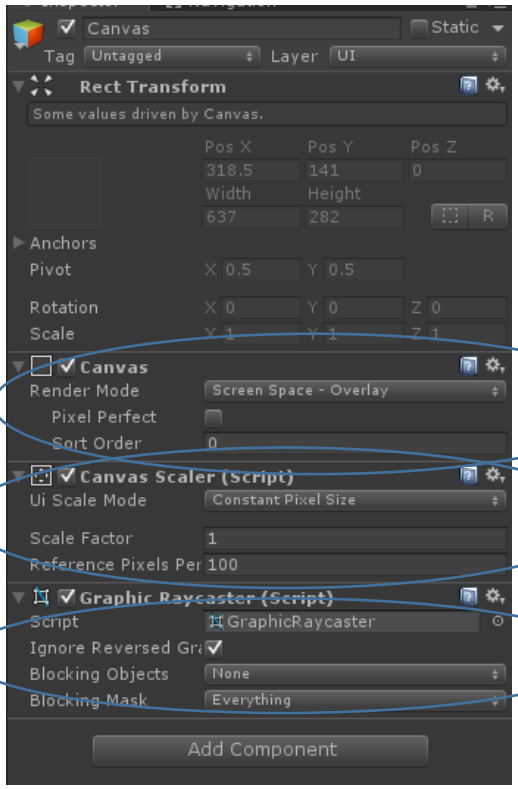
Creating an in game UI in Unity

This tutorial uses the new UI system to do a few common things in Unity, we show elements like score, time, health, etc.

Canvas

If you select it in the inspector, you'll see a whole bunch of options. As seen below, these allow you to configure all sorts of options. The key ones are highlighted;



Render Mode: allows you to choose between an overlay of the whole scene, being fixed to it's own camera or having it's own space in the world. Usually you want the first option

Canvas Scaler- how big you want it to be. A constant pixel size means no matter how many pixels the game window is it will be the same size. This isn't the best idea.

Instead it's better to set it to scale with screen size.

Graphic Raycaster- used to check if any parts have been clicked. Usually something you don't want to touch.

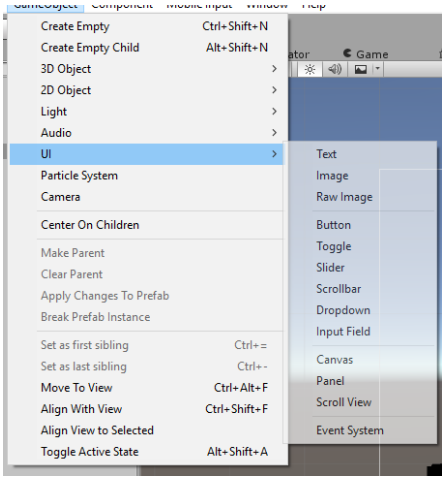
Now that's explained, set your UI Scale Mode to Scale with Screen Size and your reference resolution to 1920 x 1080.

If you double click the canvas, you'll see a white box, that white box represents the boundaries of what is seen by the player

Health bar and text

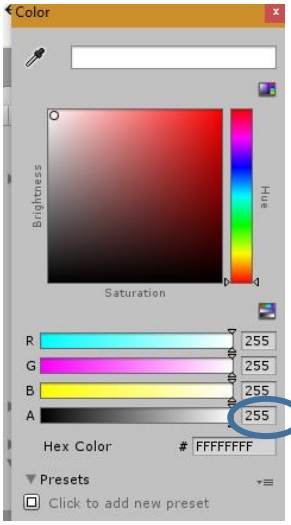
At the moment, we have no GUI and no way of knowing how much health we have. Let's get on and fix this.

Add a Slider by going to go to GameObject-> UI->Slider as shown in the screenshot to the right. If you've not already added a Canvas it will add it automatically for you. A slider is normally used as something the user can slide up and down to change values, but funnily they make quite nice looking health Bars.



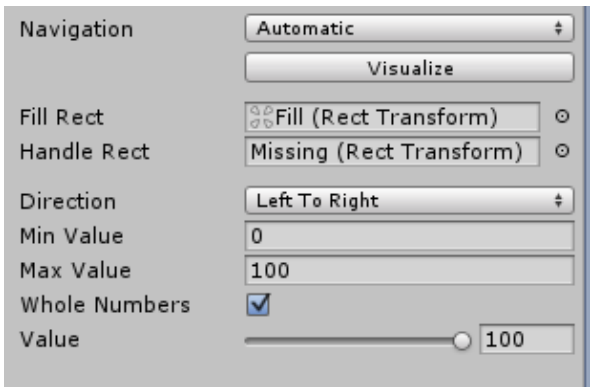
We can move the slider around like any other GameObject, so move it up to the top left corner. You'll notice it still has the "handle" on it, the bit that lets up move the value, let's get rid of that. Select the Slider, expand, click Handle Slide area and just delete the whole thing.

At the moment, the bar is white, that's not what we want; we want a coloured health bar. Click the background object and in the inspector click the white box labelled colour and a box similar to the right should appear. That's the backing colour. Change the circled bit to 0 and that will give you a transparent background



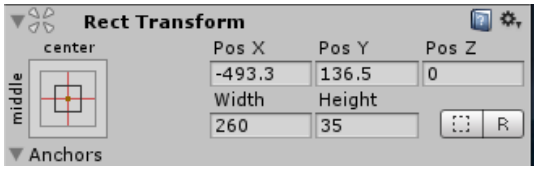
Now find Fill Area in the hierarchy and Fill and do the same process but change it to green.

One last thing, select the slider, you should, at the bottom, see some settings like the below. We're going to pay attention to the min value and max. Setting them to 0 and 100 and set the Value (representing the current value) to 100.

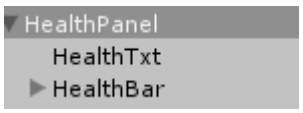


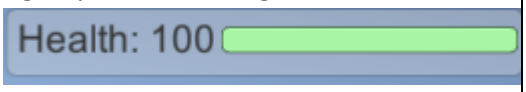
This should result in a nice pretty health bar, green coloured and looking pretty. It doesn't do much though. We need to update our health script to use it.

Before we do that we'll add some text to make it clearer. A slider looks nice but can be hard to read, we want to add a health text value. Click on Canvas and add a UI->Text item, and position it how you'd like. The settings for text are fairly self-explanatory so I won't go into too much detail, keep an eye out for best fit which alters the sizes to fit the box whatever you resize the text box too. Make your text a bright colour so it's clear and delete the "Next Text" bit, (although not really necessary). In the text bit, add Health: 100 and resize them so they're around the same size.



Finally, add a Panel, which is just an image that we can use as a background, click the little picture on the right, which gives you the options relating to how it will resize. I chose from the center. Position the bar in the top left then drag the slider and text onto it to parent it, this will give you a nice background behind the bar. Here's mine with each bit renamed appropriately.





Scripting some logic

If you're using the basic health script set up earlier we're not doing anything fancy, just a max health, a current health variable and some logic to get the health value, check if they're dead and destroy the thing if you are.

Create a new script "UIScript" this will hold all our UI stuff, so, we're going to first use it to update those health values.

First, you need to add a "using UnityEngine.UI" right at the top, this lets Unity know you're using the new UI features (not so new anymore) and to allow access to all its features in the script.

First, we need to create some variables, we'll need access to the health script and the text and slider that we're using to show the health values.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class UIScript : MonoBehaviour {

    public Health healthScript;
    public Text healthTxt;
    public Slider healthBar;
```

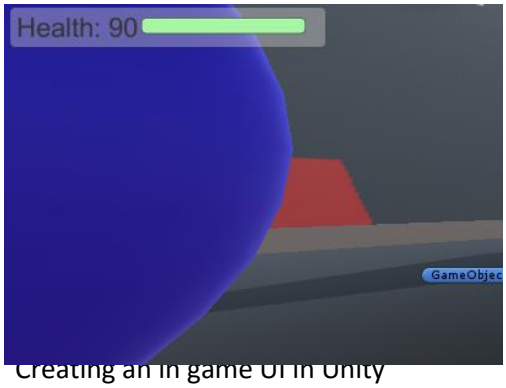
Here you can see the variables setup and the using declaration at the top, yours should look very similar to this, the variable names are straight forwards.

Now, when we start we want to update the values of the health bar and text, fairly simple, I use two helper methods in health script to do this, one that gets max health, one current. This protects us from being able to accidentally overwrite health (as we would be able to if we directly accessed it using a public variable. We also make sure the maxValue of the slider is the same as the maxHealth.

```
// Use this for initialization
void Start () {
    healthBar.maxValue = healthScript.getMaxHealth();
    healthBar.value = healthScript.getHealth();
    healthTxt.text = "Health: " + healthScript.getHealth();
}
```

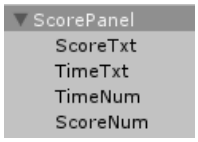
When you press start you should see the bars update, but when you're damaged nothing will change. To solve this we move it into Update. In theory we should only update when the health changes, but for this example (and quite often) it has little effect.

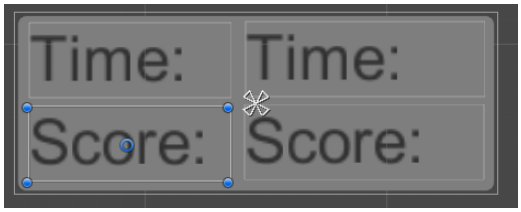
Now try it, attach it to your player and drag the objects to the fields, you'll need some way to damage your player, if you're really struggling there's two easy ways. The first is to change your currentHealth variable in the script to something less than 100, the second is to make the currentHealth variable a SerializeField. Here is an example of how it looks.



Score and Time

There's a lot more that a UI can hold, we'll show a couple of examples, the current score and the current time since the game began. Add a panel and to the panel, add two new text fields,





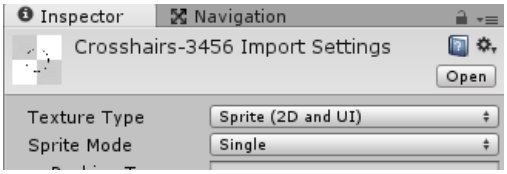
The first box is the text label for the two different boxes, the second boxes are the numbers that will go in there, the time and the score respectively. We add three new fields, scoreNum, timeNum, score, the first two are just used to hold the Text UI elements, the last one is a static variable to hold the score. A static variable means that only one copy of that variable exists no matter how many copies of the script are created, the variable belongs to the class- that is to all the "UIScripts" the benefit of this is that we can access it from anywhere. Especially if we make a public static function like updateScore that lets us call "UIScript.updateScore(50)" for instance to add 50 to the score from anywhere. We then add a bit to update to make the text of the time box to be equal to the Time the level has been running, we add (int) to force it into a an integer and thus whole number.

```
public Text scoreNum;
public Text timeNum;
static int score;
```

```
public static void updateScore(int amount)
{
    score += amount;
}

// Update is called once per frame
void Update () {
    healthBar.value = healthScript.getHealth();
    healthTxt.text = "Health: " + healthScript.getHealth();
    timeNum.text = "" + (int)Time.time;
    scoreNum.text = score + "";
}
```

Then, in any script that you want to update the score in you just call the updateScore method. See the example below, I've just modified the Damage function in Health script to check if the thing it's attached to is not the Player, if it's not then add 50 to the score then destroy it. You can use this wherever, pick ups are similar, when you trigger the pickup, add to the score.

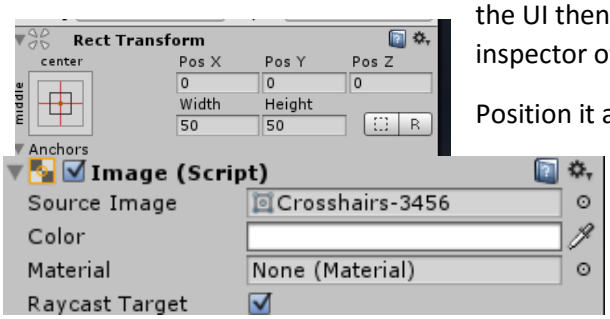
```
public void Damage(int damageValue)
{
    currentHealth -= damageValue;

    if (currentHealth <= 0) {
        if(!(gameObject.tag=="Player"))
        {
            UIScript.updateScore(50);
        }
        Destroy(gameObject);
        }
}
```



Crosshair

We need a crosshair that is in the middle of our screen (or wherever), right click on canvas and add an Image UI element. Then, find your crosshair in project resources along with this tutorial, make sure the settings when you click on it are set to "Sprite (2D and UI)" or you won't be able to use it in the UI then drag it onto the source image field in the inspector of the image component.

Position it at 0,0 and resize it, 50,50 seems to fit perfectly.



Now we can see how much health we have, see when we're losing it and see where we're shooting. This gives us a nicer feel to the game and more idea of what is going on.

GameManager

Create a new blank gameObject, call it GameManager and add a new script called GameManager to it. This will be used for a few different things but for the moment, we'll use it to feed UI references to the Player if we make a prefab for it. Tag that object "GameManager".

```
public class GameManager : MonoBehaviour {

    public Slider playerHealth;
    public Text score;
    public Text playerHealthTxt;
    public Text timeTxt;
```

A very simple script, just holding references to the UI elements, each is public so it's able to be seen from other scripts.

Update your UIScript's Start function to look like the below, this way, your player will always reference the correct place.

```
void Start () {
    GameManager manager = GameObject.FindGameObjectWithTag("GameManager").GetComponent<GameManager>();

    healthBar =manager.playerHealth;
    healthTxt = manager.playerHealthTxt;
    scoreNum = manager.score;
    timeNum = manager.timeTxt;

    healthBar.maxValue = healthScript.getMaxHealth();
    healthBar.value = healthScript.getHealth();
    healthTxt.text = "Health: " + healthScript.getHealth();
}
```

Challenges

Challenge 1: Add an in game panel that shows your controls

Challenge 2: Reskin your health bar to make it look nicer (change the sprites)

Challenge 3: For the faster enemy created in earlier challenges change the amount of score you give. This is done by creating a new variable for score given and update score with that variable.

Challenge 4: When you die, instead of destroying your character have a panel pop up with a restart button. This is much more complicated and requires you to create a panel and set it to not active when you start and to activate when you die. This isn't hugely complicated but does require some thinking. You will need to look at how to utilise buttons properly too, "SceneManager.LoadScene (SceneManager.GetActiveScene ().name);" will load the current scene (you need to use Using UnityEngine.SceneManagement;".