

# Lab - User Interface



Goals

Resources

Basic Challenge

- Heads-up display

- Pause and game over menu

- Main menu

- Assessment

- Things to try (not required for assessment)

Intermediate Challenge

- Assessment

Master Challenge

- Assessment

# Goals

- To learn how to use create advanced UI in Unity

## Resources

- UI project (<https://altitude.otago.ac.nz/cosc360/UI>) on GitLab.
- Unity tutorial on the UI components (<https://learn.unity.com/tutorial/ui-components>)
- Unity Manual UI (<https://docs.unity3d.com/Manual/UISystem.html>)

## Basic Challenge

Every game needs a User Interface (UI). Development of UI in Unity used to be a chore. However, a new UI system has been introduced in Unity 4.6. It is powerful, flexible, and relatively framework that makes relatively easy to set up a decent looking UI (at least for the basic functionality). This lab will cover just the very basics, to get you started. You'll be creating a UI for the Tower Bridge Defence game, the example game showcasing the engine's 2D support (<https://learn.unity.com/tutorial/unity-for-2d-new-workflows-in-unity-4-3>).

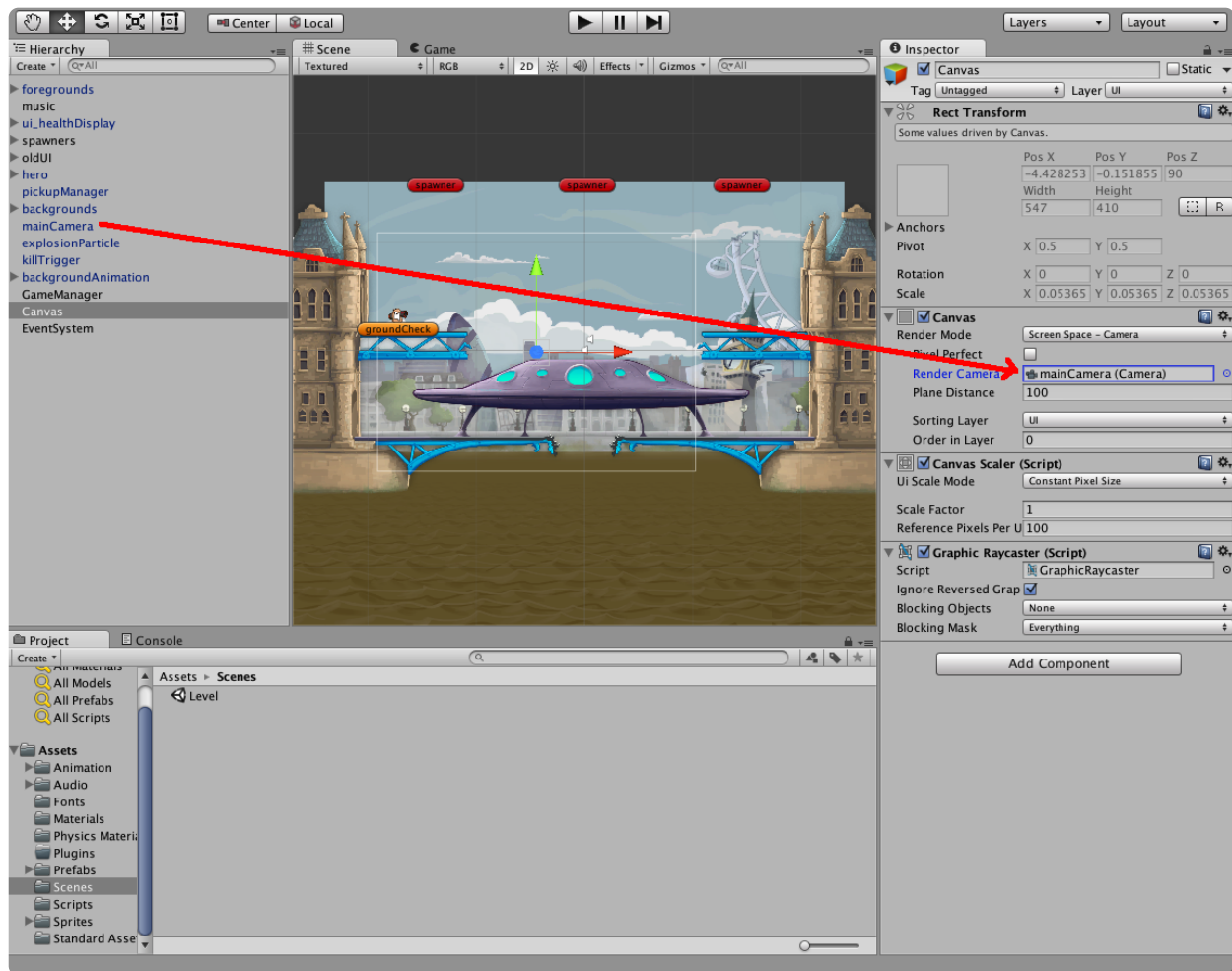
## Heads-up display

The assets provided by Unity for the Tower Bridge Defence game contain only one scene - the game's single level. Eventually you're going to create another scene for the main menu that will run on game start. However, at first you'll be modifying the existing "Level" to create a heads-up display (HUD).

- Fork the UI project (<https://altitude.otago.ac.nz/cosc360/UI>) on GitLab, then clone your fork to your hard-drive.
- Open the *UI* project, the directory of the cloned repository, in Unity. This project is based on the example Tower Bridge Defence game (with few modifications to disable the legacy UI).
- Load the "Level" scene from *Assets / Scenes* and hit play. Use arrow keys for movement, space for jumping, left shift for fire, alt for dropping bombs (I think). You'll notice that points are awarded for killing aliens (they flash briefly over the destroyed alien), but the total score is not displayed, and neither is the amount of health you have left. That's what HUD is for - to provide information about the state of the game that is of interest to the player. This exercise aims for something simple, as shown in the screenshot below - score on top-left, health bar on top-right:



- The base component of the new UI system is the canvas. Canvas is a game object, where all the UI elements go. You can have more than one canvas in the scene, but for this exercise just one will suffice. In the **Hierarchy panel** create a new *UI / Canvas* game object. You'll notice that an *Event System* (<http://unity3d.com/learn/tutorials/modules/beginner/ui/ui-events-and-event-triggers>) game object gets created as well. *Event System* is there to manage behaviour of the UI elements whenever user interacts with them - you won't be changing any of its default settings.
- In the **Hierarchy panel** select the *Canvas* game object. Set its *Canvas / Render* mode to *Screen Space - Camera*. Drag the *mainCamera* game object to *Canvas / Render Camera* property to have the canvas glued and resized to the game play screen. The outline of the canvas is the same as the outline of the camera.




Set the *Canvas / Sorting Layer* property of the *Canvas* game object to "UI". This will make the canvas render in front of everything else in the scene.

## Sorting Layers

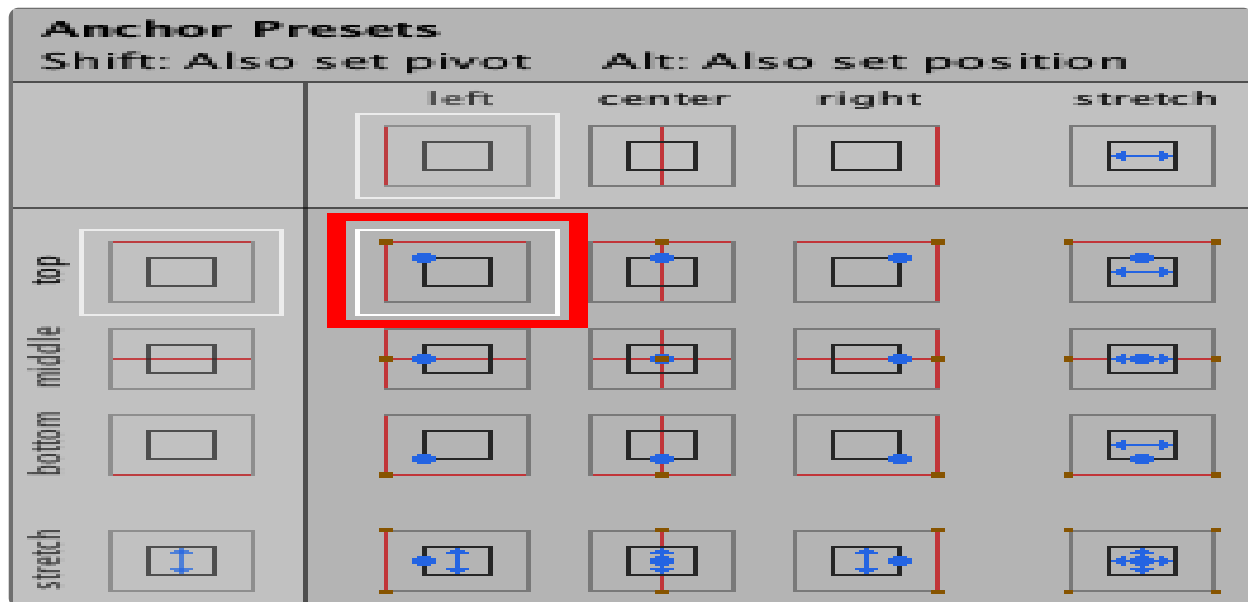


In Lab02, you have been controlling the order of rendering using the depth (Z position) of the sprites in relation to the camera. This works for all sprites within the same *Sorting Layer*. When sprites are assigned to different *Sorting Layers*, their relative depth doesn't matter - everything in one layer will render in front of (or behind) the things from the other layer. To see the rendering order of *Sorting Layers* select from the main menu *Edit / Project Settings / Tags and Layers*. In the **Inspector panel** expand *Sorting Layers*. The layer number gives the order of rendering. Sprites assigned to the "Background" (*Sorting Layer 2*) will get rendered before sprites assigned to the "Foreground" (*Sorting Layer 3*), meaning "Foreground" things will be on top of "Background" things. *Sorting Layer "UI"* is the last in rendering order, so it will be on top of everything. Sorting layers are quite handy when you want a fine control of rendering order and don't want to worry about depth of the game object.

In the **Hierarchy panel** create a *UI / Text* game object (it should appear under the *Canvas* in the hierarchy). Name it *hudScore*. Note that instead of *Transform* component there is *Rect Transform* component. All *UI* elements are positioned with the *Rect Transform* instead of the traditional *Transform*. This is because these elements are inherently 2D and they support a complex system for relative positioning and sizing with respect to their parents (in this case the *Canvas*). The purpose of this new positioning system, based on *Anchors* and *Pivots*, is to give the developer means of defining the look and behaviour of *UI* elements when resolution/size of the screen changes. This lab will not go into the details of how *Anchors* and *Pivots* work -

if you need to understand it, refer to the tutorials in the Resources section. However, you will rely on pre-defined anchor settings to help you with the layout. Click on the predefined anchors icon, , and

select the setting shown in the screenshot below:

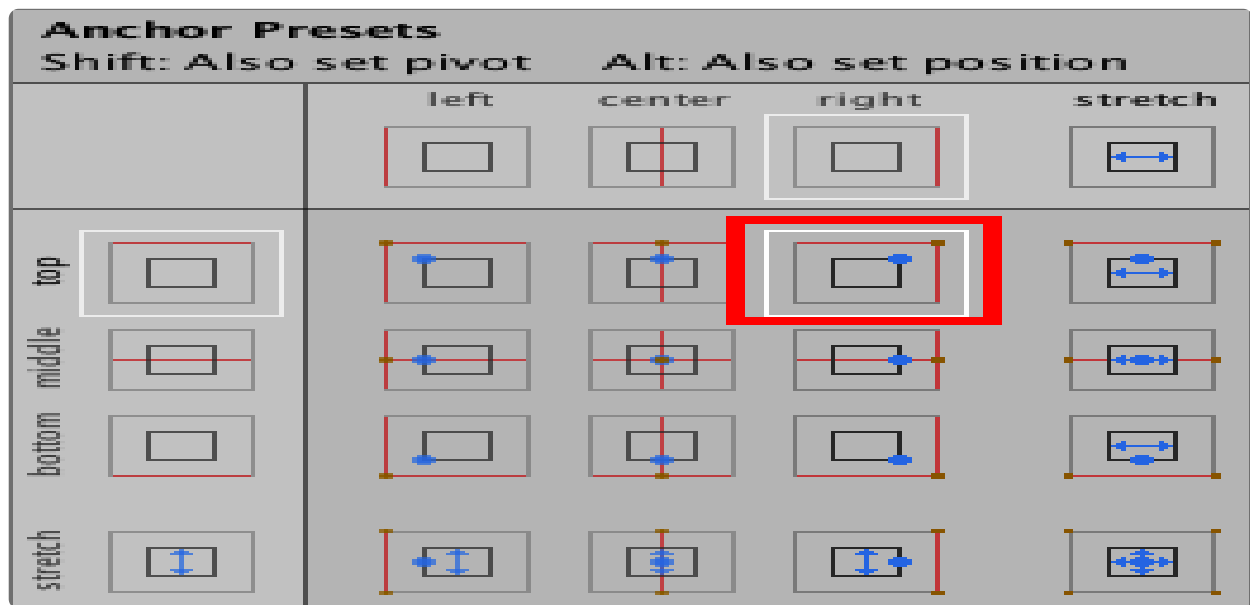


The anchor for this setting is in the left-top corner of the canvas. The X and Y position attributes of the *hudScore* game object will now give relative position of the UI element with respect to that corner. The element will maintain the same distance from that corner regardless of the screen size.

- Set the X and Y position of the *hudScore* so that it is located close the left-top corner of the canvas. I've positioned mine with the following *Rect Transform* settings: *Pos X*=267, *Pos Y*=-52, *Pos Z* = 0, *Width*=502, *Height*=80.
- Set the other properties of the *hudScore* game object as follows: its *Text / Text* to "Score", *Text / Font Size*=50, and *Text / Color* to white. Also, change the font (by clicking the little circle icon to the right of *Text / Font* box) and select "BradBunR" font from the list. This font has been already imported into the Assets of this project. However, importing other fonts to your project, if you find such a need, is as easy as importing sprites - just drag the font files into your *Assets*.
- In the **Hierarchy panel** create a *UI / Slider* game object. It should appear somewhere under the *Canvas* in the hierarchy. Name it *hudHealth*.
- ↳ Slider element comes with a number of children objects. One of them is not needed for this scene, because you're going to use this element to function as a progress bar rather than an interactive slider. Expand the hierarchy underneath *hudHealth* game object and delete (right-click and *Delete*) the *Handle Slide Area* child.
- Expand the *Fill Area* child of the *hudHealth* game object, and select the *Fill* game object. Set it's colour to something other than white - I've set mine to R=17, G=231, B=63.
- If you double click the *Fill Area* (not the *Fill*) game object in the **Hierarchy panel**, *Scene view* will zoom in on that object. Don't worry if other sprites from the scene disappear - this happens because UI canvas is actually positioned behind the scene (it still gets rendered on top, because of its assignment to the "UI *Sorting Layer*"). Take a look at the slider in *Scene view*. You should notice that the fill doesn't extend all the

way to the right in the slider window. The extra space is the space where the slider handle used to be. If you change the *Rect Transform / Right* of the *Fill Area* game object from 15 to 5 (to match the value of the *Left* property) the fill should now extend through the whole slider. The settings for the *Left* and *Right* specify the distance of the left and right edges of the *Fill Area* from the left and right edges of its parent, the *hudHealth* object. Again, if you'd like more details on how *Anchors* work refer to tutorials from the *Resources* section.

1. If you want to see the entire scene again (with all sprites), double click on *mainCamera* in the **Hierarchy** panel.
5. Anchor the *hudHealth* object to the right-top corner of the canvas (the anchor setting as shown in the screenshot below). Set the *Rect Transform* properties to the following: *Pos X*=-99, *Pos Y*=-36, *Pos Z*=0, *Width*=160, *Height*=20.



5. Player's health value is a number between 0 and 100. You need to set the minimum and maximum value of the health slider accordingly. Select *hudHealth* and set its *Slider / Min Value* to 0 and *Slider / Max Value* to 100.
7. Create a new C# script. Name it "HUDManager" and paste in the following code:

**HUDManager.cs**

```

01: using UnityEngine;
02: using System.Collections;
03: using UnityEngine.UI;
04:
05: public class HUDManager : MonoBehaviour {
06:
07:     // References to UI elements on the canvas
08:     public Text hudScore = null;
09:     public Slider hudHealth = null;
10:
11:     // References to objects that provide
12:     // information about score, health and
13:     // the game over condition
14:     Score scoreInfoProvider;
15:     PlayerHealth healthInfoProvider;
16:     Remover gameOverInfoProvider;
17:
18:     // Health value currently displayed
19:     float health;
20:
21:     // Use this for initialization
22:     void Start () {
23:         // Initilaise references to the game objects
24:         // that provide informaiton about the score,
25:         // health and game over condition
26:         scoreInfoProvider = FindObjectOfType<Score>();
27:         healthInfoProvider = FindObjectOfType<PlayerHealth>();
28:         GameObject[] objArray = GameObject.FindGameObjectsWithTag("gameoverTrigger");
29:         gameOverInfoProvider = objArray[0].GetComponent<Remover>();
30:
31:         // Set the starting health value for display
32:         health = healthInfoProvider.health;
33:     }
34:
35:     // Update is called once per frame
36:     void Update () {
37:         // Display the score
38:         hudScore.text = "Score: " + scoreInfoProvider.score;
39:
40:         // Display health - but rather than doing it in one go, change the value
41:         // gradually (over certain period of time)
42:         health = Mathf.MoveTowards(health, healthInfoProvider.health, 20*Time.deltaTime);
43:         hudHealth.value = health;
44:     }
45: }

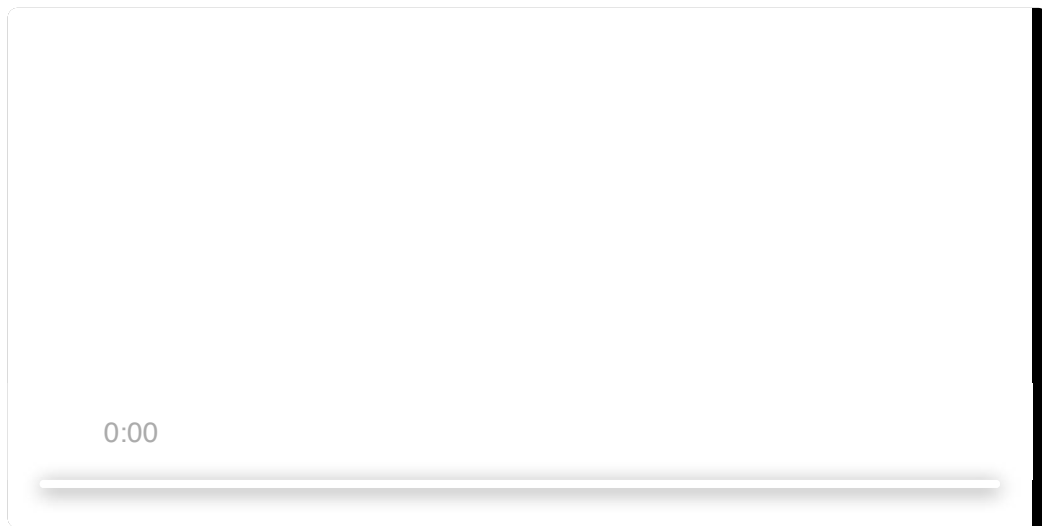
```

The "using UnityEngine.UI" imports the UI library, which allows the use of Text and Slider types as opposed to UnityEngine.Object and UnityEngine.UI.Object. The two public variables are references to the HUD elements you just placed on canvas (these references we'll get initialised in the **Inspector panel**).

The three variables ending with **InfoProvider**, are references to objects, which contain information on the current score, player's health, and game over condition. These references get initialised in the *Start()* method. For this tutorial it's not important to understand how these classes work (they can be found in *Assets / Standard Assets* by the way). All you need to know is that **scoreInfoProvided.score** gives the current score, **healthInfoProvider.health** gives the player's health (between 0 and 100), and **gameoverInfoProvider.gameover** is a boolean that will be **true** when game is over.

The **health** variable contains the currently displayed value for player's health. It is set to **healthProvider.health** in the *Start()* method. The first line in the *Update()* sets the text of the **hudScore** object to display the current score. To compute the **health**, for displaying in the health bar, the **Mathf.MoveTowards()** (<http://docs.unity3d.com/ScriptReference/Mathf.MoveTowards.html>) function is used. This function moves **health** towards **healthInfoProvider.health** gradually, producing a smooth sliding effect whenever the player loses or gains health. If you don't like that effect, replace the last two lines in *Update()* with **hudHealth.value = healthInfoProvider.health** (the slider will instantly show the new health value whenever it changes).

3. Select *Canvas* game object in the **Hierarchy panel** and add the *HUDManager* script component to it. Drag the *hudScore* and *hudHealth* game objects over the *Hud Score* and *Hud Health* variables in the **Inspector panel**.
4. Play the game - the new HUD should keep track of the total score and remaining health.



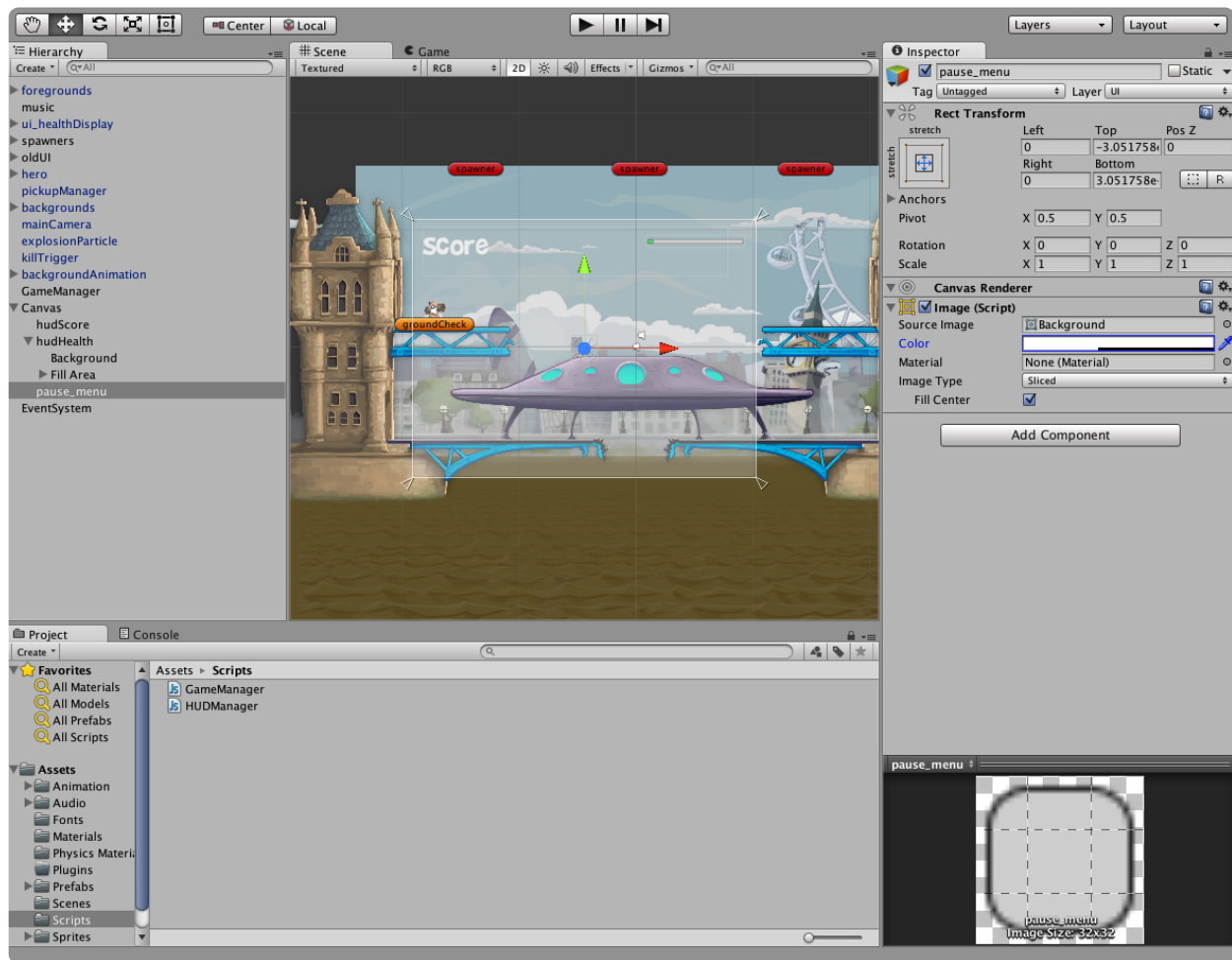
## Pause and game over menu

Next, you'll be creating a pause menu. It will also function as the game over menu. When displayed during gameplay, this menu will pause the game and give the player an option to resume or quit. The menu will also appear when game over condition is detected, providing the restart and quit options. This menu will consist of a UI panel element, placed on the same canvas where HUD elements reside. The only difference will be that the panel will be deactivated (invisible) by default and made active (visible) when user presses the ESC key, or runs out of health.





3. In the **Hierarchy panel** create a new *UI / Panel* game object. Name it *pause\_menu*. Since all UI elements must go on canvas, this new game object should be automatically made a child of the *Canvas* game object. Make sure that *pause\_menu* is below *hudScore* and *hudHealth* objects in the hierarchy - you want it to render on top of HUD.

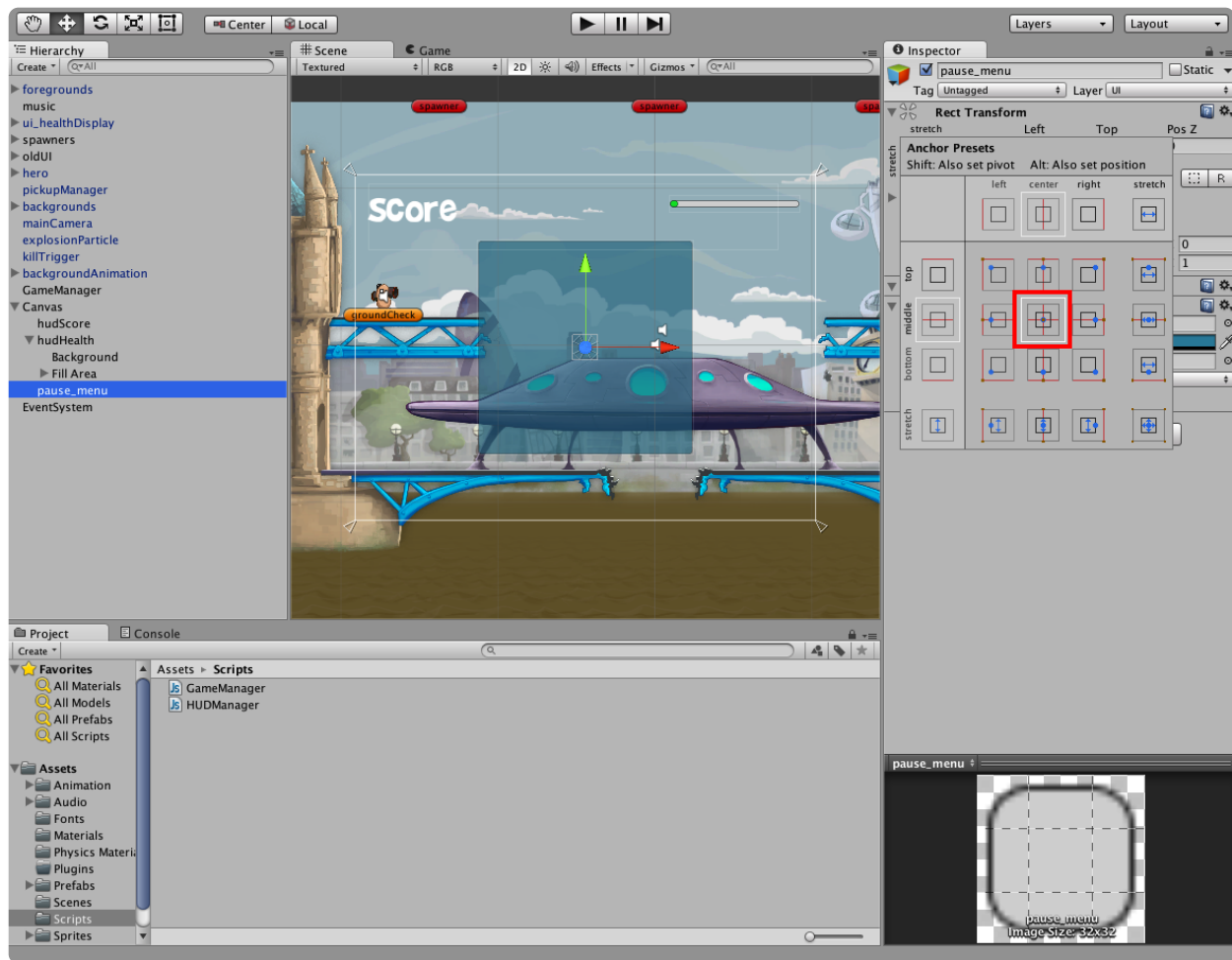


## Rendering order of UI elements



The order of UI elements in the **Hierarchy panel** dictates their rendering order on the canvas. This is only true for UI elements - hierarchy doesn't affect rendering order of other game objects. Therefore, the UI element that you want to show up on top of other UI element must be the last child of the canvas game object.

1. Change the Color and alpha mask of the *Image / Color* property of the *pause\_menu* game object to colour of your choice. I've set mine to  $R=54$ ,  $G=138$ ,  $B=165$ ,  $A=162$ .
2. By default, a *UI / Panel* object fills out the entire canvas, but that's not desired for this pause menu. Anchor the *pause\_menu* object to the middle of the canvas without resizing (anchor selection as shown in the screenshot below). Set the *Rect Transform* values to *Pos X=0*, *Pos Y=0*, *Pos Z=0*, *Width=264*, and *Height=264*.



3. With the *pause\_menu* game object selected in the **Hierarchy panel**, create a new *UI / Text* game object - it should appear as a child of the *pause\_menu* in the hierarchy (if it doesn't, drag it there yourself). Name it *resume\_text*. Set its *Text* properties as follows: *Text*="Resume", *Font* to "BradBunR", *Font Size*=40, *Alignment* to centred and *Color* to white. Position the element near the top of the pause menu panel - in my project I've set the *Rect Transform* properties as follows: *Pos X*=0, *Pos Y*=62, *Pos Z*=0, *Width*=196, and *Height*=56.
4. Duplicate the *resume\_text* game object in the **Hierarchy panel**. Rename the duplicate to *quit\_text*. Change its *Text* / *Text* string to "Quit" and position it somewhere below the "Resume" text in the pause panel - I just changed the *Rect Transform* / *Pos Y* to -51.



5. Create a new C# Script. Name it "PauseMenuManager" and paste in the following code:

**PauseMenuManager.cs**

```

001: using UnityEngine;
002: using System.Collections;
003: using UnityEngine.UI;
004: using UnityEngine.SceneManagement;
005:
006: public class PauseMenuManager : MonoBehaviour {
007:
008:     // References to text objects on the panel
009:     public Text resumeText = null;
010:     public Text quitText = null;
011:
012:     // Array storing text object with index
013:     // indicating the current selection
014:     int optionIdx = 0;
015:     Text[] optionArray;
016:
017:     // Indicates whether the game in paused mode
018:     bool pauseGame;
019:
020:     // Use this for initialization
021:     void Start () {
022:         // We have two text object in the panel,
023:         // so create an array with 2 references
024:         // and set the first referect ot resumeText
025:         // and second reference to quitText
026:         optionArray = new Text[2];
027:         optionArray[0] = resumeText;
028:         optionArray[1] = quitText;
029:     }
030:
031:     // Show the pause menu in pause mode (the
032:     // first option will say "Resume")
033:     public void ShowPause() {
034:         // Pause the game
035:         pauseGame = true;
036:         // Set the text of the first option
037:         // to "Resume"
038:         resumeText.text = "Resume";
039:         // Show the panel
040:         gameObject.SetActive(true);
041:     }
042:
043:     // Show the pause menu in game over mode (the
044:     // first option will say "Restart"
045:     public void ShowGameOver() {
046:         // Set the text of the first option
047:         // to "Restart"
048:         resumeText.text = "Restart";
049:         // Show the panel
050:         gameObject.SetActive(true);
051:     }

```

```

052:
053: // Hide the menu panel
054: public void Hide() {
055:     // Deactivate the panel
056:     gameObject.SetActive(false);
057:     // Resume the game (if paused)
058:     pauseGame = false;
059:     Time.timeScale = 1f;
060: }
061:
062:
063: // Execute a command base on command string (that string
064: // corresponds to text of the entered option
065: void ExecuteCommand(string command) {
066:
067:     switch(command) {
068:
069:         // For resume option, just hide the panel
070:         // (the pausegame flag will be set to false)
071:     case "Resume":
072:         Hide();
073:         break;
074:
075:         // For the restart option just reload current scene
076:     case "Restart":
077:         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
078:         break;
079:
080:         // For the quit option load the main menu scene
081:     case "Quit":
082:         SceneManager.LoadScene("MainMenu");
083:         break;
084:
085:     }
086: }
087:
088: // Update is called once per frame
089: void Update () {
090:     // Get a reference to the currently selected text box
091:     Text currentSelection = optionArray[optionIdx];
092:
093:     if(Input.GetKey(KeyCode.DownArrow)) {
094:         // When user presses down arrow, go to next option
095:         optionIdx++;
096:     } else if(Input.GetKey(KeyCode.UpArrow)) {
097:         // When user presses up arrow, go to previous option
098:         optionIdx--;
099:     } else if(Input.GetKey(KeyCode.Return) || (Input.GetAxis("Jump") != 0f) ) {
100:         // If user presses Enter or "Jump" key (Space), execute
101:         // the command corresponding to the current option
102:         ExecuteCommand(currentSelection.text);
103:     }

```

```

104:
105:     // Make sure that the option index indicator is within the range
106:     // of the number of options
107:     if(optionIdx < 0) {
108:         optionIdx = 0;
109:     } else if(optionIdx >= optionArray.Length) {
110:         optionIdx = optionArray.Length-1;
111:     }
112:
113:     // Set the font colour of the all option text boxes to white
114:     for(int i = 0; i < optionArray.Length; i++)
115:     {
116:         optionArray[i].color = Color.white;
117:     }
118:     // Set the font colour of the currently selected text box
119:     // to yellow
120:     currentSelection.color = Color.yellow;
121:
122:     // If game is in pause mode, stop the timeScale value to 0
123:     if(pauseGame) {
124:         Time.timeScale = 0;
125:     }
126: }
127: }
128:

```

The **resumeText** and **quitText** variables are references to two text boxes you just placed on the panel. The **optionArray** stores these two references in an array, so that they can be referenced by index of the current selection stored in the **optionIdx** variable. The **pauseGame** variable indicates whether the game is in the pause mode or not.

The **optionArray** variable is initialised on *Start()*. The two text element references are saved into the array.

In the *ShowPause()* method, the pause flag is set to true and the string value of the first text box to "Resume". The **gameObject** variable is a built-in reference to the object whose component the current script happens to be. For this script, it will be the *pause\_menu* game object. Passing the value of **true** to the *SetActive()* (<http://docs.unity3d.com/ScriptReference/GameObject.SetActive.html>) method of **gameObject** will make the pause panel visible. The panel is also activated in the *ShowGameOver()* method, but with the text of the first text box set to "Restart", and the pause flag untouched.

In the *Hide()* method, the panel is deactivated (made invisible). The *pauseGame* flag is set to false and time scale restored to 1 so that game state machines resume.

The *ExecuteCommand()* method implements actions for different options (available as a choice in the menu). The string passed in as the argument corresponds to the text of the active text box when Enter was pressed. On "Resume", the panel will become hidden (and the game resumed). On "Restart", the scene will get reloaded. On "Quit", a new scene will be loaded - it will be the "MainMenu" scene that you have yet to create.

The *Update()* function implements the controls for the menu. The down and up arrow keys change the selection of the current text box in the panel by changing the value of the **optionInx** variable. Pressing "Return" or "Space" results in a call to *ExecuteCommand()* with the string of the current selection text box passed in as an argument. The font colour of the selected text box is set to yellow. Finally, if the *pauseGame* is set, time scale is set to 0, which freezes some (but not all) aspects of the game engine.

## Pausing the game



When **Time.timeScale** is set to zero, the game engine stops executing the *FixedUpdate()* methods. It will still continue to execute the *Update()* and *OnGUI()* methods. *FixedUpdate()* is similar to *Update()* (<http://unity3d.com/learn/tutorials/modules/beginner/scripting/update-and-fixedupdate>), but it is invoked by the game engine at fixed time interval as opposed to per every frame.

3. Add the *PauseMenuManger* script component to the *pause\_menu* game object. Drag the *resume\_text* and *quit\_text* game objects over the *Resume Text* and *Quit Text* variables as shown in the screenshot below:



7. Add the following code to *HUDManager* script:

**HUDManager.cs**



```

01: using UnityEngine;
02: using System.Collections;
03: using UnityEngine.UI;
04:
05: public class HUDManager : MonoBehaviour {
06:
07:     // References to UI elements on the canvas
08:     public Text hudScore = null;
09:     public Slider hudHealth = null;
10:
11:     // References to objects that provide
12:     // information about about score, health and
13:     // game over condition
14:     Score scoreInfoProvider;
15:     PlayerHealth healthInfoProvider;
16:     Remover gameOverInfoProvider;
17:
18:     // Health value currently displayed
19:     float health;
20:
21:     // Reference to UI panel that is our pause menu
22:     public GameObject pauseMenuPanel;
23:     // Reference to panel's script object
24:     PauseMenuManager pauseMenu;
25:
26:     // Use this for initialization
27:     void Start () {
28:         // Initilaise references to the game objects
29:         // that provide informaiton about the score,
30:         // health and game over condition
31:         scoreInfoProvider = FindObjectOfType<Score>();
32:         healthInfoProvider = FindObjectOfType<PlayerHealth>();
33:         GameObject[] objArray = GameObject.FindGameObjectsWithTag("gameoverTrigger");
34:         gameOverInfoProvider = objArray[0].GetComponent<Remover>();
35:
36:         // Set the starting health value for display
37:         health = healthInfoProvider.health;
38:
39:         // Initialise the reference to the script object, which is a
40:         // component of the pause menu panel game object
41:         pauseMenu = pauseMenuPanel.GetComponent<PauseMenuManager>();
42:         pauseMenu.Hide();
43:     }
44:
45:     // Update is called once per frame
46:     void Update () {
47:         // Display the score
48:         hudScore.text = "Score: " + scoreInfoProvider.score;
49:
50:         // Display health - but rather than doing it in one go, change the value
51:         // gradually (over certain period of time)

```

```

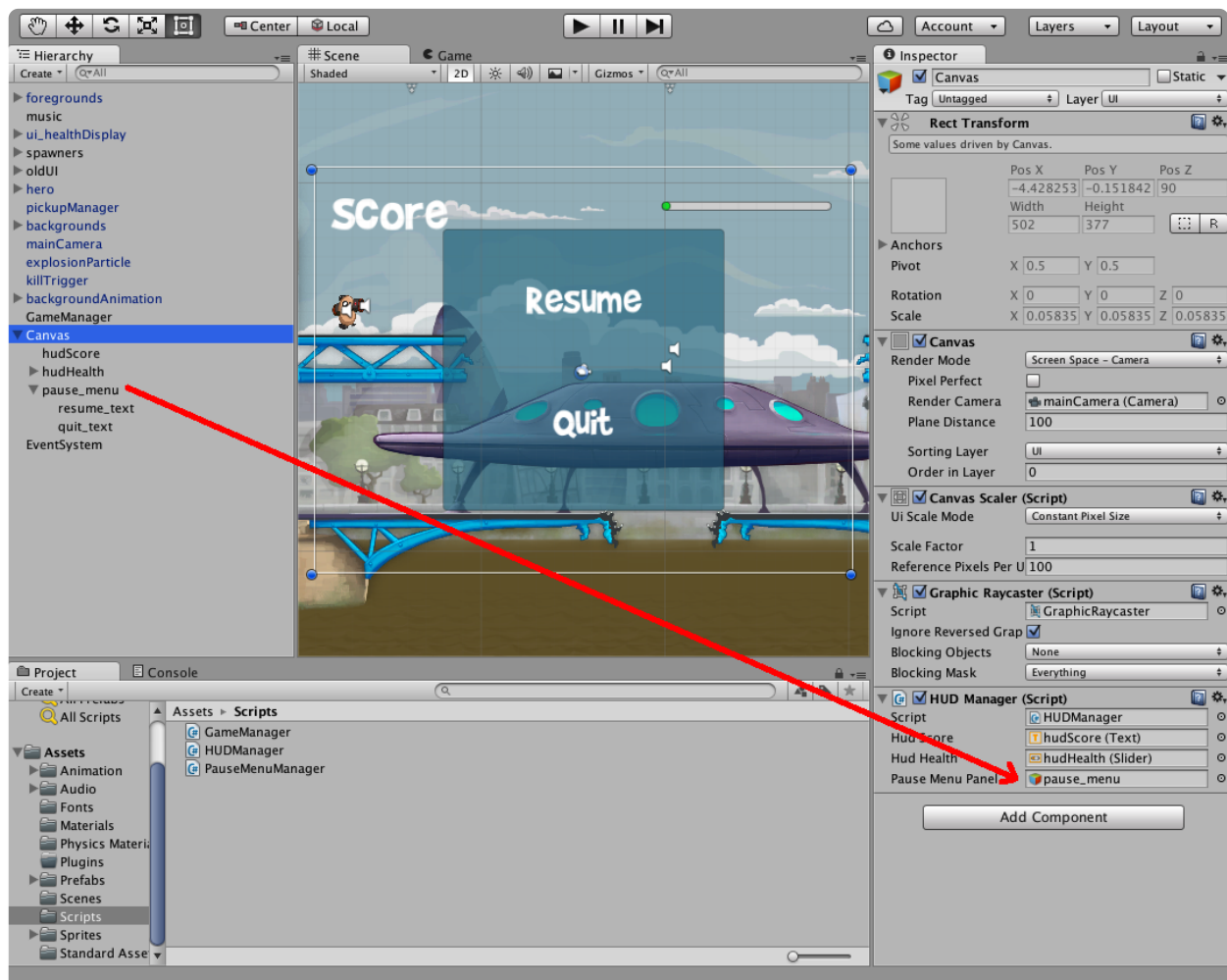
52:         health = Mathf.MoveTowards(health, healthInfoProvider.health, 20*Time.deltaTime);
53:         hudHealth.value = health;
54:
55:         if(gameoverInfoProvider.gameover) {
56:             // If gameover state detected, show the pause menu in gameover mode
57:             pauseMenu.ShowGameOver();
58:         } else if(Input.GetKey(KeyCode.Escape)) {
59:             // If user presses ESC, show the pause menu in pause mode
60:             pauseMenu.ShowPause();
61:         }
62:     }
63: }

```

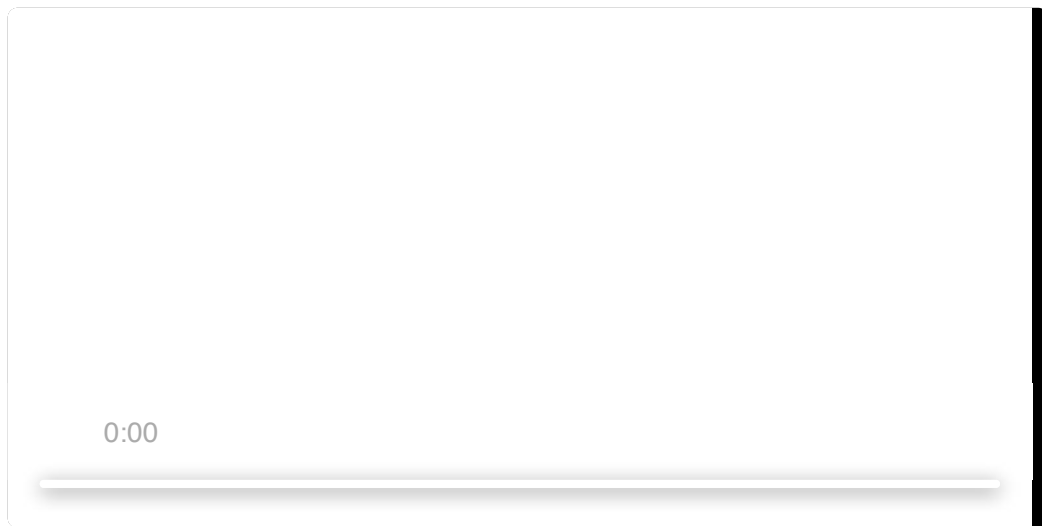
The `pauseMenuPanel` is a reference to the panel game object (you'll initialise it in a bit in the **Inspector** panel). In the `Start()` method, this reference is used to fetch a reference to the instance of its `PauseMenuManager` script component. This will enable the current script to use the `Hide()`, `ShowPause()` and `ShowGameOver()` methods in order to hide and show the panel.

The panel is hidden on `Start()`. When user presses the ESC key, it's shown in the pause mode. When game over condition is detected it's shown in the game over mode.

3. After saving the above changes, the `Pause Menu Panel` variable should appear in `HudManager` component of the `Canvas` game object. Drag the `pause_menu` game object over it as shown in the screenshot below:



- 3. At the moment, the game gets automatically reloaded when the game over condition is detected by the *GameManager* object. Since you have just configured the "HUDManager" script to display the pause panel on game over, you need to disable this automatic level reload. Select the *GameManager* object and deactivate it by clearing the checkbox near its name in the **Inspector panel**. The object is deactivated and its script will not be executed.
- 4. You might want to deactivate the *pause\_menu* game object as well, so that it doesn't cover everything in the *Scene view*. It gets deactivated on scene load anyway in the *Start()* method of the *HUDManager* script.
- 5. Play the game. Press ESC to pause and resume. If you select the "Quit" option, nothing will happen as the "MainMenu" scene hasn't been created yet. That's what you're going to do next.



## Main menu

Time to create a new scene for the main menu. The design of the menu is going to be very simple - just two buttons to start the game or quit the application, plus a bit of graphics. It's going to look like this:



OK, so I named the game "Big Ben Bean". If you can come up with a better name, feel free to use yours.

2. Create a new scene. Save it to *Assets / Scenes* and name it "MainMenu".
3. Create new *UI / Canvas*. Set its *Render Mode* to *Screen Space - Camera* and drag the *Main Camera* game object over the *Render Camera* property. Set the *Sorting Layer* to "UI".
4. Create a new *UI / Image* game object (it will go under the *Canvas*). Name it *env\_bg*. For its *Source Image* select the "env\_bg" sprite. You want this image to cover and stretch with the canvas (when screen size changes). Set the anchor and its parameters to setting shown in the screenshot below:
5. Create a new *UI / Image* game object. Name it *env\_BigBen*. For its *Source Image* select the "env\_BigBen" sprite. You want this image to stay at the bottom of the canvas while stretching to fill it horizontally. Set the anchor and its parameters to setting shown in the screenshot below:
6. Create a new *UI / Text* element. Name it *title\_txt*. Leave it anchored to the centre and set its *Rect Transform* properties as follows: *Pos X*=0, *Pos Y*=100, *Pos Z*=0, *Width*=400, *Height*=120. Set the properties of its *Text* component as follows: *Text*="Big Ben Bean" (or a better name), *Font*="BradBunR", *Font Size*=80, *Alignment* to centred and *Color* to white.
7. Create a new *UI / Image* game object. Name it *beanicon* and drag it so that it's a child of the *title\_txt* object. For its *Source Image* select the "beanicon" sprite. You want this image to stay near the left-bottom corner of the title (covering the 'B' a bit). It will get anchored to its parent, *title\_txt*, so set the anchor and its parameters to the settings shown in the screenshot below:

At this point, the hierarchy of the UI elements should be as shown in the screenshot below:

3. With the *Canvas* game object selected in the **Hierarchy panel** create a new *UI / Button* element. Name it *start\_btn*. Leave it anchored to the centre. Set its *Rect Transform* properties so that *Pos X=0*, *Pos Y=11*, *Pos Z=0*, *Width=153*, *Height=54*.
3. Buttons come with a *Text* child. Expand *start\_btn* in the **Hierarchy panel** to get to its *Text* child and its properties as follows: *Text="Start"*, *Font="BradBunR"* and *Font Size=40*.
3. Duplicate the *start\_btn* game element, rename it *quit\_btn*. Set the *Rect Transform / Pos Y* of the new button to -94, and the *Text* of its child object to "Quit".
4. That's the layout of the UI elements on the canvas. Next, you need to create methods to execute on buttons' *OnClick()* events.
2. Create a new C# Script. Name it "MainMenuManager" and paste in the following code:

### MainMenuManager.cs

```
01: using UnityEngine;
02: using System.Collections;
03: using UnityEngine.SceneManagement;
04:
05: public class MainMenuManager : MonoBehaviour {
06:
07:     public void StartGame () {
08:         // Load the "Level" scene
09:         SceneManager.LoadScene("Level");
10:     }
11:
12:     public void QuitGame() {
13:         // Quit the application
14:         Application.Quit();
15:     }
16: }
```

These are very simple functions that are to be executed when the "Start" and "Quit" buttons are clicked. They just need to be connected to appropriate UI elements.

3. Add the *MainMenuManager* script component to the *Canvas* game object.
4. Select *start\_btn* in the **Hierarchy panel**. At the bottom of the **Inspector panel** there should be an *OnClick()* window stating that "List is Empty". Press the + sign below to create a new entry. Drag the *Canvas* game object onto the window where it says "None (Object)". Now you'll be able to connect the event to methods from *Canvas* script components. Click on the box where it says "No Function" and select *MainMenuManager.StartGame()*.

5. Do the same for the *quit\_btn*, except connect it to *MainMenuManager.QuitGame()*.
6. Lastly, don't forget to add the new scene to *Build Settings* by selecting *File / Build Settings*. The "Level" scene should be on the list of *Scenes to Build* already. Press the *Add Current* button. The "MainMenu" scene should appear on the list. Drag it to the top of the list - you want to start the game with the main menu.
7. Play the game to try everything out. You need to use the mouse to interact with the buttons in the main menu. This is a bit inconsistent with the way the in-game pause menu works (which is controlled with the keyboard). This has been done in this lab in order to showcase different UI elements. When you develop your own game, consistency will be very important to the overall user experience.
8. The "Quit" button in the main menu will not do anything when you play the game in Unity Editor. That's because the call to **Application.Quit()** works only in a built game. If you want to test the functionality of that button run *File / Build & Run* from Unity Editor's main menu to play a standalone build of the game.

## Assessment

Show your work to the demonstrator for assessment.

## Things to try (not required for assessment)

- Play with the settings of UI elements to change the look and feel of the menu.
- Add a slider that controls music volume to the pause menu (the background music is a component of the *music* game object in the "Level" scene).

## Intermediate Challenge

Do a detailed UI design for an arbitrary game. The game could be the Big Ben Bean from the Basic Challenge, the game you are developing with your team, or any imagined game. If you choose to re-design UI for Big Ben Bean, it needs to be much better than the simple and boring UI you were implementing in the previous challenge. For this challenge you need detailed sketches of all the UI elements and descriptions of the functionality - mock up screens done in Pixelmator or Photoshop would be great. You need to consider, and include explanation of how it's going to be used (for instance if there are keyboard shortcuts, etc). Get some user feedback on your design (from your teammate or friends). Collect everything into a portfolio. This work should be such that anyone could pick up the portfolio and implement your vision.

## Assessment

Show your design to the demonstrator for assessment.

# Master Challenge

- Implement the entire UI from the Intermediate Challenge in Unity. The the UI elements should be entirely yours, drawn/created from scratch.

or

- Devise and complete your own UI Master Challenge - just check with the lecturer or the demonstrator first, whether the scope of the work will be sufficient for the awarded skill points.

## Assessment

Show your work to the demonstrator for assessment.