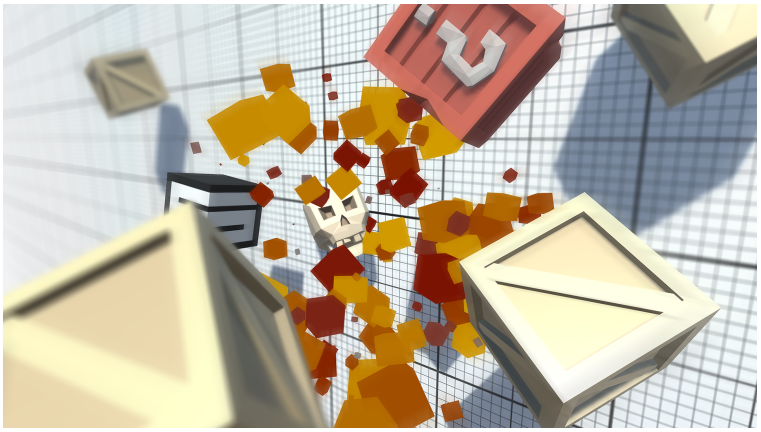
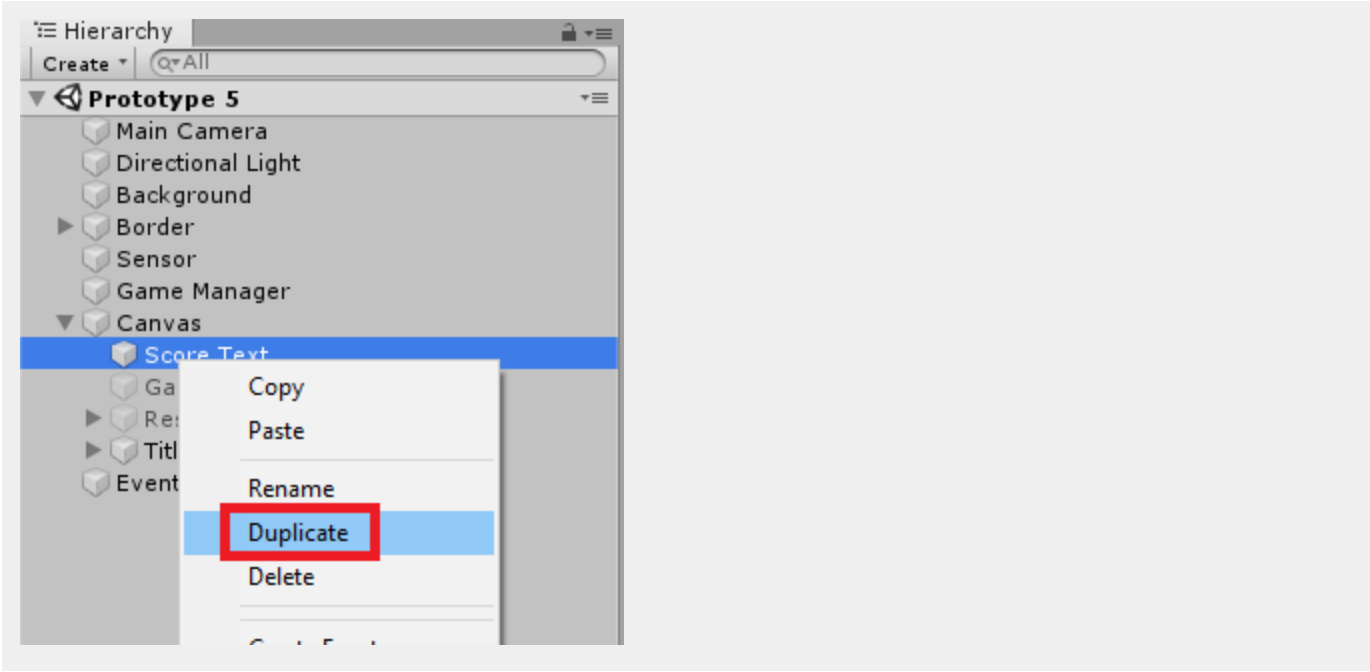
Bonus Features 5

Solution Walkthrough

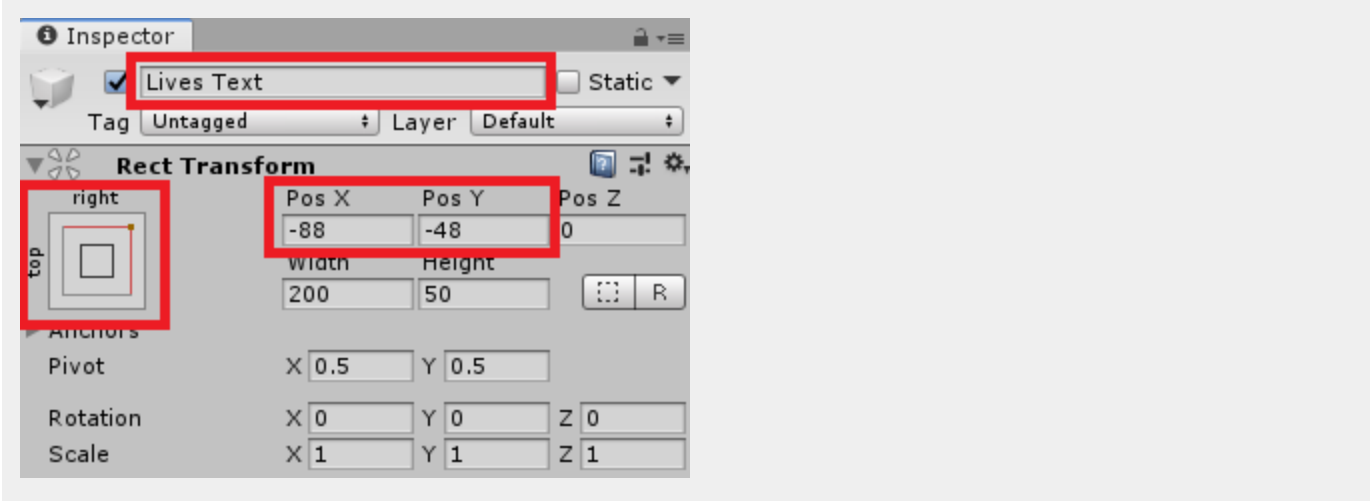


Easy - Lives UI

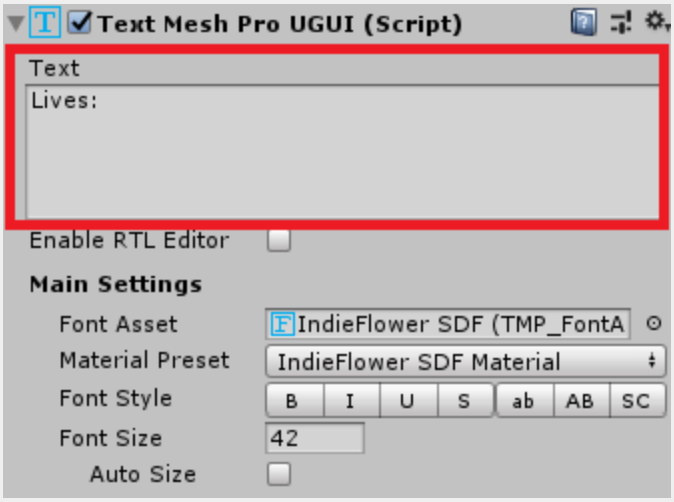
1. Duplicate the *Score Text* GameObject by right-clicking on it in the Hierarchy and selecting **Duplicate**



2. Rename the GameObject to "*Lives Text*" and align it to the Top-right of the screen.



3. Change the **Text** field in the **TextMeshPro - Text (UI)** component to "Lives:"



4. Open up “GameManager.cs” from the scripts folder and add two new variables - one for the text and the other for the amount of lives available.

```
public TextMeshProUGUI livesText;
private int lives;
```

5. Just after the **UpdateScore** method, add a new method called **UpdateLives**, this will adjust the amount of lives we have and then display the result to the Text GameObject we created earlier. It will also check to see if we have run out of lives, if we have it will call the **GameOver** method.

```
public void UpdateLives(int livesToChange)
{
    lives += livesToChange;
    livesText.text = "Lives: " + lives;
    if (lives <= 0)
    {
        GameOver();
    }
}
```

6. To make sure the text appears when we start the game, update the **StartGame** method to include the new method we created. Save the script and return to Unity.

```
public void StartGame(int difficulty)
{
    spawnRate /= difficulty;

    isGameActive = true;

    StartCoroutine(SpawnTarget());
    score = 0;
    UpdateScore(0);
    UpdateLives(3);
```

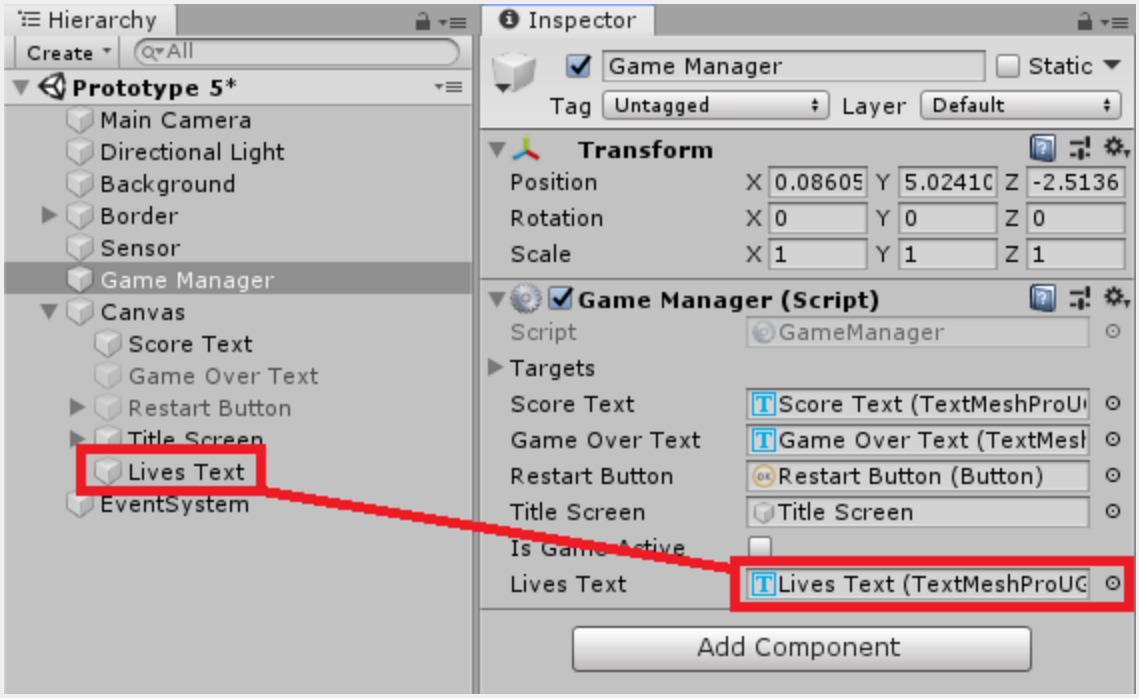
```
    titleScreen.gameObject.SetActive(false);
}
```

7. Next open up the **Target.cs** script. In the **OnTriggerEnter** method, we need to change what is being called. You will need to change what is called when an object collides with the bottom sensor and make sure it is only called when the game is active so that you don't end up with negative lives.

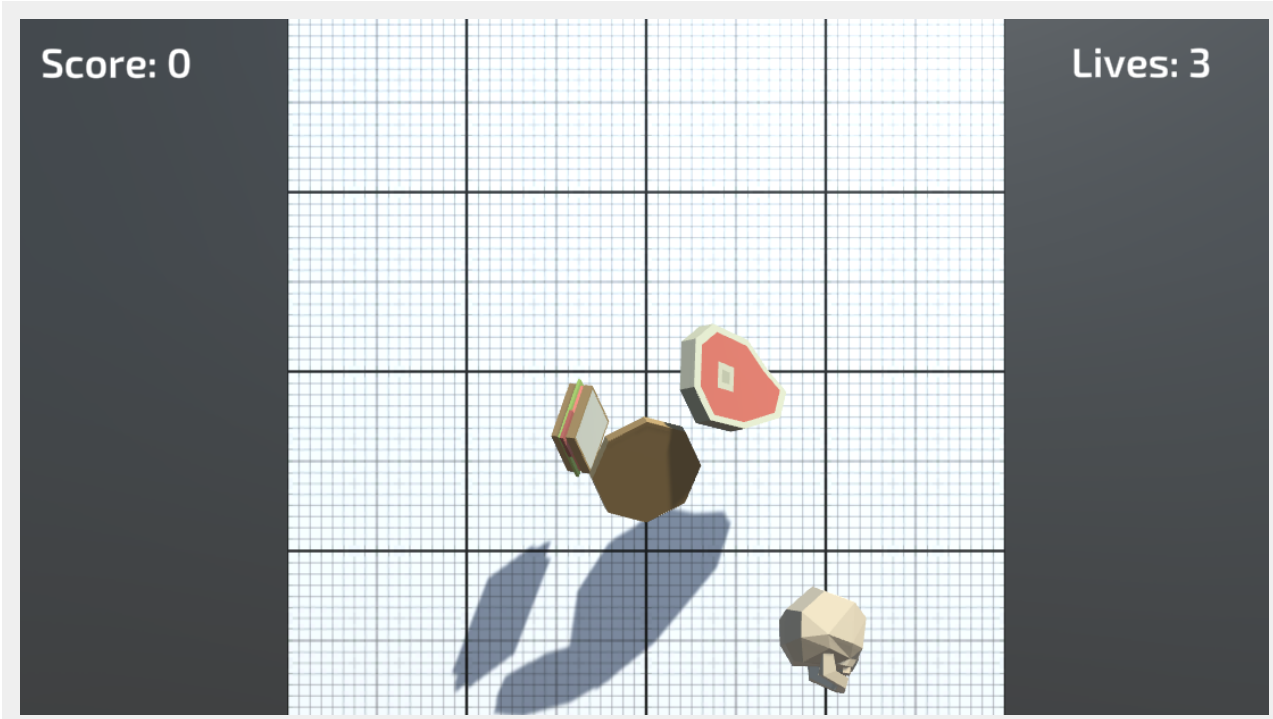
```
private void OnTriggerEnter(Collider other)
{
    Destroy(gameObject);

    if (!gameObject.CompareTag("Bad") && gameManager.isGameActive)
    {
        gameManager.GameOver();
        gameManager.UpdateLives(-1);
    }
}
```

8. Save the script and return to Unity. We now need to set up our GameManager in the Inspector. Select the GameManger in the Hierarchy to display all the components in the Inspector. Drag the Lives Text that was created earlier into the Lives Text field on the **GameManager (Script)** component.

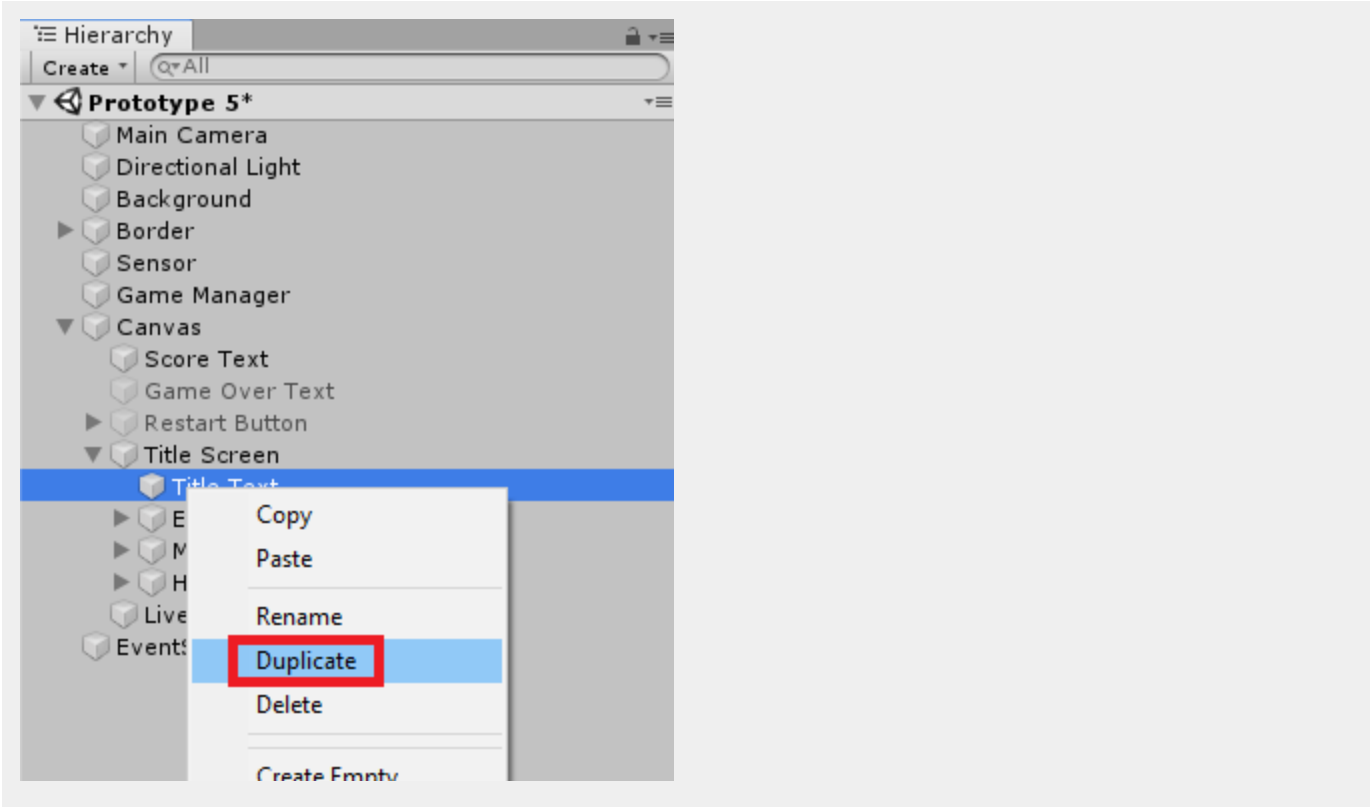


9. Save the scene and play it. Notice that when the objects fall off the screen, the lives counter goes down.

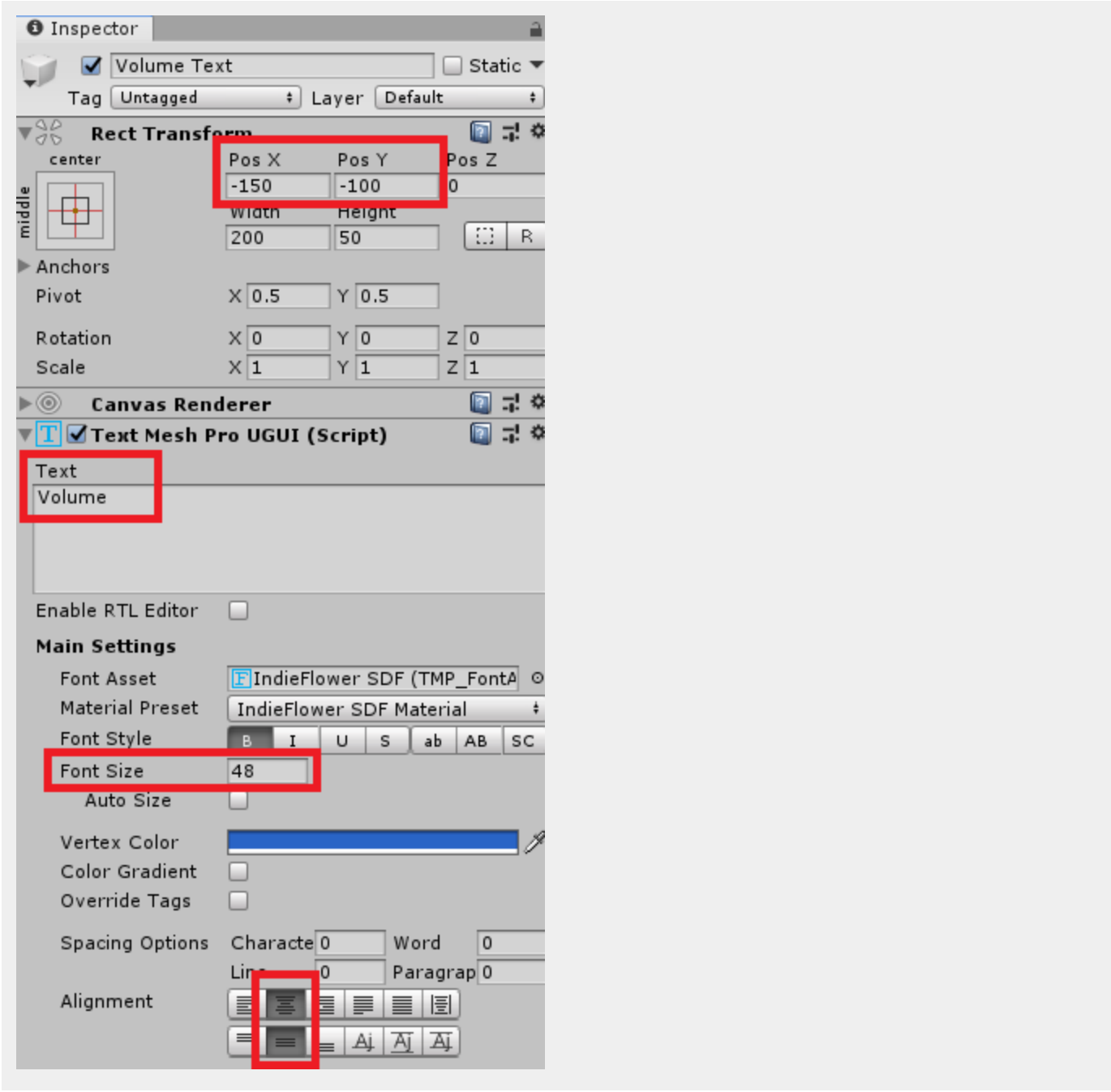

Score: 0
Lives: 3

Medium - Music Volume

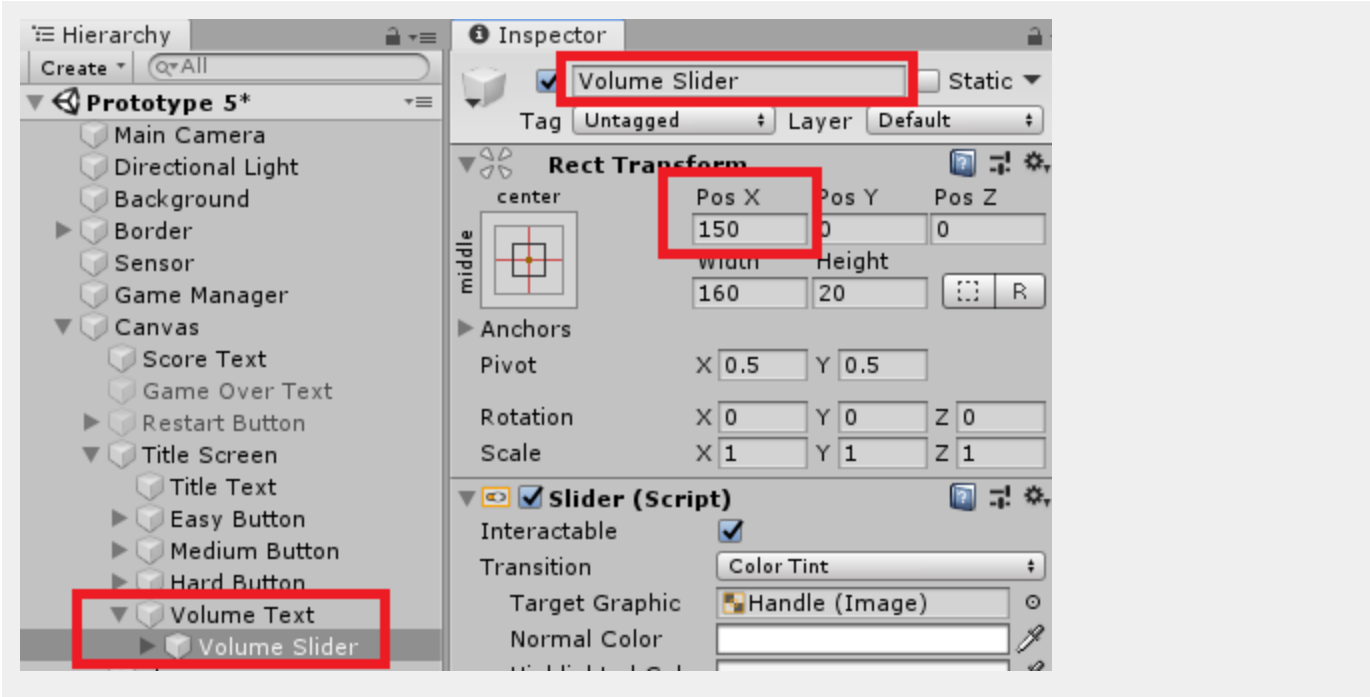
1. In the Hierarchy, navigate to **Canvas** > **Title Screen** > **Title Text**. Right-click the *"Title Text"* GameObject and select **Duplicate**.



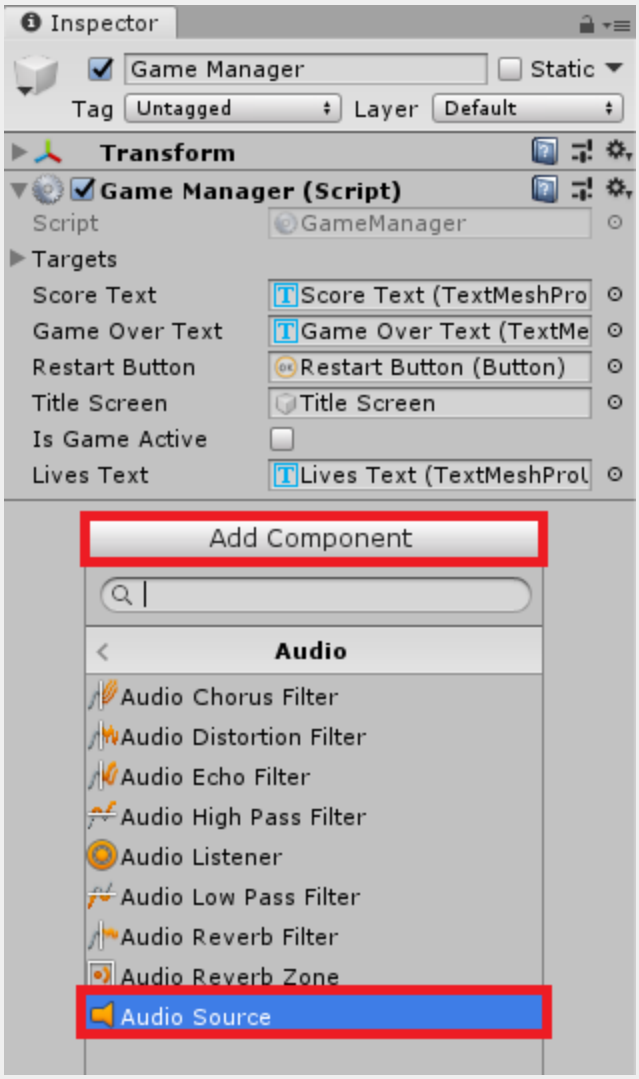
2. Rename the new Text GameObject to "*Volume Text*". Adjust the Text field of the **TextMeshPro - Text (UI)** component to say "*Volume*". Adjust the **font size** to **36**. Ensure the **Alignment** is set to **Middle** and **Center**. Position the text object below the buttons for playing the game



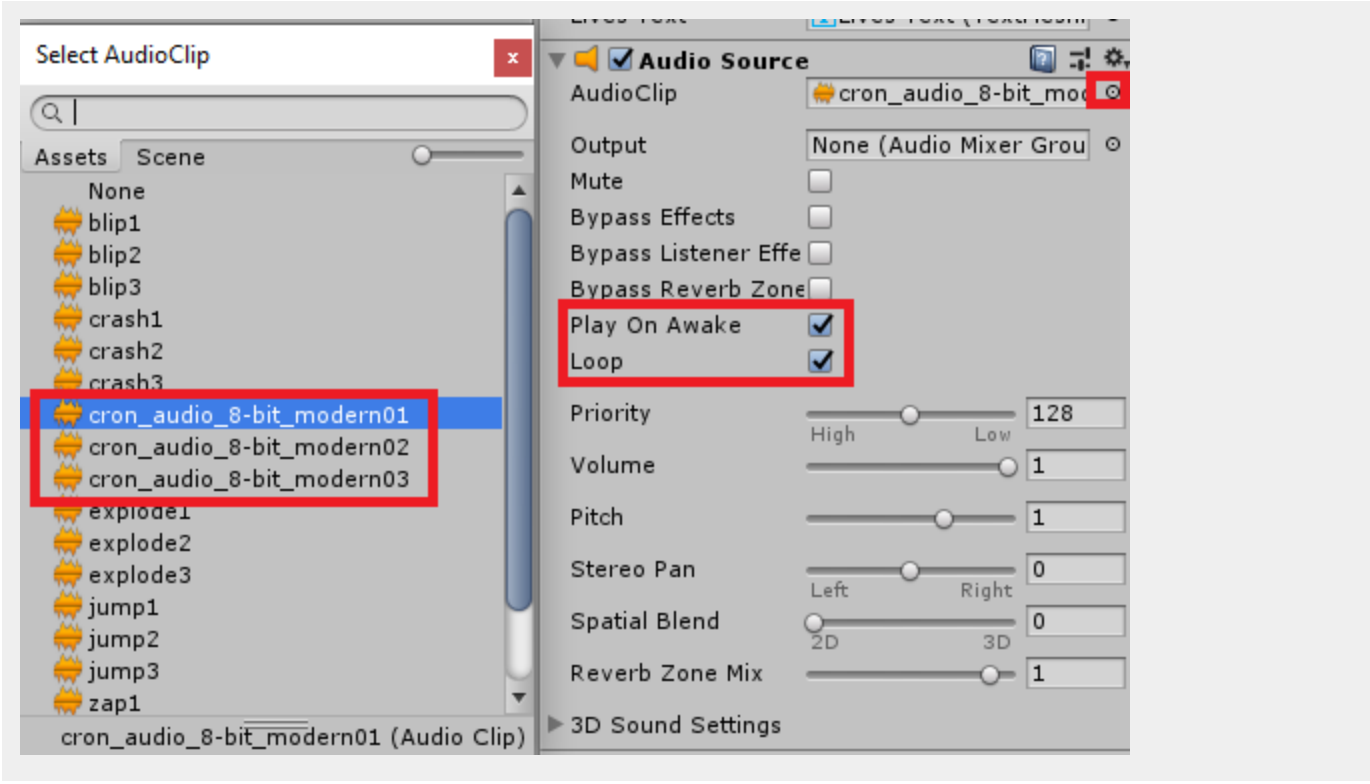
3. Right-click on the Volume Text GameObject and select **UI** > **Slider**. This will make the slider a child of the text GameObject. Rename the slider to "*Volume Slider*" and adjust the position so that it is to the right of the Text. At this point, you could also adjust the color of the slider GameObjects.



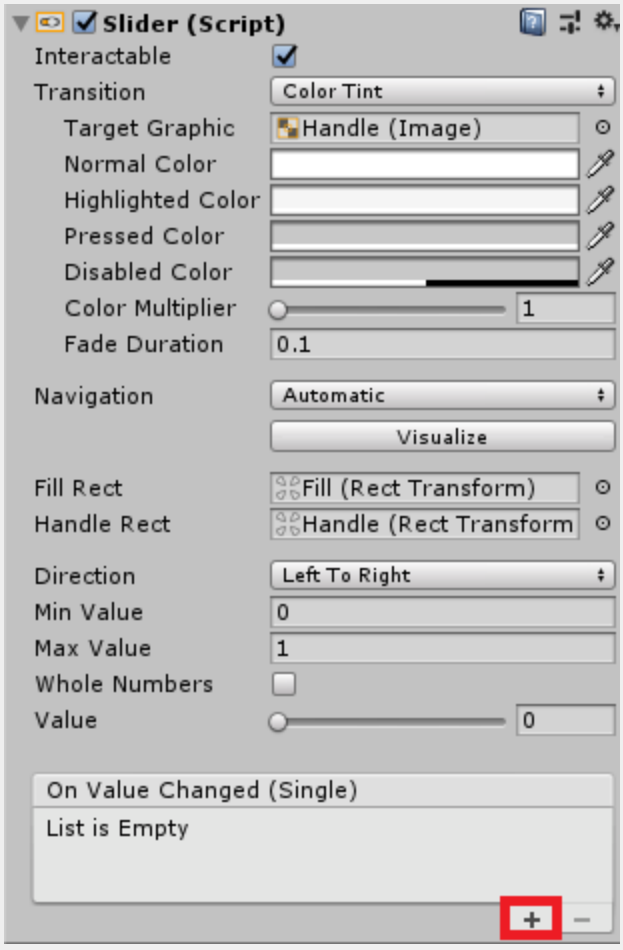
4. Select the GameManager in the Hierarchy. Add an Audio Source by clicking **Add Component** > **Audio** > **Audio Source**



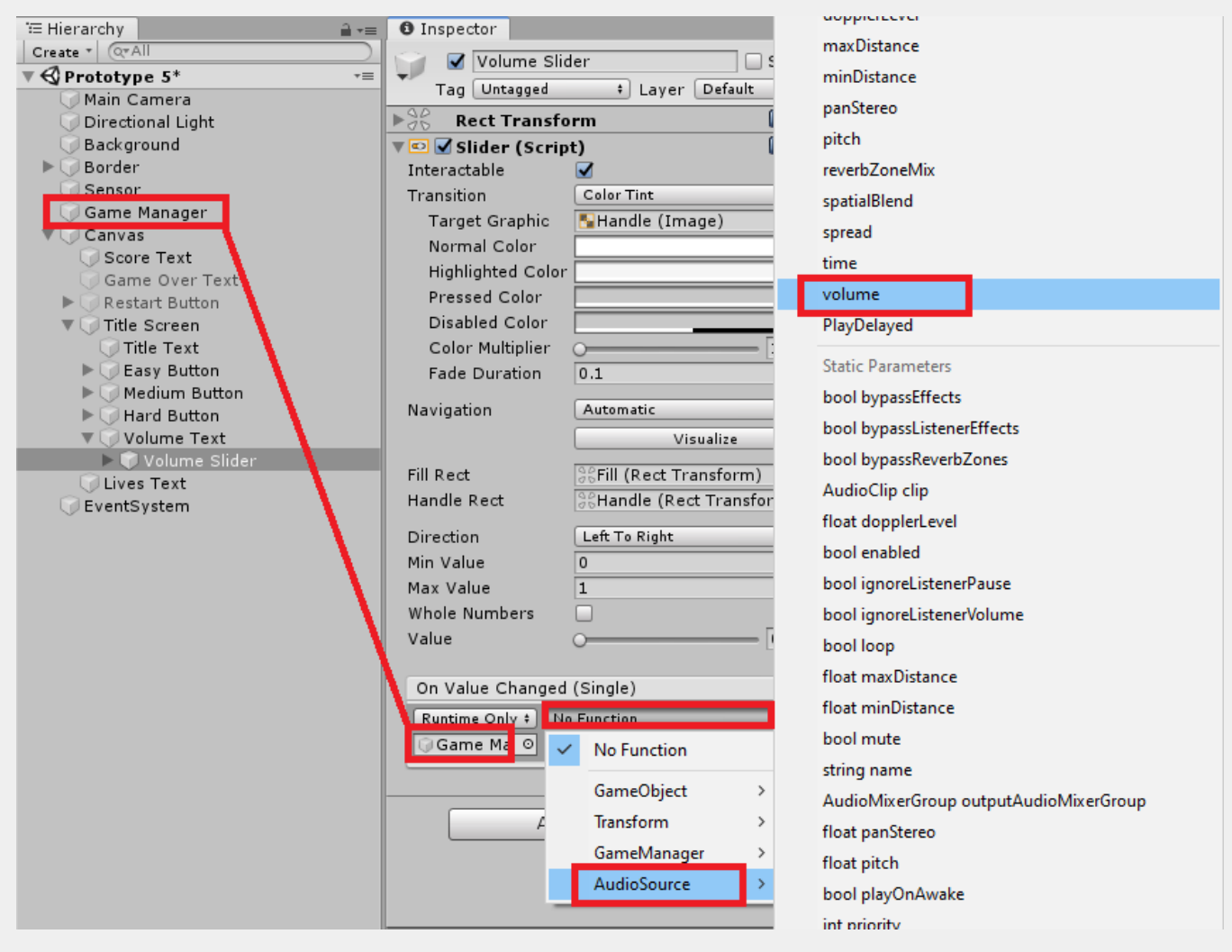
5. Add an **Audio Clip** to the **Audio Source** by clicking the selector tool next to the field. Select one of the three clips shown in the image below. Ensure that **Play On Awake** and **Loop** are both checked.



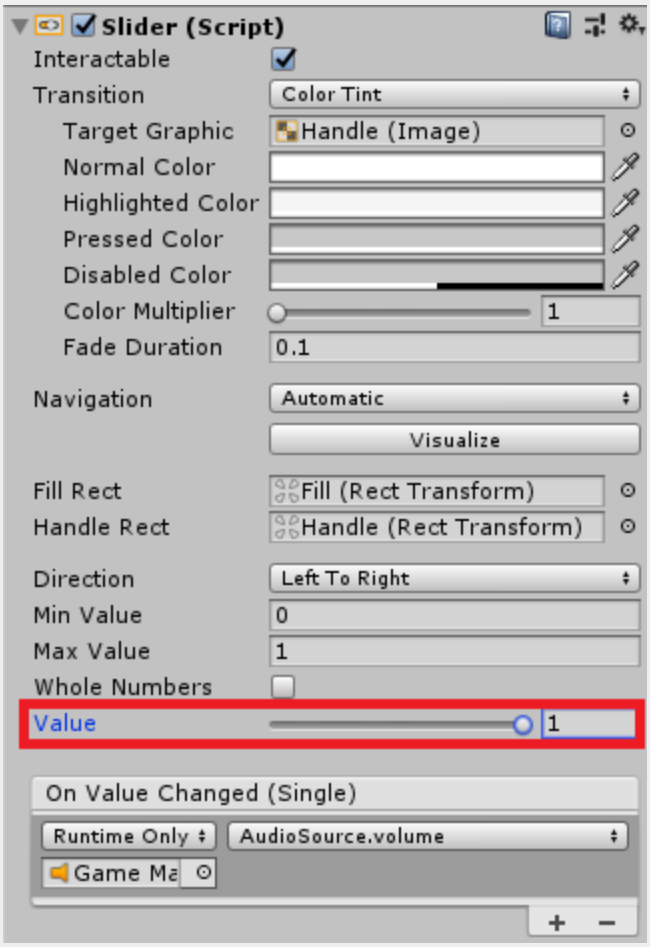
6. Select the *Volume Slider* in the Hierarchy, We will now set up the Slider functionality. At the bottom of the **Slider** component, Click the **+** on the OnValueChanged Event.



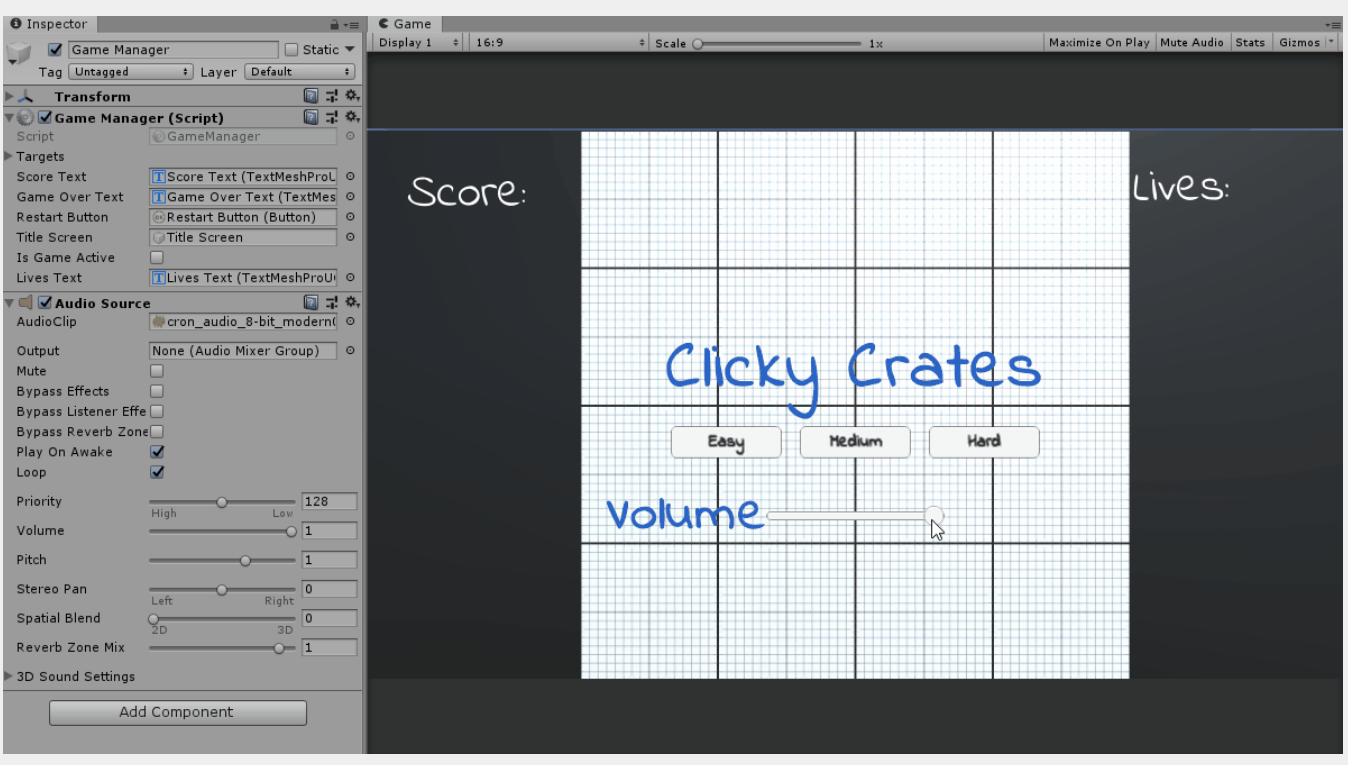
7. Drag the *Game Manager* from the Hierarchy into the **None (Object)** Field. From the **No Function** dropdown, select **Audio Source** > **Volume**.



8. The last thing you need to do is set the **Value** of the Slider. We put it to **1**, as that is the default volume of the Audio Source.

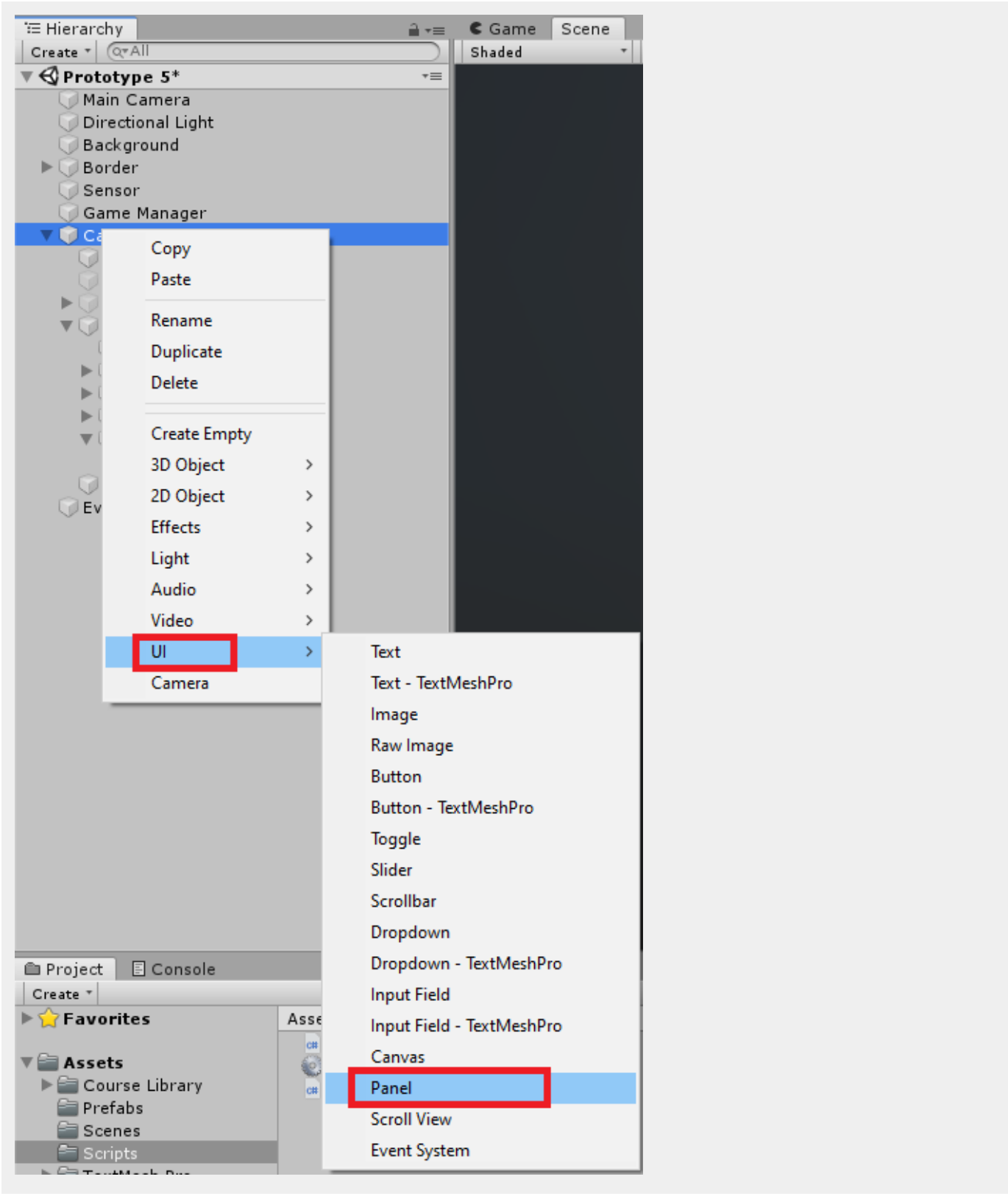


9. Save the scene and press play. The volume property on the Audio Source will change when you adjust the slider.

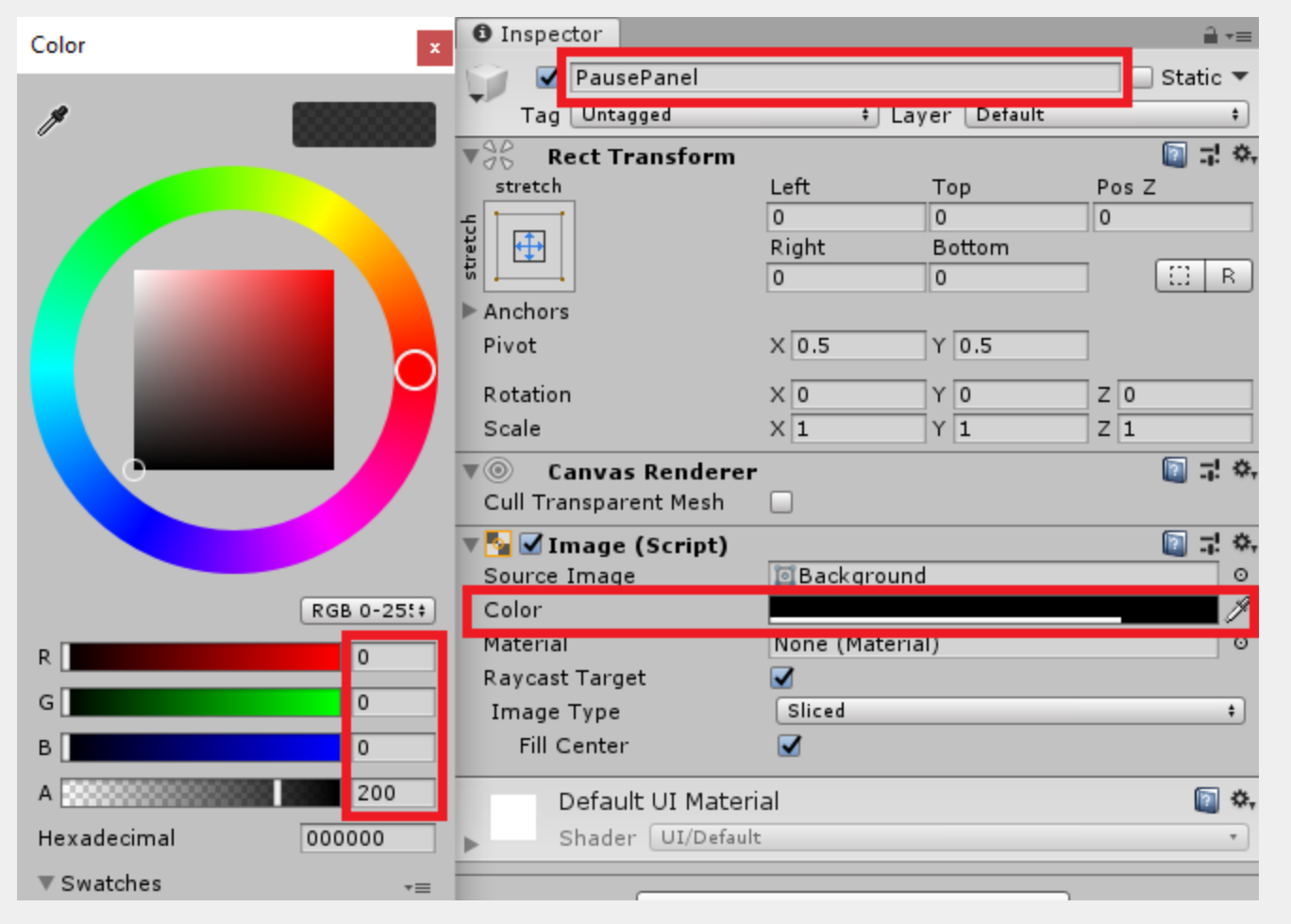


Hard- Pause Menu

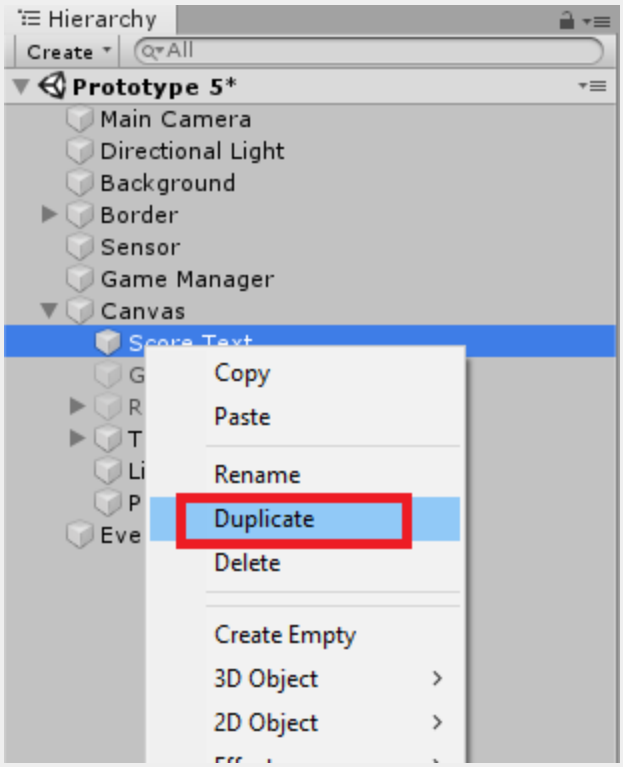
1. In the Hierarchy, right-click on the *Canvas* and select **UI** > **Panel** to create the panel for the pause menu.



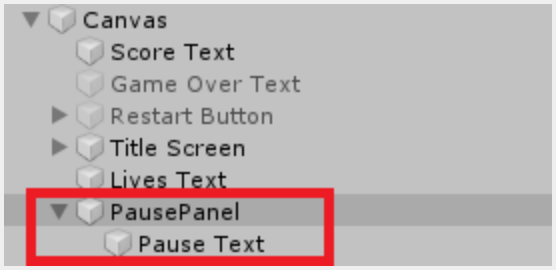
2. Rename the new panel to "*Pause Panel*". Change the **Color** of the **Image** component to a color of your choosing, we changed ours to a transparent black.



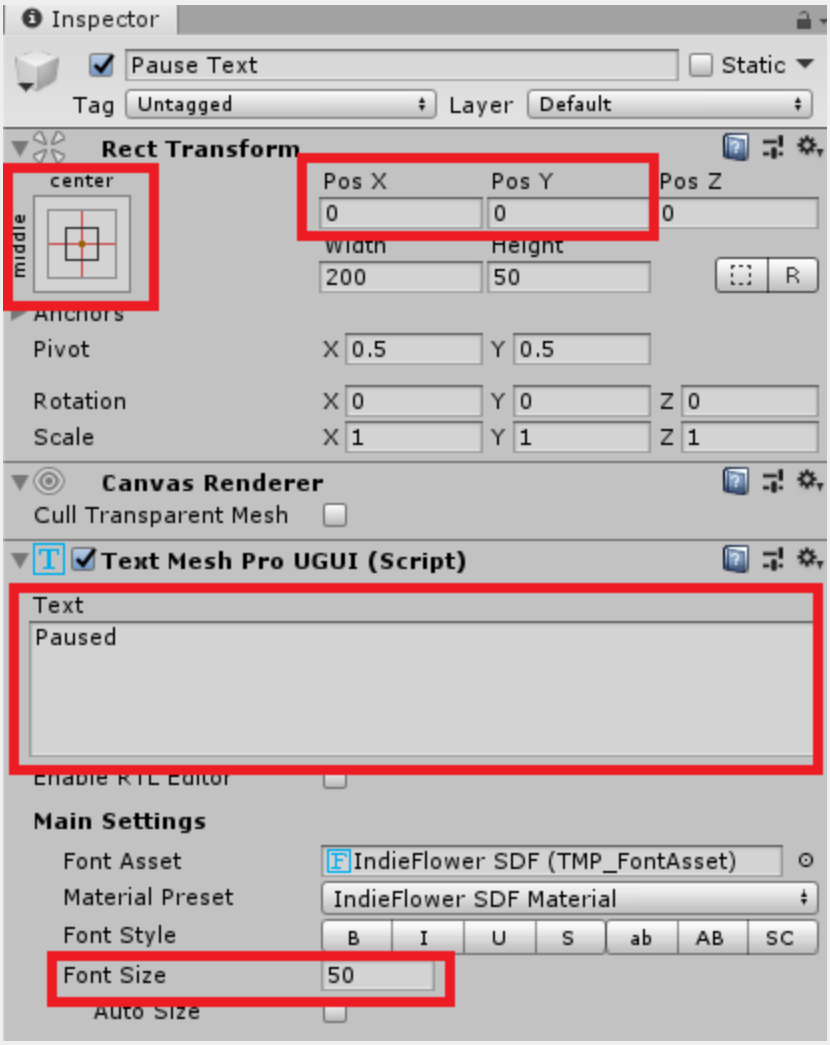
3. In the Hierarchy, right-click on the "*Score Text*" and click **Duplicate**.



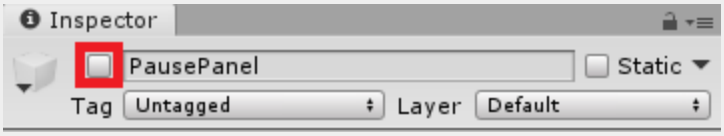
4. Rename the GameObject to *Pause Text*. Drag the Pause Text onto the Pause Panel to make it a child of the Pause Panel.



5. Adjust the position and anchor preset of the Pause Text to position the text in the middle of the screen. Change the Text field of the TextMeshPro - Text (UI) component to Paused and adjust the font size.



6. Go back to the Pause Panel and untick the checkbox next to the name in the Inspector. We don't want the pause panel to be visible unless we are paused.



7. In the Project view, navigate to the Scripts folder and open the **GameManager.cs** script. Just after the variable definitions, define two new variables:

```
public GameObject pauseScreen;
private bool paused;
```

8. After the **StartGame** method, create a new method called **CheckForPause**:

```
void ChangePaused()
{
    if (!paused)
    {
```

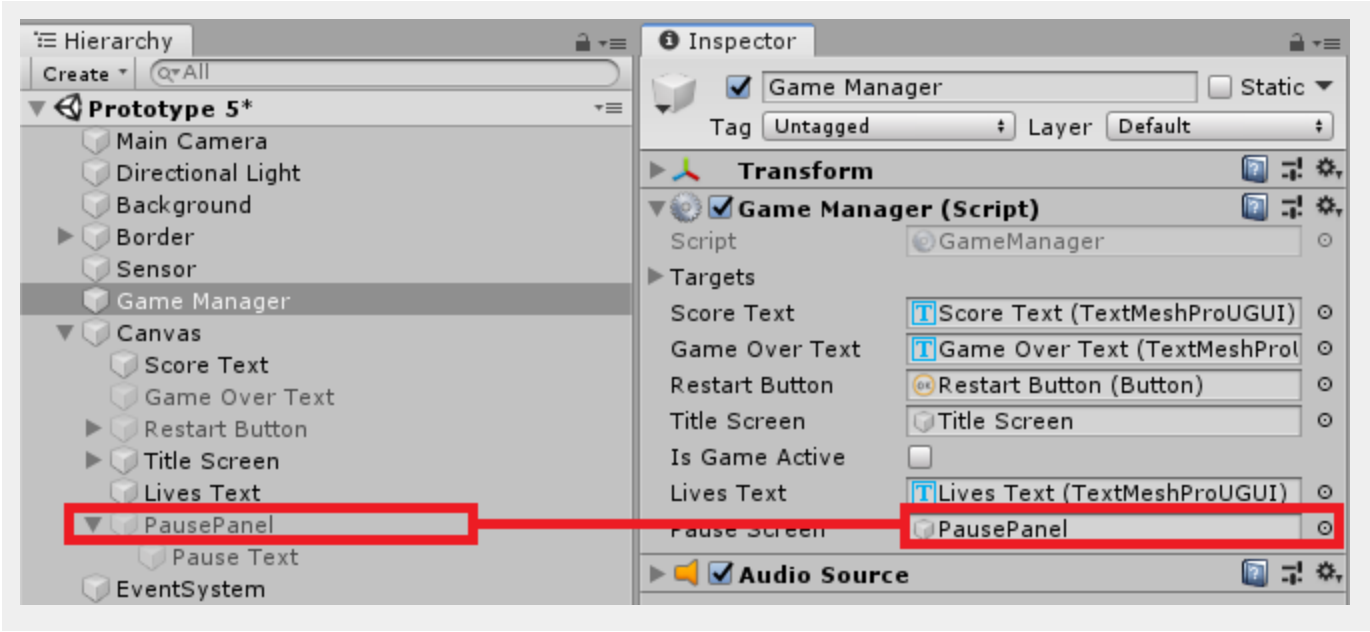
```
        paused = true;
        pauseScreen.SetActive(true);
        Time.timeScale = 0;
    }
    else
    {
        paused = false;
        pauseScreen.SetActive(false);
        Time.timeScale = 1;
    }
}
```

This method will change the *paused* boolean when it is called. When the boolean is changed to true, it **enables** the *pauseScreen* and sets the **Time.timeScale** to **0**. Setting the Time.timeScale to 0 makes it so that physics calculations are paused. When the boolean is changed to false, it **disables** the *pauseScreen* and sets the **Time.timeScale** to **1**.

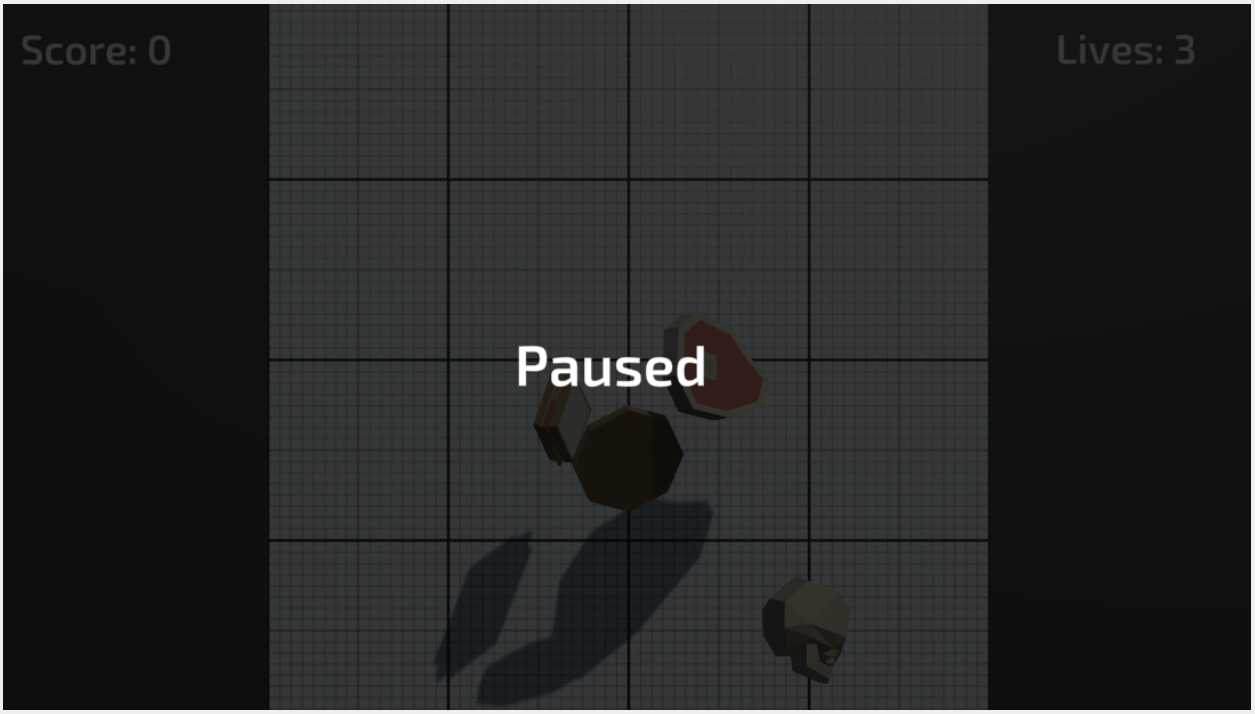
9. In the **Update** method, we need to check if a key has been pressed, if it has we need to make a call to the method we just created, **CheckForPause**. The updated **Update** method should look like this:

```
void Update()
{
    //Check if the user has pressed the P key
    if (Input.GetKeyDown(KeyCode.P))
    {
        ChangePaused();
    }
}
```

10. Save the script and head back to Unity. Select the GameManager GameObject in the Hierarchy. You will notice there is a new field on the **GameManager (Script)** component called **Pause Screen**. We need to drag the Pause Panel that was created earlier into this field.

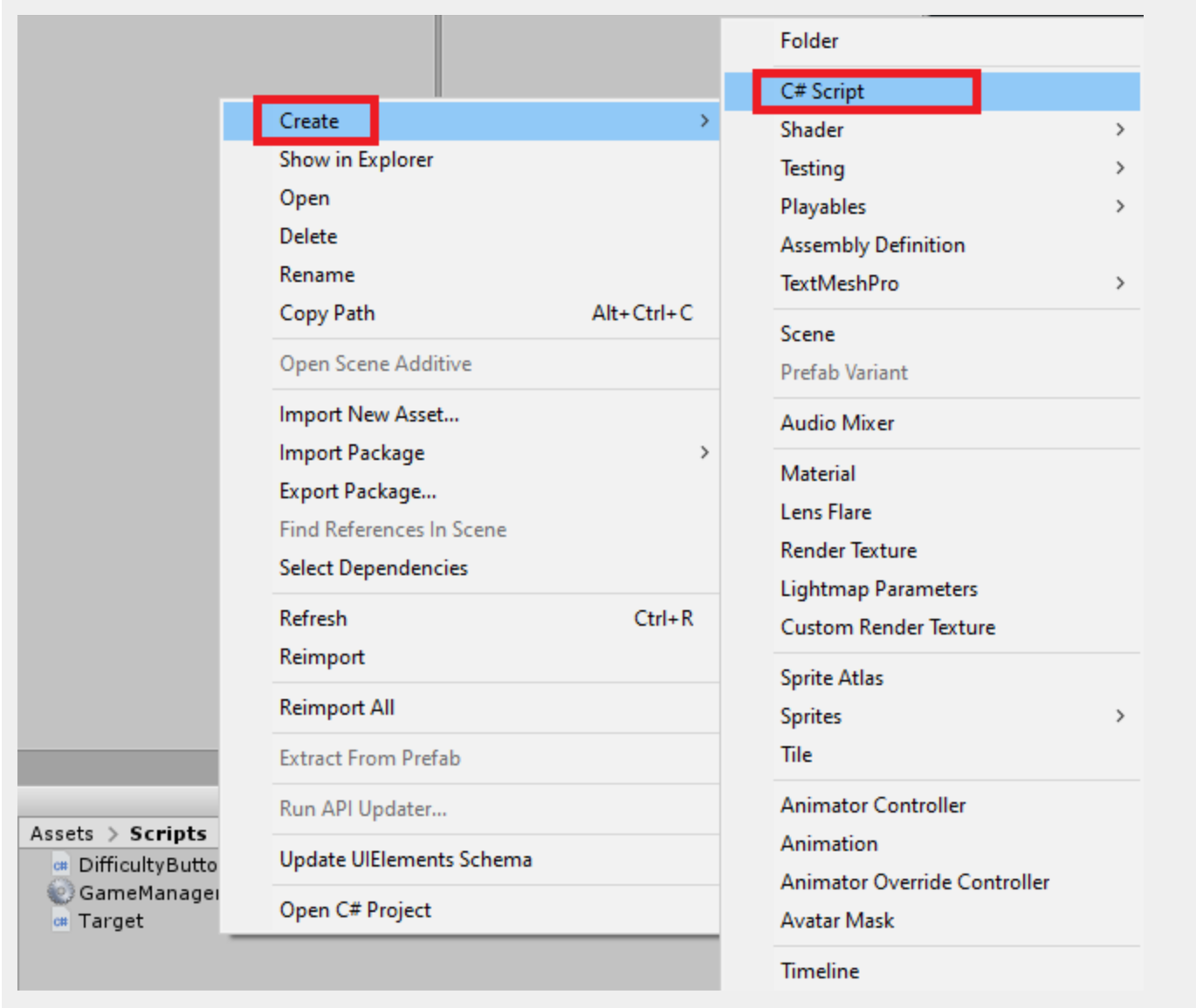


11. Save your scene and run it. Click one of the buttons to start the game, then try pressing ‘P’. The game should now pause.



Expert - Click-and-swipe

1. In the Project window, navigate to the Scripts folder. Right-click and select **Create** > **C# Script**. Name the script *ClickAndSwipe*.



2. Open up the script by double clicking on it. Before the class definition add:

```
[RequireComponent(typeof(TrailRenderer), typeof(BoxCollider))]
```

This code will ensure that a **TrailRenderer** and **BoxCollider** are on the GameObject the script is attached to.

3. Next we will define the variables needed for the script:

```
private GameManager gameManager;
private Camera cam;
private Vector3 mousePos;
```

```
private TrailRenderer trail;
private BoxCollider col;

private bool swiping = false;
```

4. Rename the **Start** method to **Awake**. We will use this to initialize our variables, as they are all private.

```
void Awake()
{
    cam = Camera.main;
    trail = GetComponent<TrailRenderer>();
    col = GetComponent<BoxCollider>();
    trail.enabled = false;
    col.enabled = false;

    gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
}
```

5. Now we're going to set up the GameObject to move with the mouse position. Below the **Update** method write the following:

```
void UpdateMousePosition()
{
    mousePos = cam.ScreenToWorldPoint(new Vector3(Input.mousePosition.x,
Input.mousePosition.y, 10.0f));
    transform.position = mousePos;
}
```

ScreenToWorld will convert the screen position of the mouse to a world position. The reason we use 10.0f on the z axis, is because the camera has the z position of -10.0f.

6. While we are creating new methods, let's create one that will be used to update the TrailRenderer and BoxCollider. In this method we will just set the enabled state to whatever the swiping boolean is.

```
void UpdateComponents()
{
    trail.enabled = swiping;
    col.enabled = swiping;
}
```

7. Let's set up these methods within the **Update** method. We will want to only set swiping to true when the left mouse button is held down. If swiping is true, we will update the mouse position.

```
void Update()
{
    if (gameManager.isGameActive)
    {
```

```
        if (Input.GetMouseButtonDown(0))
        {
            swiping = true;
            UpdateComponents();
        }
        else if (Input.GetMouseButtonUp(0))
        {
            swiping = false;
            UpdateComponents();
        }

        if (swiping)
        {
            UpdateMousePosition();
        }
    }
}
```

8. The last thing we will need to do within this script, is call the **OnCollisionEnter** Unity method. When we collide with something, we will check if it's a Target.

```
void OnCollisionEnter(Collision collision)
{
    if(collision.gameObject.GetComponent<Target>())
    {
        //Destroy the target

    }
}
```

But wait, we currently don't have a public method in the Target.cs script that allows us to destroy the object.

9. Open up the **Target.cs** script. After the **OnTriggerEnter** Method, create a new public method called **DestroyTarget**. We already have all the functionality we need within the **OnMouseDown** method, so copy that into **DestroyTarget**. The method should look like this:

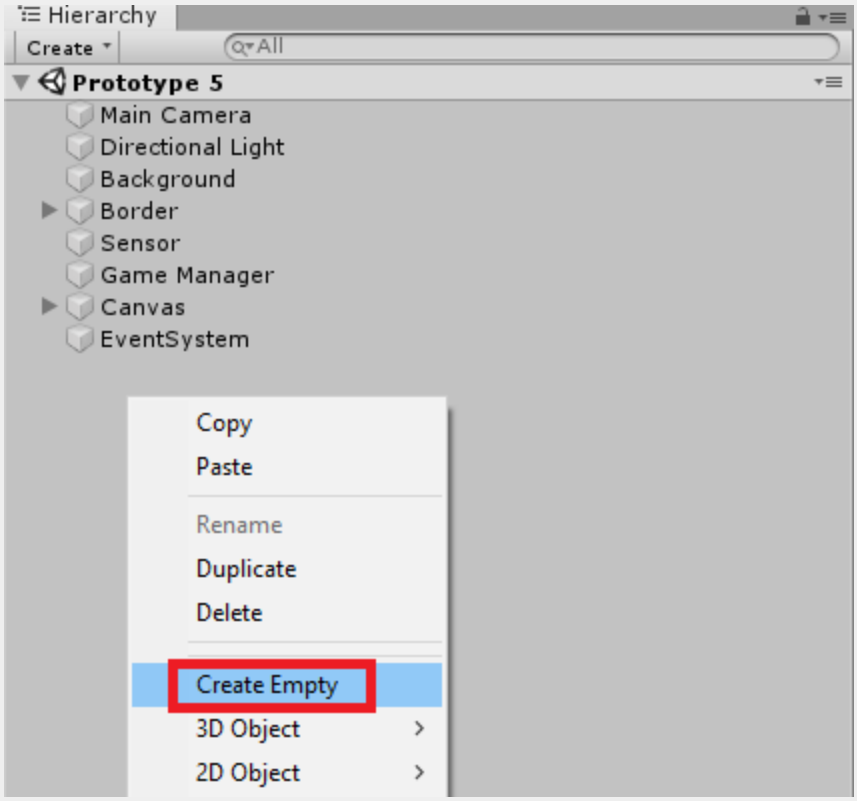
```
public void DestroyTarget()
{
    if (gameManager.isGameActive)
    {
        Destroy(gameObject);
        Instantiate(explosionParticle, transform.position,
explosionParticle.transform.rotation);
        gameManager.UpdateScore(pointValue);
    }
}
```

Comment out the **OnMouseDown** method, as we no longer need it in our game.

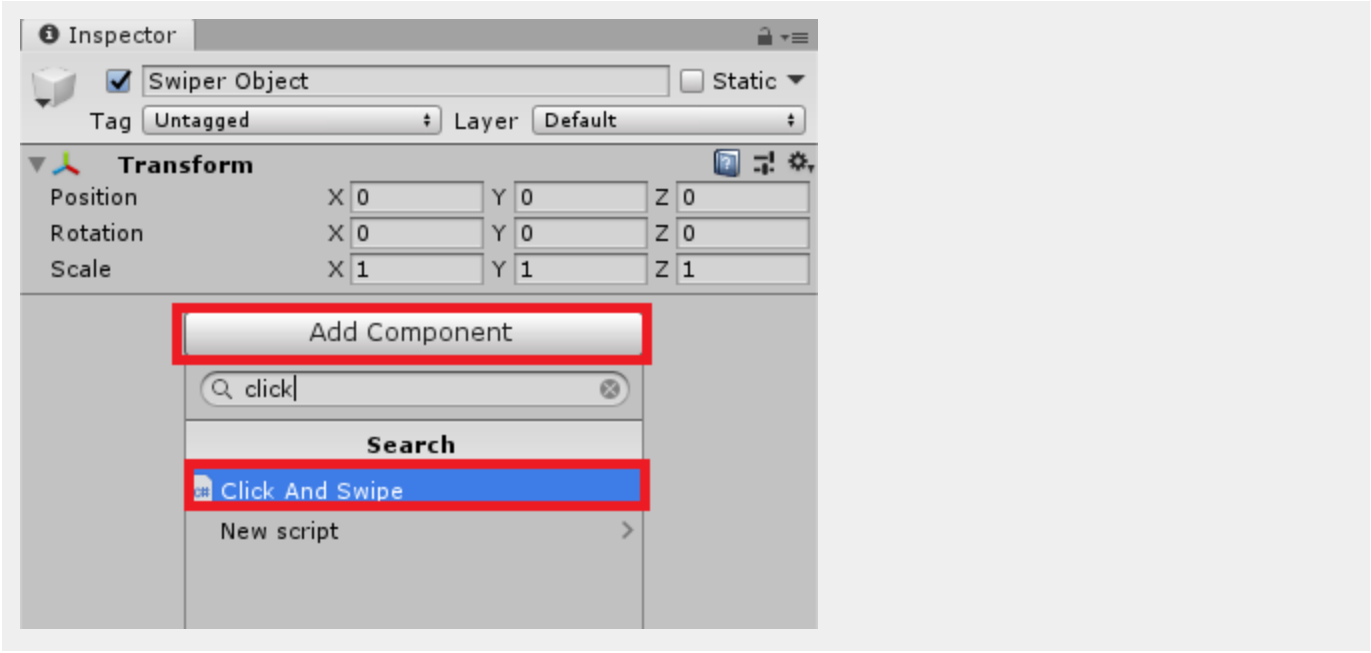
10. Save **Target.cs** and head back to the **ClickAndSwipe.cs** script and update the **OnCollisionEnter** method to look like the following:

```
private void OnCollisionEnter(Collision collision)
{
    if(collision.gameObject.GetComponent<Target>())
    {
        //Destroy the target
        collision.gameObject.GetComponent<Target>().DestroyTarget();
    }
}
```

11. Save all the scripts and head back to Unity. We'll now set up the GameObject that will hold the **ClickAndSwipe.cs** script. In the Hierarchy, right click and select **Create Empty**.

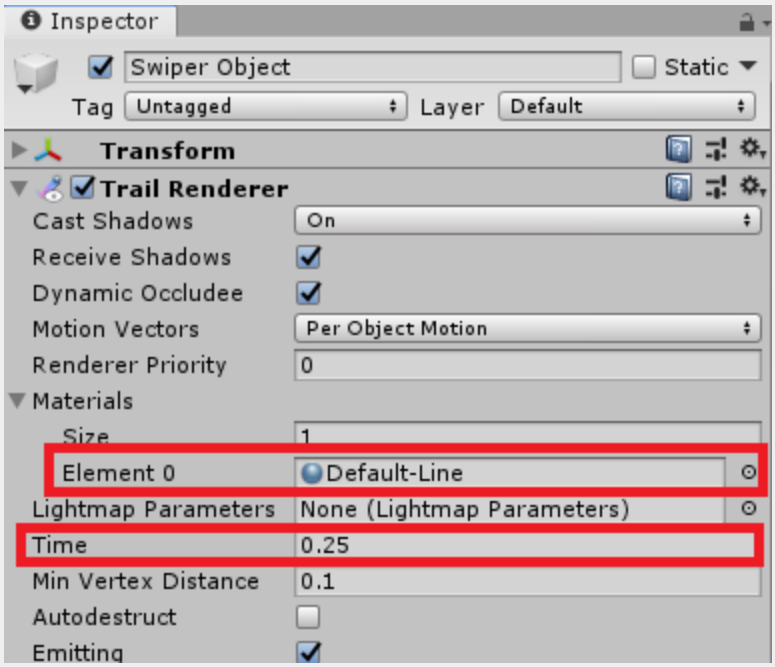


12. Rename the new GameObject to "*Swiper Object*" and then click **Add Component** and search for **ClickAndSwipe**. Add this component to the GameObject.

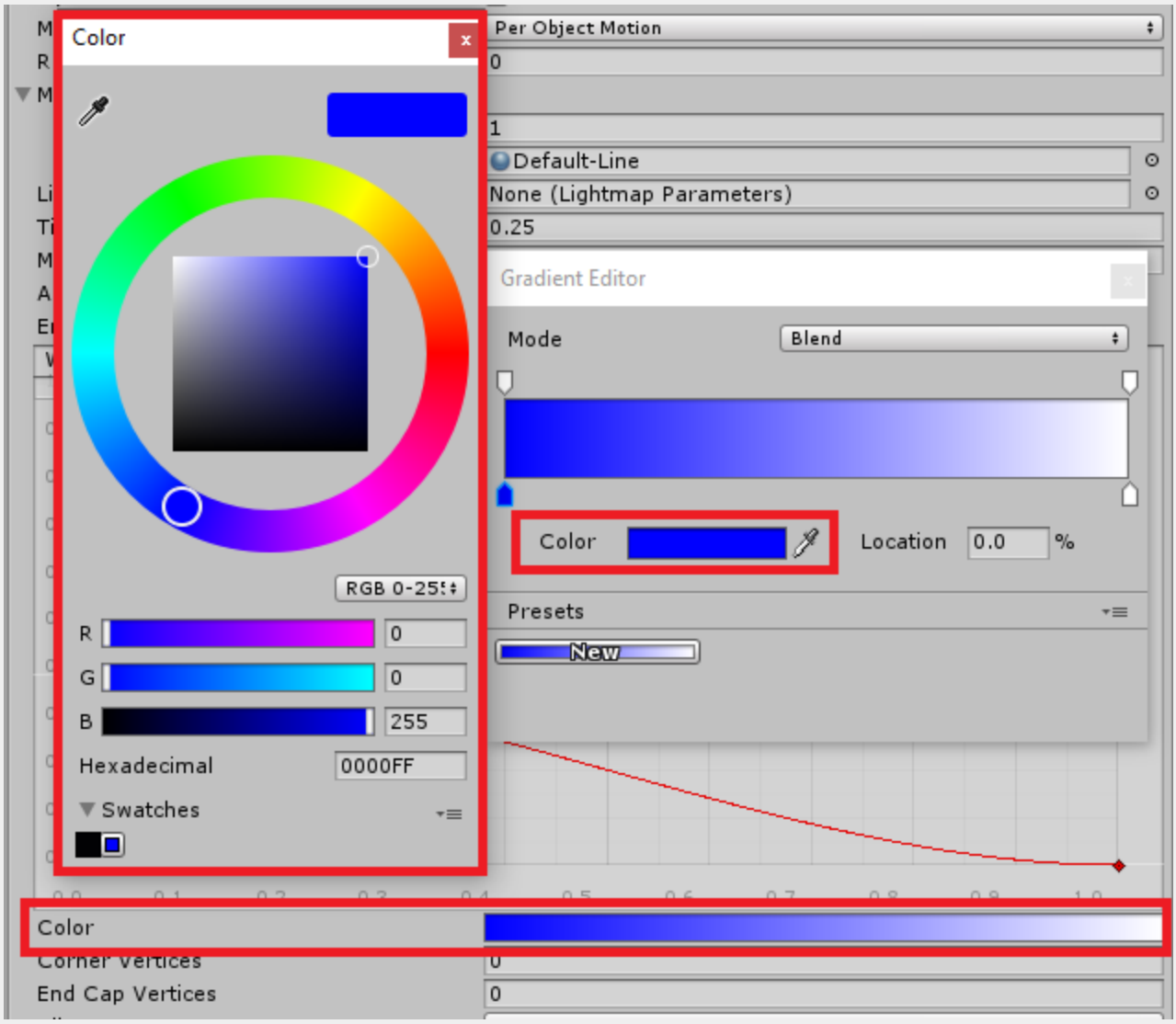


13. The script should have automatically added a **Box Collider** and **Trail Renderer** to the GameObject. Let's adjust the **Trail Renderer**. Change the **Start Width** to **0.4**. Double click at the end of the line to create a second marker. Move this one to be at **0.0**.

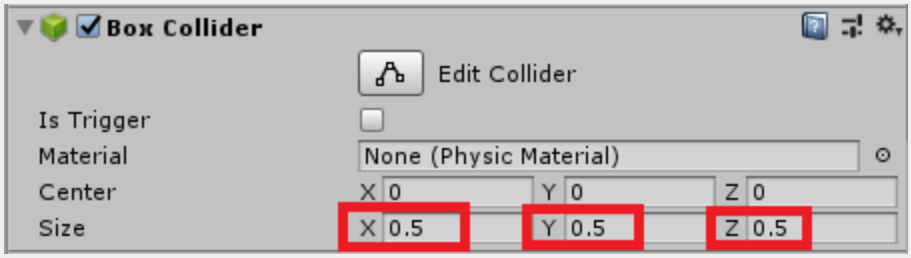
14. Next let's adjust the Time property. This will determine how long the trail appears on the screen. We set our **Time** property to **0.25**. We'll also add a **Material** to the Trail by finding the material property and clicking the circle selector next to it. Assign it the **Default-Line** Material.



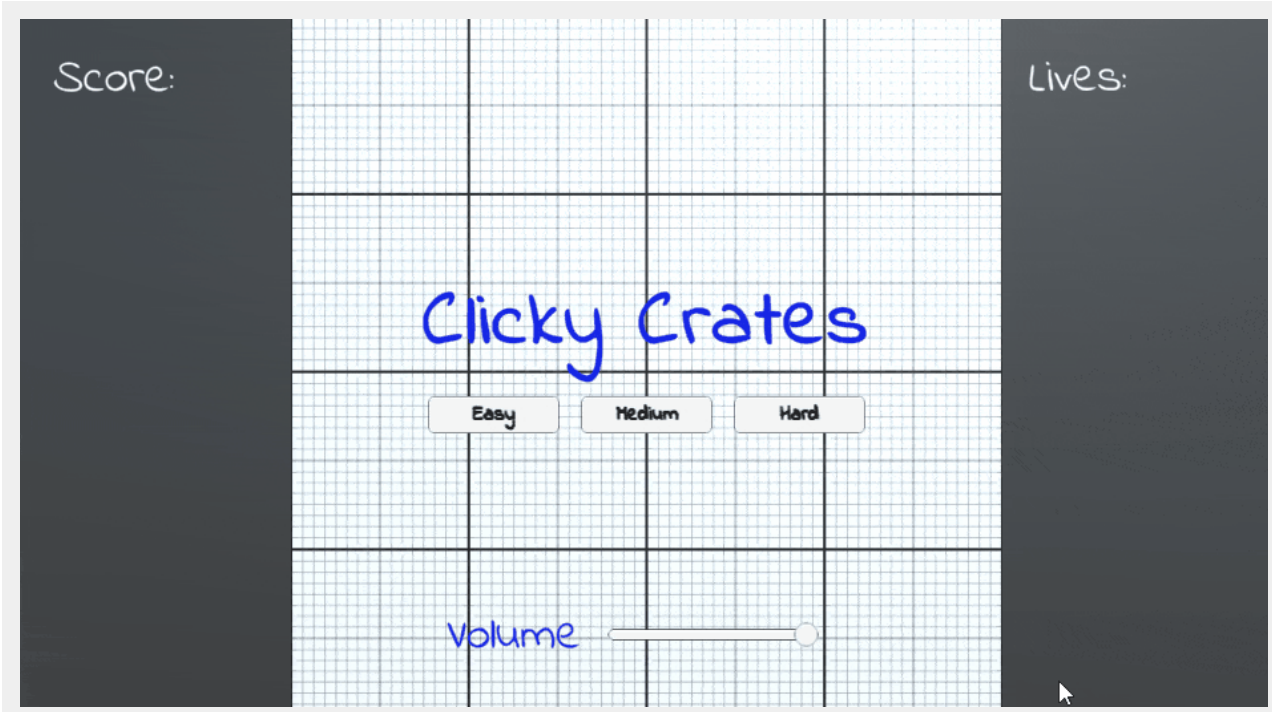
15. We can also adjust the color of the trail, by double clicking the **Color field**. Doing so will show a color gradient window. We adjust our start color to blue, to have a blue to white gradient for the trail.



16. The last thing we need to do is adjust the size of the **BoxCollider**. We adjusted ours to be **0.5** on each axis



17. Save the scene and enter Play mode. Try clicking and dragging, notice how the Trail Renderer follows the mouse.


Score:
Lives:
Clicky Crates
Easy
Medium
Hard
Volume