //餵食物給Jeffery
    }
}
```

Output

```
Jeffery ate a strawberry
```

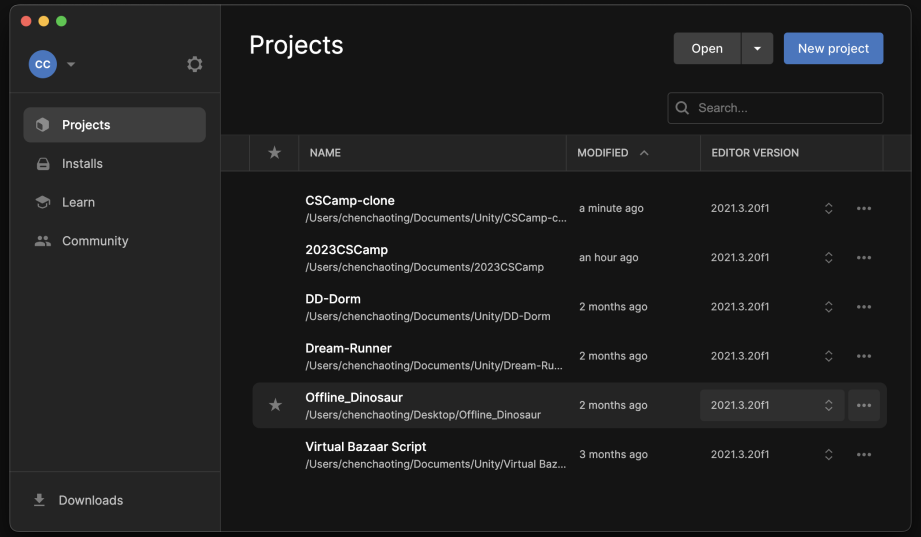
Answer

```
class Food{
    public string type;
    public Food(string type){
        this.type = type;
    }
}

class Person{
    private string name;

    public Person(string name){
        this.name = name;
    }

    public void Feed(Food food){

    }

}
```

Unity - 1 Player and Platforms

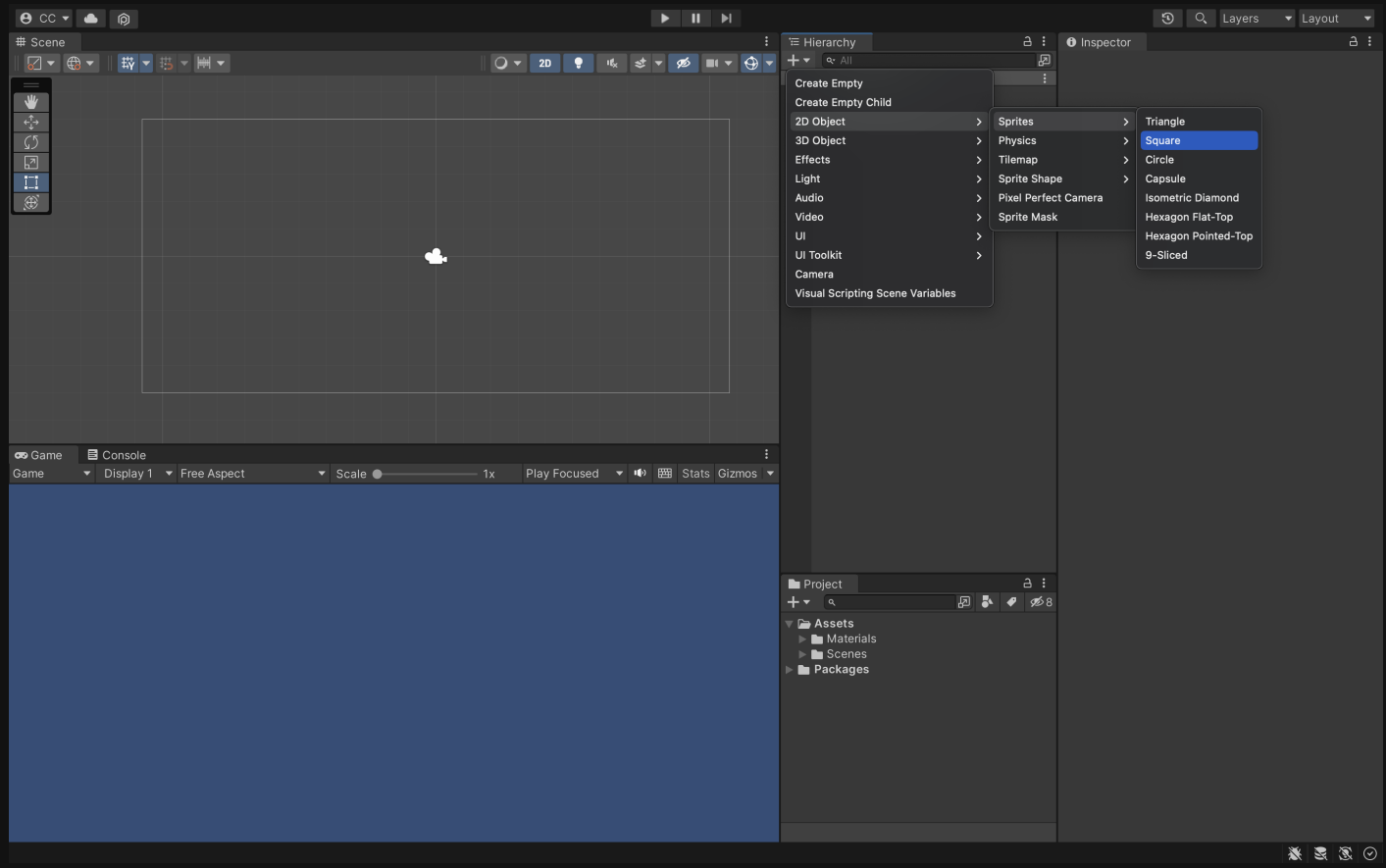
- Create a Project with Unity Hub
- Create Platforms
- Add Player
  - BoxCollider2D
  - RigidBody2D
- C# Script

Create a Project with Unity Hub

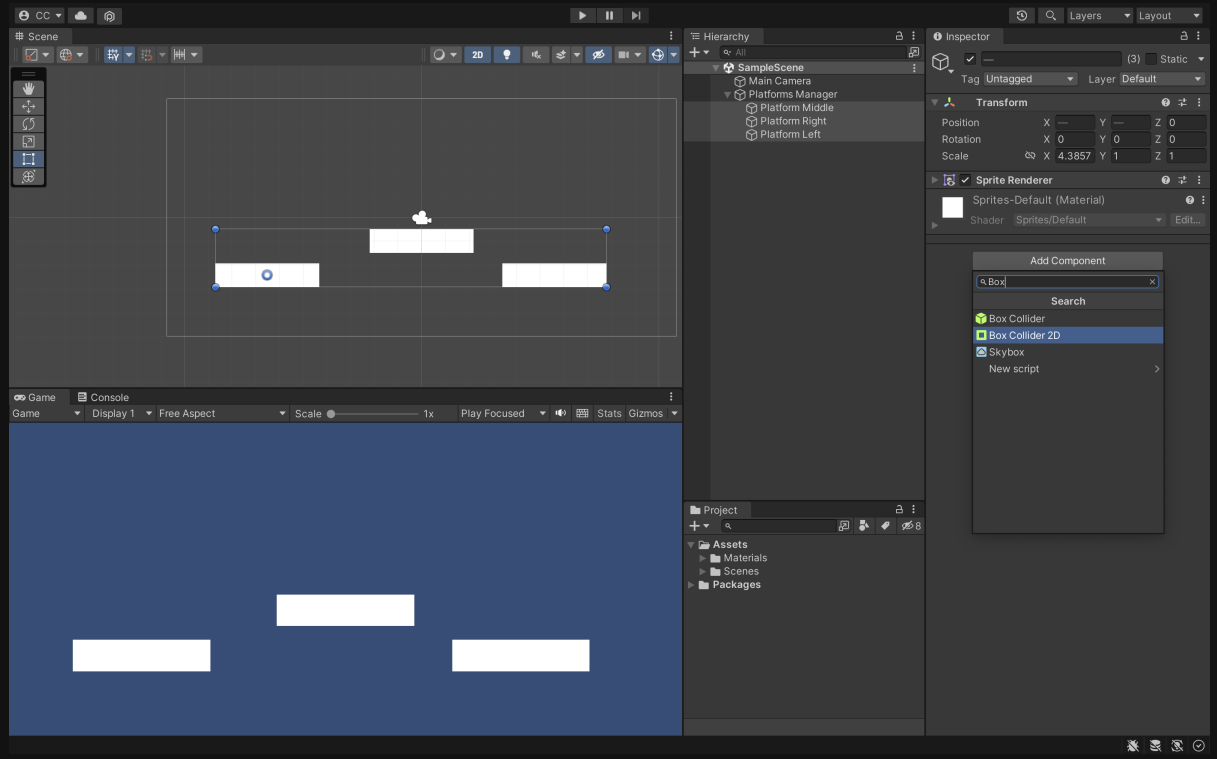


Press the `New project` and create a new one.

Create Platforms

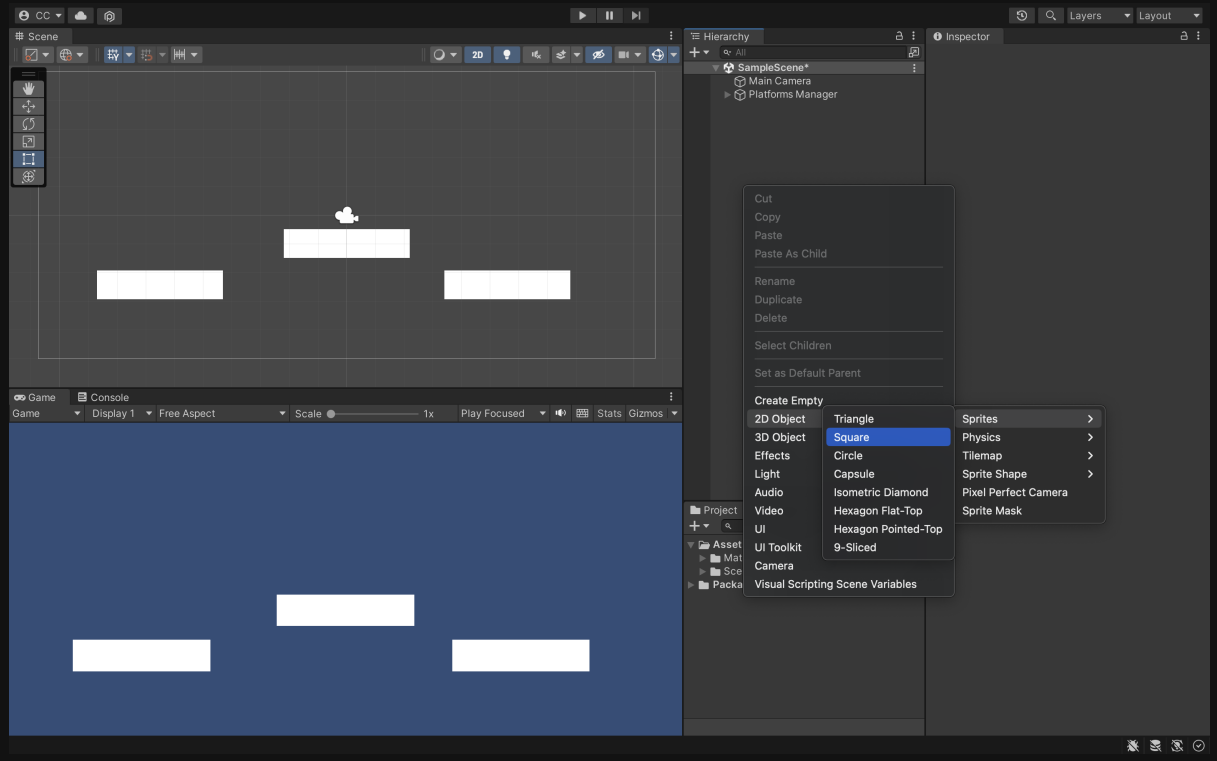


1. Add three components `2D Object/Sprites/Square` and rename to `Platform Left/Middle/Right`.
2. Give those platforms Box Collider 2D
3. Select and create empty parant as `PlatformsManager`



Add Player

1. Add player components, `2D Object/Sprites/Square` and rename to `Player`
2. Add BoxCollider2D
3. Add RigidBody2D



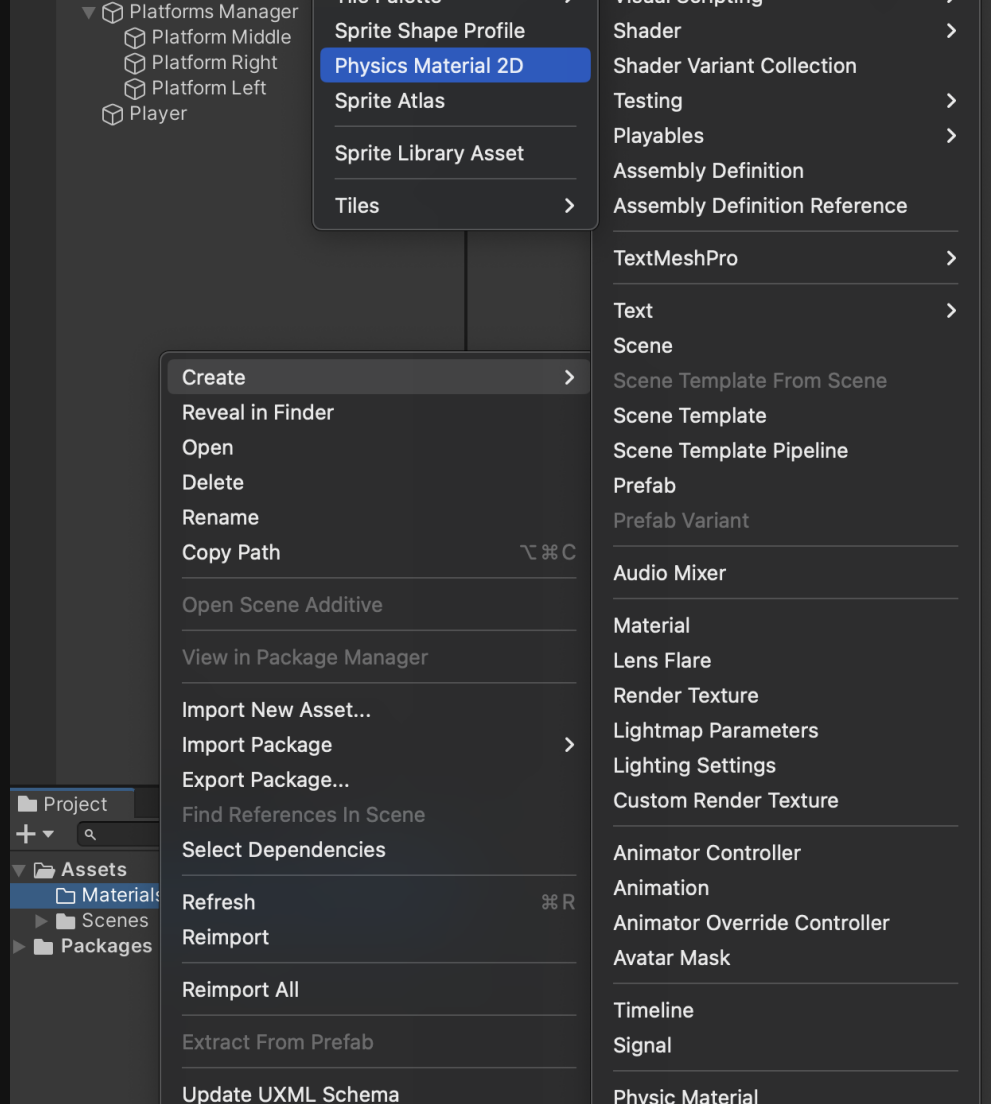
BoxCollider2D

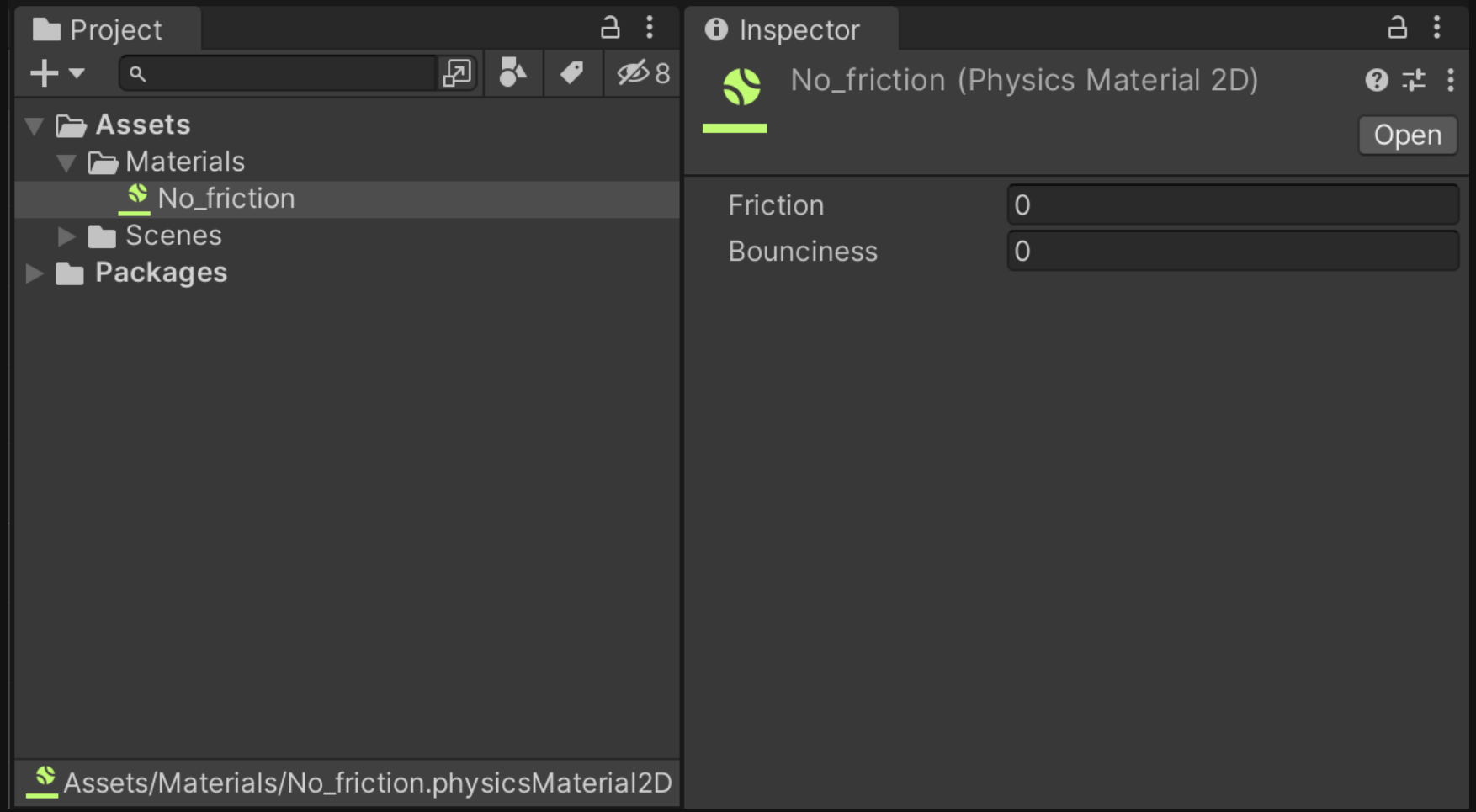
Box collider is a cuboid-shaped collision(碰撞) primitive(原始物件).

`Material: No_friction`

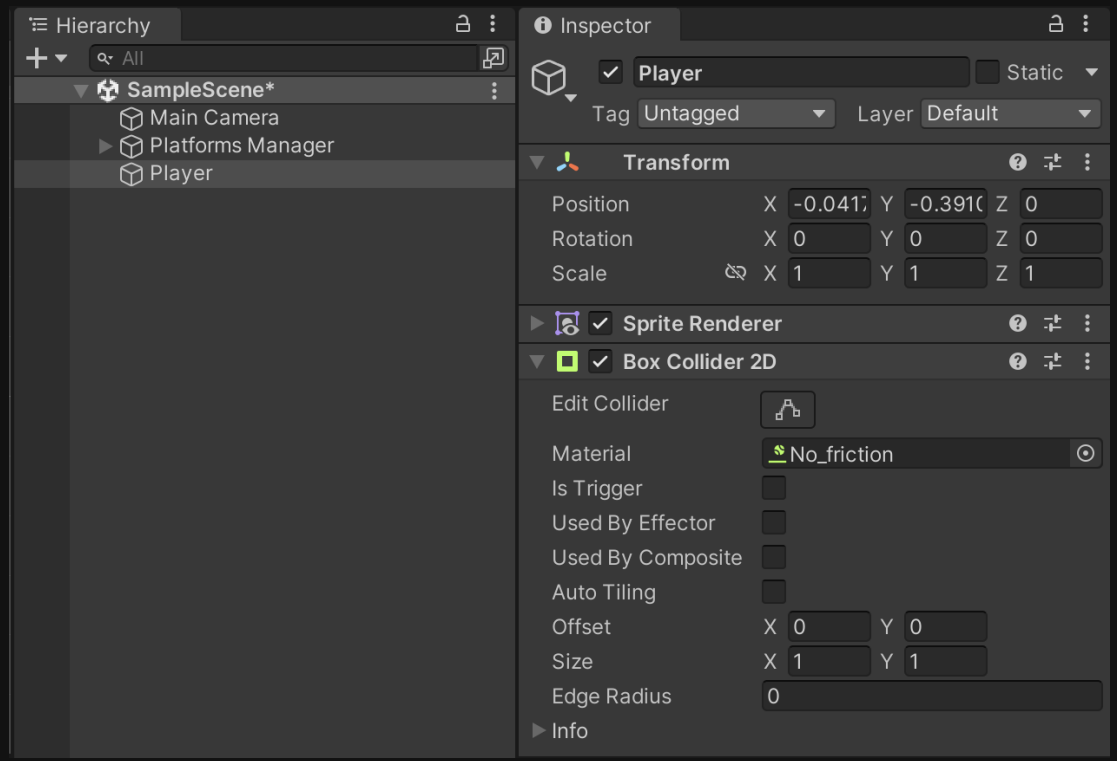
Generate No_friction

`Create/2D/Physics Material 2D`




Project
8
Assets
Materials
No_friction
Scenes
Packages
Assets/Materials/No_friction.physicsMaterial2D
Inspector
No_friction (Physics Material 2D)
Open
Friction
0
Bounciness
0

Add `No_friction` to `Player`'s `Material` section

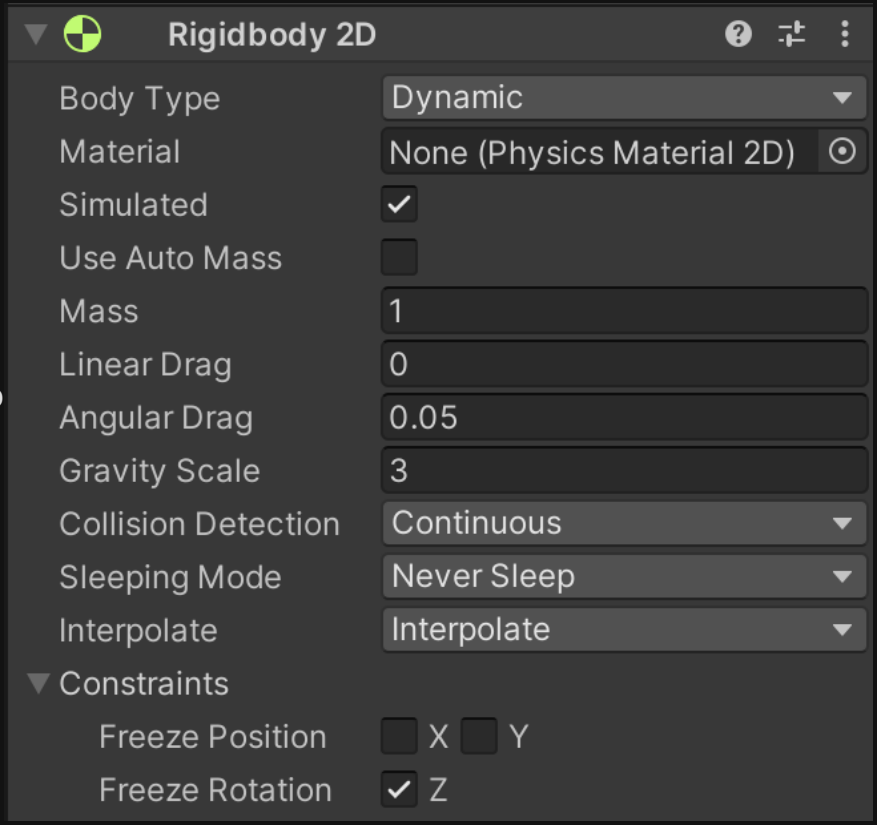


RigidBody2D

Control of an object's position through physics simulation. Adding RigidBody to an object will put its motion under the control of Unity's physics engine.

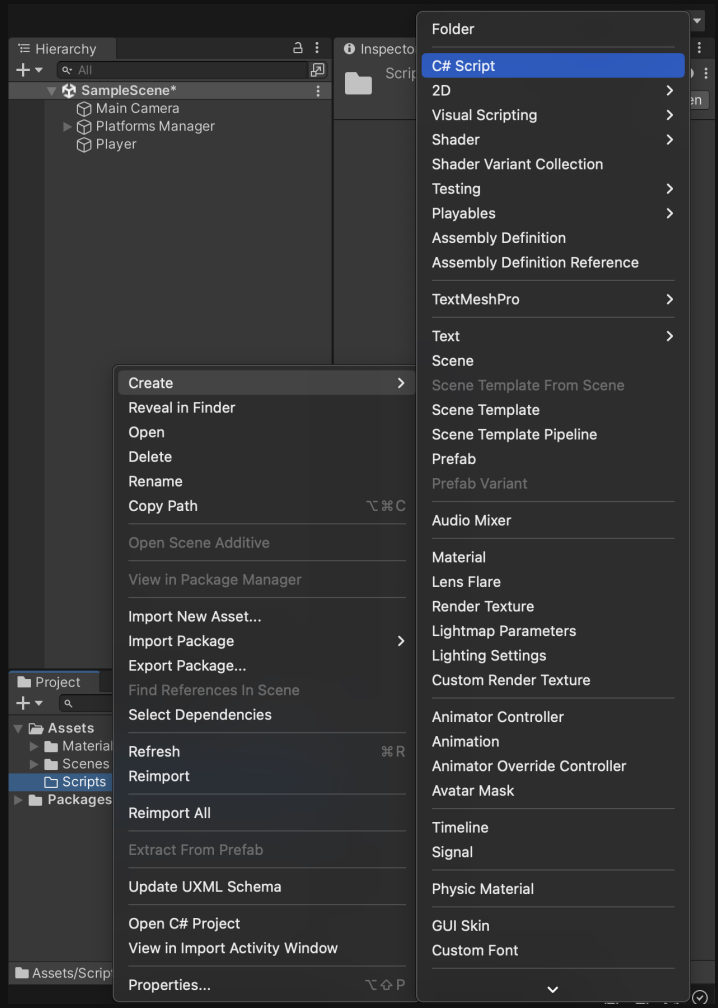
Even without any code, a rigidbody object will be *pulled downward by gravity* and and will *react to collisions* with incoming objects if the right Collider component is also present.

`Gravity Scale: 3, Collision Detection: Continuous, Sleeping Mode: Never Sleep, Interpolation: Interpolation, Constraints/Freeze Rotation: Freeze Z`



C# Script

Add script to `Scripts` folder and rename to `PlayerController`

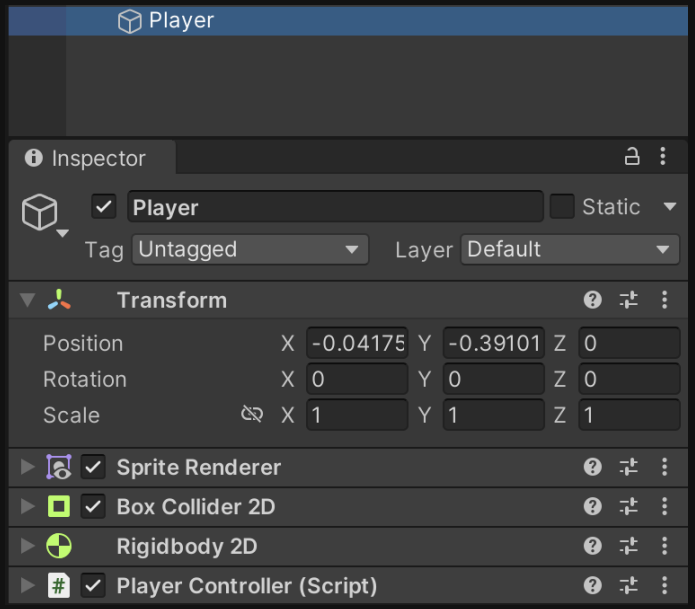


Unity - 2 PlayerController script

- Playcontroller Script
- C# script
  - Input function
  - IsGrounded function
- Setup Playcontroller Script
  - Add Ground Check
  - Add Ground Layer
  - Drag components for script

Playcontroller Script

First, drag `PlayerController` to `Player` component, double click the script to edit



C# script

```
using System.Collections;           // Include toolkit
using System.Collections.Generic;   // Include toolkit
using UnityEngine;                  // Include toolkit

public class PlayerController : MonoBehaviour
{
    // Start is called before the first frame update
    void Start() {}

    // Update is called once per frame
    void Update() {}
}
```

`void start()` is execute once when the script was called.
`void Update()` is called every frame.
`void FixedUpdate()` execute once at a fixed interval. (每單位時間)

Declare our variables

```
private float horizental; // -1, 0 or 1 (horizontal position)
private float speed = 8f;
private float jumpingPower = 12f;

// Reference the rigid body, ground check and ground layer
[SerializeField] private Rigidbody2D rb;
[SerializeField] private Transform groundCheck; // Check if player touch the ground
[SerializeField] private LayerMask groundLayer; // Check if component is ground
```

`SerializeField` will force Unity to serialize a private field. (讓使用者在 UI 對 private variables 做控制)

```
void Update()
{
    horizontal = Input.GetAxisRaw("Horizontal"); // 確認左右

    if (Input.GetButtonDown("Jump") && IsGrounded()) { // 當接觸地面時 向上跳
        rb.velocity = new Vector2(rb.velocity.x, jumpingPower);
    }

    if (Input.GetButtonUp("Jump") && rb.velocity.y > 0f) { // 按越久跳越高
        rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y * 0.5f);
    }
}

private void FixedUpdate() {
    rb.velocity = new Vector2(horizontal * speed, rb.velocity.y); // 左右增加速度
}
```

Input function

> Input method and settings are defined in `Edit > Project Settings > Input Manager`

Input.GetAxisRaw

`Input.GetAxisRaw("axisName")` will either be -1, 0 or 1.

Input.GetButtonDown

`Input.GetButtonDown("Button Name")` returns true the first frame the user releases the virtual button.

IsGrounded function

```
private bool IsGrounded() { // 確認是否著地
    return Physics2D.OverlapCircle(groundCheck.position, 0.2f, groundLayer);
}
```

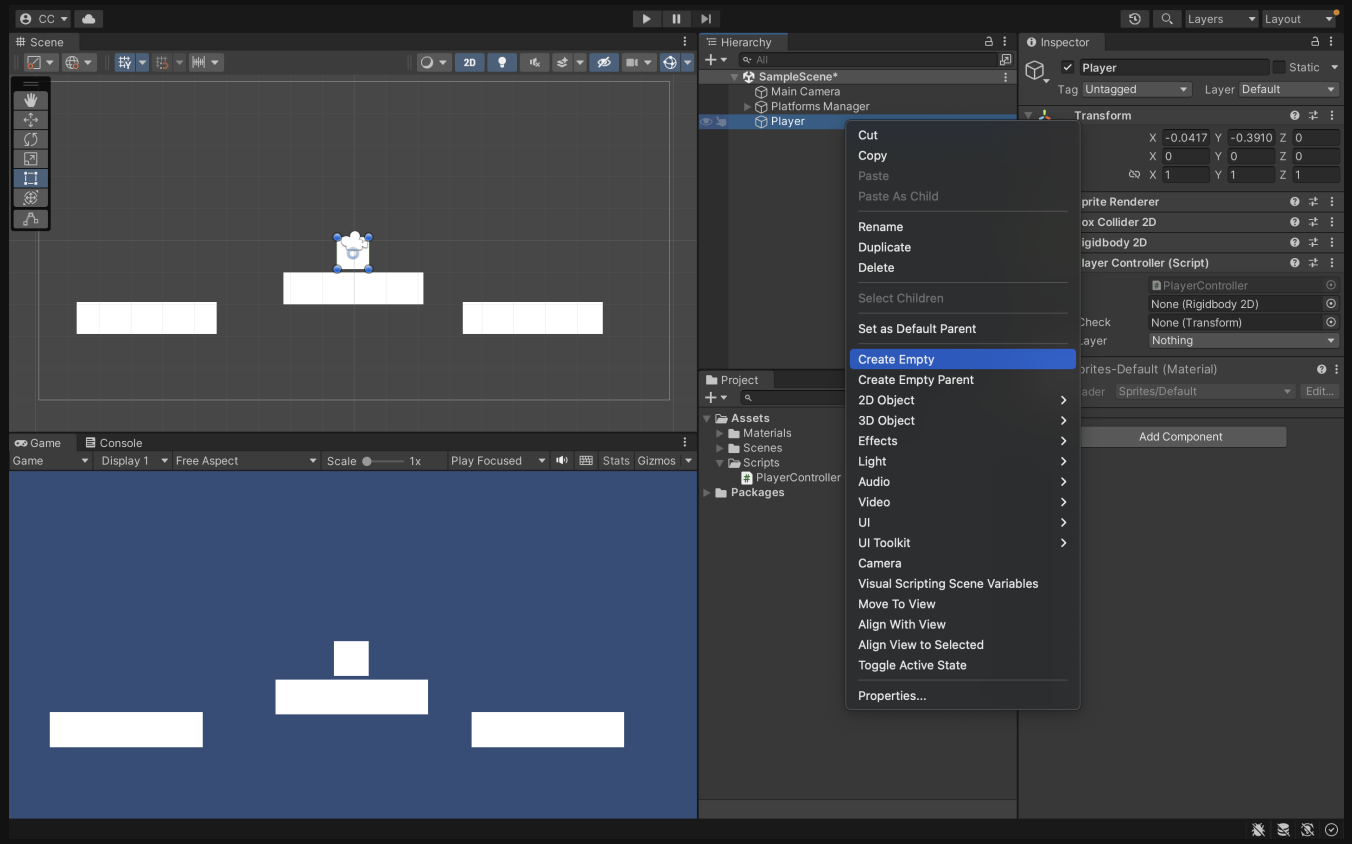
Setup Playcontroller Script

Open the script information

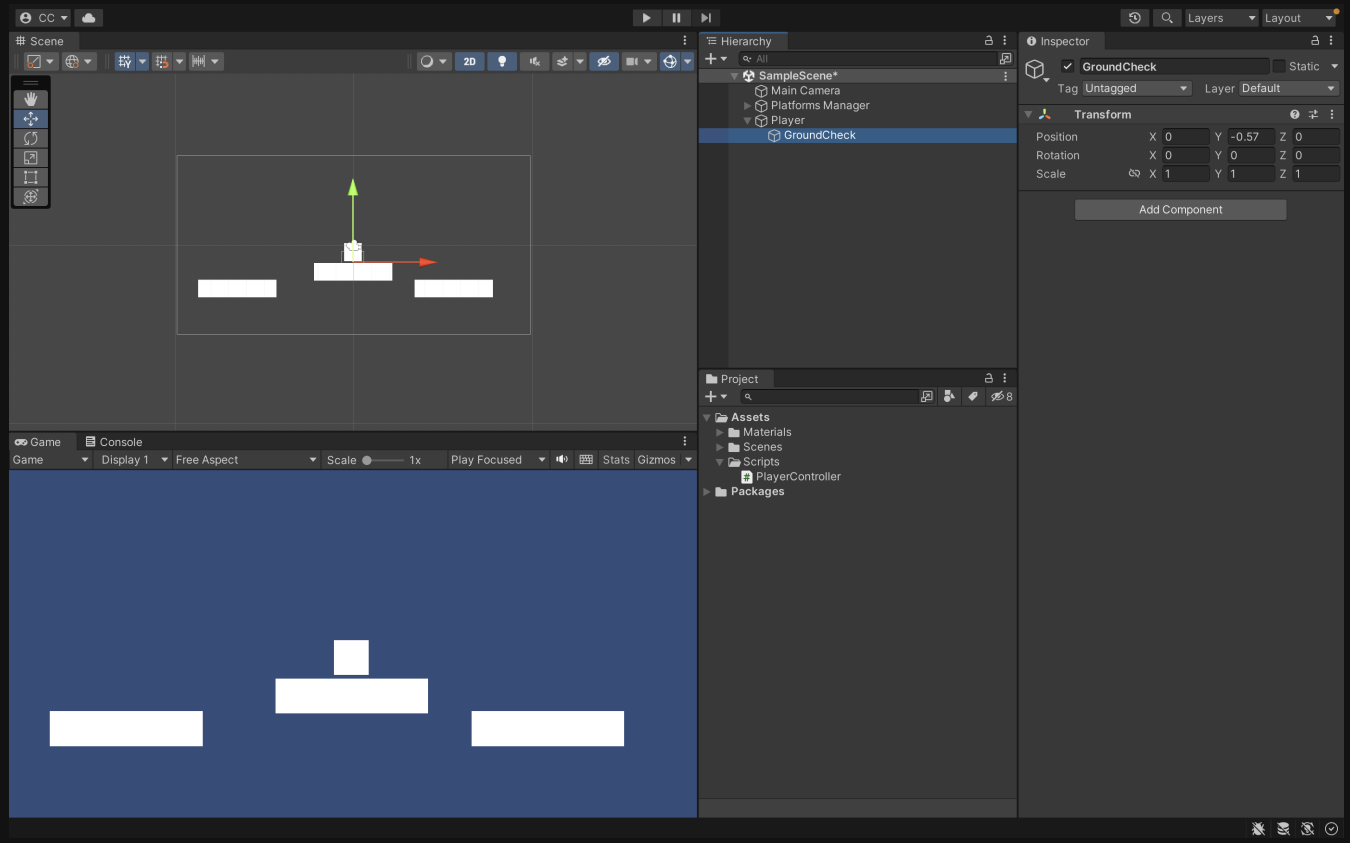

```
Rb: Player
Ground Check: GroundCheck
Ground Layer: Ground
```

Add Ground Check

Add GroundCheck components (child component of Player)

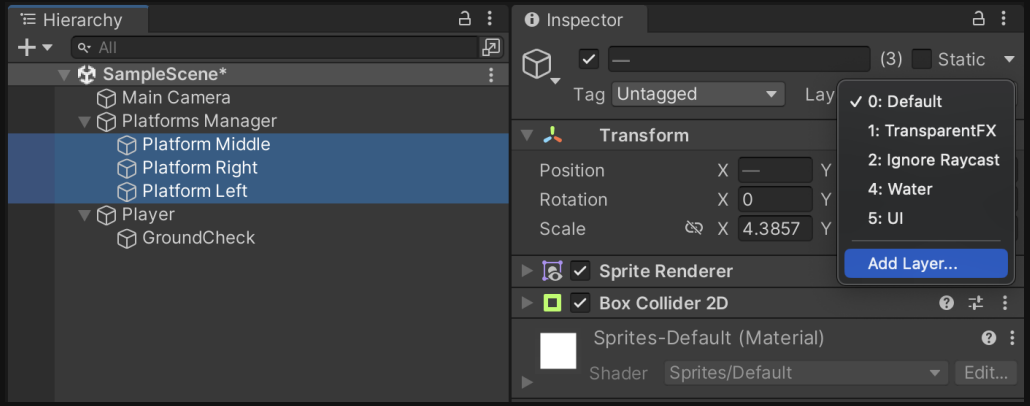


Move `GroundCheck` a little bit lower than `Player` to check if the `Player` touch the ground

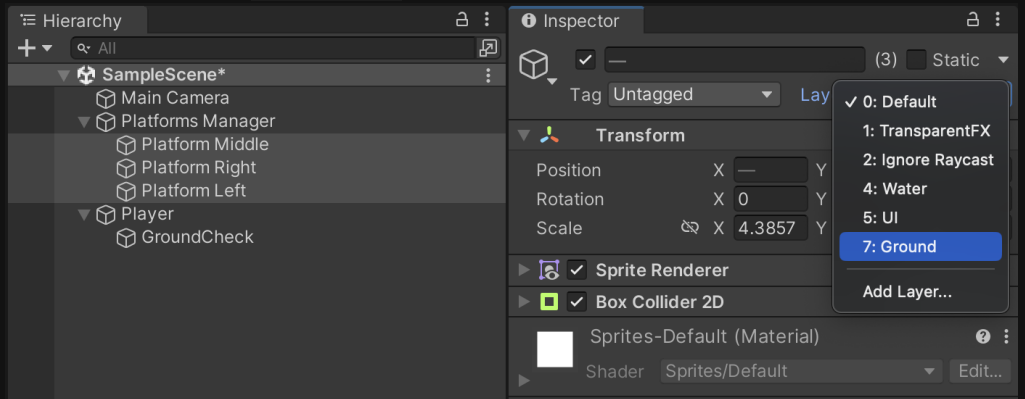


Add Ground Layer

First, add `Ground` layer.



Second, choose `Ground` layer for three platforms.



Drag components for script

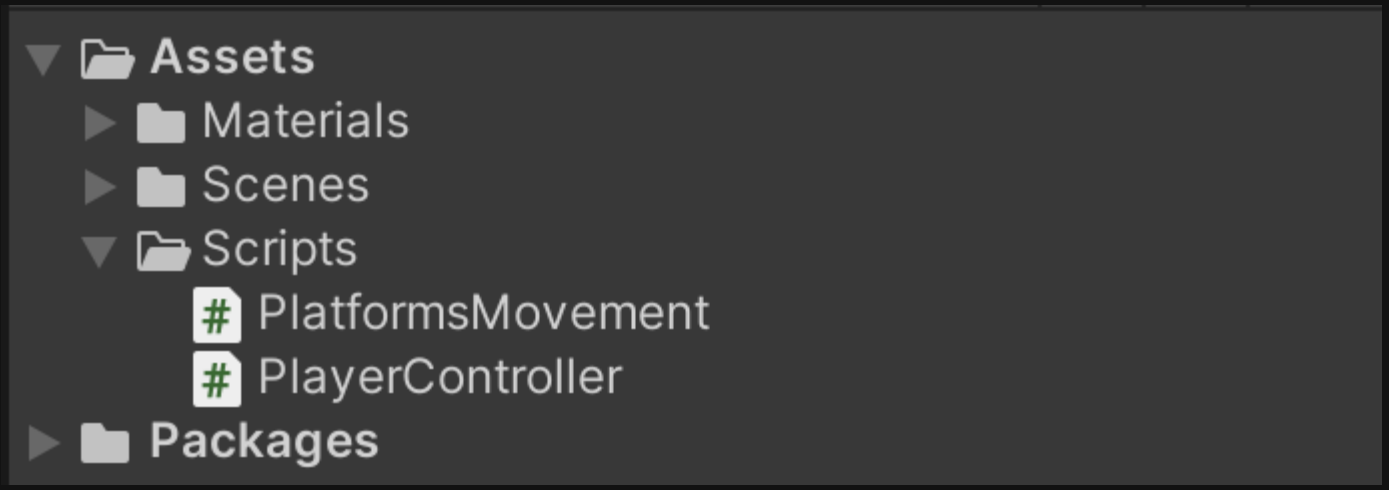

```
Rb: Player
Ground Check: GroundCheck
Ground Layer: Ground
```

Unity - 3 PlatformsMovement Script

- C# Script
- Dragging Script and Setting Components
- Stop the Tremble of Player
  - Concept
  - Setup from Unity Editor

C# Script

1. Generate a script under `Scripts` folder, rename to `PlatformsMovement`
2. Double click the file to edit



```
// Import the toolkits
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Declare the variables we will use later

```
public class PlatformsMovement : MonoBehaviour
{
    public float speed;         // Speed of the platforms movement
    public Transform[] points;  // Starting point and the turning point of the platforms
    private int i = 0;
}
```

Initialize the starting point

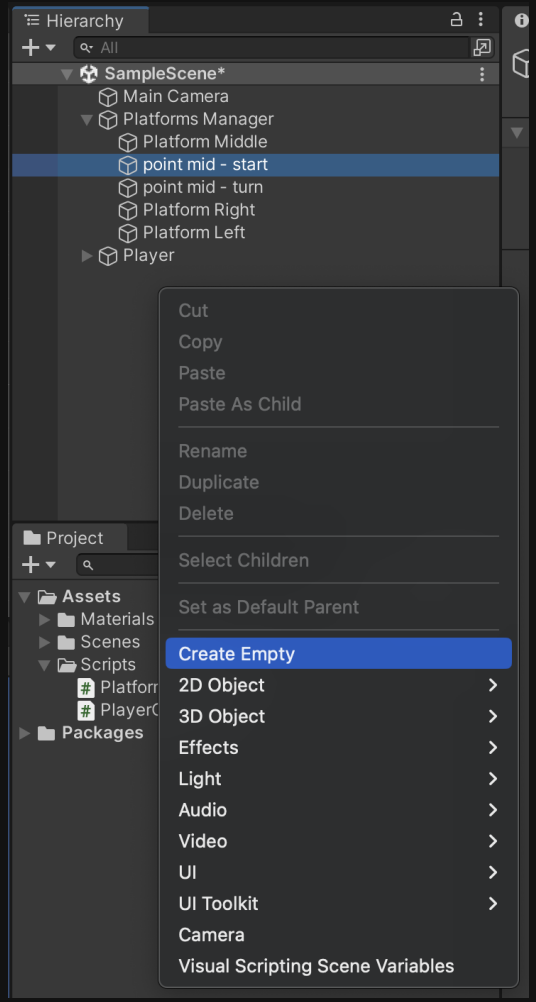
```
void Start() {
    transform.position = points[0].position; // Start from the starting point
}
```

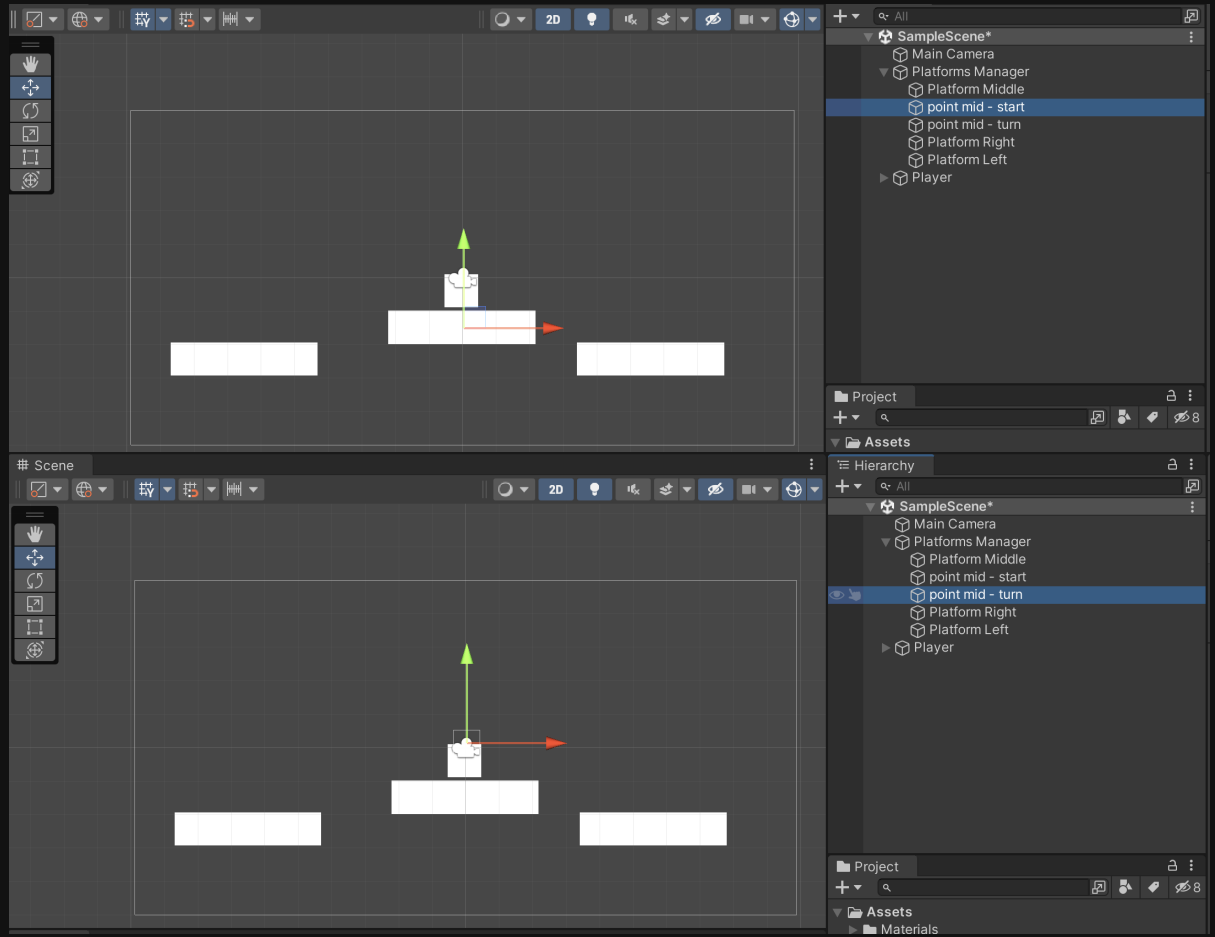
```
void Update() {
    if (Vector2.Distance(transform.position, points[i].position) < 0.09f) {
        ++i;
        if (i == points.Length) i = 0;
    }
    transform.position = Vector2.MoveTowards(transform.position, points[i].position, speed * Time.deltaTime);
}
```

`MoveTowards()`

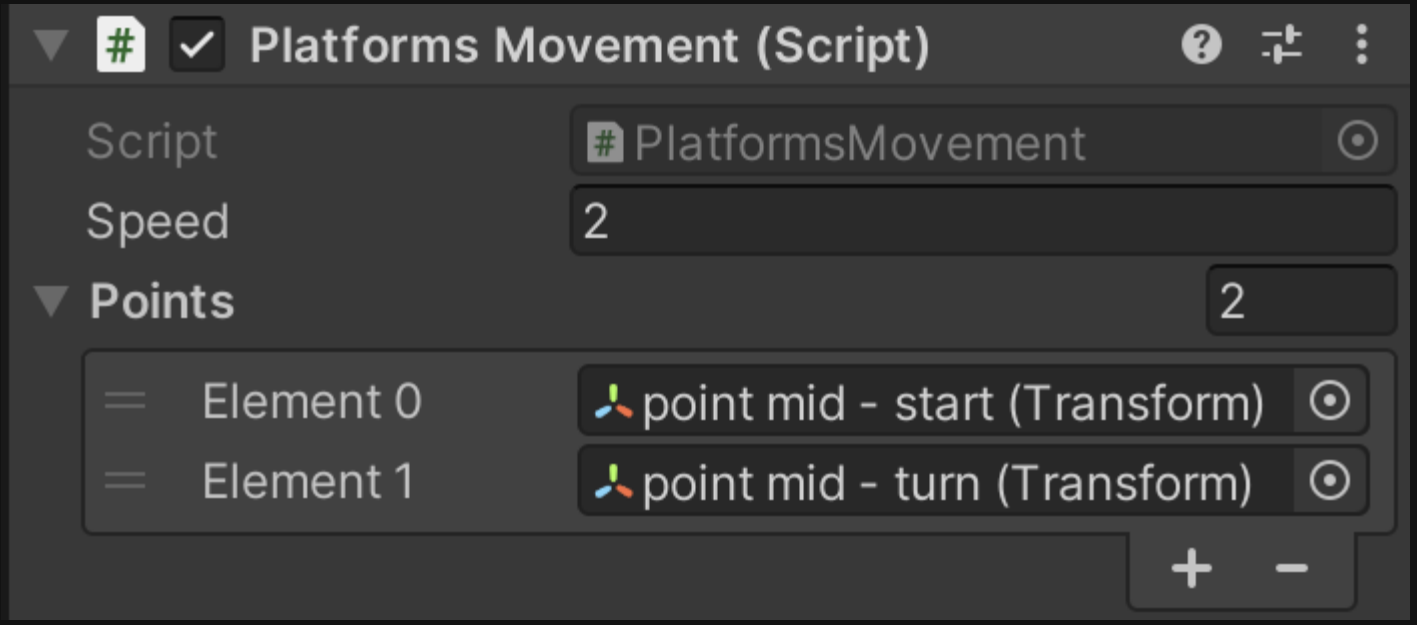
Dragging Script and Setting Components

1. Drag the script to one platform
2. Create starting point and turning point
   - Start point: The start point of the platform
   - End point: The end point of the platform




2D
All
SampleScene*
Main Camera
Platforms Manager
Platform Middle
point mid - start
point mid - turn
Platform Right
Platform Left
Player
Project
Assets
Scene
Hierarchy
Materials

Drag start point and end point to the array of positions (`public Transform[] points`)



> Repeat these steps three times for each platforms (Middle, Left, Right)

Stop the Tremble of Player

TODO: Set platform as parent of player when player is standing on the platform

Concept

1. If player collides with platform, set player's parent to platform
2. If player is moving or exit the collide, set player's parent to nothing (i.e. `null`)
3. Make sure the thing that trigger the collision is player instead of something else

1. If player collides with platform, set player's parent to platform

```
private void OnTriggerStay2D(Collider2D collision) {
    if(collision.tag == "Player") { // Make sure the object collide is `Player`
        collision.transform.SetParent(this.transform);
    }

    if (isMoving()) collision.transform.SetParent(null); // If moving, set parent to `null`
}
```

2. If player is moving or exit the collide, set player's parent to nothing (i.e. `` `null` ``)

```
public bool isMoving() { // Check if the player is moving
    // return playerController.getHorizontal() != 0 ? true : false;
    if(playerController.getHorizontal() == 0) return false;
    return true;
}

if (isMoving()) collision.transform.SetParent(null);
```

3. Make sure the thing that trigger the collision is player instead of something else

```
public bool isMoving() { // Check if the player is moving
    // return playerController.getHorizontal() != 0 ? true : false;
    if(playerController.getHorizontal() == 0) return false; // If speed of player is 0
    return true;
}

private void OnTriggerStay2D(Collider2D collision) {
    if(collision.tag == "Player") {
        collision.transform.SetParent(this.transform);
    }

    if (isMoving()) collision.transform.SetParent(null);
}

private void OnTriggerExit2D(Collider2D collision) {
    if (collision.tag == "Player") collision.transform.SetParent(null);
}
```

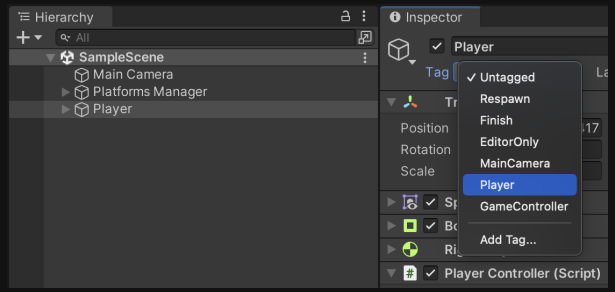
At `Playcontroller` script, add a public function for checking if player is moving

```
// PlayerController.cs
public float getHorizontal() {
    return horizental;
}
```

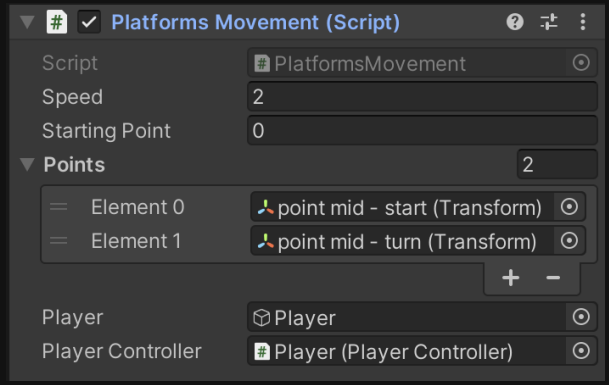
Setup from Unity Editor

1. Add player tag
2. Set player for platform
3. Add Rigidbody 2D for triggering collision

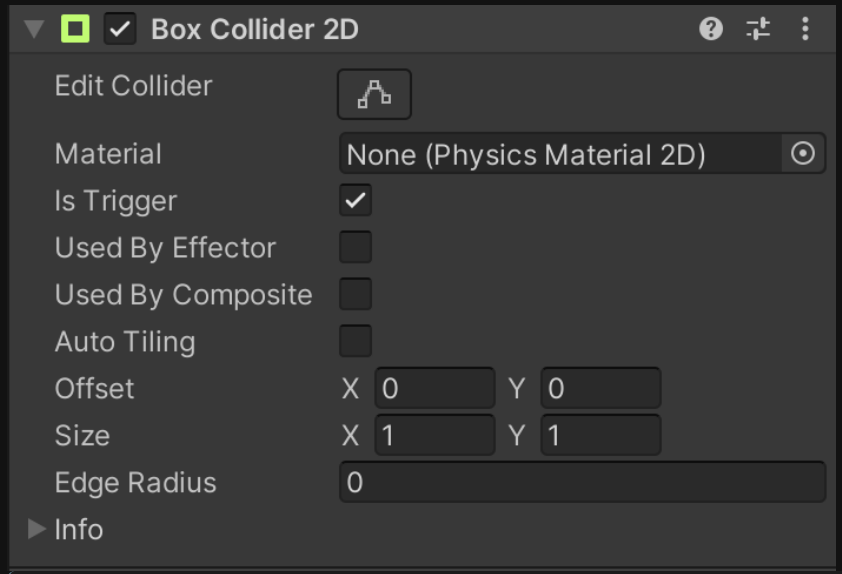
Add player tag for player



Set `player` for platform



After adding player tag, we need to add a `Rigidbody 2D` for triggering the collision. (After adding, platform will have *two* box collider)



Check the `Is Trigger` box

Finally, do the same thing for the other platforms.

Unity - 4 ObjectFalling Script

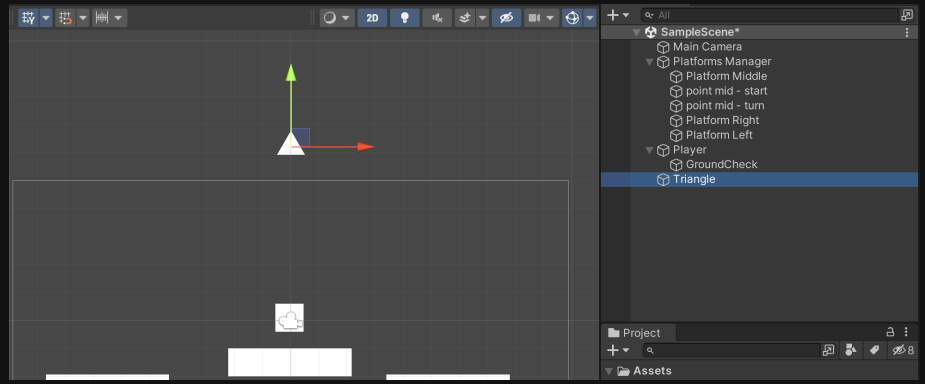
- ObjectFalling and GameManager
  - Create Triangle Prefab
  - Create the Two Scripts
- ObjectFalling

ObjectFalling and GameManager

1. Create a prefab `triangle`
2. Create two scripts, `ObjectFalling` and `GameManager`

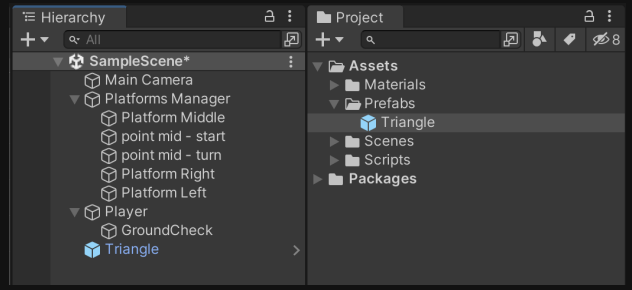
Create Triangle Prefab

1. Create a triangle

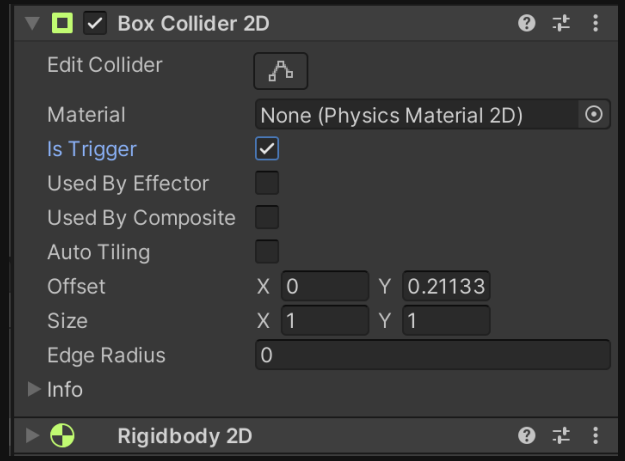


2. Create a folder `Prefabs` under `Assets` and drag `Triangle` from Scene to Prefab folder

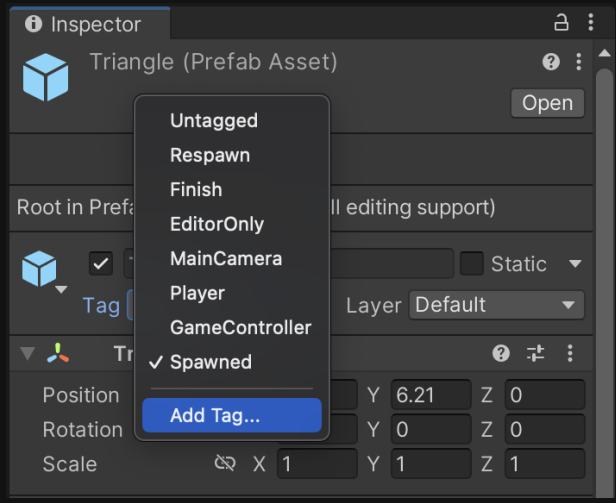
   Delete the `Triangle` from scene



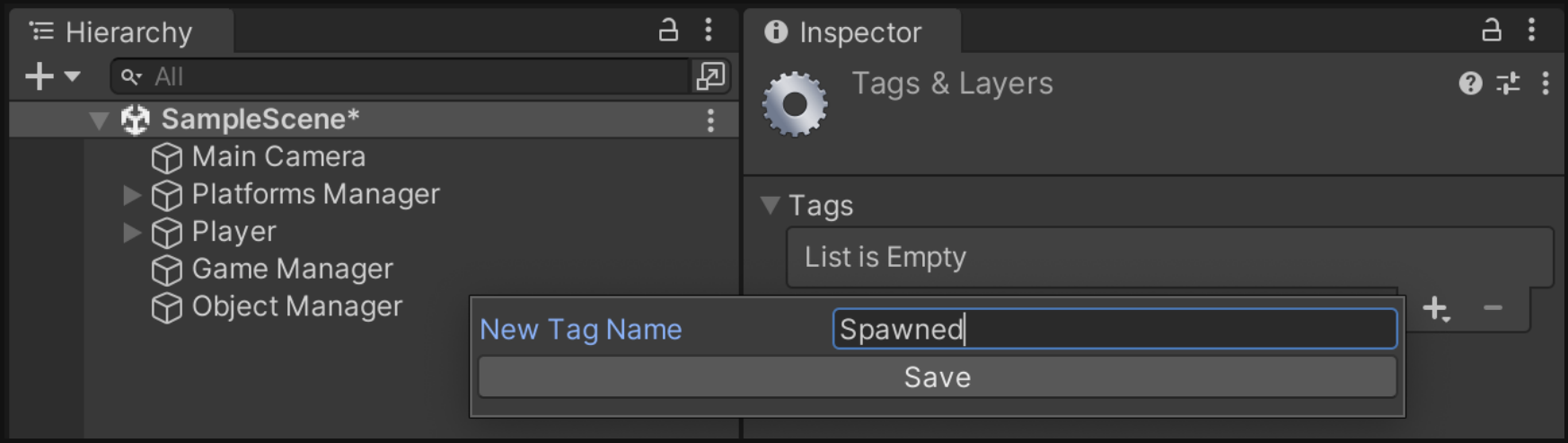
3. Give triangle `RigidBody 2D` and `Box Collider 2D` (Set triangle's `Box Collider 2D`'s `Is Trigger` to true)



4. Add `Spawned` tag to triangle

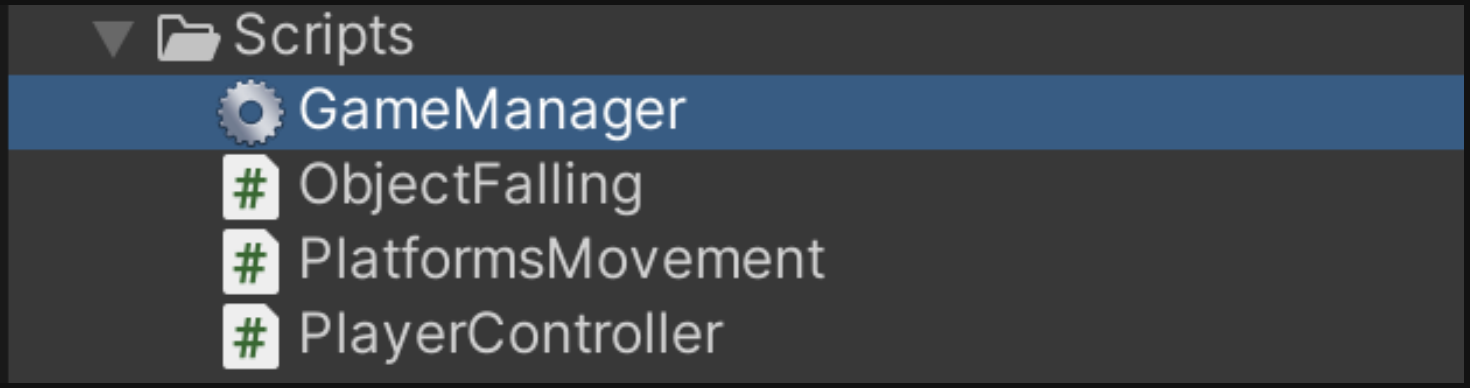


Add tag example



Create the Two Scripts

Create and double click the two scripts



ObjectFalling

Setup our variables

```
// ObjectFalling.cs
[SerializeField] private GameManager gameManager;
[SerializeField] private GameObject triangle;   // The dropping triangle
[SerializeField] private float left, right, up; // Setting the range that the triangles generate
[SerializeField] private float spawningPeriod;  // Spawning period
[SerializeField] private float buttom;          // Buttom of the camera field
private float timer;
```

Create the triangle object instances

```
// ObjectFalling.cs
private void object_Instantiate(GameObject thing) {
    Vector3 position = new Vector3(Random.Range(left, right), up, 0f);
    Quaternion quaternion = Quaternion.Euler(0, 0, Random.Range(0, 360));
    Instantiate(thing, position, quaternion );
}
```

If spawned object is out of range, destroy it.

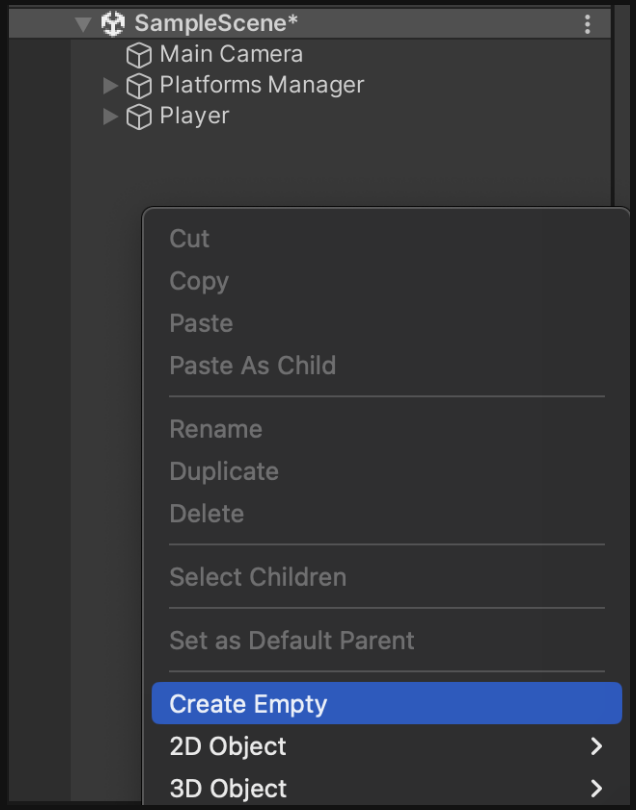
```
// ObjectFalling.cs
private void FixedUpdate() {
    foreach (GameObject spawnedObject in GameObject.FindGameObjectsWithTag("Spawned")) {
        if (spawnedObject.transform.position.y < buttom ) {
            Destroy(spawnedObject);
        }
    }
}
```

Setup a boolean to check if game is ended

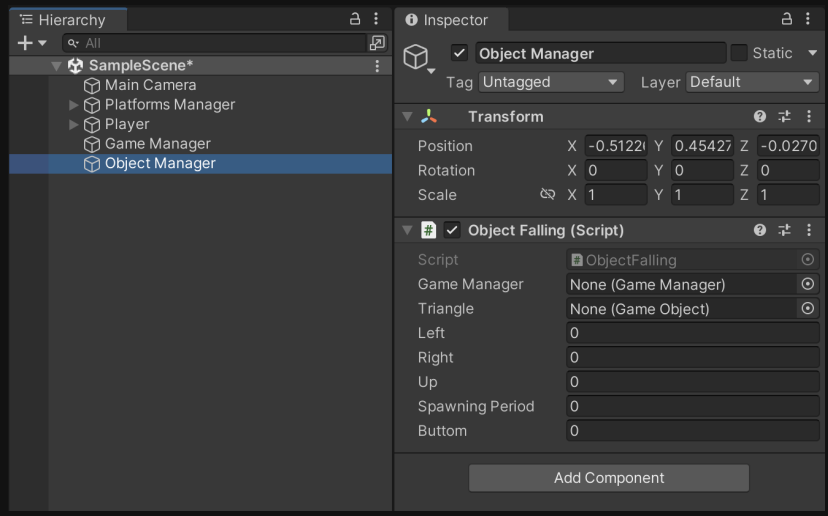
```
// GameManager.cs
public bool isGameover;
```

```
// ObjectFalling.cs
void Update()
{
    if(gameManager.isGameover == false) {   // If game isn't over
        timer += Time.deltaTime;
        if (timer >= spawningPeriod) {      // Generate a new triangle
            object_Instantiate(triangle);
            timer = 0f;
        }
    }

}
```

Create two empty, `Object Manager` and `Game Manager`



Drag `ObjectFalling` script to `Object Manager`, drag `GameManager` script to `Game Manager`.

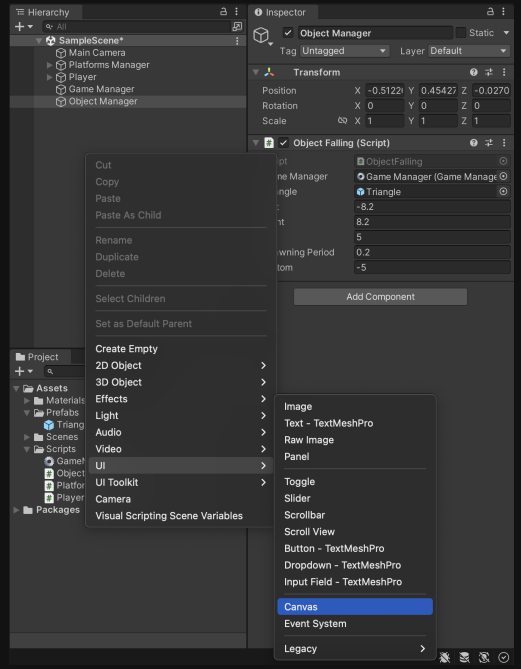


Script variables	GameObject
Game Manager	Game Manager
Triangle	Triangle from `Prefabs` folder
Left	-8.2
Right	8.2
Up	5
Spawning Period	0.2
Buttom	-5

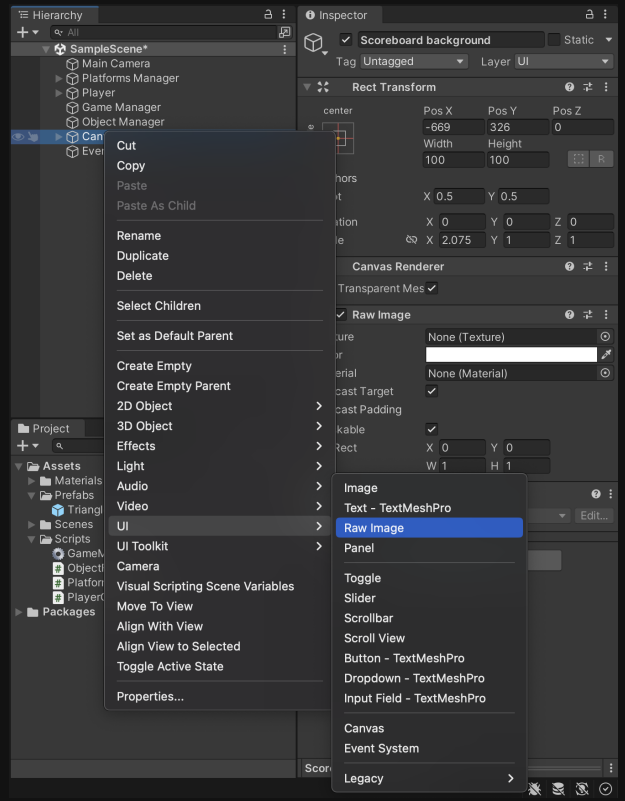
Unity - 5 UI & UX

- Create a Canvas for our UI
- GameManager Script
- PlayerInteract Script Integration

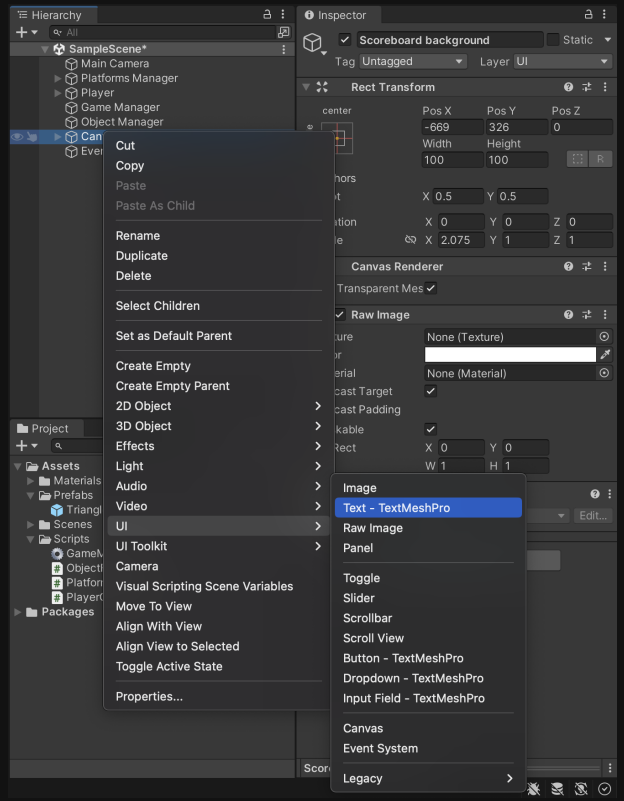
Create a Canvas for our UI



Create a `Raw Image` and rename to `Scoreboard background`



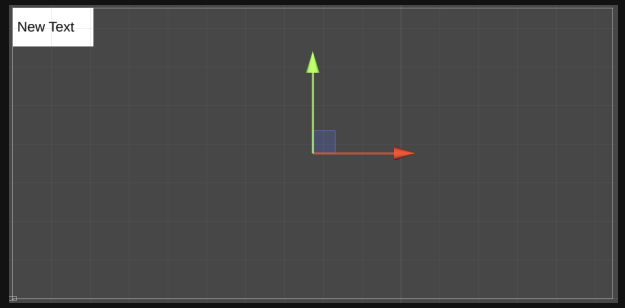
Create a `Text - TextMeshPro` and rename to `Score Text`



Import TMP Essentials if this window pops up

Move those to a place you like

Ex:



Remember to change the text color to black (Choose `vertex color`)

GameManager Script

To use `TMP_Text`, import `TMPro`

```
using TMPro;
```

Setup our variables

```
private int score;      // The score we get
public bool isGameover; // This we have written in the last section
private string CurrentScoreText = "Score: ", FinalScoreText = "Final Score: ";
[SerializeField] private TMP_Text scoreboardUI;     // Score board
```

```
// GameManager.cs
void Start() {
    start_game(); // Call the start game function
}

void Update() {
    scoreboardUI.text = CurrentScoreText + score; // Update current score
}
```

```
// GameManager.cs
private void start_game() {
    isGameover = false;
    score = 0;
}
```

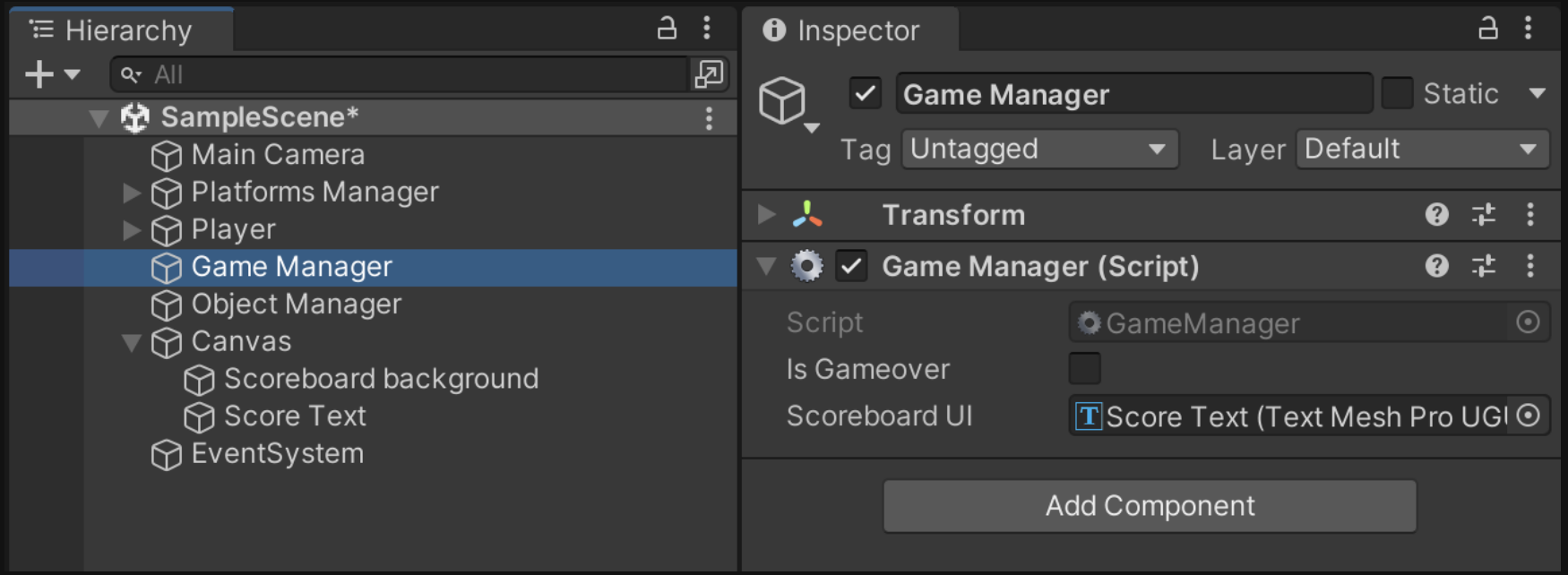
```
public void add_point(int points) {
    if (!isGameover) score += points; // If not game over, update the score.
}
```

PlayerInteract Script Integration

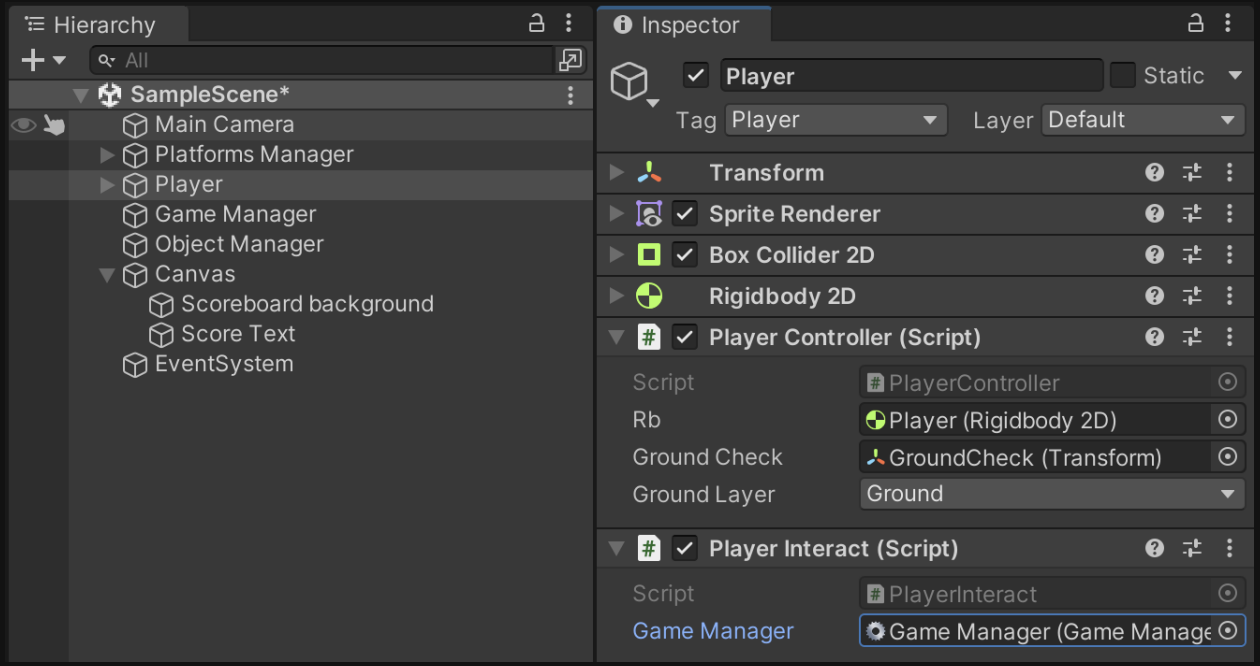
Create a `PlayerInteract` script and double click to edit

```
// PlayerInteract.cs (This script will be put on `player`)

[SerializeField] private GameManager gameManager;

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Spawned") {
        gameManager.add_point(5);
        Debug.Log("Add Point");
        Destroy(other.gameObject);
    }
}
```

Now, go back to Unity editor, grab `Score Text` to `Scoreboard UI`



Then, drag `PlayerInteract` to `Player` (Now player will have 2 scripts, `PlayerController` and `PlayerInteract`)
Drag `Game Manager` to `Game Manager`

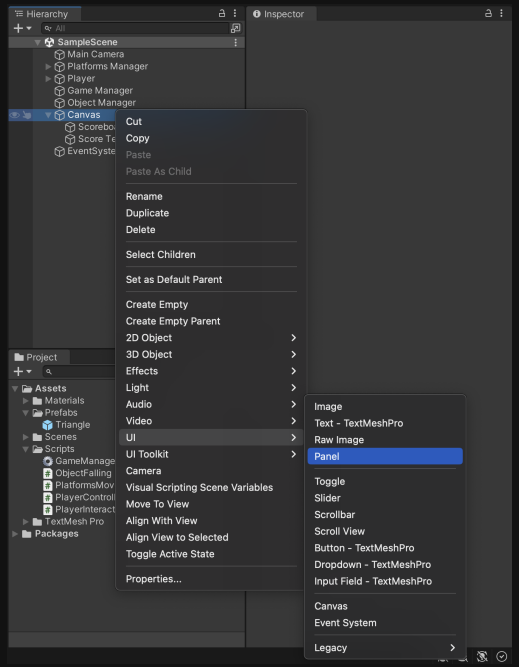


Unity - 6 - Restart

- Restart Panel
- Fine Tuning UI - 1
- Script
- Scripts Setup
- Fine Tuning UI - 2

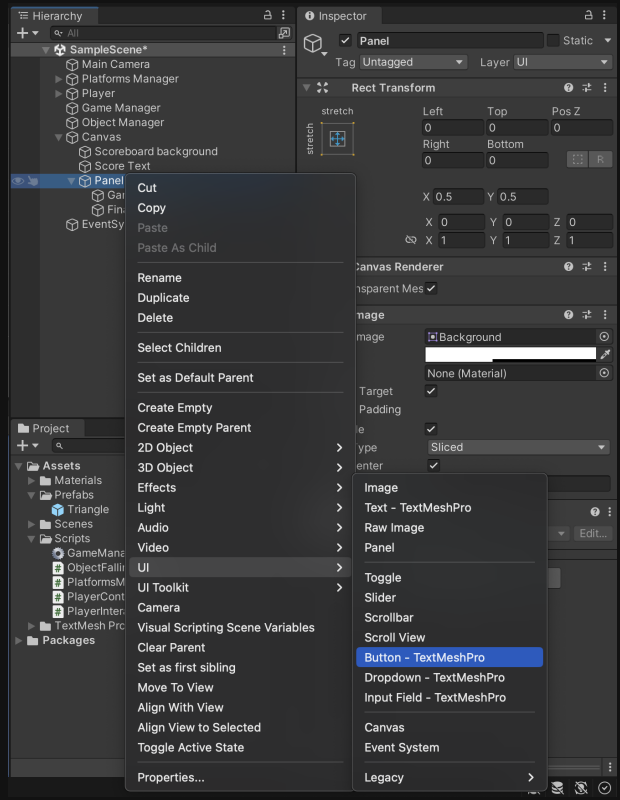
Restart Panel

First, create `Panel` inside `Canvas`, this will be the panel that shows when the game ends

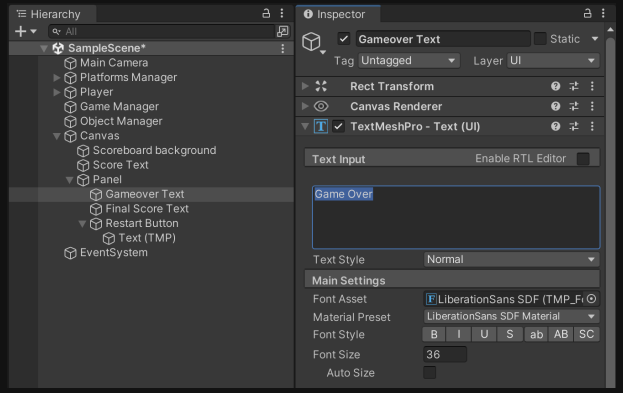


Under `Panel`, create three components:

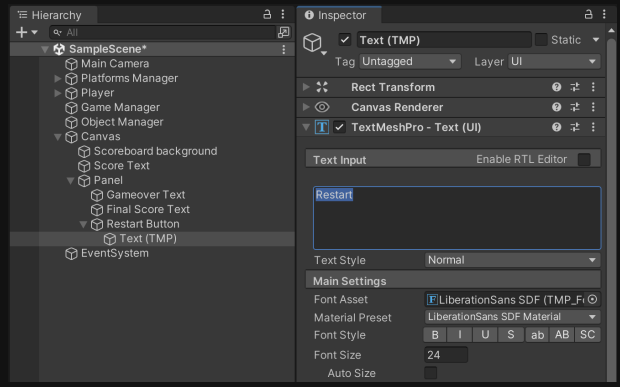
1. Gameover Text (`UI/Text - TextMashPro`)
2. Final Score Text (`UI/Text - TextMashPro`)
3. Restart Button (`UI/Button - TextMeshPro`)



Change the text inside `Gameover Text`



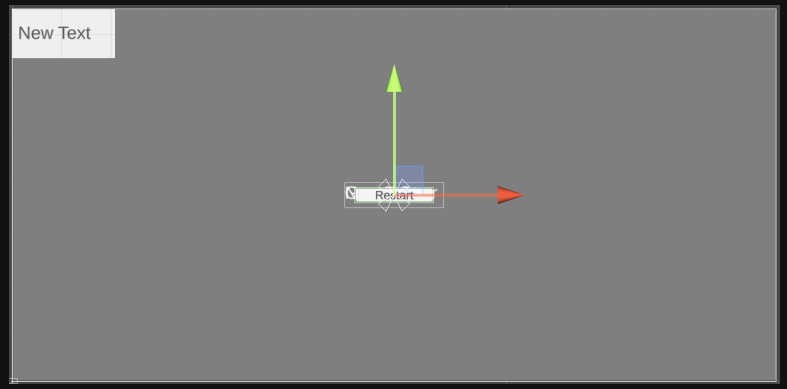
Similarily, change the text of `Restart Button` to `Restart`



Fine Tuning UI - 1

Reset the three components’ position

Before:



After:

Script

```
// GameManager.cs
using UnityEngine.UI; // Import the UI toolkit for Button

// We add the following before
private int score;
public bool isGameover;
private string CurrentScoreText = "Score: ", FinalScoreText = "Final Score: ";
[SerializeField] private TMP_Text scoreboardUI;

// New variables
[SerializeField] private GameObject gameoverPage;   // The final panel
[SerializeField] private TMP_Text finalScore;       // Final score
[SerializeField] private Button restartButton;      // Restart button
```

1. Set the panel to `non-active` when the game start.
2. Set the panel to `active` when the game end.

```
// GameManager.cs
private void start_game() {
    isGameover = false;
    gameoverPage.SetActive(false); // Set the final page to non-active when game start
    score = 0;
}

public void game_over() {
    isGameover = true; // Set isGameover to true
    finalScore.text = FinalScoreText + score; // Set up final score
    gameoverPage.SetActive(true); // Activate final panel
    restartButton.onClick.AddListener(start_game); // Listen the restart button
}
```

Setup the rebirth-mechanics

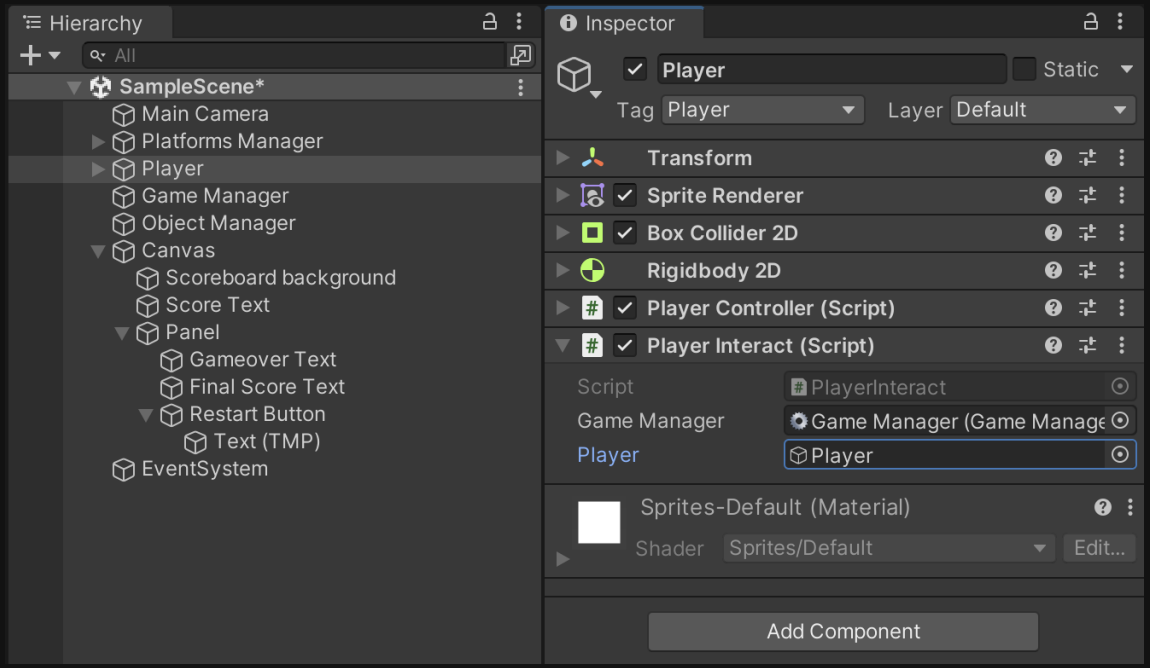
```
// PlayerInteract.cs
[SerializeField] private GameManager gameManager; // We setup this before
[SerializeField] private GameObject player;

private void reset_player() {
    player.transform.position = new Vector3(0, 0, 0);
}

void Update() {
    if(player.transform.position.y < -5) { // If player dropped
        reset_player();
        gameManager.game_over();
        Debug.Log("Fall");
    }
}
```

Recall the integration of `PlayerInteract` and `GameManager`

```
// PlayerInteract.cs
void Update() {
    if(player.transform.position.y < -5) { // If player dropped
        reset_player();
        gameManager.game_over();
        Debug.Log("Fall");
    }
}

// GameManager.cs
public void game_over() {
    isGameover = true; // Set isGameover to true
    finalScore.text = FinalScoreText + score; // Set up final score
    gameoverPage.SetActive(true); // Activate final panel
    restartButton.onClick.AddListener(start_game); // Listen the restart button
}
```

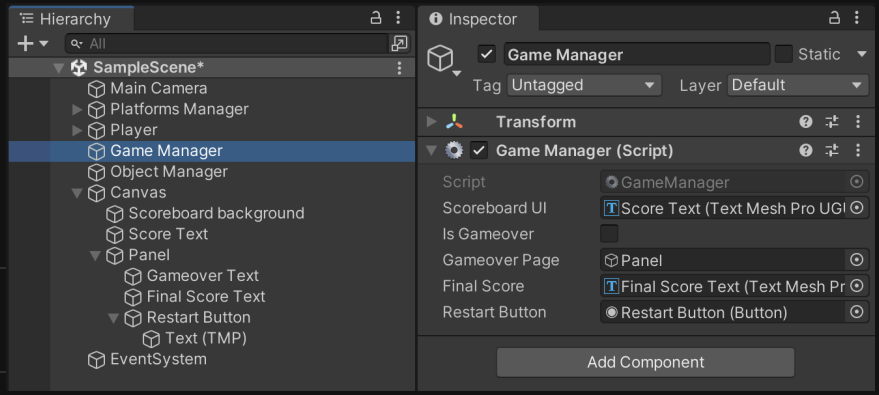
Scripts Setup

Go back to Unity editor and setup player for `PlayerInteract`



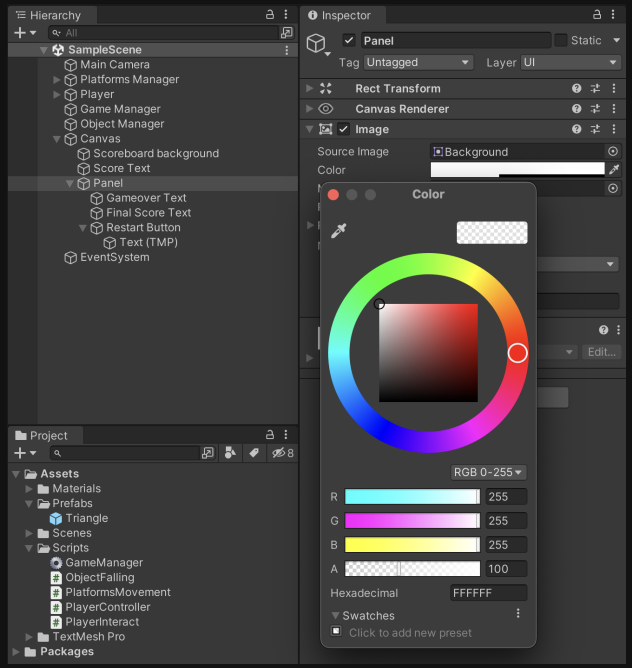
Setup `Panel` and its child components for `GameManager`

GameManager.cs	GameObject
Gameover page	Panel
Final score	Final Score Text
Restart Button	Restart Button

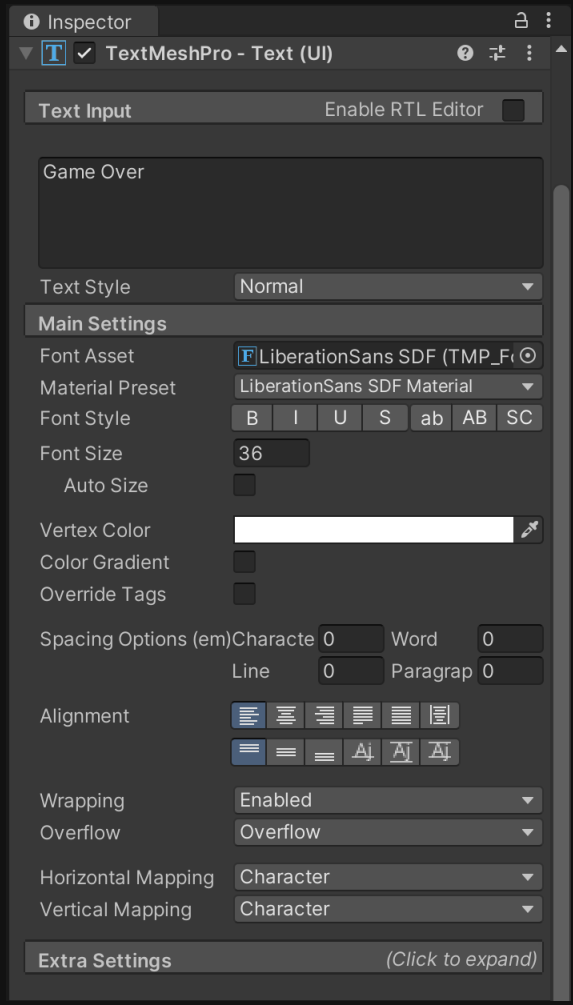


Fine Tuning UI - 2

To change the background color of panel, you can select `panel` and change it's Image color



To change the text fonts, layout or color, choose the `TextMeshPro` component and edit it in the Inspector windows



Resources

(Click to downloads)

- Slides (dark)
- Slides (light)
- GameManager.cs
- GameManager.cs
- GameManager.cs
- GameManager.cs
- GameManager.cs

Learn More

Unity · Showcases