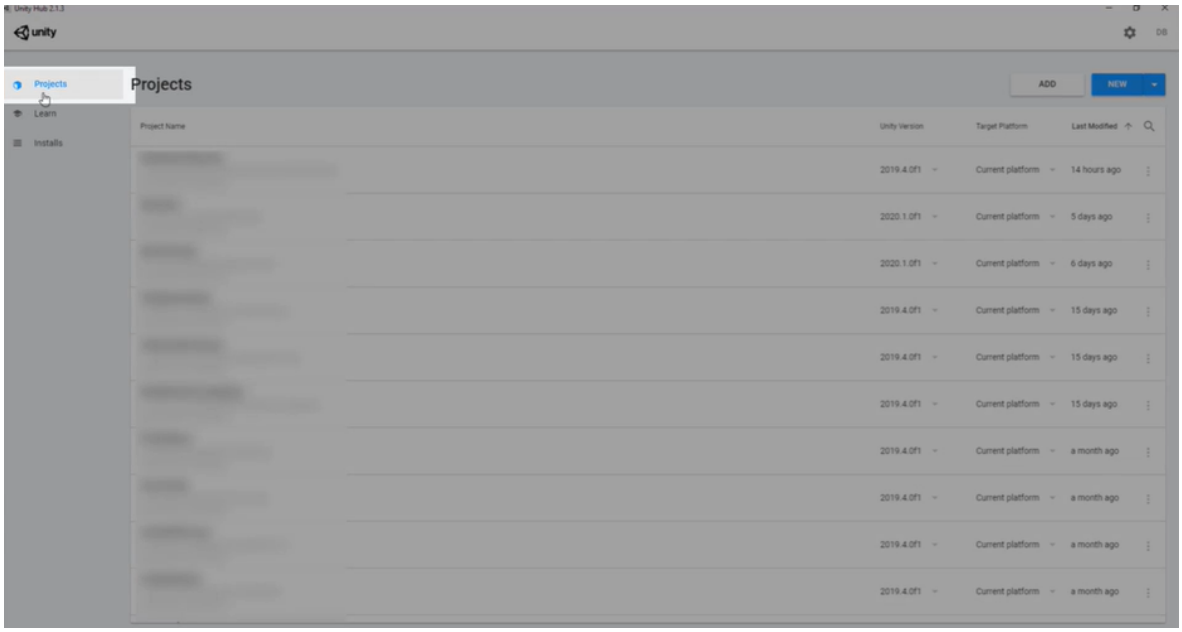
In this lesson, we're going to be setting up the project '**Mobile Road Crossing Game'** in Unity.

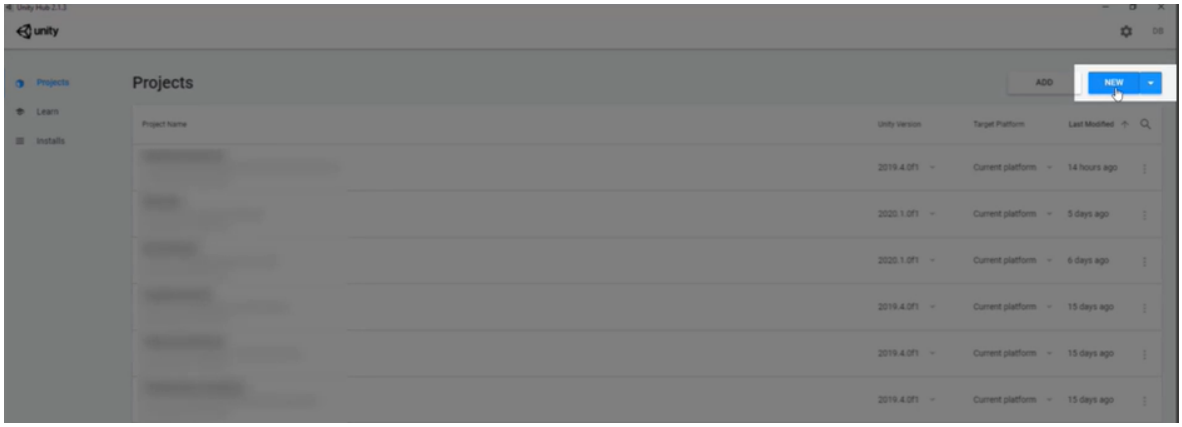
You need to have the latest version of **Unity Hub** installed if you haven't already. Download Unity Hub: https://unity3d.com/get-unity/download

Creating Unity Project

Open up Unity hub, and click on the **Projects** tab:

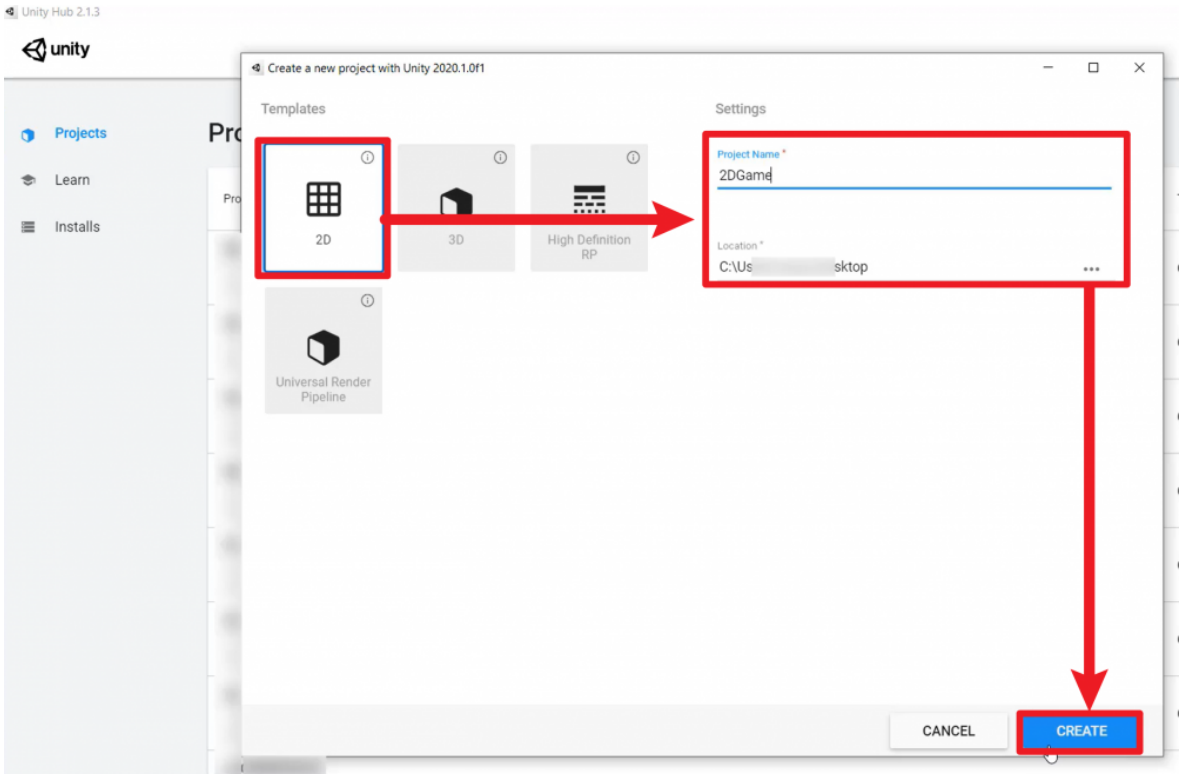


... and click on the **New** button**:**

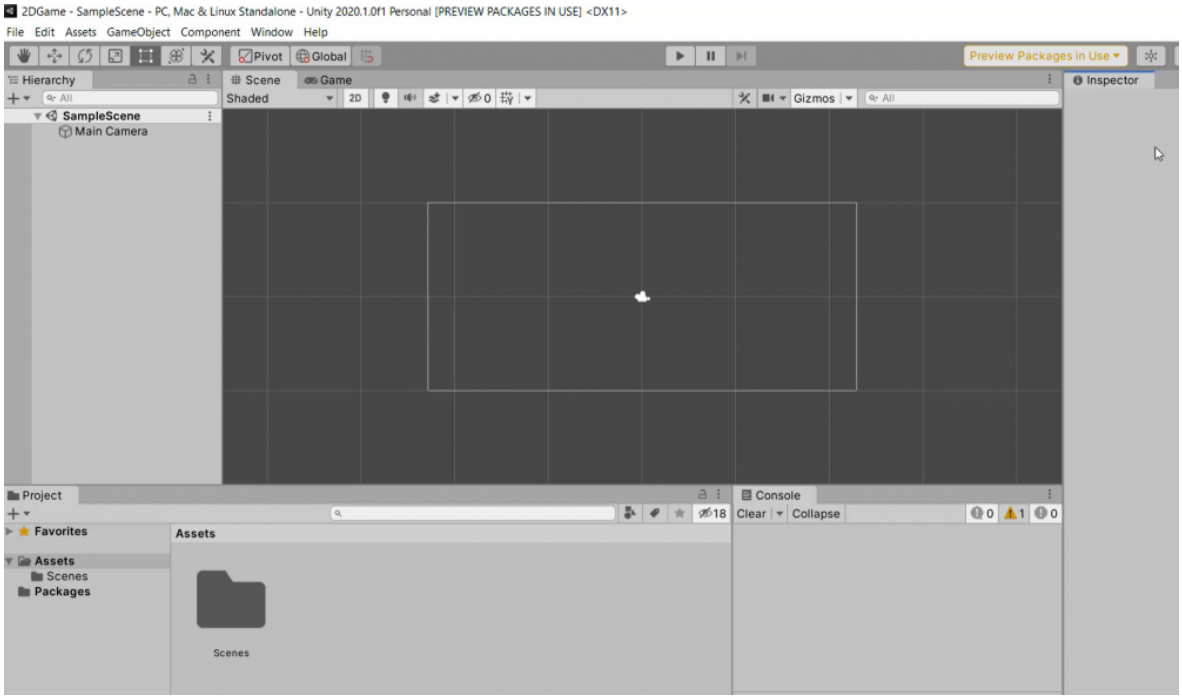


Select the **2D template**, and set the project's name (e.g. **RoadCrossingGame**) and location. Then click on the '**Create**' button.

© 2020 Zenva Pty Ltd. All rights reserved.



We now have an empty **2D Scene** open inside of our brand new Unity project.

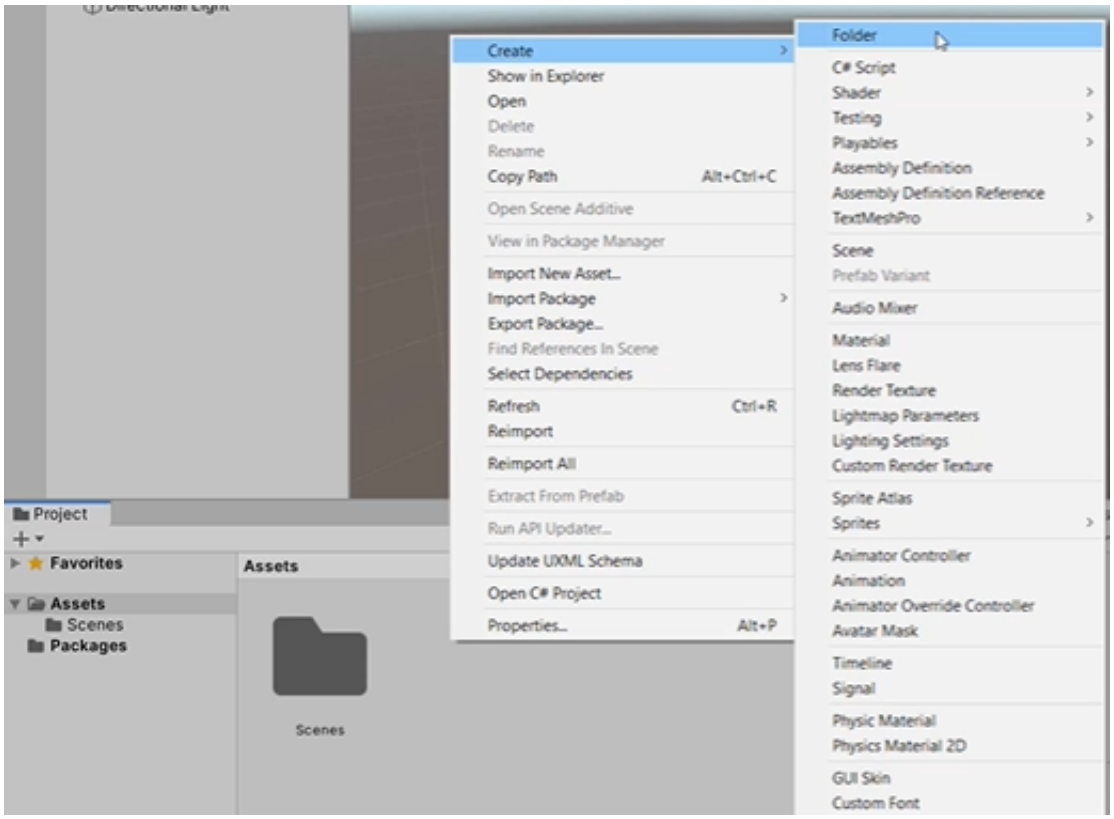


Importing Sprites

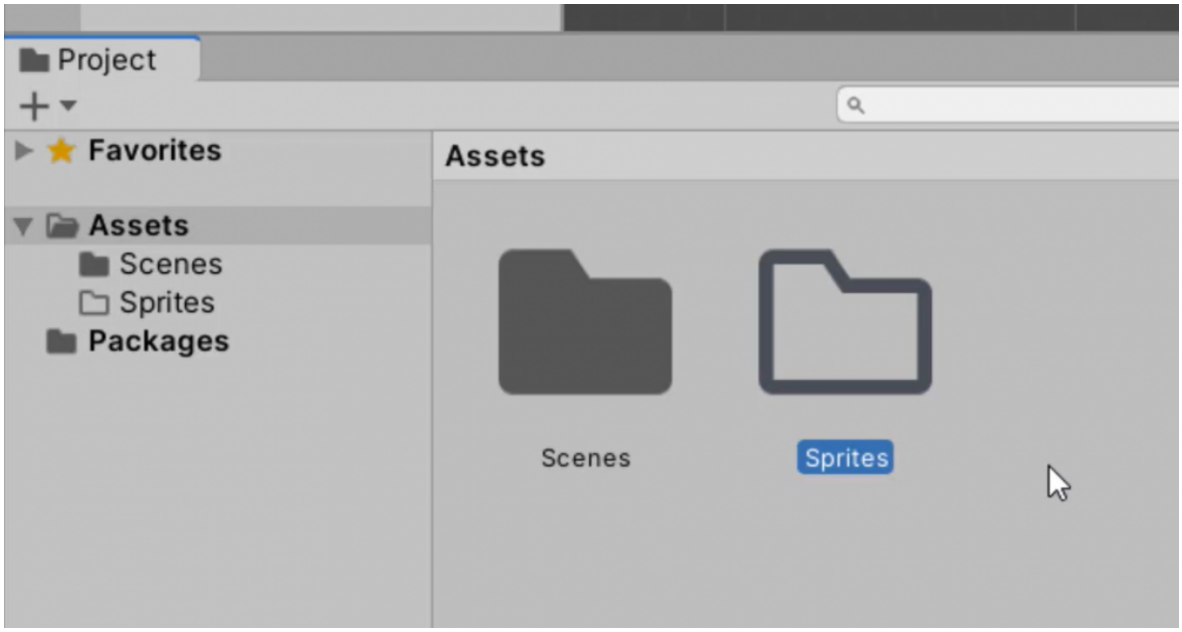
First of all, we're going to **create a new folder** to store our 2D sprites.

Right-click on the project panel, and go to **Create > Folder**.

© 2020 Zenva Pty Ltd. All rights reserved.



Then we're going to **name** the folder as "Sprites".



Next, we're going to download the **Assets.zip** file from the **Course Files** tab.

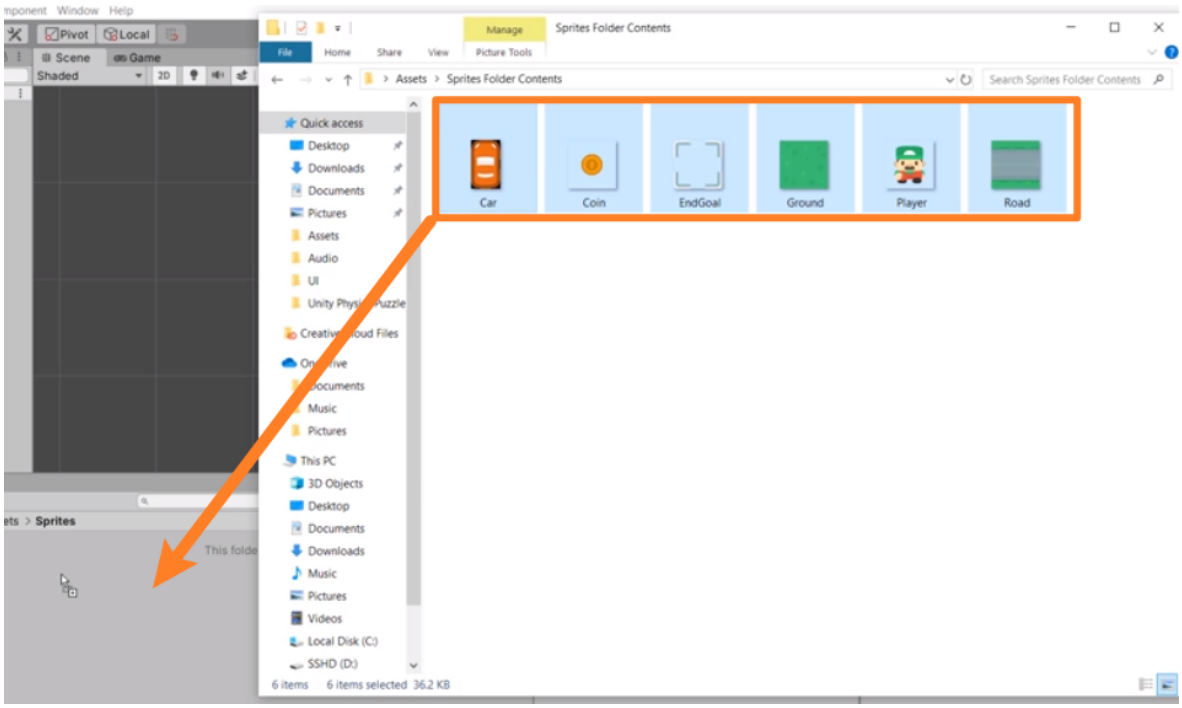
© 2020 Zenva Pty Ltd. All rights reserved.

Course files

Build a Mobile Road Crossing Game - Assets

Build a Mobile Road Crossing Game - Course Files

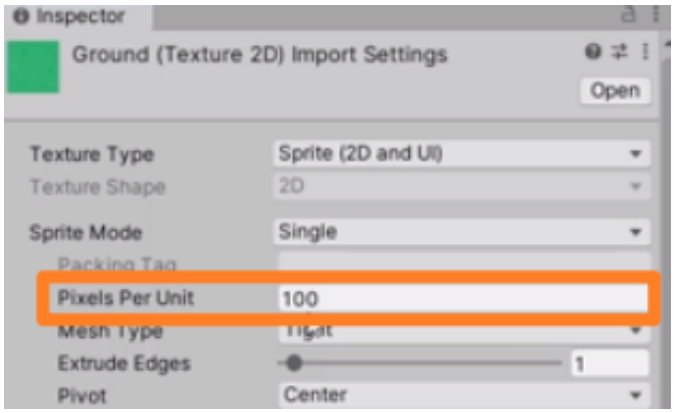
Then we're going to select the files inside the extracted **.zip** file and **drag** them into the Unity Project tab.



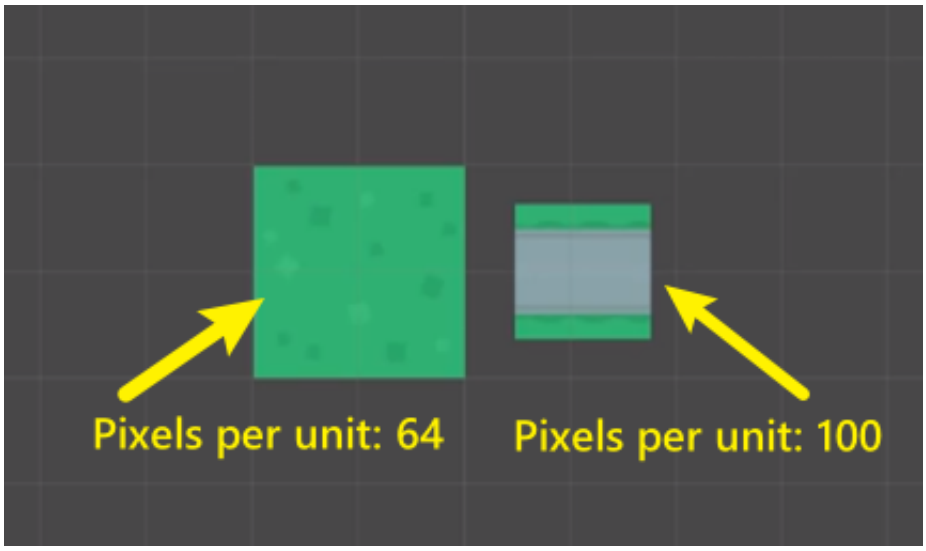
Now we've successfully imported the files into our project. Note that these assets are sourced from a public domain game asset website called **Kenny.nl** (https://www.kenney.nl/assets)

When you click on one of the sprites, you'll see that **Pixels Per Unit** is set to 100 by default.

© 2020 Zenva Pty Ltd. All rights reserved.

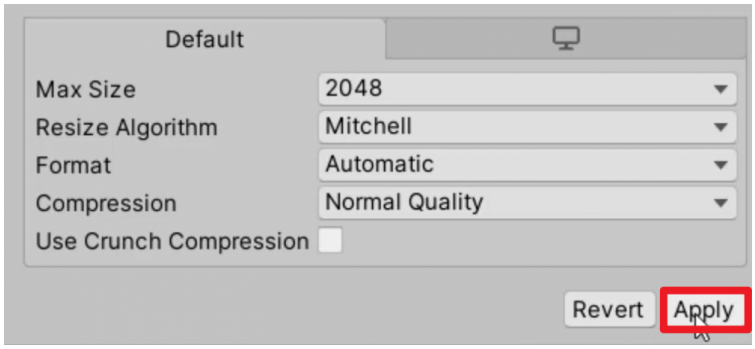


This means that for every 100 pixels wide it is going to equal one unit in Unity, and since our sprites dimensions are **64 x 64** pixels, they will appear smaller than one unit.



(Each of these grid tiles is half a unit long in Unity.)

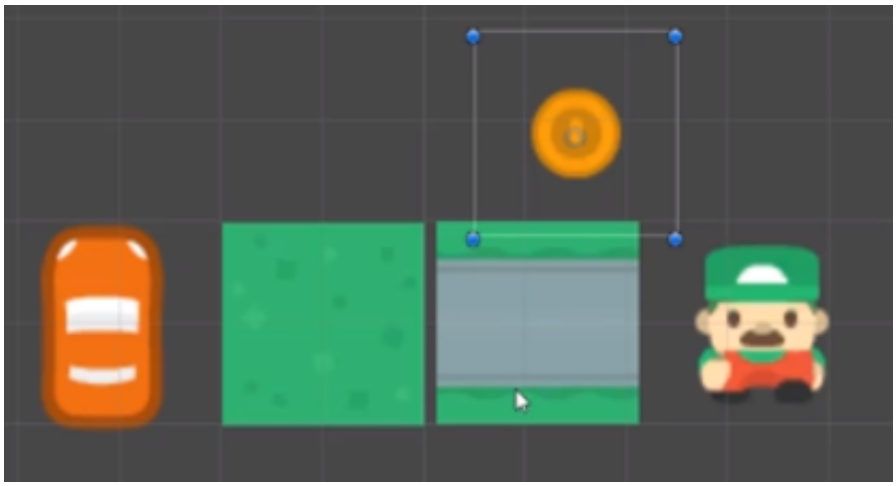
We're going to **select all** the Sprites (Ctrl+A or Command+A), change the **Pixels Per Unit** value to be 64, and click on the **Apply** button.



Now when you **drag in** any sprite into the **Scene** view, you'll see that it takes up exactly two

© 2020 Zenva Pty Ltd. All rights reserved.

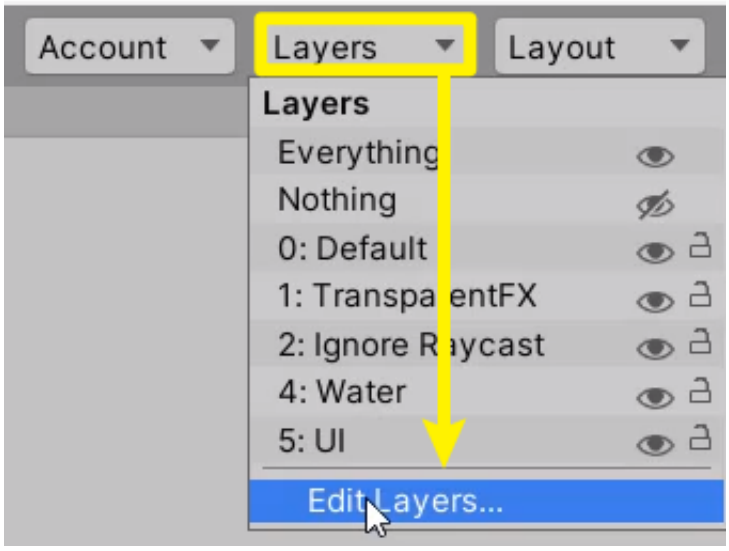
squares for each side, which is equal to **1 x 1** in Unity.



Setting up Sorting Layers

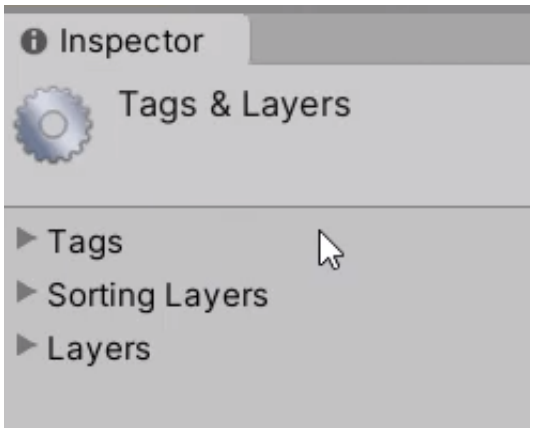
In Unity, every Sprite has a default **sorting layer**, and it helps us determine which sprites are going to be rendered on top of others.

To edit Sorting Layers, select **Layers > Edit Layers** from the top-right corner of the screen.

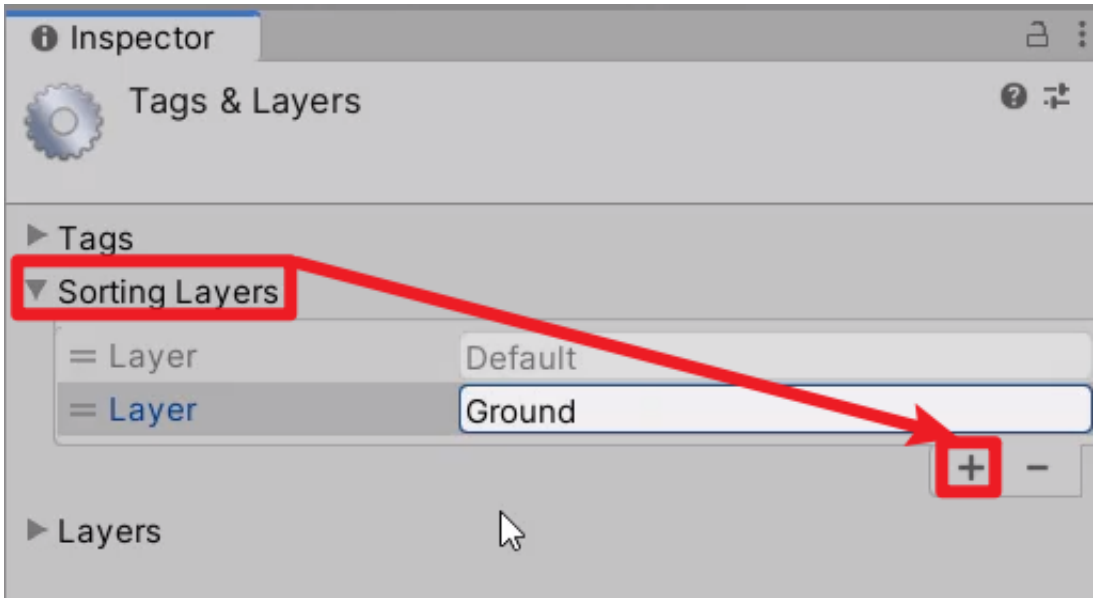


This will open up the **Tags & Layers** window in the Inspector.

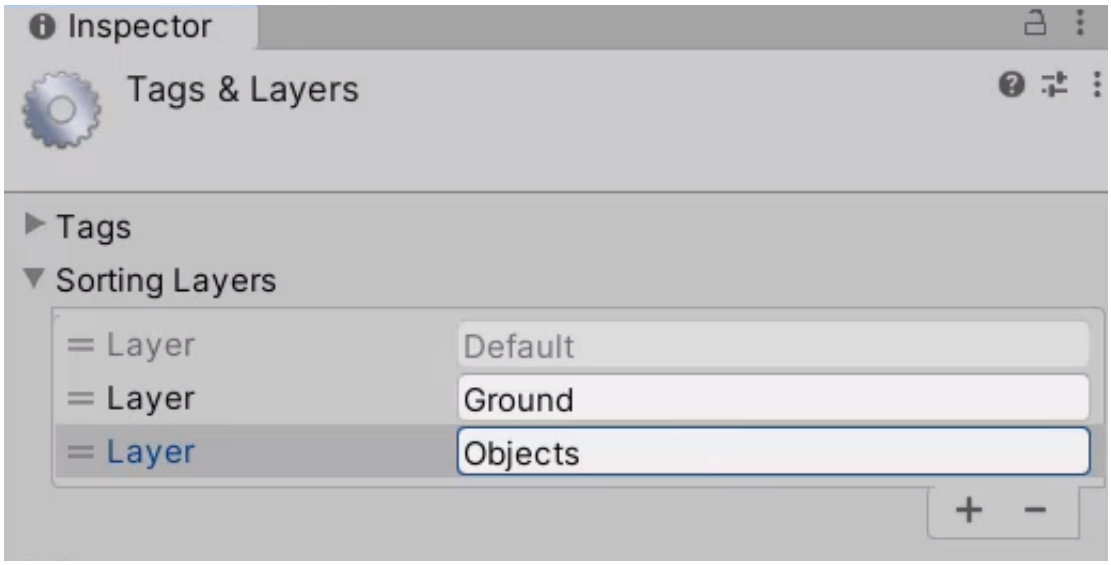
© 2020 Zenva Pty Ltd. All rights reserved.



Let's open up the Sorting Layers drop-down and **create a new layer** called "Ground". This is what we're going to assign to the **Ground** sprite and the **Road** sprite.

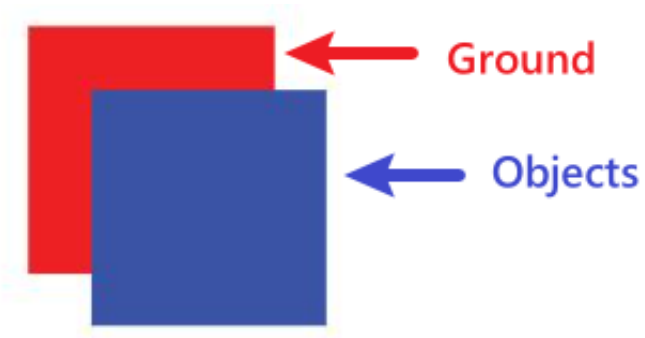


We're going to create another layer called "**Objects**" and assign it to the **Player**, **Coin**, **Car**, and **EndGoal**.



© 2020 Zenva Pty Ltd. All rights reserved.

Since this layer is underneath Ground, this basically means that it is going to be rendering **on top** of all the Ground sprites.



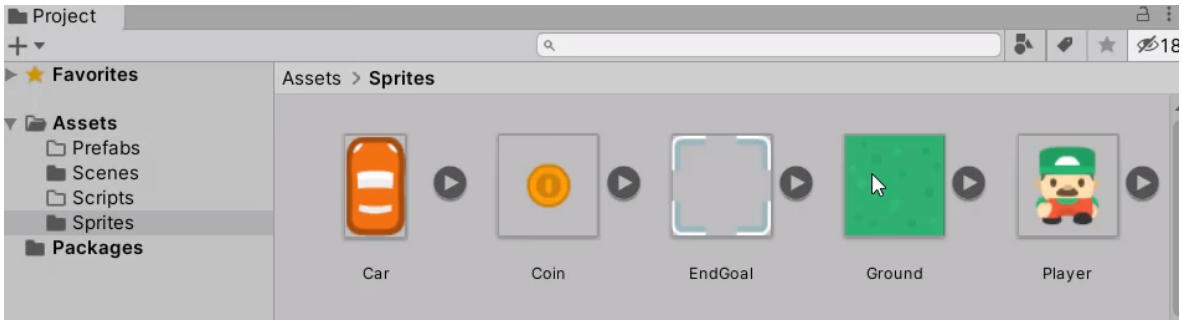
© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be setting up a Player **GameObject***; this is going to be the object which is going to act as our player.

*(A **GameObject** is the base class for all entities in Unity that you can attach various different components onto.)

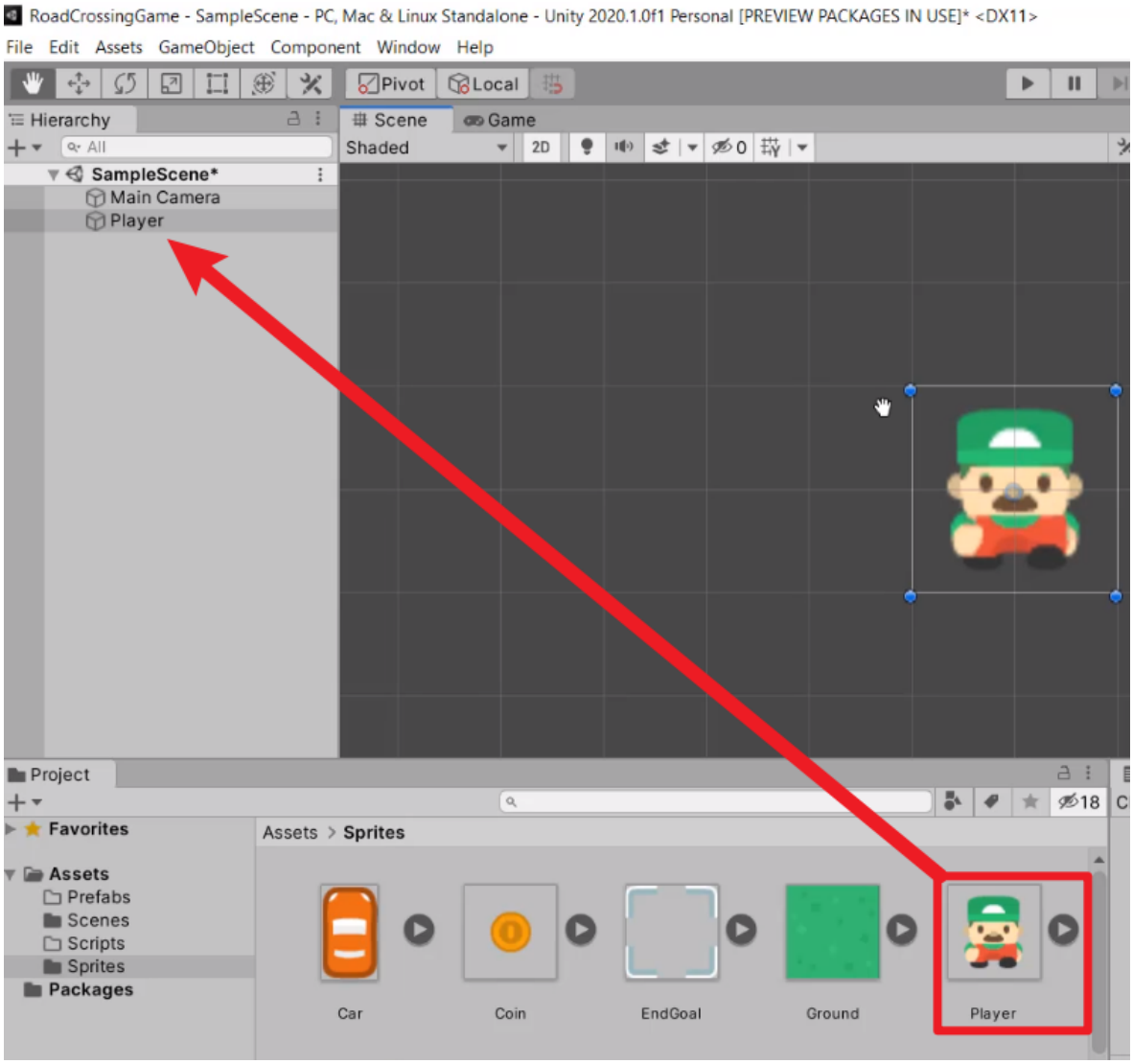
Creating A GameObject

Let's open up the **Assets > Sprites** folder,

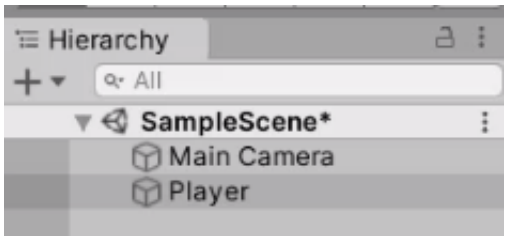


and **drag** our player sprite into the **Hierarchy**.

© 2020 Zenva Pty Ltd. All rights reserved.



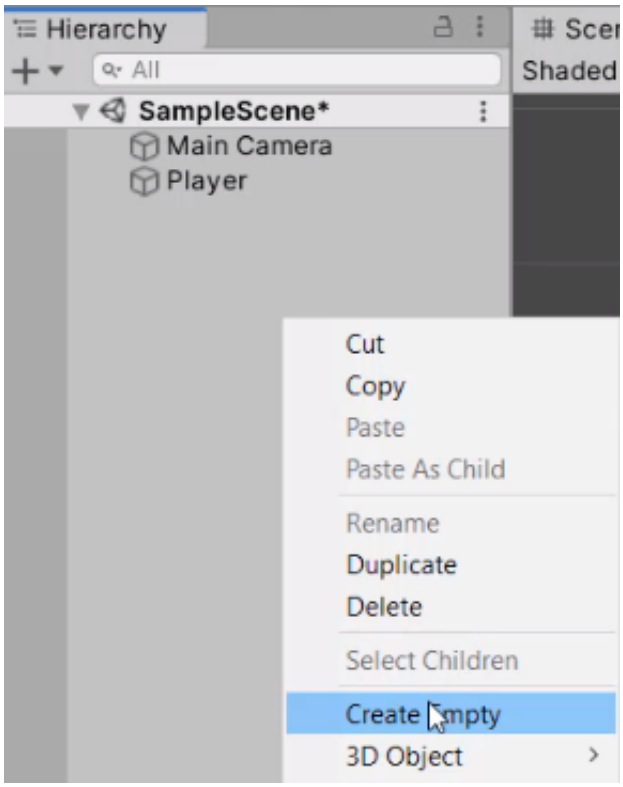
This is going to create a brand new **GameObject** called Player.



Adding Components

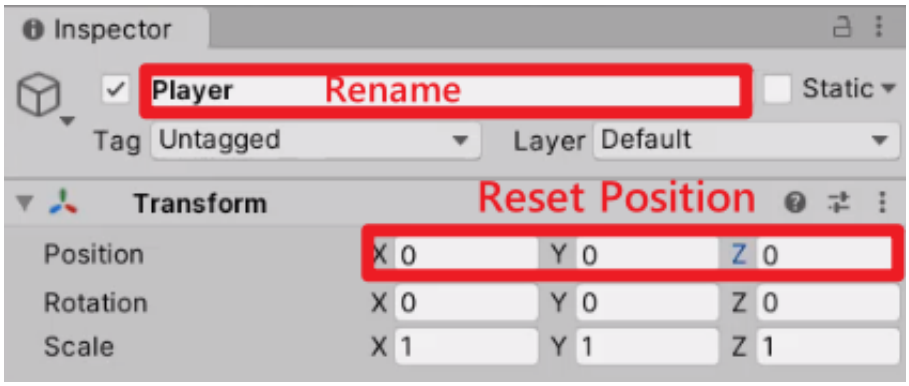
We're going to create a new **empty** GameObject **(Right-click > Create Empty)**

© 2020 Zenva Pty Ltd. All rights reserved.

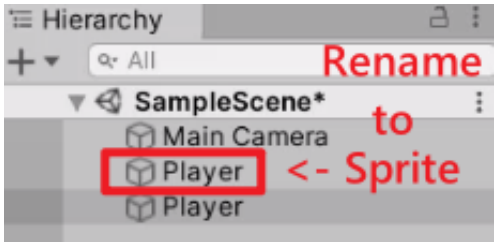


(It's generally good practice to have the most parent object to be **empty** as it allows you to finely adjust the child sprite position and pivot.)

We're going to **rename** this GameObject as "Player", and reset its position to 0, 0, 0.

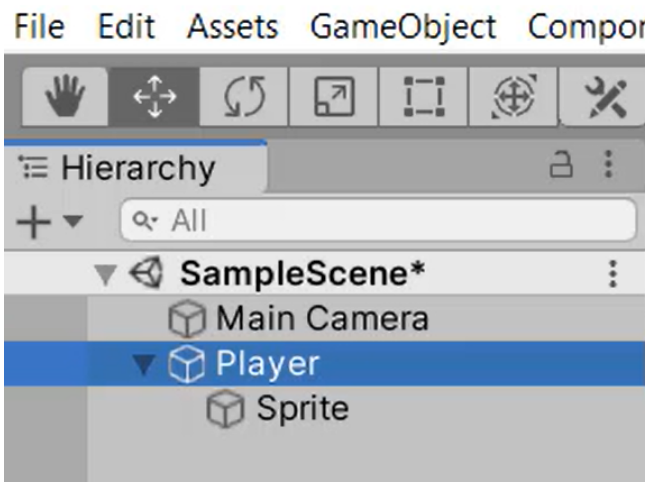


And then, we're going to **rename** our player sprite to "Sprite"

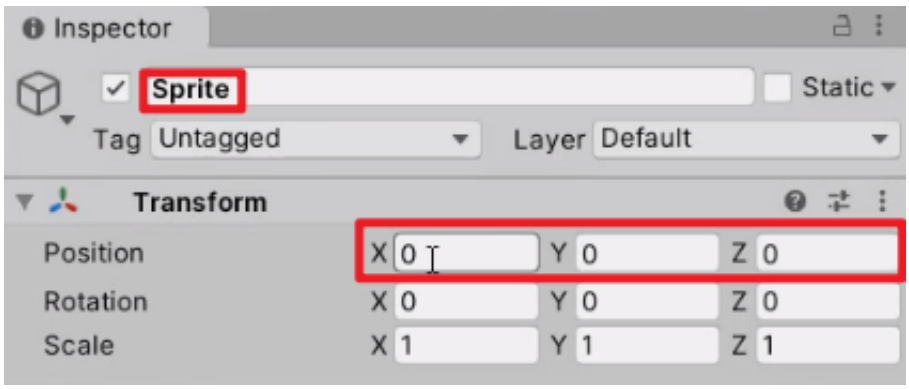


and **drag** it in as a **child** of "Player".

© 2020 Zenva Pty Ltd. All rights reserved.

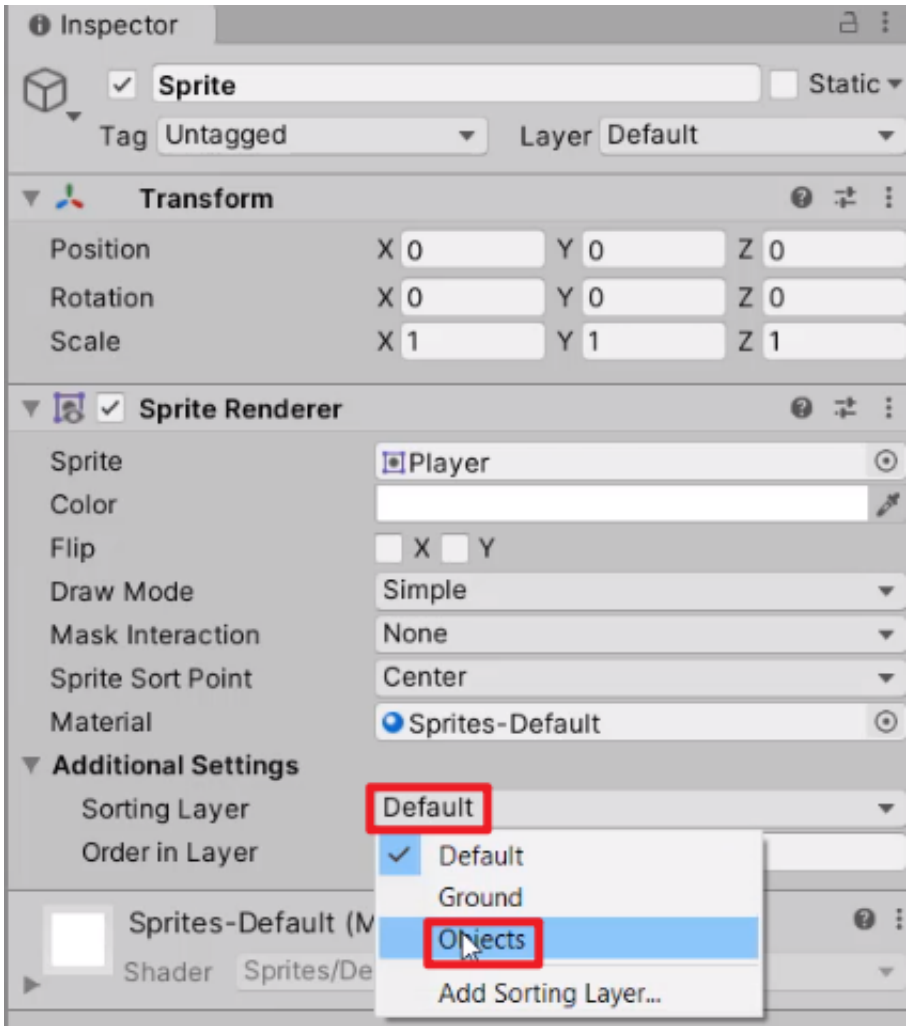


Make sure that the Sprite position is at **0,0,0** as well.



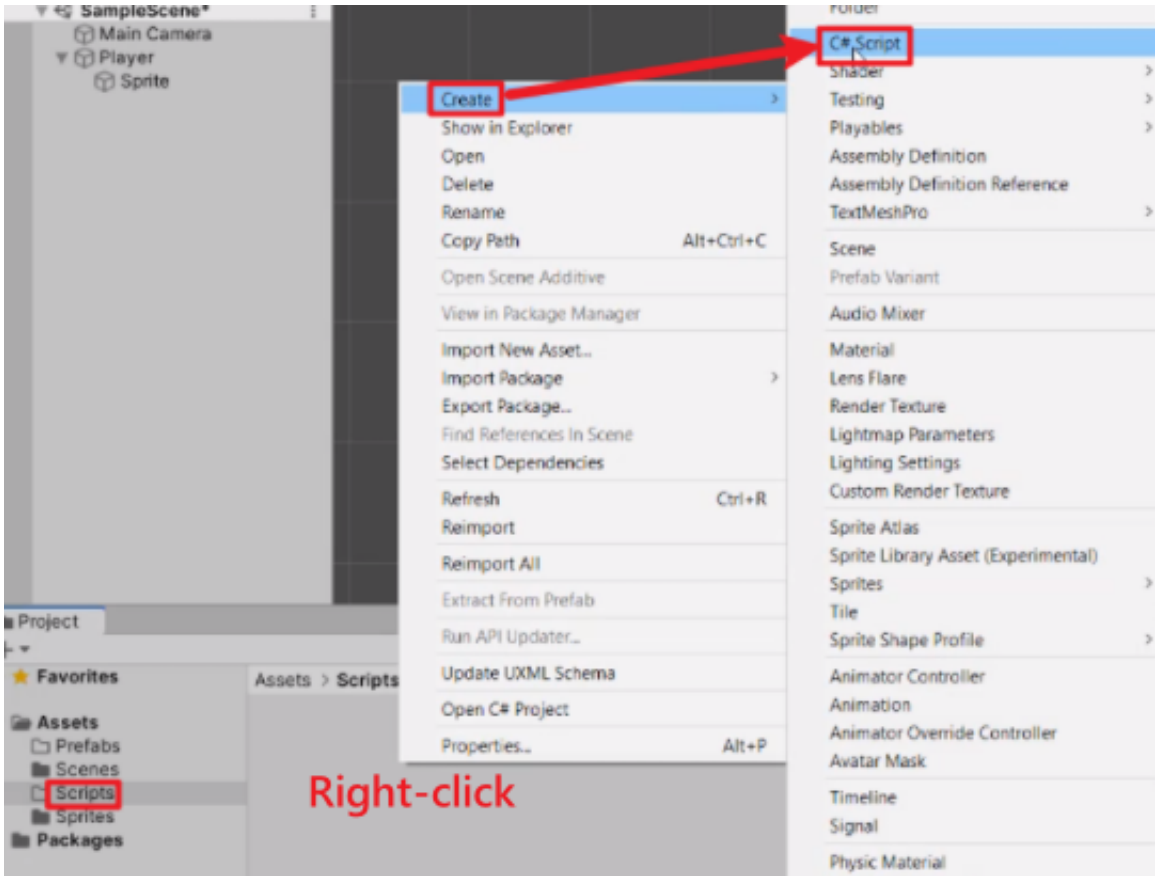
To render this object on top of the **Ground** sprite, we're going to change the **sorting layer** from "Default" over to "**Objects**".

© 2020 Zenva Pty Ltd. All rights reserved.



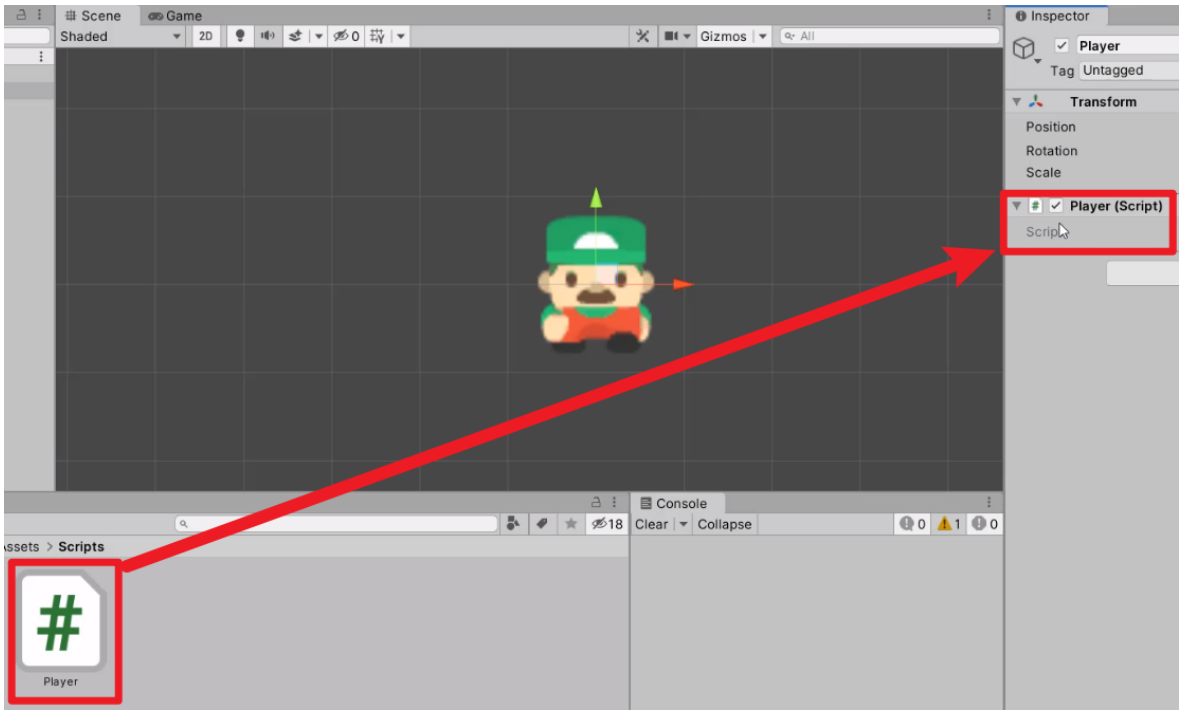
Now we're going to create a new **C# script** called "Player". In our Scripts (Assets > Scripts) folder, **Right-click** > **Create** > **C# Script**:

© 2020 Zenva Pty Ltd. All rights reserved.

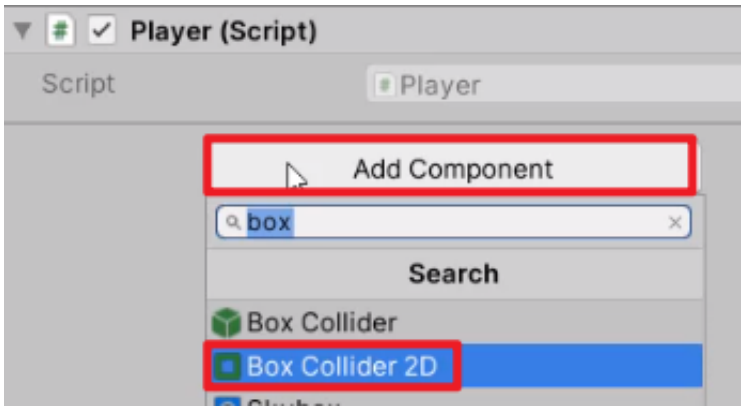


Select the Player script, and **drag** that into the Player **Inspector**.

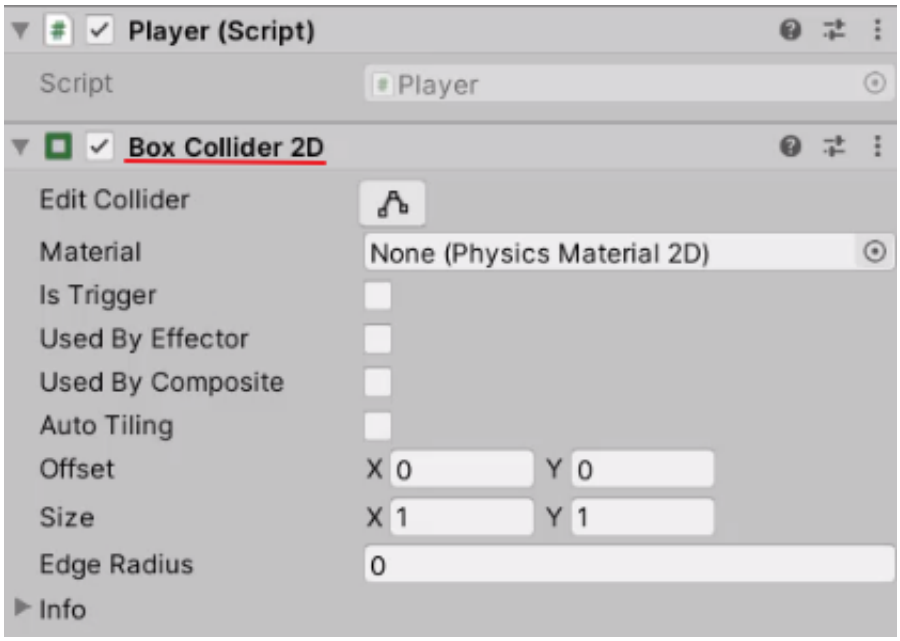
© 2020 Zenva Pty Ltd. All rights reserved.



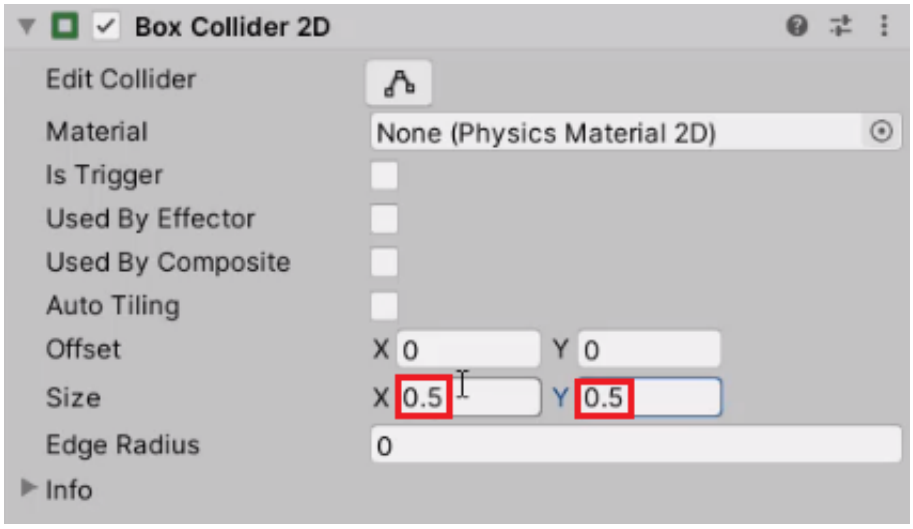
To add in a Box 2D Collider, click “**Add Component**” > type “**Box**” > select “**Box Collider 2D**“.

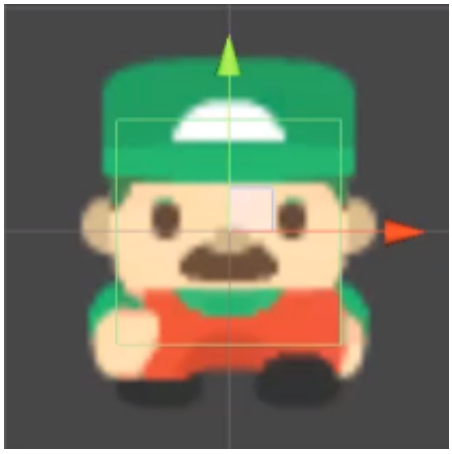


© 2020 Zenva Pty Ltd. All rights reserved.



And then, bring the size on **X and Y down to 0.5** in order to make the object a bit smaller than the actual sprite.

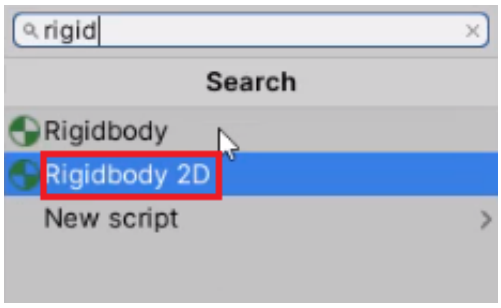


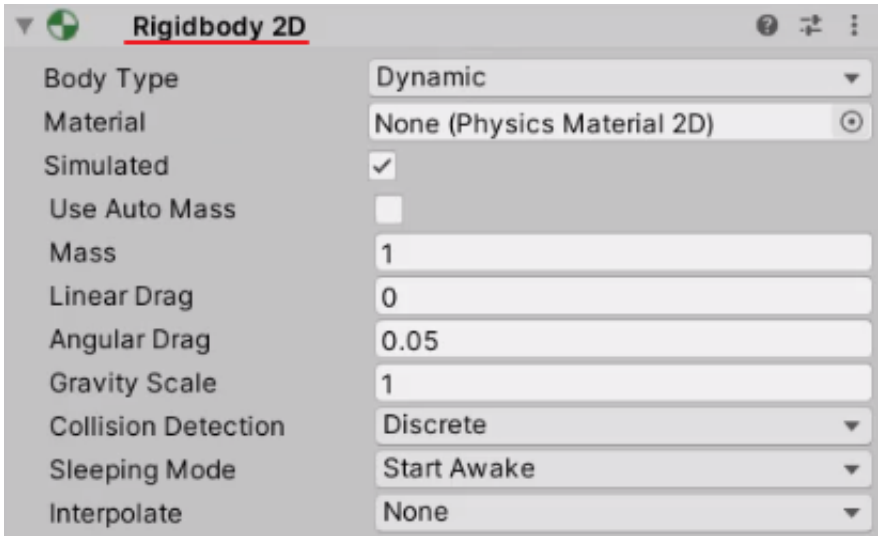


We're now going to add a **Rigidbody2D.** To do that, click on **Add Component >** type "**Rigid**" >

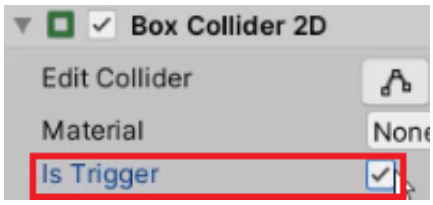
© 2020 Zenva Pty Ltd. All rights reserved.

select “**Rigidbody 2D**“

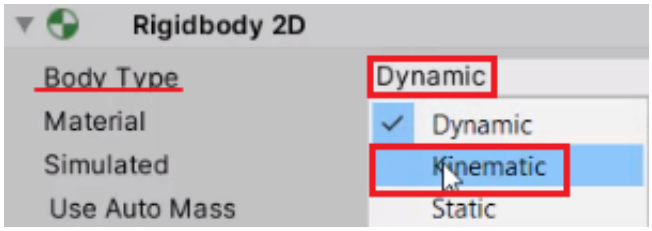




Let’s also make a Box Collider 2D a **Trigger**, so that we can actually go through other colliders.

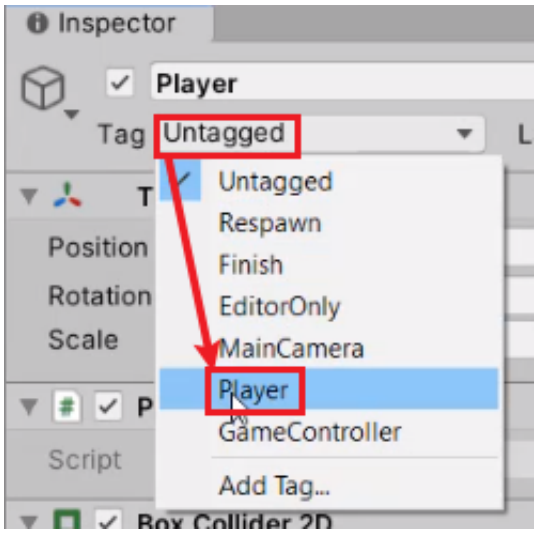


And down in our Rigidbody 2D, change **Body Type** from “**Dynamic”** to “**Kinematic”**. Kinematic behavior basically prevents the Rigidbody2D from reacting to gravity or applied forces.

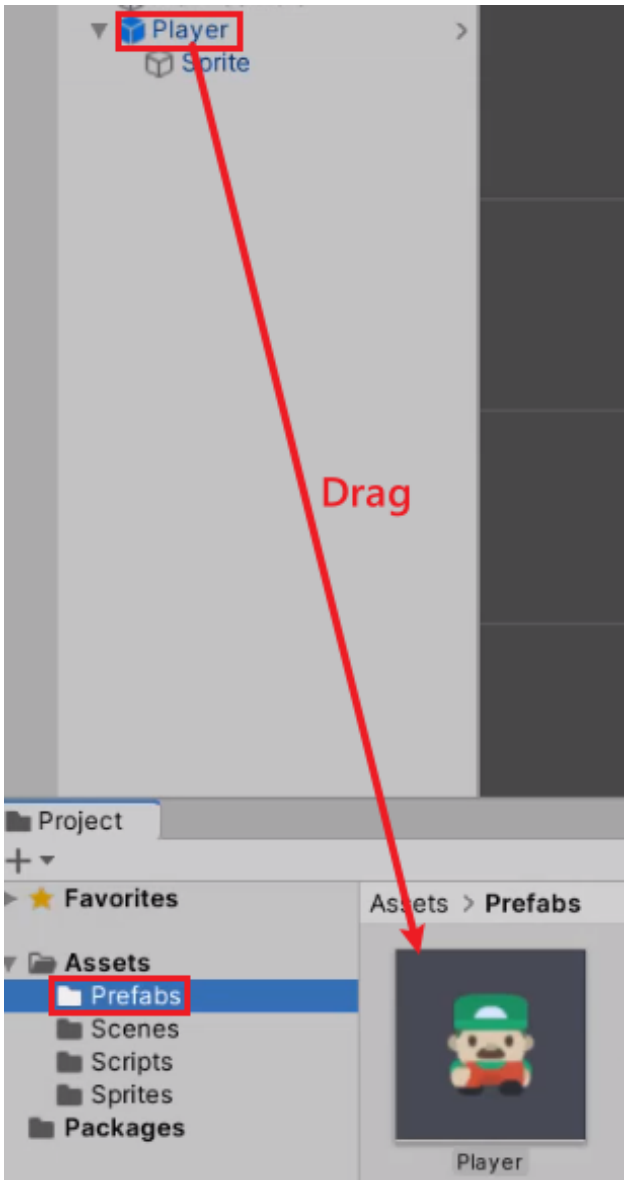


Finally, let’s go ahead up in the **Inspector** and change the **Tag** of the player over to “**Player**“.

© 2020 Zenva Pty Ltd. All rights reserved.



Save the settings, and then go over to the **Prefabs** folder, drag **Player** in as a Prefab so you can use it in the future levels.



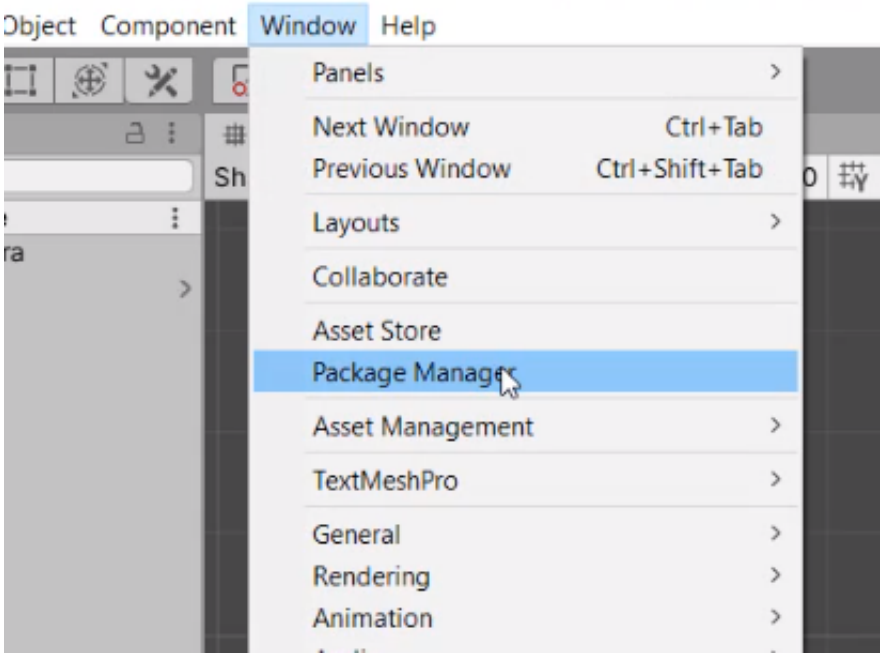
© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be looking at the **Device Simulator** inside of Unity.

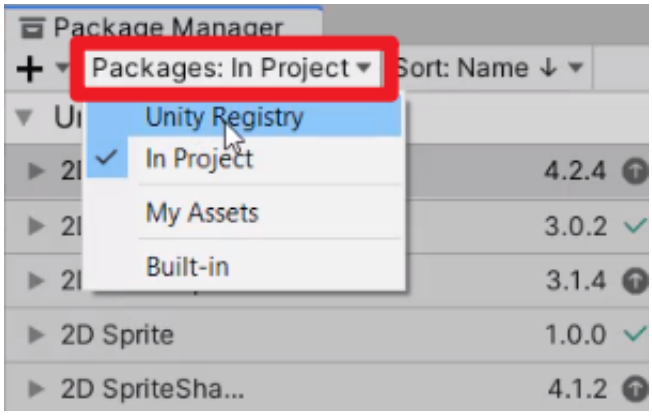
Device Simulator is a package you can install as an alternative to the default Game view. It allows us to get a preview of how an app will look at various different aspect ratios and resolutions.

Installing Device Simulator

To install Device Simulator, we can go to **Window > Package Manager**:

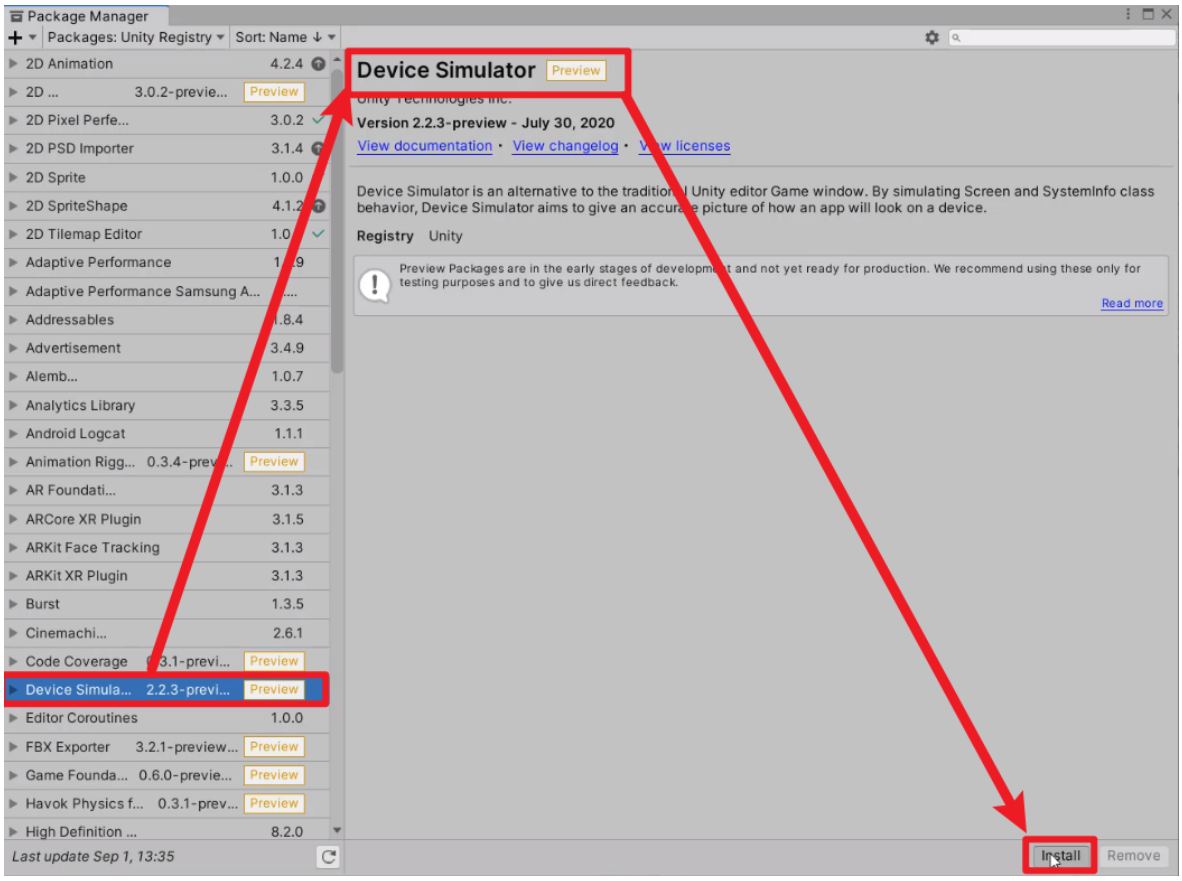


And inside of the **Package Manager**, we can click on **Packages: In Project** and select '**Unity Registry'**.



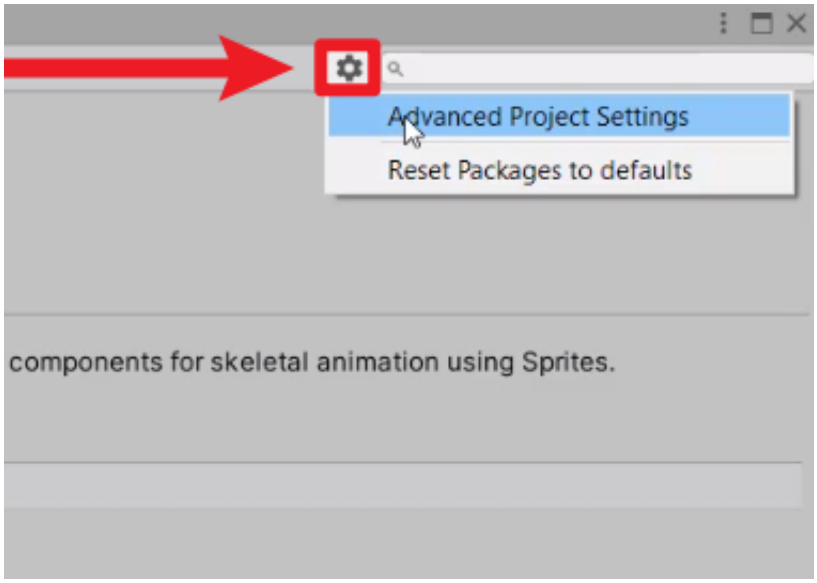
Then select **Device Simulator** and click on **Install**.

© 2020 Zenva Pty Ltd. All rights reserved.



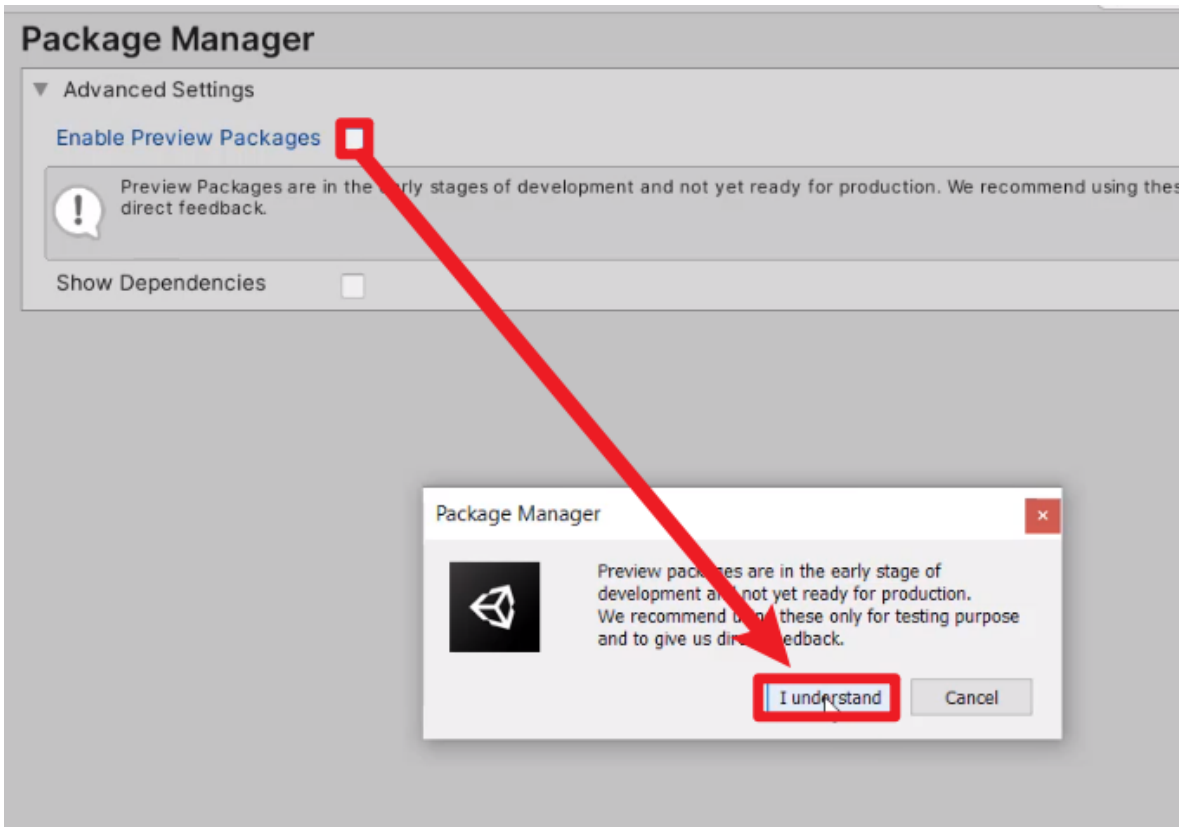
If the Device Manager doesn't appear in the list, it may be hidden as a **Preview Package**.

To enable viewing preview packages, click on the **Gear** icon at the top right, and select **Advanced Project Settings**.



Click on **Enable Preview Packages**, and click on 'I understand'.

© 2020 Zenva Pty Ltd. All rights reserved.

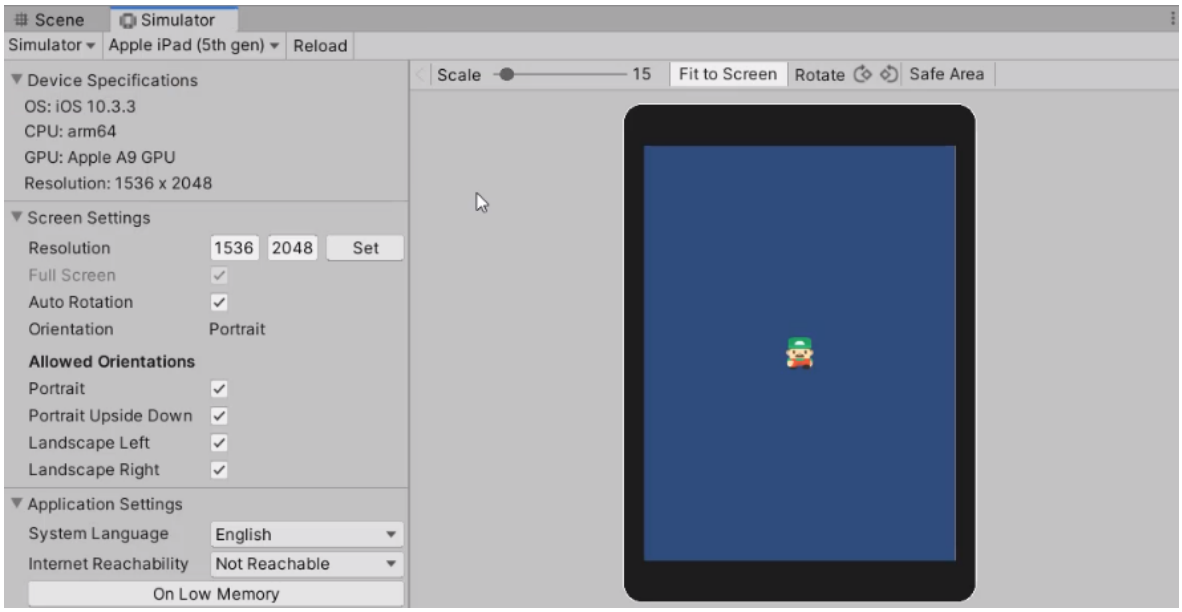


Configuring Device Simulator

After installing, you will be able to click on the dropdown that says **Game**, and then select **Simulator**.

The screen will then switch over to the **Simulator View**, where we can change the device that we want to look through.

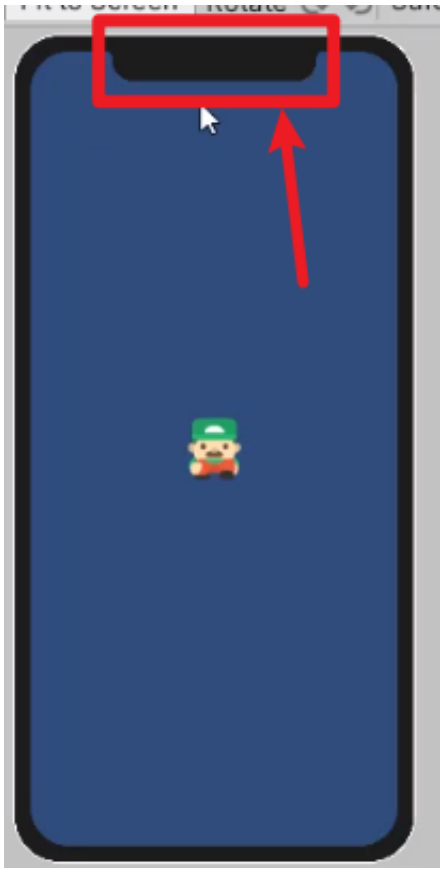
© 2020 Zenva Pty Ltd. All rights reserved.



When you click on the default device– **Apple iPad (5th Gen)**, you will see there's a large list of various different mobile devices.

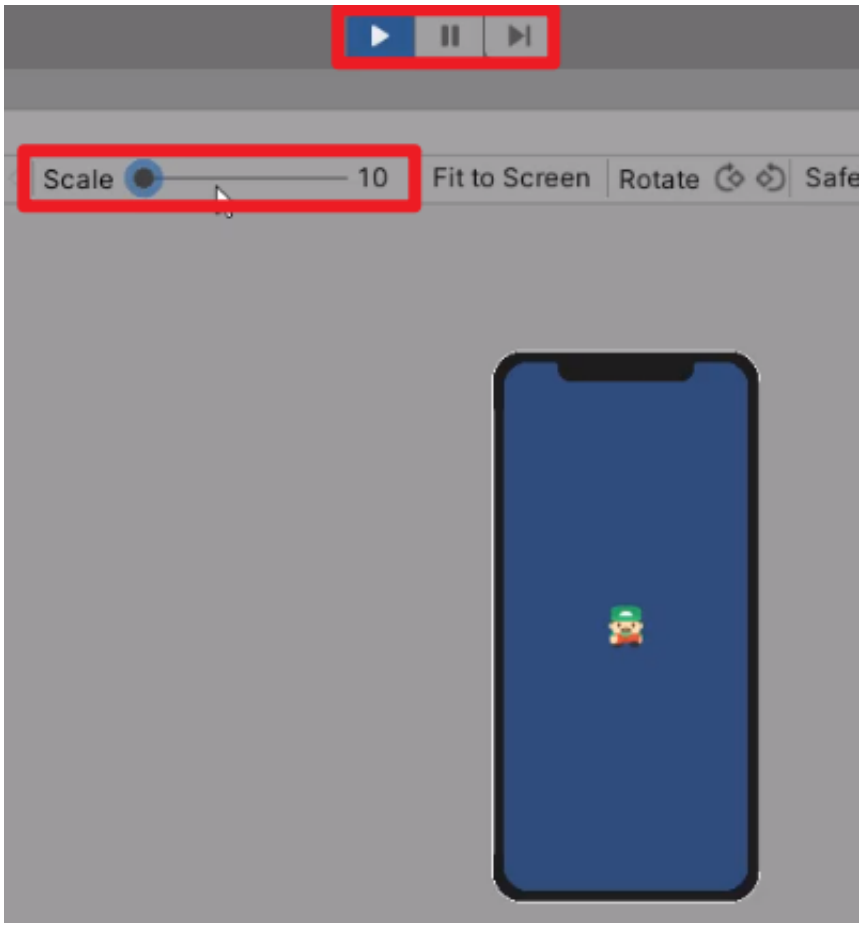
This is a great way of checking how our **UIs** will look on a mobile device, without having to actually build it and testing it out. Note that some UI elements may be cut off by any of the bevels or notches.

© 2020 Zenva Pty Ltd. All rights reserved.

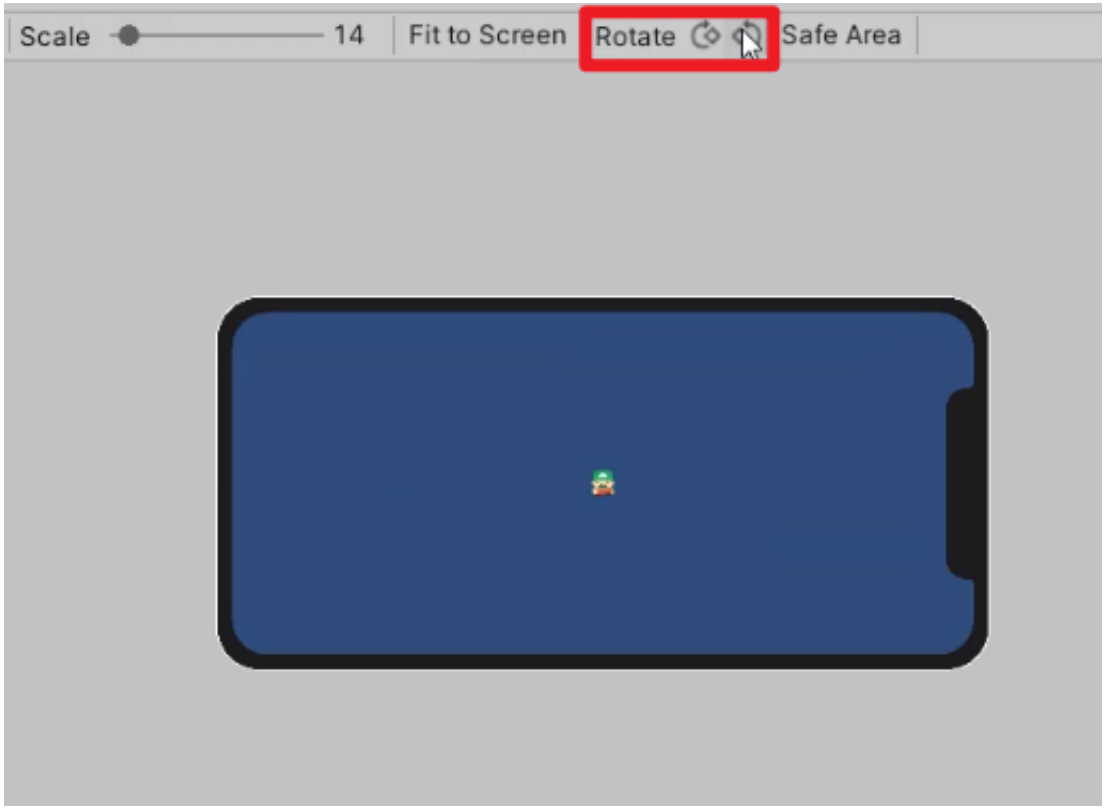


To test the app, just press the **Play** button and play it in the view, just like our normal game view. You can also **zoom in/out** using the **Scale** bar.

© 2020 Zenva Pty Ltd. All rights reserved.

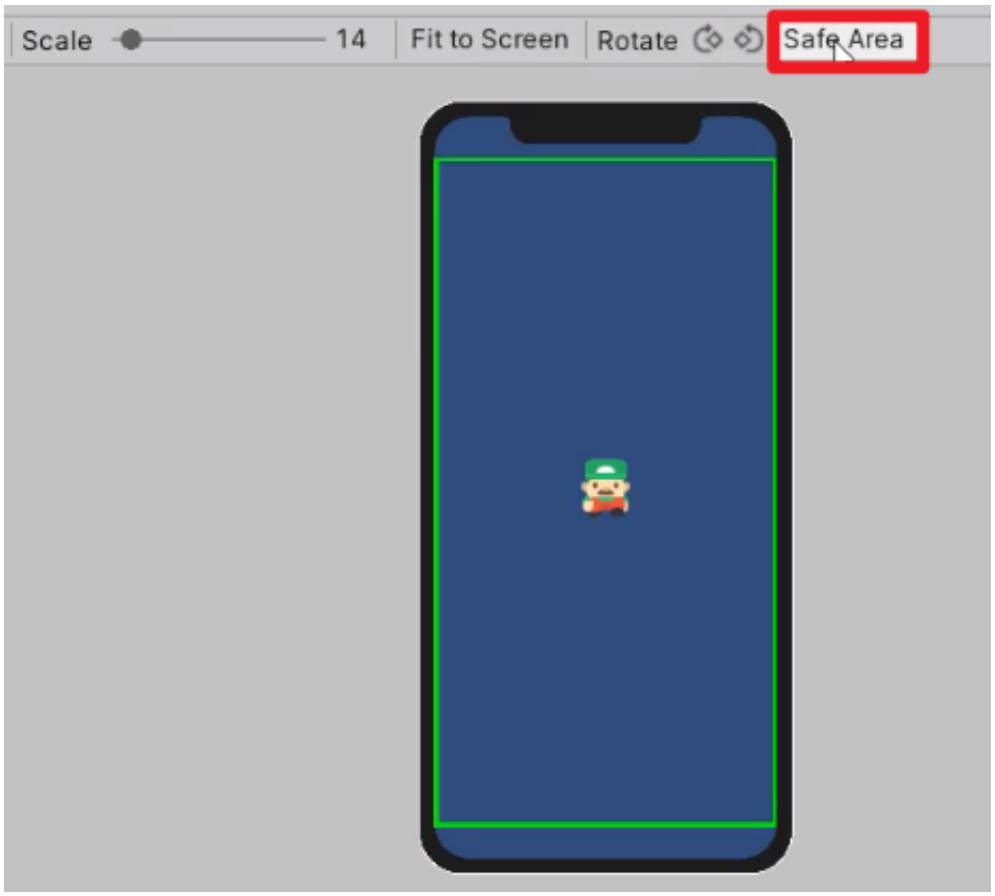


You can also **rotate** the screen by using these buttons.



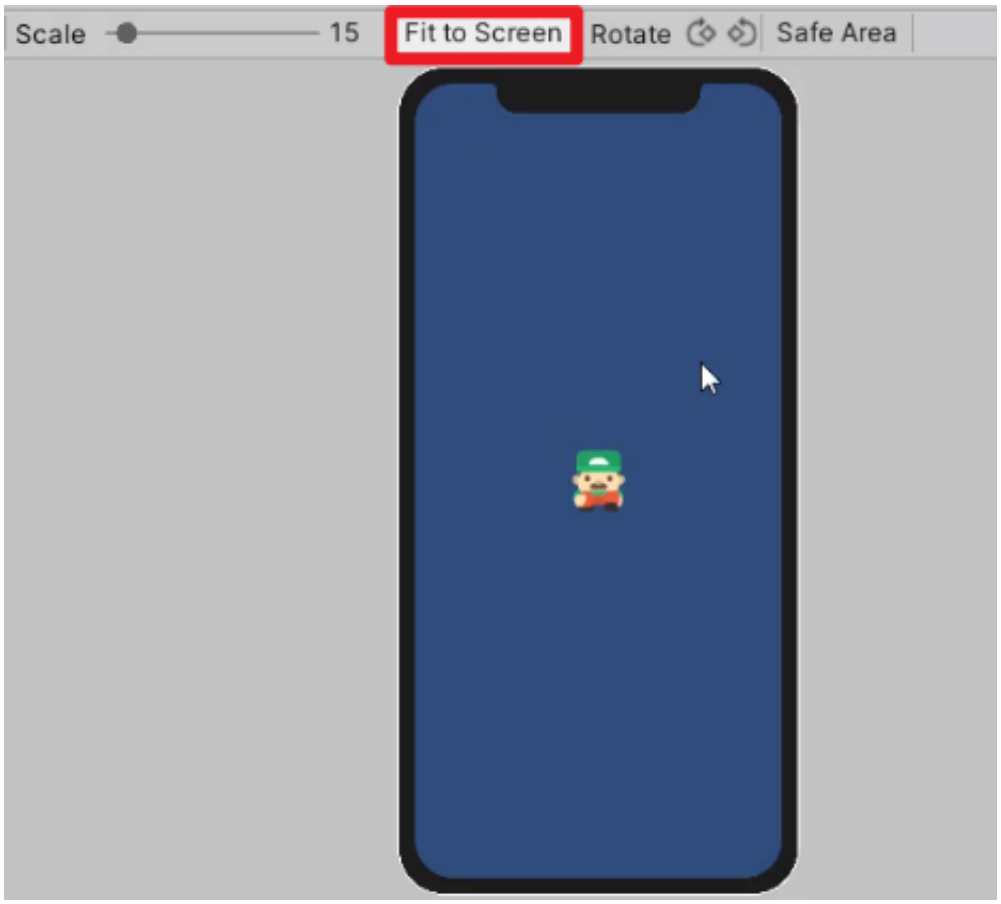
© 2020 Zenva Pty Ltd. All rights reserved.

You can click **Safe Area** to toggle where it is recommended that you build your UI.

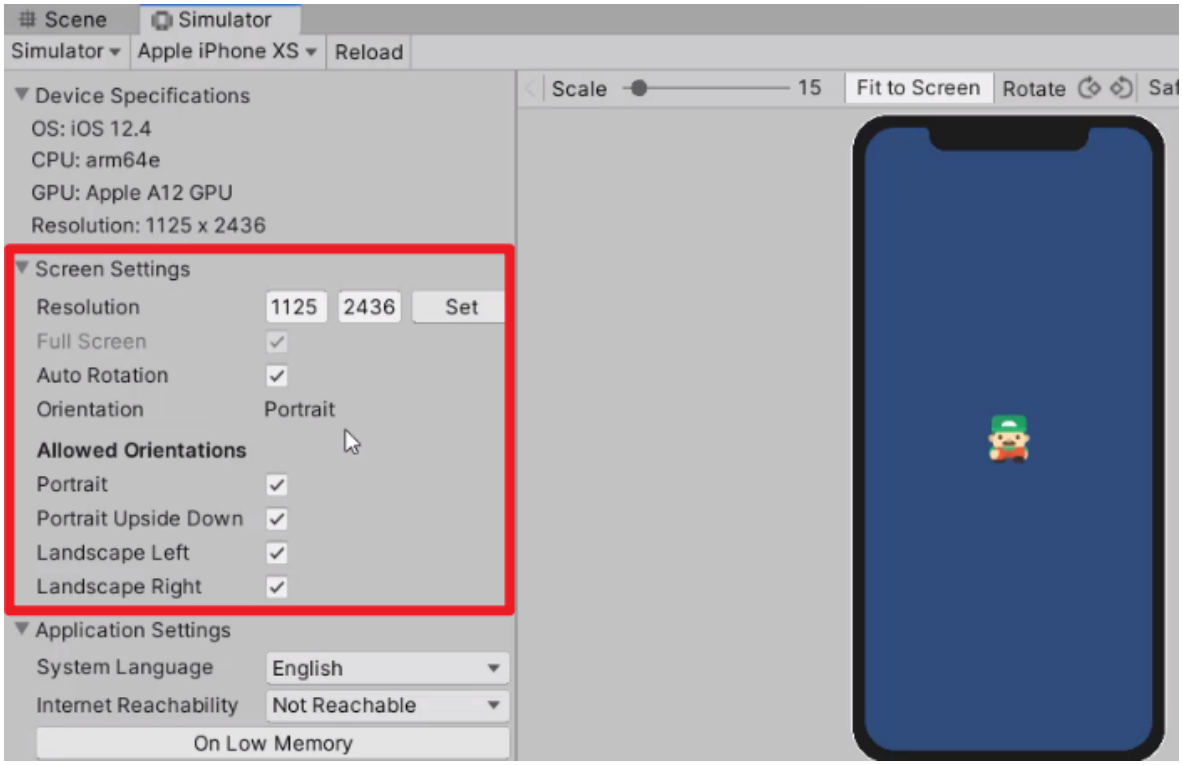


Fit to Screen resets the scale and as large as it can with the phone on the screen.

© 2020 Zenva Pty Ltd. All rights reserved.



We can also change the **allowed Orientations**, **Resolution**, and **Auto-rotation**.

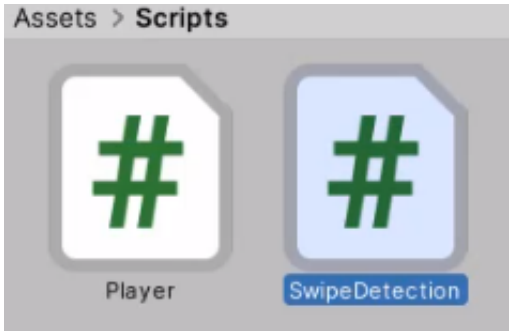


© 2020 Zenva Pty Ltd. All rights reserved.

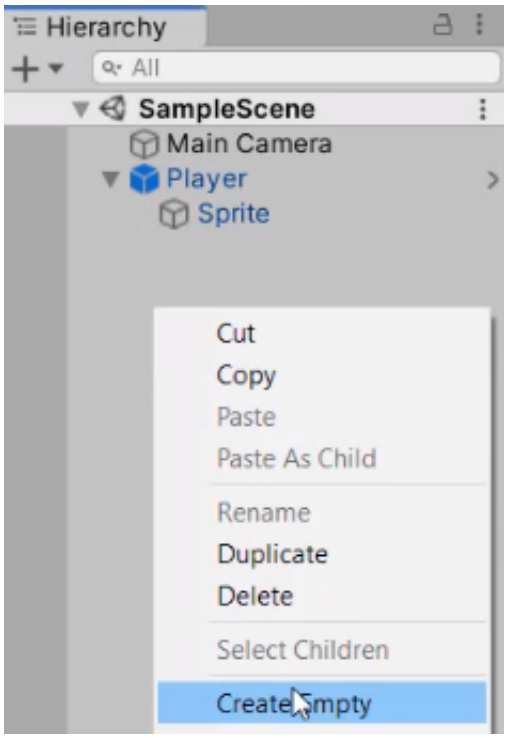
In this lesson, we're going to begin setting up the ability for our game to **detect touches** for moving the player.

Creating A GameManager

Let's create a new C# script called **SwipeDetection** in the Scripts folder.

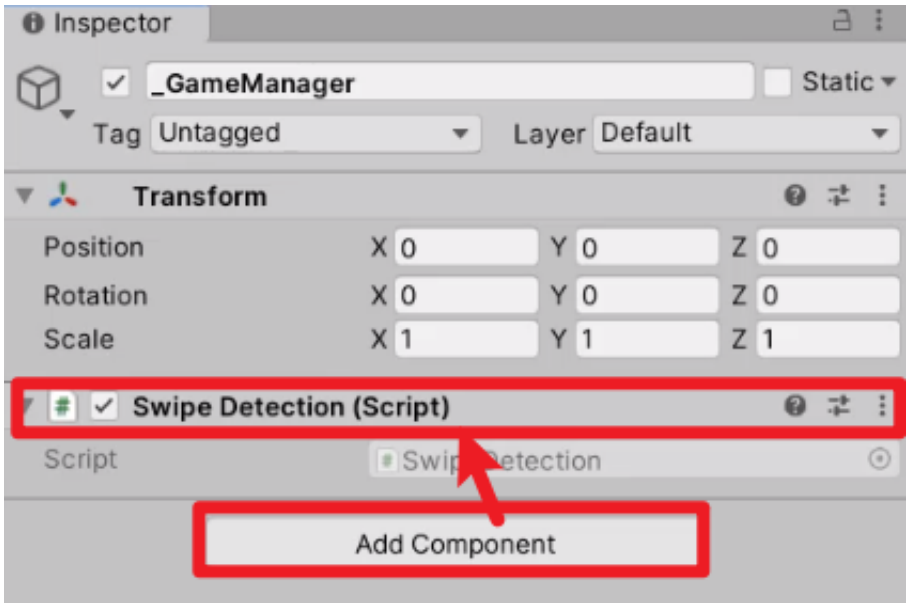


We're going to create a new empty gameObject, and rename it to **_GameManager**.



We will then attach the **SwipeDetection** script to **_GameManager**.

© 2020 Zenva Pty Ltd. All rights reserved.



Adding Swipe Detection Functionality

Inside of the **SwipeDetection** script, we're going to check if we have **swiped** the finger left/right/up, and **move towards** the direction of the swipe.

First of all, we need to create a variable to store our **Player** so that we can, later on, tell it to move:

```
public Player player;
```

Next, we need to create a **Vector2** that tells us the first touchpoint of your finger on the screen, and we'll check to see if it's left, up, or right.

```
public Player player;
private Vector2 startPos;
```

Then we need to create another variable to set **how many pixels** do we have to swipe in order for it to be detected. We're going to set this a **public integer** with a default value of 20.

```
public Player player;
private Vector2 startPos;
public int pixelDistToDetect = 20;
```

And lastly, we need to create a **boolean** to check if we have a finger down.

```
public Player player;
private Vector2 startPos;
public int pixelDistToDetect = 20;
private bool fingerDown;
```

© 2020 Zenva Pty Ltd. All rights reserved.

Using these variables, we're going to check to see if we have pressed a finger down in the **Update** function.

```
void Update()
{
    if(fingerDown == false && Input.touchCount > 0 && Input.touches[0].phase == Touch
Phase.Began)
}
```

To detect if a finger is down for the first time correctly, we put three different conditions:

- Is the **fingerDown** currently set to false?
- Is **Input.touchCount*** greater than 0? (*Input.touchCount will increase as more touches are **detected** on the screen at a time.)
- Has the first touch (**touches[0]**) just Began? (= Is this the first frame the finger is down?)

```
void Update()
{
    if(fingerDown == false && Input.touchCount > 0 && Input.touches[0].phase == Touch
Phase.Began)
    {
        // If so, we're going to set the startPos to the first touch's position,
        startPos = Input.touches[0].position;
        // ... and set fingerDown to true to start checking the direction of the swip
e.
        fingerDown = true;
    }
}
```

If the finger is down, then we can start checking which **direction** the finger moves into. For example, if the finger ends up moving away from the start position along the **y-axis**, it means it just swiped **upwards**.

```
//Is our finger touching the screen?
if (fingerDown)
{
    //Did we swipe up?
    if(Input.touches[0].position.y >= startPos.y + pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe up");
    }
}
```

Now for testing purposes, we're going to **duplicate** this code and use it in terms of the mouse. So instead of **Input.touches[]**, we're going to use **Input.GetMouseButtonDown** to detect mouse clicks.

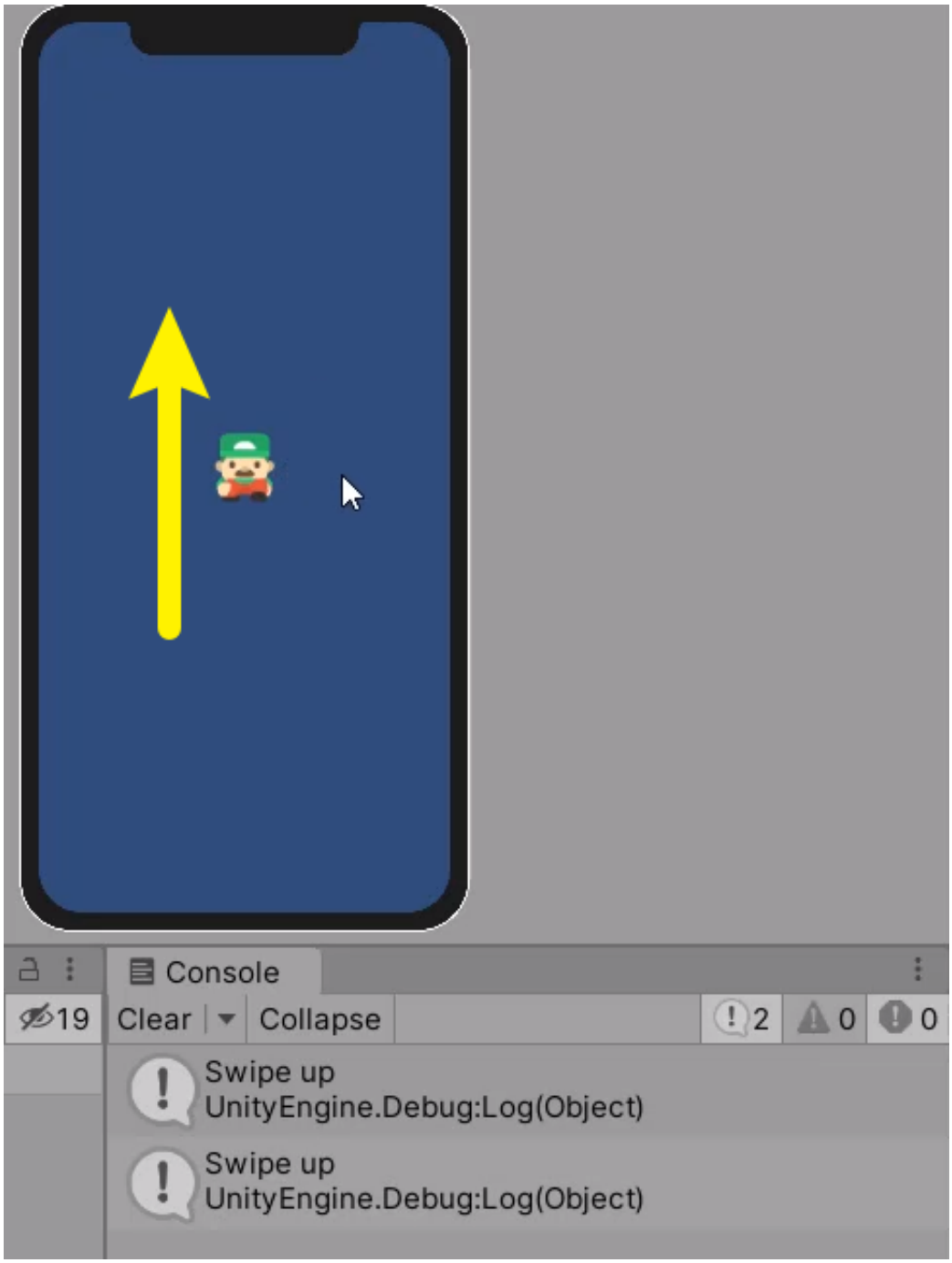
```
//Testing for PC
```

© 2020 Zenva Pty Ltd. All rights reserved.

```
if (fingerDown && Input.GetMouseButtonDown(0))
{
    startPos = Input.mousePosition;
    fingerDown = true;
}
if (fingerDown)
{
    //Is the current mouse position 'pixelDistToDetect' away from 'startPos' on y-
axis?
    if(Input.mousePosition.y >= startPos.y + pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe up");
    }
}
```

Let’s save it and press **Play**.

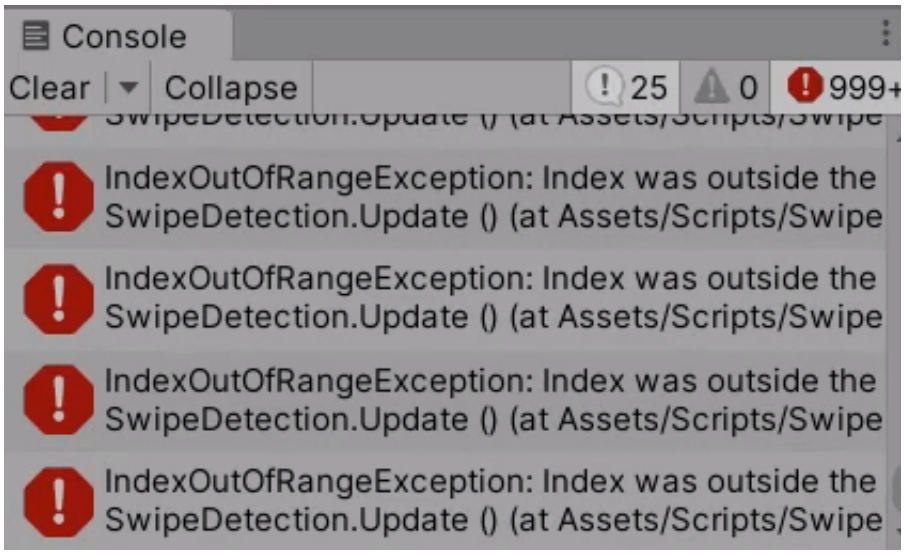
© 2020 Zenva Pty Ltd. All rights reserved.



You can see that the Debug.Log works as intended as you **click and drag** upwards.

However, if your mouse moves off the screen, the script will not be able to calculate the next touch position correctly and return these errors.

© 2020 Zenva Pty Ltd. All rights reserved.



To prevent this, we need to **reset** the **startPos** by setting **fingerDown** to **false** whenever our mouse button is up.

```
//If the mouse button is up,
if(fingerDown && Input.GetMouseButtonUp(0))
{
    //startPos will be reset
    fingerDown = false;
}
```

We can also add the same functionality for touch inputs. Note that we need to check if the first finger is released from the screen, i.e. if the **TouchPhase** is set as '**Ended**'.

```
//If the finger is released from the screen,
if(fingerDown && Input.touchCount > 0 && Input.touches[0].phase == TouchPhase.Ended)
{
    //startPos will be reset
    fingerDown = false;
}
```

© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be adding in left and right swipe detections.

Inside of **SwipeDetection.cs**, we're going to add **else if** statements below where we detect if our finger swiped upwards:

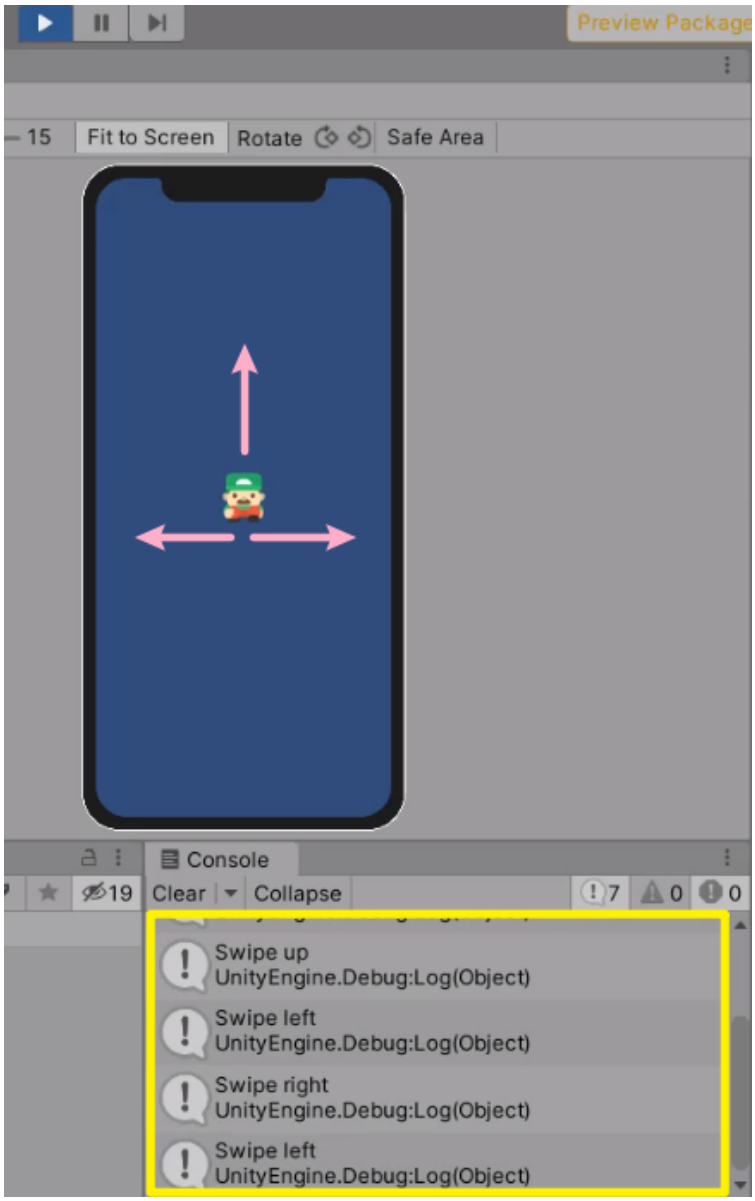
```
//Is our finger touching the screen?
if (fingerDown)
{
    //Did we swipe up?
    if(Input.touches[0].position.y >= startPos.y + pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe up");
    }
}
```

So to detect left and right swipes, we can detect the change in **x-position** of **Input.touches**:

```
//Is our finger touching the screen?
if (fingerDown)
{
    //Did we swipe up?
    if(Input.touches[0].position.y >= startPos.y + pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe up");
    }
    //Did we swipe left?
    else if(Input.touches[0].position.x <= startPos.x - pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe left");
    }
    //Did we swipe right?
    else if(Input.touches[0].position.x >= startPos.x + pixelDistToDetect)
    {
         fingerDown = false;
         Debug.Log("Swipe right");
    }

}
```

If you save and hit **Play**, you'll see that **Debug.Log** works as intended:

© 2020 Zenva Pty Ltd. All rights reserved.



Moving Player

We're going to replace **Debug.Log** and start actually moving our player. To do that, let's first open up **Player.cs** and declare these variables.

```
public class Player : MonoBehaviour
{
     //Stores how long it takes to get from one tile to the other
     public float moveSpeed;
     //Where is this player trying to get to
     private Vector3 targetPos;
}
```

Inside the **Start** function, we're going to set the **target position** to be the current position, i.e. **transform.position**:

© 2020 Zenva Pty Ltd. All rights reserved.

```
public class Player : MonoBehaviour
{
     //Stores how long it takes to get from one tile to the other
     public float moveSpeed;
     //Where is this player trying to get to
     private Vector3 targetPos;

     void Start ()
     {
          targetPos = transform.position;
     }
}
```

Next, we're going to create a new **public void** function called **Move**, which is going to carry over a Vector3 (**move direction**) as a parameter.

Then we can update our target position by this given **move direction**.

```
public void Move (Vector3 moveDirection)
{
     targetPos += moveDirection;
}
```

Now, inside the **Update** function, we need to actually move our player **towards** the target position using **Vector3.MoveTowards**.

```
private void Update ()
{
     //Move towards the target
     transform.position = Vector3.MoveTowards(transform.position, targetPos, moveSpee
d * Time.deltaTime);
}
```

Note that **moveSpeed** is multiplied by **Time.deltaTime** to convert the speed unit from per frame to **per second**. Let's save the Player script and go back to **SwipeDetection.cs.**

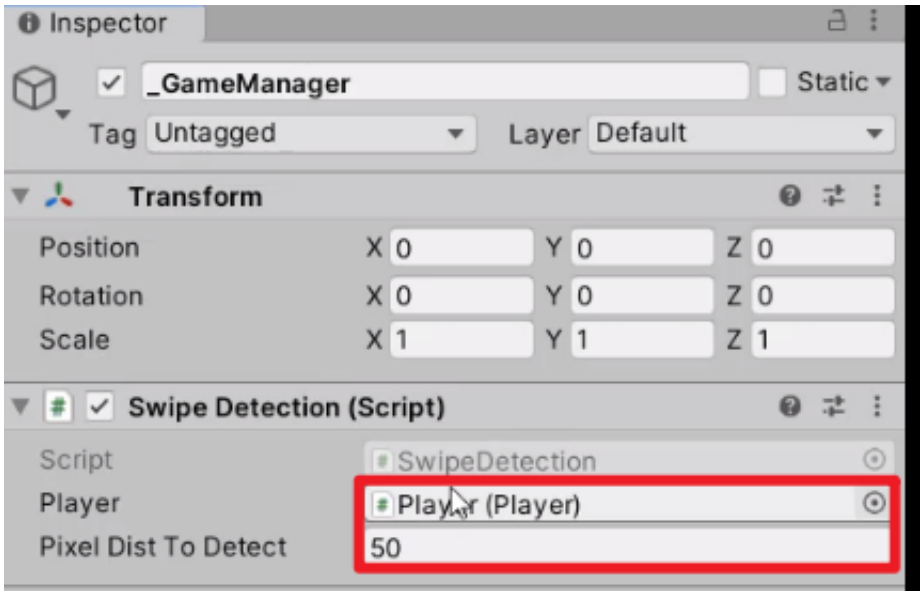
We're now going to replace all the **Debug.Log** with the player script's **Move** function, passing in a direction to move towards.

```
//Is our finger touching the screen?
if (fingerDown)
{
    //Did we swipe up?
    if(Input.touches[0].position.y >= startPos.y + pixelDistToDetect)
    {
         fingerDown = false;
         //Move upwards
         player.Move(Vector3.up);
    }
```

© 2020 Zenva Pty Ltd. All rights reserved.

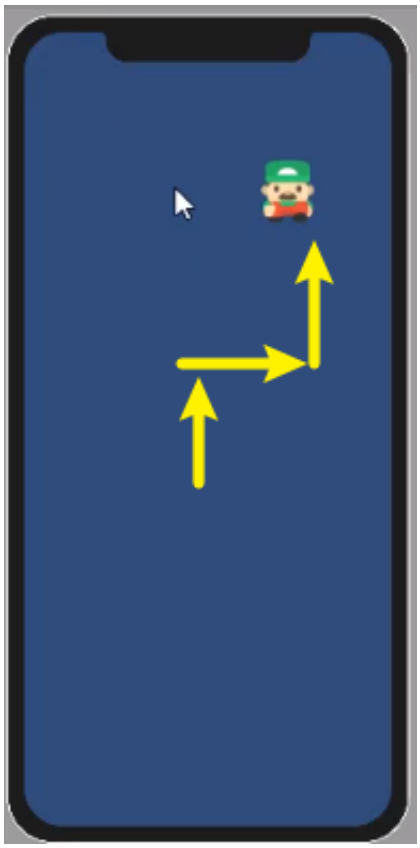
```
    //Did we swipe left?
    else if(Input.touches[0].position.x <= startPos.x - pixelDistToDetect)
    {
         fingerDown = false;
         //Move left
         player.Move(Vector3.left);
    }
    //Did we swipe right?
    else if(Input.touches[0].position.x >= startPos.x + pixelDistToDetect)
    {
         fingerDown = false;
         //Move right
         player.Move(Vector3.right);
    }

}
```

Lastly, we're going to assign a value for **PixelDistToDetect**, and link our **Player** gameObject into the player property.



If we hit Play, we can now swipe up, left, right to move our player.

© 2020 Zenva Pty Ltd. All rights reserved.

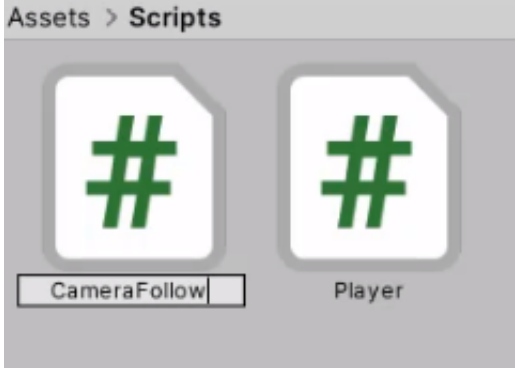


© 2020 Zenva Pty Ltd. All rights reserved.

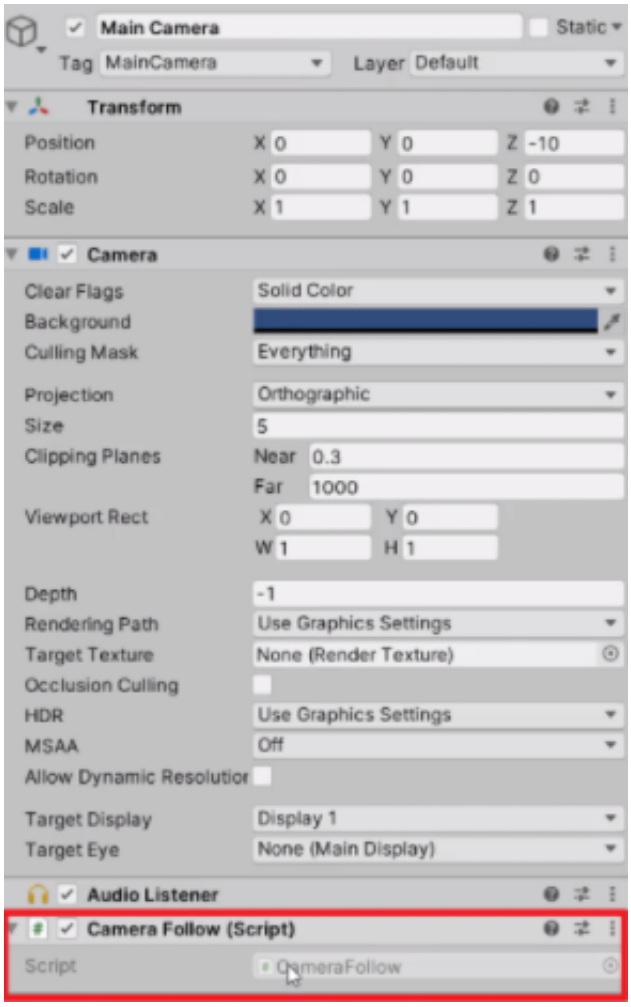
In this lesson, we're going to make our camera follow the player, preventing them from going off the screen.

Creating A CameraFollow Script

First of all, we're going to create a new script called "CameraFollow" inside **Assets > Scripts**.



We're going to **attach** this script to our **Main Camera**:



We can then **double-click** the script and open it up in Visual Studio. Here, we're going to create two new variables:

© 2020 Zenva Pty Ltd. All rights reserved.

```
public Transform target;
public Vector3 offset;
```

The **target** will be used to reference a Transform component to follow, and the **offset** will be used to maintain a certain distance away from the player.

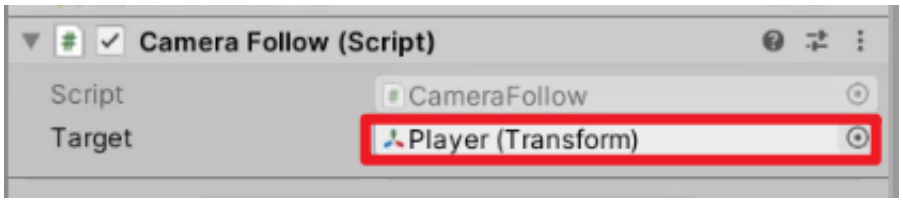
Inside the **Update** function, we're going to update our camera's position so that it follows the target's position, but with some **offset** (-10) so that our camera doesn't sit inside the player.

```
//Update is called once per frame.
void Update()
{
   transform.position = new Vector3(0, target.position.y, -10) + offset;
}
```

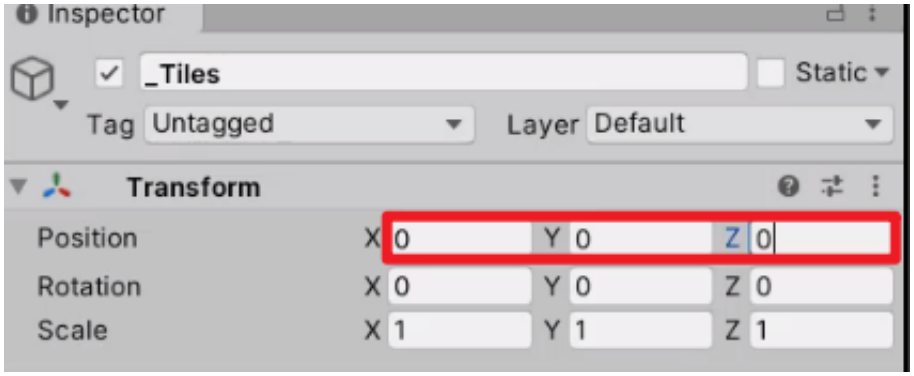
Let's hit save and return to our editor.

Setting Up Camera

Now we're going to drag in the **Player** into the **Target** slot:

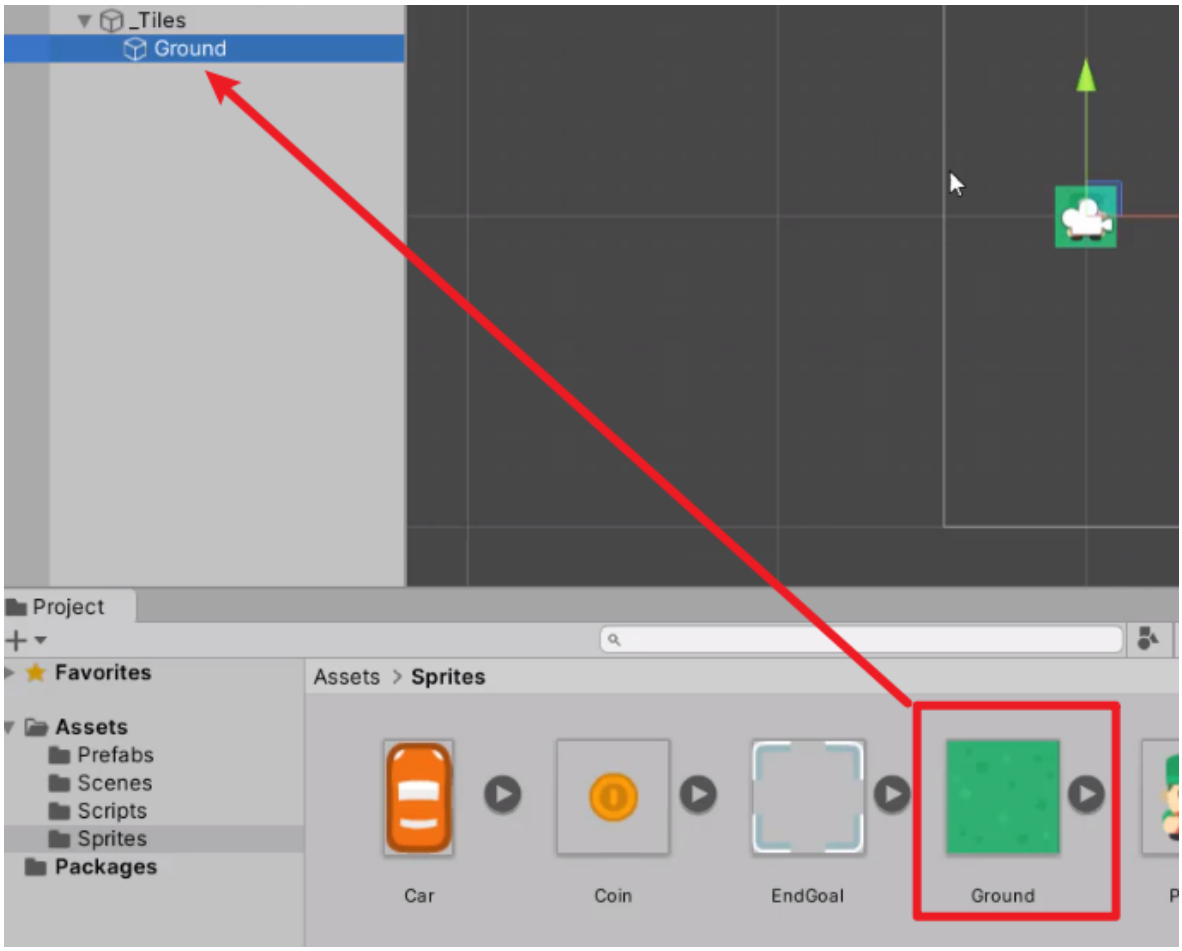


We're also going to create a new object called **_Tiles** and reset its **position**.



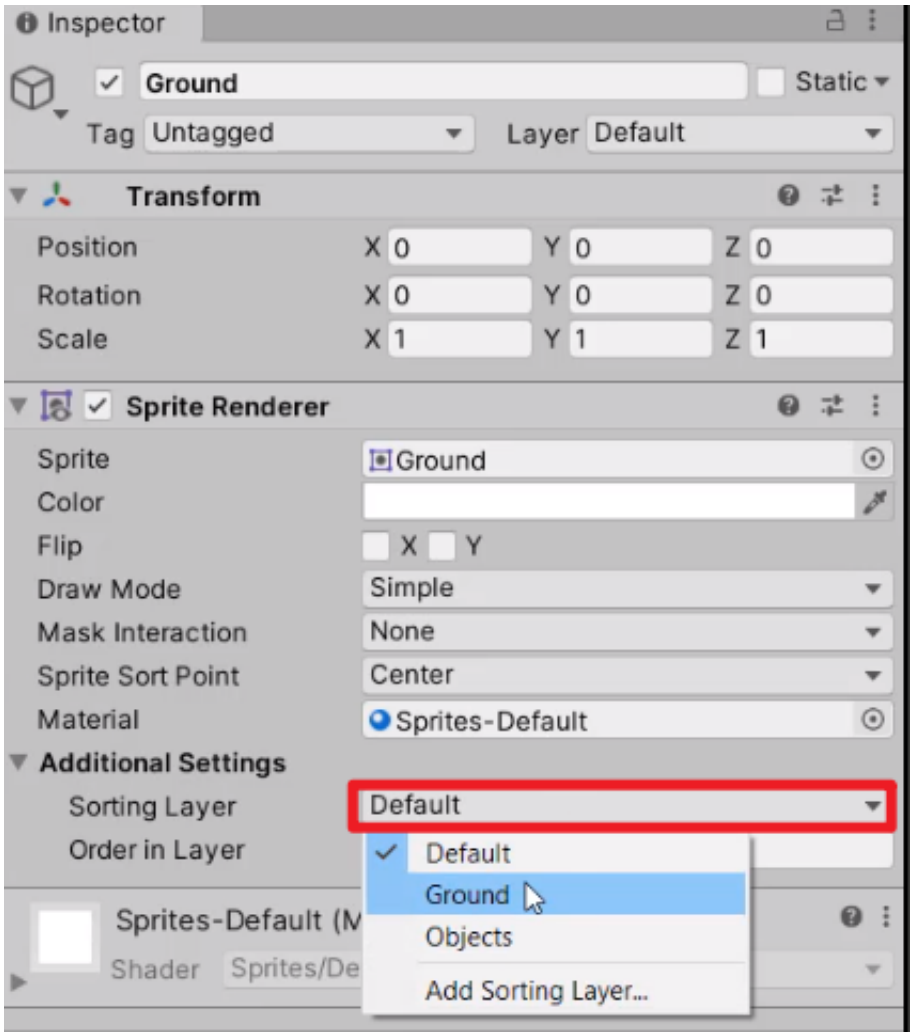
Then, we're going to drag in a sprite called **Ground** as a child of **_Tiles**.

© 2020 Zenva Pty Ltd. All rights reserved.



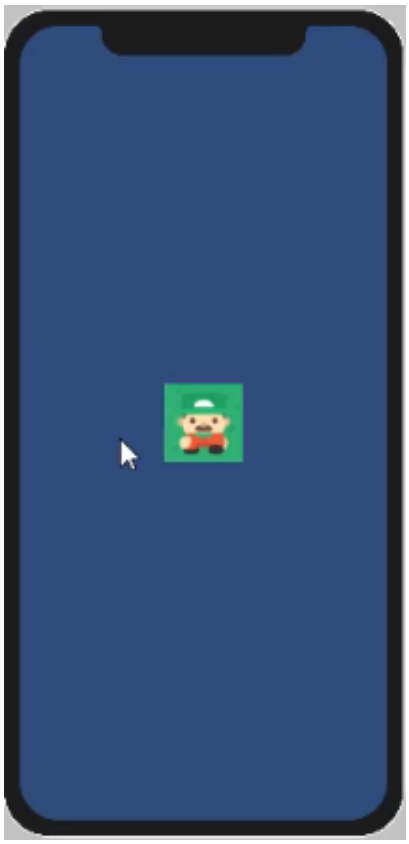
Make sure to set the **Sorting layer** of the sprite to be “Ground”.

© 2020 Zenva Pty Ltd. All rights reserved.

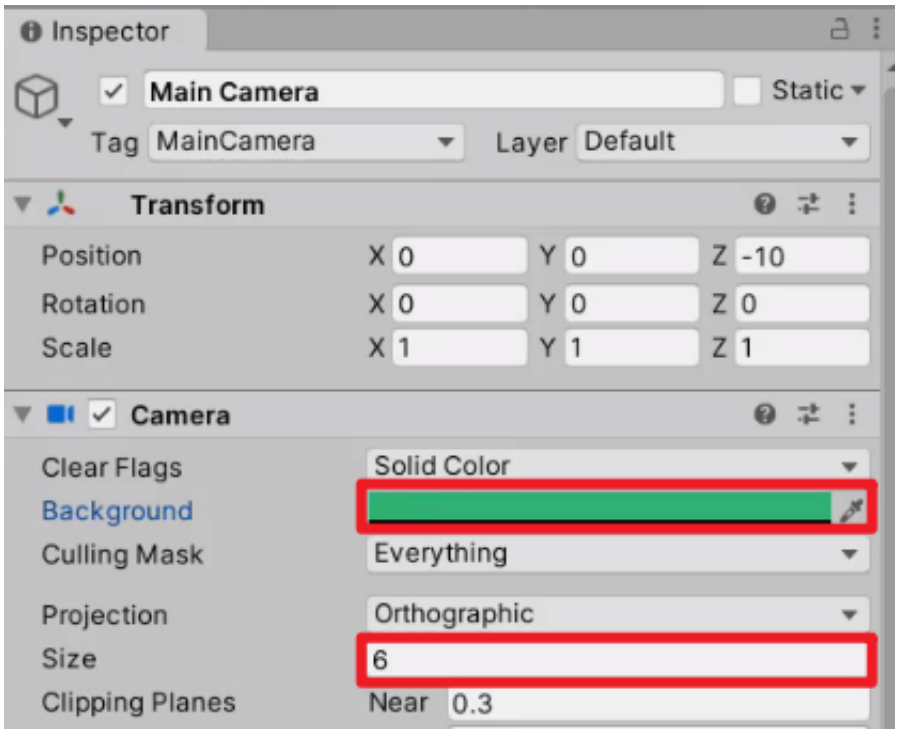


By doing so, the tile should appear behind our player.

© 2020 Zenva Pty Ltd. All rights reserved.

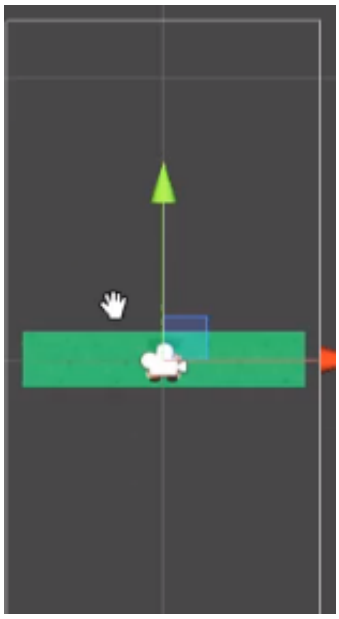


We're going to set the **Background Colour** to be green and the **Camera Size** to be 6 for a better view:

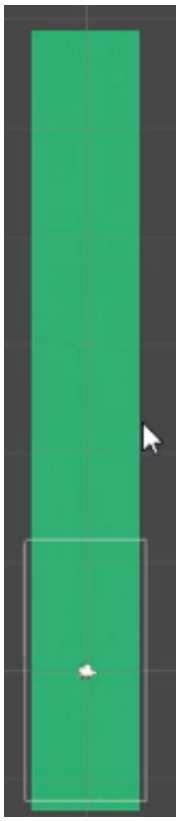


Now we can **duplicate** (**Ctrl + D** or **Command +D**) the tile as much as we want.

© 2020 Zenva Pty Ltd. All rights reserved.



We're going to set a **vertical** strip of these tiles so our player can move up and down:



Clamping Position

We also need to **clamp** the player's position so they can't go further than the bounds of the screen.

© 2020 Zenva Pty Ltd. All rights reserved.

To do that, we're going to open up **Player.cs** and create a new variable:

```
public int edgeDistance;
```

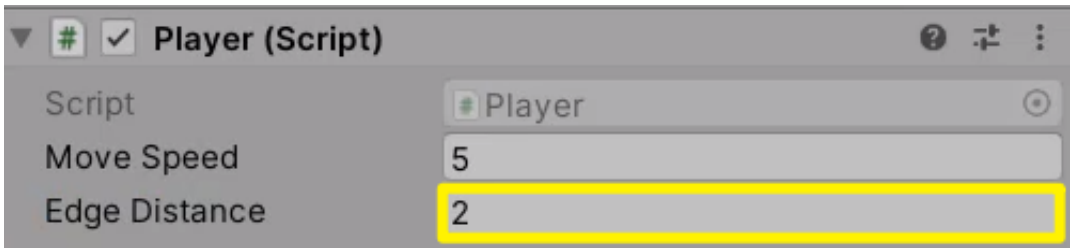
Then in the **Move** function, we need to check if our target position is already at the edge.

```
public void Move (Vector3 moveDirection)
{
     //If the target position is already at the edge,
     if(Mathf.Abs(targetPos.x) == edgeDistance)
     {
         //Then the function will stop here.
         return;
     }
     targetPos += moveDirection;
}
```

Note that **targetPos.x** will become negative when we swipe left, so we used **Mathf.Abs** to convert it into a full positive number.

© 2020 Zenva Pty Ltd. All rights reserved.

Let's save the script, return to the Editor and set the value for **edgeDistance** to 2.



If we hit Play now, you'll see that when we get to the edge, you **can't** move either direction anymore.

This is because our current clamping condition is not taking into consideration if we're moving **away** from the edge, or **towards** the edge.

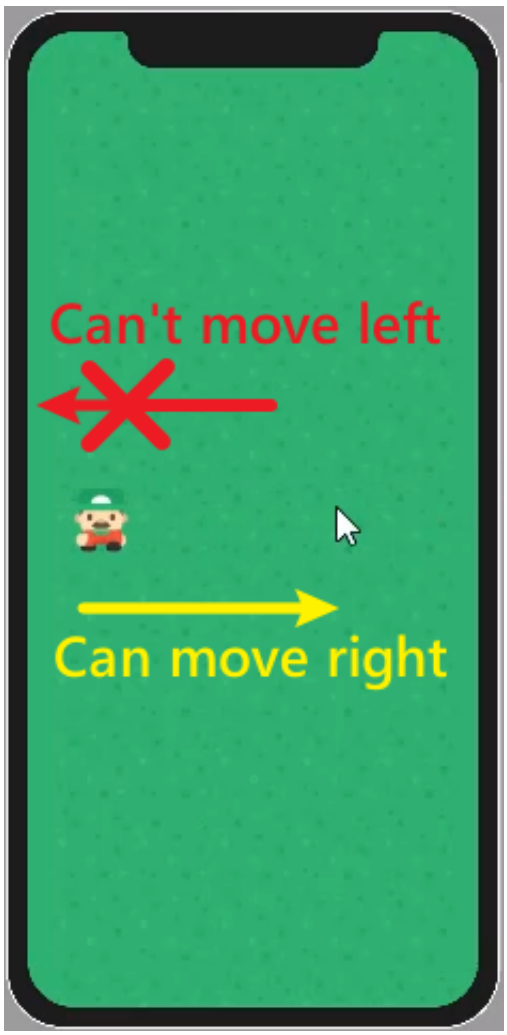
To fix this, we can change it to check if our target position will be **greater** than the edge distance after we move (hence adding **moveDirection.x** to **targetPos.x**).

```
public void Move (Vector3 moveDirection)
{
     //If we're moving away from the edge,
     if(Mathf.Abs(targetPos.x + moveDirection.x) > edgeDistance)
```

© 2020 Zenva Pty Ltd. All rights reserved.

```
        {
            //Then the function will stop here.
            return;
        }
        targetPos += moveDirection;
    }
```

And we should now be able to move **right**, while we can't go further than the **edgeDistance**.

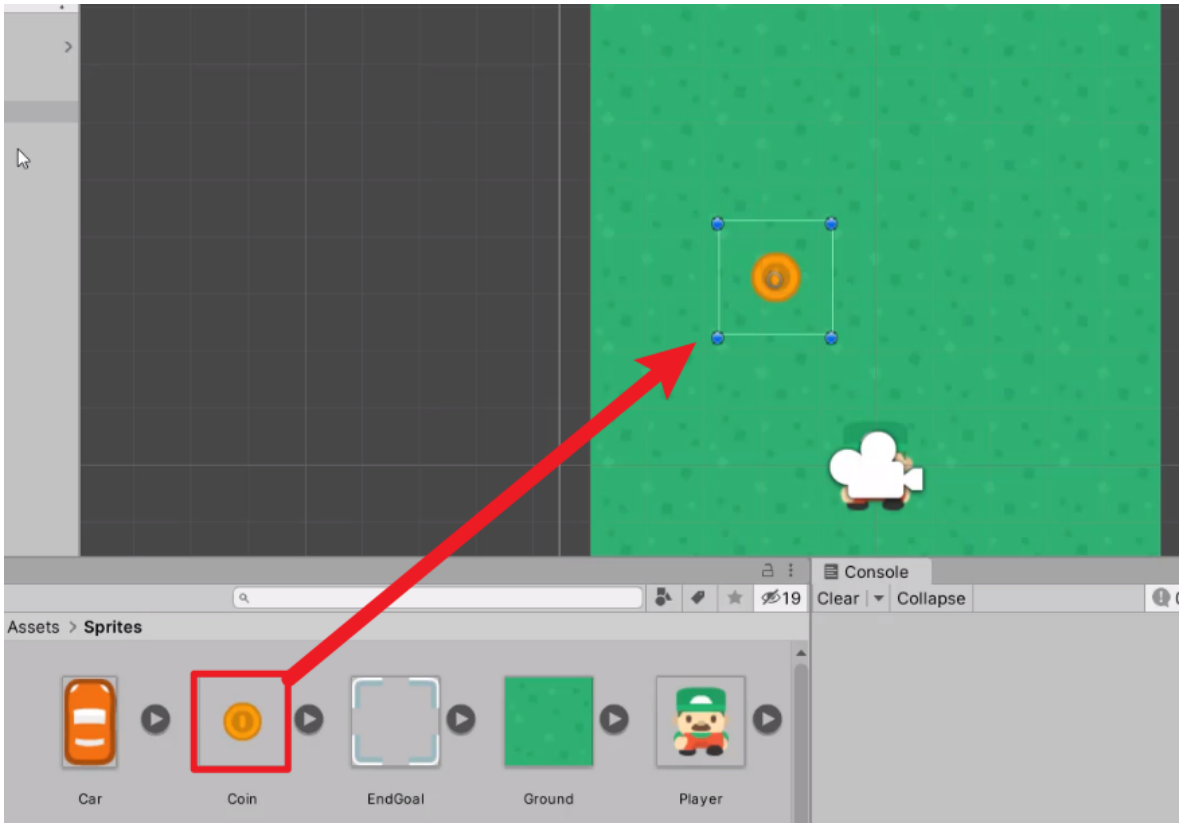


© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be creating a coin that the player can pick up to get a score.

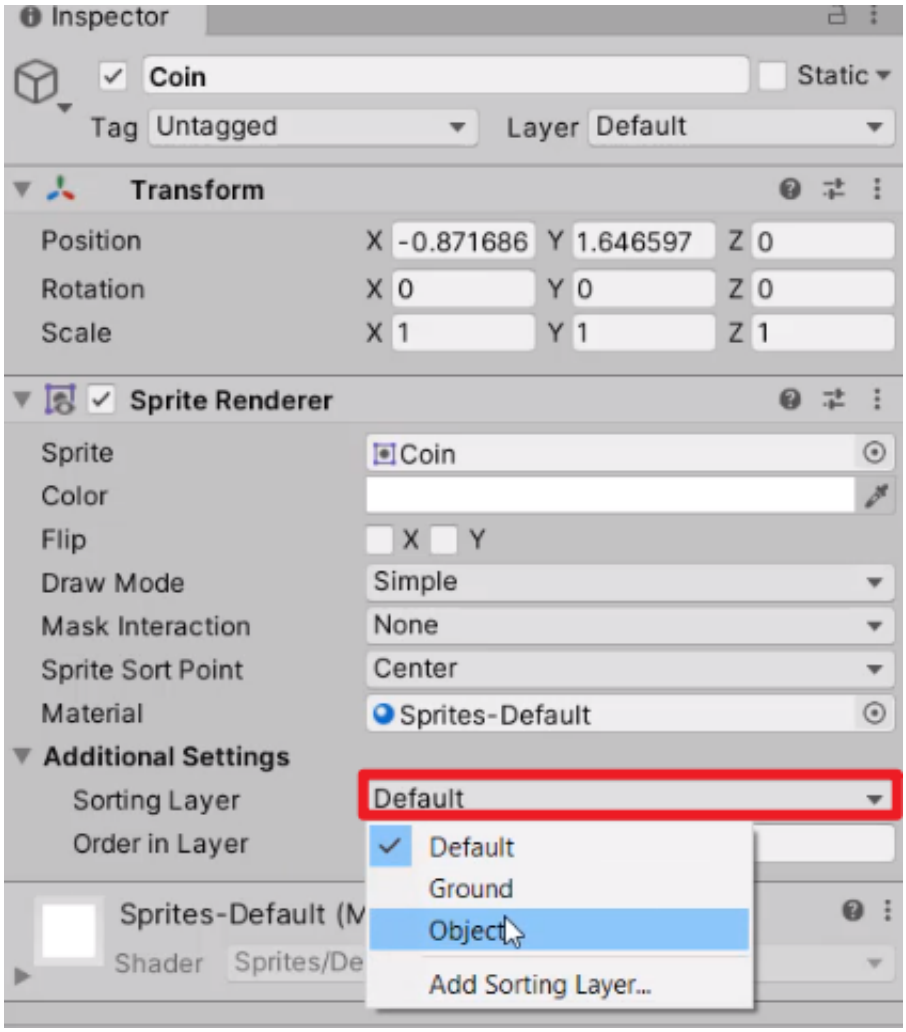
Creating Coin GameObject

Go to **Assets > Sprites** and drag our coin model in as a **child** of the Coin empty gameObject.

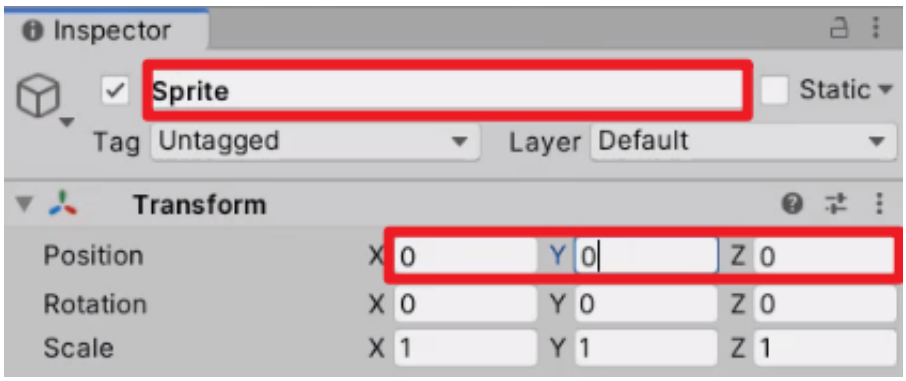


The coin sprite will initially appear **behind** the ground. Let's change the **Sorting layer** from default to '**Objects**', and it should appear on top of Ground objects.

© 2020 Zenva Pty Ltd. All rights reserved.

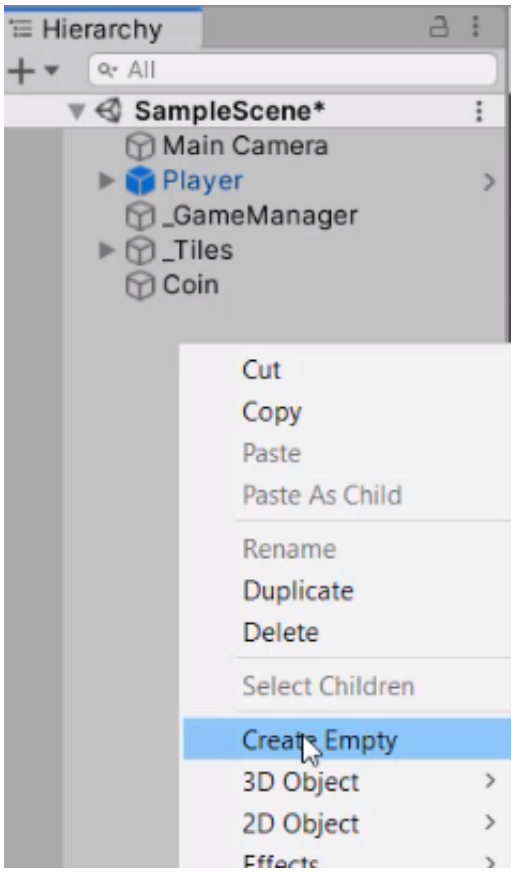


We're going to **reset** the position to 0, 0, 0, and **rename** it to "Sprite".

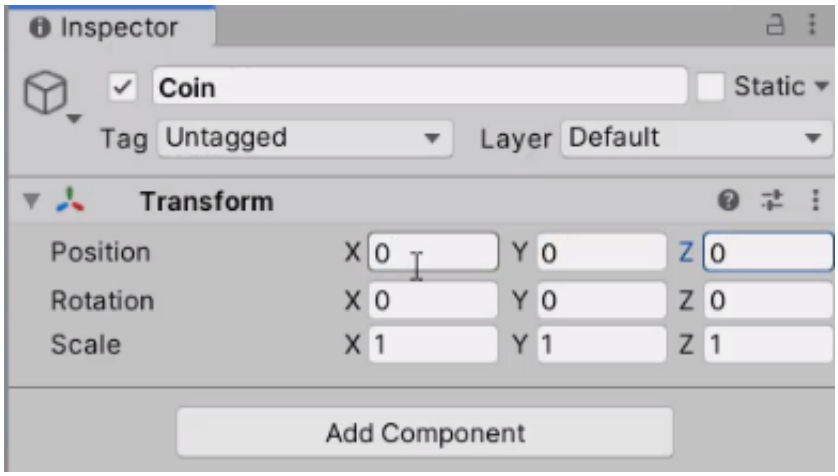


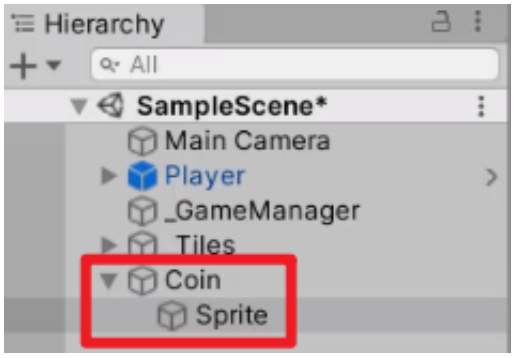
Now we're going to create an **empty** GameObject that will become a parent object of the sprite.

© 2020 Zenva Pty Ltd. All rights reserved.



Let's rename it to "Coin", and reset its **position** to 0, 0, 0.

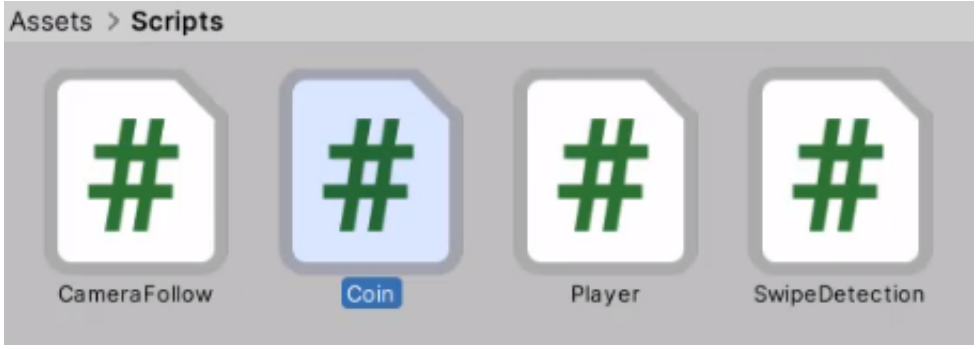




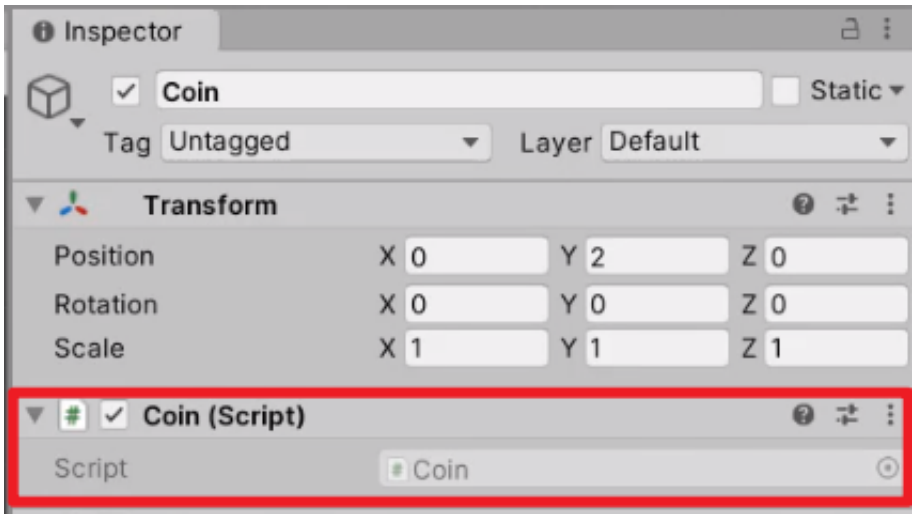
© 2020 Zenva Pty Ltd. All rights reserved.

Creating A Coin Script

We're going to **create a new script** called "Coin" (Right-click > Create New > C# Script):



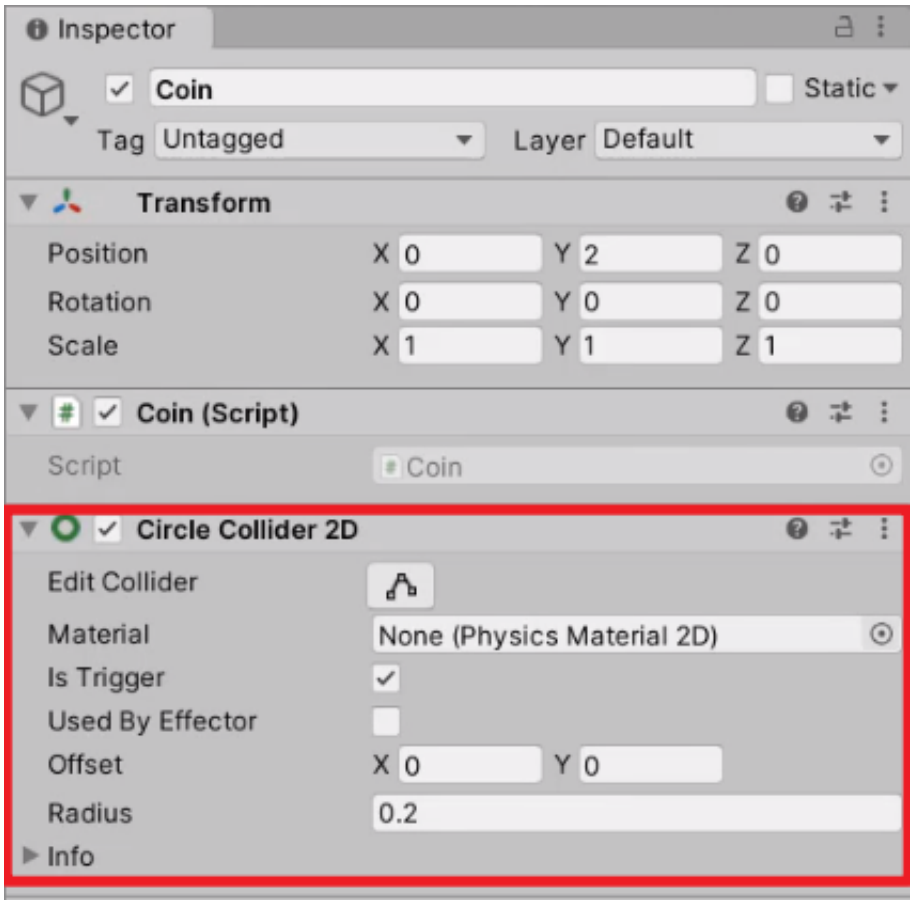
We can then go ahead and **attach** this (drag-and-drop) to the Coin GameObject.



Adding A Collider

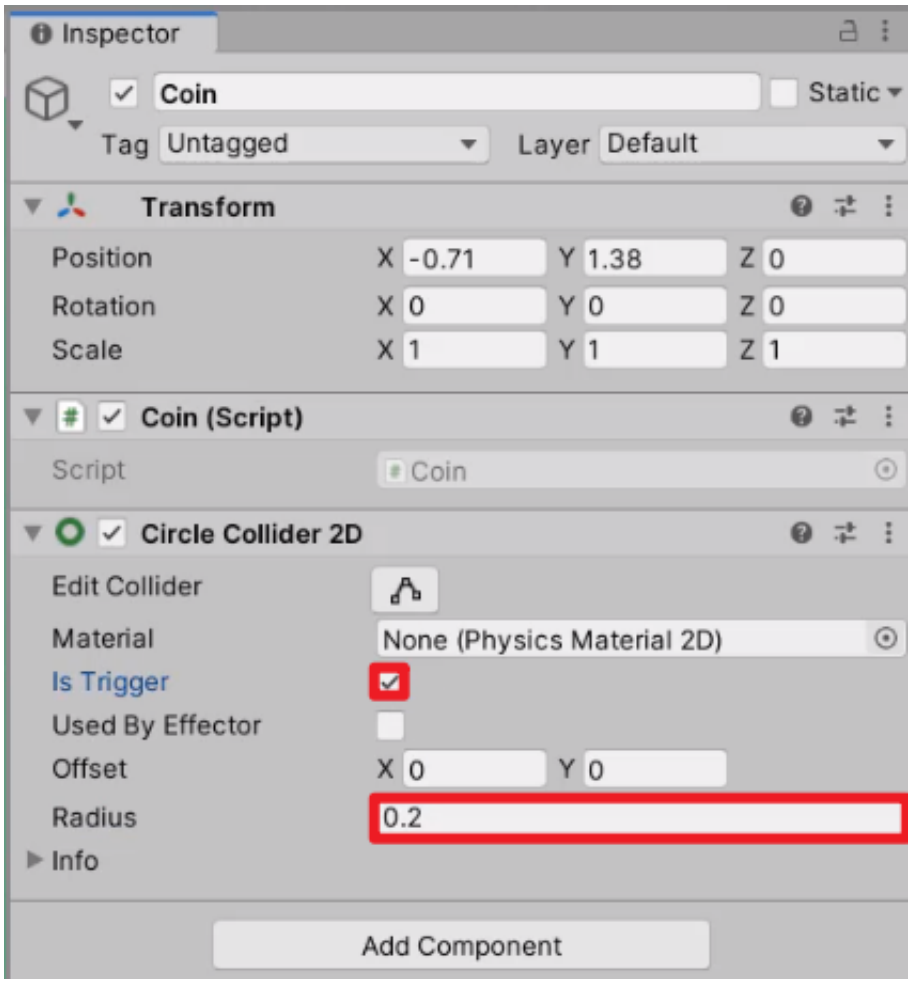
Next, we're going to add in a **collider** to detect trigger collisions when we go into this object. (**Add Component > CircleCollider2D**)

© 2020 Zenva Pty Ltd. All rights reserved.



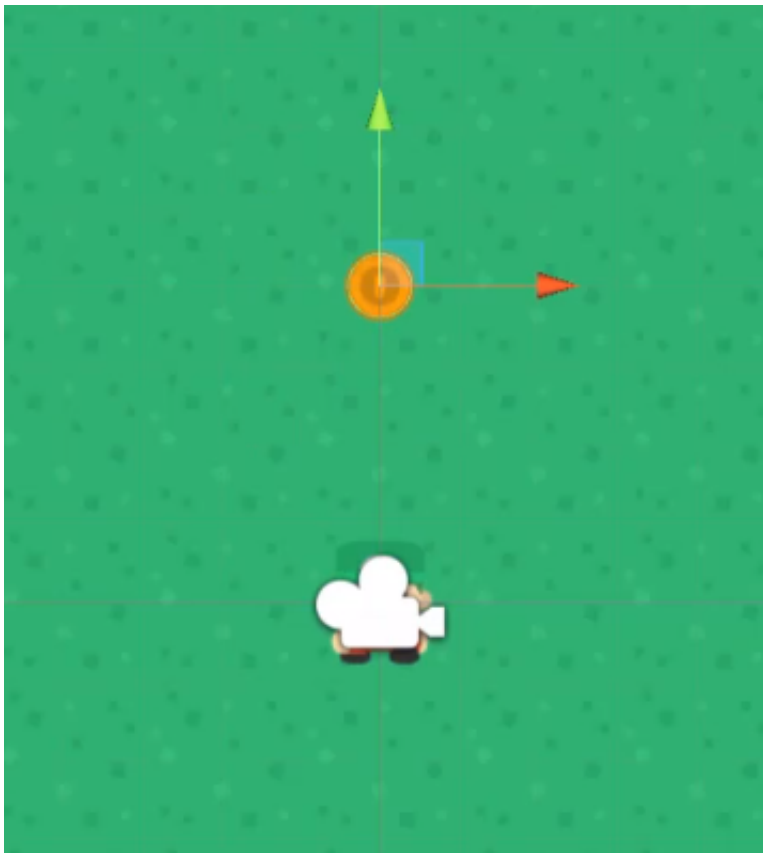
We're going to set the **Radius** to 0.2 and enable the "**Is Trigger**" property, as we don't really want to bounce off this coin.

© 2020 Zenva Pty Ltd. All rights reserved.



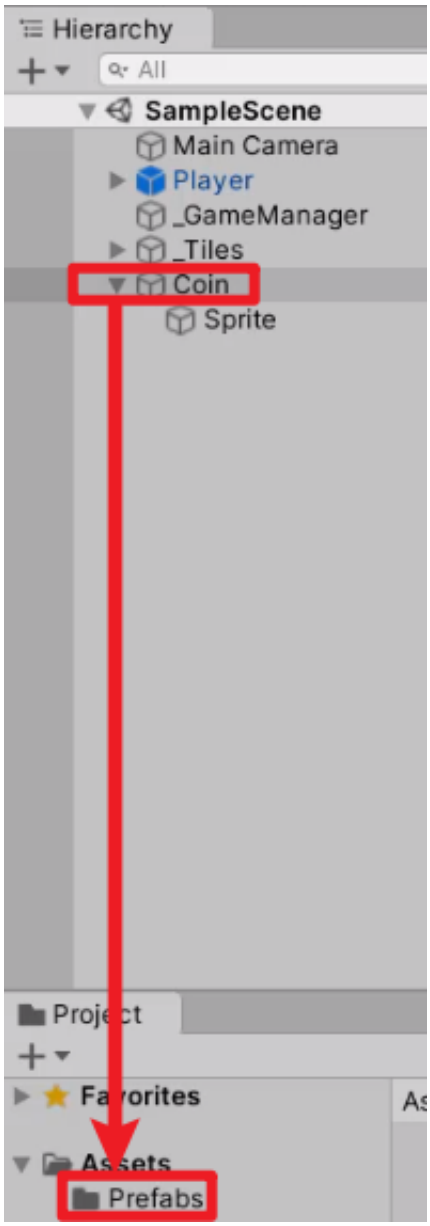
We're going to set its **position** to (0, 2, 0) so that it's aligned to the center of our tile grid:

© 2020 Zenva Pty Ltd. All rights reserved.



With all the steps above done, we can save our Coin object as a **Prefab** and start placing as many coins as we want.

© 2020 Zenva Pty Ltd. All rights reserved.



Implementing Score

Here inside our Coin Script, we're going to declare a new float variable called **amount**. We're going to set its default value to 1.

```
public class Coin : MonoBehaviour
{
    //What value is the coin?
    public int amount = 1;
}
```

We also need to go over to our **Player** script and create a variable to store the player's **score**.

```
public class Player : MonoBehaviour
{
```

© 2020 Zenva Pty Ltd. All rights reserved.

```
    public int score;
    ...
}
```

We’re then going to create a new function called “**AddScoreText**” inside the Player script, which takes in an integer variable (i.e. score to add).

```
public void AddScoreText (int amount)
{
     score += amount;
}
```

Back to our Coin script, we’re going to set up the **OnTriggerEnter2D** function to check if the object this coin collided with has the tag of **Player**.

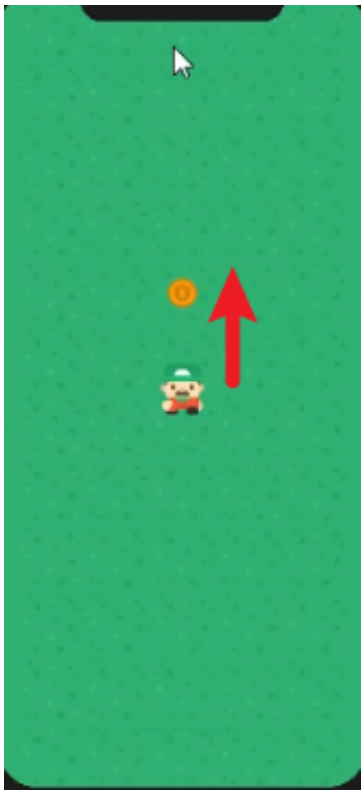
```
public class Coin : MonoBehaviour
{
    //What value is the coin?
    public int amount = 1;

    private void OnTriggerEnter2D (Collider2D collision)
    {
         if(collision.CompareTag("Player"))
         {
         }
    }
}
```

If so, we can access the player script component to call the **AddScore** function.

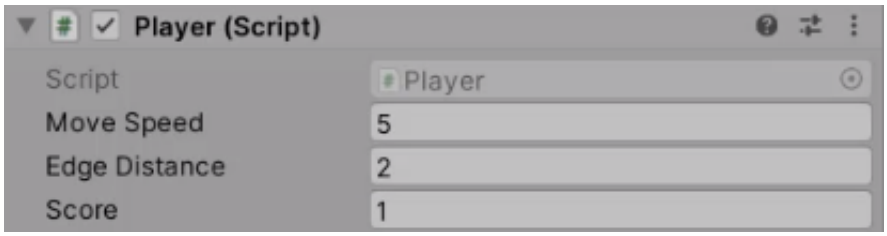
```
public class Coin : MonoBehaviour
{
    //What value is the coin?
    public int amount = 1;

    private void OnTriggerEnter2D (Collider2D collision)
    {
         if(collision.CompareTag("Player"))
         {
              //If player hits this object, add to the player's score by 'amount'.
              collision.GetComponent<Player>().AddScoreText(amount);
              //After collecting the coin, destroy it
              Destroy(this.gameObject);
         }
    }
}
```

© 2020 Zenva Pty Ltd. All rights reserved.

Let's save both scripts and hit **Play.**



Keep an eye on the Score variable in the Inspector. Once we go and collect the coin, you'll see that our **Score** variable increases.

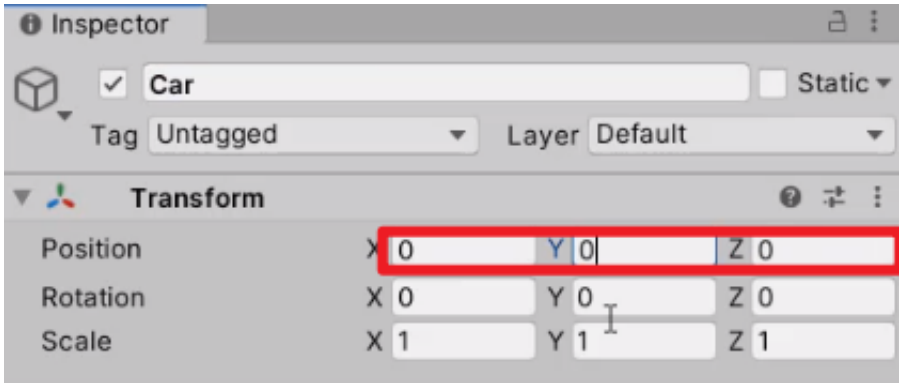


© 2020 Zenva Pty Ltd. All rights reserved.

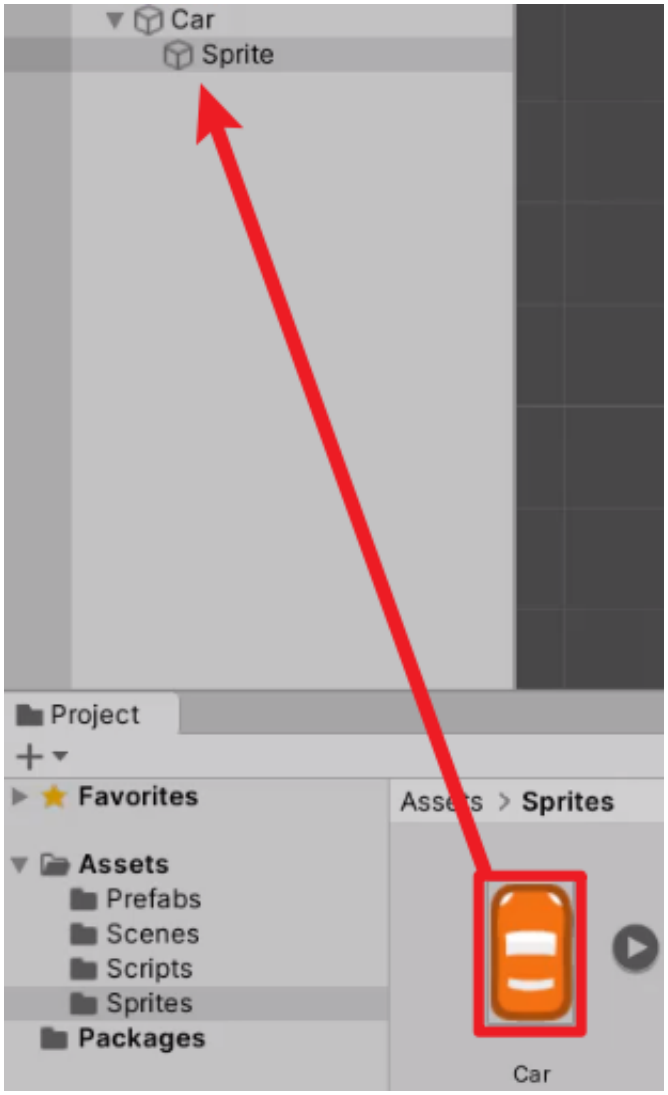
In this lesson, we're going to be working on the main obstacle in the game– cars. Cars are going to be moving left and right as our player moves up along the screen.

Car GameObject

Let's create a new gameObject named "Car" and set the **position** to be 0, 0, 0.

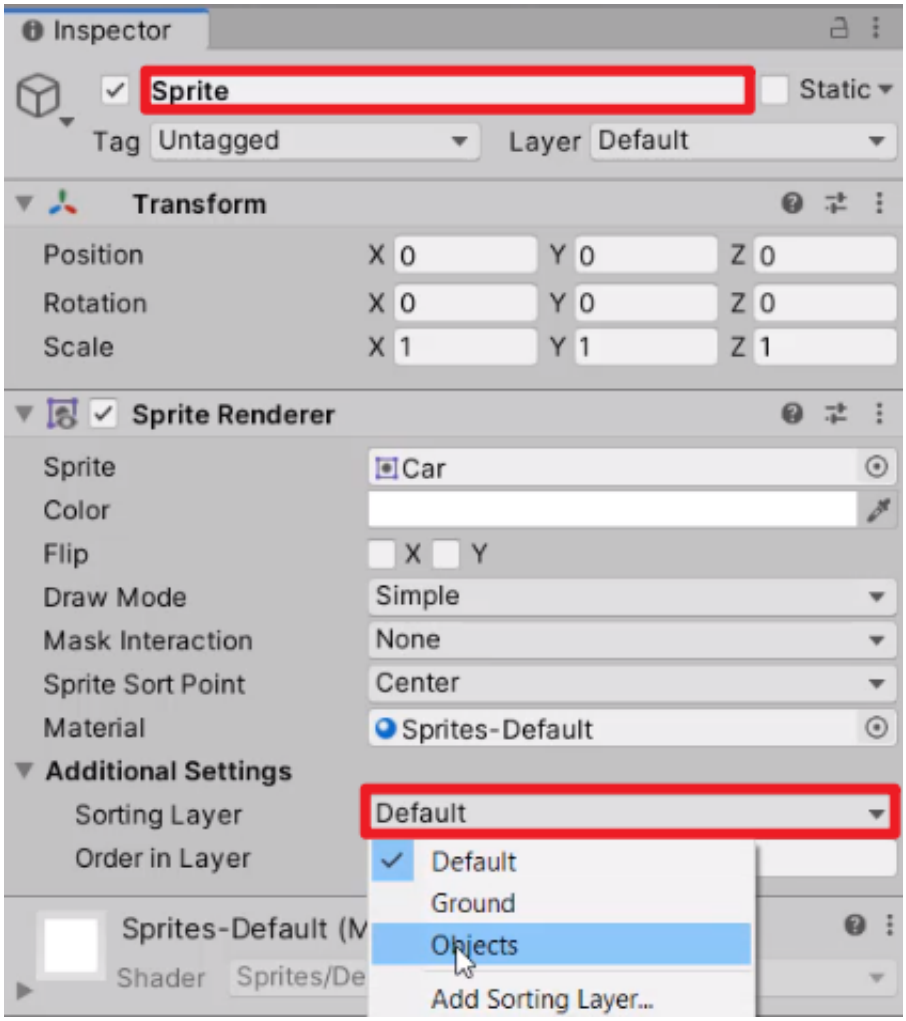


And just as we did with the Coin, we're going to drag in a **Sprite** as a child.



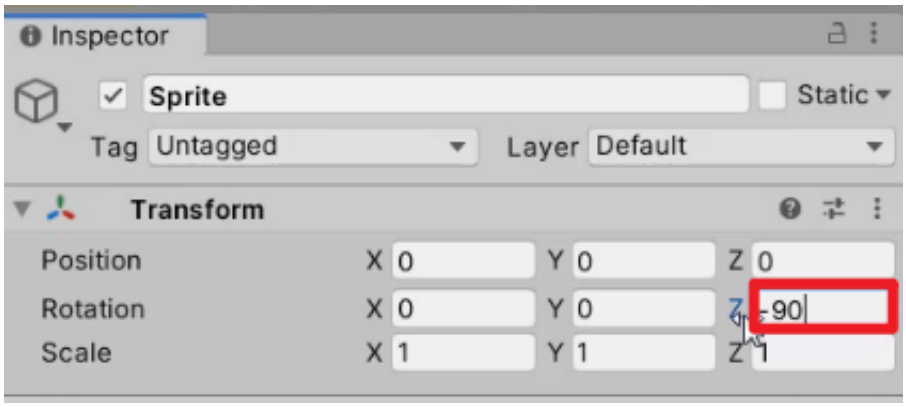
© 2020 Zenva Pty Ltd. All rights reserved.

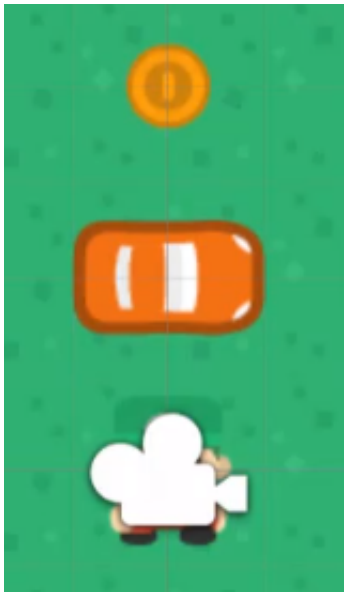
We're going to set the **Sorting Layer** to **Objects**, so it appears on top of the ground.



We're going to **rotate** the sprite 90 degrees clockwise (= negative 90 on the Z-axis) so that it is facing the right side of the screen.

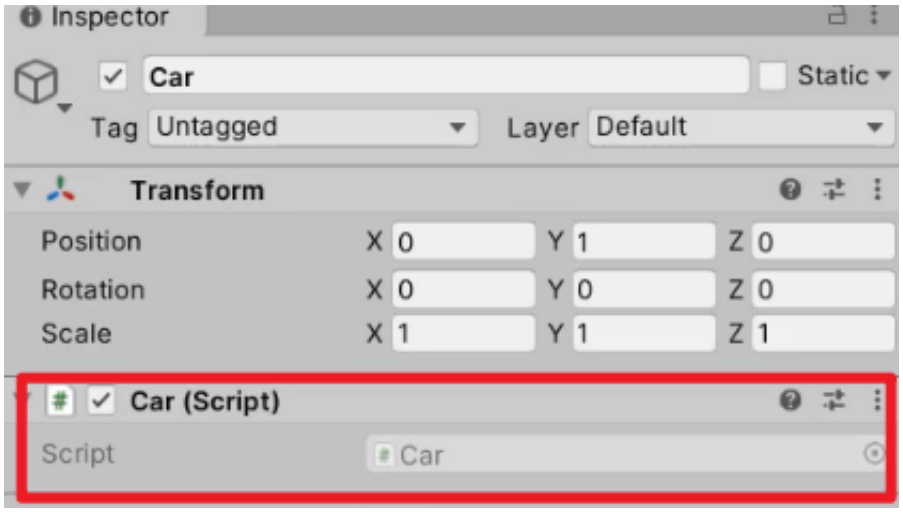
© 2020 Zenva Pty Ltd. All rights reserved.





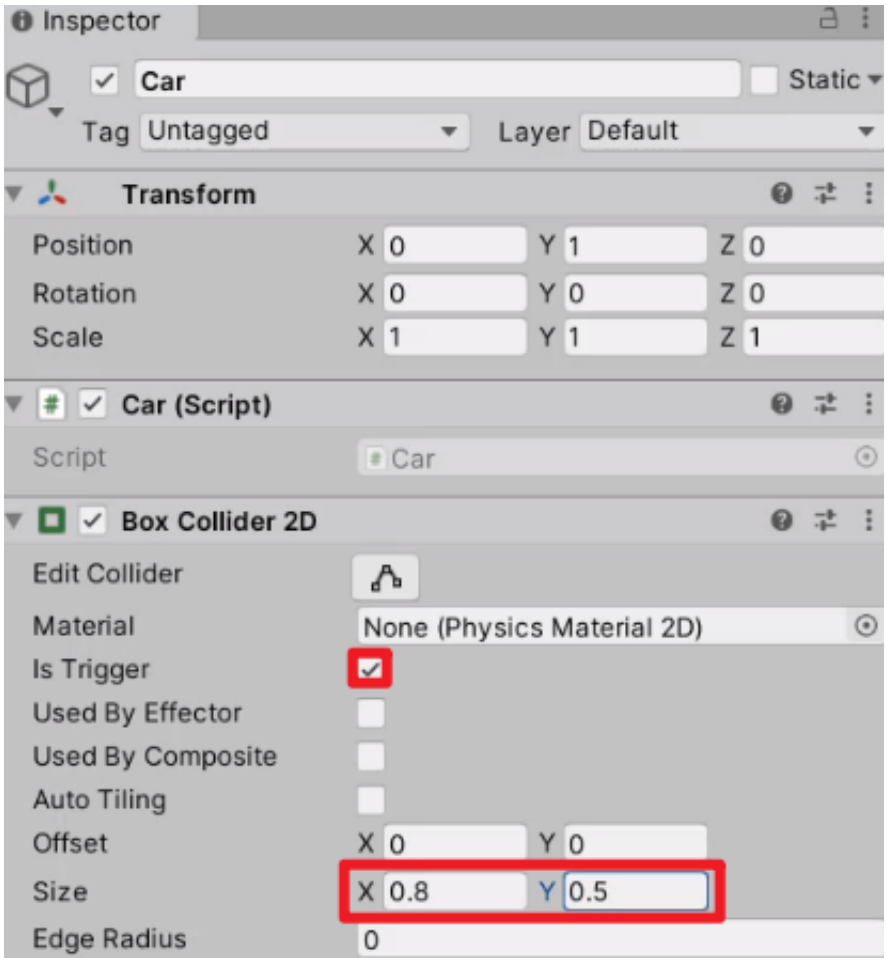
Adding Components

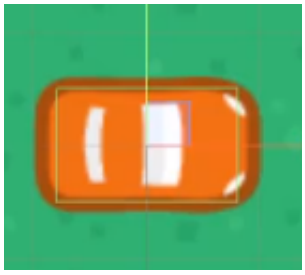
We're going to create a new **C# script** called "Car" and attach it to our Car object.



We're then going to add a **Box Collider 2D** component, set it to be a **Trigger,** and adjust its **Size** to (0.8, 0.5).

© 2020 Zenva Pty Ltd. All rights reserved.





Moving The Car

We're going to make our car move between two positions: a **start position** and an **end position**.

```
public class Car : MonoBehaviour
{
    public float speed;
    public Vector3 startPos;
    public Vector3 endPos;
}
```

Inside the **Update** function, we're going to move our car towards the **endPos** over time using **Vector3.MoveTowards**.

© 2020 Zenva Pty Ltd. All rights reserved.

```
public class Car : MonoBehaviour
{
    public float speed;
    public Vector3 startPos;
    public Vector3 endPos;

    void Update ()
    {
         //Move towards the end position
         transform.position = Vector3.MoveTowards(transform.position, endPos, speed *
 Time.deltaTime);
    }
}
```

Note that we multiplied the speed by **Time.deltaTime** to convert it from per frame to **per second**.

Once it reaches the end position, the car is going to get teleported back to the start position and keep on going in that loop.

```
public class Car : MonoBehaviour
{
    public float speed;
    public Vector3 startPos;
    public Vector3 endPos;

    void Update ()
    {
         //Move towards the end position
         transform.position = Vector3.MoveTowards(transform.position, endPos, speed *
 Time.deltaTime);

         //If it reaches the end position,
         if(transform.position == endPos)
         {
              //Teleport the car back to the start position
              transform.position = startPos;
         }

    }
}
```

Let’s go to our Player script and create a new function named **GameOver**. We’re going to be setting up a UI for this in the future lessons, but for now, let’s just send a message to the console.

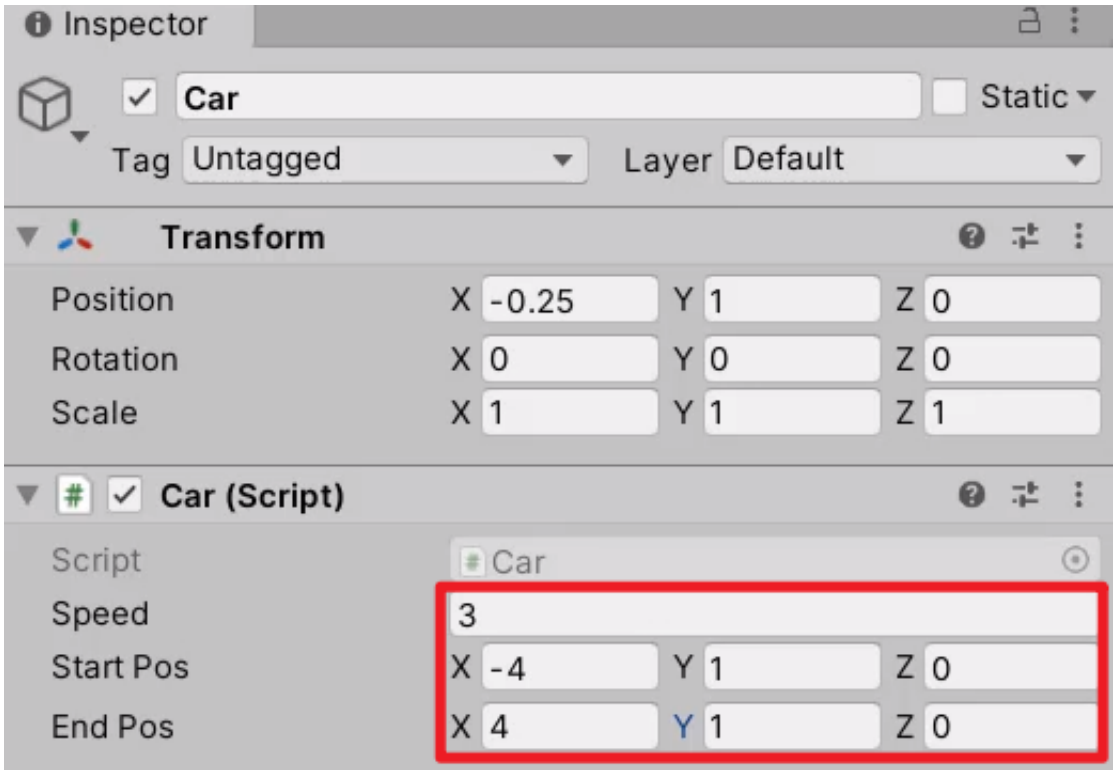
```
public class Player : MonoBehaviour
{
    ...
    public void GameOver ()
    {
        Debug.Log("Game Over");
    }
```

© 2020 Zenva Pty Ltd. All rights reserved.

Back inside our Car script, we're going to use **OnTriggerEnter2D** so that when the car hits the Player, we can call the **GameOver** function.

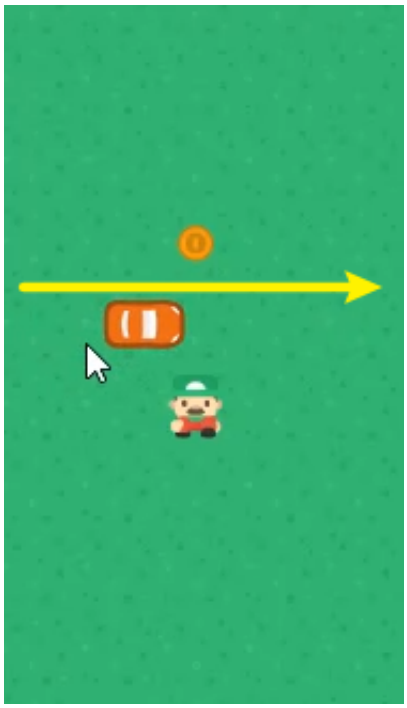
```
public class Car : MonoBehaviour
{
    ...
    private void OnTriggerEnter2D (Collider2D collision)
    {
        if(collision.CompareTag("Player"))
        {
            //If player hits this object, call GameOver()
            collision.GetComponent<Player>().GameOver();
        }
    }
}
```

Let's save and go back to the Editor now, and set values for **Speed**, **StartPos,** and **EndPos**.

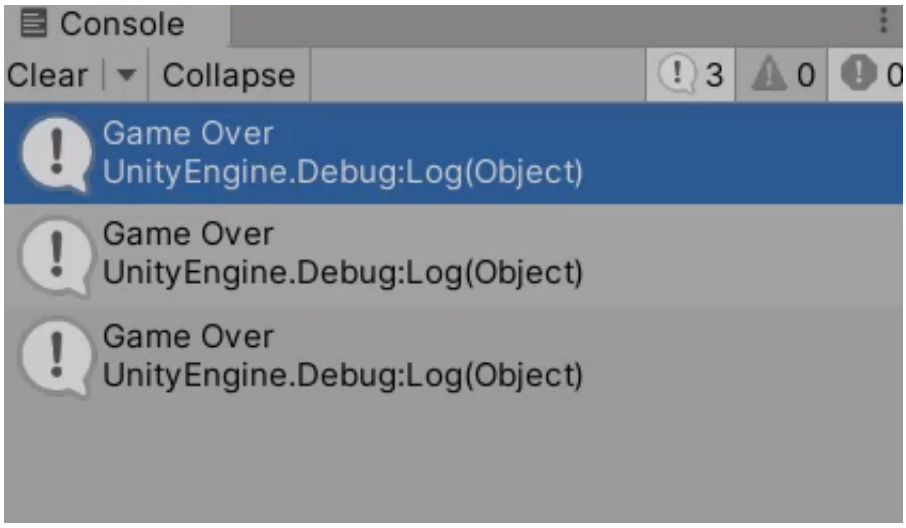


If you press **Play**, you'll see that our car moves from left to right.

© 2020 Zenva Pty Ltd. All rights reserved.

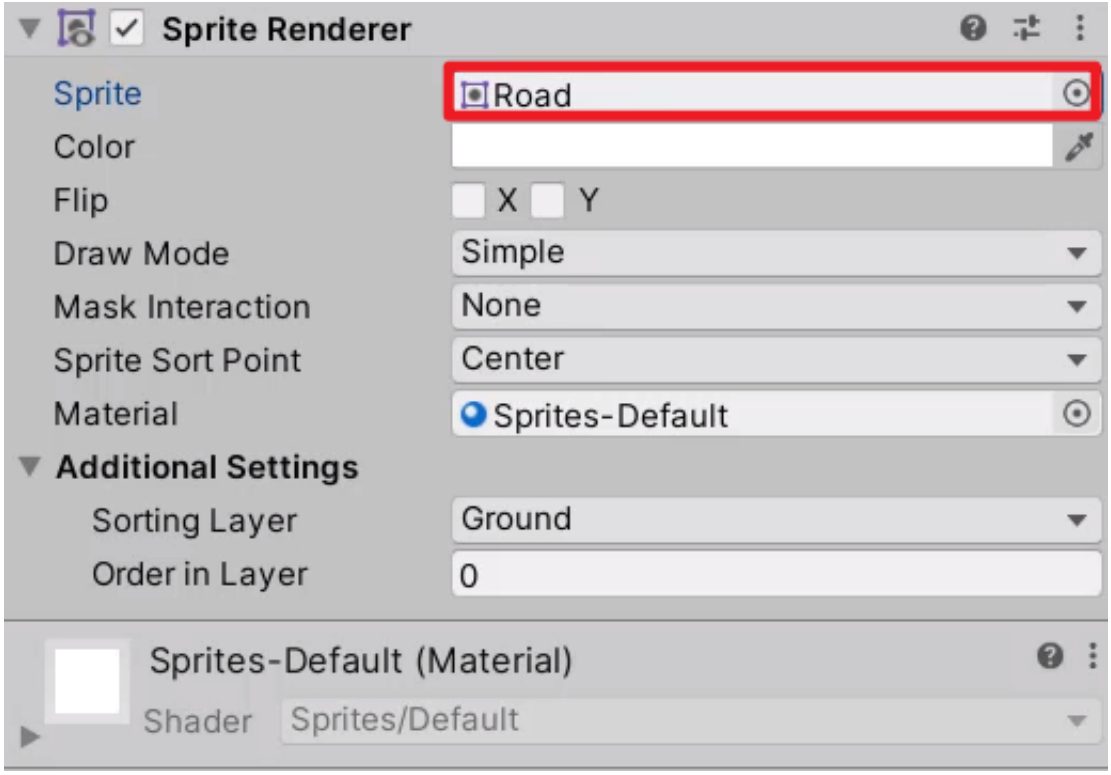


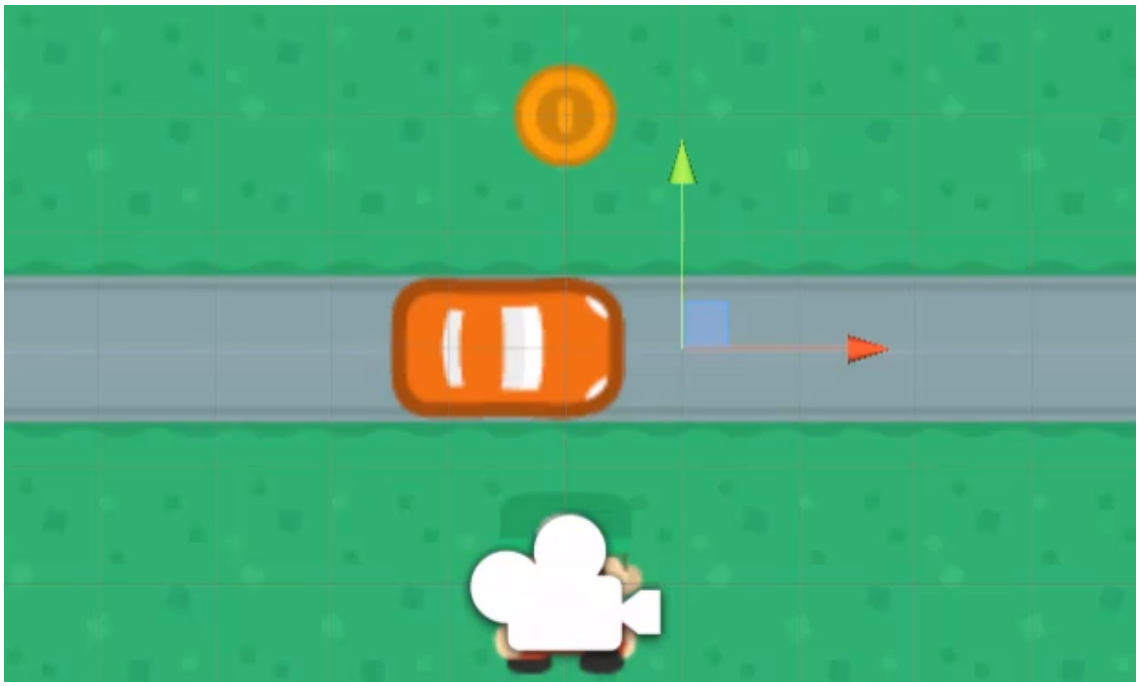
And when we get hit by the car, we should be able to see this message on the console.



Lastly, we're going to select the tiles underneath the car, and set its **Sprite** to **Road**.

© 2020 Zenva Pty Ltd. All rights reserved.

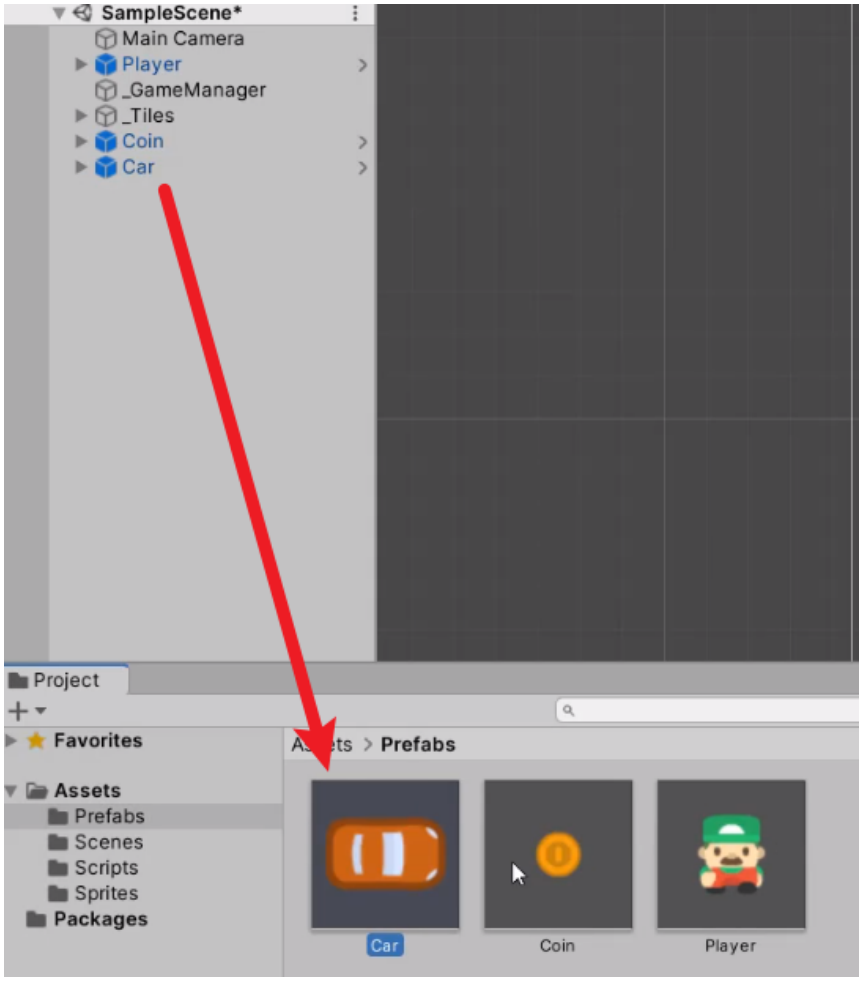




© 2020 Zenva Pty Ltd. All rights reserved.

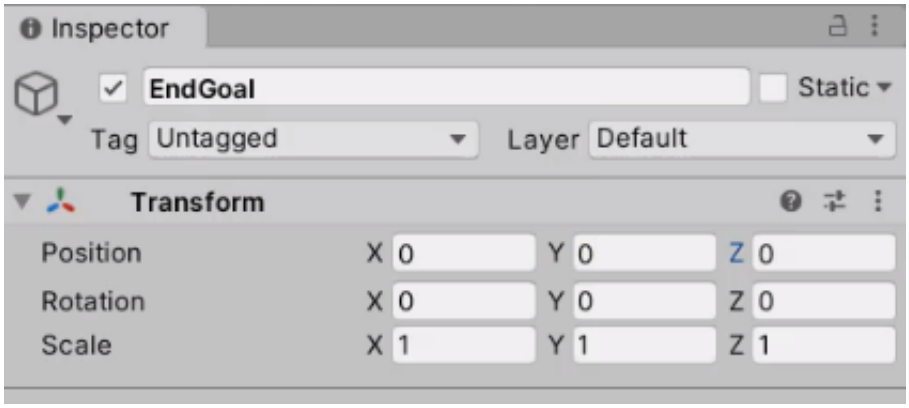
In this lesson, we're going to be working on the **End Goal**— the end of the level where the player should reach in order to win the game.

First of all, make sure that your essential game objects are saved as **Prefabs**:



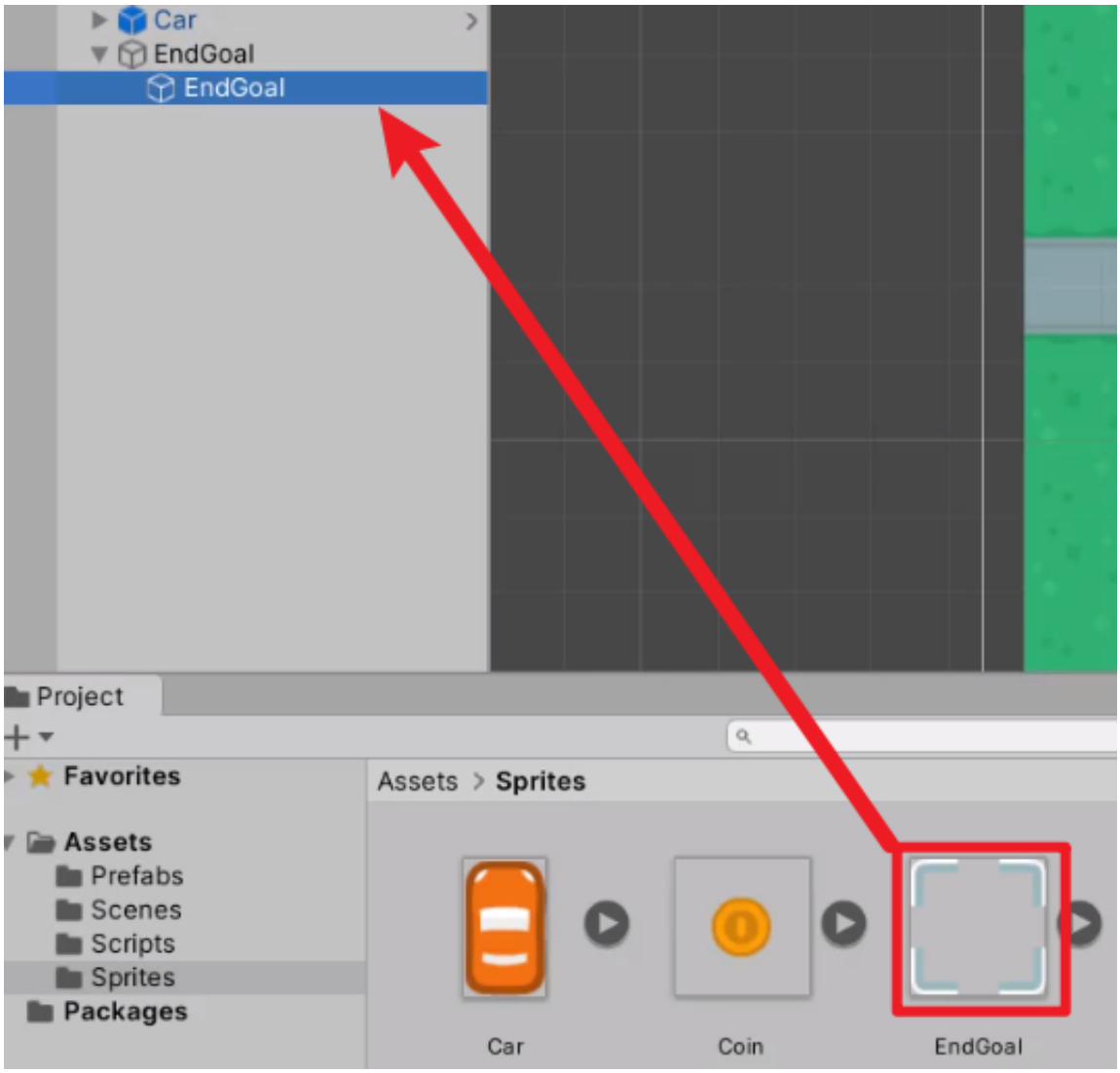
Creating EndGoal

Let's create a new **Empty gameObject**, name it "EndGoal" and set its position to (0, 0, 0).



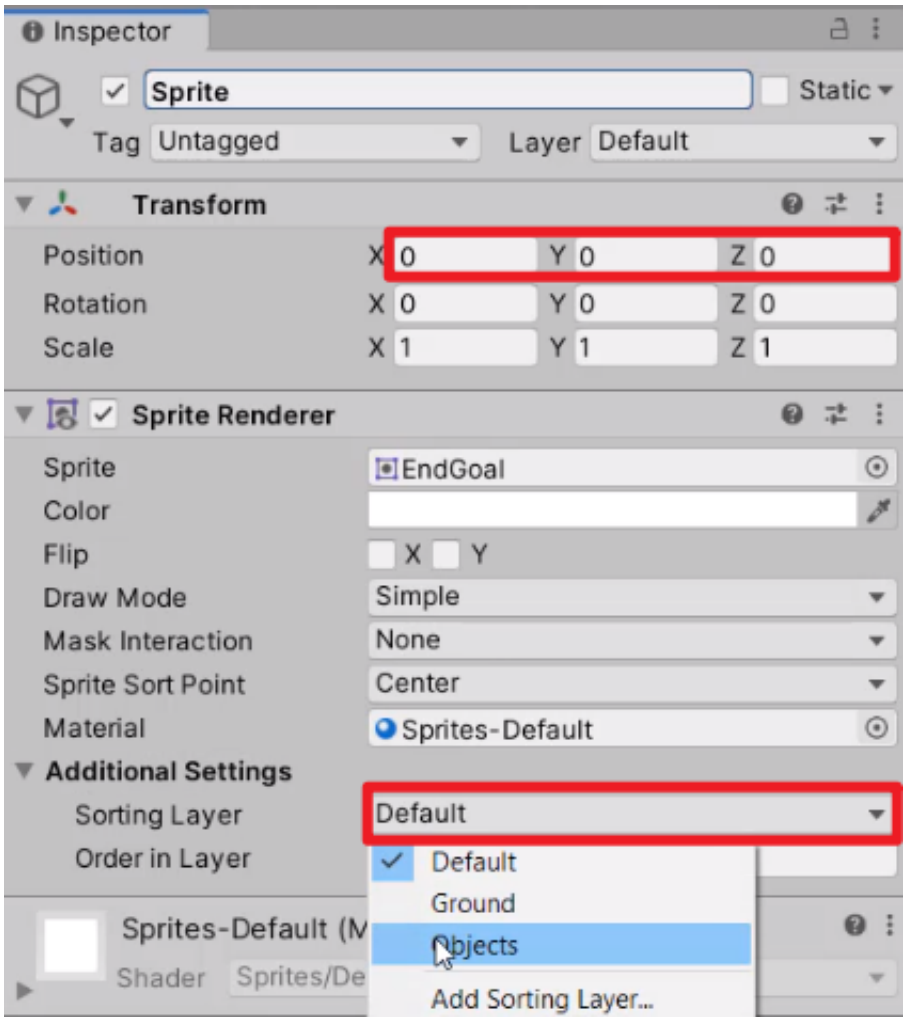
Then, let's go over to the Sprites folder (**Assets > Sprites**), select **EndGoal**, and drag it in as a child object to the **EndGoal** gameObject.

© 2020 Zenva Pty Ltd. All rights reserved.

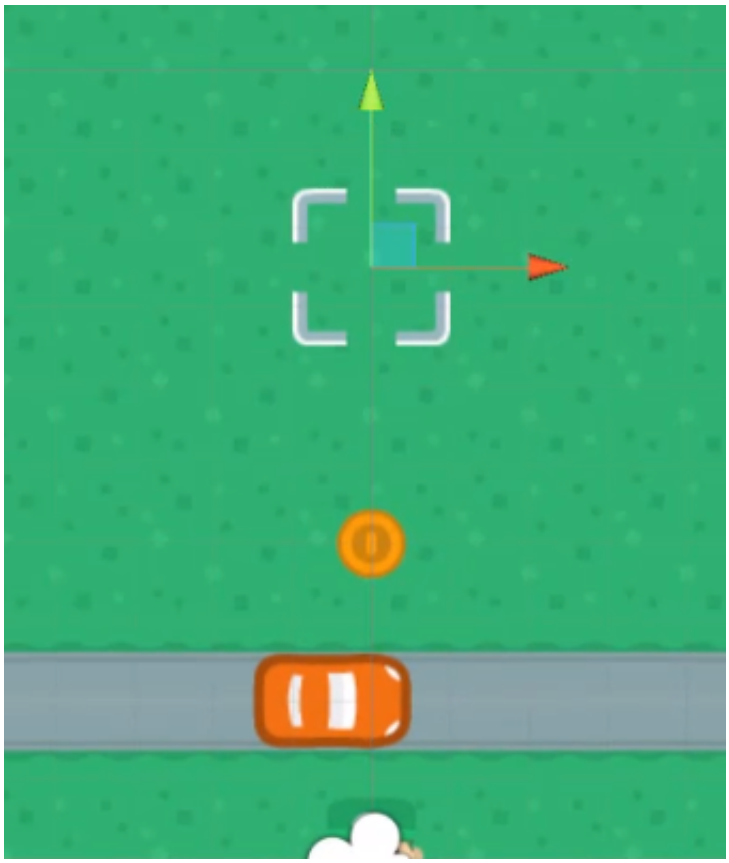


Make sure to set the sprite's **Sorting Layer** to be **Objects** so that it appears on top of the ground, and reset its **position** to be (0, 0, 0).

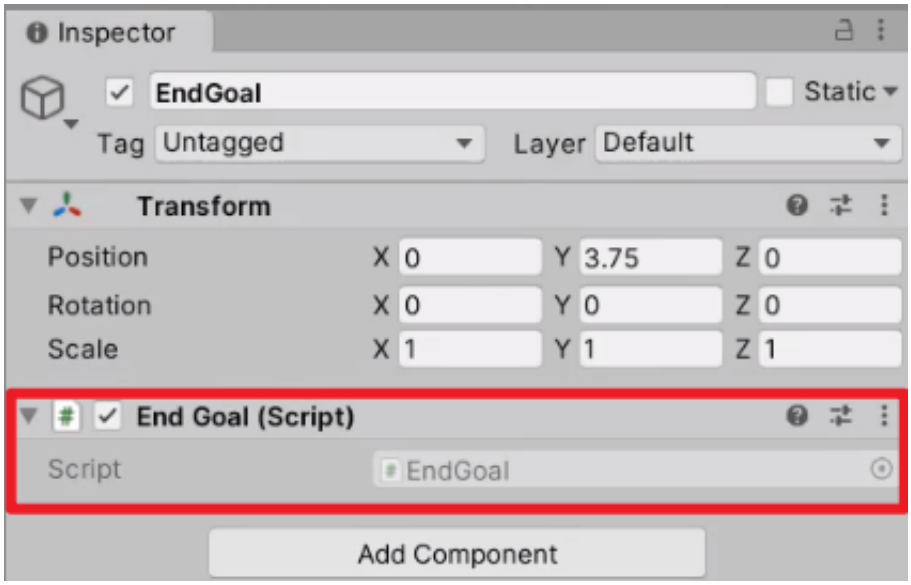
© 2020 Zenva Pty Ltd. All rights reserved.



© 2020 Zenva Pty Ltd. All rights reserved.

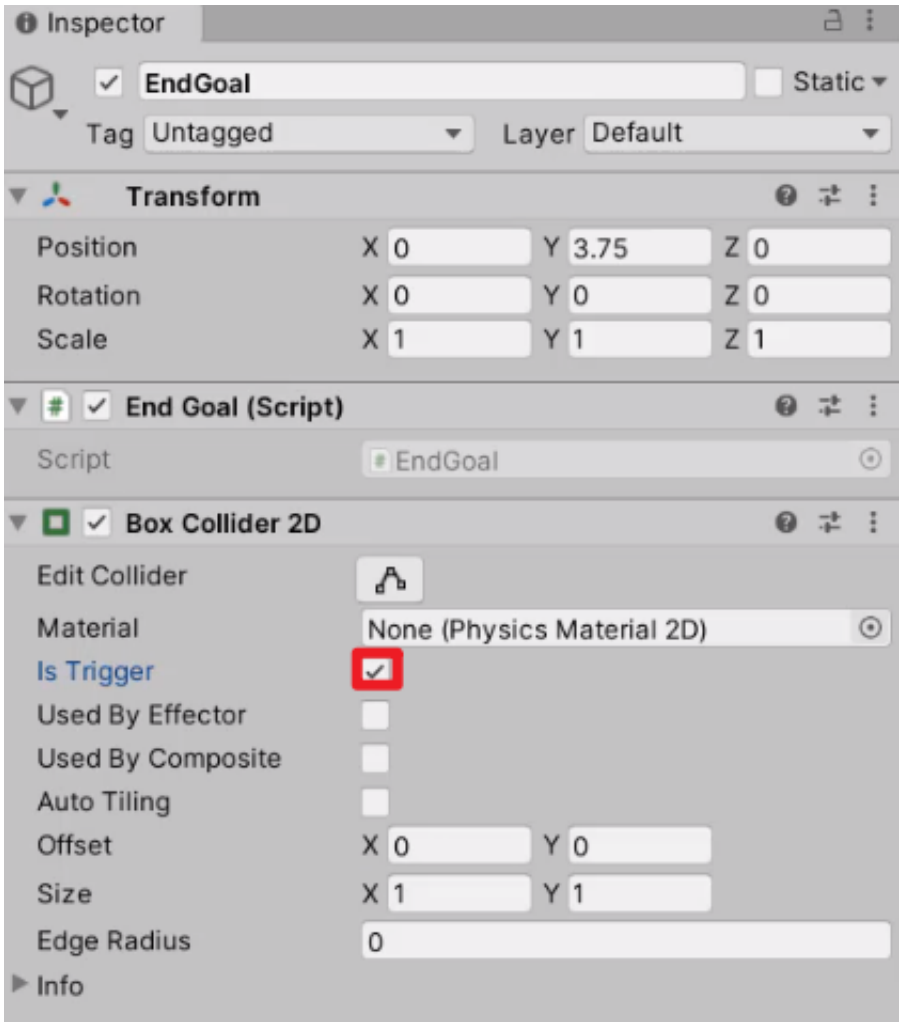


Create a new **C# script** called “EndGoal” and **attach** it to the object.



We’re going to add a **Box Collider 2D** component and set it to be a **Trigger.**

© 2020 Zenva Pty Ltd. All rights reserved.



Scripting End Game Functionality

We're going to create an **OnTriggerEnter2D** function, which detects if it has collided with an object tagged as "Player":

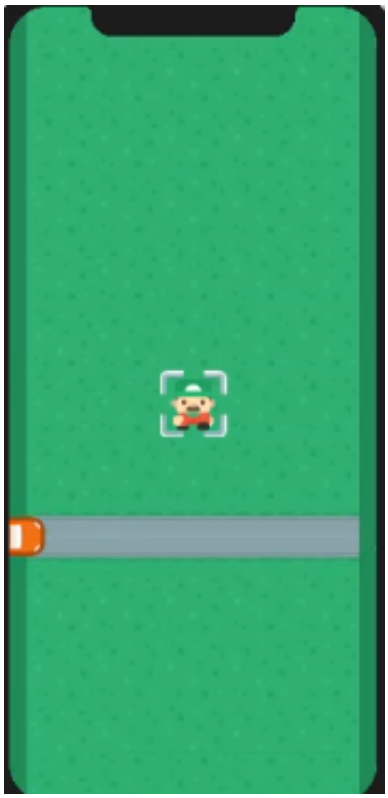
```
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.CompareTag("Player"))
    {
         collision.gameObject.GetComponent<Player>().Win();
         //To-do: create a function called Win() in the Player script.
    }
}
```

We're then going to go over to **Player.cs** and create a new function called **Win()**. Inside the function, we're going to add in a **Debug.Log** for now:

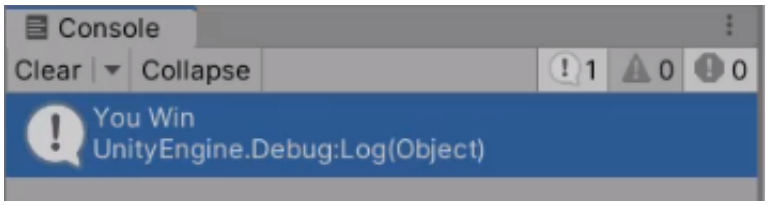
```
public void Win()
{
    Debug.Log("You Win!");
}
```

© 2020 Zenva Pty Ltd. All rights reserved.

Now let’s hit save and press **Play**.



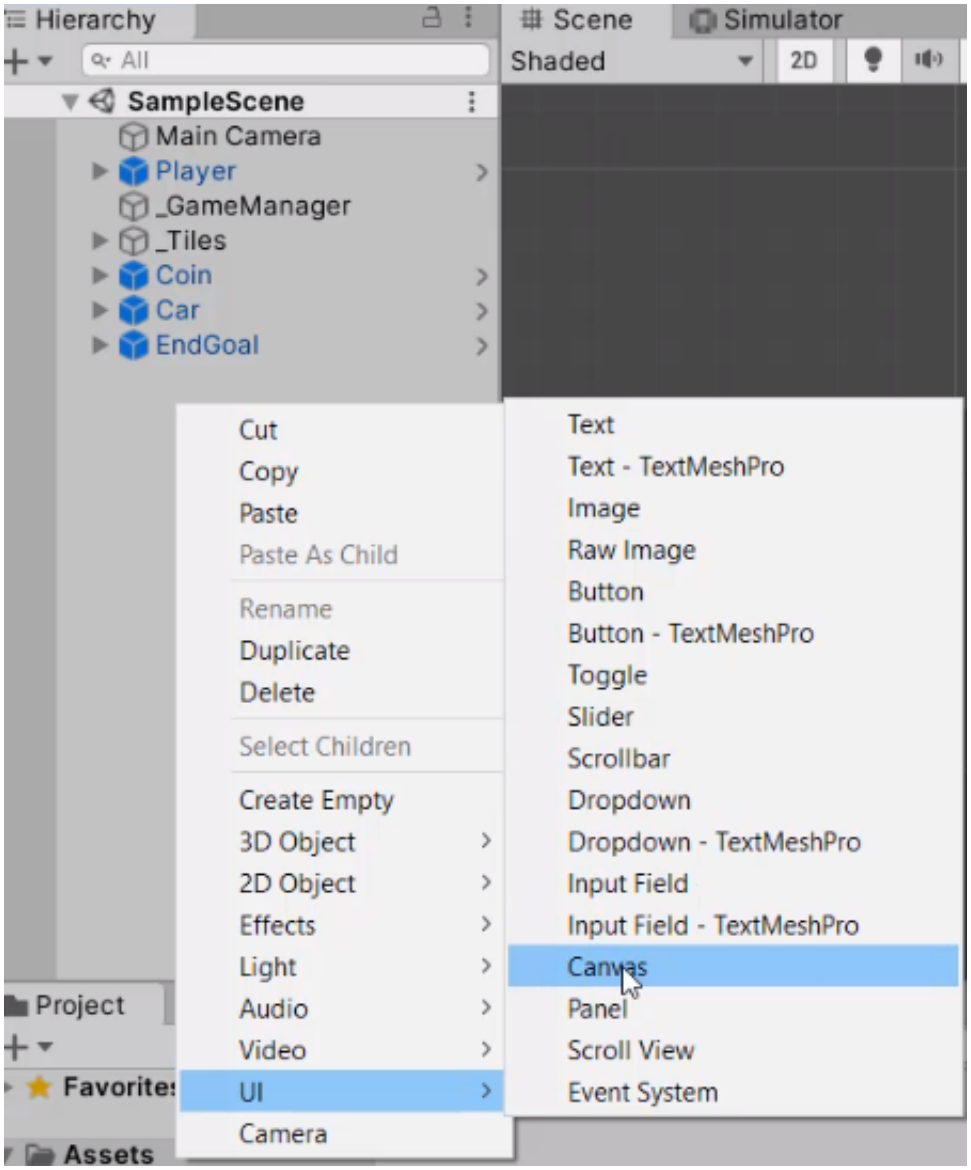
The moment we enter the EndGoal, we should be able to see the **Debug.Log** message in the console.



© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be setting up the **UI** (User Interface) for our Win screen.

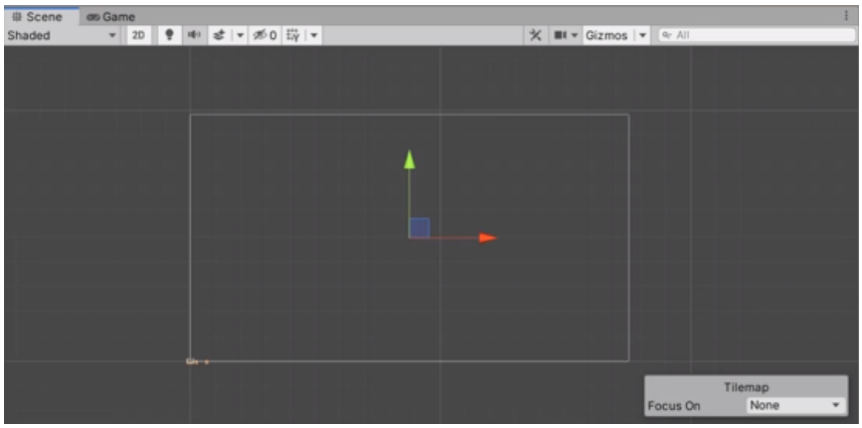
First, we're going to create a new gameObject called **Canvas,** which contains all of the UI elements. To create a canvas, right-click on the **Hierarchy > UI > Canvas**.



When you create any UI element object, a **Canvas** object will be created automatically if there isn't one in the scene already.

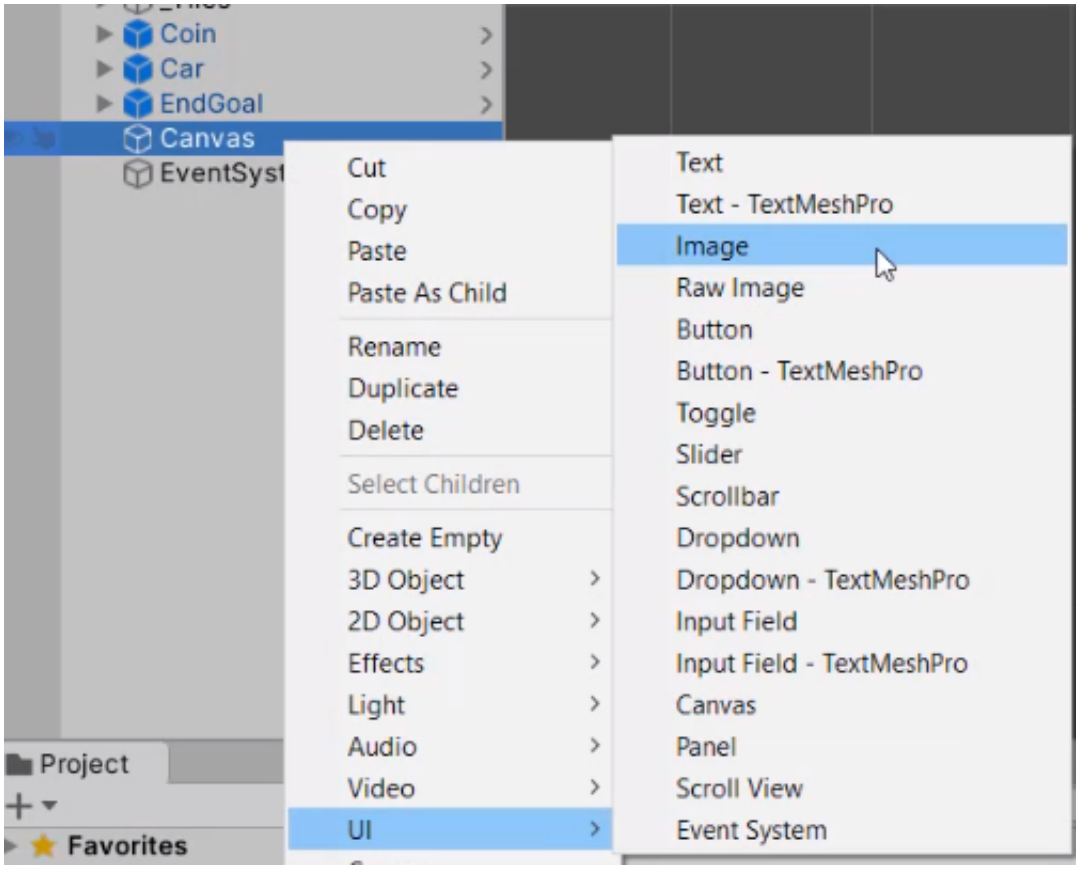
If you select the Canvas and **press F**, the screen will be zoomed out to **focus** on it.

© 2020 Zenva Pty Ltd. All rights reserved.

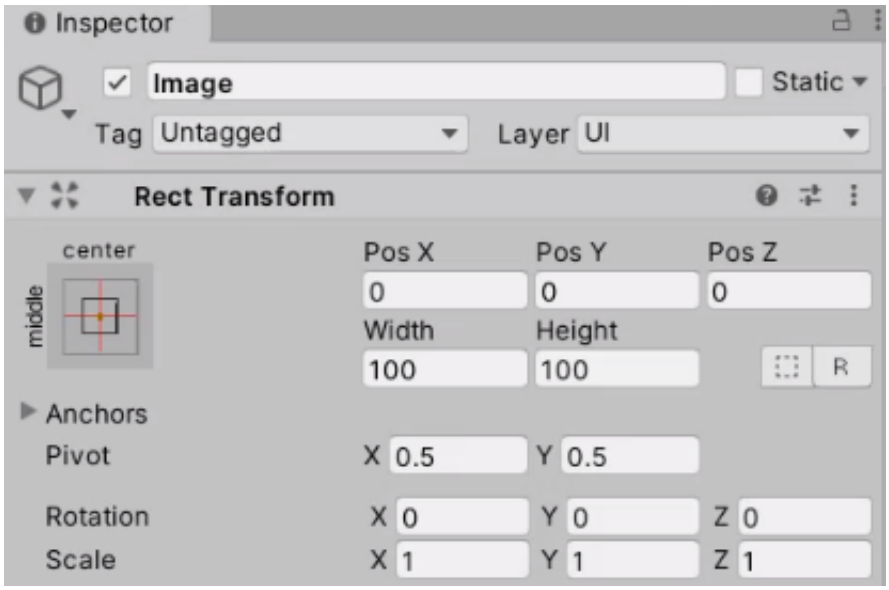


Adding Image To Canvas

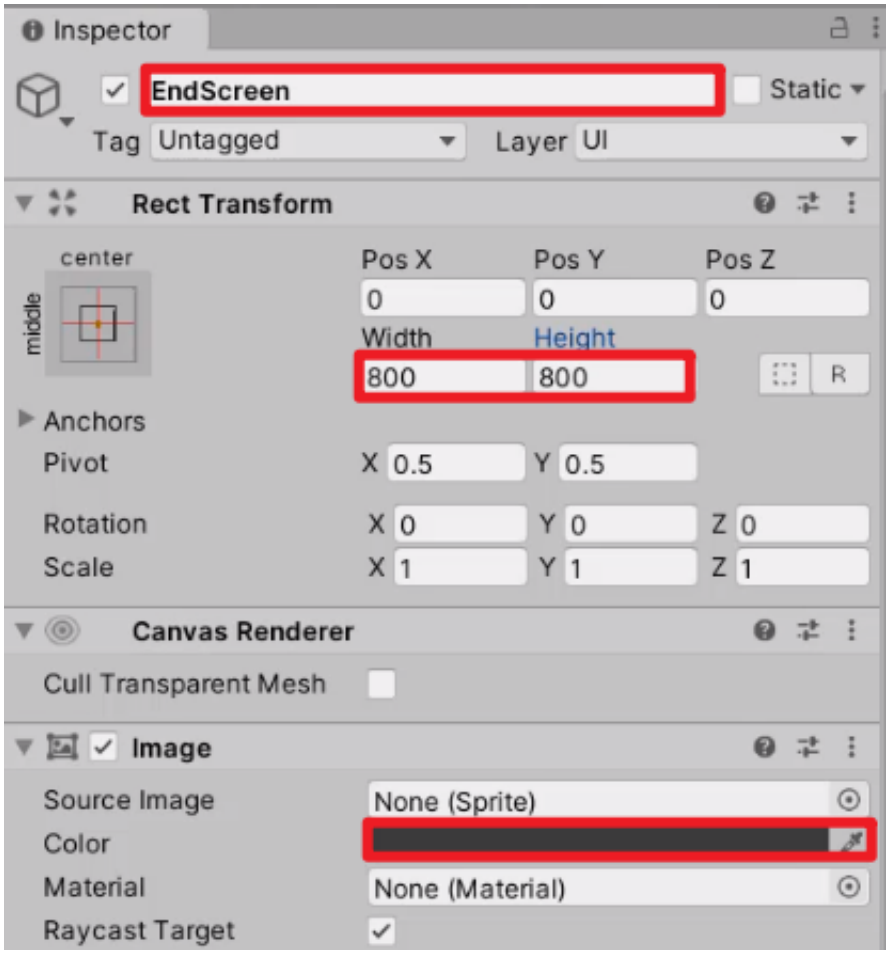
You can add an **Image** object to the Canvas if you **right-click** on it, and go to **UI > Image**.



© 2020 Zenva Pty Ltd. All rights reserved.

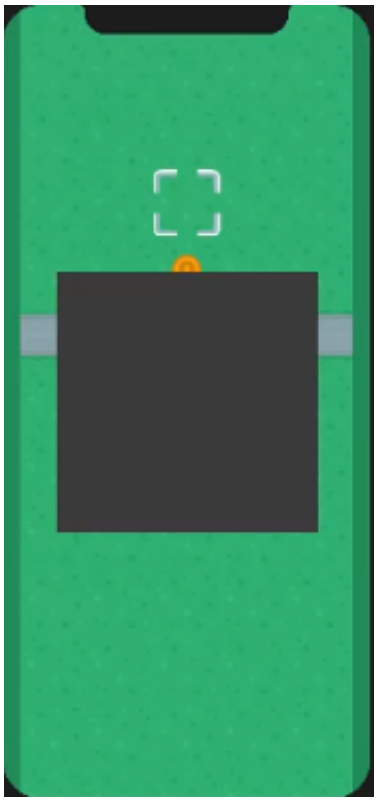


We're going to **rename** this image to be "EndScreen", set the **Color** to be black, and set its **position** to be 800 x 800.



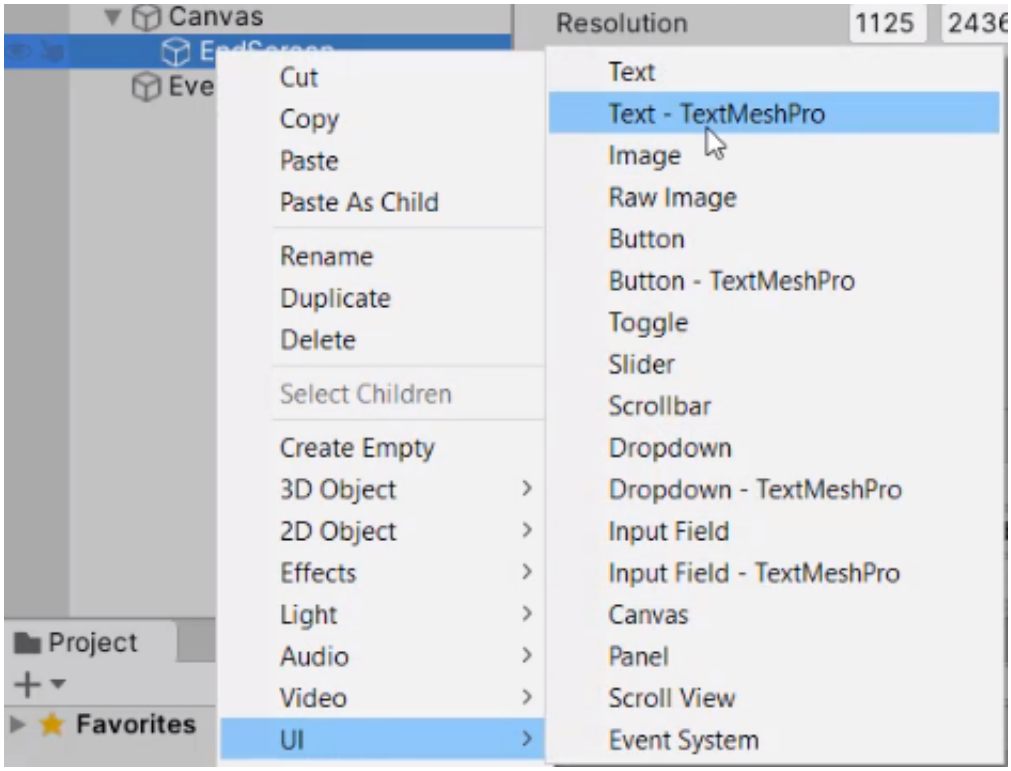
This is what EndScreen is going to look like in the actual game.

© 2020 Zenva Pty Ltd. All rights reserved.



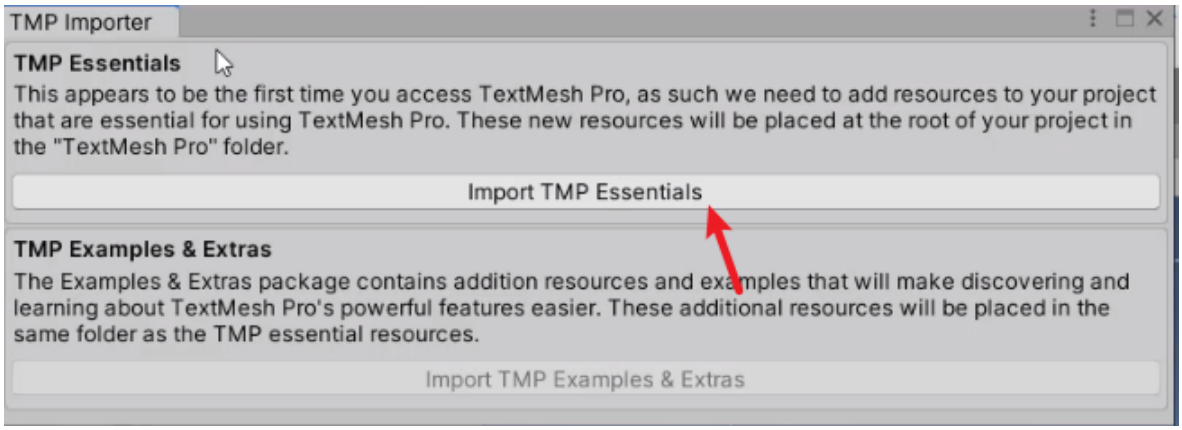
Adding Text To Canvas

You can add a **Text** object to the Canvas if you **right-click** on it, and go to **UI > Text – TextMeshPro**.

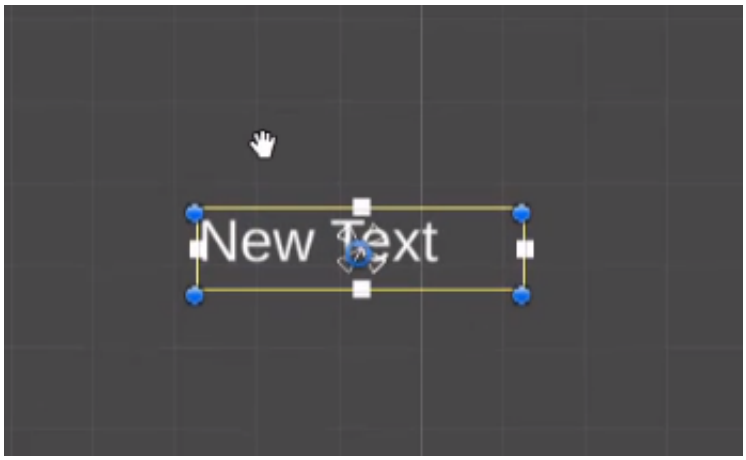


You may see this pop-up window. Click '**Import TMP Essentials**'.

© 2020 Zenva Pty Ltd. All rights reserved.



Once that's done, you'll see a text appearing in the Scene view.

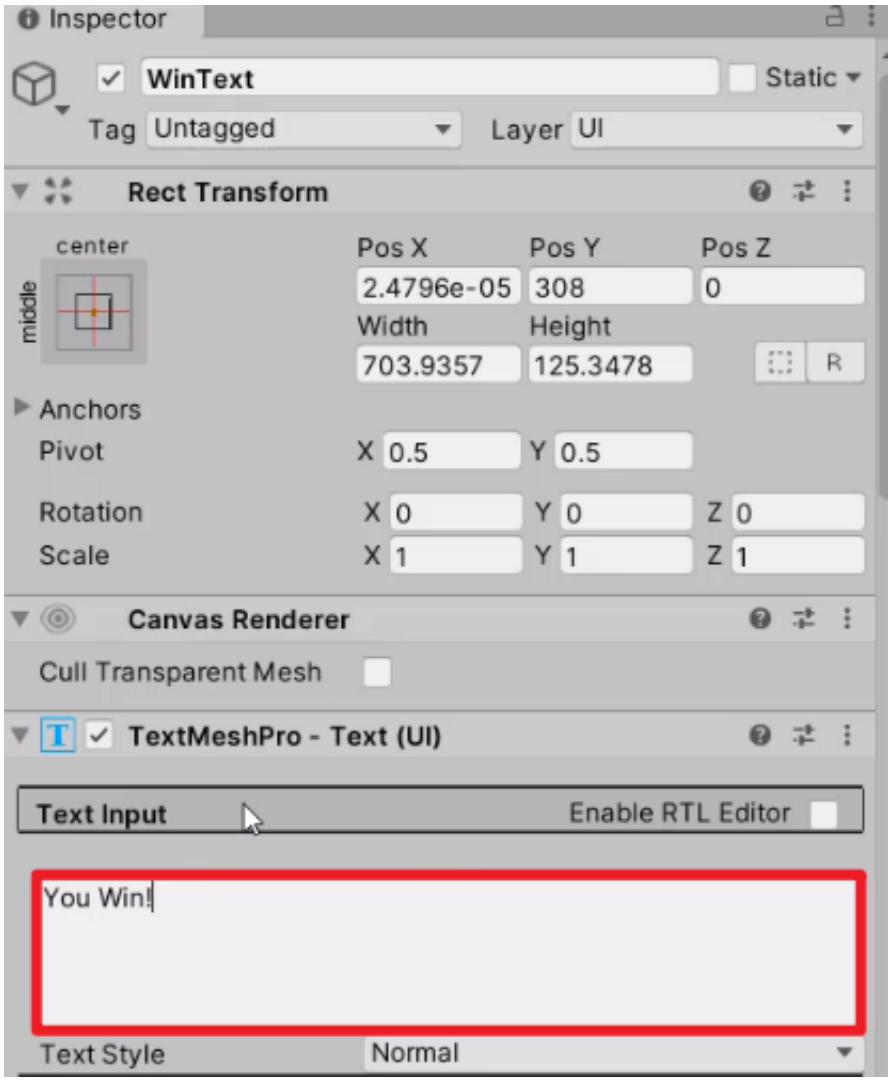


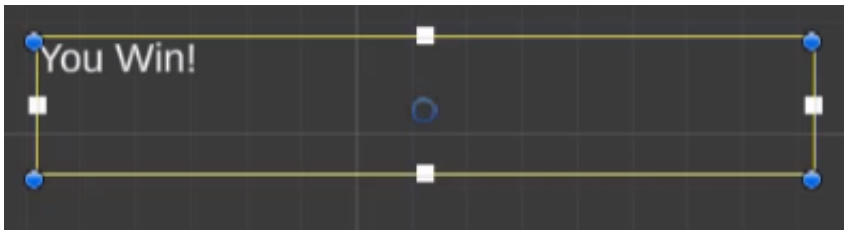
Modifying UI Element

You can click and drag on the blue circles to change the **size** of the textbox.

We're going to change the **Text Input** to "You Win!":

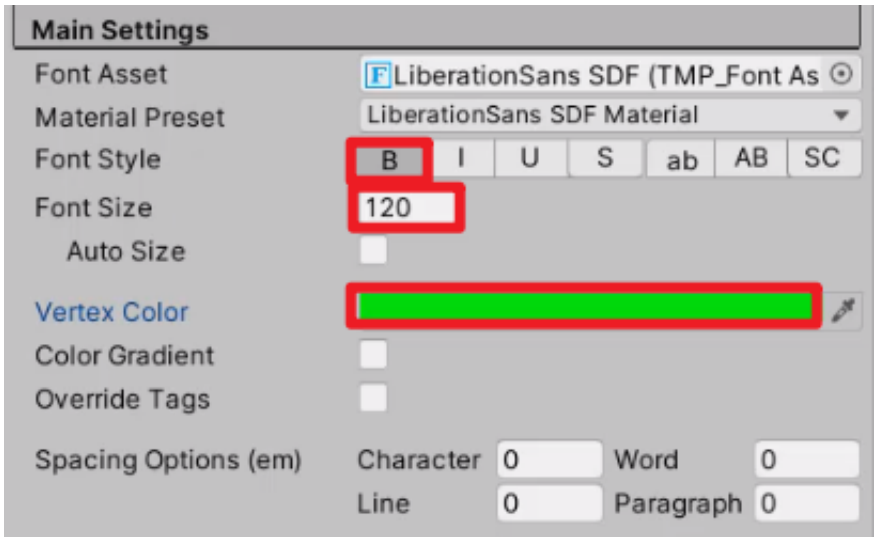
© 2020 Zenva Pty Ltd. All rights reserved.



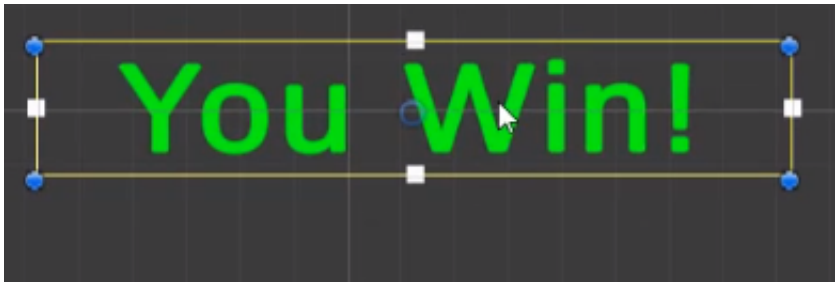


And we're going to change its **font size** to be 120, make it **bold**, and change the **vertex color** in the **Font settings**.

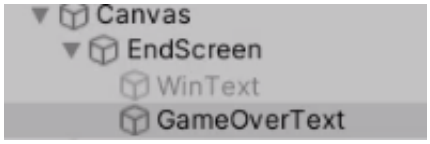
© 2020 Zenva Pty Ltd. All rights reserved.



The text is currently aligned to the top right. Let's change it to be aligned at the **Center**.



Now, we also need to **duplicate** (**Ctrl+D** or **Cmd +D**) this text for a version when the player loses. We're going to call this "GameOverText".

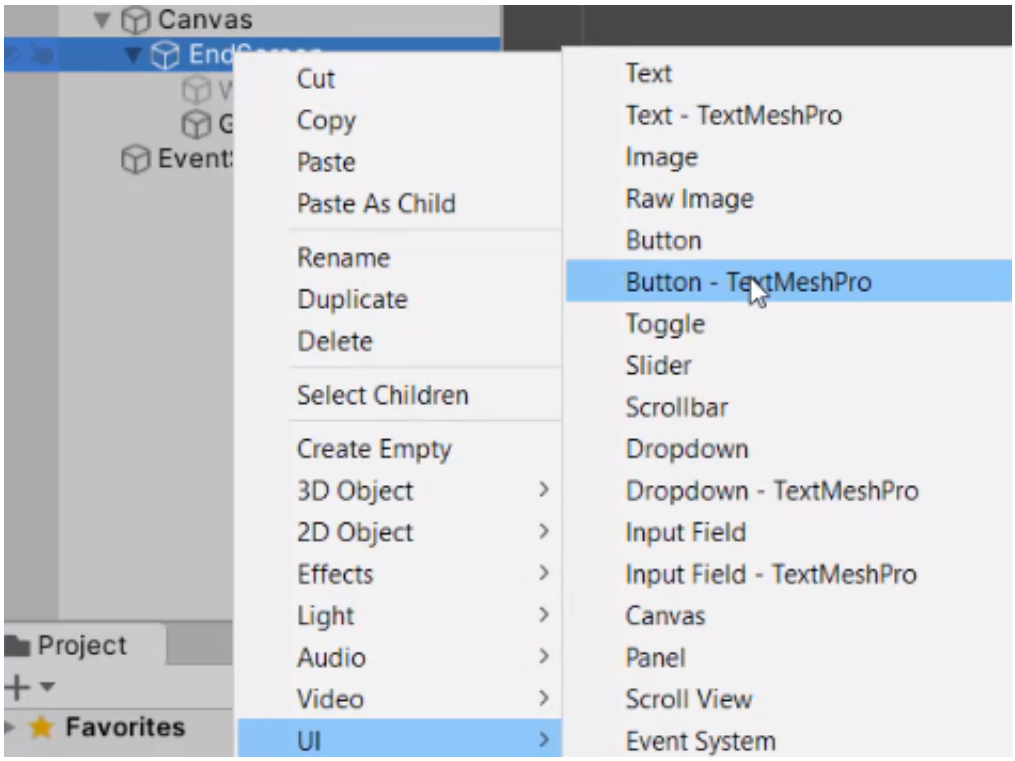


We're going to change the **text input** to "Game Over!", and change the font color to red.

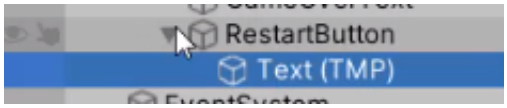
Adding A Button

We're going to create a new **Button** GameObject to **EndScreen**, and rename this to be "RestartButton".

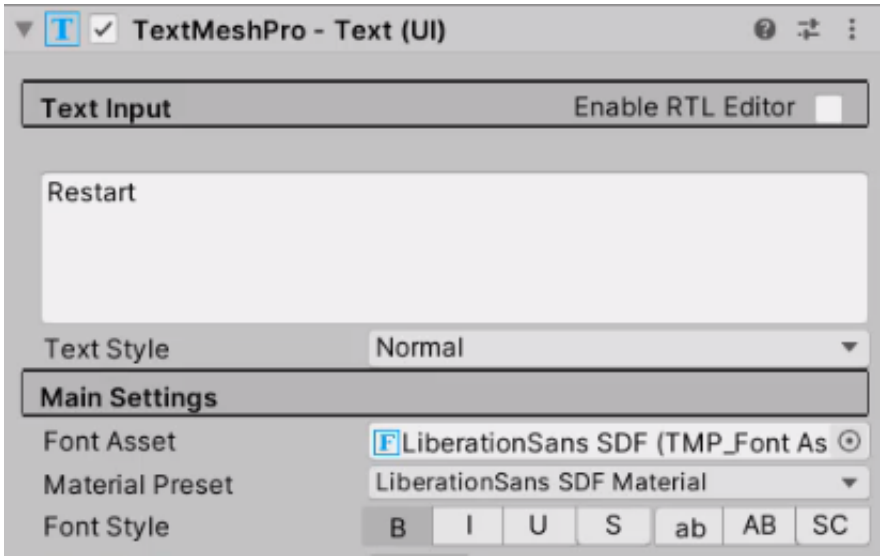
© 2020 Zenva Pty Ltd. All rights reserved.



Inside the Button, there is a **Text** element as a child object.



We're going to select it and type in "Restart":

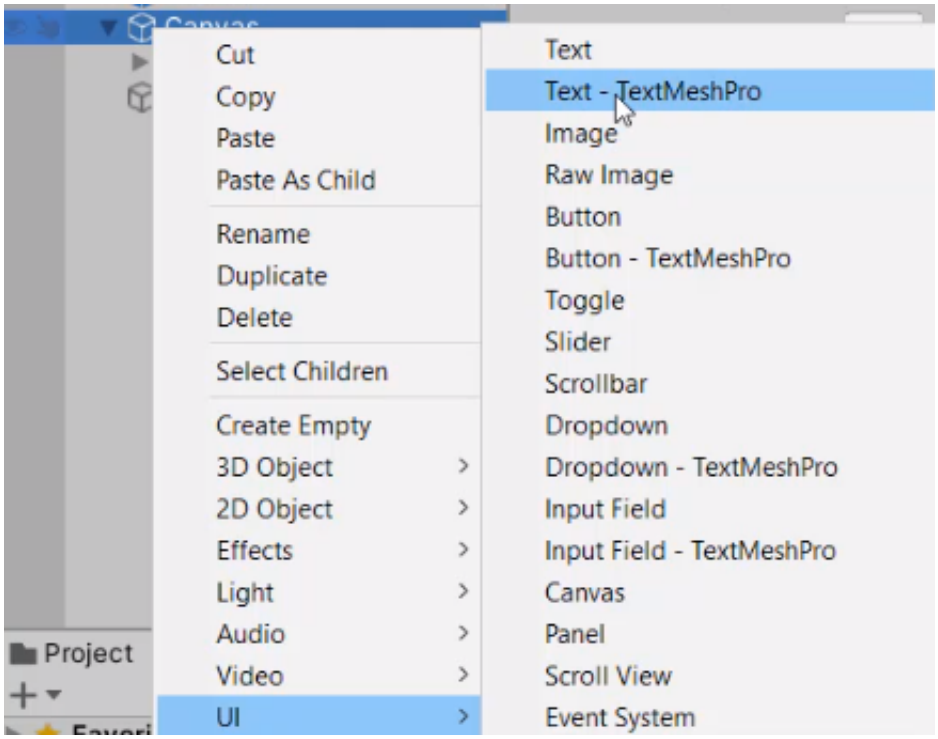


And now we have a simple Game Over screen.

© 2020 Zenva Pty Ltd. All rights reserved.

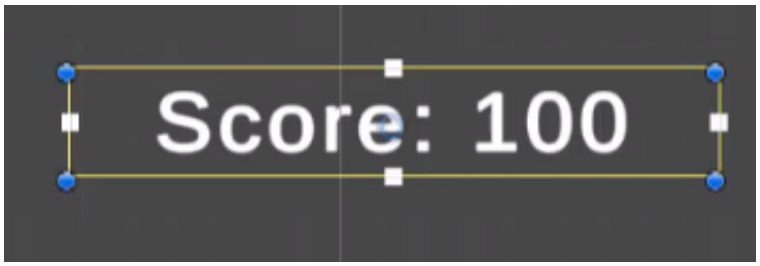
Creating Score UI

Lastly, we're going to add a **Text** to the Canvas GameObject for the player's **Score**.



Again, we can play around with the font settings. We changed the font style to bold, increased the font size to 100, set the alignment to be center:

© 2020 Zenva Pty Ltd. All rights reserved.



Modifying Anchor Point

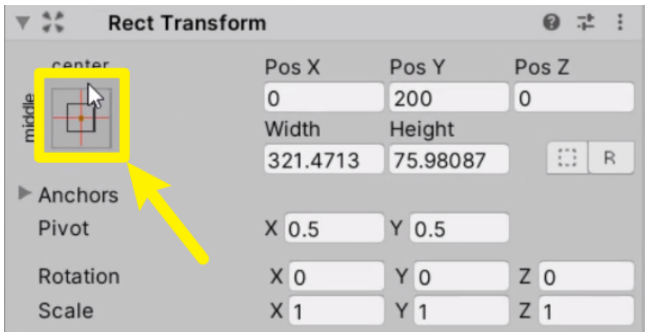
The **Anchor Handles** are represented by four triangles as shown below:

These handles ensure that our UI element is **positioned** and **scaled** appropriately to fit varying resolutions (aspect ratios).

The anchor handles of our text are currently positioned at the **center** of the screen.

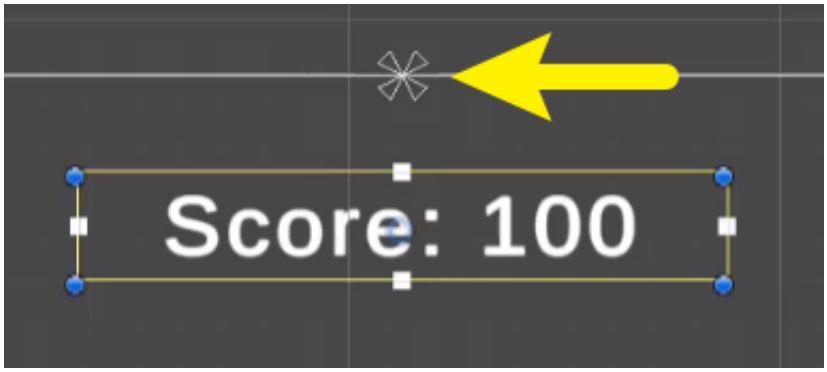
To change their position, you can click on this icon in the **Rect Transform** component.

© 2020 Zenva Pty Ltd. All rights reserved.

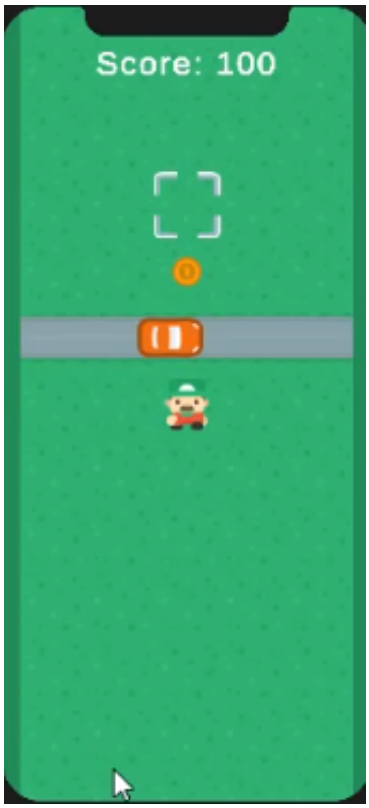


And then we can simply click on one of the presets to change our **anchor position**:

Now our anchoring is fixed at the **top center** of the screen, meaning that the text will appear at the top regardless of the screen resolution.



© 2020 Zenva Pty Ltd. All rights reserved.

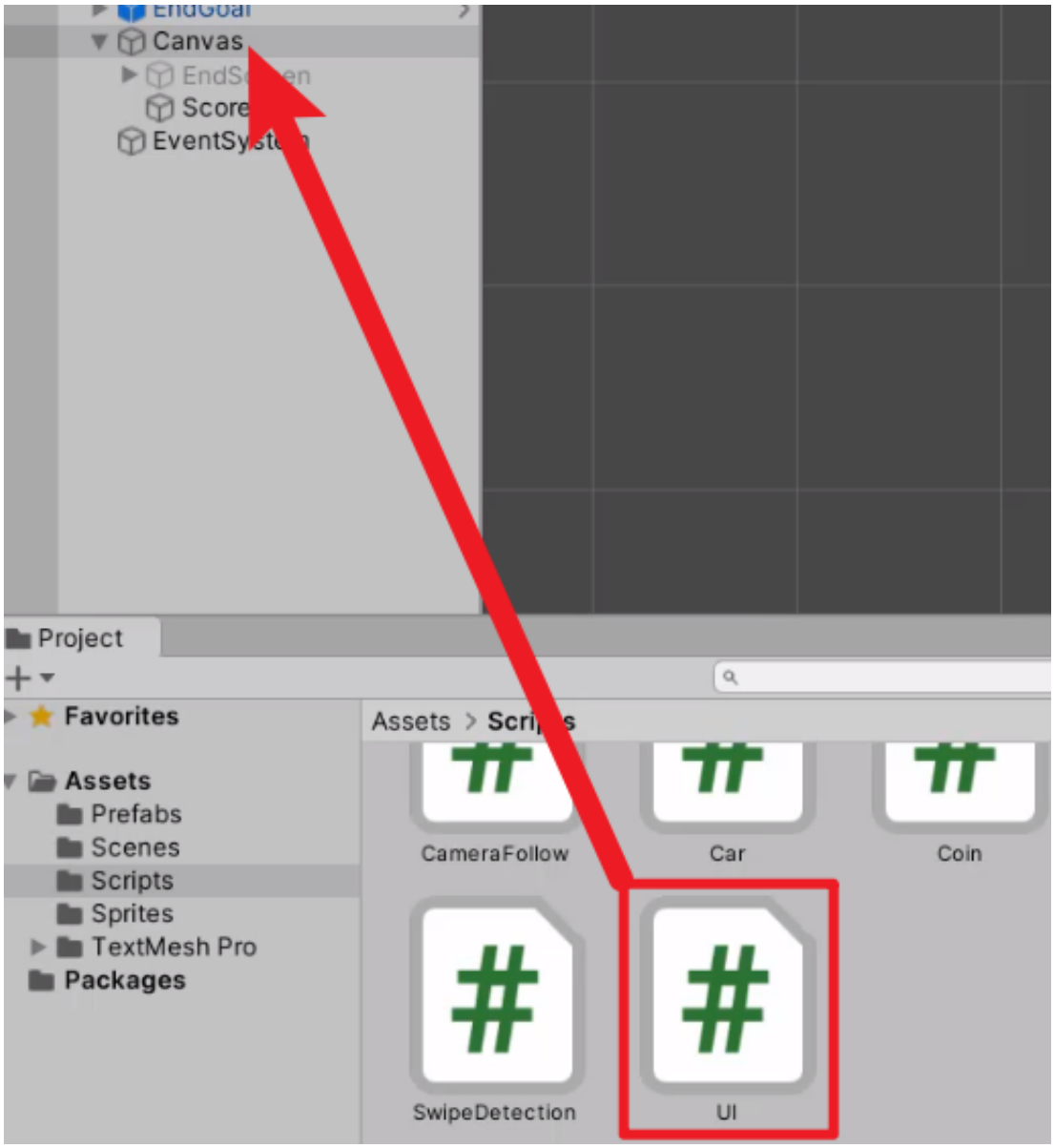


Creating UI Script

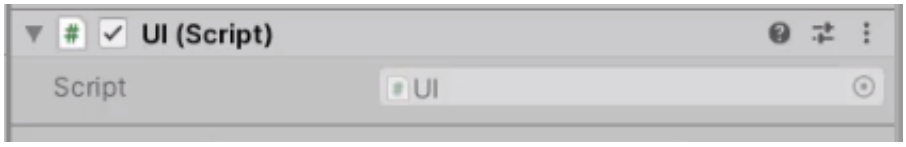
Let’s go ahead and create a new **C# script** called “UI”...

...and **attach** this to our Canvas.

© 2020 Zenva Pty Ltd. All rights reserved.



We're going to be using this script to update the **Score** text in the next lesson.



© 2020 Zenva Pty Ltd. All rights reserved.

In this lesson, we're going to be scripting our **UI** (User Interface) which we created in the previous lesson.

Let's go ahead and open up the "UI" script that we created in the previous lesson.

Creating Variables

First, we need to import the TextMesh Pro library (**TMPro**) at the top of our script. This allows us to use the **TextMeshPro** component:

```
using TMPro;
```

We can now declare a **TextMeshProUGUI** variable to reference our score text.

```
public TextMeshProUGUI scoreText;
```

We also need to declare **public GameObjects** to reference our **end screen**, **win text,** and **game over text**.

```
public GameObject endScreen;
public GameObject winText;
public GameObject gameOverText;
```

Lastly, we're going to make a **static** UI instance for the **Singleton** pattern. This ensures that there is only one instance of UI canvas that is globally accessible and available at all times.

```
public static UI instance;
```

Then inside of the **Awake** function, we can assign this class to the one and only instance.

```
private void Awake()
```

© 2020 Zenva Pty Ltd. All rights reserved.

```
{
    instance = this;
}
```

Updating UI components

Now that our variables are all set up, let's create a new function called "**UpdateScoreText**", that takes in an integer variable for the score to set.

```
public void UpdateScoreText (int score)
{

}
```

This function will update our **score text** (string) to be the same as the score (integer). Note that we need to convert the score (integer) to be string by using **ToString()**:

```
public void UpdateScoreText (int score)
{
    scoreText.text = score.ToString();
}
```

After this, we need to basically have a function that gets called once the game has **ended**. The ending state will be determined by the value of **didWin**, which will return either true or false.

```
public void SetEndScreen(bool didWin)
{
    //Enable the end screen
    endScreen.SetActive(true);
    //Enable the win text if didWin is true
    winText.SetActive(didWin);
    //Enable the gameover text if didWin is false
    gameOverText.SetActive(!didWin);
}
```

Finally, we can create a function for **restarting** the game. This function will get called when we press the restart button on our end screen.

```
public void OnRestartButton()
{
    //Re-load the current scene
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

And remember, in order to use **SceneManager** we need to import the **SceneManagement** library.

© 2020 Zenva Pty Ltd. All rights reserved.

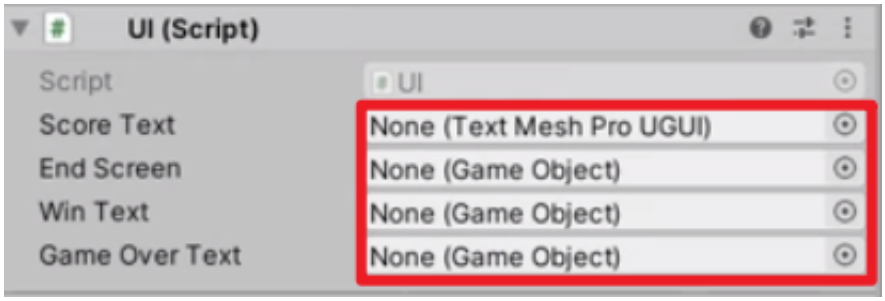
```
using UnityEngine.SceneManagement;
```

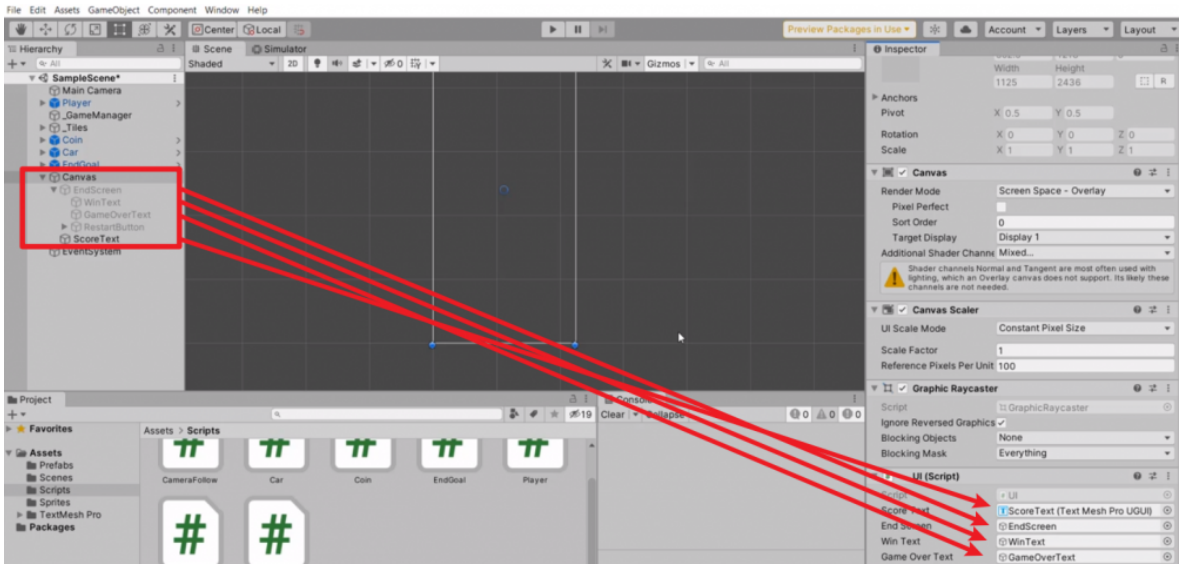
Now, let's go over to our **Player** script and replace **Debug.Log** with our **UI.instance**:

```
public void AddScore(int amount)
{
    score += amount;
    UI.instance.UpdateScoreText(score);
}

public void GameOver ()
{
    UI.instance.SetEndScreen(false);
}

public void Win ()
{
    UI.instance.SetEndScreen(true);
}
```

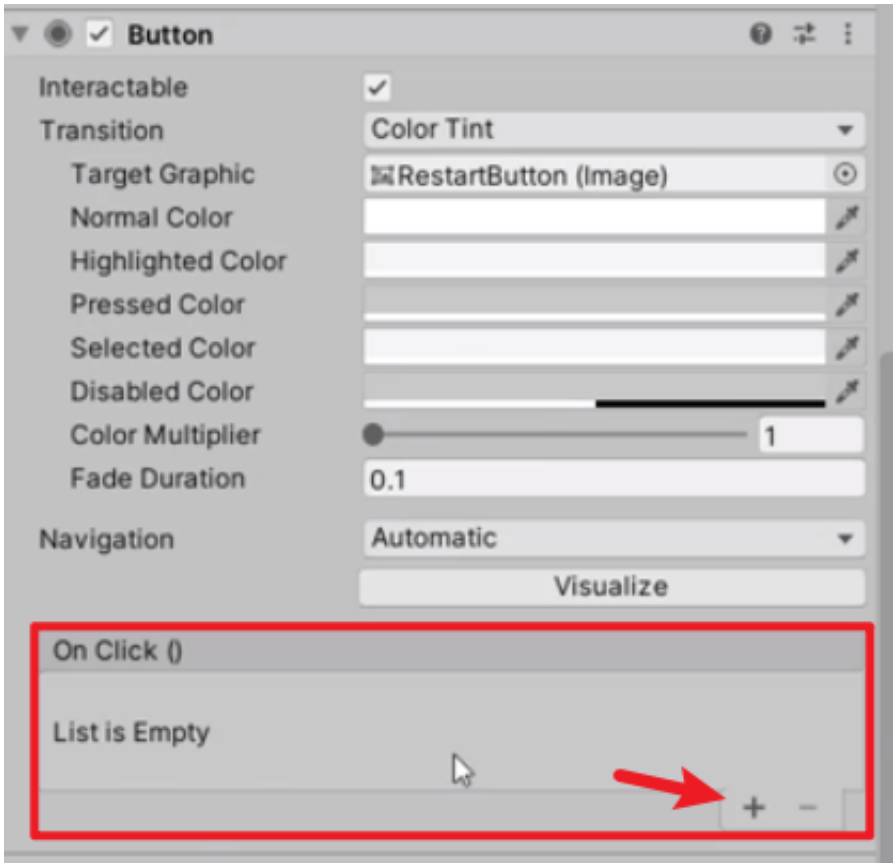
Referencing GameObjects

After saving the script, let's drag relevant objects into these empty fields of our UI script.

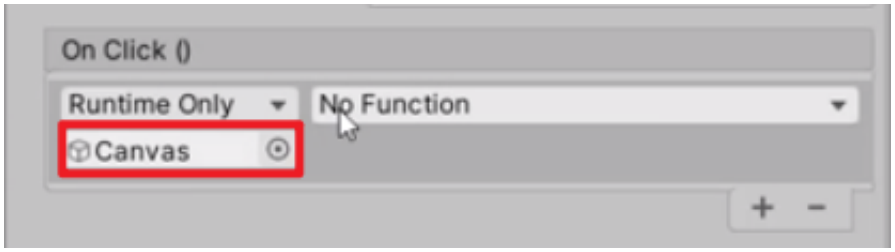




Now, for the **RestartButton**, we can select that and scroll down to the **Button** component where it says **OnClick**. This is a list of functions that get called when the button is clicked.

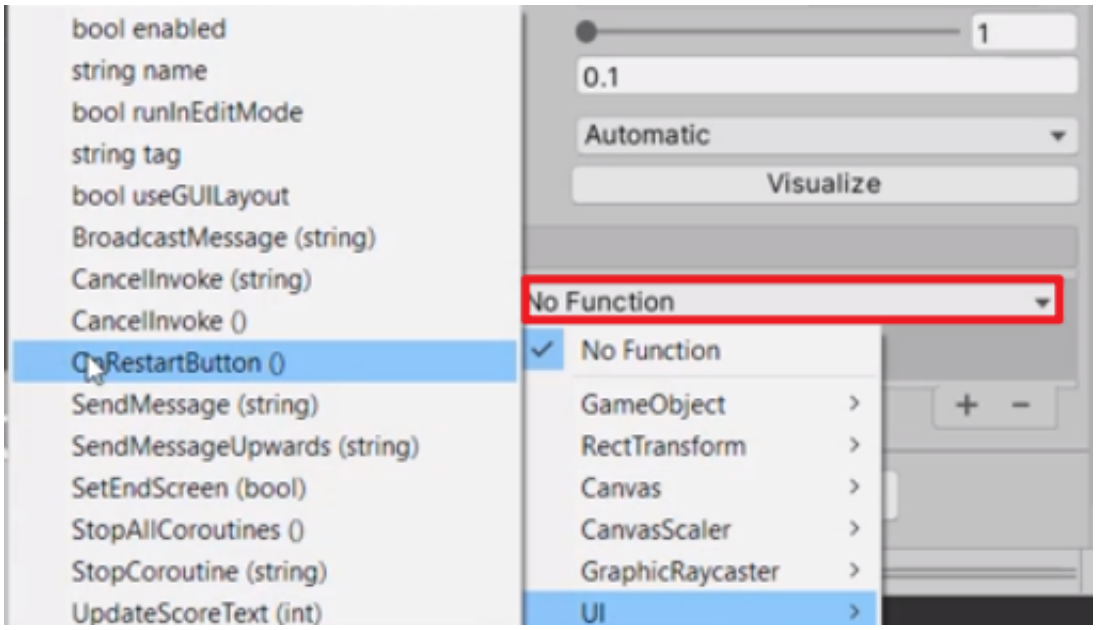


We can **add** a new event by clicking the **+ icon** and drag in our **Canvas** as the object,



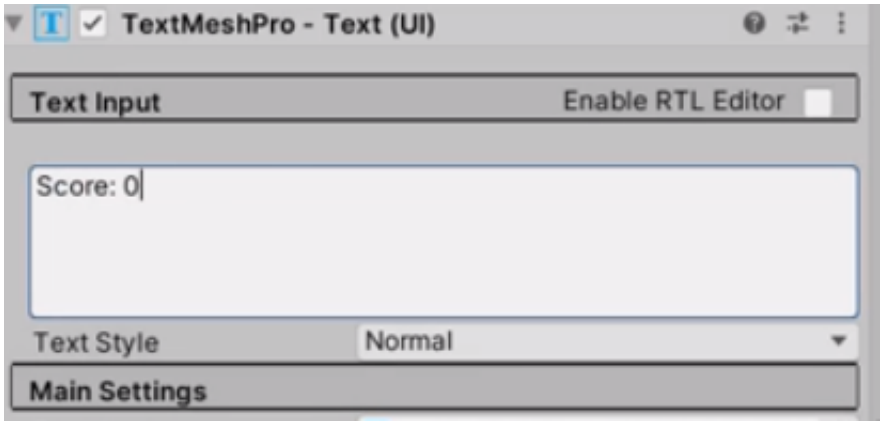
Then we can set the function to be **UI > OnRestartButton()**.

© 2020 Zenva Pty Ltd. All rights reserved.



So when we click the restart button, it's going to call this **OnRestartButton()** function inside of the UI script attached to the **Canvas**.

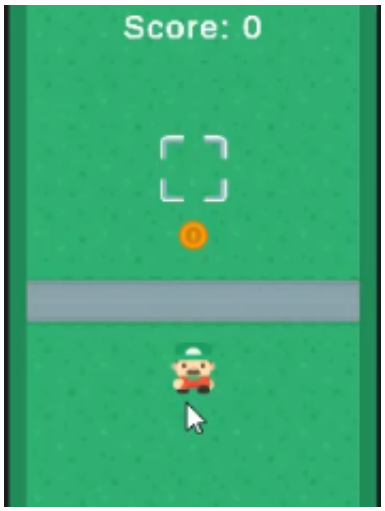
We're going to set the **default** value of our **score text** to be **0**,



and now we're ready to test out the game.

Testing The Game

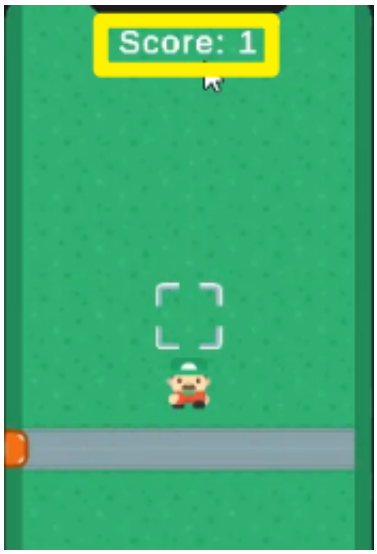
© 2020 Zenva Pty Ltd. All rights reserved.



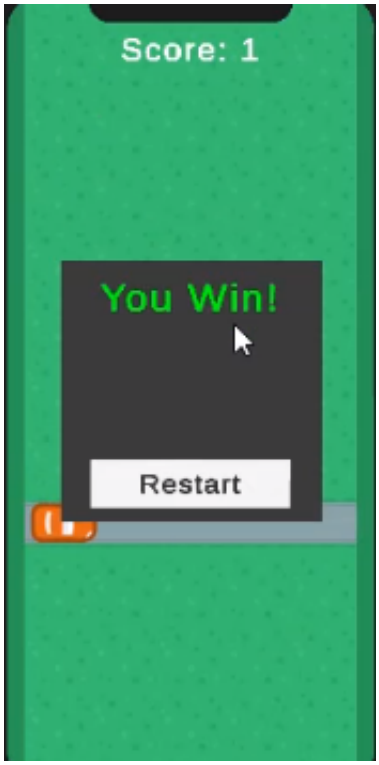
We should see that the **Game Over** text shows up when we get hit by the car,

...and the **score** goes up as we pick up a coin.

© 2020 Zenva Pty Ltd. All rights reserved.

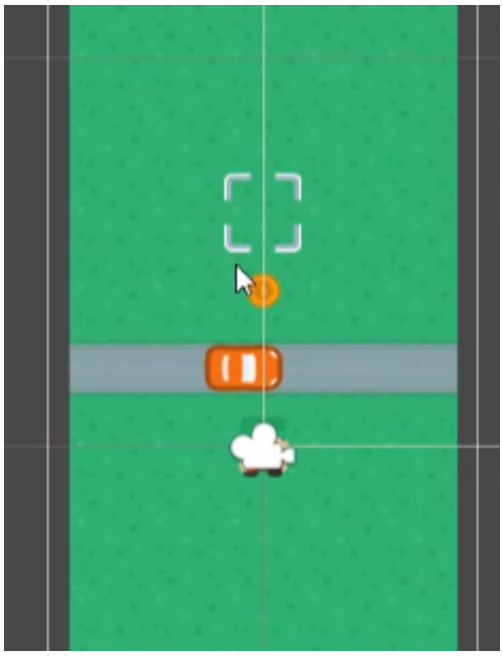


Finally, when we get to the **end goal**, you should be able to see this win text.



© 2020 Zenva Pty Ltd. All rights reserved.

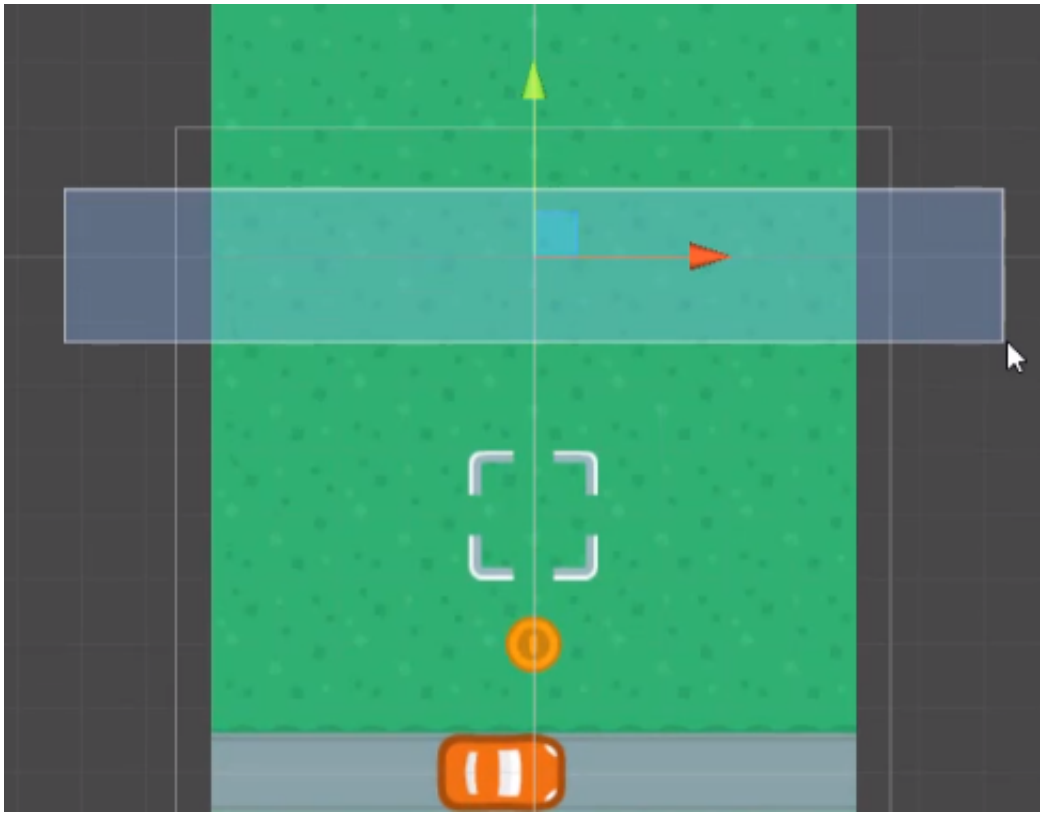
In this lesson, we're going to be finishing off our game by **building our level**.



Making Road Tiles

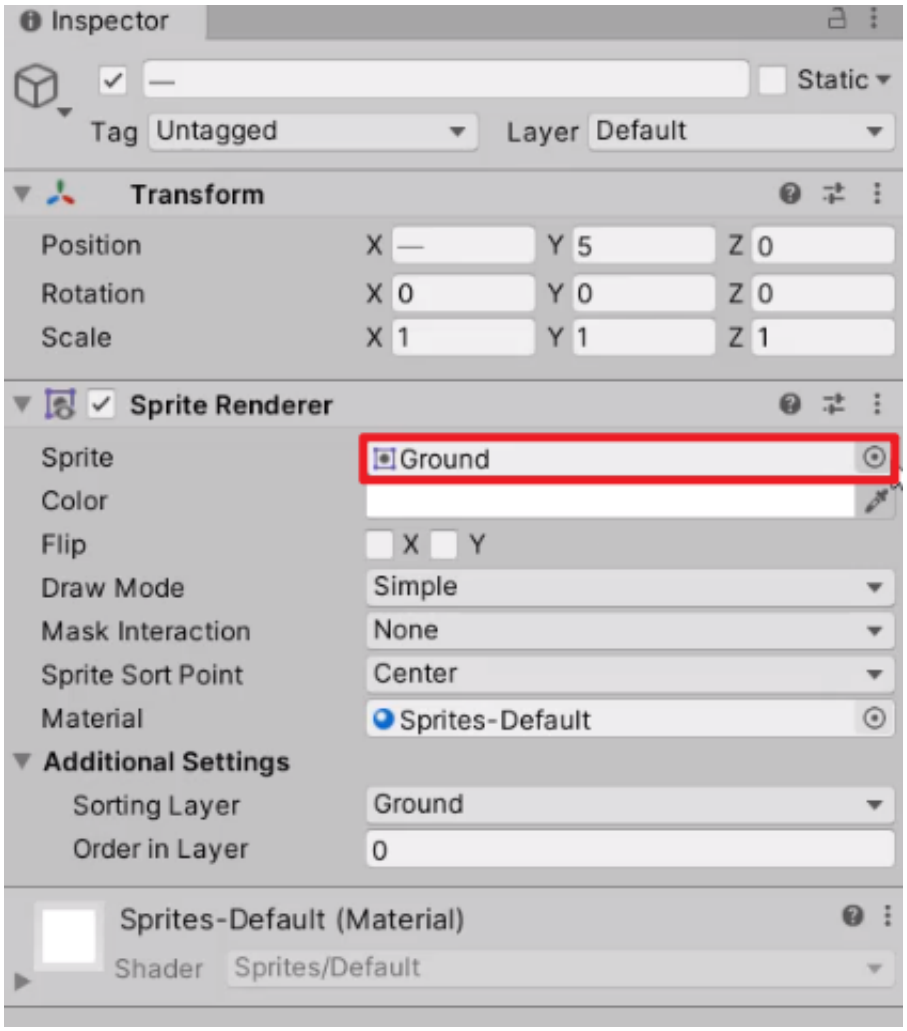
Let's start off by figuring out where all of our different cars to go– and then start filling in the road tiles.

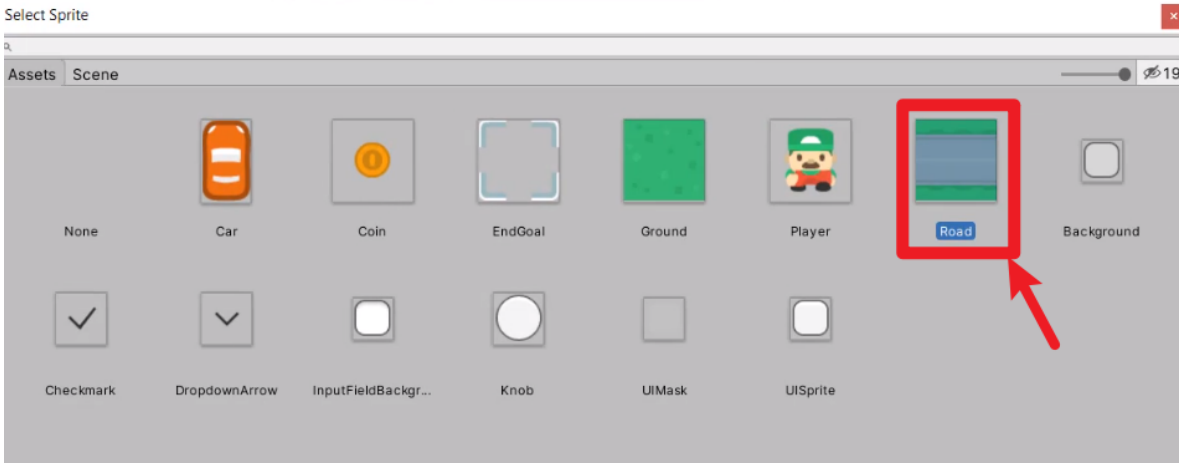
We're going to select these tiles to turn them into **Roads**:



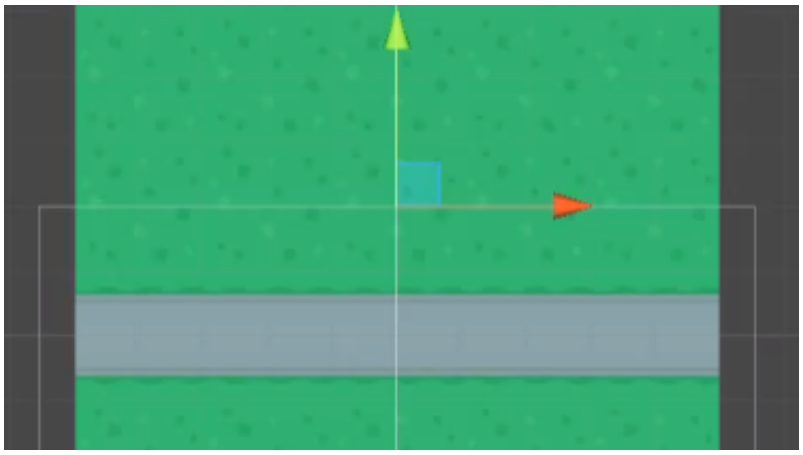
© 2020 Zenva Pty Ltd. All rights reserved.

To do that, we need to go to the **Sprite Renderer** and change the sprite from ‘Ground’ to ‘Road’:

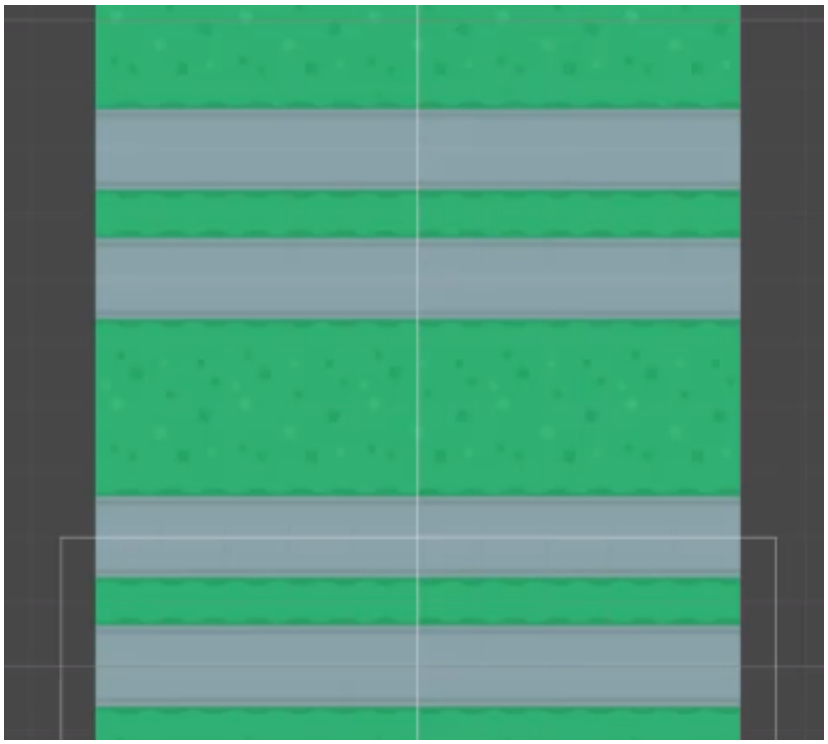




© 2020 Zenva Pty Ltd. All rights reserved.



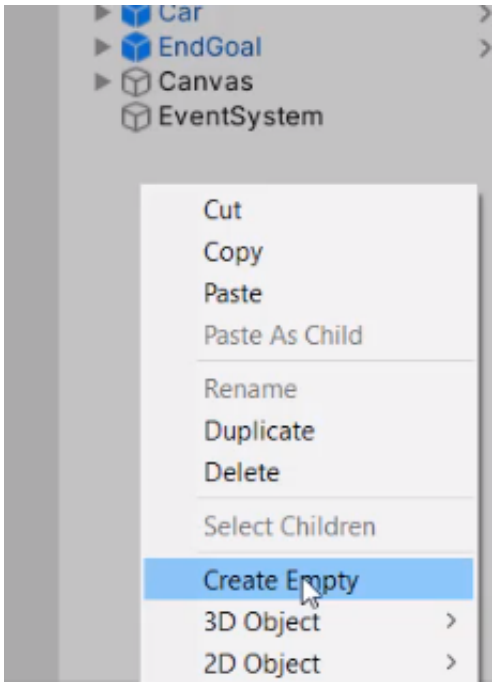
We're going to repeat this step with the overall road layout in mind.

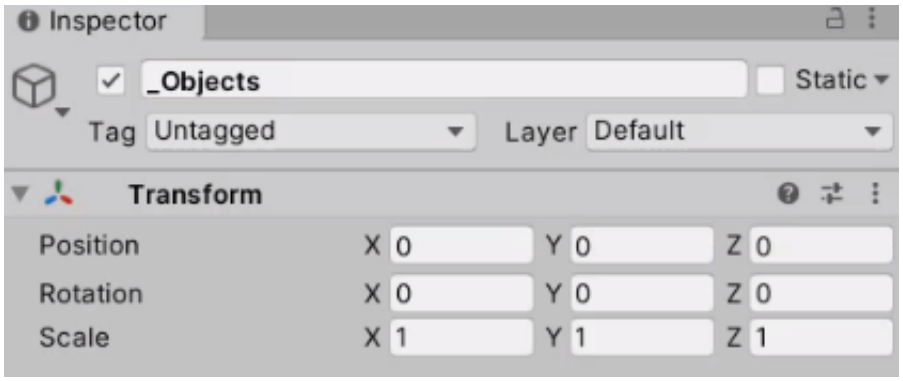


Adding Cars

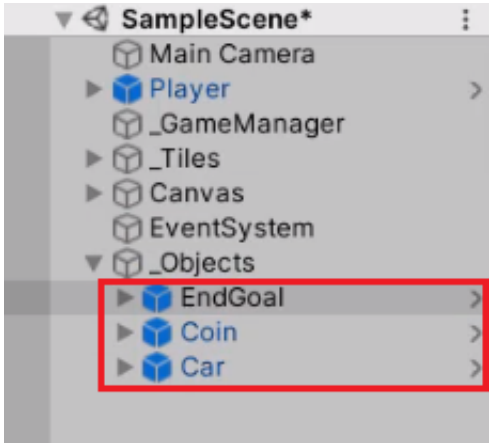
Now let's create a new **empty** gameObject and call it **_Objects**,

© 2020 Zenva Pty Ltd. All rights reserved.



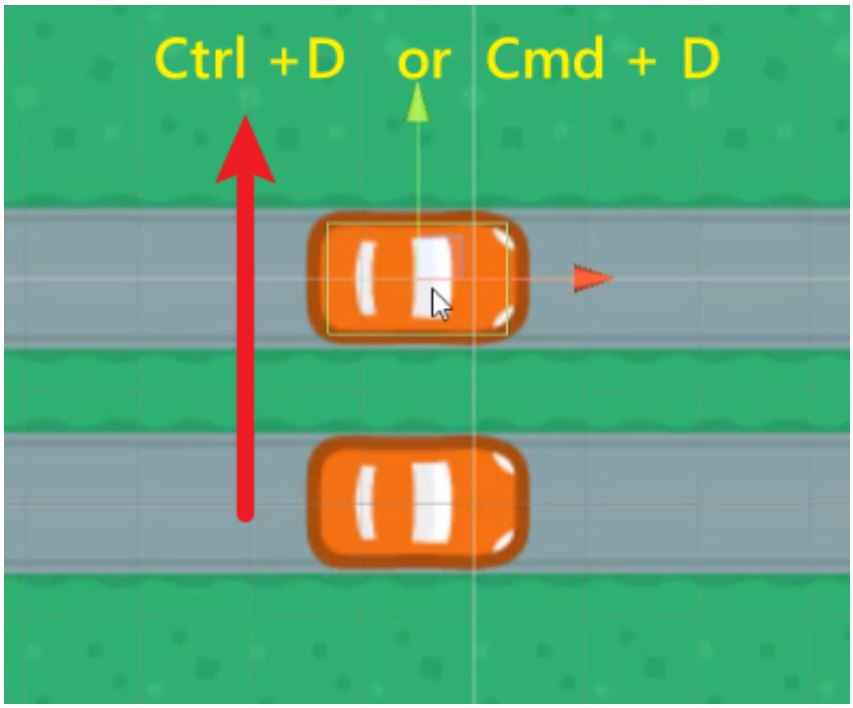


This is where we are going to be storing game objects, such as our cars and coins.

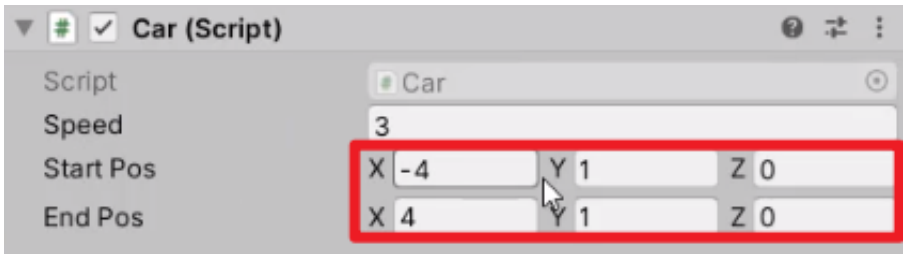


Let’s copy and paste the **Car** GameObject and drag it up to each road.

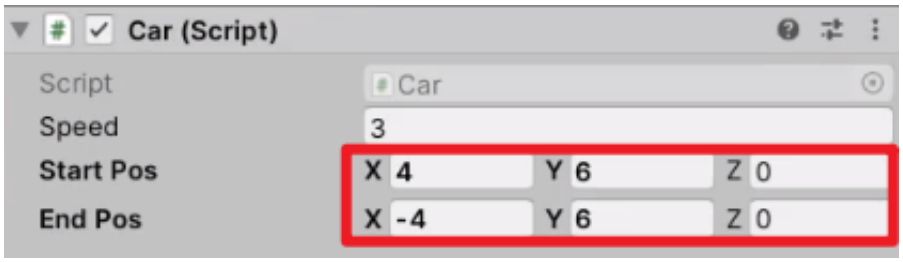
© 2020 Zenva Pty Ltd. All rights reserved.



Make sure to change the **startPos** and **endPos** since these are going to be at different positions along the **y-axis**.

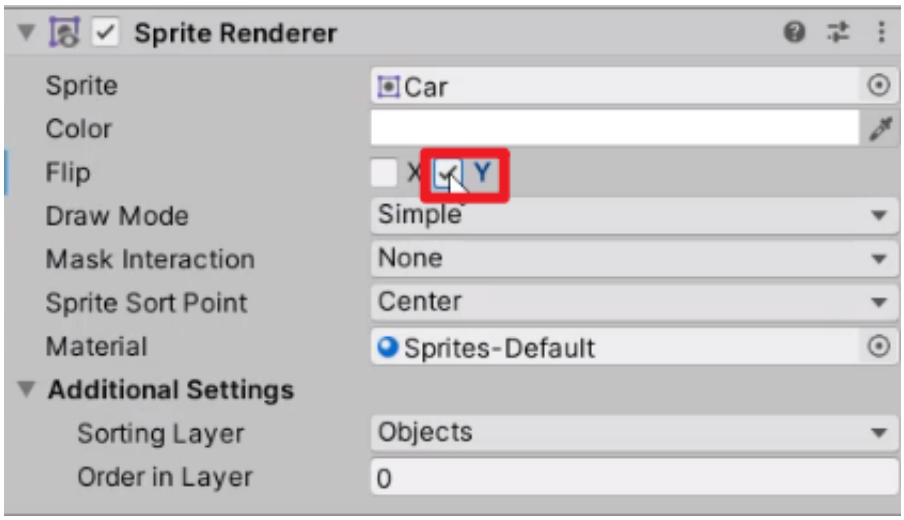


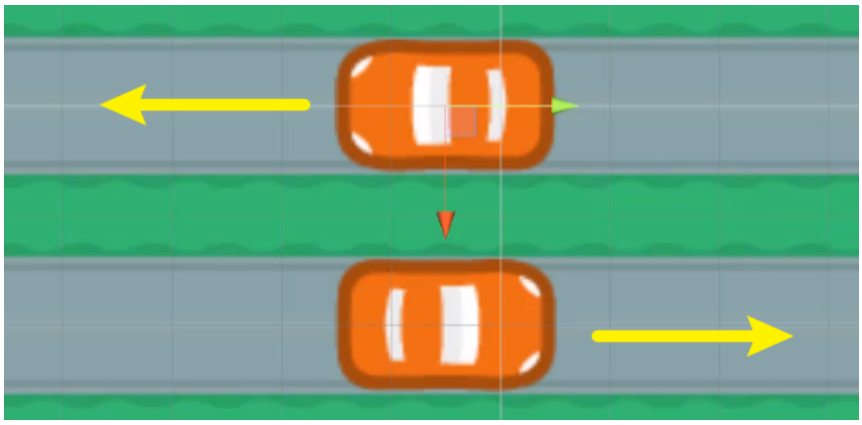
We also want to make it move in the **opposite** direction. To do that, we can simply switch over the **startPos** and **endPos** on the **x-axis**:



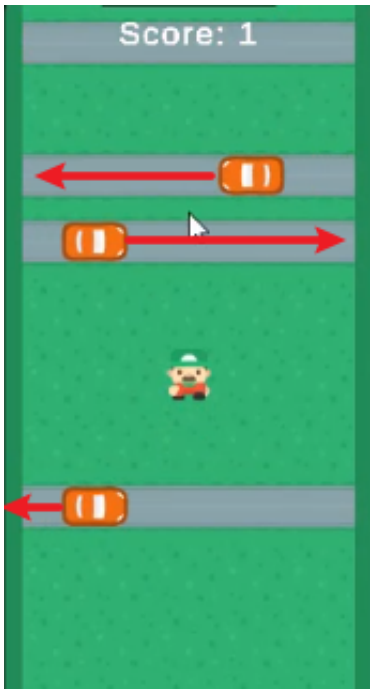
And we need to select the **Sprite** and enable **Flip Y** so the car faces the opposite direction.

© 2020 Zenva Pty Ltd. All rights reserved.



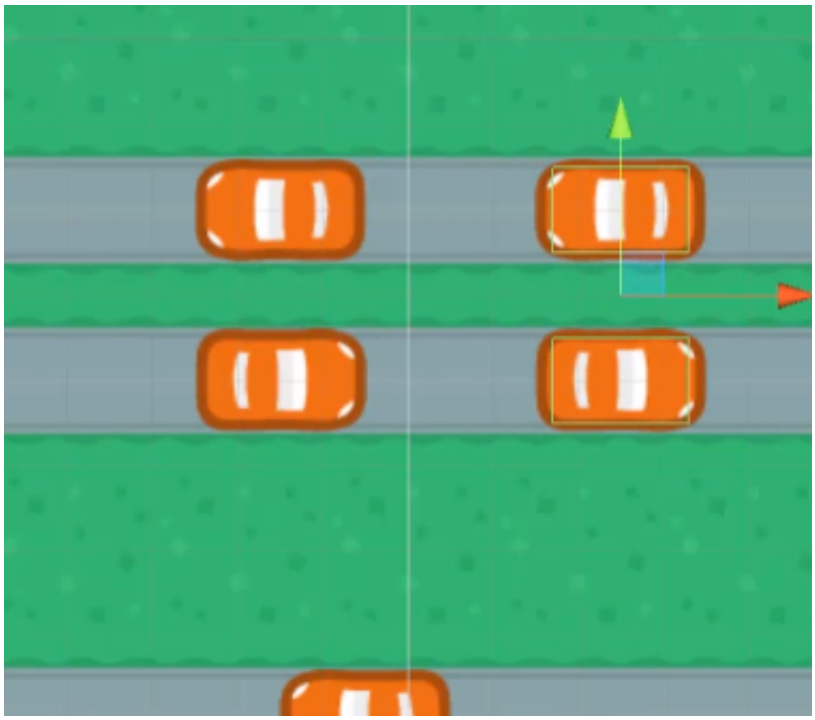


If you press **Play**, you will see that the cars are moving in different directions.

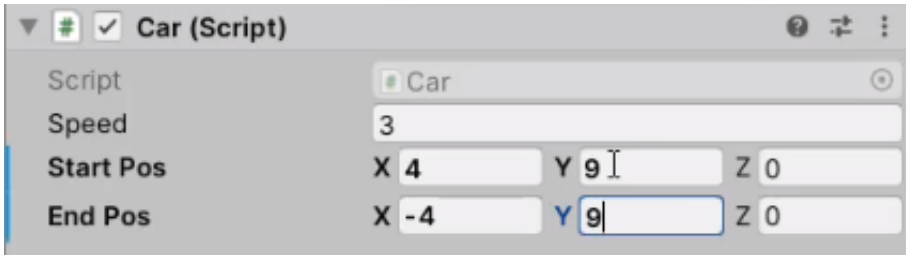


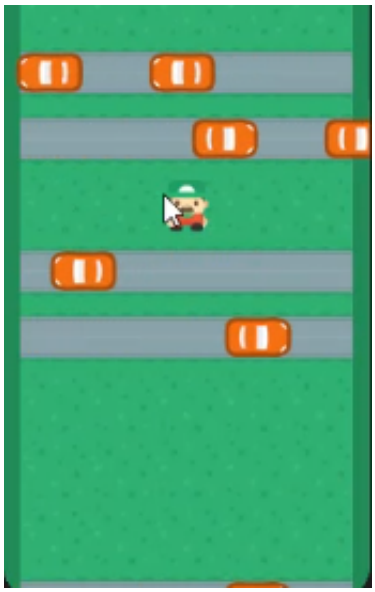
Let's make it a bit more difficult by adding in **multiple cars per road**. So to do that, we're going to copy and paste them up at the top:

© 2020 Zenva Pty Ltd. All rights reserved.



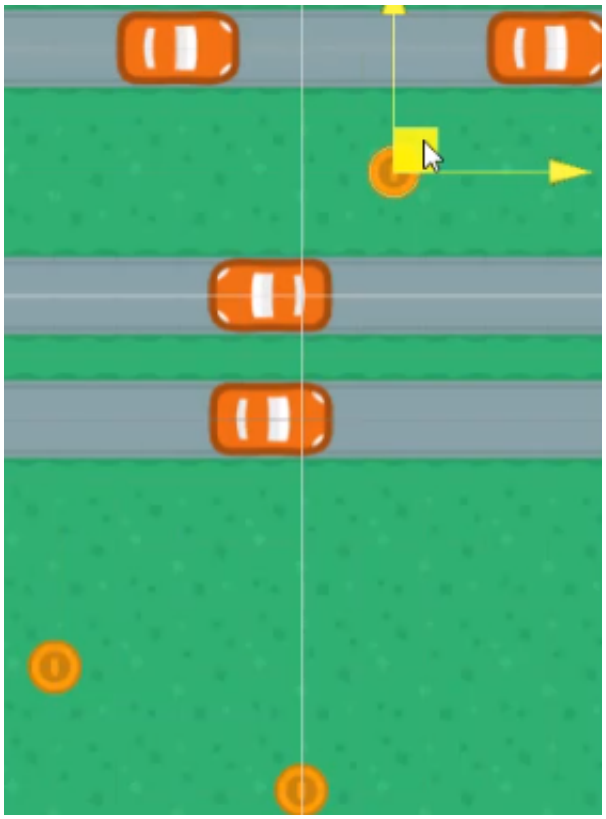
And we just need to make sure that the **startPos** and **endPos** have their **y-positions** the same as what the object is by default.





We just need to fill up the rest of the roads with cars and place some coins– this part is entirely up to you. Feel free to come up with your own creative level design.

© 2020 Zenva Pty Ltd. All rights reserved.



There we go, this is our mobile road crossing game. Congratulations on completing yet another course!

© 2020 Zenva Pty Ltd. All rights reserved.