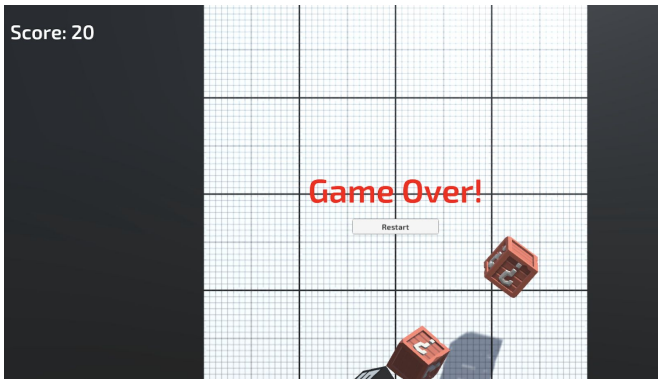
5.3 Game Over

Steps:

Example of project by end of lesson



Length: 60 minutes

Overview: We added a great score counter to the game, but there are plenty of other game-changing UI elements that we could add. In this lesson, we will create some “Game Over” text that displays when a “good” target object drops below the sensor. During game over, targets will cease to spawn and the score will be reset. Lastly, we will add a “Restart Game” button that allows the player to restart the game after they have lost.

Project Outcome: When a “good” target drops below the sensor at the bottom of the screen, the targets will stop spawning and a “Game Over” message will display across the screen. Just underneath the “Game Over” message will be a “Reset Game” button that reboots the game and resets the score, so the player can enjoy it all over again.

Learning Objectives: By the end of this lesson, you will be able to:
- Make UI elements appear and disappear with .SetActive
- Use Script Communication and Game states to have a working “Game Over” screen
- Restart the game using a UI button and Scene Management

Step 1: Create a Game Over text object

If we want some "Game Over" text to appear when the game ends, the first thing we'll do is create and customize a new UI text element that says "Game Over".

1. Right-click on the **Canvas**, create a new *UI* > ***TextMeshPro - Text*** object, and rename it "Game Over Text"
2. In the inspector, edit its **Text**, **Pos X**, **Pos Y**, **Font Asset**, **Size**, **Style**, **Color**, and **Alignment**
3. Set the "Wrapping" setting to "Disabled"

- **Tip:** The center of the screen is the best place for this Game Over message - it grabs the player's attention

Step 2: Make GameOver text appear

We've got some beautiful Game Over text on the screen, but it's just sitting and blocking our view right now. We should deactivate it, so it can reappear when the game ends.

1. In GameManager.cs, create a new ***public TextMeshProUGUI gameOverText;*** and assign the **Game Over** object to it in the inspector
2. **Uncheck** the Active checkbox to **deactivate** the Game Over text by default
3. In ***Start()***, activate the Game Over text

- **Don't worry:** We're just doing this temporarily to make sure it works

```
public TextMeshProUGUI gameOverText;

void Start() {
  ...
  gameOverText.gameObject.SetActive(true); }
```

Step 3: Create GameOver function

We've temporarily made the "Game Over" text appear at the start of the game, but we actually want to trigger it when one of the "Good" objects is missed and falls.

1. Create a new ***public void GameOver()*** function, and **move** the code that activates the game over text inside it
2. In Target.cs, call ***gameManager.GameOver()*** if a target collides with the **sensor**
3. Add a new "Bad" tag to the **Bad object**, add a condition that will only trigger game over if it's *not* a bad object

```
void Start() {
  ... gameOverText.gameObject.SetActive(true); }

public void GameOver() {
  gameOverText.gameObject.SetActive(true); }

<------>
private void OnTriggerEnter(Collider other) {
  Destroy(gameObject);
  if (!gameObject.CompareTag("Bad")) { gameManager.GameOver(); } }
```

Step 4: Stop spawning and score on GameOver

The "Game Over" message appears exactly when we want it to, but the game itself continues to play. In order to truly halt the game and call this a "Game Over', we need to stop spawning targets and stop generating score for the player.

1. Create a new ***public bool isGameActive;***
2. As the **first line** In ***Start()***, set ***isGameActive = true;*** and in ***GameOver()***, set ***isGameActive = false;***
3. To prevent spawning, in the ***SpawnTarget()*** coroutine, change ***while (true)*** to ***while (isGameActive)***
4. To prevent scoring, in Target.cs, in the ***OnMouseDown()*** function, add the condition ***if (gameManager.isGameActive) {***

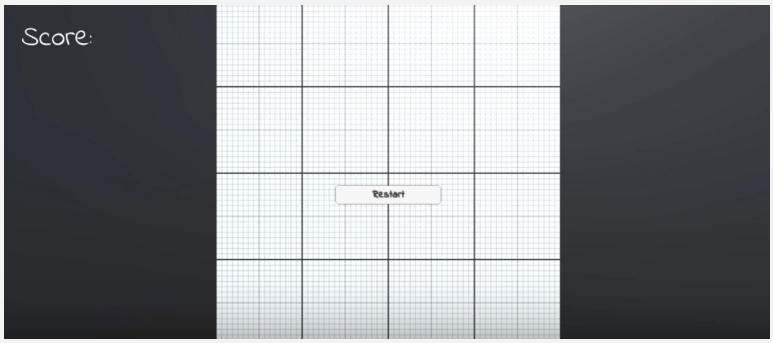
```
public bool isGameActive;

void Start() {  ... isGameActive = true;  }

public void GameOver() {  ... isGameActive = false; }

IEnumerator SpawnTarget() {  while (true isGameActive) {  ...  }
<------>
private void OnMouseDown() {
  if (gameManager.isGameActive) { ... [all function code moved inside]  }}
```

Step 5: Add a Restart button

Our Game Over mechanics are working like a charm, but there's no way to replay the game. In order to let the player restart the game, we will create our first UI button

1. Right-click on the **Canvas** and *Create > UI > Button*
2. Rename the button "Restart Button"
3. Temporarily **reactivate** the Game Over text in order to reposition the Restart Button nicely with the text, then **deactivate** it again
4. Select the Text child object, then edit its **Text** to say "Restart"**,** its **Font, Style,** and **Size**

- **New Concept:** Buttons



Step 6: Make the restart button work

We've added the Restart button to the scene and it LOOKS good, but now we need to make it actually work and restart the game.

1. In GameManager.cs, add ***using UnityEngine.SceneManagement;***
2. Create a new ***void RestartGame()*** function that reloads the current scene
3. In the **Button's** inspector, click **+** to add a new **On Click event**, drag it in the **Game Manager** object and select the ***GameManager.RestartGame*** function

- **New Concept:** Scene Management
- **New Concept:** On Click Event
- **Don't worry:** The restart button is just sitting there for now, but we will fix it later

```
using UnityEngine.SceneManagement;

void RestartGame() {
  SceneManager.LoadScene(SceneManager.GetActiveScene().name); }
```

Step 7: Show restart button on game over

The Restart Button looks great, but we don't want it in our faces throughout the entire game. Similar to the "Game Over" message, we will turn off the Restart Button while the game is active.

1. At the top of GameManager.cs add ***using UnityEngine.UI;***
2. Declare a new ***public Button restartButton;*** and assign the **Restart Button** to it in the inspector
3. **Uncheck** the "Active" checkbox for the **Restart Button** in the inspector
4. In the ***GameOver*** function, activate the **Restart Button**

- **Tip:** Adding "using UnityEngine.UI" allows you to access the Button class

```
using UnityEngine.UI;

public Button restartButton;

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

Lesson Recap

New Functionality	• A functional Game Over screen with a Restart button • When the Restart button is clicked, the game resets
New Concepts and Skills	• Game states • Buttons • On Click events • Scene management Library • UI Library • Booleans to control game states
Next Lesson	• In our next lesson, we'll use buttons to really add some difficulty to our game