Game Programming Using Unity 3D

Lesson 14: Main Menu GUI

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital Inc.

Admin and Co-founder, Unity Philippines Users Group

December 2011

Overview

Again, we'll be continuing where we left off from the previous lesson. If you haven't done Lesson 13, you need to do it before you can continue here.

Now we'll start fleshing out the GUI of our game. We'll start with a *Main Menu*. A Main Menu is the first thing a player can interact with in a game. This is where he can start a game, change some settings, and exit the game, among other things.

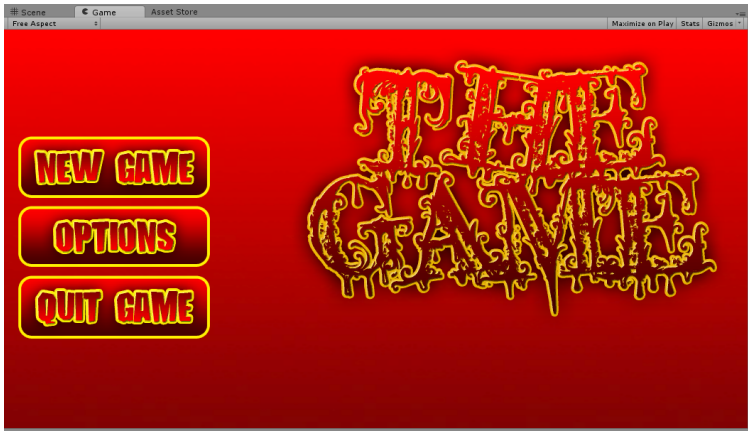


Illustration 1: A preview of what our Main Menu will look like.

Creating The Main Menu

Create a new scene. Immediately save it in the Scenes folder as “MainMenu.unity”.

Create a new C# script. Name it “MainMenuGui”.

Go to your lesson assets folder an import the following files into your project's Gui folder:

1. ButtonBg.png
2. ButtonBgPressed.png
3. ButtonNewGame.png
4. ButtonOptions.png
5. ButtonQuit.png
6. MainMenuBg.png
7. Title.png

Those are the files that we'll be using for the Main Menu.

Go through each of the images and set their Texture type to "GUI". This will ensure the images will have their proper height and width.

Open your MainMenuGui script.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     void OnGUI()
10     {
11         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
12     }
13 }
```

We simply draw the image, stretched to the current width and height of the screen. A good GUI is one that can adapt to various screen resolutions.

Attach your MainMenuGui to the MainCamera in the scene. Drag the MainMenuBg image that we have in the Project View into the corresponding slot in the MainMenuGui script component.

Test the game. You should see a simple red background.

Now add this code:

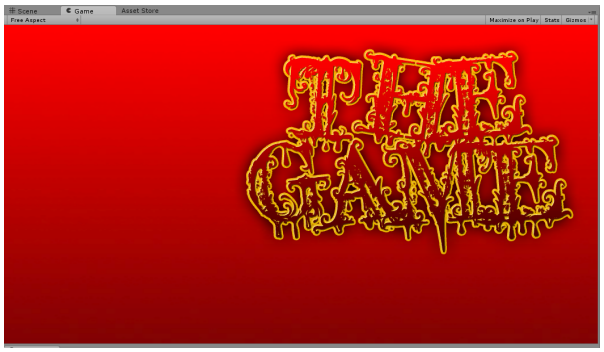
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     [SerializeField]
10     Texture2D _title;
11
12     void OnGUI()
13     {
14         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
15
```

```
16          GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
17                          _title);
18      }
19 }
```

We'll now display the title image. This time, we're displaying it at the right edge of the screen. Notice the first argument to the Rect constructor: Screen.width - _title.width - 20. This means the title gets drawn 20 pixels away from the rightmost edge of the screen.

Now drag the corresponding Title image from the Project View into the slot in the MainMenuGui script component.

Test the game. The title should now appear.



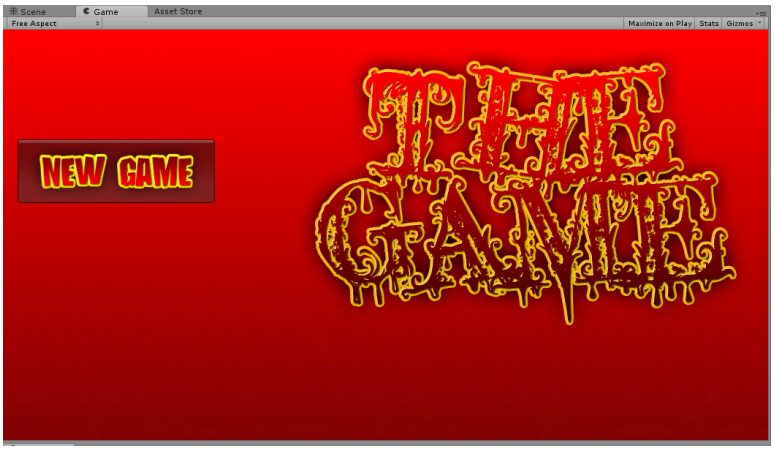
Now we'll add the buttons. Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06      [SerializeField]
07      Texture2D _mainMenuBg;
08
09      [SerializeField]
10      Texture2D _title;
11
12      [SerializeField]
13      Texture2D _newGameButton;
14
15      void OnGUI()
16      {
17          GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
18
19          GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
20                          _title);
21
22          if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
23          {
24          }
25      }
26 }
```

GUI.Button displays a clickable button on the screen. It works pretty much the same way as GUI.DrawTexture, but GUI.Button returns true when it gets clicked. This is why we enclose it inside an if statement. We'll put code inside the if later, when the user presses the “New Game” button.

Drag the ButtonNewGame image from the Project View into the corresponding slot in your MainMenuGui script component.

Test the game now. You should see a button that you can click.



Now we'll add the rest of the buttons:

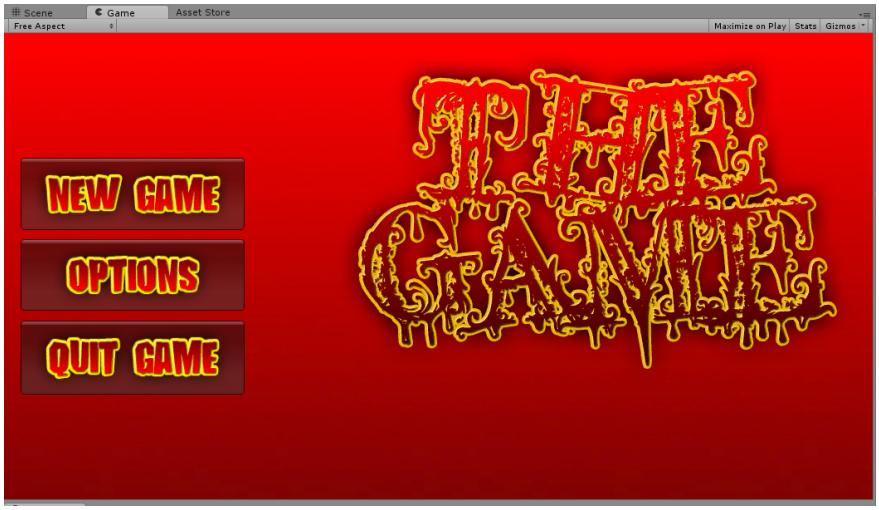
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _mainMenuBg;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _newGameButton;
14
15     [SerializeField]
16     Texture2D _optionsButton;
17
18     [SerializeField]
19     Texture2D _quitButton;
20
21     void OnGUI()
22     {
23         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
24
25         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
26                         _title);
27
28         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
29         {
30         }
31
32         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
33         {
34         }
35
36         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
```

```
37          {
38          }
39      }
40 }
```

The 2nd argument to the Rect constructors are simply ways to properly space our buttons vertically.

Drag the proper images to the MainMenuGui script component.

Test the game to make sure they get displayed correctly.

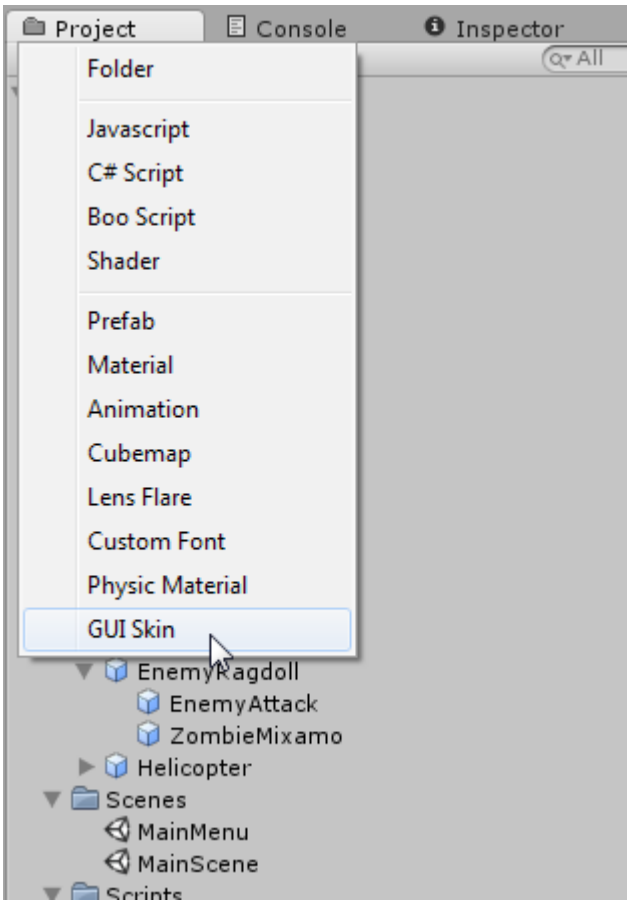


Changing The Buttons' Skin

The buttons' black background isn't fitting with the rest of our GUI's theme. We can actually change that. We do that in Unity by creating a *GUI Skin*. A GUI Skin determines the GUI's look, what fonts it uses, and other things related to the display of GUIs.

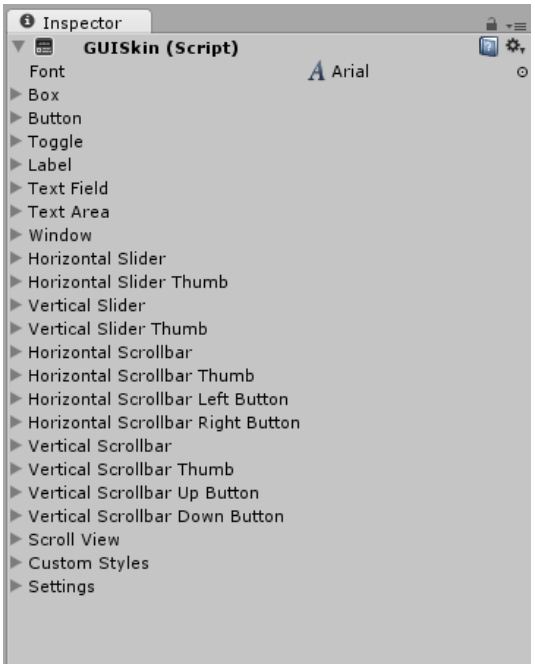
Create a new GUI Skin by clicking the “Create” button found in your Project View.

Choose GUI Skin in the menu that pops up.



Name the new GUI Skin as “GuiSkin”.

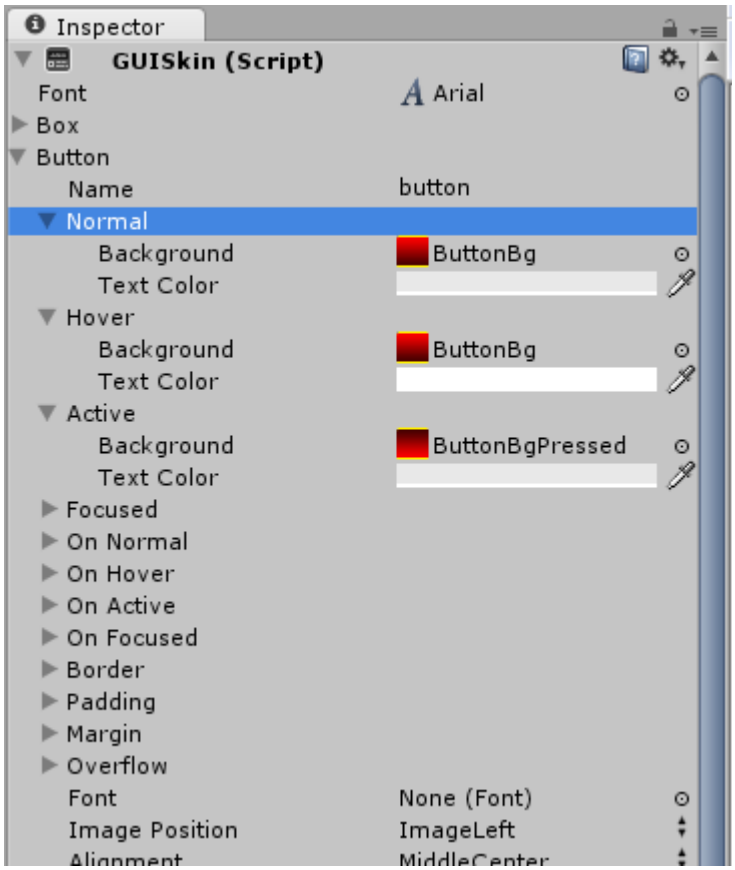
Click on your new GUI Skin. The Inspector will show the various GUI objects. Clicking on the triangle beside each will show the settings for them. Each of these entries are called *GUI Styles*.



We'll change the skin for the buttons. So click on the triangle beside the entry named “Button”.

Under the Button, expand the entries named “Normal”, “Hover”, and “Active”. Those are the settings

for the images used for the button.

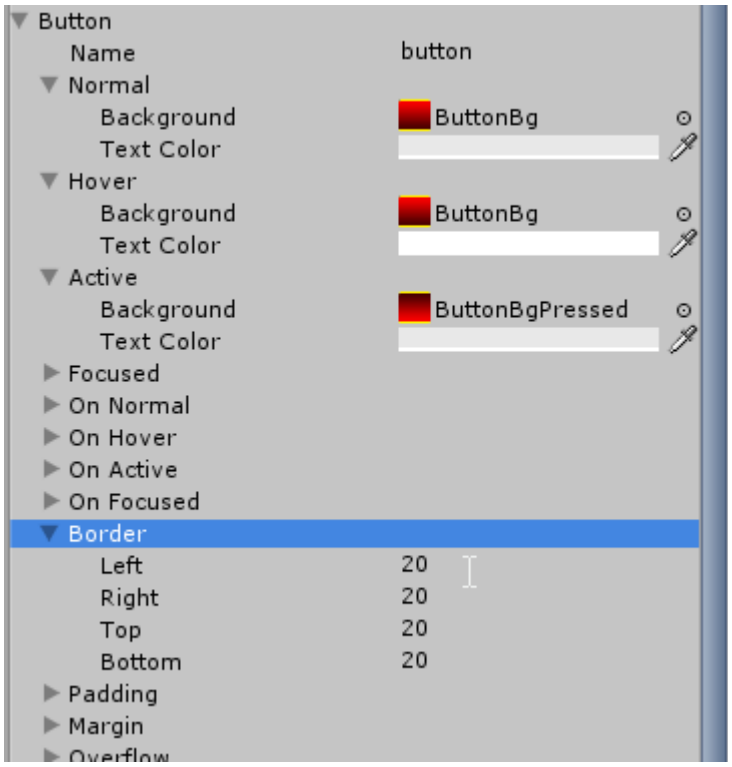


Each of them will have an entry named “Background”. Those are the slots that determine what image is used for the button. You'll see they currently use the black image that you see when you run your game.

Drag the “ButtonBg” image that we have in our Project View into the Background slot of “Normal” and “Hover”. “Active” on the other hand, is the image used when the button is pressed. So use our “ButtonBgPressed” image there.

Also, find the entry named “Border”. This determines which parts of the image gets stretched when the button is specified to be bigger than the background image used.

Specify 20 on all sides.



Now the GUI Skin is ready to be used.

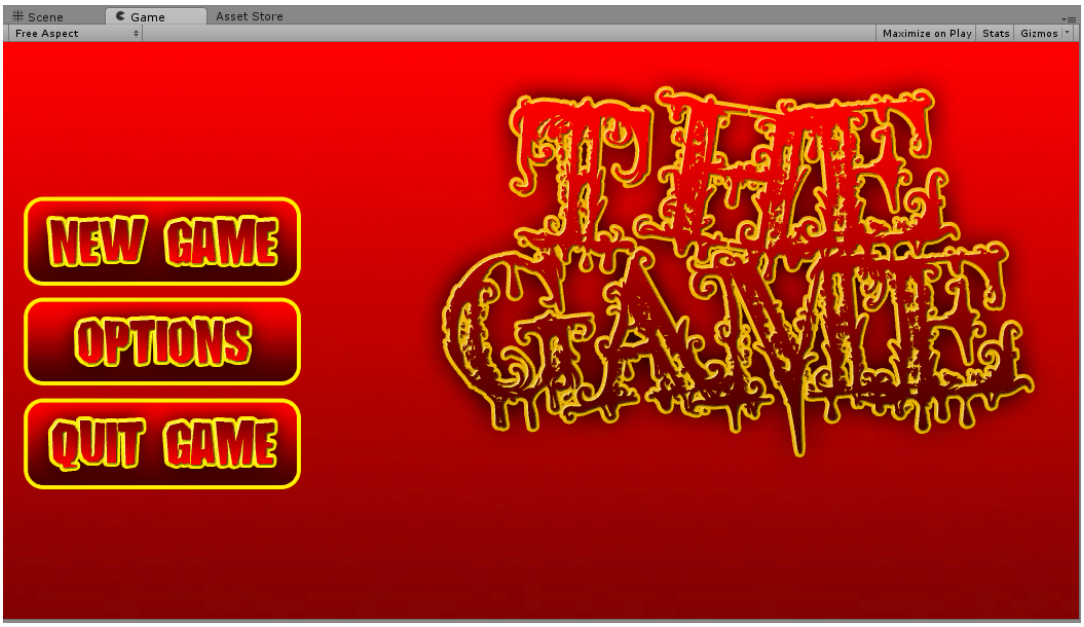
To make use of this GUI Skin, add this code to the MainMenuGui:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
```

```
30          GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                                  _title);
32
33          if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34          {
35          }
36
37          if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
38          {
39          }
40
41          if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
42          {
43          }
44      }
45 }
```

Now your MainMenuGui script component will have a GUI Skin slot. Drag the GUI Skin that we made there.

Test the game. It should now make use of the ButtonBg images that we have.



Adding Code To The New Game Button

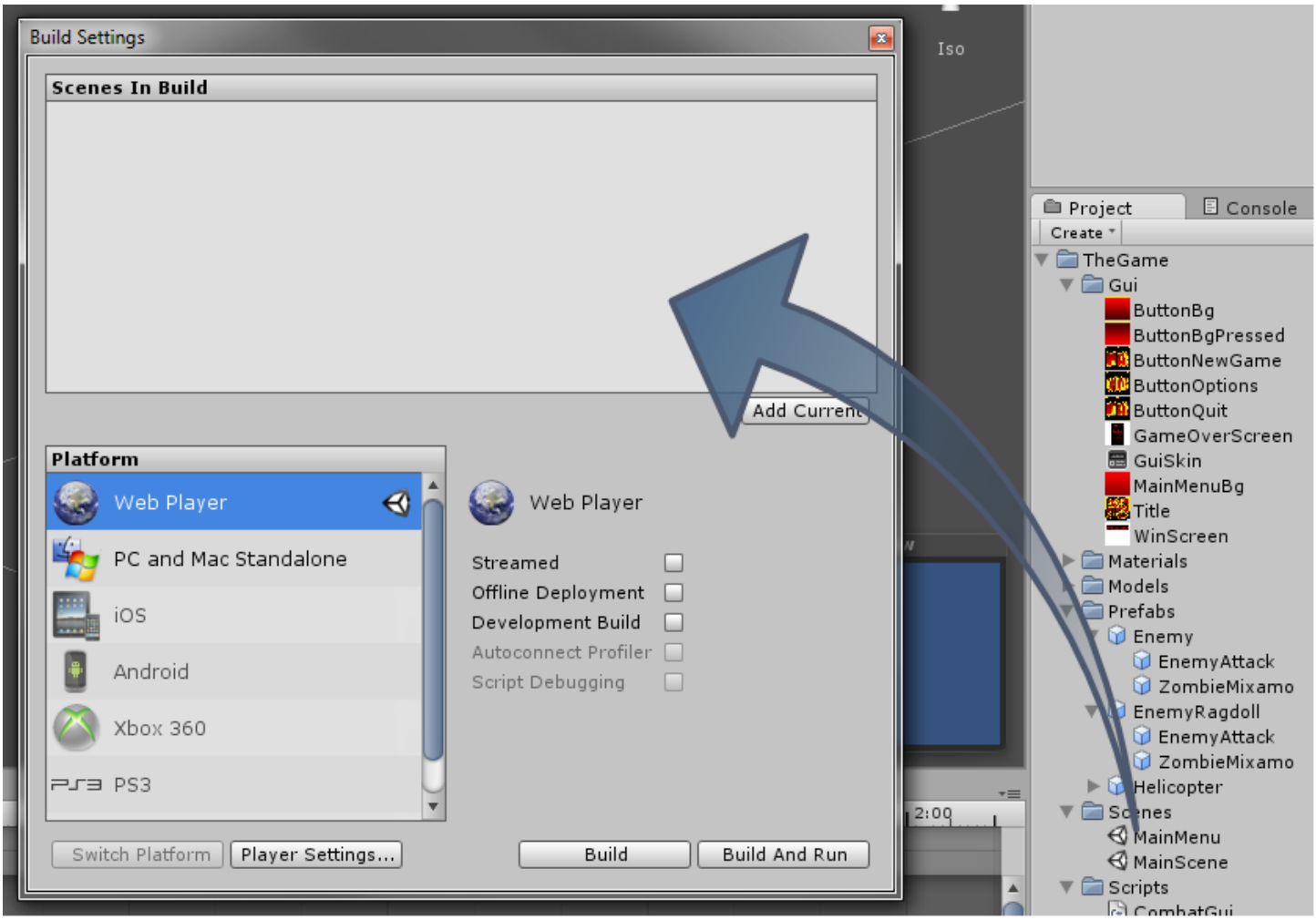
When the user clicks on New Game, the game should start. This is simple. We'll just load the previous scene that we worked on (the one with the player, zombies, and the helicopter).

First, we need to tell Unity which scenes are part of the game. Go to File > Build Settings....

You'll see the Build Settings window. Inside is a list that says “Scenes In Build”. We need to add both of our scenes to this list.

Find the Scenes folder in your Project View. You should have the two scenes MainMenu and MainScene. Drag them into the “Scenes In Build” list. Make sure to add MainMenu first before MainScene since the Main Menu should appear first when the game is run.

If you put MainScene first, just re-order them by dragging the entry for MainMenu to the top.



Now go to your MainMenuGui script and add this code:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  MainMenuGui : MonoBehaviour
05 {
06       [SerializeField]
07       GUISkin _guiSkin;
08
09       [SerializeField]
10       Texture2D _mainMenuBg;
11
12       [SerializeField]
13       Texture2D _title;
14
15       [SerializeField]
16       Texture2D _newGameButton;
17
18       [SerializeField]
19       Texture2D _optionsButton;
20
21       [SerializeField]
22       Texture2D _quitButton;
```

```
23
24      void  OnGUI()
25      {
26            GUI.skin = _guiSkin;
27
28            GUI.DrawTexture(new  Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30            GUI.DrawTexture(new  Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                                    _title);
32
33            if  (GUI.Button(new  Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34            {
35                  Application.LoadLevel("MainScene");
36            }
37
38            if  (GUI.Button(new  Rect(20, 150  + 88  + 10, 270, _optionsButton.height), _optionsButton))
39            {
40            }
41
42            if  (GUI.Button(new  Rect(20, 150  + (2*(88  + 10)), 270, _quitButton.height), _quitButton))
43            {
44            }
45      }
46 }
```

Application.LoadLevel will change the currently loaded scene. All the game objects from the previous scene will be removed.

Test the game now. If you click on “New Game”, you should start with our combat scene.

Adding Code To The Quit Button

Making the program exit is simple. Add this code:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  MainMenuGui : MonoBehaviour
05 {
06      [SerializeField]
07      GUISkin _guiSkin;
08
09      [SerializeField]
10      Texture2D _mainMenuBg;
11
12      [SerializeField]
13      Texture2D _title;
14
15      [SerializeField]
16      Texture2D _newGameButton;
17
18      [SerializeField]
19      Texture2D _optionsButton;
20
21      [SerializeField]
```

```
22      Texture2D _quitButton;
23
24      void  OnGUI()
25      {
26          GUI.skin = _guiSkin;
27
28          GUI.DrawTexture(new  Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30          GUI.DrawTexture(new  Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                                  _title);
32
33          if  (GUI.Button(new  Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34          {
35              Application.LoadLevel("MainScene");
36          }
37
38          if  (GUI.Button(new  Rect(20, 150  + 88  + 10, 270, _optionsButton.height), _optionsButton))
39          {
40          }
41
42          if  (GUI.Button(new  Rect(20, 150  + (2*(88  + 10)), 270, _quitButton.height), _quitButton))
43          {
44              Application.Quit();
45          }
46      }
47  }
```

Just note that this will only work if you deploy your game as a standalone .EXE or Mac executable file.

Creating The Options Screen

The last thing we need to create is the Options Screen. Here we can change settings like the difficulty and volume.

Here's a preview of what our Options Screen looks like:

Create a new C# script and name it “OptionsGui”.

Add this code:

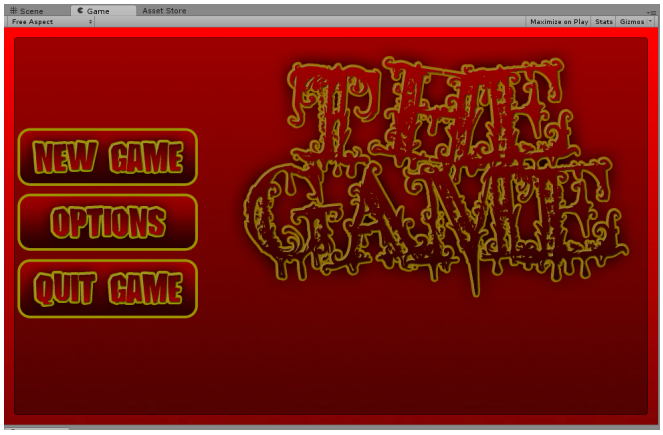
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     void OnGUI()
10     {
11         GUI.skin = _guiSkin;
12     }
13 }
```

Just like what we did on the MainMenuGui, we'll make use of a GUI Skin.

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     void OnGUI()
10     {
11         GUI.skin = _guiSkin;
12
13         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
14     }
15 }
```

Then we'll draw a simple box on screen. Test this script. Attach it on the Main Camera game object and assign the proper GUI Skin.

You'll see a black box covering the screen. That's supposed to be the background of the Options Screen. We need to change the image that that uses as well.



Go to your GuiSkin in the Project View. Select it. In the Inspector, there should be an entry named “Box”. Expand it and find the entry named “Normal”. Expand that as well and you'll see a property named “Background”, which currently uses the black image that we've seen.

Drag the ButtonBg image that we have in the Project View into that Background slot. Find the “Border” property of “Box” as well and set them all to 20, just like what we did for “Button”.

Run the game now. It should show a red box instead.

Now we'll need to import a few more images from our lesson assets folder. Find these files in the Gui folder:

1. ButtonOk.png
2. CheckBoxChecked.png
3. CheckBoxUnchecked.png
4. SliderBg.png
5. SliderThumb.png
6. TextEasy.png
7. TextHard.png
8. TextNormal.png
9. TitleDifficulty.png
10. TitleOptions.png
11. TitleSound.png

Copy them all to your Project's Gui foldier. And **select each and change their Texture Type to “GUI”**.

Now add this code to the OptionsGui script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     void OnGUI()
13     {
```

```
14            GUI.skin = _guiSkin;
15
16            GUI.Box(new  Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
17
18            GUI.DrawTexture(new  Rect(20, 20, _title.width, _title.height), _title);
19        }
20 }
```

Go back to the Unity Editor and assign the TitleOptions image from the Project View into the OptionsGui script component in your Main Camera.

Test the game now. It should look like this:

Now add this code:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  OptionsGui : MonoBehaviour
05 {
06        [SerializeField]
07        GUISkin _guiSkin;
08
09        [SerializeField]
10        Texture2D _title;
11
12        [SerializeField]
13        Texture2D _ok;
14
15        void  OnGUI()
16        {
17            GUI.skin = _guiSkin;
18
19            GUI.Box(new  Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
20
21            GUI.DrawTexture(new  Rect(20, 20, _title.width, _title.height), _title);
22
23            if  (GUI.Button(new  Rect(Screen.width - 30  - _ok.width - 10, Screen.height - 30  - _ok.height
24                - 10, _ok.width, _ok.height), _ok))
25            {
26            }
27        }
28 }
```

We added a button here to close the Options Screen. The coordinates for the Rect are done so that it snaps to the lower right corner of the Options Screen, no matter what screen resolution is used.

Assign ButtonOk for the Texture2D here.

Your GUI should look like this now:

Now we'll add code to actually close the Options Screen:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = true;
16
17     void OnGUI()
18     {
19         if (!_isOpen)
20         {
21             return;
22         }
23
24         GUI.skin = _guiSkin;
25
26         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
27
28         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
29
30         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
31             - 10, _ok.width, _ok.height), _ok))
32         {
33             Close();
34         }
```

```
35      }
36
37      void Close()
38      {
39          _isOpen = false;
40      }
41 }
```

To determine whether the Options Screen is open or not, we make use of a boolean, _isOpen (line 15). When the OK button is pressed, that boolean is set to false (line 39), so the code will stop displaying the Options Screen (line 19).

Test it now. If you click on the OK button, the Options Screen should close.

Making The Main Menu Open The Options Screen

Right now the Options Screen is automatically shown at the start of the game. Instead it should show up only when the user pressed the Options button from the Main Menu. Se we'll fix that.

Open the OptionsGui script and change the code to this:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06      [SerializeField]
07      GUISkin _guiSkin;
08
09      [SerializeField]
10      Texture2D _title;
11
12      [SerializeField]
13      Texture2D _ok;
14
15      static bool _isOpen = false;
16
17      void OnGUI()
18      {
19          if (!_isOpen)
20          {
21              return;
22          }
23
24          GUI.skin = _guiSkin;
25
26          GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
27
28          GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
29
30          if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
31              - 10, _ok.width, _ok.height), _ok))
32          {
33              Close();
34          }
35      }
36
```

```
37      void Close()
38      {
39          _isOpen = false;
40      }
41 }
```

We simply set the initial value for _isOpen to false, so it won't show up at the start.

Now we'll add code to set it to true when the Options button is pressed. Add this code to OptionsGui:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06      [SerializeField]
07      GUISkin _guiSkin;
08
09      [SerializeField]
10      Texture2D _title;
11
12      [SerializeField]
13      Texture2D _ok;
14
15      static bool _isOpen = false;
16
17      static public void Open()
18      {
19          _isOpen = true;
20      }
21
22      void OnGUI()
23      {
24          if (!_isOpen)
25          {
26              return;
27          }
28
29          GUI.skin = _guiSkin;
30
31          GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
32
33          GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
34
35          if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
36              - 10, _ok.width, _ok.height), _ok))
37          {
38              Close();
39          }
40      }
41
42      void Close()
43      {
44          _isOpen = false;
45      }
46 }
```

We'll use a function to signal the Options Screen to open. We'll just call that function in the Main

Menu. So open your MainMenuGui script and add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), _mainMenuBg);
29
30         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.width, _title.height),
31                         _title);
32
33         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height), _newGameButton))
34         {
35             Application.LoadLevel("MainScene");
36         }
37
38         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height), _optionsButton))
39         {
40             OptionsGui.Open();
41         }
42
43         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.height), _quitButton))
44         {
45             Application.Quit();
46         }
47     }
48 }
```

Test the game. You should be able to open the Options Screen when you click on the Options button, and then close it with the OK button.

Unintentionally Clicking On Buttons Underneath The Options Screen

If you test the game, you'll find that you can press the New Game button even while its being

obscured by the Options Screen. Unity doesn't prevent a button from being pressed even if its obstructed, so we have to do that check ourselves.

The idea is, in the Main Menu, we allow the if statement in the GUI.Buttons to continue only if the Options Screen is not open. We just check if the _isOpen is false.

First we need the _isOpen variable to be accessible by public. Open OptionsGui and add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     static bool _isOpen = false;
16     static public bool IsOpen { get{ return _isOpen; } }
17
18     static public void Open()
19     {
20         _isOpen = true;
21     }
22
23     void OnGUI()
24     {
25         if (!_isOpen)
26         {
27             return;
28         }
29
30         GUI.skin = _guiSkin;
31
32         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
33
34         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
35
36         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.height - 30 - _ok.height
37             - 10, _ok.width, _ok.height), _ok))
38         {
39             Close();
40         }
41     }
42
43     void Close()
44     {
45         _isOpen = false;
46     }
47 }
```

Then add this code to MainMenuGui:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MainMenuGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _mainMenuBg;
11
12     [SerializeField]
13     Texture2D _title;
14
15     [SerializeField]
16     Texture2D _newGameButton;
17
18     [SerializeField]
19     Texture2D _optionsButton;
20
21     [SerializeField]
22     Texture2D _quitButton;
23
24     void OnGUI()
25     {
26         GUI.skin = _guiSkin;
27
28         GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),
29                         _mainMenuBg);
30
31         GUI.DrawTexture(new Rect(Screen.width - _title.width - 20, 20, _title.
32                         width, _title.height), _title);
33
34         if (GUI.Button(new Rect(20, 150, 270, _newGameButton.height),
35             _newGameButton) && !OptionsGui.IsOpen)
36         {
37             Application.LoadLevel("MainScene");
38         }
39
40         if (GUI.Button(new Rect(20, 150 + 88 + 10, 270, _optionsButton.height),
41             _optionsButton) && !OptionsGui.IsOpen)
42         {
43             OptionsGui.Open();
44         }
45
46         if (GUI.Button(new Rect(20, 150 + (2*(88 + 10)), 270, _quitButton.
47             height), _quitButton) && !OptionsGui.IsOpen)
48         {
49             Application.Quit();
50         }
51     }
52 }
```

Adding The Difficulty Options

Now we'll add checkboxes to control if the game is easy or hard.

Add this code to the OptionsGui:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     static int _selectedDifficulty = 0;
22
23     static bool _isOpen = false;
24     static public bool IsOpen { get{ return _isOpen; } }
25
26     static public void Open()
27     {
28         _isOpen = true;
29     }
30
31     void OnGUI()
32     {
33         if (!_isOpen)
34         {
35             return;
36         }
37
38         GUI.skin = _guiSkin;
39
40         GUI.Box(new Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
41
42         GUI.DrawTexture(new Rect(20, 20, _title.width, _title.height), _title);
43
44         GUI.DrawTexture(new Rect(50, 150, _titleDifficulty.width,
45                                  _titleDifficulty.height), _titleDifficulty);
46
47         _selectedDifficulty = GUI.SelectionGrid(new Rect(100, 250, 280, 400),
48                                                 _selectedDifficulty, _difficulties, 1);
49
50         if (GUI.Button(new Rect(Screen.width - 30 - _ok.width - 10, Screen.
51             height - 30 - _ok.height - 10, _ok.width, _ok.height), _ok))
52         {
53             Close();
54         }
55     }
```

```
56
57      void  Close()
58      {
59          _isOpen = false;
60      }
61 }
```

In line 44, we just draw the image for the word that says “Difficulty:”.

In line 47, we make use of new GUI called Selection Grids. They work as a group of checkboxes where you can check only one at a time. This is what we use to make the player select if he wants easy, normal, or hard difficulty.

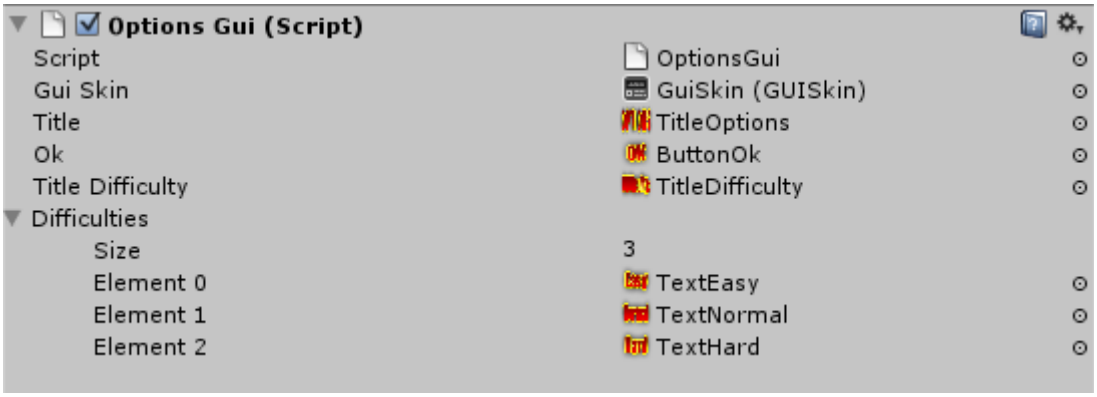
GUI.SelectionGrid is a function that returns the new value that was selected. It will be an integer specifying which checkbox was selected, starting from 0. So 0 means the 1st checkbox was selected, 1 means the 2nd checkbox was selected, and so on.

The 2nd argument to GUI.SelectionGrid is also the same variable used to hold the which was selected. It needs this so it knows which checkbox to display as checked.

The 3rd argument to GUI.SelectionGrid is the array of Texture2D's used to display the text for each checkbox. You can also use an array of strings if you wanted.

The last argument to GUI.SelectionGrid is the number of columns. Actually, Selection Grids are displayed in a tabular format, having rows and columns. Specifying 1 as the number of columns make it a vertical list.

Add the proper images to your OptionsGui script component. For the difficulties, they exist as an array of Texture2D's. Click on the triangle to expand the Difficulties property. Type 3 in the size since we have three difficulties (easy, normal, and hard). Then drag the proper images into the empty slots.



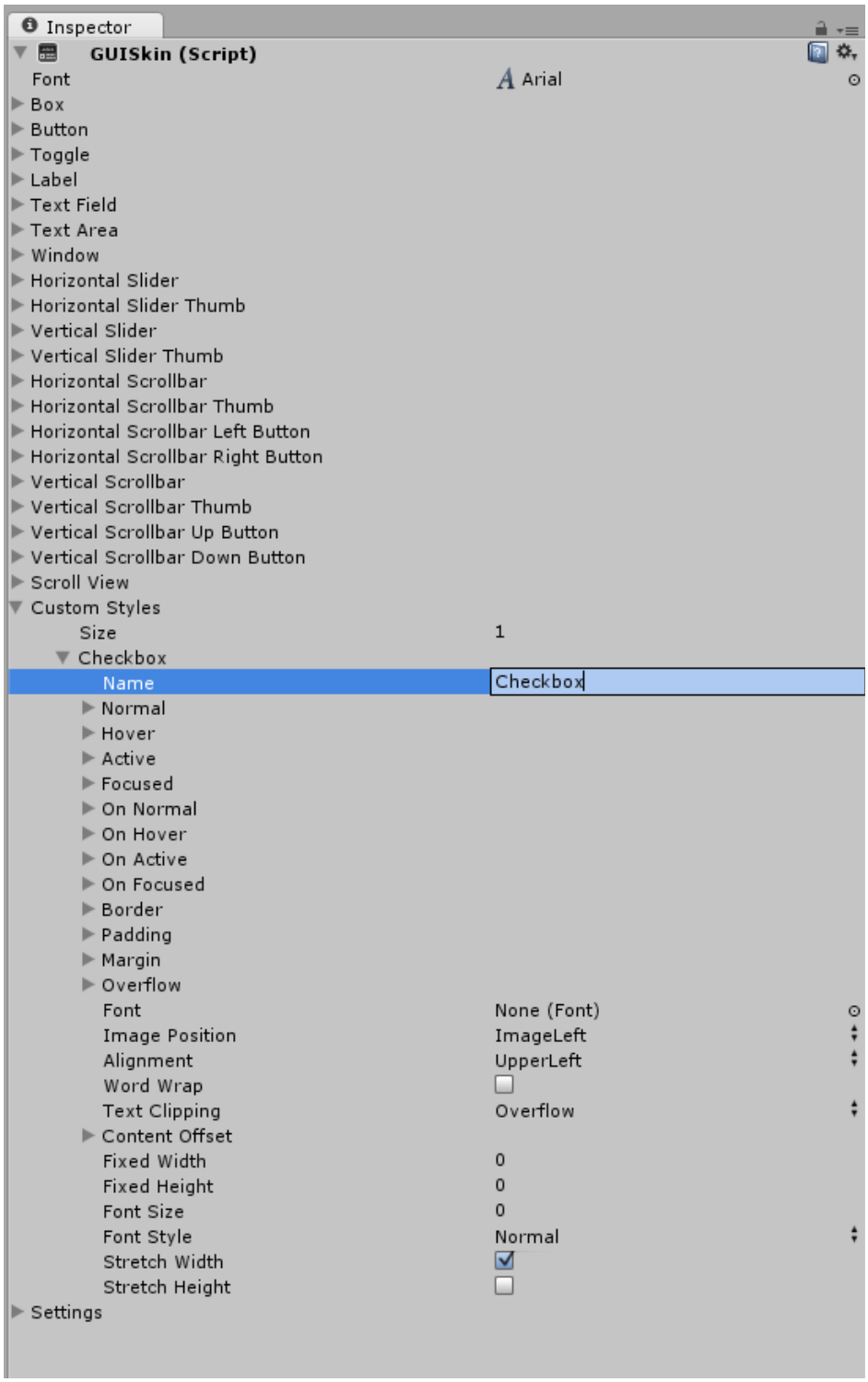
Now test the game.

You'll find it doesn't look exactly right.

This is because we still need to tell it to make use of the checkbox images that we have.

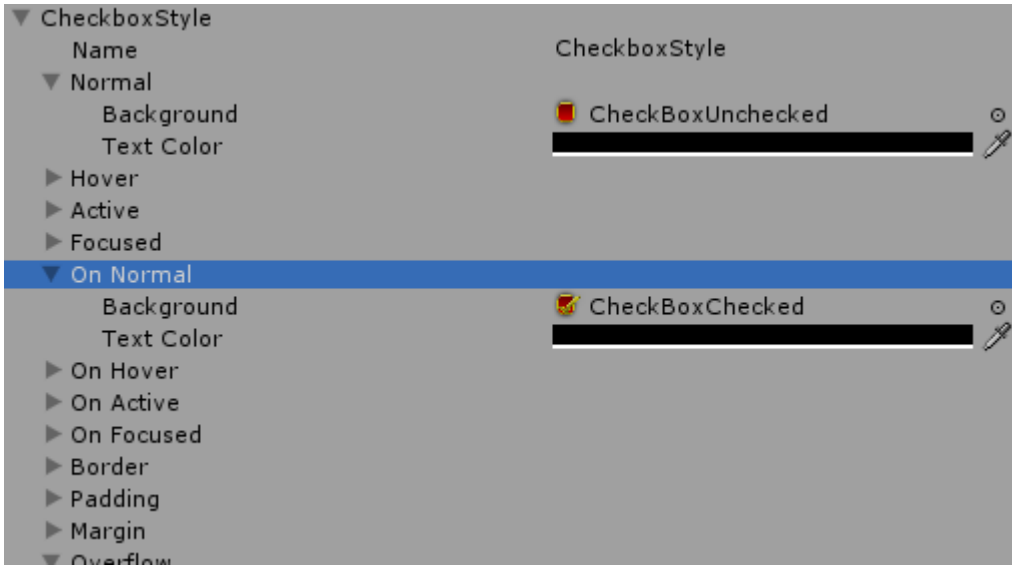
Go to the Unity Editor window. Select the GUI Skin that you have in the Project View. That's where we need to specify the checkbox look.

Click on the triangle beside “Custom Styles” to expand it. Expand “Element 0” the same way. Change its name to “CheckboxStyle”.



Expand its “Normal” and “On Normal” settings. Both of those will have a “Background” setting that

you can change. Those are where you should put the checkbox images. Put “CheckBoxUnchecked” on “Normal”. Put “CheckBoxChecked” on “On Normal”.



Now we need to tell our difficulty selection to make use of this custom style. Go back to your OptionsGui script and add this code:

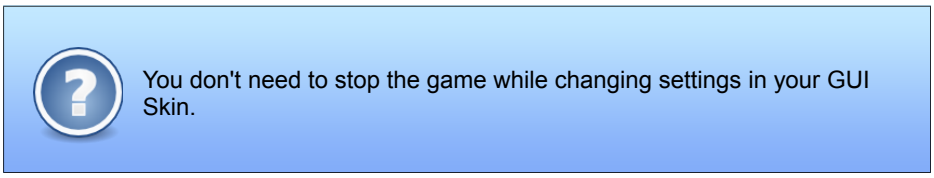
```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     static int _selectedDifficulty = 0;
22
23     static bool _isOpen = false;
24     static public bool IsOpen { get{ return _isOpen; } }
25
26     static public void Open()
27     {
28         _isOpen = true;
29     }
30
31     void OnGUI()
32     {
33         if (!_isOpen)
```

```
34          {
35              return;
36          }
37
38          GUI.skin = _guiSkin;
39
40          GUI.Box(new  Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
41
42          GUI.DrawTexture(new  Rect(20, 20, _title.width, _title.height), _title);
43
44          GUI.DrawTexture(new  Rect(50, 150, _titleDifficulty.width,
45                                    _titleDifficulty.height), _titleDifficulty);
46
47          _selectedDifficulty = GUI.SelectionGrid(new  Rect(100, 250, 280, 400),
48                                                  _selectedDifficulty, _difficulties, 1,
49                                                  "CheckboxStyle");
50
51          if  (GUI.Button(new  Rect(Screen.width - 30  - _ok.width - 10, Screen.
52              height - 30  - _ok.height - 10, _ok.width, _ok.height), _ok))
53          {
54              Close();
55          }
56      }
57
58      void  Close()
59      {
60          _isOpen = false;
61      }
62  }
```

You simply need to add the name of the style that you want to use as the last argument to GUI.SelectionGrid.

Test the game now.

The checkboxes are showing but they're still not being displayed correctly. We'll need to change some settings to the GUI Skin.



Go to your GUI Skin. Find your “CheckboxStyle”. Find the “Overflow” setting. Expand it to see the four settings underneath. Change them to the following values:

- Left: 0
- Right: -72
- Top: 0
- Bottom: 0

Find the “Fixed Width” and “Fixed Height” settings. Change them to these:

- Fixed Width: 192
- Fixed Height: 97

That fixes the size of the checkboxes. The only problem left is that the text is obstructing the checkbox image.

Find the setting called “Content Offset”. Expand it and change the settings to these:

- X: 77
- Y: 0

That's almost it. One last change is to adjust the Y properly.

Find the setting called “Alignment”. Change it to “MiddleLeft”.

That's it!

Adding GUI To Control Sound Volume

We actually don't have sound yet in our game but we'll prepare for it here. We'll add a slider that the user can use to control the volume of the sounds in the game.

Open OptionsGui and add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class OptionsGui : MonoBehaviour
05 {
06     [SerializeField]
07     GUISkin _guiSkin;
08
09     [SerializeField]
10     Texture2D _title;
11
12     [SerializeField]
13     Texture2D _ok;
14
15     [SerializeField]
16     Texture2D _titleDifficulty;
17
18     [SerializeField]
19     Texture2D[] _difficulties;
20
21     [SerializeField]
22     Texture2D _titleSound;
23
24     static int _selectedDifficulty = 0;
25
26     static float _soundVolume = 1.0f;
27
28     static bool _isOpen = false;
29     static public bool IsOpen { get{ return _isOpen; } }
30
31     static public void Open()
32     {
33         _isOpen = true;
```

```
34      }
35
36      void  OnGUI()
37      {
38          if  (!_isOpen)
39          {
40              return;
41          }
42
43          GUI.skin = _guiSkin;
44
45          GUI.Box(new  Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
46
47          GUI.DrawTexture(new  Rect(20, 20, _title.width, _title.height), _title);
48
49
50          GUI.DrawTexture(new  Rect(50, 150, _titleDifficulty.width,
51                                  _titleDifficulty.height), _titleDifficulty);
52
53          _selectedDifficulty = GUI.SelectionGrid(new  Rect(100, 250, 280, 400),
54                                              _selectedDifficulty, _difficulties, 1,
55                                              "CheckboxStyle");
56
57
58          GUI.DrawTexture(new  Rect(440, 150, _titleSound.width, _titleSound.
59                                  height), _titleSound);
60
61          _soundVolume = GUI.HorizontalSlider(new  Rect(490, 250, Screen.width-490-
62                                  50, 20), _soundVolume, 0.0f, 1.0f);
63
64          if  (GUI.Button(new  Rect(Screen.width - 30  - _ok.width - 10, Screen.
65              height - 30  - _ok.height - 10, _ok.width, _ok.height), _ok))
66          {
67              Close();
68          }
69      }
70
71      void  Close()
72      {
73          _isOpen = false;
74      }
75  }
```

GUI.HorizontalSlider is used to create a draggable slider. _soundVolume is what we use to store the current volume, a value from 0.0 to 1.0. With 1.0 being full 100% volume. A value of 0.5 would mean 50% volume.

The 1st argument to GUI.HorizontalSlider is its dimensions. Note that its made so that the width of the slider is snapped to the right edge of the screen. This is made using the formula found there (Screen.width-490-50). Screen.width holds whatever the screen resolution's width is. 490 is the same 490 used as its X-coordinate, and 50 is some padding to add space.

The 2nd argument to GUI.HorizontalSlider is the current value of the volume, so it knows where to display the slider's knob.

The 3rd and 4th arguments to GUI.HorizontalSlider specify the minimum and maximum return value. If the user dragged the slider's knob to the far right, GUI.HorizontalSlider would return the maximum

value. If it was dragged to the far left, it would return the minimum value. Here we specified 0.0 and 1.0 since the volume should be in that range only.

Save, go back to Unity and wait for it to recompile. Add the “TitleSound” from your Project View into the slot in the OptionsGui.

Now test the game.

Again, we'll need to change the GUI Skin to make use of our images. Select your GUI Skin and find the entries labeled “Horizontal Slider” and “Horizontal Slider Thumb”. Those are the styles that we need to change.

Expand “Horizontal Slider”. Like the others, find its “Normal”, and under that, “Background”. Drag the “SliderBg” from your Project View into that “Background” slot.

Expand “Horizontal Slider Thumb”. Here, expand the “Normal”, “Hover”, and “Active”. Change all their “Background” settings to the “SliderThumb” in your Project View.

Test the game. It won't look quite right yet. We'll need to adjust the default positioning.

Look again at “Horizontal Slider”. Change these values:

- Overflow
  - Left: 0
  - Right: 0
  - Top: 25
  - Bottom: -25
- Fixed Height: 55

Adjusting the Overflow this way moves the image upwards. We also give a fixed height of 55 since that's the image's original height in pixels. We wouldn't want it to be accidentally stretched or squashed.

In “Horizontal Slider Thumb”, change these values:

- Overflow
  - Left: 0
  - Right: 0
  - Top: 28

    - Bottom: -28
- Fixed Width: 62
- Fixed Height: 62

That should fix it.

Making The Options' Settings Persistent

Right now, when you exit the game, the values in the Options Screen revert to their original values. We need a way to save these values and load them back upon restart.

Unity provides an easy way to do this, using the PlayerPrefs class.

Saving Values

Add this code to the OptionsGui script:

```
                                                    .
                                                    .
                                                    .
36      void  OnGUI()
37      {
38          if  (!_isOpen)
39          {
40              return;
41          }
42
43          GUI.skin = _guiSkin;
44
45          GUI.Box(new  Rect(15, 15, Screen.width - 30, Screen.height - 30), "");
46
47          GUI.DrawTexture(new  Rect(20, 20, _title.width, _title.height), _title);
48
49
50          GUI.DrawTexture(new  Rect(50, 150, _titleDifficulty.width,
51                                    _titleDifficulty.height), _titleDifficulty);
52
53          _selectedDifficulty = GUI.SelectionGrid(new  Rect(100, 250, 280, 400),
54                                                  _selectedDifficulty, _difficulties, 1,
55                                                  "CheckboxStyle");
56          PlayerPrefs.SetInt("Difficulty", _selectedDifficulty);
57
```

```
58
59          GUI.DrawTexture(new  Rect(440, 150, _titleSound.width, _titleSound.
60                                  height), _titleSound);
61
62          _soundVolume = GUI.HorizontalSlider(new  Rect(490, 250, Screen.width-490-
63                                  50, 20), _soundVolume, 0.0f, 1.0f);
64          PlayerPrefs.SetFloat("SoundVolume", _soundVolume);
65
66
67          if  (GUI.Button(new  Rect(Screen.width - 30  - _ok.width - 10, Screen.
68               height - 30  - _ok.height - 10, _ok.width, _ok.height), _ok))
69          {
70               Close();
71          }
72      }
73
74      void  Close()
75      {
76          _isOpen = false;
77      }
78 }
```

In line 56, PlayerPrefs.SetInt stores whatever integer value you give it to a corresponding label, called, a key. If you know hash tables, it works the same way.

Here, PlayerPref keys are strings. We used “Difficulty” as our key for the difficulty selection. If we wanted to retrieve the value stored from there, we just use the same key when getting it.

That value is saved in the user's hard drive, but you don't need to worry about file names, folder paths, or what operating system your user is using. Unity handles that for you.

In line 64, PlayerPrefs.SetFloat works largely the same way, only that it stores a float instead of an integer.

Loading Values

To load back these values upon restart, we use the counterpart GetInt and GetFloat values. Add this code to the OptionsGui script:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  OptionsGui : MonoBehaviour
05 {
06      [SerializeField]
07      GUISkin _guiSkin;
08
09      [SerializeField]
10      Texture2D _title;
11
12      [SerializeField]
13      Texture2D _ok;
14
15      [SerializeField]
16      Texture2D _titleDifficulty;
17
18      [SerializeField]
```

```
19      Texture2D[] _difficulties;
20
21      [SerializeField]
22      Texture2D _titleSound;
23
24      static  int  _selectedDifficulty = 0;
25
26      static  float  _soundVolume = 1.0f;
27
28      static  bool  _isOpen = false;
29      static public  bool  IsOpen { get{ return  _isOpen; } }
30
31      static public  void  Open()
32      {
33           _isOpen = true;
34      }
35
36      void  Start()
37      {
38           _selectedDifficulty = PlayerPrefs.GetInt("Difficulty", 1);
39           _soundVolume = PlayerPrefs.GetFloat("SoundVolume", 1.0f);
40      }
41
42      void  OnGUI()
43      {
44           if  (!_isOpen)
45           {
46                return;
47           }
48
49           GUI.skin = _guiSkin;
                                                    .
                                                    .
                                                    .
```

As mentioned, we use the counterpart GetInt and GetFloat values. Also take note that we make sure to use the same key used when the values were saved.

In Conclusion...

You've made a Main Menu, and found out how to use custom GUI skins for your buttons. You also learned how to save and load values to the hard drive.

Next we'll work on adding visual effects and sound effects in the game.