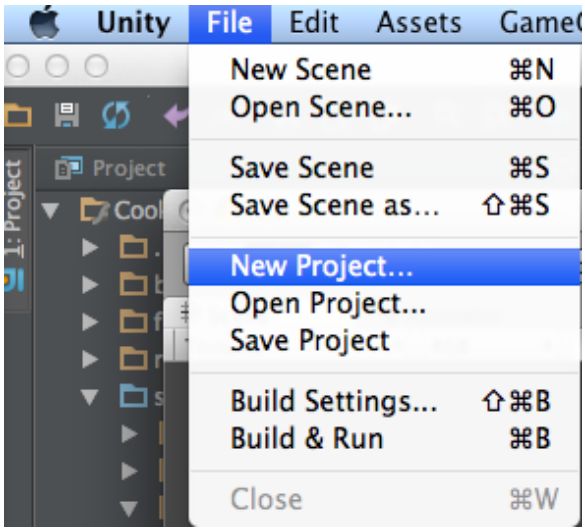
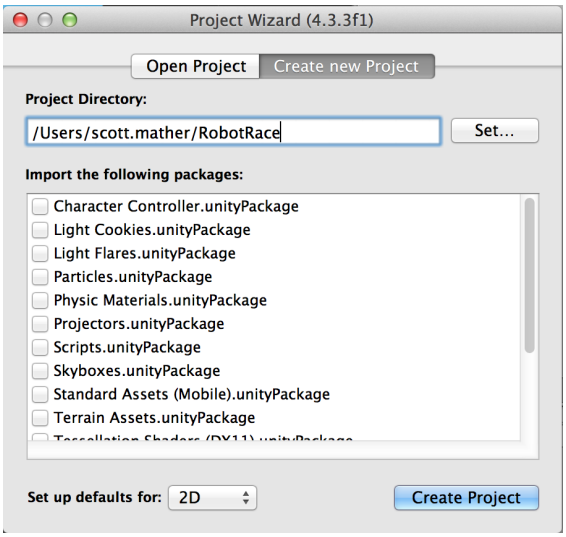
Robot Race

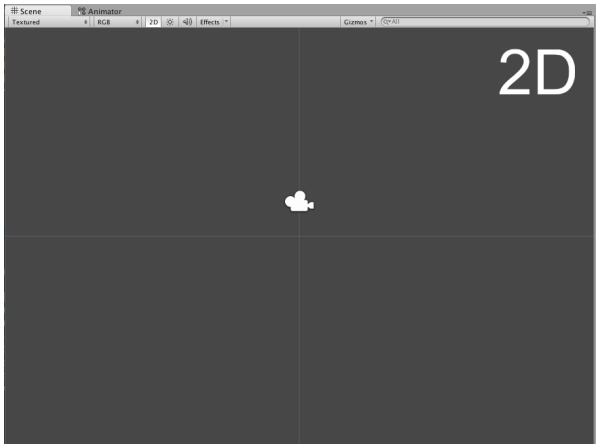
1. Create a new unity 2D Project.

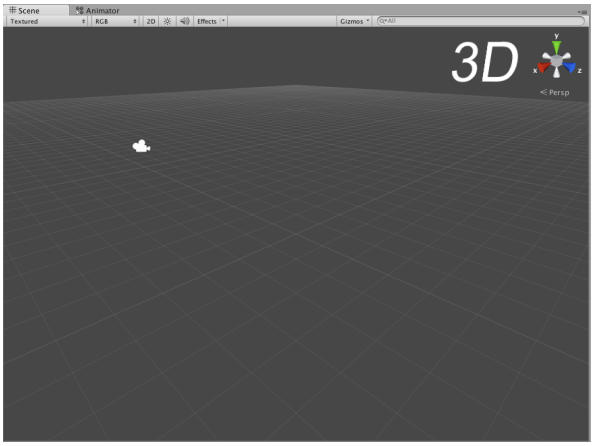
To create a new project in unity, select ‘File > New Project’ and then in the window that pops up choose a name for your project and make sure you change 3D to 2D in the dropdown. You can ignore all of the Import package options and just click ‘Create Project’.



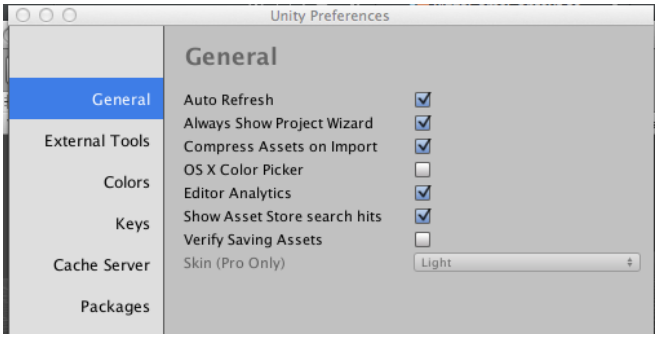


Note: There is a bug in some versions of Unity where even after selecting 2D in the Project Wizard, the actual project created is still set up as 3D. You will notice this because the Scene view will look quite different! Below you can see a comparison of the 3D and 2D scene views:



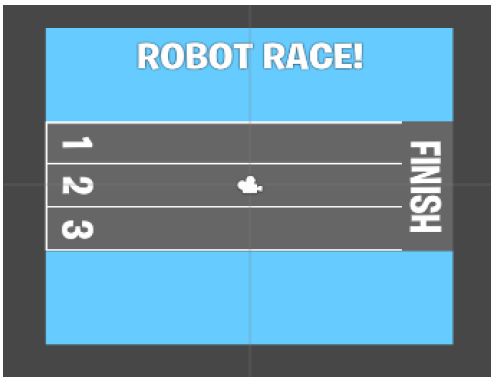


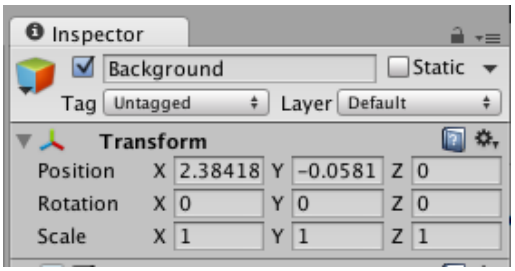
You can fix this bug by going to ‘Unity > Preferences’ in the main menu and under General check “Always Show Project Wizard”. Now if you quit Unity and reopen it you will be greeted by the same project wizard as earlier but this time when you try to create a 2D project it will work. The bug seems to be related to creating a project whilst another project is open.



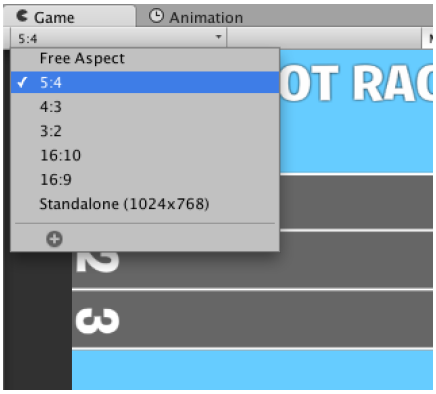
2. Adding the assets.

Adding images into your 2D project is very simple! Firstly locate the assets you downloaded with this tutorial and under the Images folder drag the Background.png into the scene view. You can reposition the image in various ways, you can click and drag the image around or edit the x and y positions of the image in the Inspector on the right hand side of the screen.

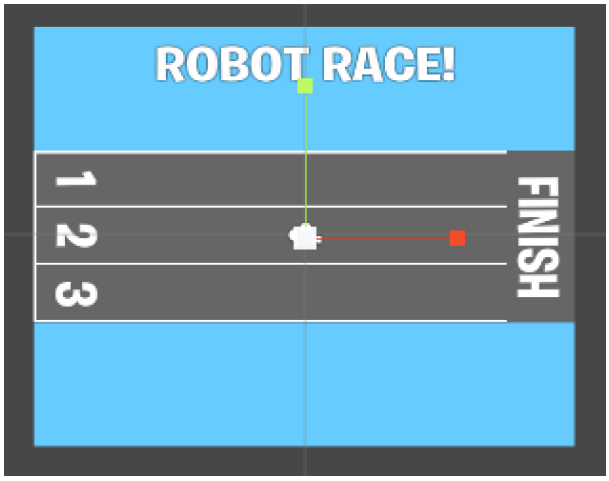




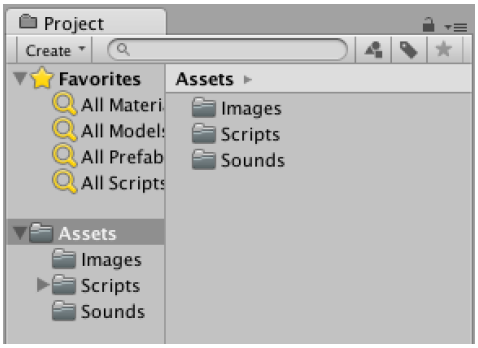
In the game view you will see the image against a blue background. It is recommended that you change the Aspect of this view to something other than Free Aspect so that you can see the edges of the screen. For this tutorial we will go with 5 : 4.



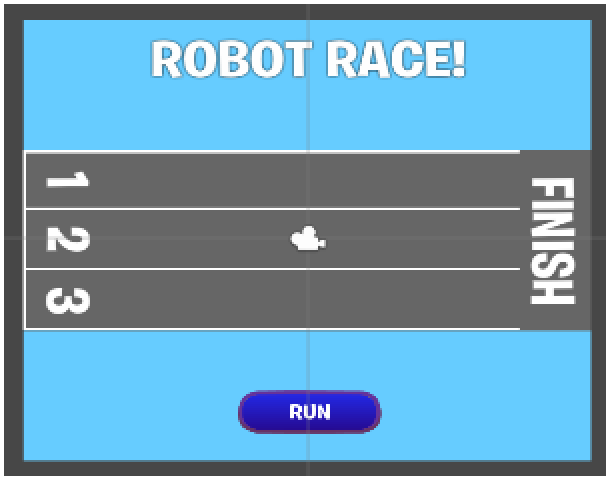
Now that you can see the edges of the screen let's scale the background image so it fits the full view. To scale you can drag the corners of the image, however If you change to the scaling tool (shown below) you can click the white box in the centre of the transform and scale it uniformly (by the same amount) in both directions. You can also change the scale in the Inspector window.



To make the rest of the tutorial easier you can drag the remaining assets that you downloaded into the Assets window of unity so that we can quickly access them later.

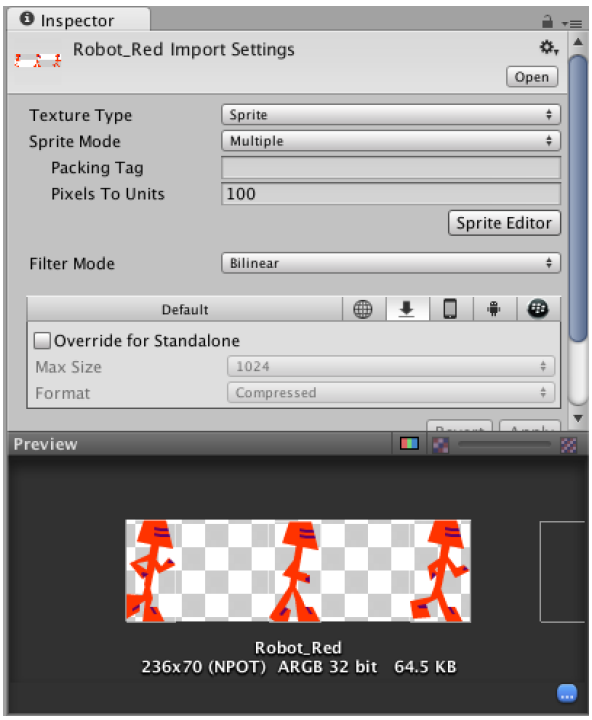


Next, drag the Run_Button.png from the 'Assets/Images' folder into our scene and position it like so:

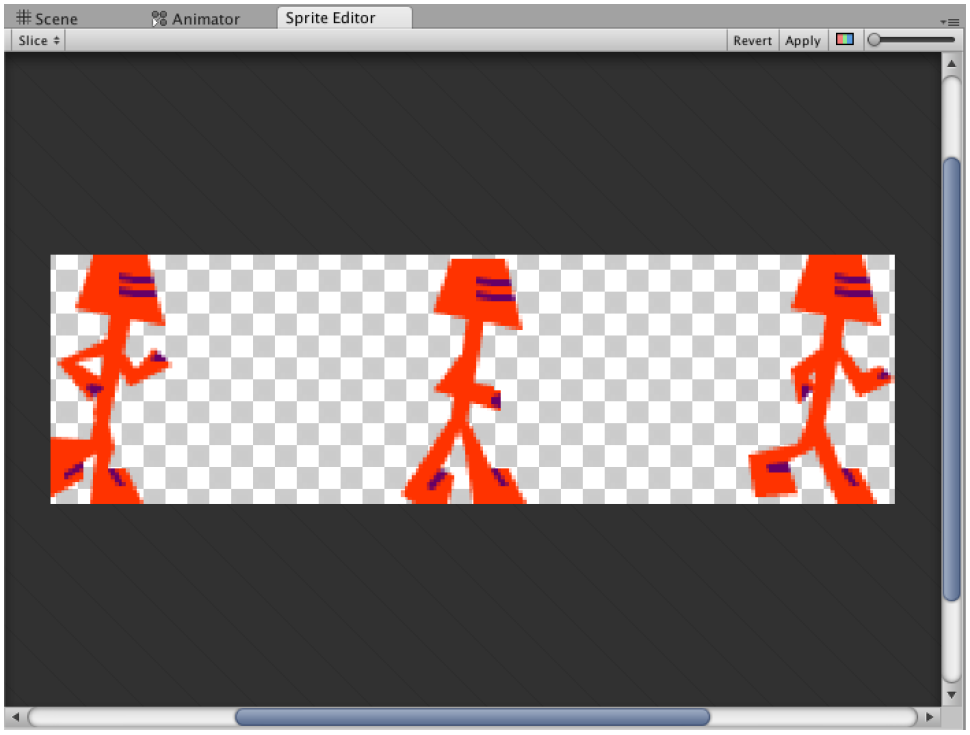


3. Create the player animation.

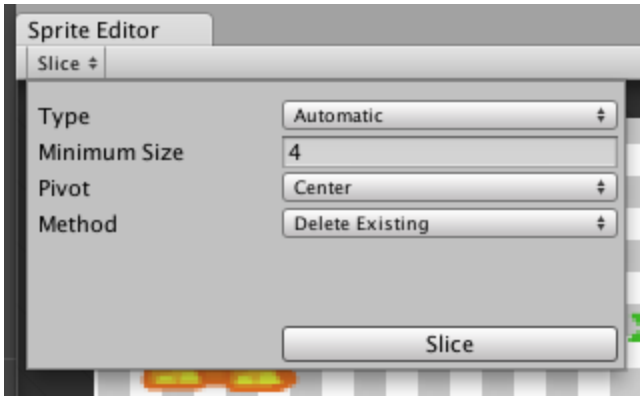
So far we have just added static images to our scene but let's add an animation now. In the 'Assets/Images' folder select one of the 3 Robot images and you will see lots of information appear in the Inspector Window.



As our image contains multiple frames we need to change the 'Sprite Mode' property from 'Single' to 'Multiple' and then click on the 'Sprite Editor' button that appears. This should cause the Sprite Editor window to appear:



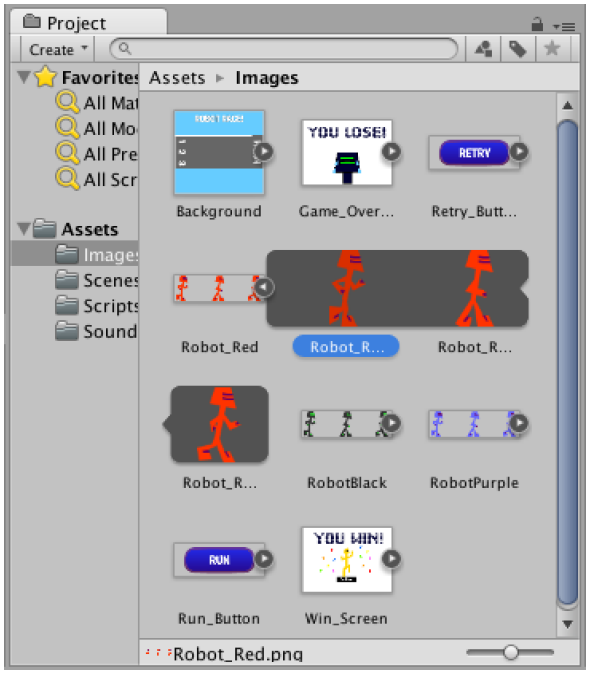
From here you can either use your cursor to click and draw rectangles around the images to create 4 rectangles/sprites or, the more efficient way is to click the ‘Slice’ button at the top left.

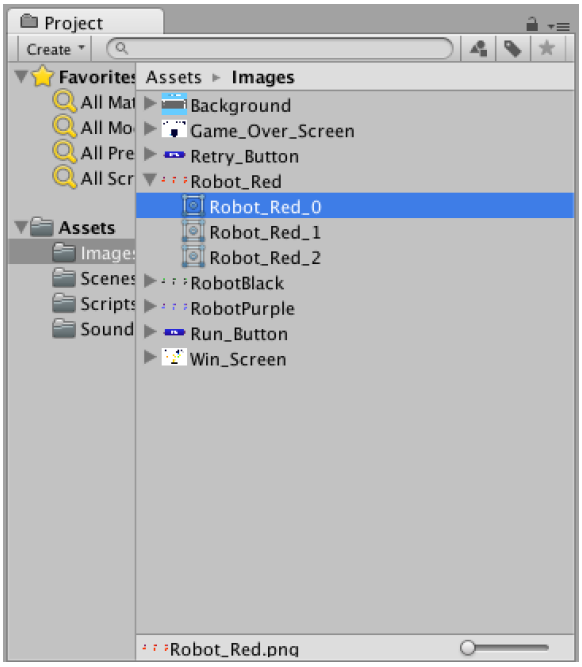


From here you can just ignore the options click the new ‘Slice’ button. It should automatically create rectangles around the images for you. In order to save these changes you need to click the ‘Apply’ button in the top right of the window:

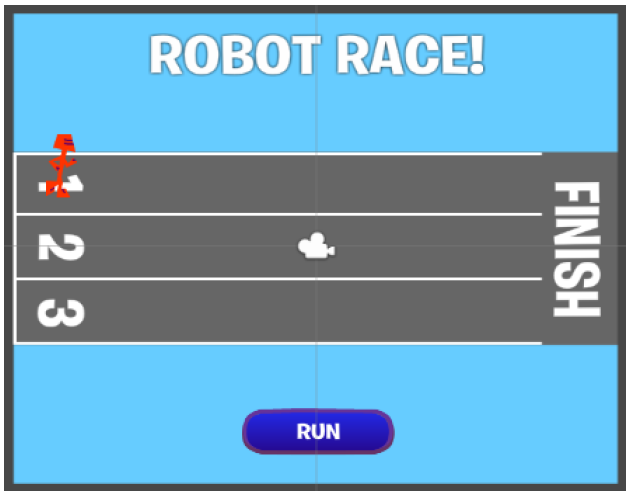
Now you can close this window and return to the scene.

If you look at the Assets window now and click the small arrow next to the robot you edited you will see the individual frames that got sliced. You will also notice at the bottom of the assets view is a slider, if you move this all the way to the left all of the assets will be displayed as a list, whereas if you move it to the right you can see thumbnail previews of the assets:





Multi select each of the three frames of the robot (Click Robot_Red_0 then hold shift and click Robot_Red_2) and drag them into the Scene window. A dialog will appear asking you to name your new animation, so let's call it PlayerAnimation. You will want to position the animation somewhere near the start line as shown below:

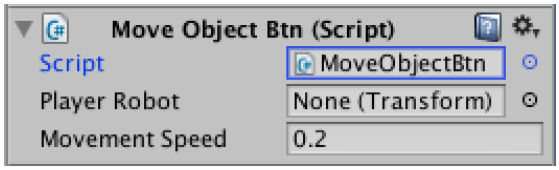


Now if you click the ‘Play‘ button at the top of the screen you will see that your player is animating walking!

4. Getting the player moving.

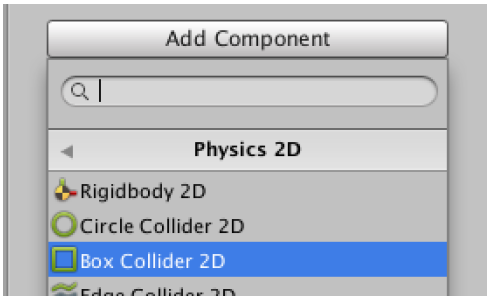
Ok so we’ve got some images on screen and we’ve got an animation working! Now we need to get some input working and to get the player moving across the screen. In the ‘Assets> Scripts‘ folder is a script called ‘MoveObjectBtn’. Drag this script onto the ‘Run_Button’ game object in the hierarchy. Now double click the script to open up MonoDevelop so we can take a look at it.

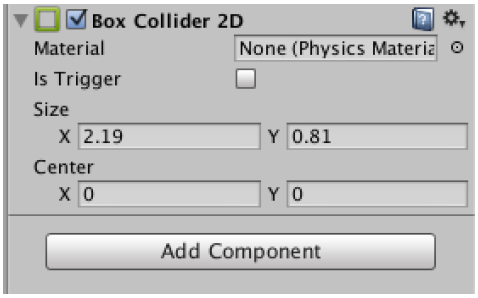
In the script we have two ‘public’ variables - playerRobot and movementSpeed. By using the word public we are making those variables visible within Unitys Inspector. Click on the Run_Button game object and look at the Inspector and you will see them there, together with default values.



The values we are using are ‘Transform’ and ‘Float’. A Transform is a group of information about an objects position, rotation and scale and a Float is a number that doesn’t have to be whole.

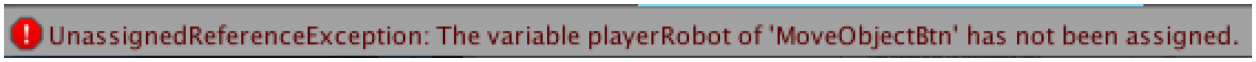
Our script has one function called ‘OnMouseDown’ this is a special Unity function that will only be called if the object that the script is running on has a ‘Collider’. So in Unity, with the Run_Button selected click ‘Add Component’ in the inspector then select ‘Physics 2D > Box Collider 2D’. Now whenever the Run button is clicked the OnMouseDown function will get called!



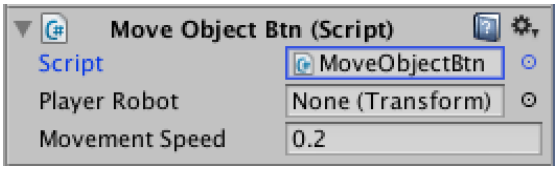


The OnMouseDown function has one line of code which simply calls a function called ‘Translate’ on the Transform variable. Translate moves an object the specified amounts in the X, Y and Z axis. For this game we just want to move the player across the X axis so we use movementSpeed as the first value and then 0 for the other two.

If you click play now and click the ‘Run’ button you will notice you get an error in the console below the scene:



This is a common mistake in Unity and simply means that we forgot to link the Player object to the script on the run button. To do this first select the Run_Button so that you can see the following in the Inspector:



Then Click and Don’t release the Player object and drag it onto the Player Robot variable to set it. Now try playing the game again and you will see that the player now moves every time you click the button!

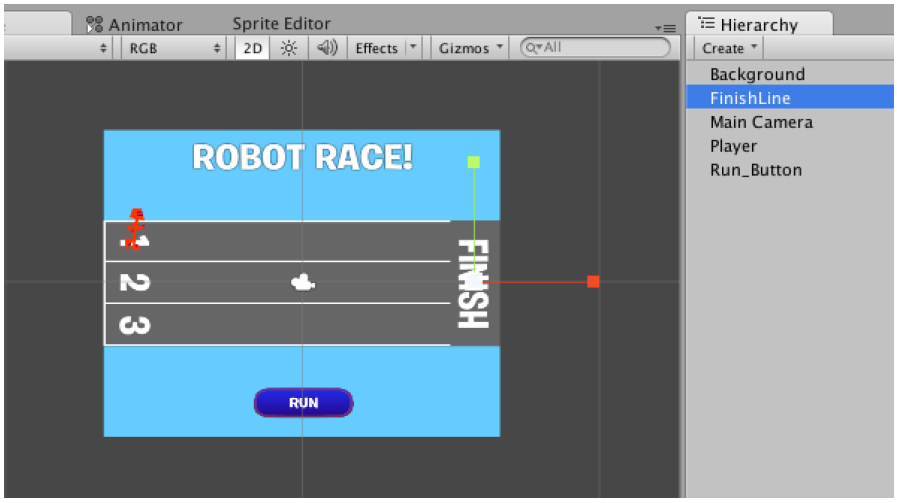
5. Create a win scenario.

Ok we’ve got animation and movement but it’s not a real game unless we can win! So firstly let’s make a winning screen. To do this we are going to utilise scenes in Unity. Firstly save the scene you are working in (Either cmd+S or File> Save Scene). Save the scene as ‘GameScene’.

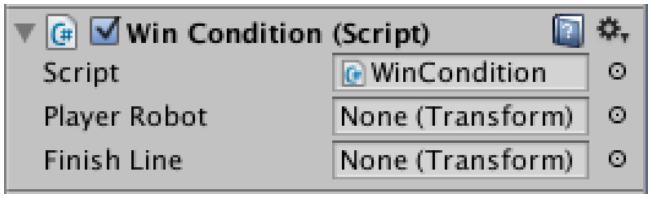
We are going to create a new scene so select ‘File > New Scene’ in the menu bar to create a new scene similar to how the project looked when we started. Again save this scene but this time as ‘WinScene’.

Now to set up the scene, look for the image ‘Win_Screen’ under ‘Assets>Images’ and drag it into the scene. Make sure to position and scale it so that it fits the screen. When you are happy save it again and double click the ‘GameScene’ scene object in the assets folder to go back to the original scene.

Now we have our winning scene we need to set up the conditions to show it! What we really want is to show it when the player crosses the finish line! To do this we are going to create a marker of where that finish line is. In the menu bar select ‘GameObject > Create Empty’ you will see that the object is added to your scene and although it doesn’t have anything inside it, you can still position it using the transform or inspector. Move the object so that it is where the finish line is. You will also want to rename this in the Hierarchy to ‘FinishLine’ so that you know what it is.



Ok now look in the ‘Assets > Scripts‘ folder for a script called ‘WinCondition’ drag this script onto any of the objects in your hierarchy, it doesn’t matter which but I would suggest the Run_Button so that the scripts are grouped together.

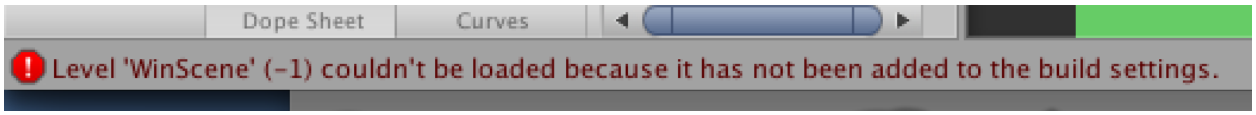


You will notice that this script has two variables that you need to set. So like we did before click

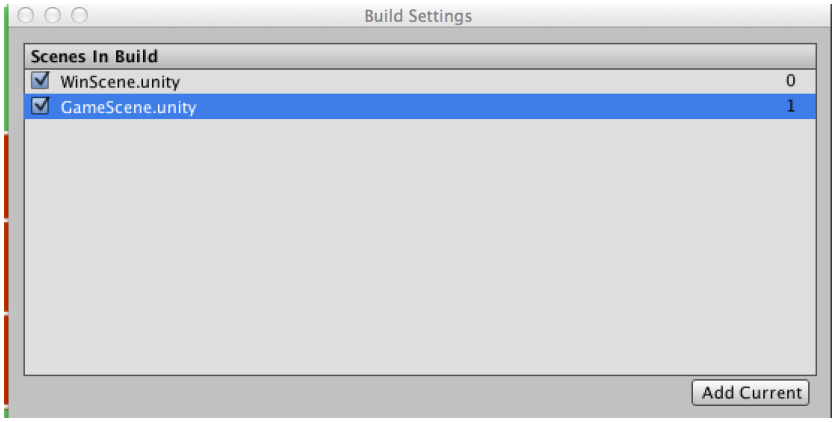
and drag the Player and Finish Line onto the corresponding variables to set them.

Double click the ‘WinCondition’ script so that we can take a look at how it works. The variables here are very similar to before but this time we have a new function ‘Update’ every unity script has an Update function by default and it gets called every frame so it is perfect for checking to see if we have won. The function simply compares the X position of the Player against the X position of the Finish Line and if it is greater it calls a special Unity function : Application.LoadLevel() which lets us load a different scene of the game.

If you play the game now and move the player past the finish line you will notice you get another error in the console:



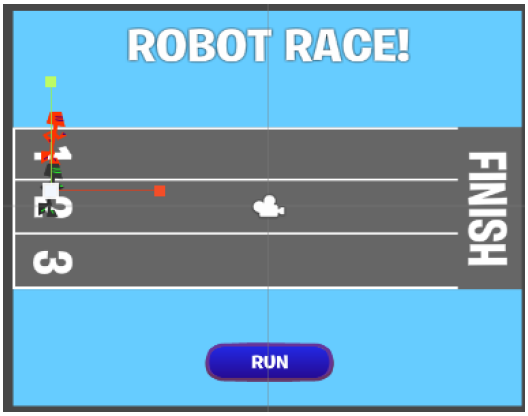
This is another common mistake when using multiple scenes in Unity. Unity requires that all Scenes used within a game be added to a list in Build Settings. You can open these settings via ‘File > Build Settings’ or cmd + B. You can then drag both of the scenes into this view to add them.

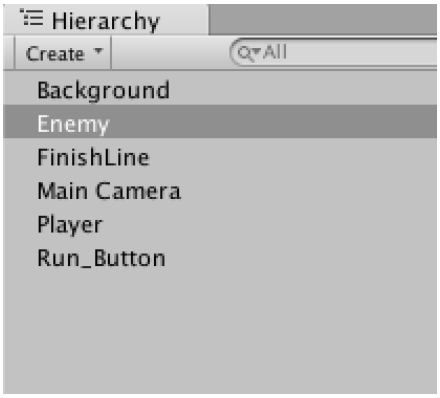


Once they are added close the window and try the game again. This time when you cross the winning line the Win screen should appear!

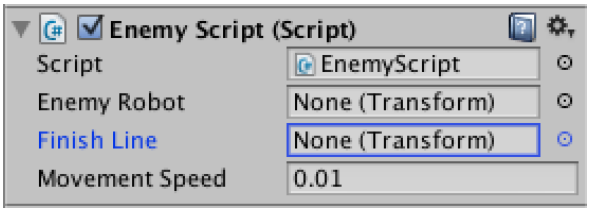
6. Creating an enemy.

We can win the game but we need to add a way to lose it too! In the Assets >Images folder select another Robot to use as the enemy. Edit the image to create multiple sprites like we did for the player at the start of the tutorial and drag the frames into the scene to create a new animation. I suggest calling this “EnemyAnimation” and position it on the start line above/below the player. You may also want to rename it to Enemy in the hierarchy. Your setup should look like this now:





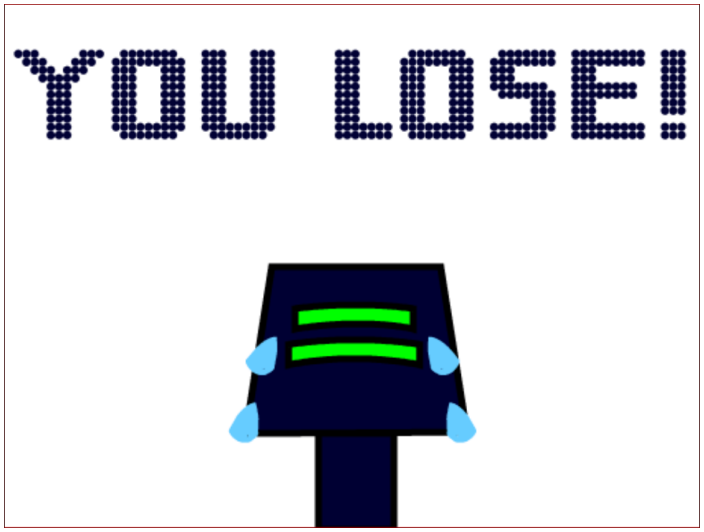
Under the ‘Assets > Scripts’ folder is one final script called ‘EnemyScript’ drag this onto the Enemy object in your hierarchy.



You will notice that the variables here are a combination of the ones used in both previous scripts. So make sure to drag the Enemy and Finish Line objects onto the corresponding variables and set the Movement speed to something higher than 0. Unlike the player however, the enemy will move every frame so the speed will need to be much slower, I suggest 0.002.

If you open the script you will see that the code here is simply a combination of the previous two scripts we used. The difference is we are moving and checking the position of the Enemy object this time and if the Enemy crosses the finish line then Unity will attempt to load a scene called “LoseScene”. Obviously we haven’t created a Lose scene yet, so let’s do that now. It’s just the same process as earlier, save and create a New Scene called “Lose Scene” (You can use the “Game_Over_Screen” asset for this scene) and add the scene to the Build Settings.

Now if you play the game and let the enemy cross the finish line first you will see the losing screen!



7. Adding some music.

Finally to add a little atmosphere to our game we can add some sounds! Sounds are very easy to add You can just drag them onto any object in the scene and they will auto play when the game starts. Try adding ‘Assets > Sounds > Game_Loop’ to the Main Camera in the GameScene and hitting play! Behold sound is now playing when you start the game!

Challenges:

As an extra challenge to help you learn the parts covered in this tutorial, why not see if you can do the following:

1. Add the win and lose sounds to the win and lose scenes.

2. Add a retry button to the win and lose scenes that loads the ‘GameScene’.

3. Add a 2nd enemy that runs at a different speed.

Acknowledgments:

Sound files obtained from http://www.nosoapradio.us and are avaialble under the Creative Commons Attribution Licence.

Character sprites were obtained from http://opengameart.org/content/neon-armor-side-fight-sprite-standrunpunch and are also available under the Creative Commons Attribution Licence.