CODE NINJAS®

Silver Belt Ninja Guide
Activity 13: Amazing Ninja Worlds Part 2

Activity 13

Amazing Ninja Worlds – Part 2

By the end of the game, you will be able to track and respond to player actions by creating a checkpoint and pickup system and changing game scenes.

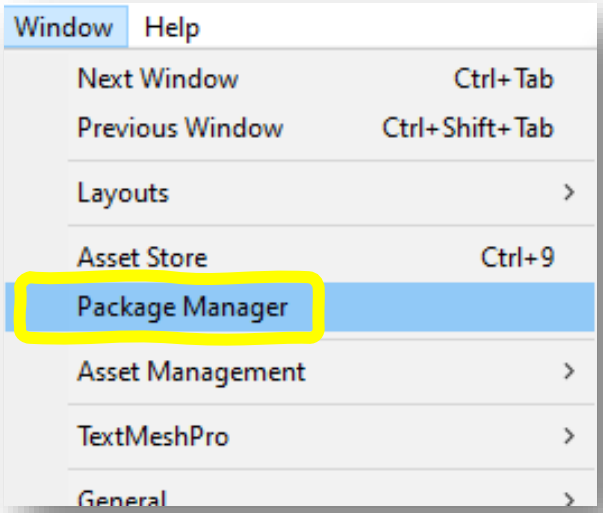
When the player is hurt, they get sent back to a checkpoint which also happens to be where they started the game. A checkpoint is a great way to save the player's progress so that if they are hurt after a difficult portion of the game, they get sent back to after that part instead of having to go back all the way to the beginning.

After the player gets past the vines near the beginning of the level, there's a second set of vines. If the player gets hurt there, they'll have to deal with both sets again and again. Let's change that.

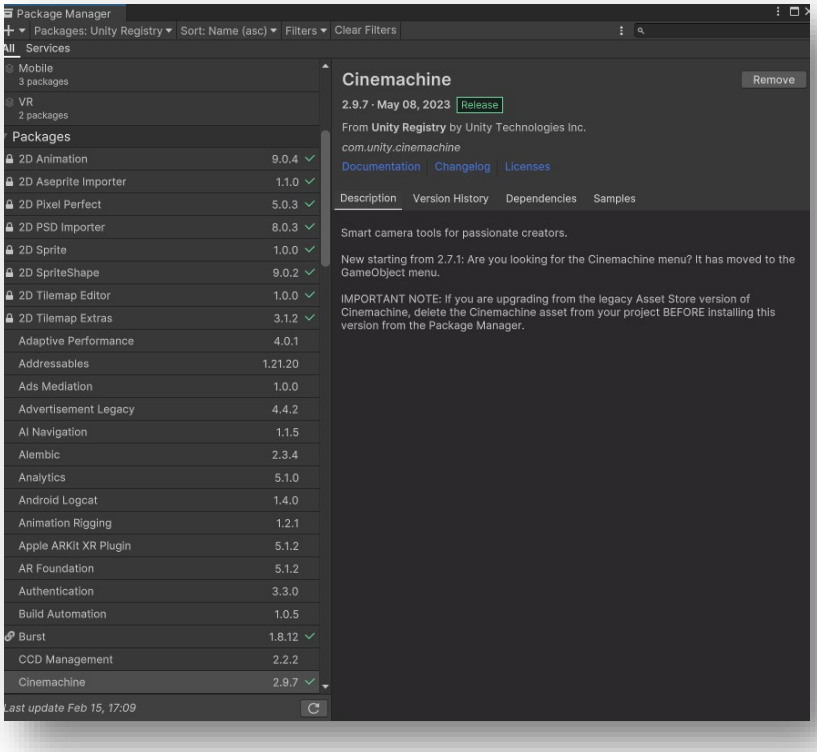
1 Start a new Unity Project and name it *YOUR INITIALS –Amazing Ninja Worlds 2*. Select **3D core**.

2 We will be using Cinemachine to control our camera. This will help us create awesome camera shots and angles for our game.
Go ahead and open **Window > Package Manager**.

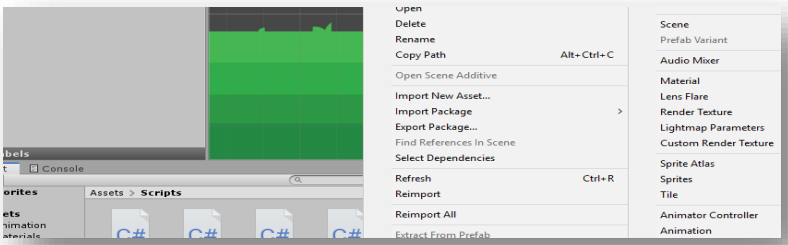
Be patient if it doesn't open right away. It might take a minute to load.



3 Find **Cinemachine** in the Unity Registry and click **Install** in the top right of the window.

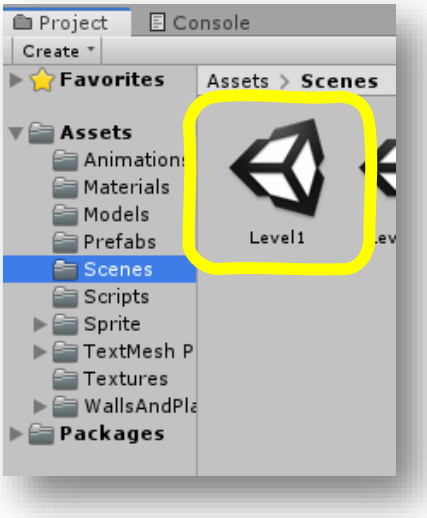


4 We've created a starter pack to give you a head start! To use it, import the **Activity 13 – ANWP2.unitypackage** by going to **Assets > Import Package > Custom Package> All > Import**.



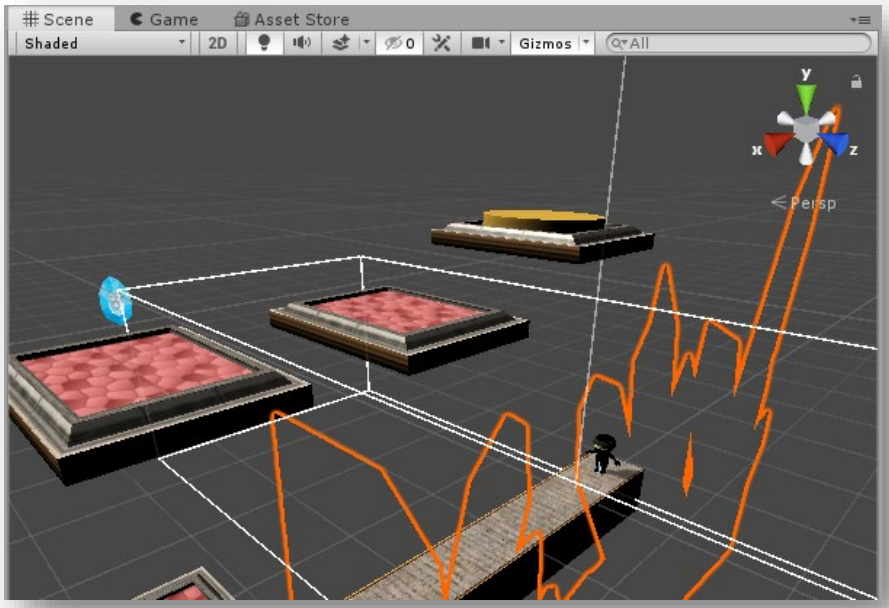
5 To open the starter package, double-click on the **Level1** scene.

You can find this in the **Project** tab under **Assets > Scenes > Level1**.



This will load the scene with all our game objects!

Make sure can see all the objects and the assets in the scene tab.



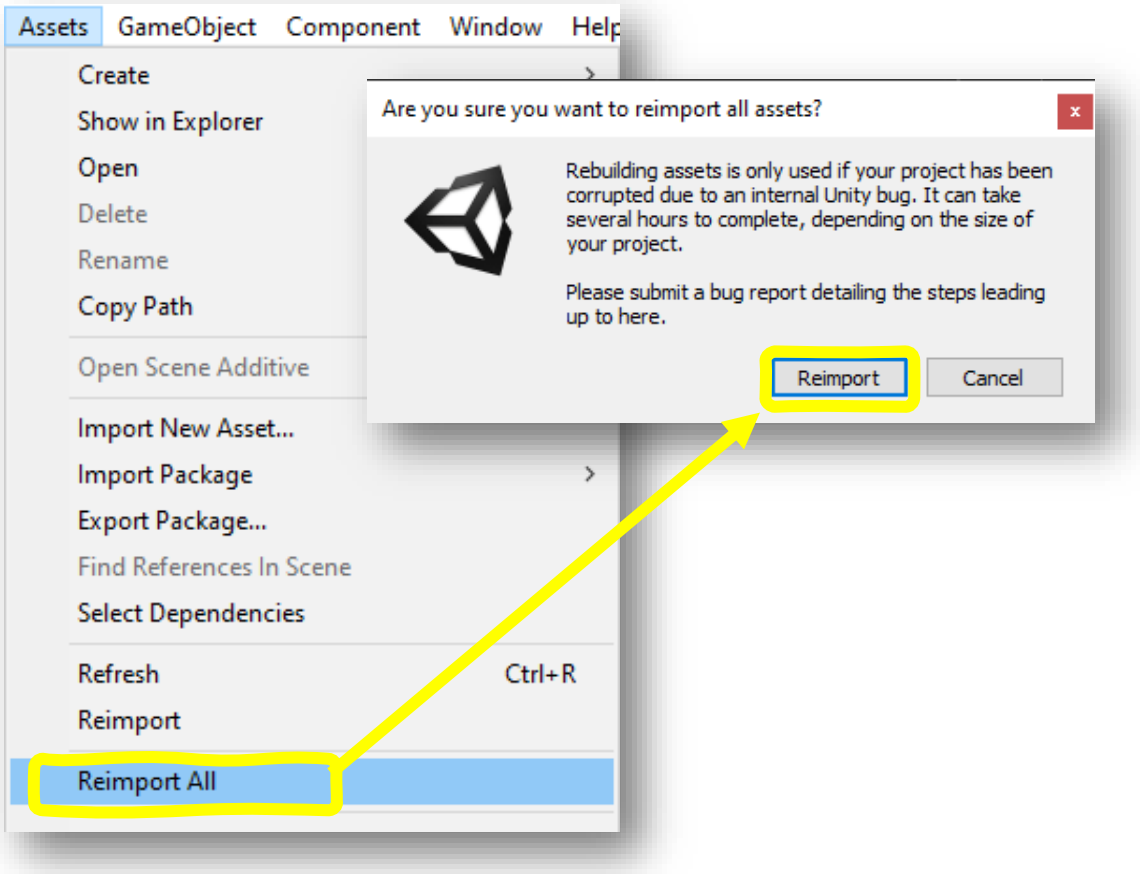
6

We should check that Cinemachine is working like it is meant to. In the **Hierarchy** tab, you should see a list of all the game objects in the scene.

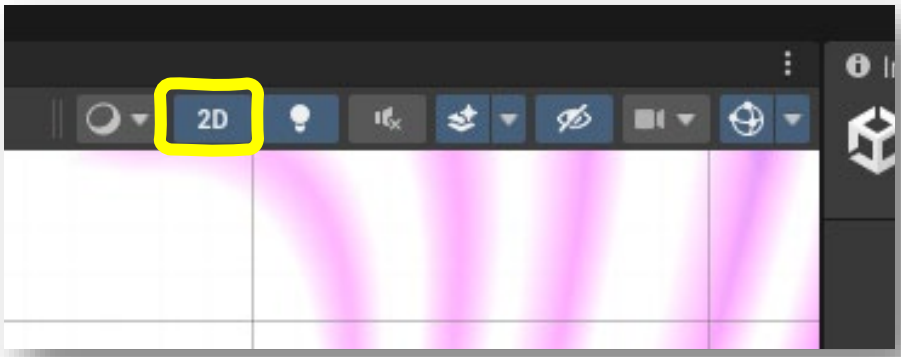
If you see a little grey and red icon attached to the Main Camera (shown in the picture), Cinemachine is working!

If you do not see it, go into the **Assets** tab and press **Reimport All**.

Then after Unity is done importing, check the Main Camera again.


Assets
GameObject
Component
Window
Help
Create
Show in Explorer
Open
Delete
Rename
Copy Path
Open Scene Additive
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh
Ctrl+R
Reimport
Reimport All
Are you sure you want to reimport all assets?
Rebuilding assets is only used if your project has been corrupted due to an internal Unity bug. It can take several hours to complete, depending on the size of your project.
Please submit a bug report detailing the steps leading up to here.
Reimport
Cancel

7 Make sure you click the 2D button on the bar on top of the Scene tab. This will align your camera properly.

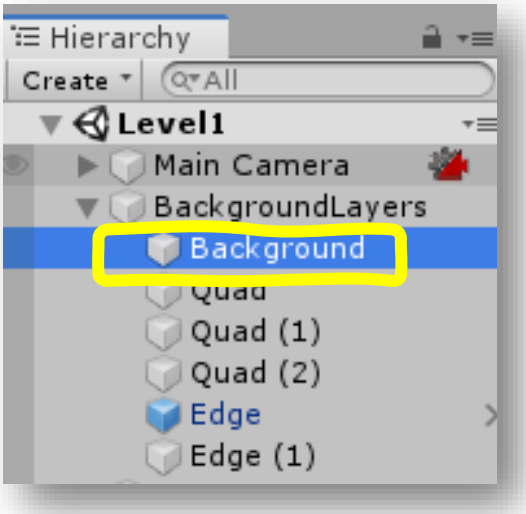


8 In games, a checkpoint is a position in the world that the player is warped to if they fail a difficult challenge.

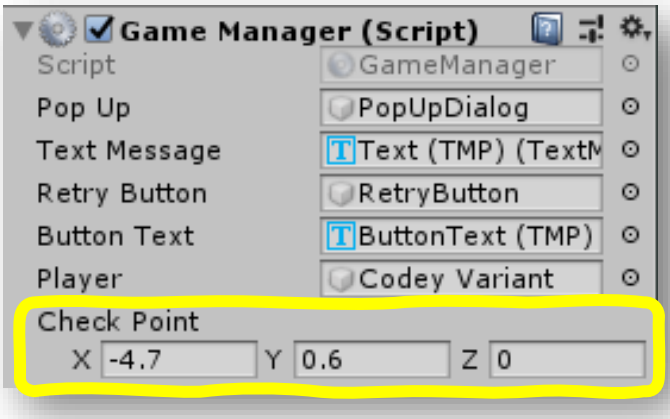
Have you ever played a difficult game where if you lost you got sent back to the start of the level? That can be very frustrating! Part of game design is always thinking about the player.

9 How could we program a checkpoint system? What data would we need to store? How could we move Codey to the checkpoint? Discuss some possible ideas with your Code Sensei before moving on.

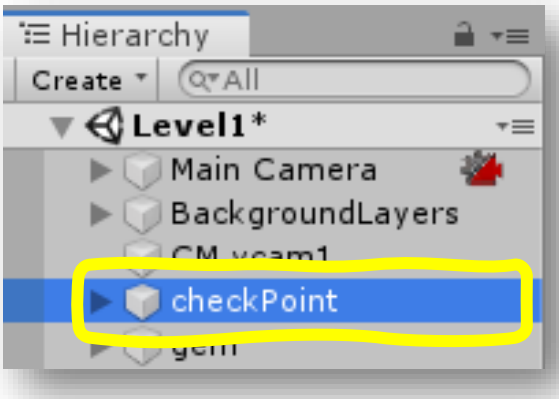
10 In the Hierarchy, find the Background game object that is located inside of the BackgroundLayers game object.



11 In the Inspector, find the Game Manager (Script) component. The Ninja World game is set up so this object keeps track of the player's current checkpoint. Right now, the checkpoint is initialized to be Codey's starting position.

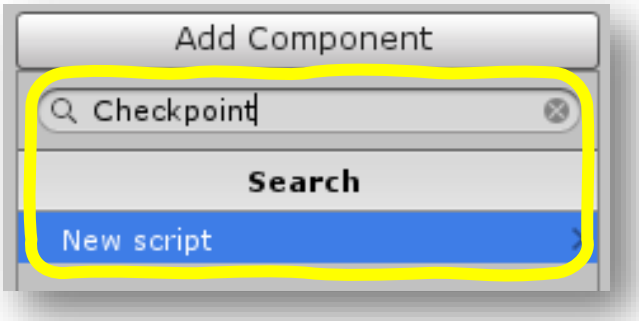


12 Now that we know how Ninja World stores checkpoints, we need to figure out how to update them! Find the existing checkPoint game object in the Hierarchy and click on it to open it in the Inspector.

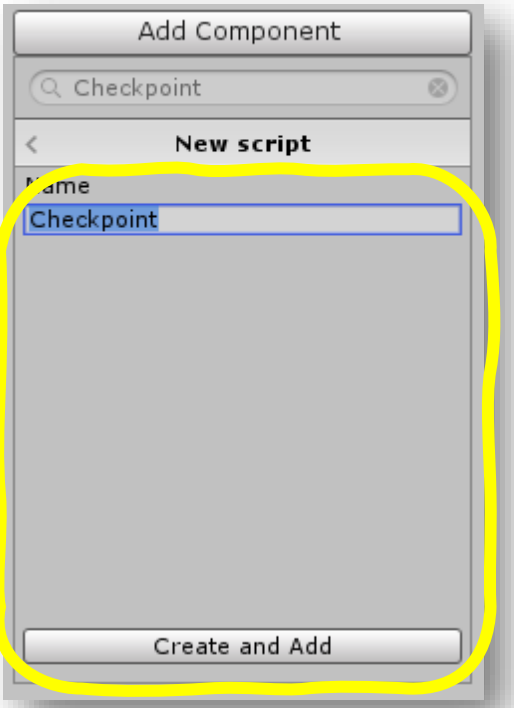


13 Click the Add Component button.

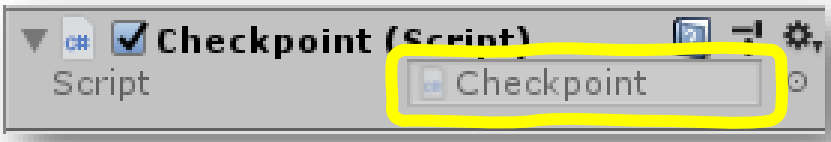
14 Type "Checkpoint" and select "New Script".



15 Click “Create and Add” to create and attach a new script named “Checkpoint”.



16 Double click on the Script Checkpoint box in the Inspector to open it in Visual Studio.



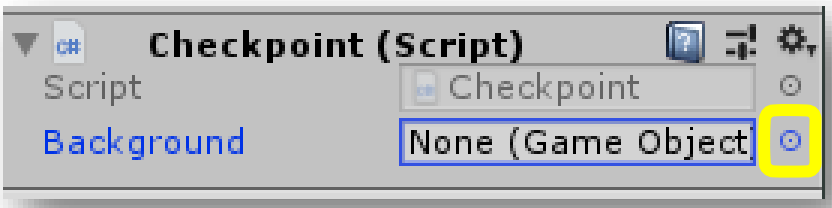
17 We do not need the Start or Update functions so delete them both.

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

0 references
public class Checkpoint : MonoBehaviour
{

}
```

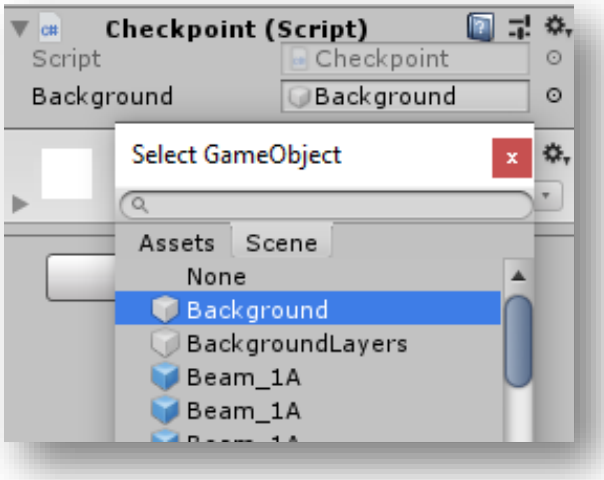
18 Based on our earlier investigation, we know that the background object needs to know the about Codey's checkpoint. Add public GameObject background; inside the two curly brackets.

```
public class Checkpoint : MonoBehaviour
{
    public GameObject background;
}
```

19 Save your script and go back to Unity. Click the little circle next to Background None (Game Object) to open the GameObject selector.



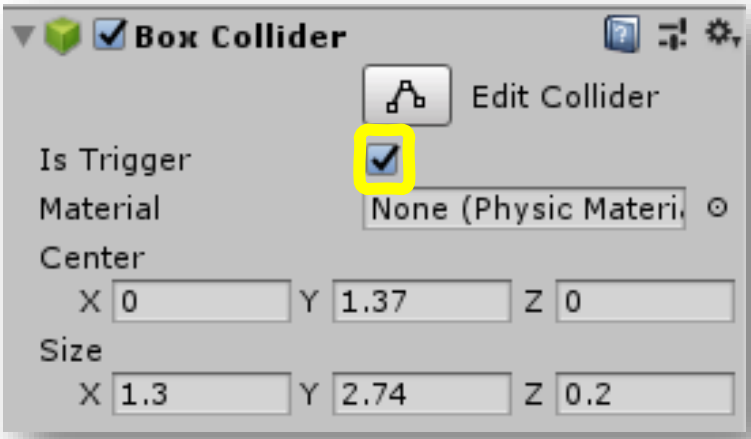
20 Find "Background" and click it. Close out of the Select GameObject window. We can now reference and use the background in our Checkpoint script!



21

While in the Inspector, look at the checkpoint's Box Collider component.

We have Is Trigger enabled. This means that we can check to see if Codey collides with the checkpoint object, but the object will not interact with Codey's movement.

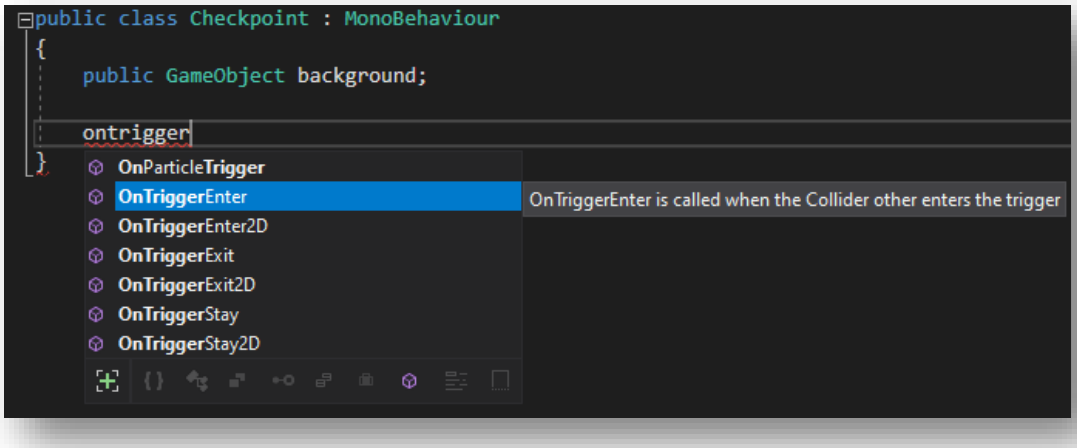


22

Open the Checkpoint script in Visual Studio.

After the `public GameObject background;` line, start typing "ontrigg" and Visual Studio should pop up a box of suggestions for you.

Double click on OnTriggerEnter to have the function automatically created for you!



23 If it didn't work, type `private void OnTriggerEnter(Collider other) { }`, making sure to leave an empty line between the curly brackets.

```
public class Checkpoint : MonoBehaviour
{
    public GameObject background;

    private void OnTriggerEnter(Collider other)
    {

    }
}
```

24 What do we need to do when Codey runs into the checkpoint? We need to grab the position of the checkpoint object and ask the background to update its checkpoint variable to the new position.

Inside the OnTriggerEnter function create a new Vector3 variable named newCheckpoint and set it equal to the checkpoint game object's position by typing
`Vector3 newCheckpoint = transform.position;`

```
private void OnTriggerEnter(Collider other)
{
    Vector3 newCheckpoint = transform.position;
}
```

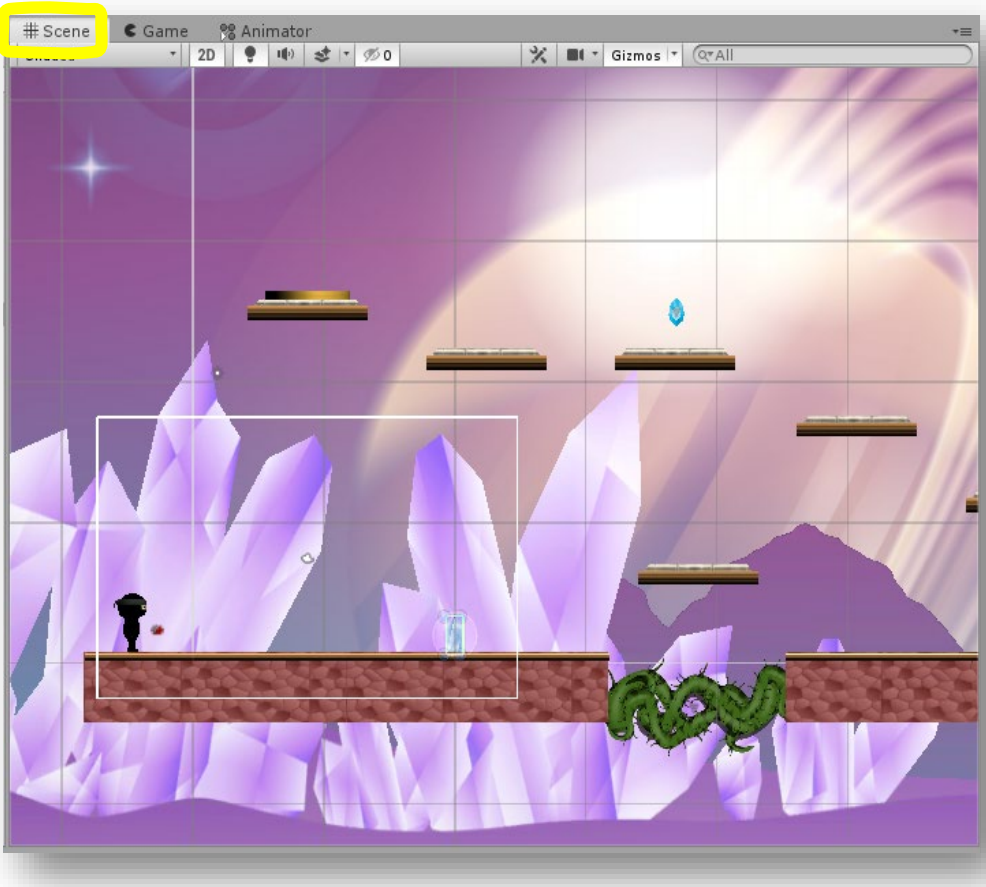
25 In order to send this new checkpoint to the background, we need to find the background's GameManager component set its checkpoint variable to be equal to our new checkpoint.

After the newCheckpoint line, type
`background.GetComponent<GameManager>().checkPoint = newCheckpoint;.`

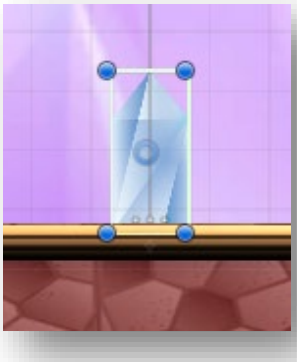
26 Play your game and test out your new checkpoint!

- What happens when you touch the checkpoint and run into vines?
- What happens when you run out of lives?
- What happens if you jump over the checkpoint and touch the vines?

27 In Unity, open the game's scene by clicking on the Scene tab.



28 Click on the checkpoint crystal in the scene to select it.



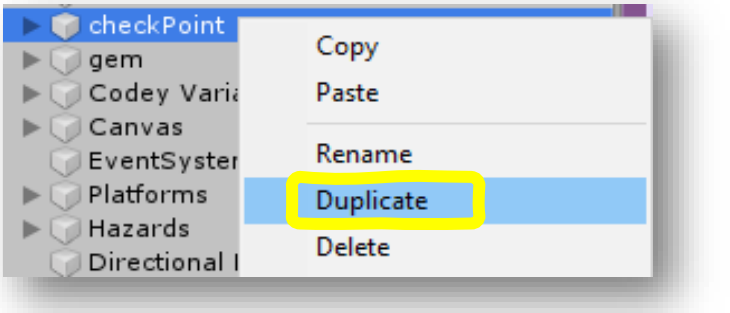
29 Place the crystal where you think there should be a checkpoint. This is up to you! It can be anywhere!

What happens when you place it in the air? On a platform? In vines?

30 Playtest your game and settle on a good place for your checkpoint!

31 What if you want to add a second checkpoint somewhere else in the level? All you need to do is duplicate our existing checkpoint!

32 Find the checkPoint game object in the Hierarchy. Right click it and select Duplicate.



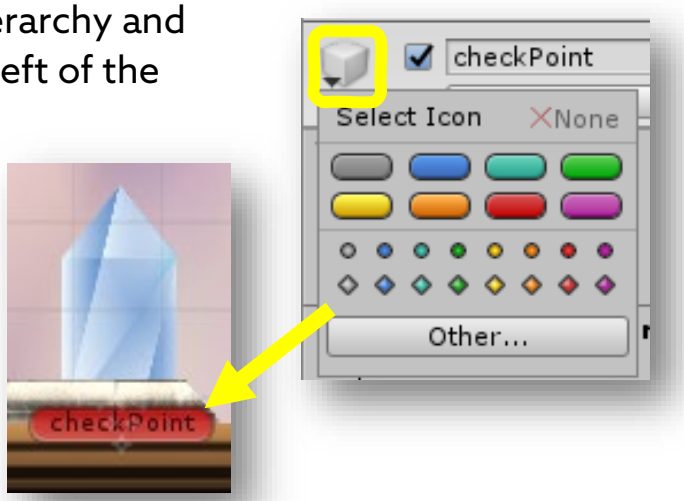
33 You should now have a new game object called checkPoint (1) in your scene.

34 This object is a perfect copy of the original checkpoint, down to the position in the scene.

But how do we know which is which?

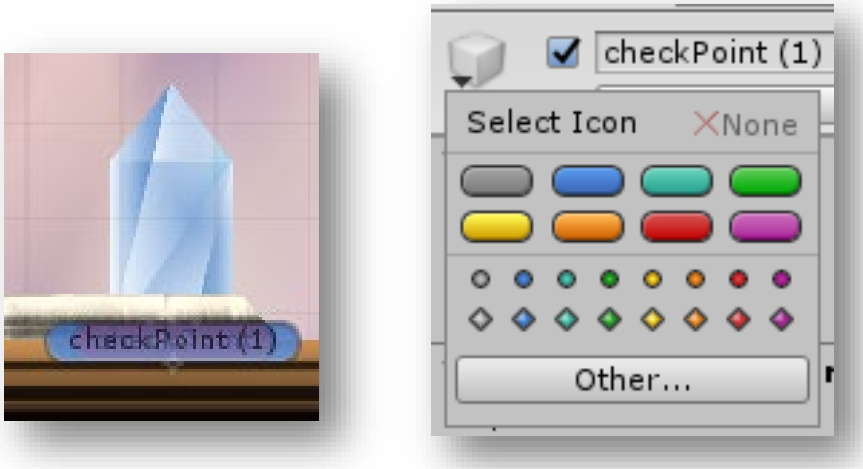
Click on checkPoint in the Hierarchy and look at the Inspector. To the left of the name is a gray cube.

Click on the cube to open the Select Icon window. Select one of the ovals and see what changes in the scene!

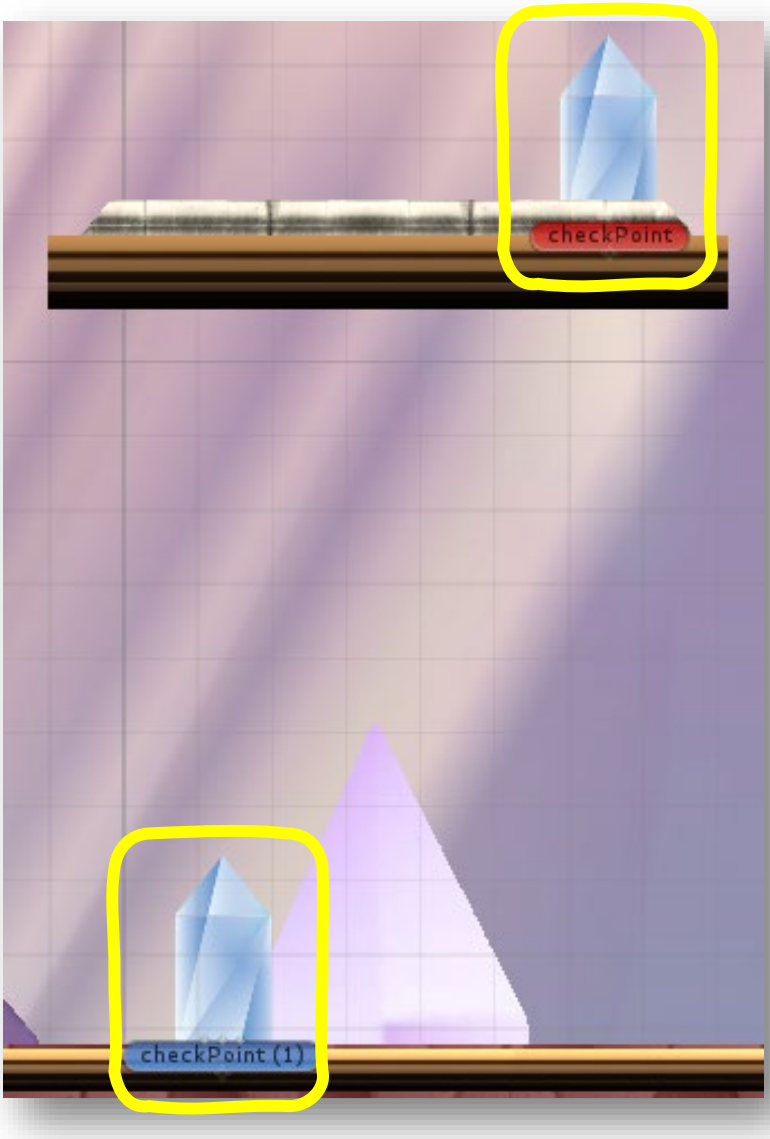


35 Click on checkpoint (1) in the Hierarchy and look at the Inspector. Click on the gray cube to open the Select Icon window. Select a different color than you did for the first checkPoint object.

36 Click on the new checkPoint (1) object and drag it to somewhere else in the scene.



37 Now that we can tell them apart, move checkPoint (1) somewhere else in the scene.



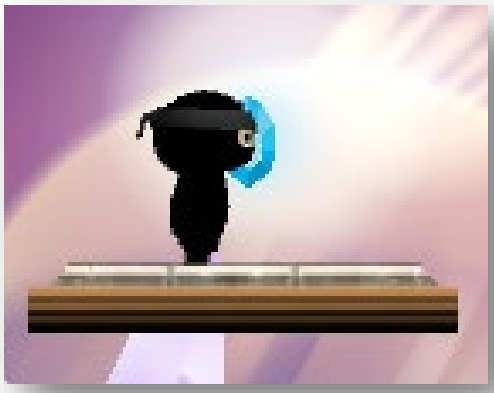
38 **Playtest** your game.

Try to play each possible outcome with checkpoints!

- Skip the first checkpoint and run into the vines.
- Skip the second checkpoint and run into the vines.
- Touch both checkpoints and run into the vines.
- Touch the first, then the second, and then the first again before running into the vines.

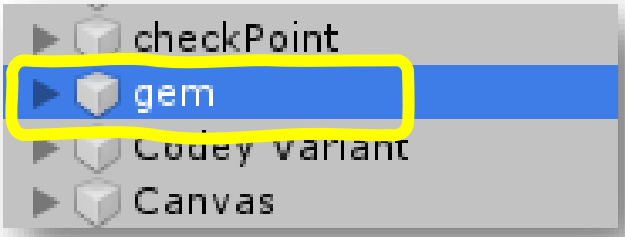
39 The goal of this level is to use the teleporter at the right side of the scene. We need to add and code a key to unlock the teleporter for Codey!

40 Find the blue gem on a platform in the top center of the scene. What happens now when Codey touches the gem? Nothing!

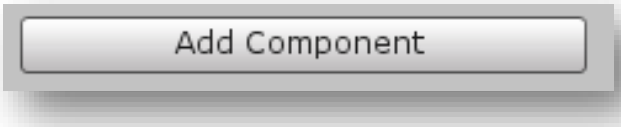


41 Like the checkpoint, the gem has a box collider set as a trigger. We just need to create the script and code it!

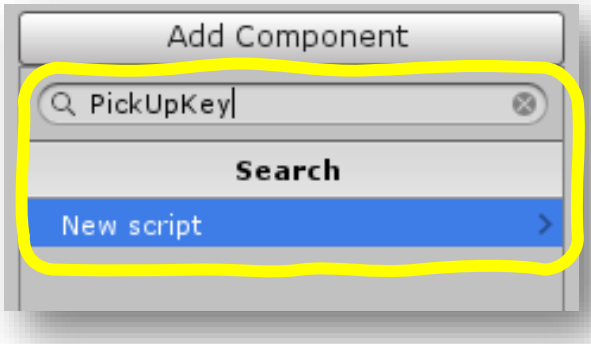
42 Click on the gem game object in the Hierarchy.



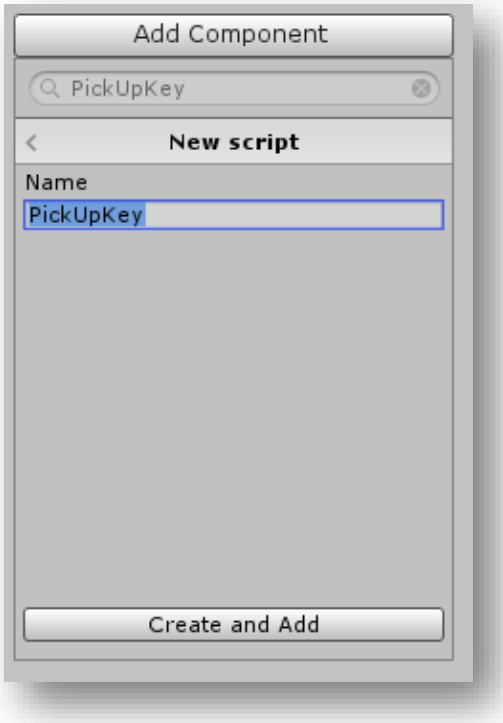
43 In the Inspector, click on Add Component.



44 In the box, type "PickUpKey" and click on the New Script option.



45 Make sure the name is correct and click Create and Add to create the script and add it to the gem component.

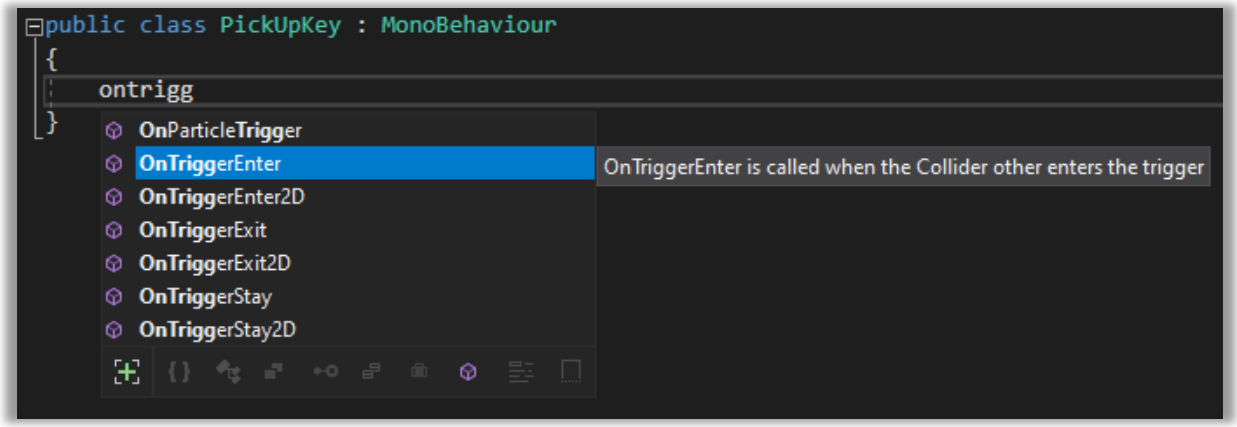


46 Double click on the Script PickUpKey box in the Inspector to open it in Visual Studio.

47 We do not need the Start or Update functions so delete them both.

```
public class PickUpKey : MonoBehaviour
{

}
```

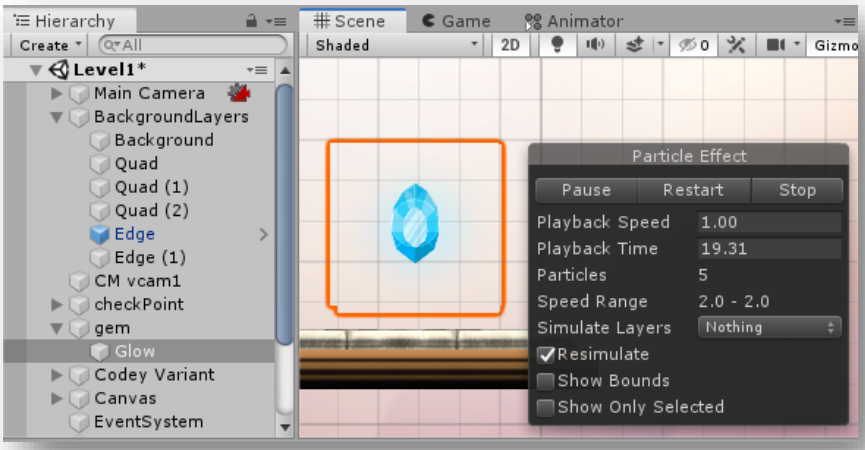
48 We want to detect when Codey touches our gem, so start typing "ontrigg" and Visual Studio should pop up a box of suggestions for you. Double click on OnTriggerEnter to have the function automatically created for you!



49 If it didn't work, type `private void OnTriggerEnter(Collider other) { }`, making sure to leave an empty line between the curly brackets.

```
public class PickUpKey : MonoBehaviour
{
    0 references
    private void OnTriggerEnter(Collider other)
    {

    }
}
```

50 When Codey touches the gem, we need to disable the gem and the attached particle system. Think back to the Colors game you made earlier in this belt. Do you remember how you disabled an object and its particle system?



51 If we disable the parent gem object, we will at the same time disable the child Glow particle system! Inside the OnTriggerEnter function, type `gameObject.SetActive(false);` to grab the gem's game object and disable it.

```
private void OnTriggerEnter(Collider other)
{
    gameObject.SetActive(false);
}
```

52 Save your script and play your game. What happens when Codey touches the gem now? The gem sprite and the glow from the attached particle system should disappear!

Now that you have the gem, try to find a teleporter to leave the level. Nothing happens!

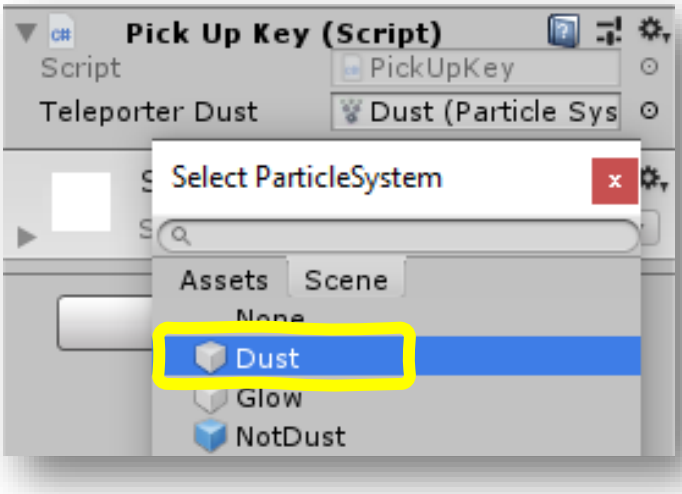
We just need to create a script and write a few more lines of code to get our teleporter working!

53

We can use a particle system to communicate to the player that the teleporter is active. We need to enable the teleporter's particles after Codey touches the gem. We first need to create a public variable for the teleporter's particle system inside the PickUpKey script. Before the OnTriggerEnter function, type `public ParticleSystem teleporterDust;` to create a variable we can set in the inspector.

```
public class PickUpKey : MonoBehaviour
{
    public ParticleSystem teleporterDust;

    private void OnTriggerEnter(Collider other)
    {
        gameObject.SetActive(false);
    }
}
```

54 In the gem's Inspector, find the Pick Up Key (Script) component. Click the little circle next to Teleporter Dust None (Particle System) to open the Select ParticleSystem menu. Find and click on Dust to attach it to the PickUpKey script as the teleporterDust variable.



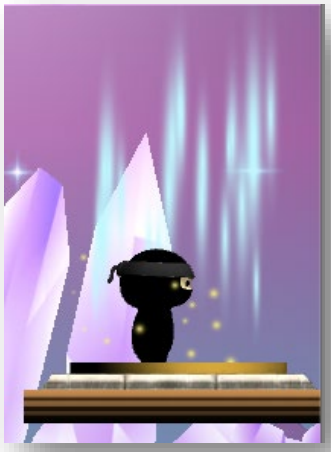
55 Back in the PickUpKey script, we need to tell the teleporterDust to play after Codey picks up the gem. After the gameObject.SetActive(false); line, add `teleporterDust.Play();` to tell the particle system to play!

```
private void OnTriggerEnter(Collider other)
{
    gameObject.SetActive(false);
    teleporterDust.Play();
}
```

56

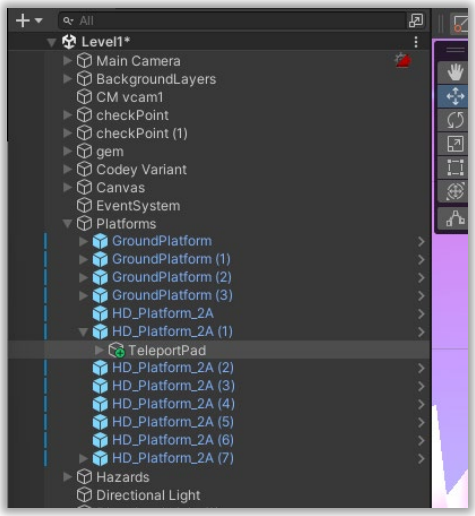
Playtest your game. Look at the teleporter on the right of the scene before and after you pick up the gem.

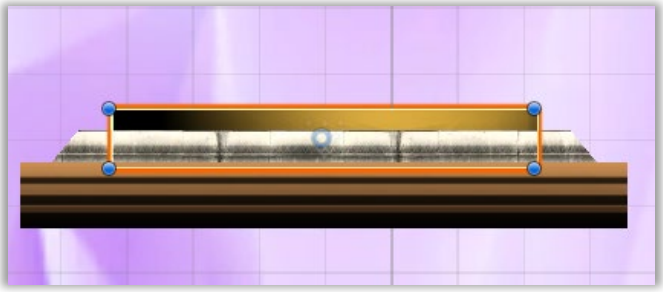
The teleporter still doesn't send Codey anywhere, but the player can now see when the teleporter is active thanks to the particle system!



57

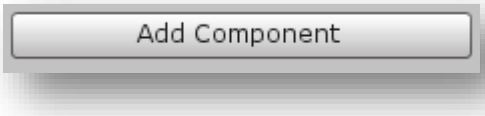
In the scene, click on the teleporter to select it in the Hierarchy and the Inspector. Alternatively, you can find it directly in the Hierarchy in **Platforms -> HD_Platform_2A (1) -> TeleportPad**



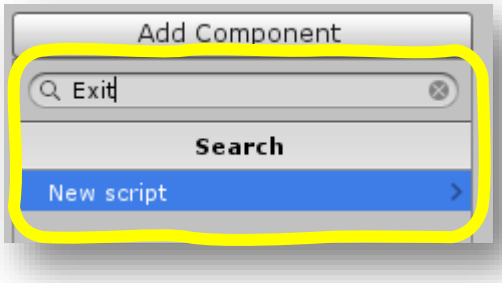


58

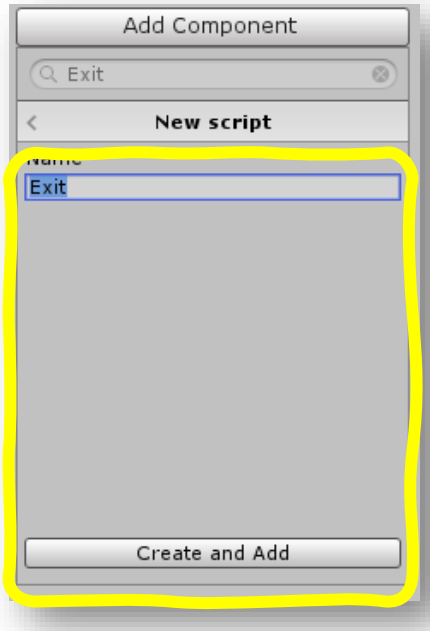
In the Inspector, click on Add Component.



59 Type “Exit” and select “New Script”.



60 Make sure the script name is “Exit” and click the Create and Add button to create the script and attach it to the teleport object.



61 In the inspector, double click the grey box that says Script Exit to open the script in Visual Studio.

62 We do not need the Start or Update functions, so we can delete them both.

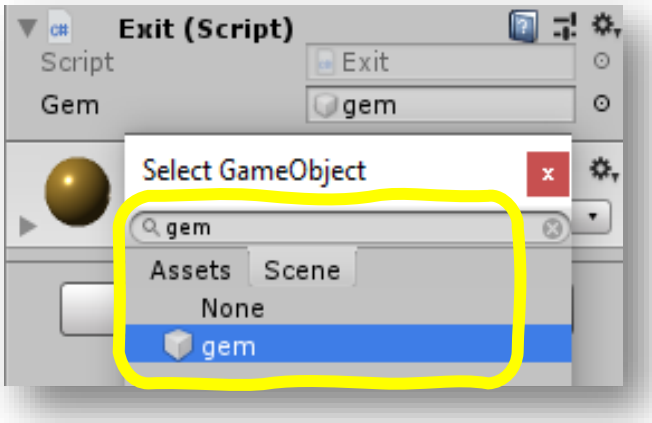
```
public class Exit : MonoBehaviour
{

}
```

63 The teleporter needs to know if Codey has picked up the gem, so we need to create a public reference to the gem so we can connect it to the teleporter through the Inspector. Inside the two curly brackets, type `public GameObject gem;`

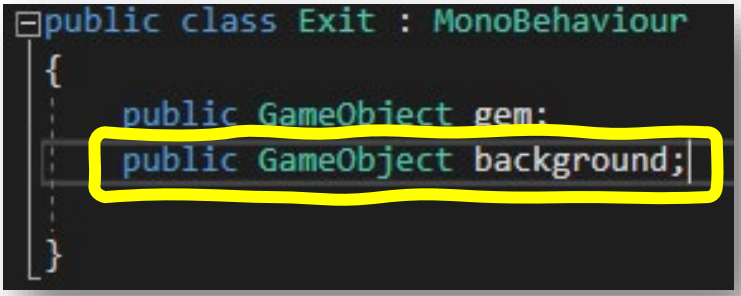
```
public class Exit : MonoBehaviour
{
    public GameObject gem;
}
```

64 In the Inspector, click the little circle next to Gem None (Game Object) to open the GameObject selector window.

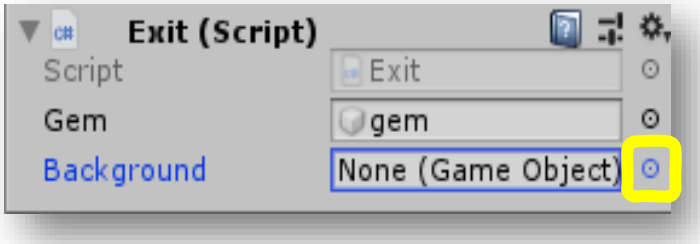
65 In the window, search for and select “gem” to attach it to the gem variable inside of our Exit script.



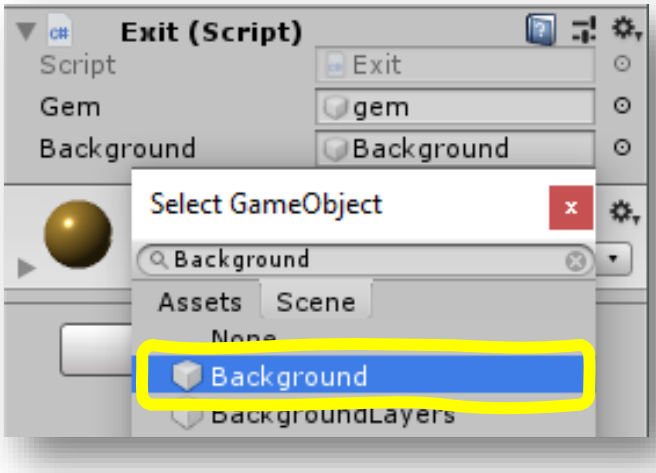
66 The teleporter also needs to know about the TeleportOpen function. This function will teleport Codey to another scene! The TeleportOpen function lives inside the GameManager script. The GameManager script is attached to the scene's background object After the public GameObject gem; line type `public GameObject background;` to create a public variable we can control in the Inspector.



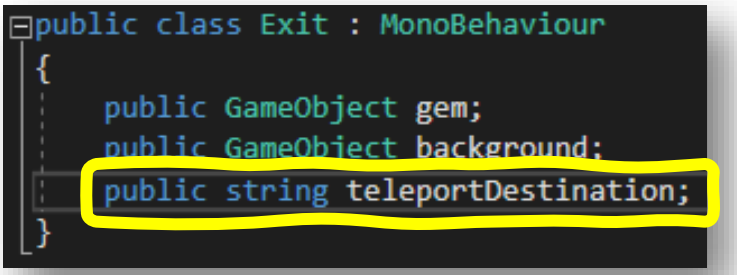
67 In the Inspector, click the little circle next to Background None (Game Object) to open the GameObject selector window.



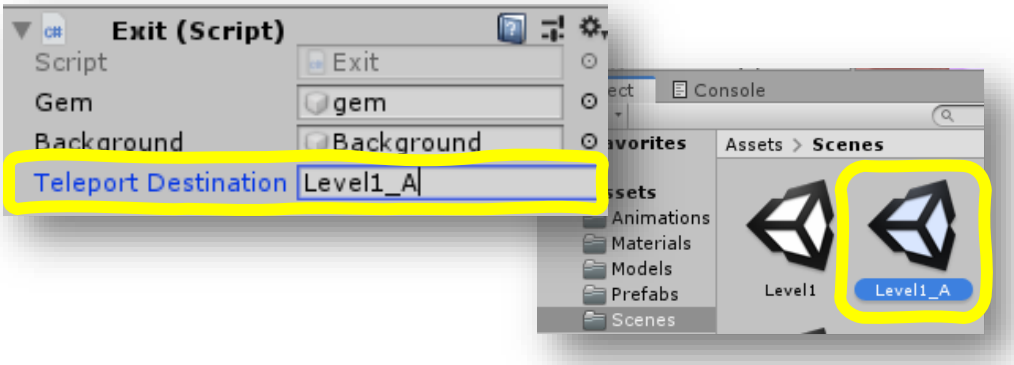
68 In the window, search for and select "Background" to attach it to the gem variable inside of our Exit script.



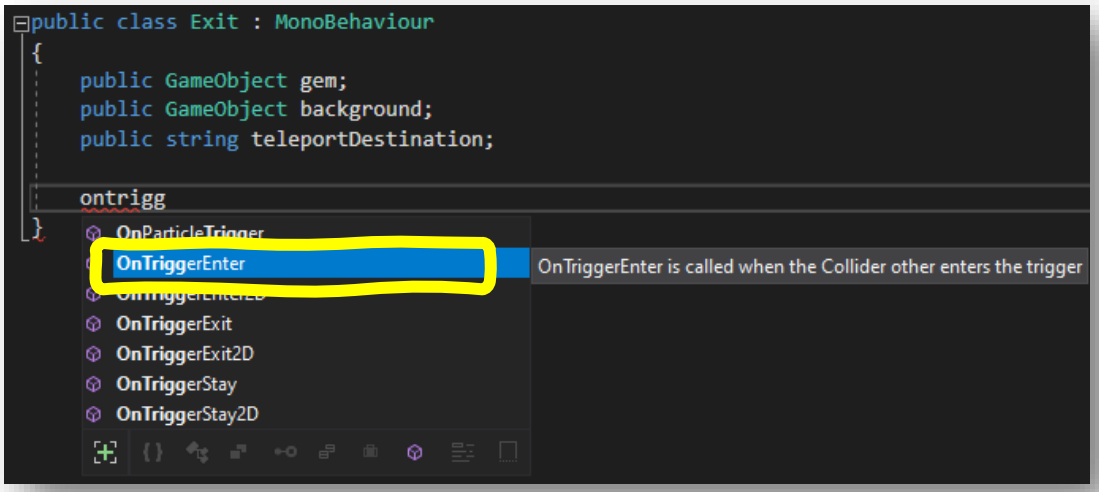
69 The final thing that the teleporter needs to know about is where to send Codey. To do this we will create a public string object called teleportDestination so we can set it in the Inspector. After the public GameObject background; line type `public string teleportDestination;`



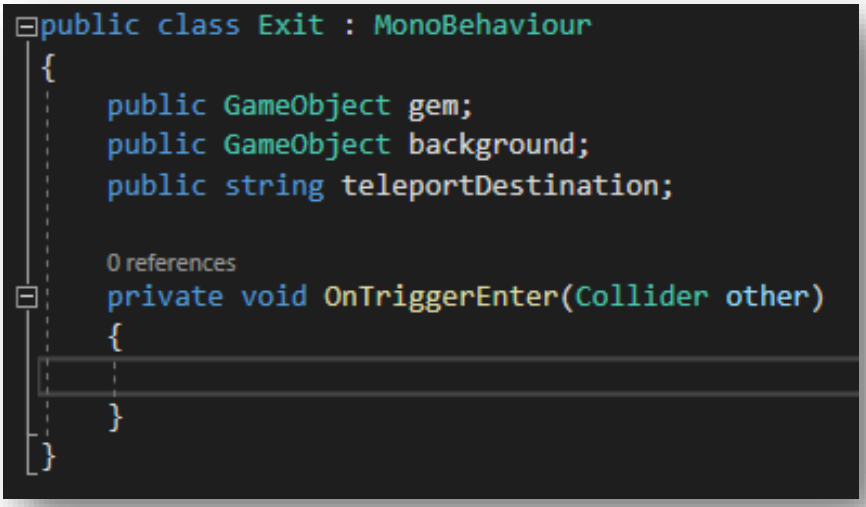
70 In the Inspector, give Teleport Destination a value of Level1_A. Make sure you type it in exactly to match the name of the scene in the game.



71 Now that we have our variables set up, we want to detect when Codey is touching our teleporter, so after the public string teleportDestination; line start typing "ontrigg" and Visual Studio should pop up a box of suggestions for you.
Double click on OnTriggerEnter to have the function automatically created for you!



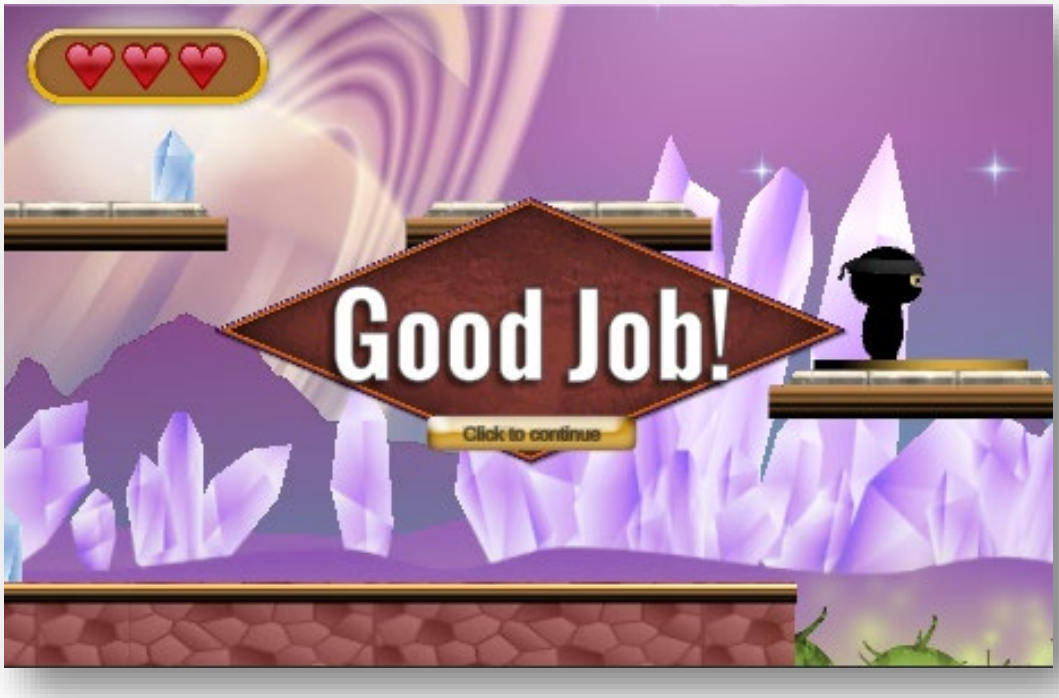
72 If it didn't work, type `private void OnTriggerEnter(Collider other) { }`, making sure to leave an empty line between the curly brackets.



73 When Codey enters the teleporter, we want him to teleport to next level based on our teleportDestination variable. We can do that by asking the background's GameManager component to run the TeleportOpen function with our teleportDestination string. Basically, we are asking the teleporter to send Codey to another one of our scenes. Inside the OnTriggerEnter function type `background.GetComponent<GameManager>().TeleportOpen(teleportDestination);`

```
private void OnTriggerEnter(Collider other)
{
    background.GetComponent<GameManager>().TeleportOpen(teleportDestination);
}
```

74 Play the game and collect the gem and go to the teleporter. Did something happen when you collided with the teleporter before you picked up the gem?



We need to create the logic that requires Codey to have touched the gem before enabling the teleporter!

75 In the OnTriggerEnter function, create an if statement by typing if () { } and put the background.GetComponent line of code inside of the if statement's curly brackets.

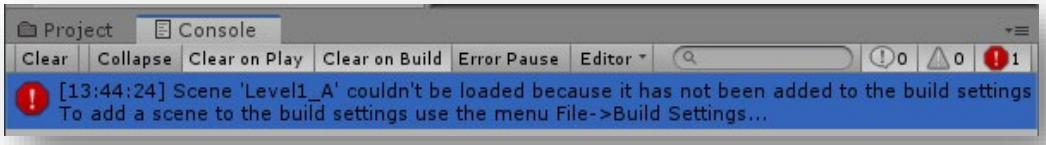
```
private void OnTriggerEnter(Collider other)
{
    if ()
    {
        background.GetComponent<GameManager>().TeleportOpen(teleportDestination);
    }
}
```

76 Inside of the parentheses, we need to see if the gem is disabled before teleporting Codey. Type `gem.activeInHierarchy == false` inside the parentheses to run the body of the if statement only if the gem is not active in the Hierarchy.

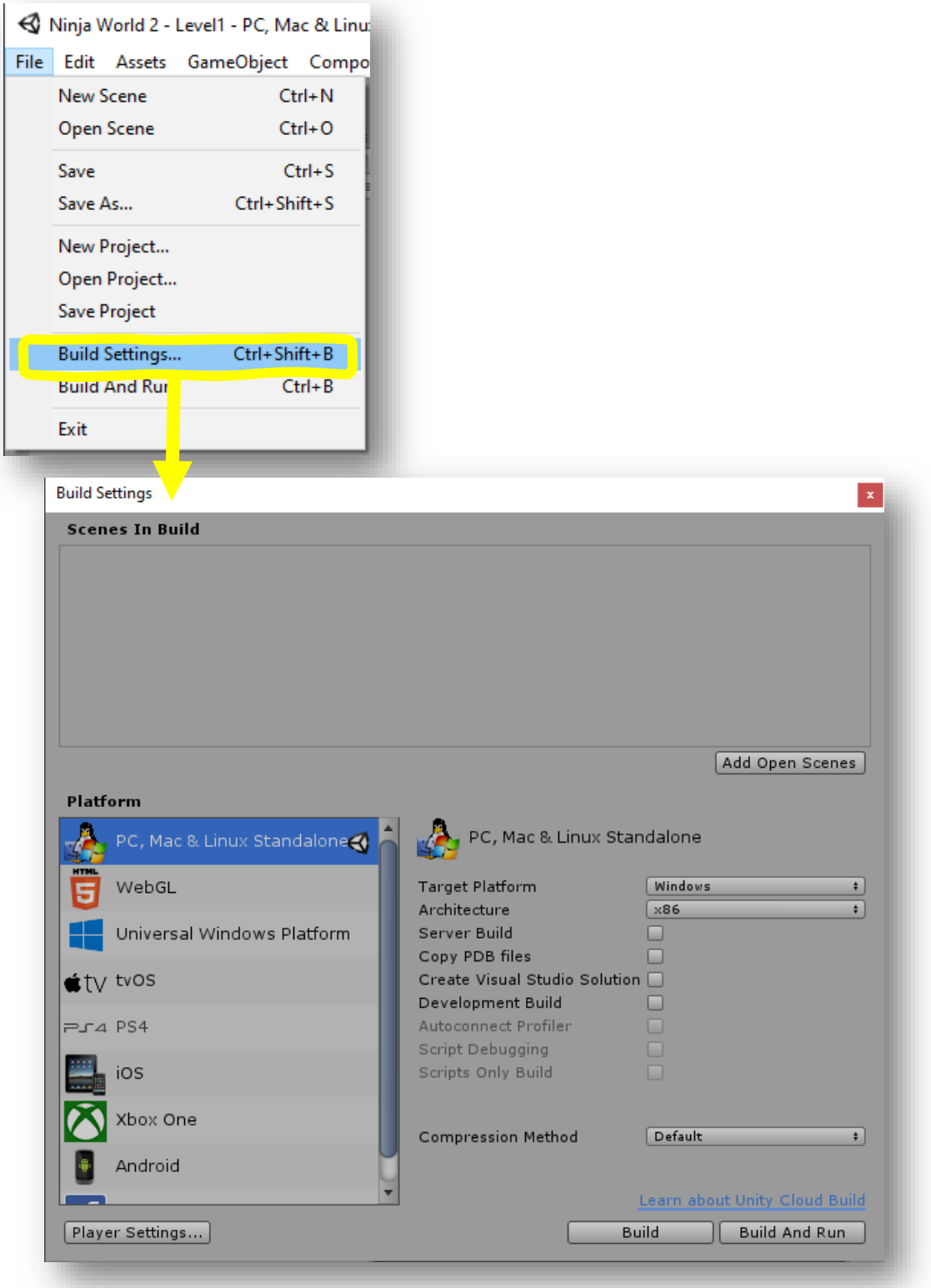
```
private void OnTriggerEnter(Collider other)
{
    if (gem.activeInHierarchy == false)
    {
        background.GetComponent<GameManager>().TeleportOpen(teleportDestination);
    }
}
```

77 Now play your game, making sure to touch the teleporter before you collect the gem. Nothing should happen until after you collect the gem and then touch the teleporter.

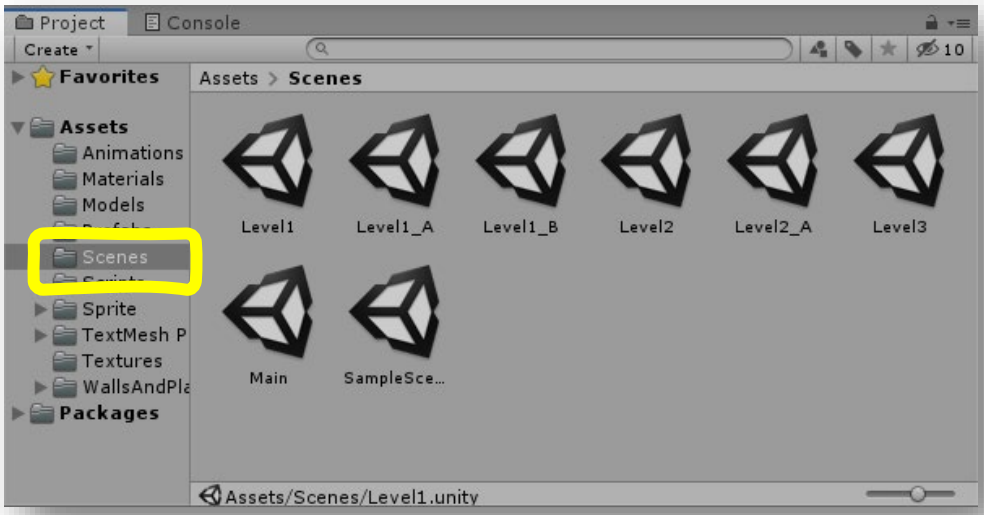
Once you get the gem and touch the active teleporter, what happens? You should get a Good Job screen with a Click to Continue button. What happens when you click that button? Did you get this error? Unity can't find the Level1_A scene because it is not loaded in the Build Settings menu! We can easily fix that!



78 In Unity, go to File -> Build Settings to open the Build Settings window.

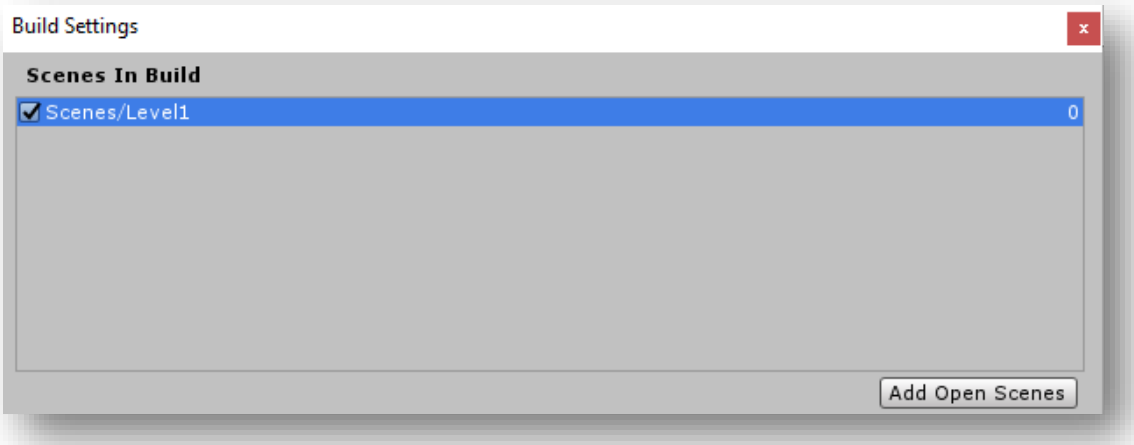


79 Also open the Scenes folder located in the Project tab in the Assets -> Scenes

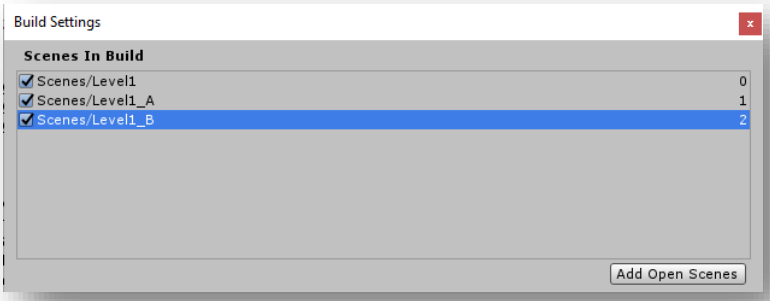


folder.

80 Click, hold, and drag the Level1 scene to the box under Scenes in Build.



81 Repeat this for the Level1_A and Level1_B scenes.



82 Adding these scenes to the Scenes in Build setting will tell Unity to load all three scenes whenever you play the game. It can now find the teleporter's destination of Level1_A!

83 Play your game and see what happens when Codey enters an activated teleporter!