CODE NINJAS®

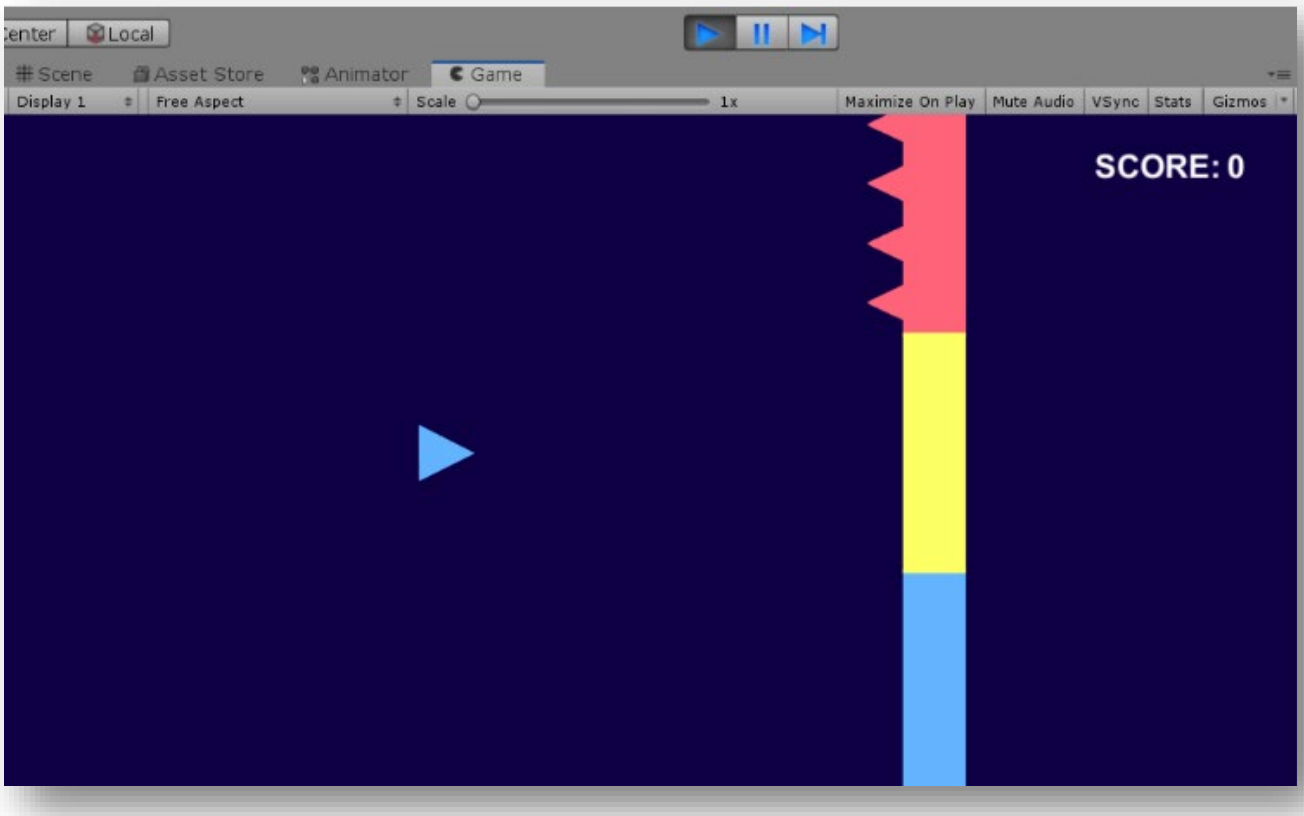
Silver Belt Ninja Guide
Activity 12: World of Color

Activity 12

World of Color

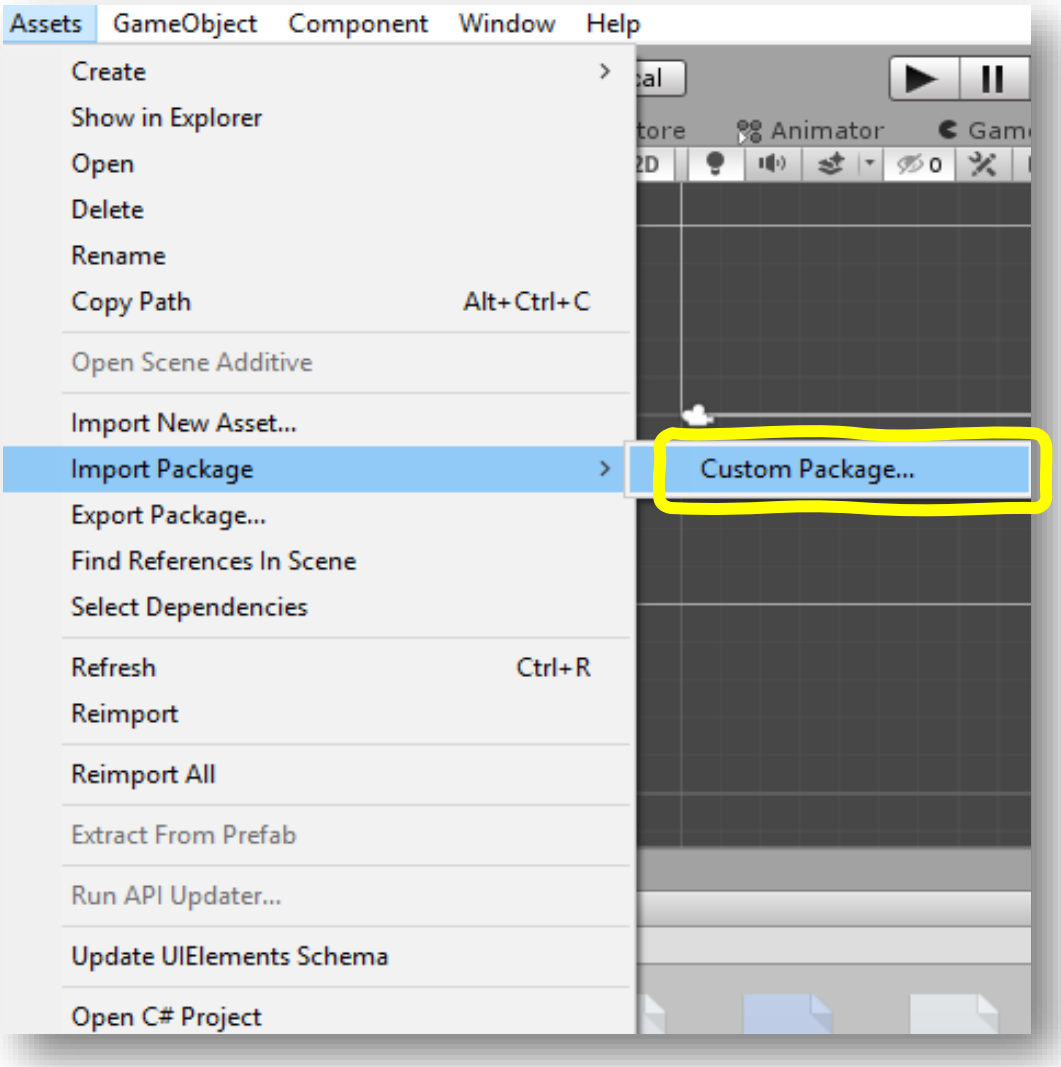
You will create game over mechanics by simulating player death using `gameObject.SetActive`. Then, a delayed level restart will be made with `Invoke` and `SceneManager`. Particle systems will be reviewed as you use critical thinking to implement their functions for a new situation.

World of Color...or is it shape? Stay alert because the rules change just as much as the player! If the player is a square, match the player color to the obstacle. If the player is a triangle, match the player shape to the obstacle.



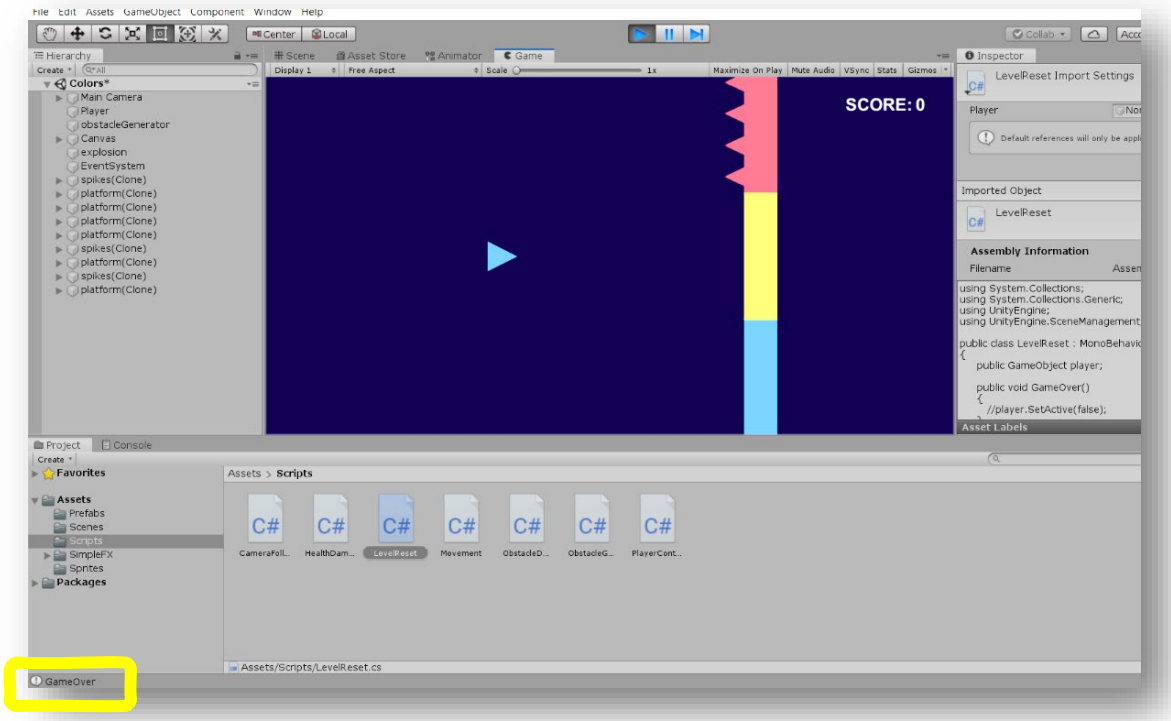
1 Start a new Unity Project and name it *YOUR INITIALS – World of Color*. Select **2D core**.

2 We've created a starter pack to give you a head start! To use it, import the **Activity 12 – World of Color.unitypackage** by going to **Assets > Import Package > Custom Package> All > Import**.

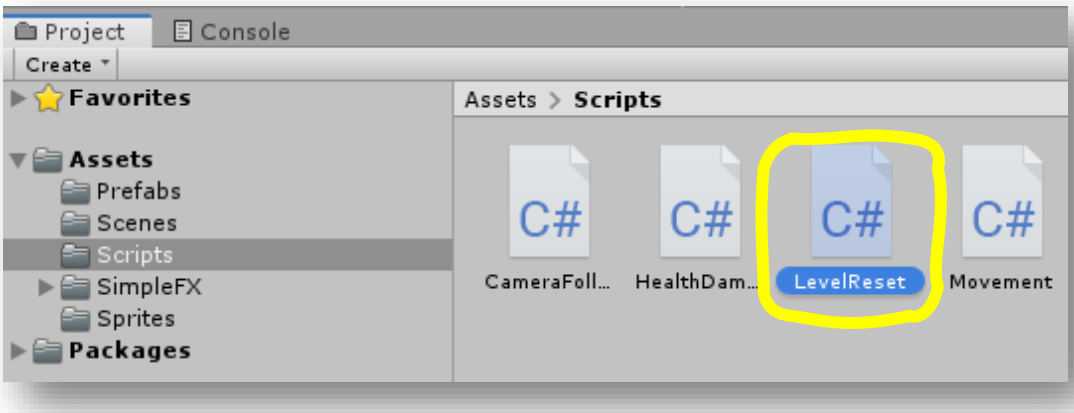


3 Open the **"Colors"** scene in the Scenes folder. Press play to test the game. There's no game over! The **console** tells you when game over should happen, but there's no player death or level reset.

Without a clear challenge and goal, the game is confusing and not fun, so let's set up a game over.



4 In the **Project** window, under the **scripts** folder, open the **LevelReset** script.



5 Here is what your LevelReset script looks like right now. There is already a new function included called `public void GameOver()`.

```
public class LevelReset : MonoBehaviour
{

    2 references
    public void GameOver()
    {

    }
}
```

Public Void

*There are two references to public void GameOver(). Like variables, functions can be public. Instead of being made visible in the inspector, public voids are made visible to **other scripts**. Let's see how this is used:*

This is a piece of th

```
if(!other.CompareTag("obstacle")){
    if(other.tag != gameObject.tag){
        LevelReset.GameOver();
```

er hits an obstacle,

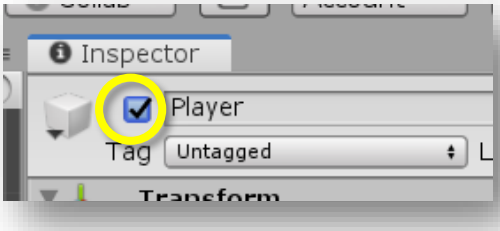
6 What should happen for game over when the player hits the wrong obstacle? There must be a clear visual indicator that the player did something wrong and can no longer play. Let's make the player disappear using `gameObject.SetActive()`.

7 In `public void GameOver()`, set player active **false**. Since the script is on the player, we can use gameObject to get a reference to the player.

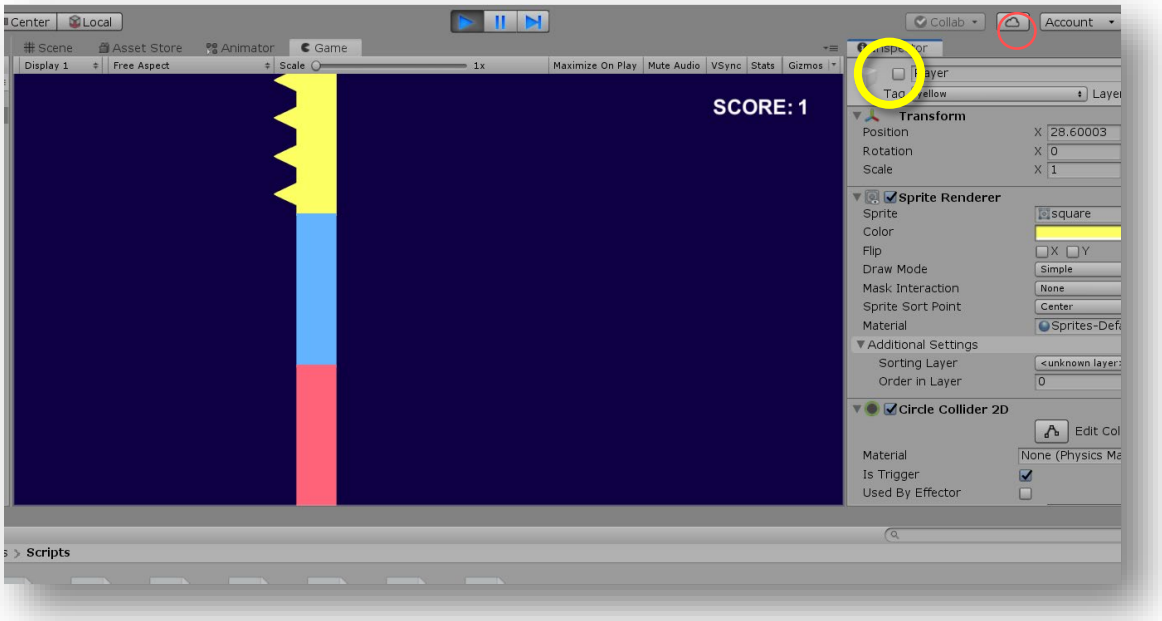
```
public void GameOver()
{
    gameObject.SetActive(false);
}
```

8

Save and close the script. In the Unity window, select the **Player** and make sure the **Inspector** is scrolled to the top. Notice the box next to the game object name.



Keep an eye on that and press play!
When you hit the wrong obstacle, what happens to the player?



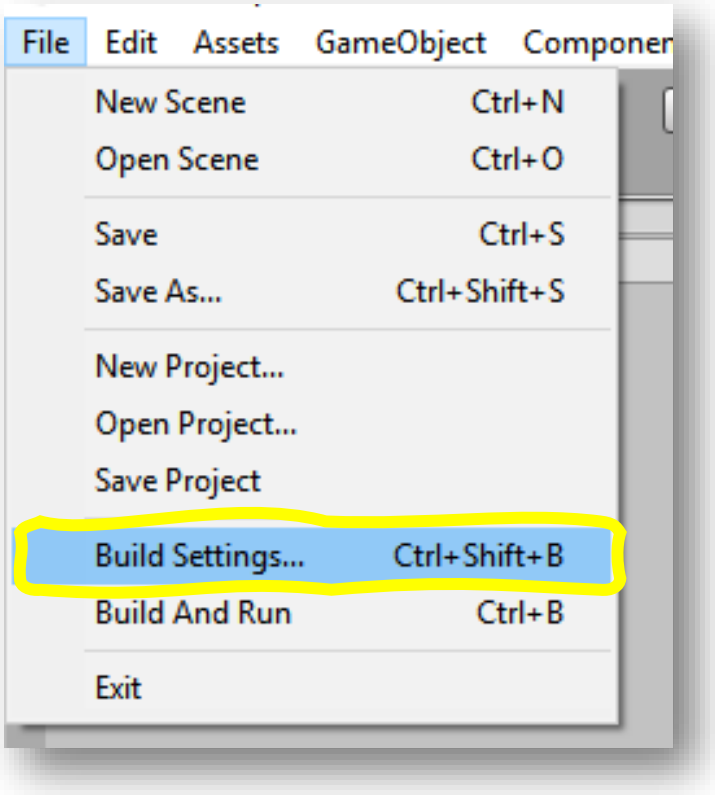
In the **Inspector**, the box becomes unchecked. This turns off everything attached to the player, including the sprite renderer (which makes the player visible), colliders, and all scripts.

9

Great job! Game over is now clear, but what if we want the game to automatically restart after player death? Everything related to restarting a level or changing between levels is handled with Unity's `Scene Management`.

Remember that scenes represent the levels.

10 Let's get our scenes setup before we start coding. Click on the **File** tab, then **Build Settings**.



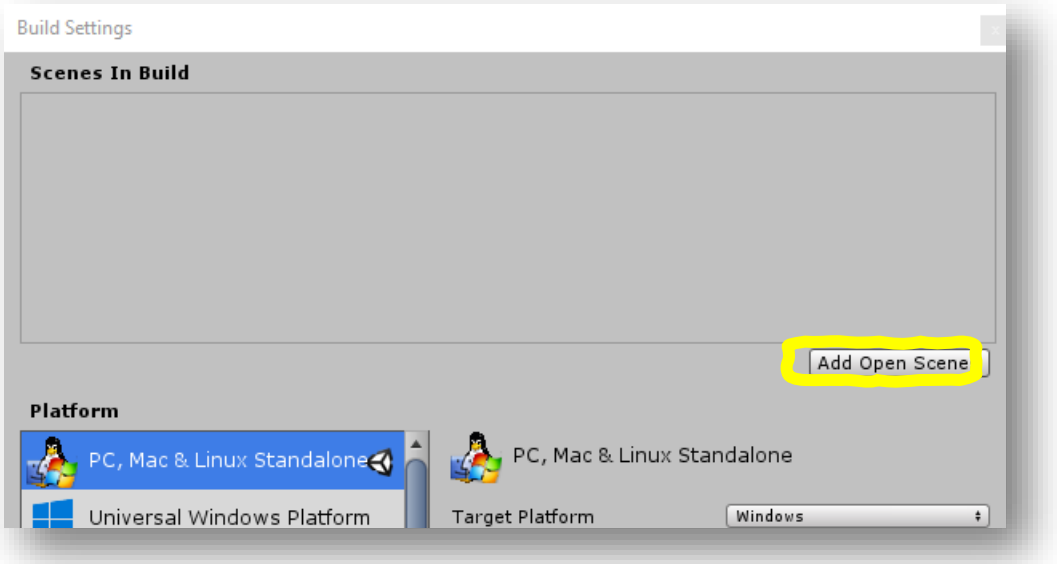
Building

Building a game is the final step. It creates an independent executable app, meaning it can be played and shared outside of Unity. The Build Settings define the settings for the game build, such as the scenes to include and the platform for your game- Windows, Mac, Linux, etc.

11 A new window will open. Focus on the top part of the window, titled **Scenes in Build**.

This tells Unity which scenes to include in the game build. Right now, the list is empty. If there are any scenes in the build, select them and press the delete key to clear them out.

To add the current scene, click on the **Add Open Scenes** button.



12 The name of the scene- Colors- will appear.

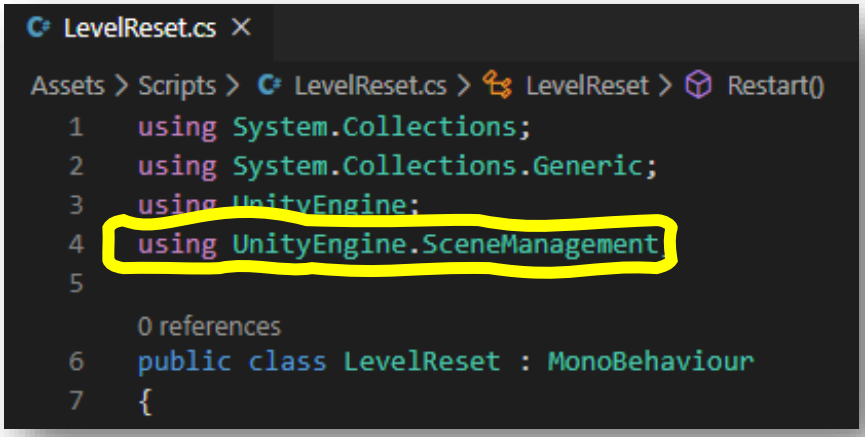
Notice on the right, there is the number 0.

This is the number of your scene.

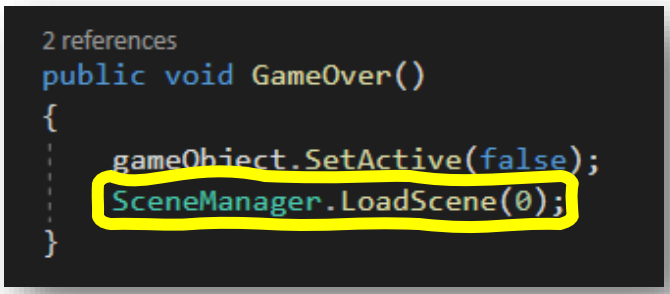
This is the only scene in our game, but if we add another scene, can you guess what its number would be?

That's right: 1! Keep in mind, whatever scene is labeled 0 is the scene that is loaded when the game starts once built.

13 With the scene set up in the build settings, we can now use it in our code! Open the **LevelReset** script. Add the Unity namespace **SceneManagement** to the top of your code. This stores functions related to handling scenes.



14 To load or reload a scene, we will use `SceneManager.LoadScene()`. This looks at the build settings and reloads the scene with the number you give it. Remember, Unity starts counting scenes at 0 not 1. The Colors scene is scene 0, so we want the line to read:



15 **Save** your script and press play.

16 You did it, your level restarts on player death! But it restarts exactly on player death. This seems too fast for the person playing to realize it was game over and to get ready for the restart. `Invoke()` will solve this.

Invoke takes two parameters: ("functionName", time). It waits the set time before calling the function.

17 We need a new function to place the reload in. First delete the previously included `SceneManager.LoadScene()`. After `public void GameOver()`, add a function called `Reload()`:

```
2 references
public void GameOver()
{
    gameObject.SetActive(false);
}

0 references
void Reload()
{

}
```

18 In void `Reload()`, add `SceneManager.LoadScene(0)`:

```
void Reload()
{
    SceneManager.LoadScene(0);
}
```

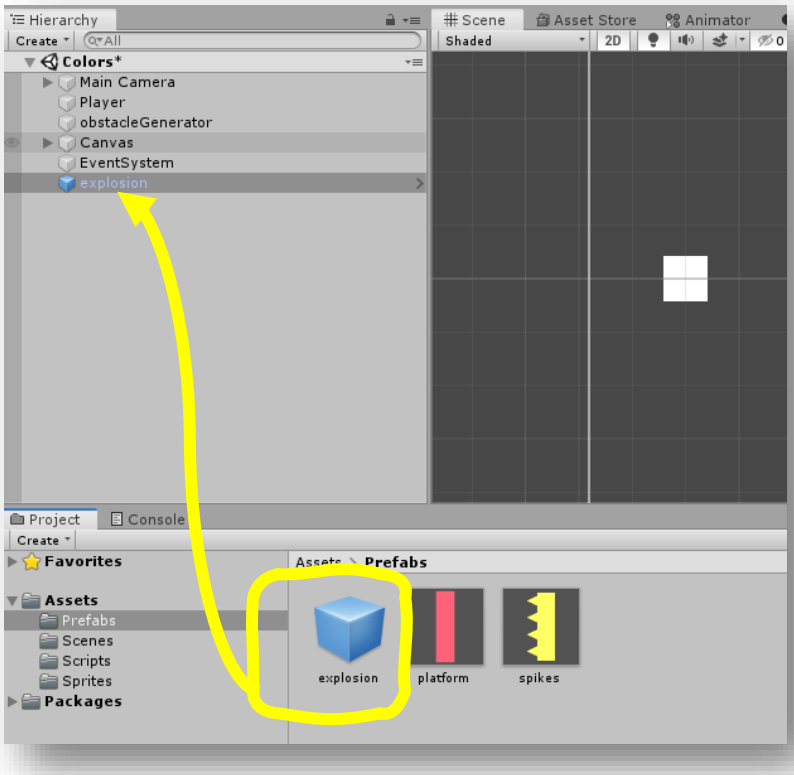
19 Remember `public void GameOver()` is called when the player hits an obstacle. To start the invoke timer at player death, include it in `GameOver()`. We will have the Invoke wait two frames:

```
public void GameOver()
{
    gameObject.SetActive(false);
    Invoke("Reload", 2);
}

0 references
void Reload()
{
    SceneManager.LoadScene(0);
}
```

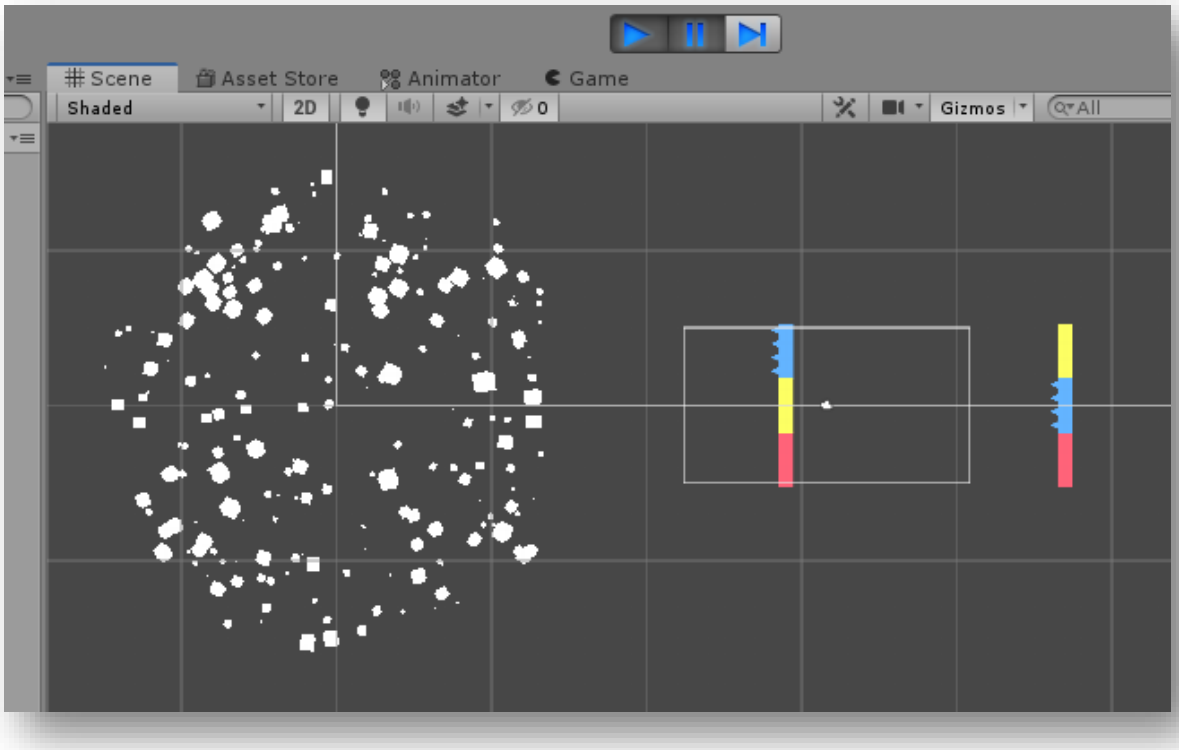
20 Press **play** and test out the game.

21 Amazing job! Can we make it even better? Right now, the player just disappears when it hits an obstacle; what if we add a particle system? Particle systems are a great way to catch people's attention and adds that extra element of fun and exaggeration. In your **Project** window, under the **Prefab** folder, drag the **explosion** prefab into the **Hierarchy**.



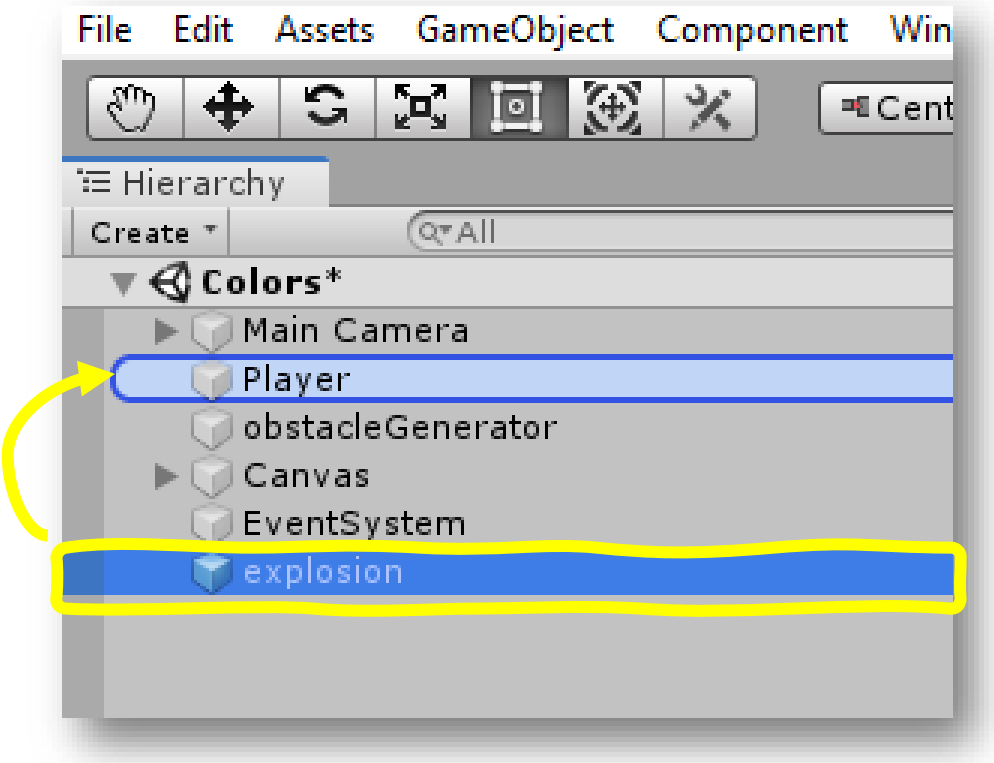
22

Let's think, the explosion needs to play on player death, but the player is constantly moving forward. As is, our explosion will just stay in the same position. The picture below shows this:

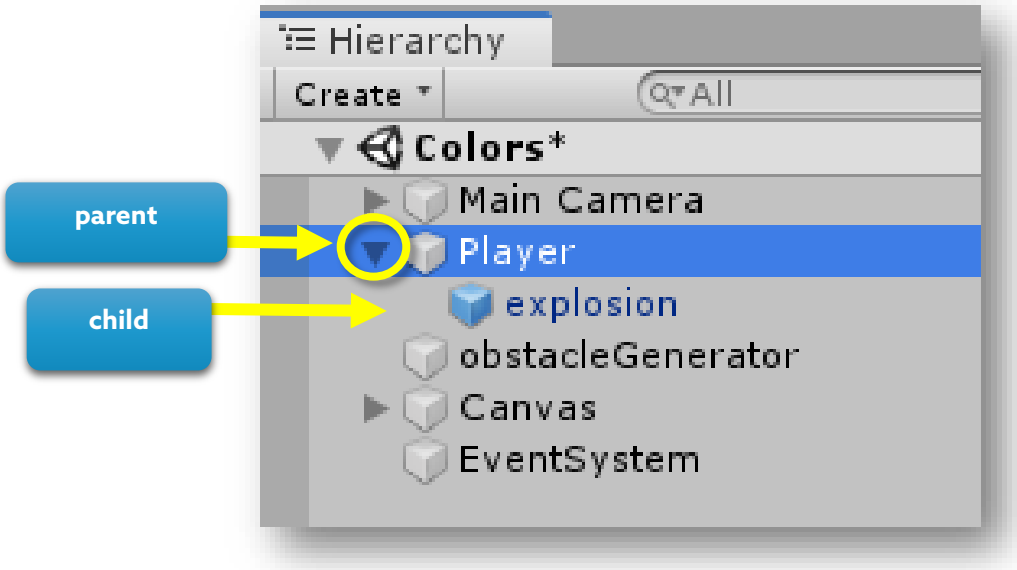


*How did we test this? When you press play, make sure you switch to the **Scene** tab. Then, scroll out so you can see the full scene. Wait until the player hits the wrong obstacle. If need be, you can press Pause to pause the game, and then the Skip to move forward one frame.*

23 How do we get the explosion to follow the player? Remember, the **Hierarchy** is laid out like a family tree- with parent and child objects. Child objects follow the parent. Click and drag the **explosion** on top of **Player**, let go when there is a blue box around Player, like so:



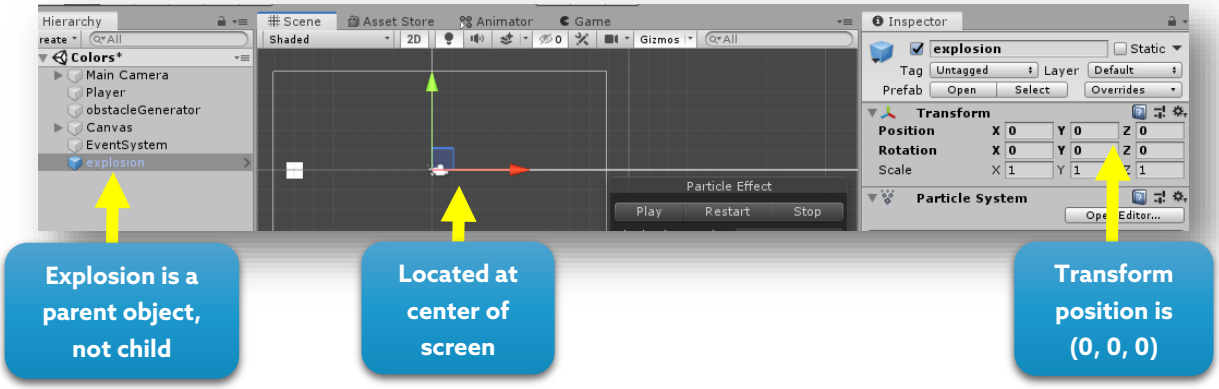
A grey triangle will appear to the left of **Player**. If you click on it, you will see Player's hierarchy, with the explosion as a child object.

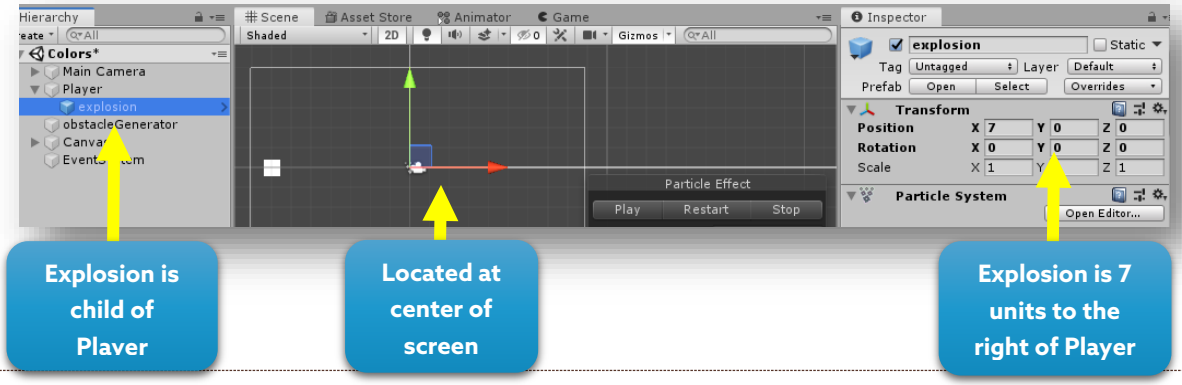


24

Now the explosion will move forward with the player. We still need to adjust one more thing! Click on explosion and look in the **Inspector** at its **Transform**. When you place an object in the scene, its transforms are based on the screen, with 0 being the center. When an object becomes a child, its transforms are based on the parent, so 0 becomes the center of the parent.

The pictures below illustrate how an object's transforms change when it becomes a child:





25 We need the **explosion** to be at the center of **Player**, so make sure its **Transforms** are as follows:

26 Press play. The explosion is now at the center of Player, but still only plays once right at the start.

Remember `ParticleSystem.Stop()` and `.Play()`?

Let's use those to stop the explosion at Start and play it at Game Over.

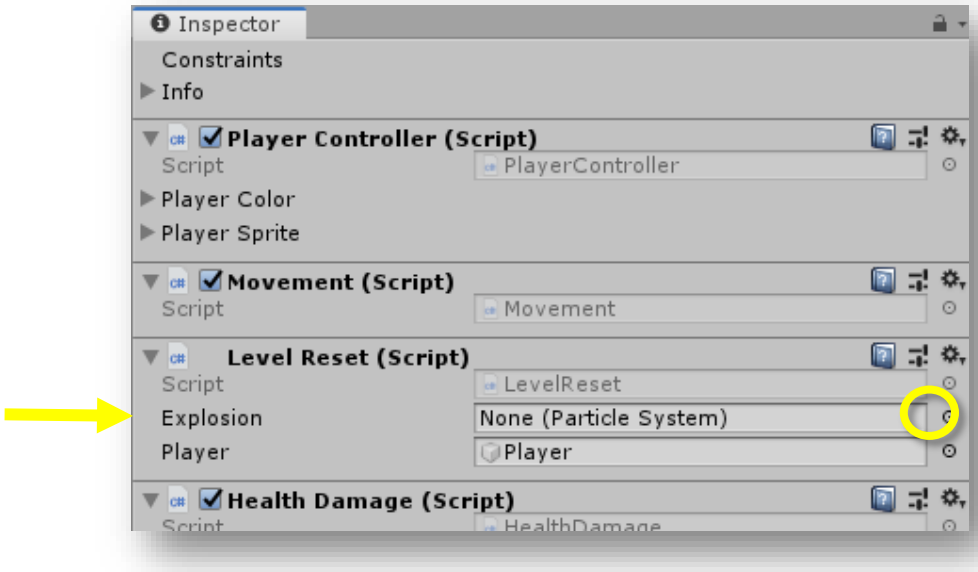
Open your **LevelReset** script.

27 Declare the **particle system explosion**:

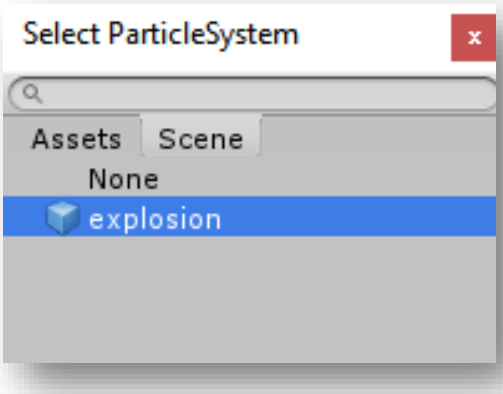
```
public class LevelReset : MonoBehaviour
{
    public ParticleSystem explosion;

    2 references
    public void GameOver()
    {
```

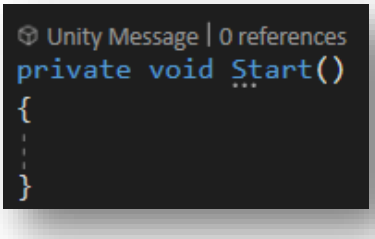
28 Save your script. Switch back to the Unity window. In the **Hierarchy**, click on **Player**. In the **Inspector**, scroll down to the **Level Reset** script component. Click on the circle next to the empty **Explosion** slot:



29 In the pop-up window, select **explosion**.



30 Open the **LevelReset** script. We did not want the particle to play at the start of the game. So which function do we need to add back into our script? `void Start()`:



31 In `void Start()`, stop the particle system from playing:

```
void Start()
{
    explosion.Stop();
}
```

32 Where should we put `explosion.Play()`? Think about when we want it to play. Add it to your code:

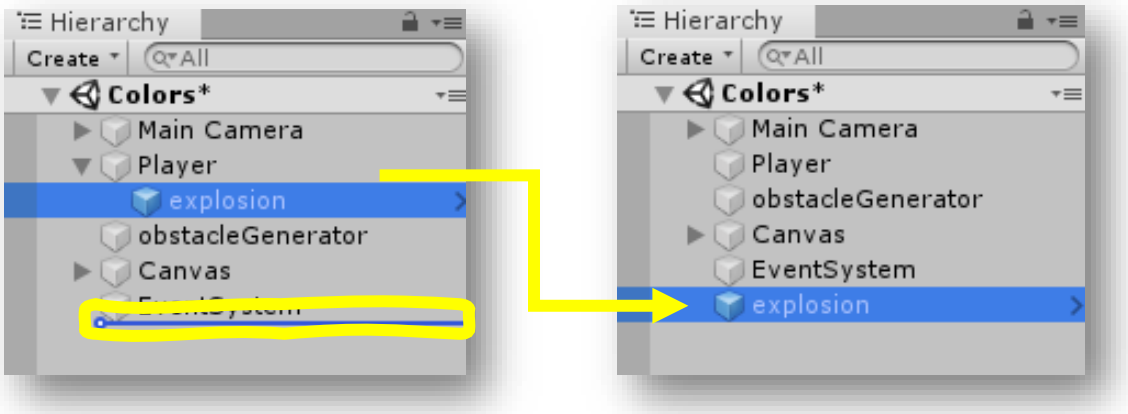
```
2 references
public void GameOver()
{
    gameObject.SetActive(false);
    Invoke("Reload", 2);
    explosion.Play();
}
```

33 Press play to test that the explosion only plays on player death.

34 It's not playing the explosion! Our code makes sense, but it's not working. Brainstorm some reasons why. I'll give you a hint: `SetActive()`.

Explosion was made a child of Player so it would follow its transforms, which it does! But remember adding `player.SetActive(false)`? The child follows the parent: if the parent is set to false, so is the child. This means the particle system is set to play, but at the same time deactivated. To fix this, we must code the player follow.

35 Unparent explosion from Player. To do this, click and drag on **explosion**. Do not release it on top of another object (if you do, hit ctrl and z on your keyboard to undo). Release when there is a blue line, like so:



36 Open the **LevelReset** script. Add a line below the Invoke line. This will set the position of the explosion to the player before we play it.

```
2 references
public void GameOver()
{
    gameObject.SetActive(false);
    Invoke("Reload", 2);
    explosion.transform.position = transform.position;
    explosion.Play();
}
```

37 **Save** your script. Press **play** and test out your game.

38 Great problem solving, Ninja! Your game over code has made the game more fun and understandable! Test out your hard work. Can you keep up with the constantly changing World of Color?