CODE NINJAS®

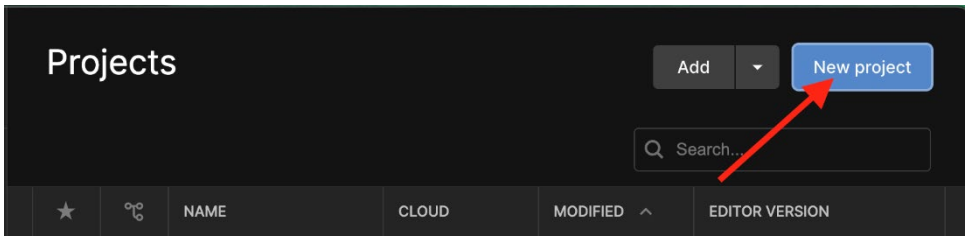
Bronze Belt Ninja Guide

Activity 04: Sketch Head

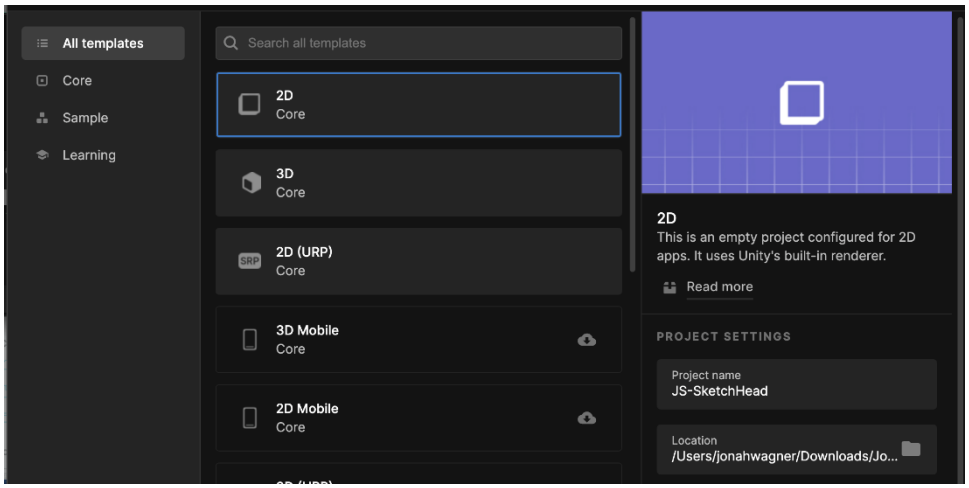
Activity 4: Sketch Head

In this activity, we will continue using physics and colliders, only this time we will add more features to the game, such as a "camera follow" mechanic and a generator object!

1 Open the Unity Hub application on your computer. Select the blue **New Project** button in the right-hand corner.

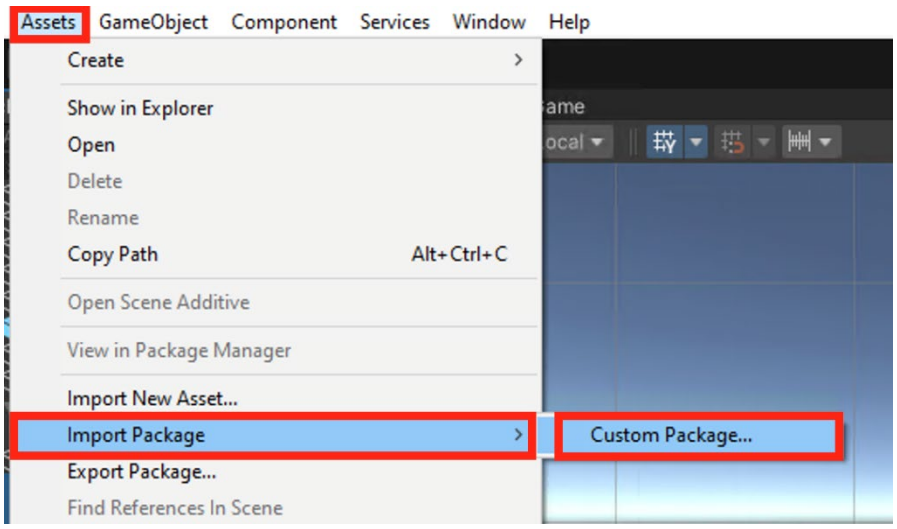


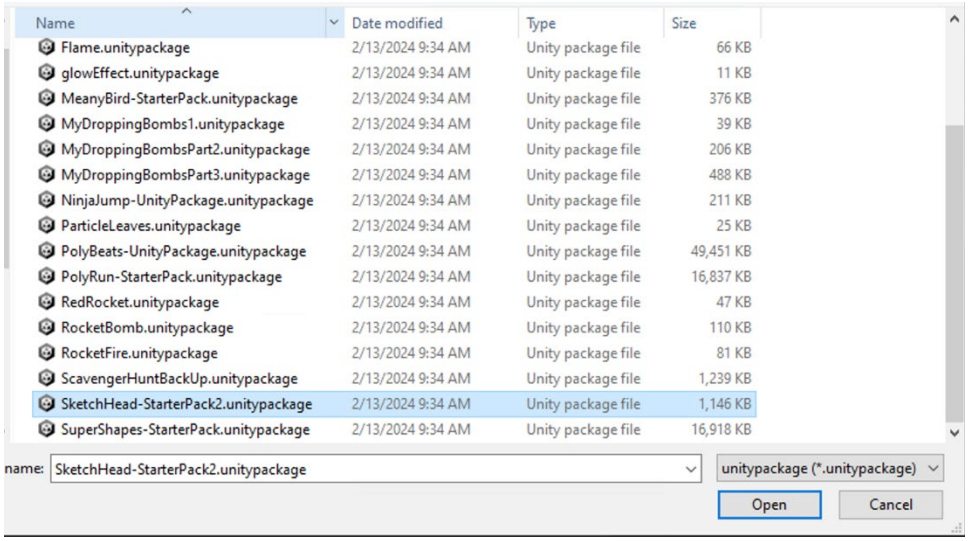
2 In the templates column select **2D.** Next, type your first and last initials and the name of the project. For example, John Smith would save the project as JS-SketchHead. Select the folder location where you typically save your Unity files.



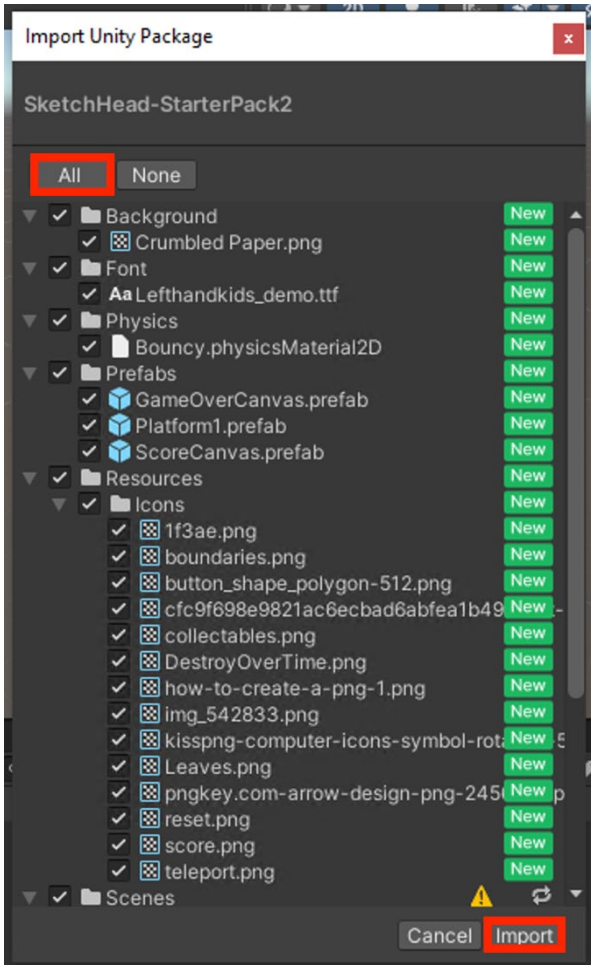
3 As soon as Unity has loaded all the necessary assets into the program, it will open. As you can see, we don't have assets or scripts in the project. Let's fix that now.

Go to the **Assets** menu. Select **Import Package** and click **Custom Package**. This allows us to import any package that has been compressed in Unity. In the folder with the Unity Assets, locate and open the **Activity 04 – SketchHead - StarterPack** Unity package. Once you do this, a menu will open displaying everything inside the package.





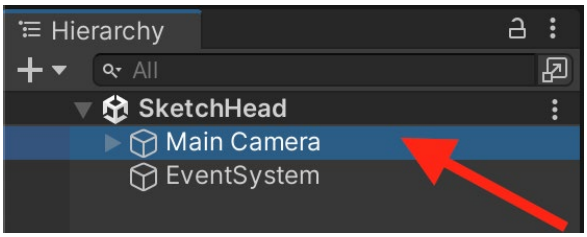
4 Since these packages are specially made for each game, we want all materials inside the Unity package selected. If they are not all selected, click the **All** button. Click **Import** and everything will be imported into the project.



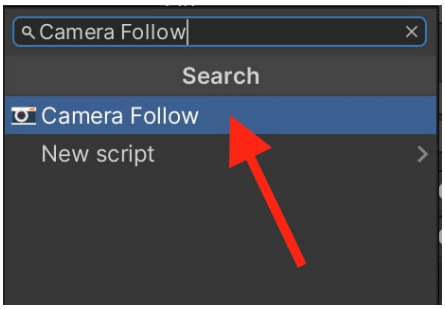
5 Now click **Scenes** under the **Assets** window and open the **SketchHead** scene by double-clicking. Like the last game, in the Game window, ensure Free Aspect is selected then click the plus and change the resolution to **600x800**.

6 Let's start configuring the Main Camera game object before we start to configure our player object. We need to select the **Main Camera** in the Hierarchy. This step is needed because the camera will focus on the player by default.

Select the **Main Camera**, go to the Inspector window and click **Add Component**.



7 Once you click **Add Component**, type ***Camera Follow*** in the search box. Select the **Camera Follow** script.



8 Open the script and add the following:

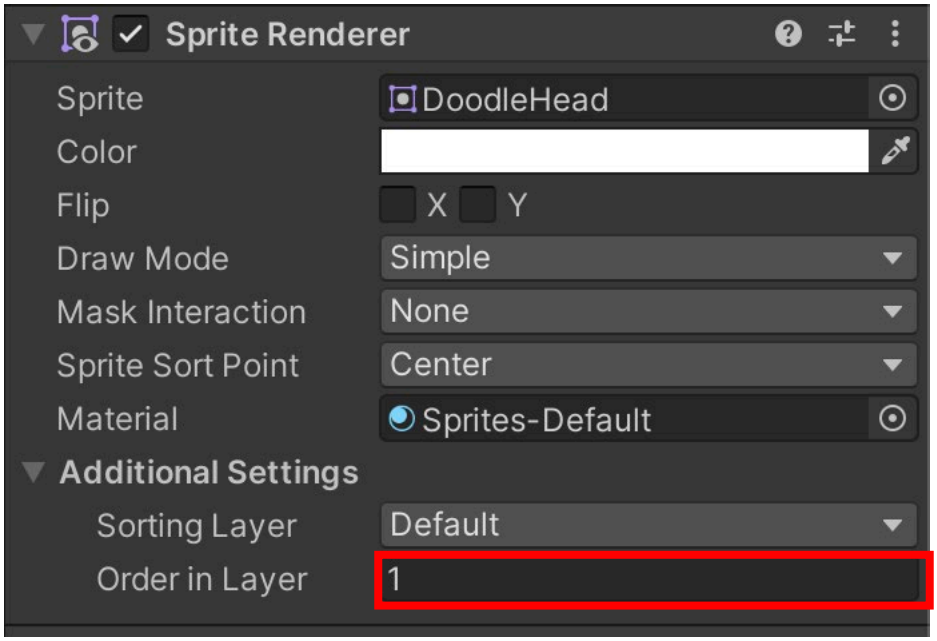
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    //Target object position
    [Header ("Target Object")]
    public Transform target;
    // Start is called before the first frame update
    void Start()
    {

    }
```

Next type inside the **void Update** function,

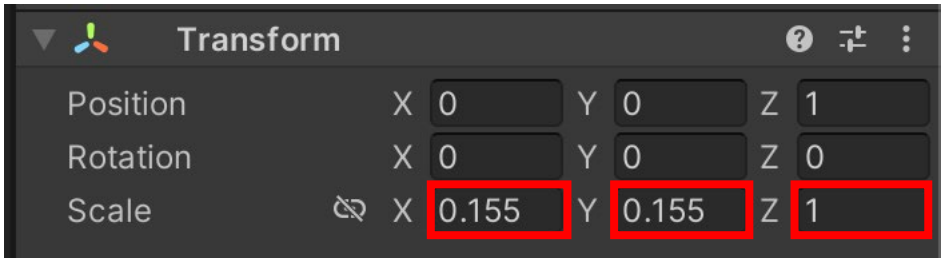
```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    if (target.position.y > transform.position.y)
    {
        transform.position = new Vector3(0, target.position.y, -10f);
    }
}
```

You can delete the void Start function as it is not needed in this activity. Then save the script and return to Unity!

9 Now, go to the **Sprites** folder and select the **DoodleHead** sprite. Drag the **DoodleHead** sprite into the **Hierarchy**. It should now be shown in the Game view. If not, change the order in layer from **0** to **1** in the **Sprite Renderer** component.

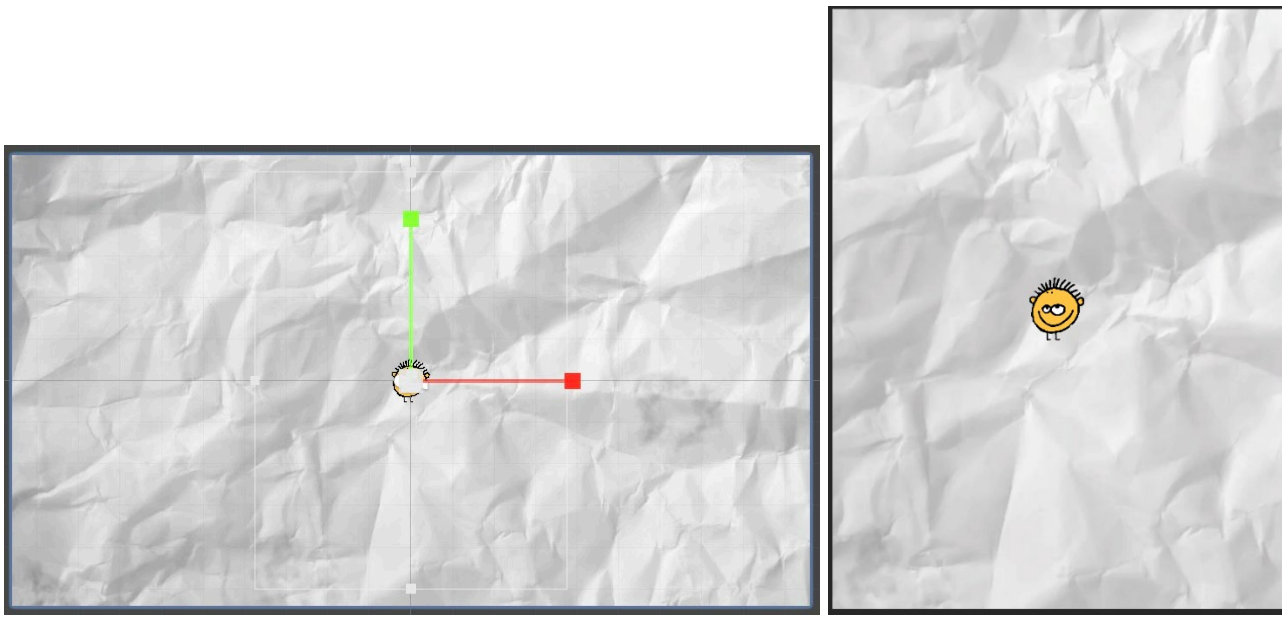


10 Now the DoodleHead is too big for the Game view, so let's change its size. In the Inspector window, with the **DoodleHead** still selected, change the size of the DoodleHead to these scale values:

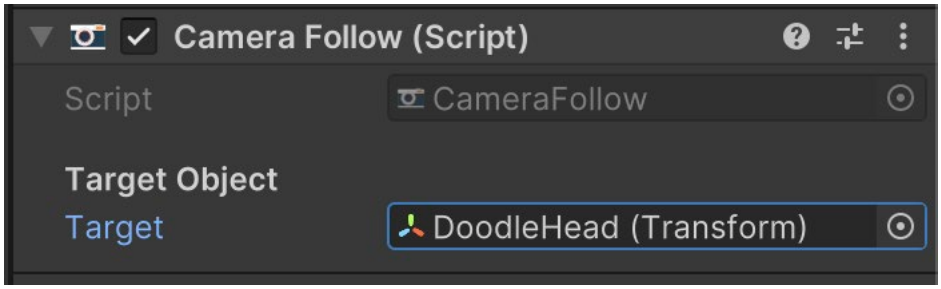


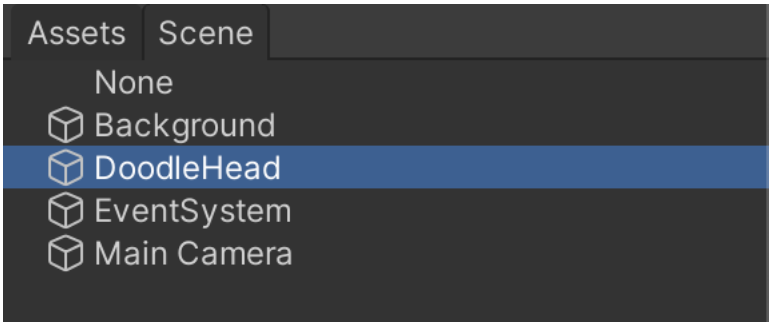
Scale values: X: 0.155, Y: 0.155, Z: 1

Your DoodleHead should look like this in both the Scene and Game windows:

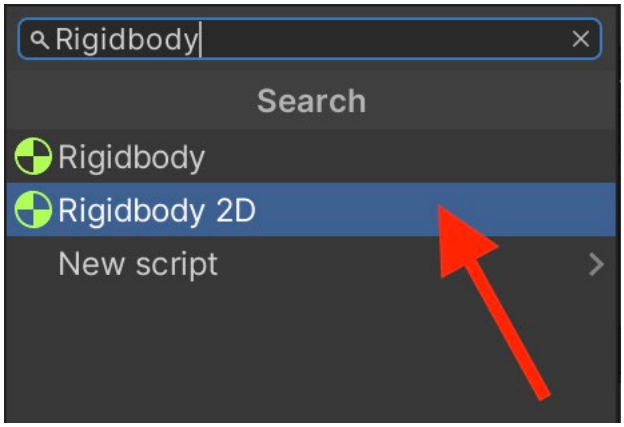


11 Select the **Main Camera** object, where we will select our target object in the Camera Follow script component. Select the **Properties** button to the right of the text box and in the Scene menu, select **DoodleHead**.

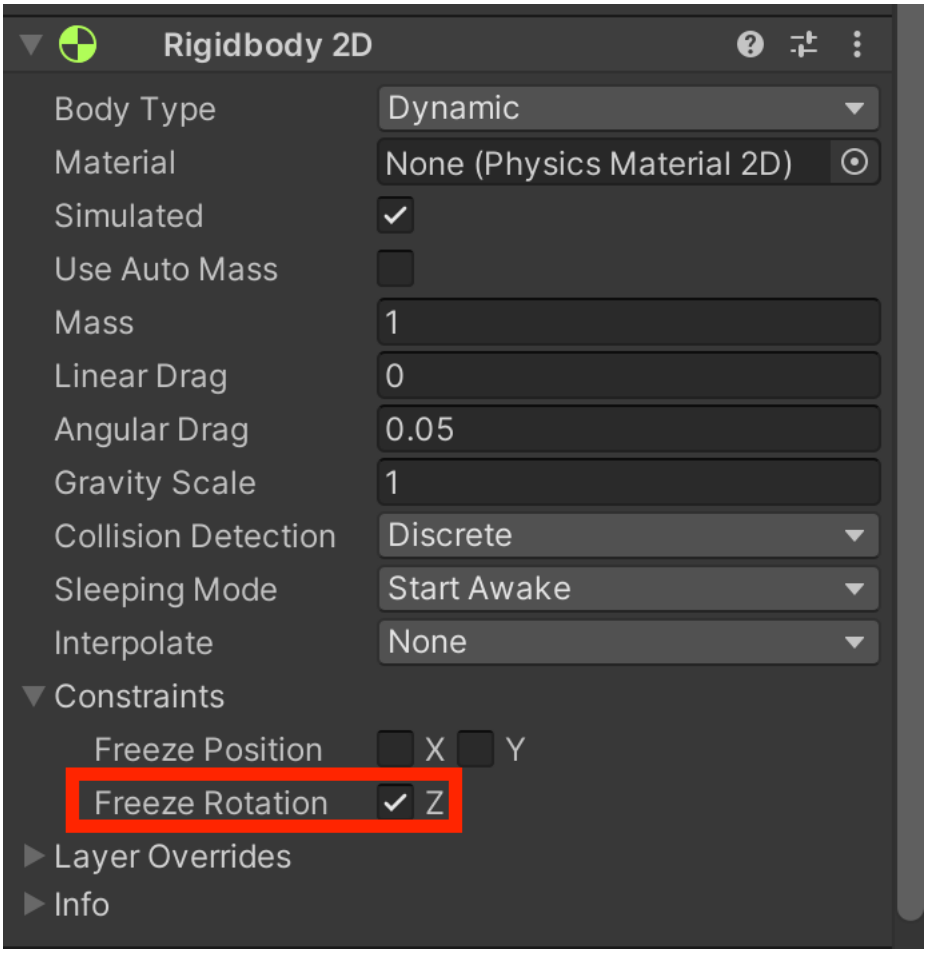




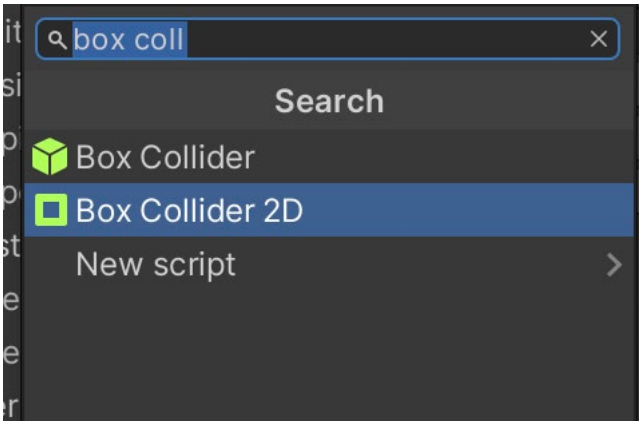
12 Now, select the **DoodleHead** object and click **Add Component**. Type ***Rigidbody*** in the search box and select the **Rigidbody 2D** component.

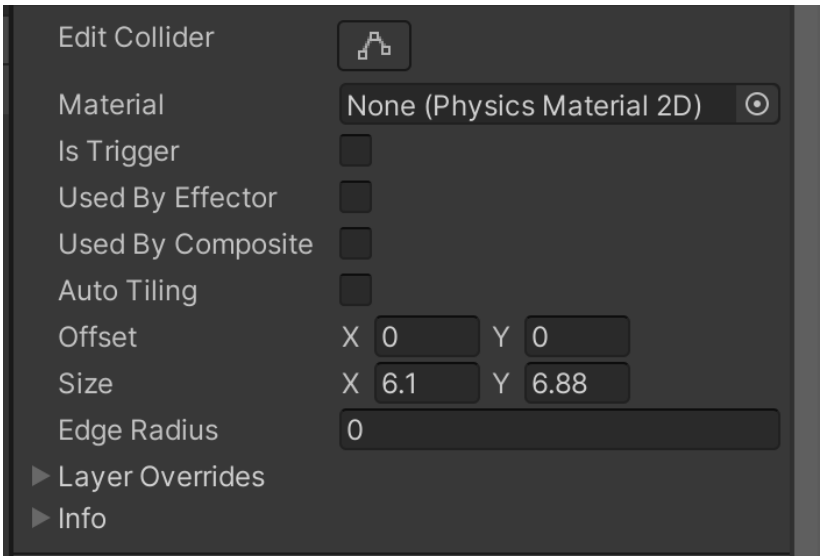


13 Select **Constraints** in the **Rigidbody2D** component. Then, select the check box labeled **Freeze Rotation** for the Z axis. Constraints are the rules or limitations applied to the movement and interaction of objects that have a Rigidbody component.

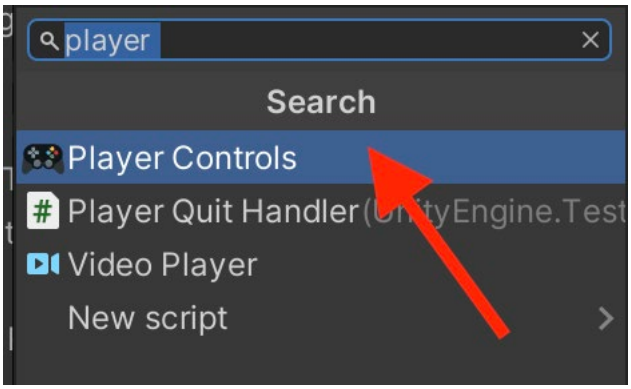


14 Afterward, click Add Component with the DoodleHead still selected. Then type Box Collider 2D in the search box and select Box Collider 2D. Match the settings to the image provided below.





15 Ensure that the **DoodleHead** is still selected, then select the **Add Component** button in the Inspector menu. Then, type ***Player*** in the search bar and select **Player Controls**.



16 Open the Player Controls script and add the following code inside the class below the void Start function:

```
public class PlayerControls : MonoBehaviour
{
    [Header("Rigidbody Settings")]
    [Tooltip("Rigidbody2D object that is stored")]
    public Rigidbody2D rb;

    [Tooltip("Downward speed of the object")]
    public float downSpeed = 20f;

    [Header("Movement Settings")]
    [Tooltip("Movement speed of the object")]
    public float movementSpeed = 10f;

    [Tooltip("Movement direction of the object")]
    public float movement = 0f;

    private SpriteRenderer spriteRenderer;

    // Start is called before the first frame update
    void Start()
```

Next, type inside the **void Start** function:

```
// Start is called before the first frame update
void Start()
{
    // Store the components in variables
    rb = GetComponent<Rigidbody2D>();
    spriteRenderer = GetComponent<SpriteRenderer>();
}
```

Next, type inside the **void Update** function:

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //Get the horizontal input and calculate the movement direction
    movement = Input.GetAxis("Horizontal") * movementSpeed;

    //Flip the sprite based on movement direction
    spriteRenderer.flipX = movement < 0;
}
```

Create a new **void FixedUpdate** function and add this:

```
void FixedUpdate()
{
    //Vector2 which is (x,y) velocity
    //equals to the velocity of the rigidbody2D
    Vector2 velocity = rb. velocity;
    //Velocity of the x axis equals to
    //the direction movement on the x axis
    // of the character.
    velocity.x = movement;
    //Rigidbody2D velocity equals to
    //velocity of the object
    rb.velocity = velocity;
}
```

Create a new **void OnCollisionEnter2D** function and add this:

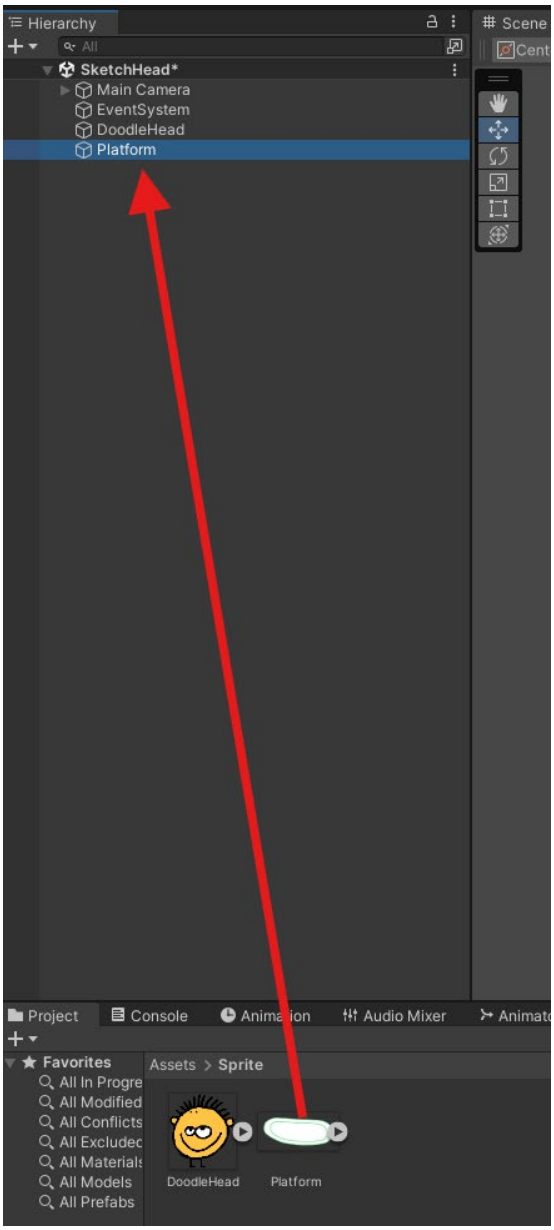
```
private void OnCollisionEnter2D(Collision2D collision)
{
    // If the object collides with something and is moving downwards,
    // adjust the y-component of its velocity
    if (rb.velocity.y <= 0)
    {
        rb.velocity = new Vector2(rb.velocity.x, downSpeed);
    }
}
```

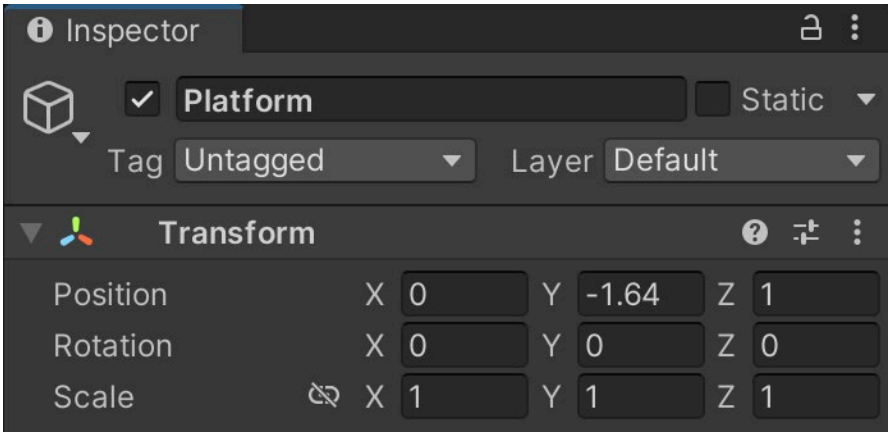
You can now save the script and return to Unity!

17 In the **Player Controls** component, click the **Properties** button next to the **Rigidbody** input field.

18 Select the **DoodleHead** object.

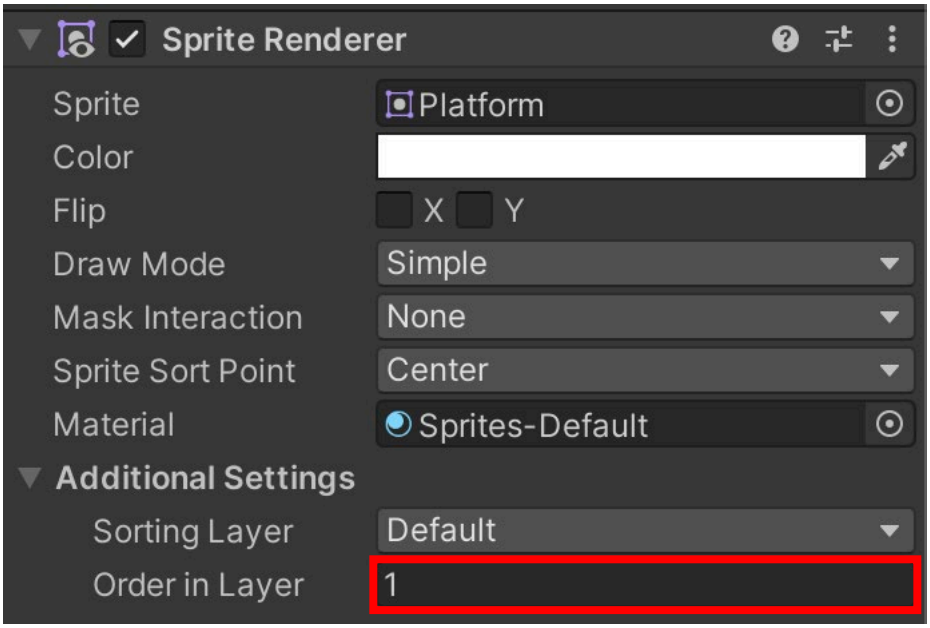
19 Open the **Sprites** folder, then drag the platform to the **Hierarchy**. Move the platform under the **DoodleHead**.



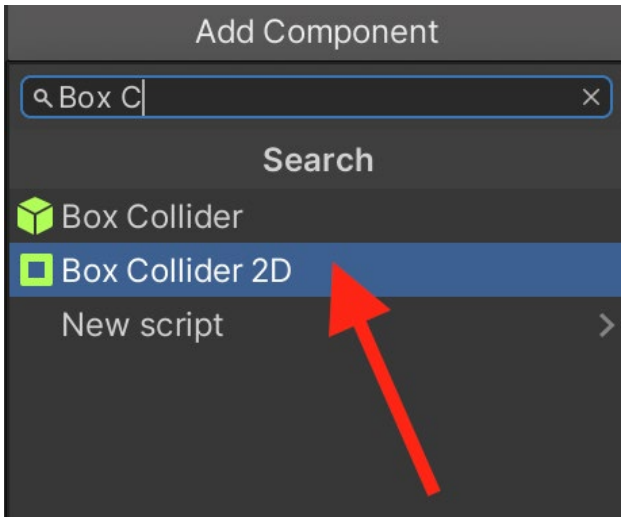


Position values: X: 0, Y: -1.64, Z: 1

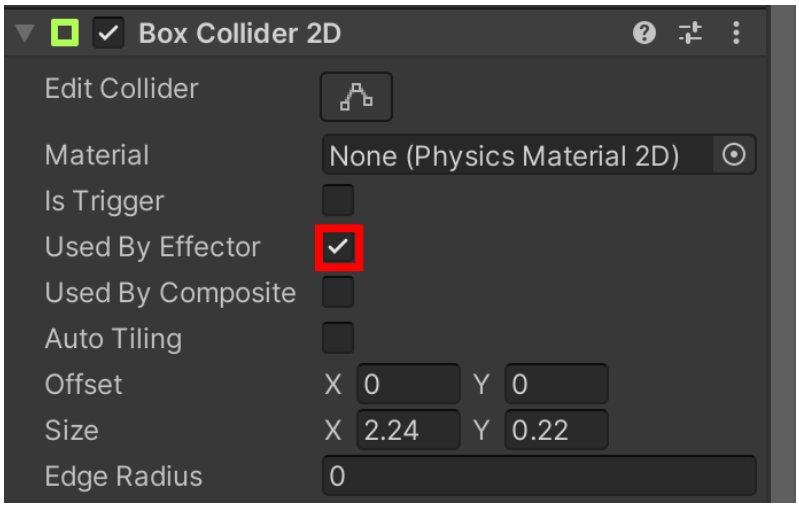
Sometimes the sprite still may not show up. In that case, change the order in layer from **0** to **1** in the Sprite Renderer component.



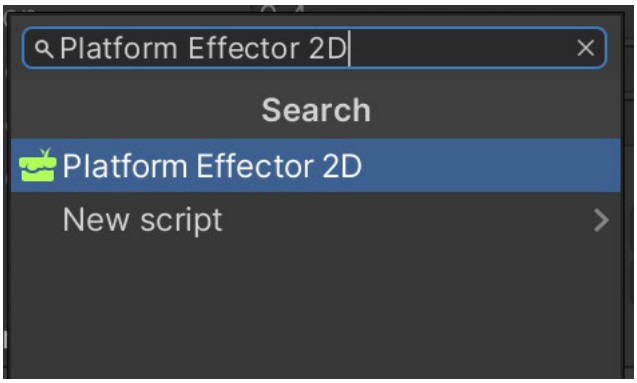
20 With the platform still selected, click **Add Component**. In the search box, type ***Box Collider*** and select **Box Collider 2D**.



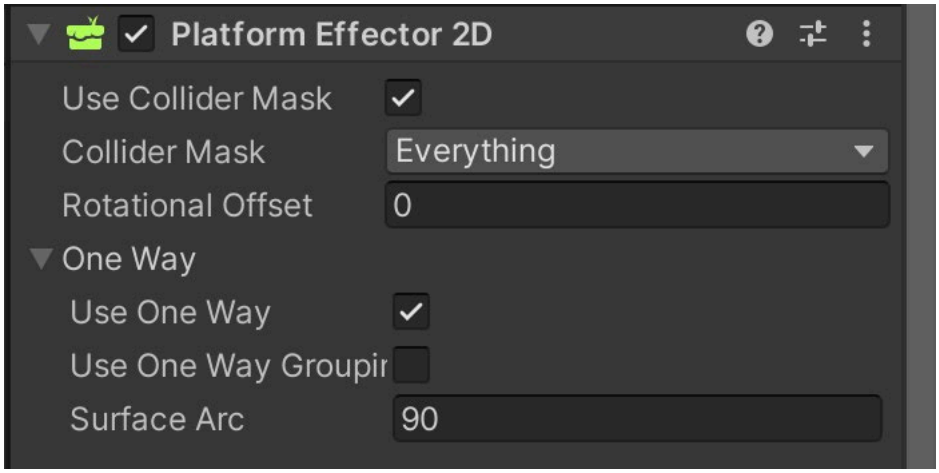
Update the **X** and **Y** size of the Box Collider 2D component to match the image. Select the **Used By Effector** check box.



21 Click **Add Component** again. Type ***Platform Effector 2D*** into the search box and select **Platform Effector 2D**.



Once added, change the **Surface Arc** to 90.



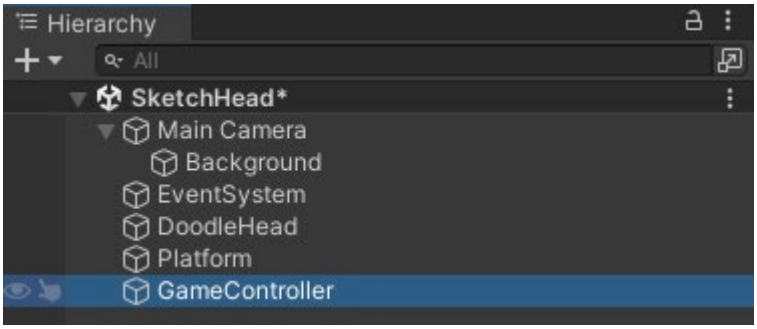
22 Now, let's play the scene. After clicking play, switch back to the scene view while the game is playing. You should see the DoodleHead character jump up very high. You will notice that when the DoodleHead's Y position is less than the Camera position, the Camera will stop following.

23 Stop the game.

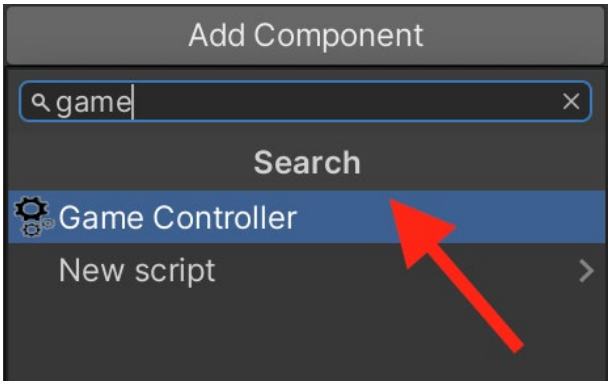
24

Now it's time to spawn more platforms to collide on. This will be done using a method called **object pooling**. This method involves spawning in many objects at the beginning of the game, and then re-using them during the game, instead of destroying and re-creating objects.

Create a new game object and rename it ***GameController***.



With the GameController selected, click **Add Component**. In the search box, type **Game**. Find and select the **Game Controller** script.



25 Because there is only one GameController object in the game and other scripts will want to refer to this GameController, it's a good idea to make this into a **singleton**.

Singletons are special types of scripts that allow only one of them to run at a time. When an object with a singleton script is spawned in, it checks if there is already a version of the script in existence, and if not, then it becomes the acting version.

To create a singleton at the top of the GameController script, create a static variable to store the active GameController instance. Remember that static variables keep the same variable even on multiple scripts.

```
public class GameController : MonoBehaviour
{
    public static GameController instance;
```

26 After the singleton, add some additional variables to store the platform GameObject and the y position to spawn in platforms at.

```
public static GameController instance;

[Header("Platform Object")]
public GameObject platform;
public float yPos = 0;
```

27

Inside the **void Start** function, check if there is not already an instance. If there isn't, set the instance to be the current script, and call the **SpawnInitialPlatforms** function (which will be created in the next step).

Type this code inside the **void Start** function:

```
void Start()
{
    if (!instance)
    {
        instance = this;
        SpawnInitialPlatforms();
    }
}
```

28

Create a new **void SpawnInitialPlatforms** inside the class below **void Start**, then add this code:

```
private void SpawnInitialPlatforms()
{
    for (int i = 0; i < 100; i++)
    {
        SpawnPlatform();
    }
}
```

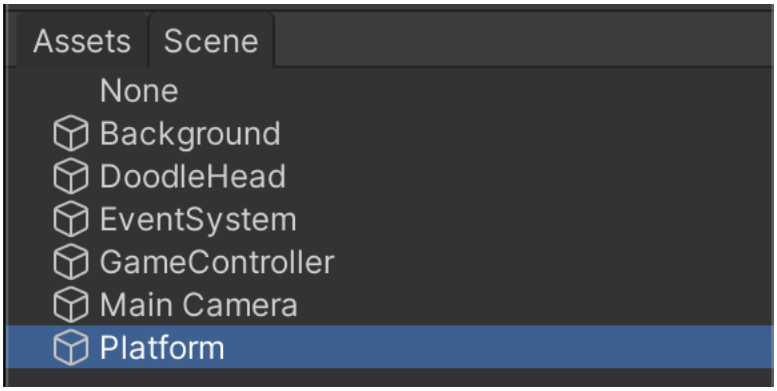
29

Lastly, create a new **public void SpawnPlatform** inside the class and below **void SpawnInitialPlatforms** and add this code:

```
private void SpawnPlatform()
{
    float xPosition = Random.Range(-3f, 3f);
    Instantiate(instance.platform, new Vector3(xPosition, instance.yPos, 0), Quaternion.identity);
    instance.yPos += 2.5f;
}
```

Notice that it is using the **instance** variable's yPos! You can now delete the **void Update** function. Save the script and return to Unity!

30 Within the GameController Inspector input field, search and select the **Platform** game object.



31 Now, let's play the scene. There should be platforms spawning for the player to jump on!

32 Stop the game.

33

Now, there are over 100 platforms! However, the player will eventually reach the top. The platforms should move back up to the top of the game after they have moved off screen.

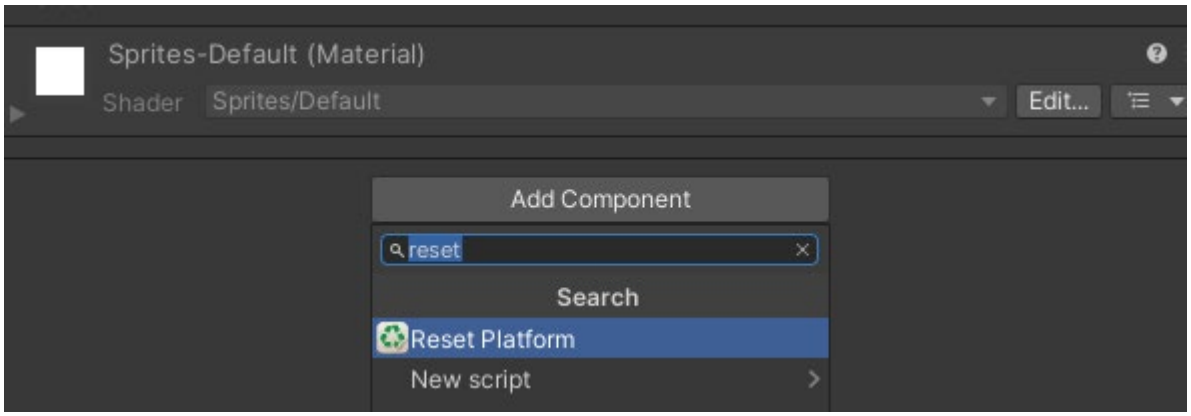
Start by adding a **public static void MovePlatformToTop** function to the GameController script. By making the function static, the function can be called without having to find the GameController object! Add the code in below:

```
public static void MovePlatformToTop(GameObject platform)
{
    float xPosition = Random.Range(-3f, 3f);
    platform.transform.position = new Vector3(xPosition, instance.yPos, 0);
    instance.yPos += 2.5f;
}
```

Notice again that it is using the instance's yPos!

34

Next, the platforms need to use this method when they are no longer visible. In Unity, select the Platform, then select **Add Component**. In the search bar type **ResetPlatform**. Add the script.



35 Open the script. Delete the Start and Update methods. Add the code below.

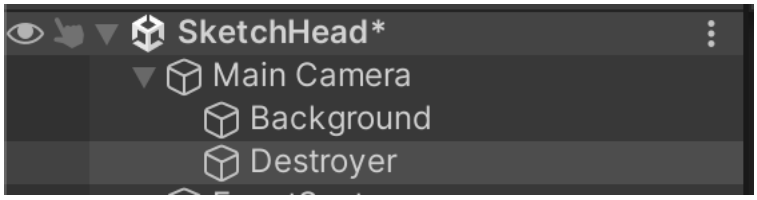
```
public class ResetPlatform : MonoBehaviour
{
    Unity Message | 0 references
    private void OnBecameInvisible()
    {
        GameController.MovePlatformToTop(gameObject);
    }
}
```

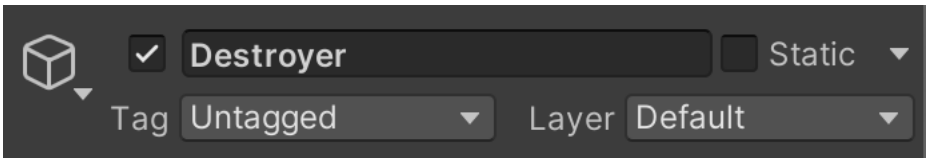
This uses a built-in method that runs when the GameObject is no longer on screen. Notice how the **MovePlatformToTop** method can be used now that it's static!

36 Save the script and return to Unity. Notice that the original platform changes y position after it moves off the screen!

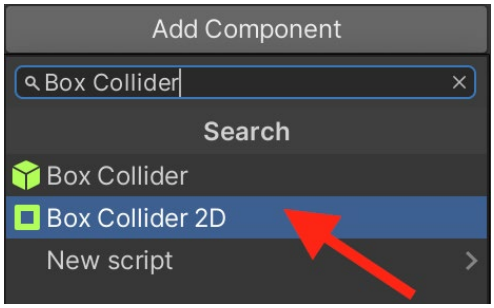
37 Now that the platforms are taken care of, let's make the player able to lose. To do so, we must add a new GameObject inside the **Main Camera** object. Rename this GameObject to ***Destroyer*** in the Inspector.

Remember, to move the game object inside the Main Camera, you can drag it up to the Main Camera.

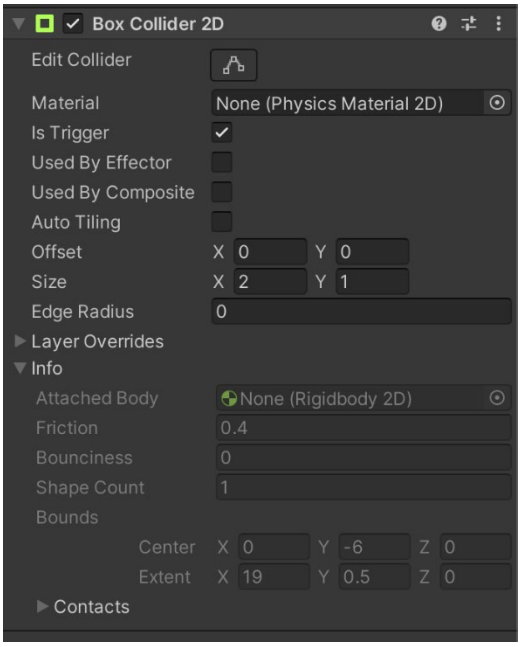




38 With **Destroyer** still selected, click **Add Component**. In the search box, type ***Box Collider*** and then select **Box Collider 2D**.



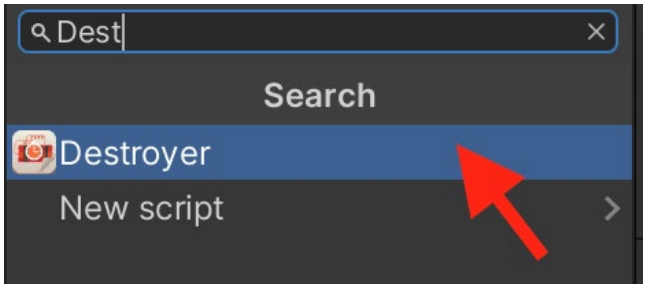
Change your GameObject and the Box Collider 2D component to match the information shown in the images below.



Position values: X: 0, Y: -6, Z: 0; **Scale Values**: X: 19, Y: 1, Z: 1

Ensure that **Is Trigger** is checked in the **Box Collider 2D** component and that you update the **Offset** and **Size** to match the image.

39 With the **Destroyer** GameObject still selected, click **Add Component**. In the search box, type ***Destroyer*** and select the **Destroyer** component.



40 Open the **Destroyer** Script and delete the Start and Update methods. Create a new **void OnTriggerEnter2D** function inside the class. An error will appear but don't worry, we will resolve that shortly.

```
public class Destroyer : MonoBehaviour
{
    Unity Message | 0 references
    public void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject.Find("DoodleHead").SetActive(false);
        GameController.GameOver();
    }
}
```

41

Now that we just added the **void OnTriggerEnter2D** function in the last step, we need to update our code in the Game Controller script to fix the error. Reopen the **Game Controller** script and add the following code inside the class:

```
[Header("Platform Object")]
public GameObject platform;
float yPos = 0;

[Header("Game Over UI Canvas Object")]
public GameObject gameOverCanvas;
```

After the **SpawnPlatforms** function, add a **static void GameOver** function and update the code to match below. Again, note that making the method static allows it to be called from the Destroyer script without having to find the component in the game!

```
public static void GameOver()
{
    //Game Over Canvas is set to active
    instance.gameOverCanvas.SetActive(true);
}
```

You can now save the script and return to Unity!

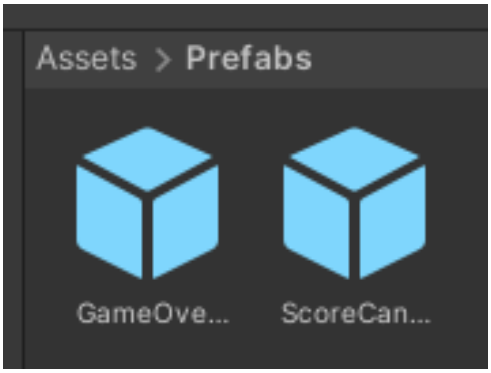
42 Go to the **Prefabs** section of the **Assets** folder to find the **GameOverCanvas** object, then place it into the **Hierarchy**.

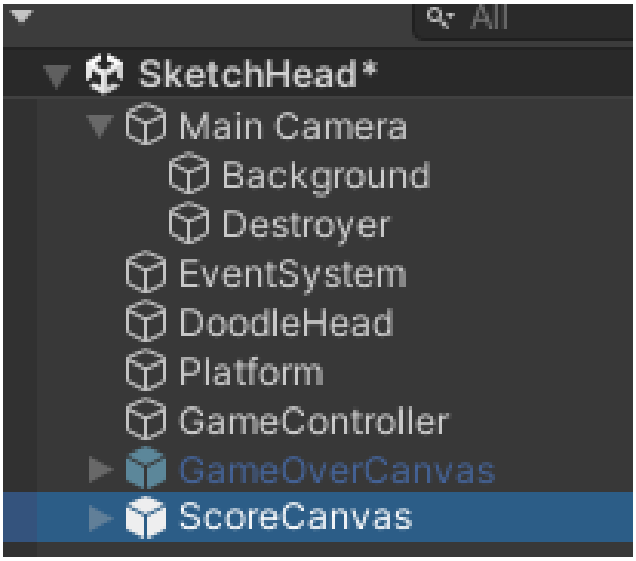
43 Select the **GameController** in the Hierarchy, then find the **Game Controller** script component. Find **Game Over UI Canvas Object** then search in the input field for the **GameOverCanvas** game object.

44 Play the scene. You will notice that if the object collides with the collider, the player will be invisible, and a game over screen appears!

45 Stop the game.

46 Now for the part you have been waiting for! Let's add scoring to the game! In the Assets folder, go to the Prefabs section then place the ScoreCanvas into the Hierarchy.





47

We need to adjust the code in the Player Controls to update the score. Reopen the **Player Controls** Script that is attached to the DoodleHead and add the following code inside the class:

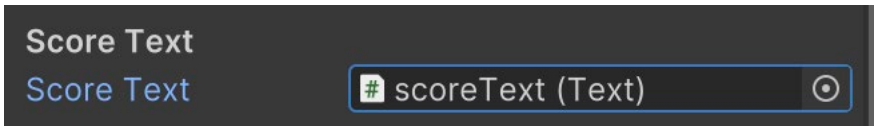
```
[Header("Rigidbody Settings")]
[Tooltip("Rigidbody2D object that is stored")]
public Rigidbody2D rb;

[Tooltip("Downward speed of the object")]
public float downSpeed = 20f;

[Header("Movement Settings")]
[Tooltip("Movement Speed of the object")]
public float movementSpeed = 10f;

[Tooltip("Movement direction of the object")]
public float movement = 0f;

private SpriteRenderer spriteRenderer;

//Score of game
[Header("Score Text")]
public Text scoreText;
private float topScore = 0.0f;
```

Inside the **void Update** function in the same script, add the following code:

```
//if players velocity is greater than 0
//and position on the y axis is greater
//than the score
if (rb.velocity.y > 0 && transform.position.y > topScore)
{
    //score equals players position
    topScore = transform.position.y;
}
//Text for the score equals to the top score
scoreText.text = "Score: " + Mathf.Round(topScore).ToString();
```

Now save the script and return to Unity!

48 Select the **DoodleHead** in the Hierarchy. Under the Inspector, scroll down to find the **Score Text**. Click on the circle and replace none with **scoreText**.



49 Play the scene. You will notice that the higher the object goes, the more the score increases.

50 Great job! You can now **save** your game, then **export** it. **Submit** your game on the Dojo so a Code Sensei can grade it!