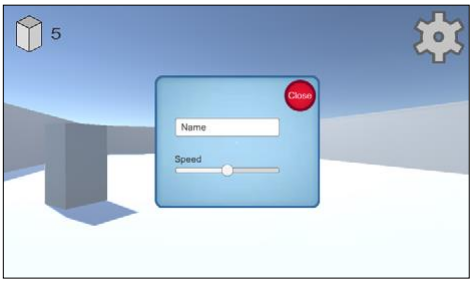
# Unity Game Engine

Introduction to Unity – 2D GUI in 3D Game



# 2D Interface Display

- 2D interface display for a 3D game
- Every game needs information displays in addition to the virtual scene the gameplay takes place in.
- In a game, though, text and buttons are often an additional overlay on top of the game view, a kind of display called a **HUD** (heads-up display.)



# HUD

- A **H**eads-**U**p **D**isplay (**HUD**) superimposes graphics on top of the view of world.
- The concept of a HUD originated with military jets so that pilots could see crucial information without having to look down. Similarly, a GUI superimposed on the game view is referred to as the HUD.



# HUD – Why do game need?

- HUD is one of the most under-appreciated but most important element of modern game development
- HUD provides vital information about player characters' status and the state of the world
- HUD is often key to shaping player behavior.
  - The urgency with which a health indicator signals a terrible wound, or a mini-map emphasizes a quest icon, can dramatically affect how players interact with a game and what gameplay elements get priority.

# HUD Examples - 1

- ***Clash Royale (RTS, multiplayer, mobile game by Supercell)***
  - a HUD is designed to surface player controls instead of tucking them several layers deep in menus, and allow to swipe through multiple tabs instead of treating them as discrete menus


https://www.youtube.com/watch?v=PIPIB9VMEE0



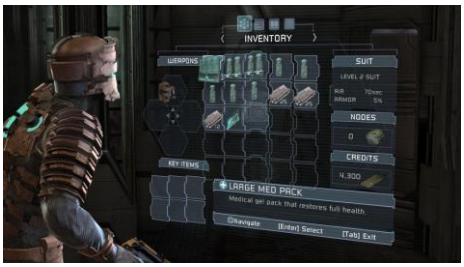
# HUD Examples - 2

- ***Half Life 2 (FPS by Valve Cooperation)***
  - **The HUD look (monochromatic amber palette) is part of the game's unique identity. A HUD was minimal, clean and dynamic. It was also aural in that the game didn't rely purely on visual elements to convey information.**


https://www.youtube.com/watch?v=UKA7JkV51Jw

# HUD Examples - 3

- ***Dead Space (shooter + survival horror by EA)***
  - **In-game HUD : health meter on the player's back, diegetic interface (interface shared by the user and character)**


https://www.youtube.com/watch?v=jqXVw_dWnk8



# GUI Programming in Unity

# Immediate Mode GUI (IMGUI) vs Game-Object Based UI

- IMGUI
  - Recall the target cursor displayed in a previous lecture. That GUI system is entirely based on code, with no work in Unity's editor. This is an example of IMGUI.

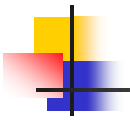


# Immediate Mode GUI (IMGUI) vs Game-Object Based UI

```
using UnityEngine;
using System.Collections;
public class BasicUI : MonoBehaviour {
  void OnGUI() {
    if (GUI.Button(new Rect(10, 10, 40, 20), "Test")) {
      Debug.Log("Test button");
    }
  }
}
```

If you attach this script to any object in the scene, you will put a clickable button on the screen.

- The core of the code is the OnGUI() method.
- Every MonoBehaviour automatically responds to OnGUI() that runs every frame after the 3D scene is rendered.



# Immediate Mode GUI (IMGUI) vs Game-Object Based UI

- The Immediate Mode GUI system is commonly used for:
  - Creating in-game debugging displays and tools.
  - Creating custom inspectors for script components.
  - Creating new editor windows and tools to extend Unity itself.



# Immediate Mode GUI (IMGUI) vs Game-Object Based UI

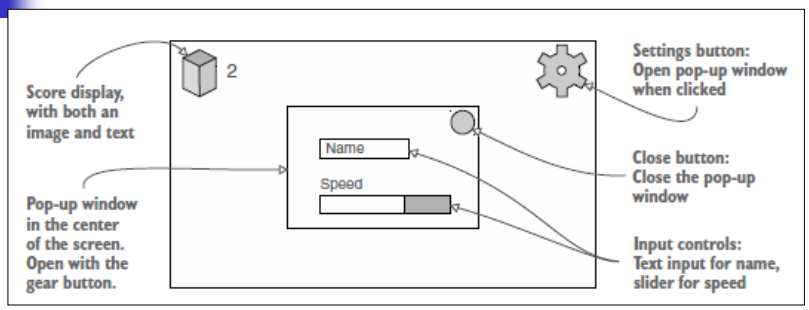
- The IMGUI system is **NOT** generally intended to be used for normal in-game user interfaces that players interact with.
- For that you should use Unity's main **GameObject-based UI** system.
- GameObject based UI offers far better tools to work with the visual design and layout of the UI for editing and positioning UI elements.

# GUI Development Steps

1. Planning the interface & create/collect UI assets
2. Placing UI elements on the display
3. Programming interactions with the UI elements
4. Making the GUI respond to events in the scene
5. Making the scene respond to actions on the GUI



# 1. Planning the Layout




Sample GUI

- A **score display** and a **settings button** in the corners of the screen over the main game view.
- The settings button will bring up a **pop-up window**.
  - That window will have a **text field**, a **slider** and a Close **button**.

# Import Images as Sprites

- This UI requires some images to display for things like buttons.

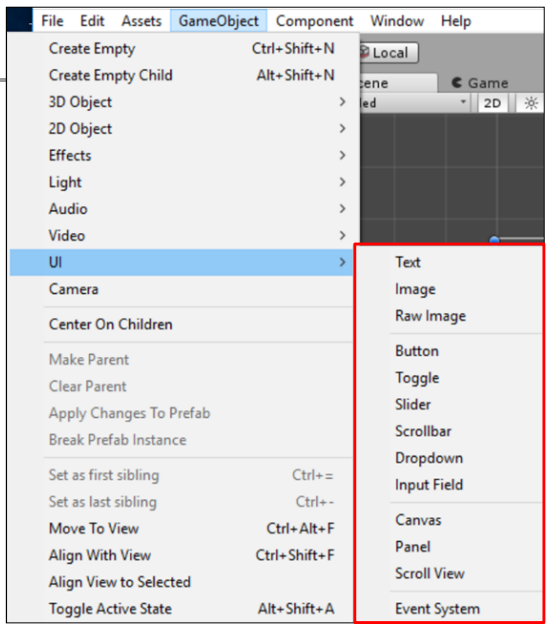
- Import images.
- If needed, **set** them to **Sprite** in Inspector.



# 2. Place UI Elements on the Display

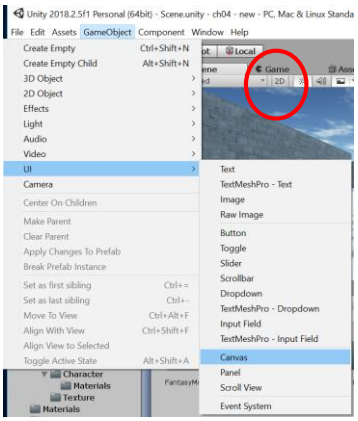
- What are UI elements?

https://docs.unity3d.com/Manual/UISystem.html



# 2. Place UI Elements on the Display

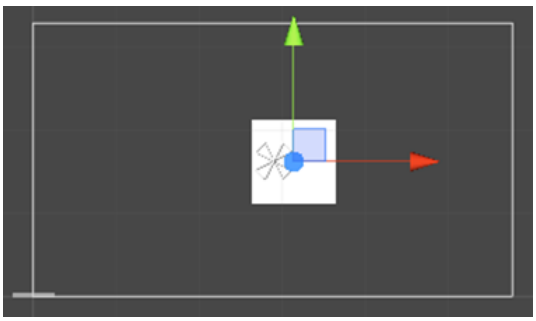
- **Step 1: Create a "Canvas" for the interface**
  - GameObject > UI > Canvas.
  - Name it HUD Canvas.
  - Switch to 2D view mode






# Canvas

- The **Canvas** is the area that all UI elements should be inside.
- The Canvas area is shown as a rectangle in the Scene View.



# Canvas

- **Canvas** uses the **EventSystem** object to help the Messaging System.
  - So an EventSystem will be created along with a Canvas.
- **Draw order of elements:** UI elements in the Canvas are drawn in the same order they appear in the Hierarchy. The first child is drawn first, the second child next, and so on.
  - If two UI elements overlap, the later one will appear on top of the earlier one.
  - To change which element appear on top of other elements, simply reorder the elements in the Hierarchy by dragging them.

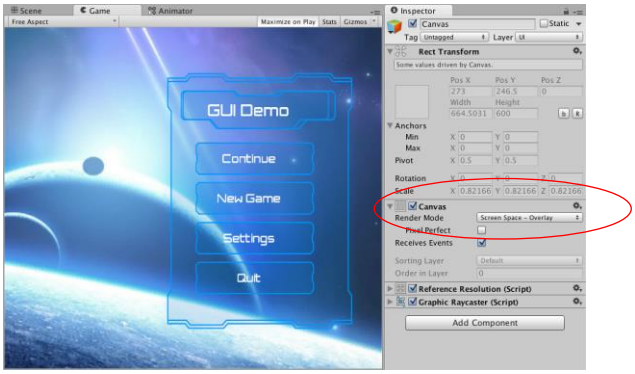


# Canvas Setting

- Choose the canvas in the Hierarchy.
- The canvas has a number of settings that you can adjust.
  - **Render Mode** option; leave this at the default setting (*Screen Space - Overlay)*
  - Pixel Perfect setting – select check box for the rendering to subtly adjust the position images to be perfectly crisp and sharp.

# Canvas - Render Mode

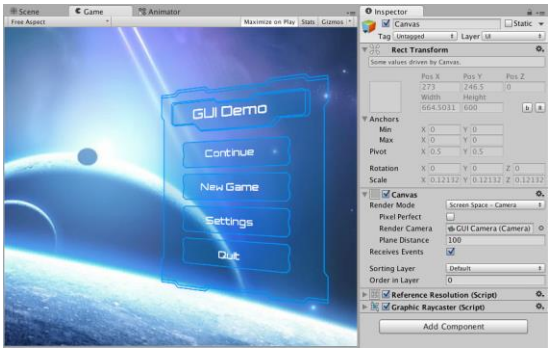
- **Screen Space – Overlay:** places UI elements on the screen rendered on top of the scene. If the screen is resized or changes resolution, the Canvas will automatically change size to match this.





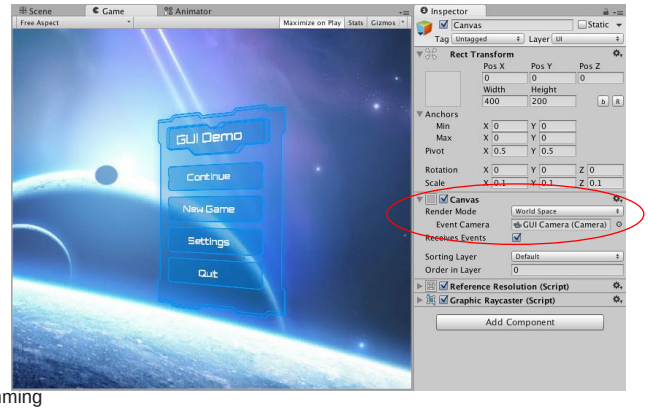

# Canvas - Render Mode

- **Screen Space – Camera :** In this render mode, the Canvas is placed a given distance in front of a specified **Camera**. The UI elements are rendered by this camera, which means that the Camera settings affect the appearance of the UI. If the Camera is set to **Perspective**, the UI elements will be rendered with perspective.



# Canvas - Render Mode

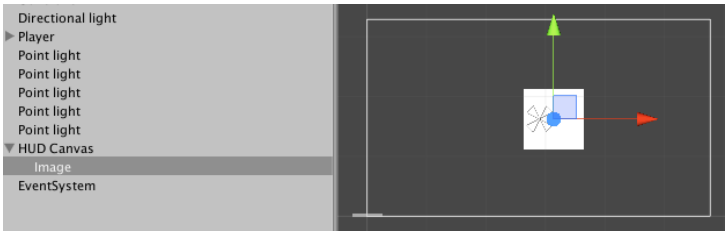
- **World Space :** In this render mode, the Canvas will behave as any other object in the scene. UI elements will render in front of or behind other objects in the scene based on 3D placement. This is useful for UIs that are meant to be a part of the world.





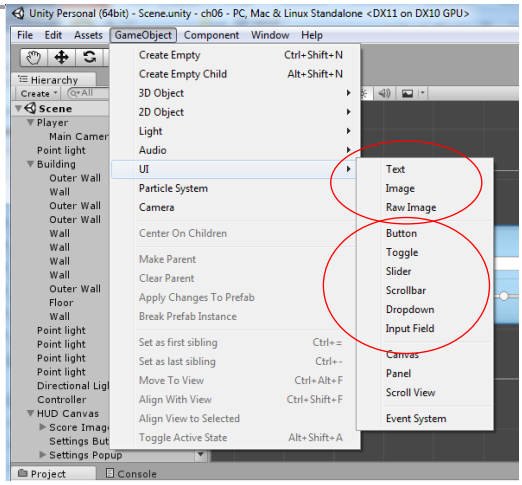

# 2. Place UI Elements on the Display

- **Step 2: Create an UI element (eg image)**
  - GameObject > UI > Image



# UI Elements

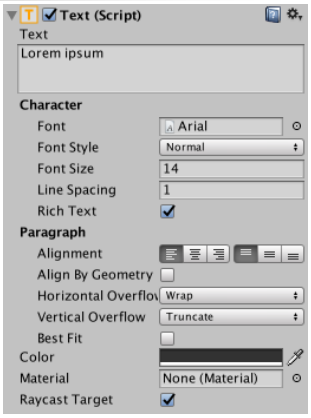
- Visual Components
  - Text (Label)
  - Image
- Interactive Components
  - Button
  - Toggle
  - Slider
  - Input Field
  - …





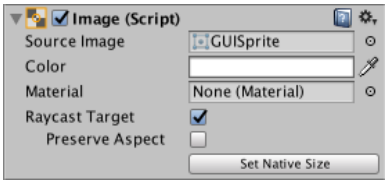

# UI Elements - Text

- The **Text** component (a.k.a a Label) has a Text area for entering the text that will be displayed.
- It is possible to set the font, font style, font size and whether or not the text has rich text capability.



# UI Elements - Image

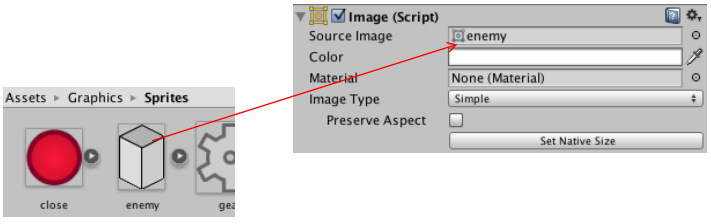
- An Image has a Rect Transform component and an **Image** component.
- A sprite can be applied to the Image component
- Its colour can be set in the Color field.
- A material can also be applied to the Image component.






# 2. Place UI Elements on the Display

- **Step 3: Assign the Sprite to the Image UI element**



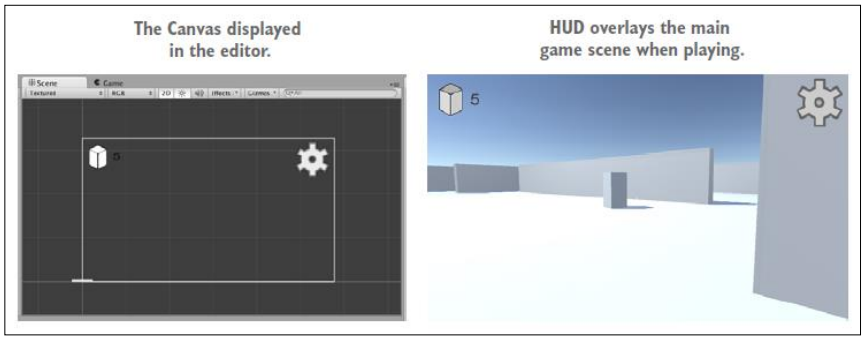
# 2. Place UI Elements on the Display

- **Step 4: Make an UI element be a child of the canvas**
  - UI elements need to be a child of the canvas object in order to display correctly.




# 2. Place UI Elements on the Display

- **Step 5: Roughly position the UI elements (an image, a button, and one text) into their corners.**



# Rect Tool

- **Rect Tool**
  - Every UI element is represented as a rectangle for layout purposes.
  - The Rect Tool can be used to move, resize and rotate UI elements.



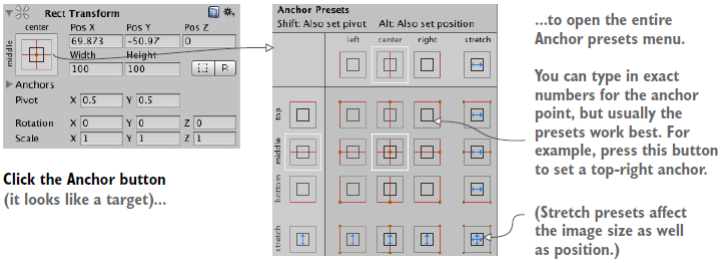
# Rect Transform

- **Rect Transform**
  - The **Rect Transform** is a new transform component that is used for all UI elements instead of the regular **Transform** component.
  - **Rect Transform**s have position, rotation, and scale just like regular Transforms, but it also has a width and height, used to specify the dimensions of the rectangle.


- Resize
- Scale
- Rotation

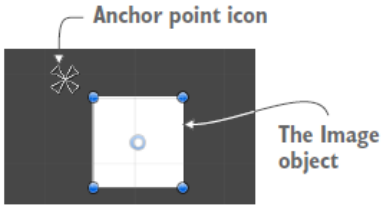
# 2. Place UI Elements on the Display

- **Step 6: Fine tune UI positons.**
  - Let's change anchors of Enemy UI and Gear UI elements.
  - Select the image object. Anchor settings will appear right below the transform component.



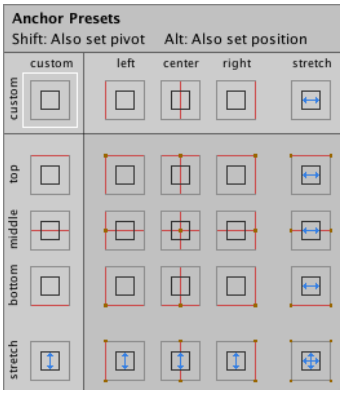



# Anchors




- All UI objects have an anchor, displayed in the editor as a target X.
- The *anchor* of an object is the point where an object attaches to the canvas. It determines what that object's position is measured relative to.
- An anchor is a flexible way of positioning objects on the UI.
- While the object stays in place relative to the anchor point, the anchor moves around relative to the canvas.
- By default, UI elements have their anchor set to Center.

# Anchors

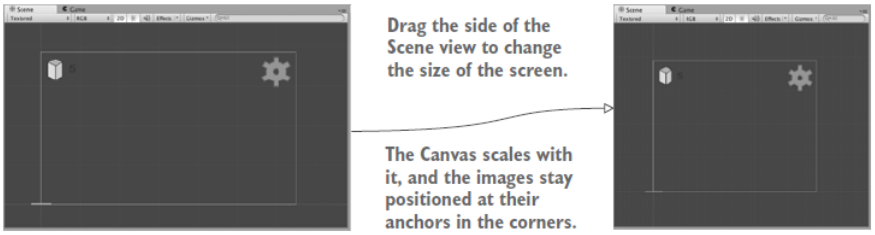



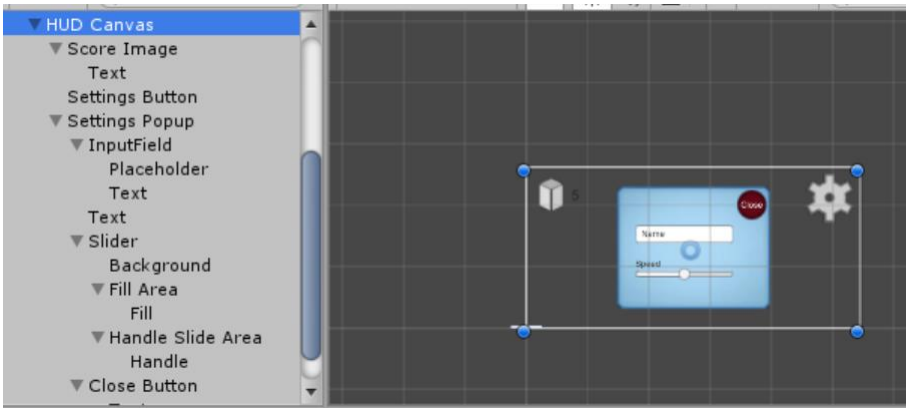

Anchor Presets dropdown. From here you can quickly select from some of the most common anchoring options.



# 2. Place UI Elements on the Display

- Anchors stay in place while the screen changes.








# 3. Programming interactivity in the UI

- RayShooter.cs
  - You need to have a mouse cursor.
  - Previously you lock and hide the mouse cursor, a behavior that works for the controls in an FPS game.
  - That interferes with using the UI.
  - Remove those lines from RayShooter.cs so that you can click on the HUD.

```
//Cursor.lockState = CursorLockMode.Locked;
//Cursor.visible = false;
```

# 3. Programming interactivity in the UI

- Adding a GUI check to the code in RayShooter.cs

```
using UnityEngine.EventSystems;
...
void Update() {
  if (Input.GetMouseButtonDown(0) &&
    !EventSystem.current.IsPointerOverGameObject()) {
      Vector3 point = new Vector3(
      camera.pixelWidth/2, camera.pixelHeight/2, 0);
      ...
```

Mouse click to shoot an enemy should not work when the mouse is clicked over an UI element.



# 3. Programming interactivity in the UI

- Steps to program interactive UIs
  - Create a UI controller object in the scene
  - Write a script to call when the UI is operated.
  - Attach that script to a controller object in the scene.
  - Link UI elements (such as buttons) to the controller object with that script.

# UIController.cs

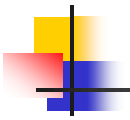
- Create a script called UIController. Attach this to a UI controller object.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class UIController : MonoBehaviour {
[SerializeField] private Text scoreLabel;
  void Update() {
    scoreLabel.text =
      Time.realtimeSinceStartup.ToString();
  }
  public void OnOpenSettings() { // callback function
    Debug.Log("open settings");
  }
}
```



# 3. Programming interactivity in the UI

- Drag the score label (the text object we created before) to the UIController's text slot, `scoreLabel`.
- The code in UIController sets the text displayed on that label.
  - Currently the code displays a timer to test the text display; that will be changed to the score later.



# 3. Programming interactivity in the UI

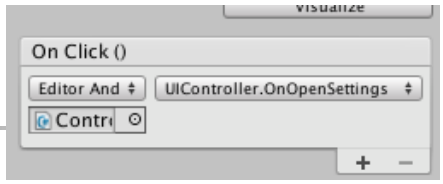
- Most of the interaction components have some things in common. They are selectables, which means they have shared built-in functionality for visualising transitions between states (normal, highlighted, pressed, disabled), and for navigation to other selectables using keyboard or controller.



# Interactive UI Elements - Button

- **Button :** A Button has an **OnClick** UnityEvent to define what it will do when clicked.

# Button

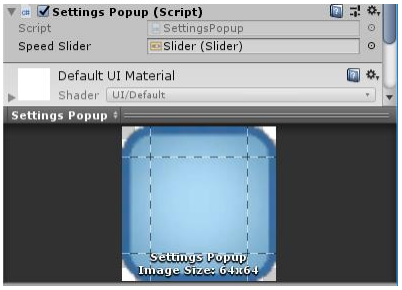



- Select the button.
- Find **OnClick** panel in Inspector.
- Click the + button to add an entry to that panel.
  - Each entry defines a single function that gets called when that button is clicked; the listing has both a slot for an object and a menu for the function to call.
- Drag the UI controller object to the object slot, and then look for UIController in the menu; select **OnOpenSettings()** in that section.



# Creating a pop-up window

- Create a new UI element image by GameObject > UI > Image.
- Drag a sprite to that Source Image slot to set this image.
- Resize it accordingly.
- Create SettingPopup.cs and attach to the UI element.



# SettingPopup.cs

- Create SettingPopup.cs and attach to the UI element (Image object for popup).

```
using UnityEngine;
using System.Collections;
public class SettingsPopup : MonoBehaviour {
  public void Open() {
    gameObject.SetActive(true);
  }
  public void Close() {
    gameObject.SetActive(false);
  }
}
```



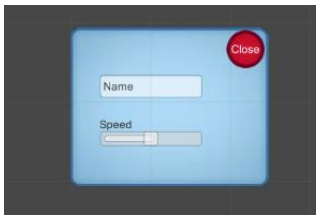
# UIController.cs

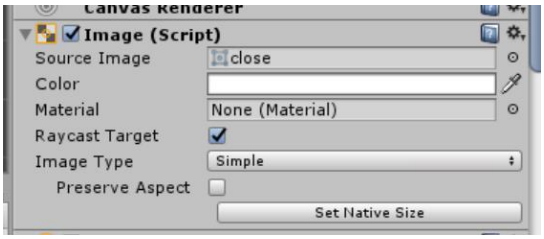
- Modify the script. Then, drag the pop-up to UIController.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class UIController : MonoBehaviour {
..
[SerializeField] private SettingsPopup settingsPopup;
    void Start() {
          settingsPopup.Close();
    }
    ...
    public void OnOpenSettings() {
          settingsPopup.Open();
    }
}
```

# Close Button for the Popup

- GameObject > UI> Button
- Make a child of the popup.
- Position the new button in the top-right corner of the popup.
- Go to UI element's Source Image property, and then click Set Native Size to correctly resize the image.
- Create a text label, child of the button. Select the text and type "Close" in the text field.

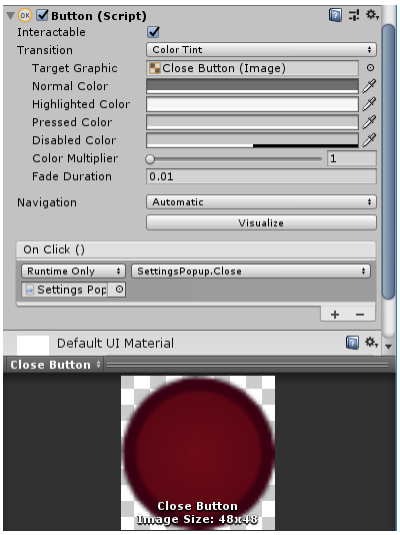





# Close Button for the Popup

- To make the button close the pop-up, it needs an OnClick entry.
- Click the + button on the button's OnClick panel and drag the pop-up window (with SettingPopup.cs) into the object slot, and choose Close() from the function list.



# Resources

- 4 sprite images for UI elements.