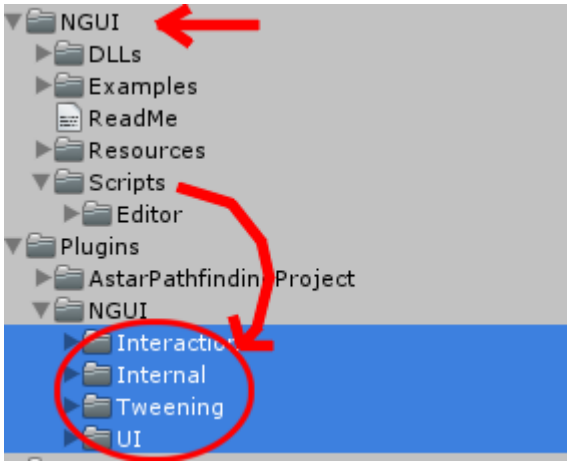
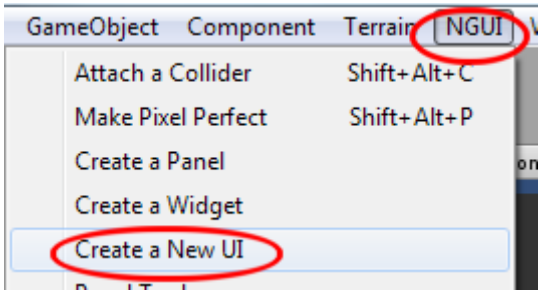
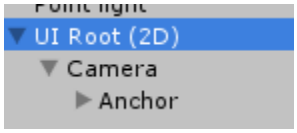
Session 7 – NGUI, AStar & Combat

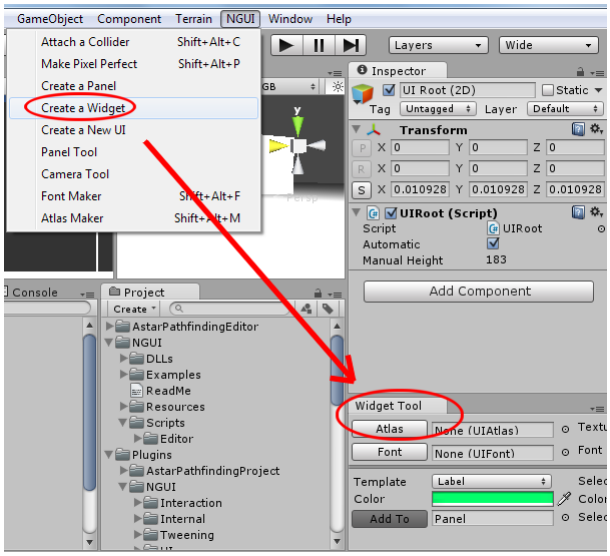
Adding NGUI

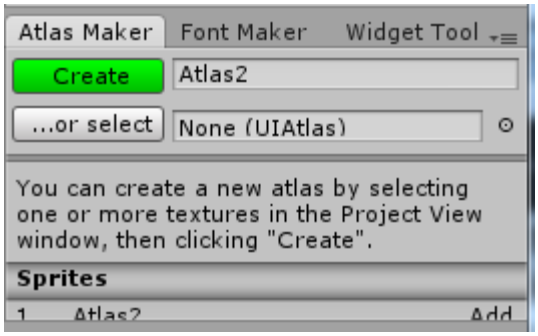
1. Install NGUI from package provided on the G drive called ngui_free.unitypackage. it will self install as long as you have Unity open so double click on your copy – do not run from network.
2. To make it work with Javascript as well once installed make a new folder under Standard Assets called Plugins. Then add a folder under that called NGUI. Then from the original NGUI->Scripts folder drag these four folders over.
3. 

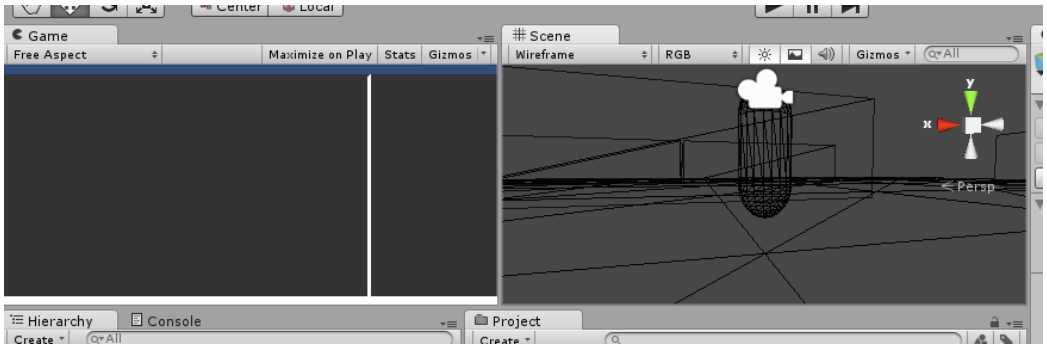
4. Make a new scene.
5. From the NGUI menu select Create New UI
6. 

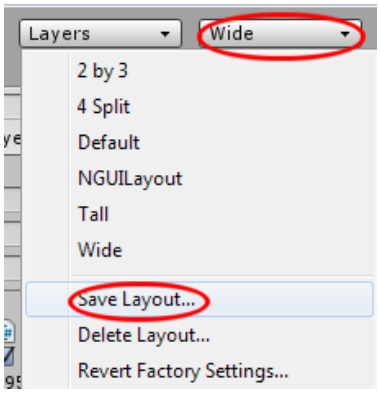
7. It will appear in your scene Hierarchy like this
8. 

9. Now from the NGUI menu open Create A Widget. Then dock the window below the Inspector as shown.

10. 

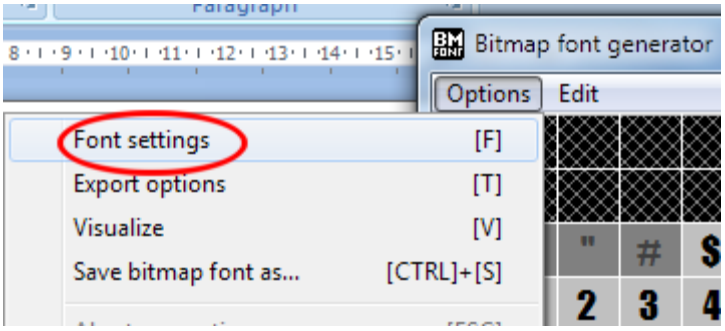
11. Repeat for Font Maker and Atlas Maker by putting them along side each other so it looks like this
12. 

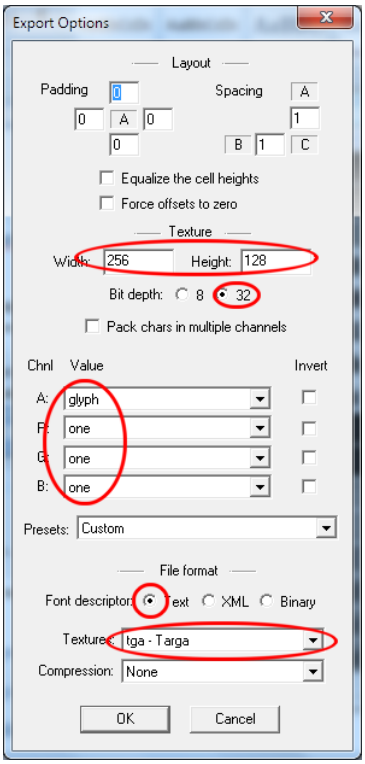
13. Snap you scene away from your Game window and park side by side like this
14. 

15. Now save this layout as NGUILayout by clicking on the Layout button and selecting Save New Layout. Give it the name NGUILayout – now you can toggle between the preset layouts and yours.
16. 

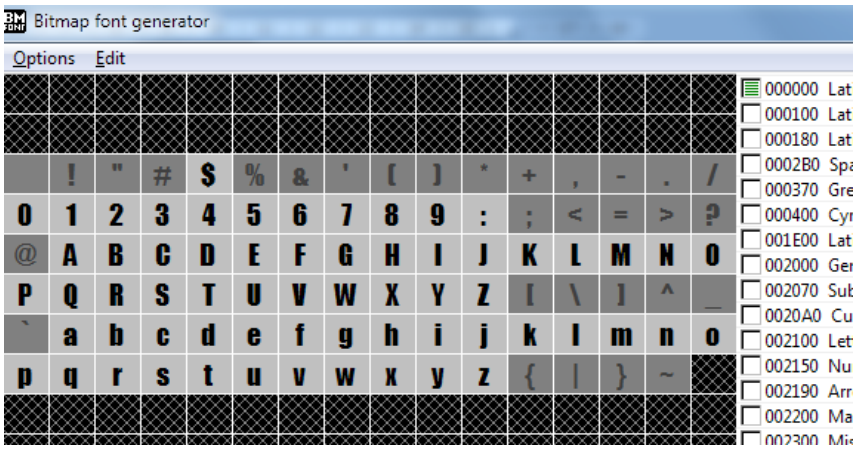
Export your own font.

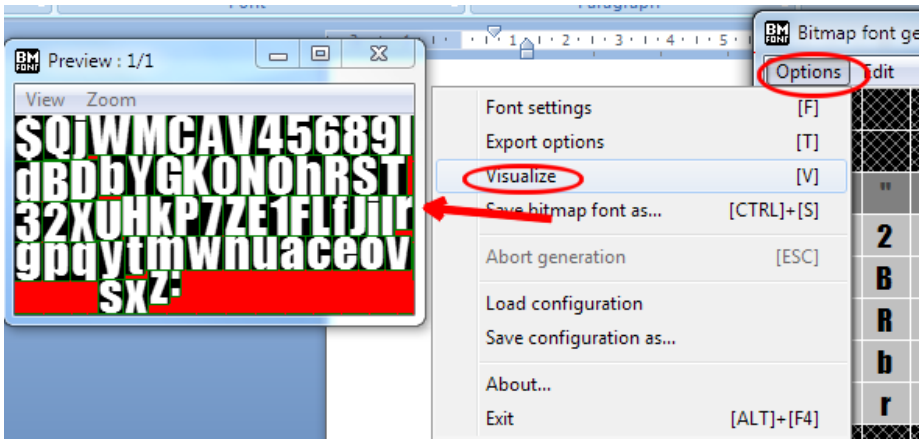
You'll need your own fonts for each UI at each particular size for your game. Here's how to make one.

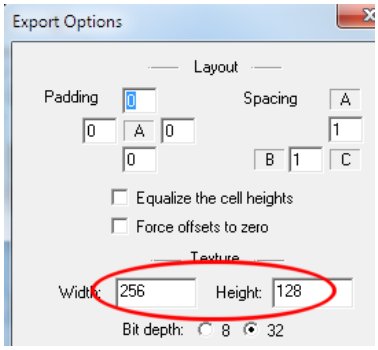
1. Install Bitmap Font generator from file called install_bmfont_1.13.exe
2. Run Bitmap Font generator –select font from first menu
3. 

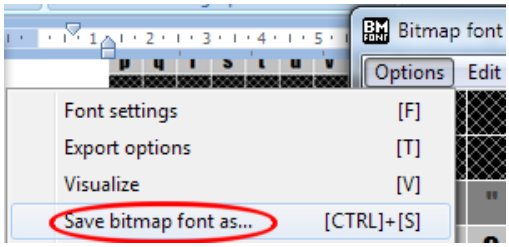
4. Change as following – make sure it's 32bit otherwise come settings won't come up.
5. 

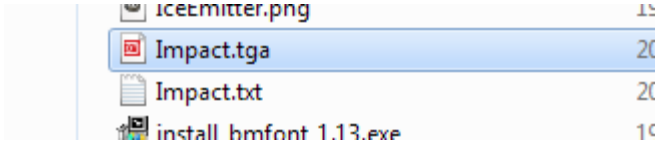
6. Select none of the characters – toggle CTRL+A
7. Drag over characters you want – typically alphanumerics, colon, dollars and whatever you think you'll need.

8. 

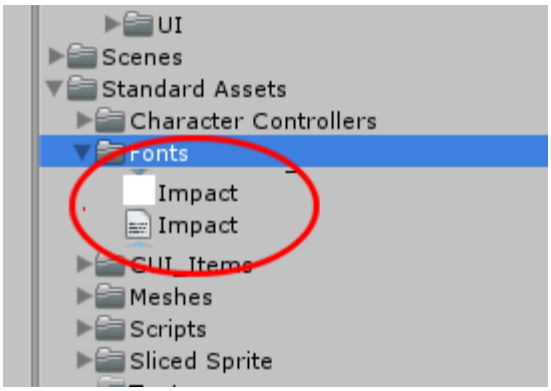
9. To make sure it packs the fonts into the screen space on export hit Visualise.
10. 

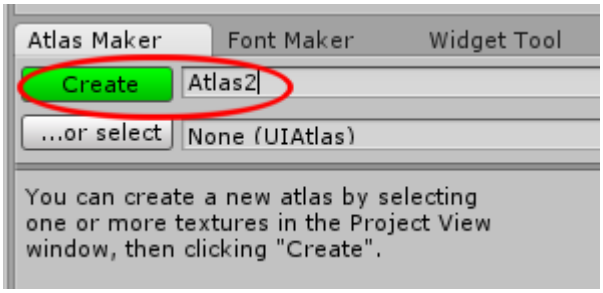
11. If it doesn't pack nicely change the texture size under Options->Export Options
12. 

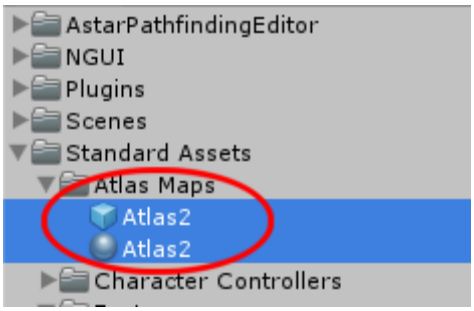
13. With all that in place now click on Options->Save Bitmap font
14. 

15. Save in at away from the Unity folders as you will need to rename it in a minute.
16. Rename the *.fnt file to *.txt
17. Take off the trailing _0 from the tga. It should look like this now.
18. 

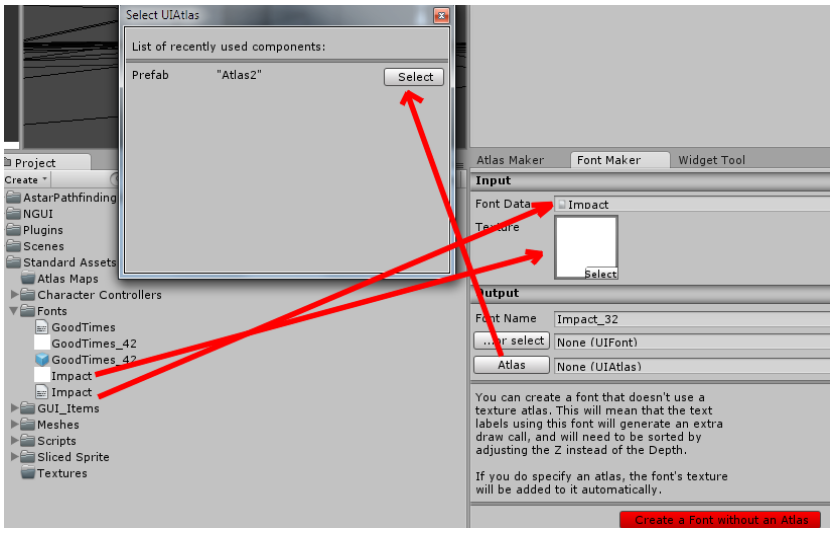
Import font into Unity

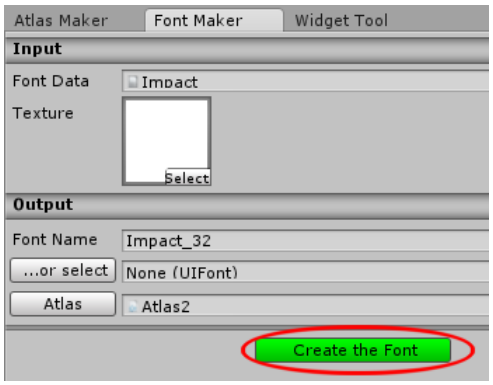
1. Create a new folder under Standard Assets called Fonts
2. Drag the *.txt and *.tga files into that folder. It should look like this.
3. 

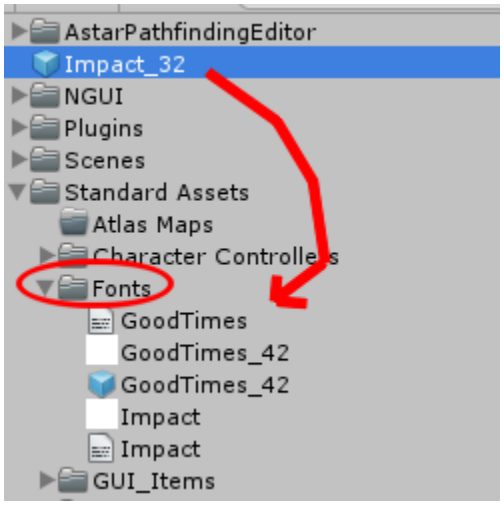
4. It's almost ready to use, we just need to make an atlas material of it.
5. Click on the Atlas Maker tab, name your Atlas and click on Create
6. 

7. Under Standard assets create a new folder called Atlas Maps and drag the two Atlas files over to it.
8. 

9. Click on Font Maker. Drag the white font file across to the texture and the text file to the edit field as shown. If no Atlas is selected, click on the Atlas button to select one.

10. 

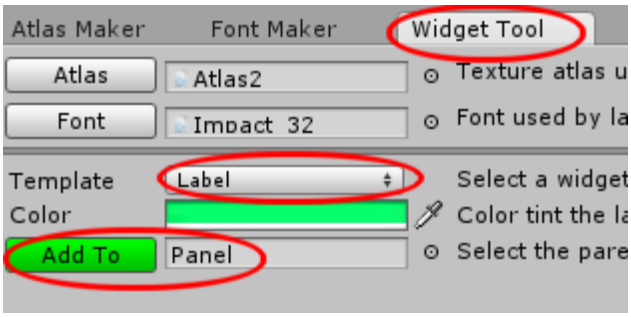
11. Click on Create the Font now.
12. 

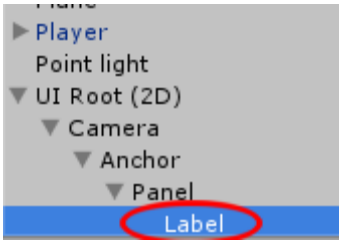
13. Drag the created material into the Font folder as shown. Now you are ready to make a Label.
14. 

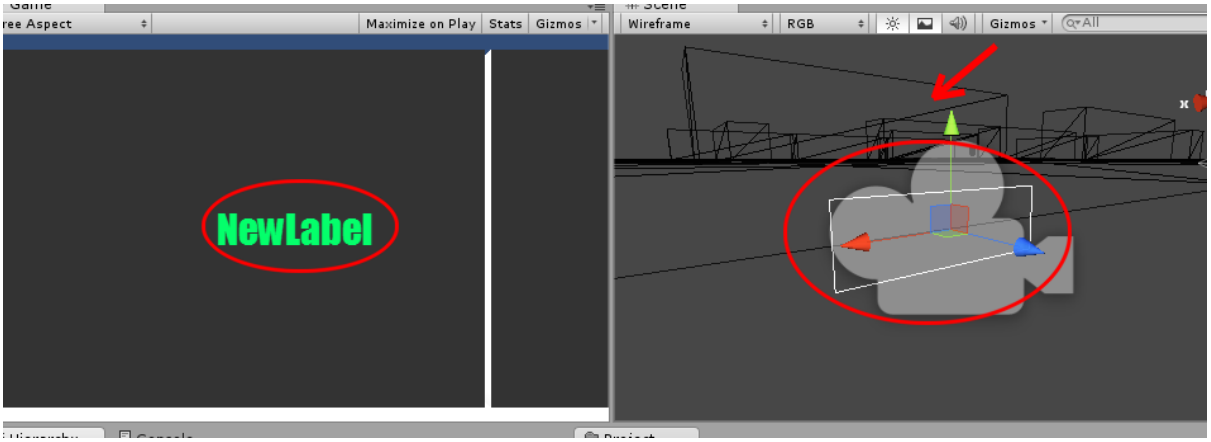
Creating a score label.

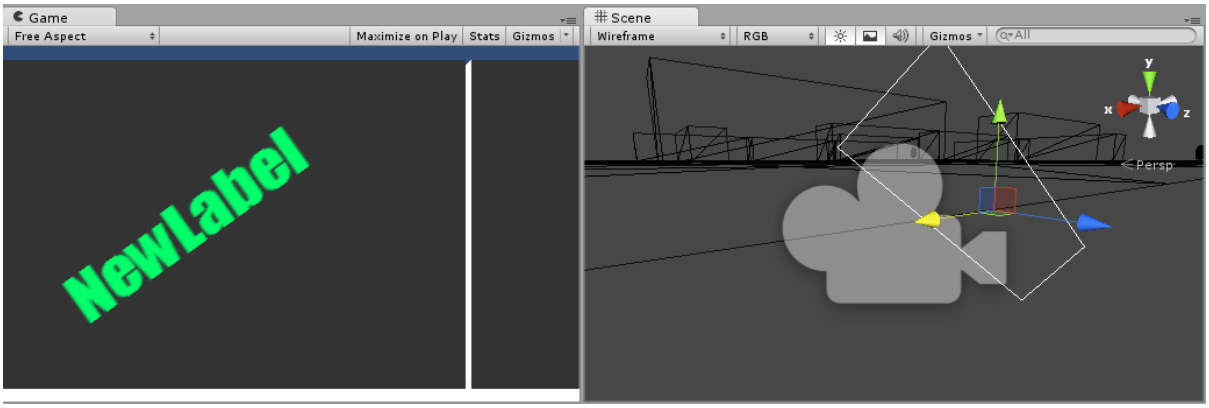
Now you've set all this up it's actually easy to make labels on the screen.

1. Click on the Widget pane.
2. Select Label as the template.
3. Select Add To Panel

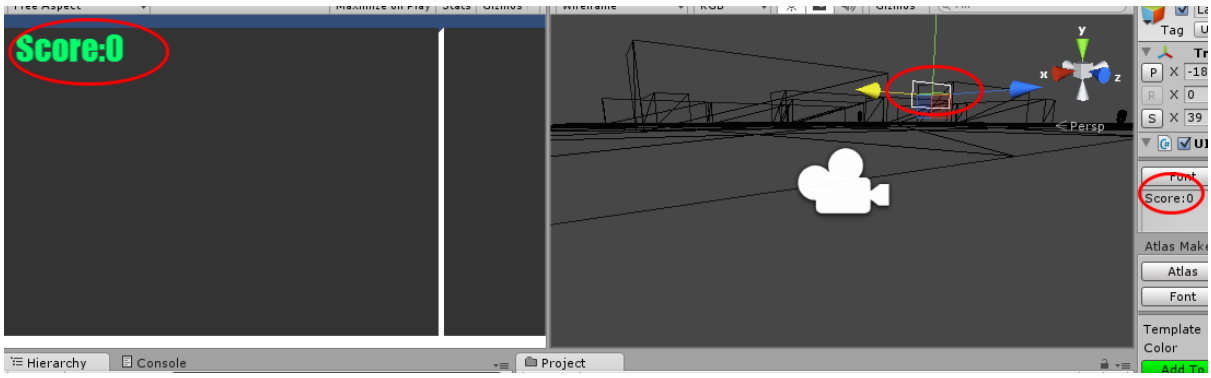
4. 

5. It will appear in the scene under its own special camera. This is why you need to see Scene and Game at the same time.
6. You will now see it in the scene hierarchy.
7. 

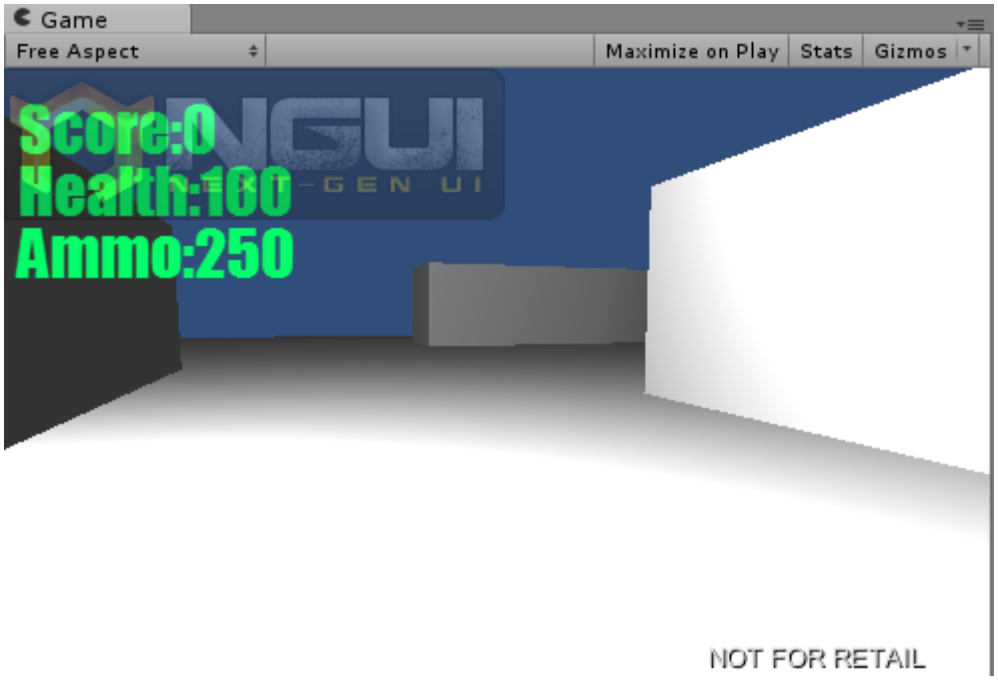
8. You will notice it is a different camera to the scene and has no relation to the scene other than being stuck to the Main Camera.
9. 

10. Using the scale, move and rotate you transform the label in relation to front screen the player will see in game
11. 

12. Change the label in the Inspector to Score:0 and place upper left.

13. 

14. Add health and ammo using the same technique.

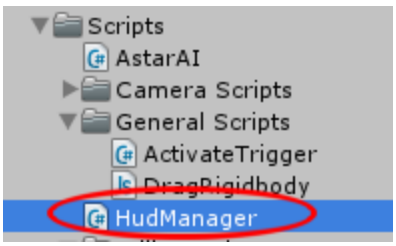
15. 

16. In the Inspector name them HealthLabel, ScoreLabel and AmmoLabel.

Now in script if we want to change them we just call something like...

```
UILabel c = GameObject.Find("ScoreLabel").GetComponent<UILabel>();

c.text = "Score: " + score;
```

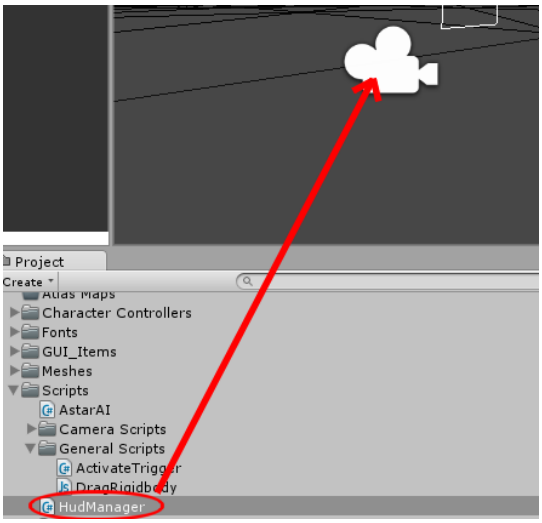
1. To do this make a new C# script called HudManager.



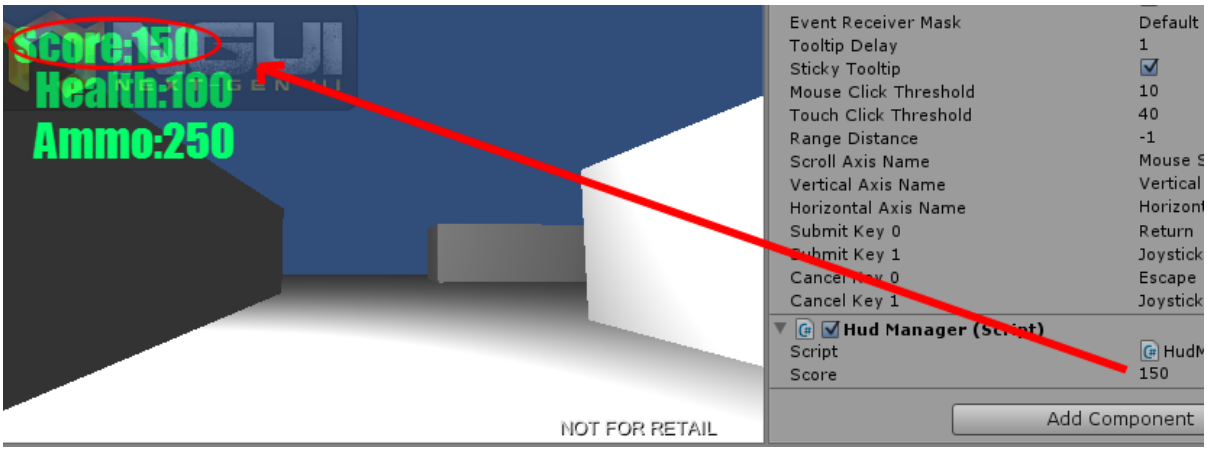
2. Double click to open it in MonoDevelop. Add the following code to it.

```
using UnityEngine;
using System.Collections;

public class HudManager : MonoBehaviour {

    public int score = 0;

    // Use this for initialization
    void Start () {

        UILabel c = GameObject.Find("ScoreLabel").GetComponent<UILabel>();
        c.text = "Score: "+score;

    }

    // Update is called once per frame
    void Update () {

    }
}
```

3. Add the script to the GUI camera.



4. To test the score change the exposed (public)score var to 150.



Making AStar track player

Let's change the AstarAI to expose a transform for any object, and modify the code so it looks for that object if assigned every 3 seconds.

1. Open AstarAI in MonoDevelop.
2. At the top change as provided.

```
using Pathfinding;
public class AstarAI : MonoBehaviour {
    //The point to move to
    public Vector3 targetPosition;

    public Transform targetGameObject;

    public Vector3 oldTargetPosition;

    private Seeker seeker;
    private CharacterController controller;

```

3.
4. In the OnStart function add this code.

```
    public void Start () {
        seeker = GetComponent<Seeker>();
        controller = GetComponent<CharacterController>();

        // check to see if it has a game object target
        if (targetGameObject)
        {
            // create a new path to target object
            seeker.StartPath (transform.position,targetGameObject.position, OnPathComplete);

            // callback to check position of target object over time
            Invoke ("checkForTargetObject",3);

            // remember current position heading to
            oldTargetPosition = targetGameObject.position;
        }
        else
        {
            //Start a new path to the targetPosition, return the result to the OnPathComplete function
            seeker.StartPath (transform.position,targetPosition, OnPathComplete);

        }
    }
```

5.

6. Write the function to check for the object its seeking every 3 seconds.

```
            seeker.StartPath (transform.position,targetPosition, OnPathComplete);

        }
    }
    public void checkForTargetObject()
    {
        if (path == null) {
            //We have no path to move after yet
            CancelInvoke("checkForTargetObject");
            return;
        }

        if (currentWaypoint >= path.vectorPath.Count) {
            // run out of points
            CancelInvoke("checkForTargetObject");
            Debug.Log ("End Of Path Reached");
            buildNewPath();
        }
        // if target objects position has changed more than the size of distance to next way point
        // then start a new path
        if (Vector3.Distance (targetGameObject.position,oldTargetPosition) > nextWaypointDistance)
        {
            buildNewPath();
        }

        // call again
        Invoke("checkForTargetObject",3);

    }
```

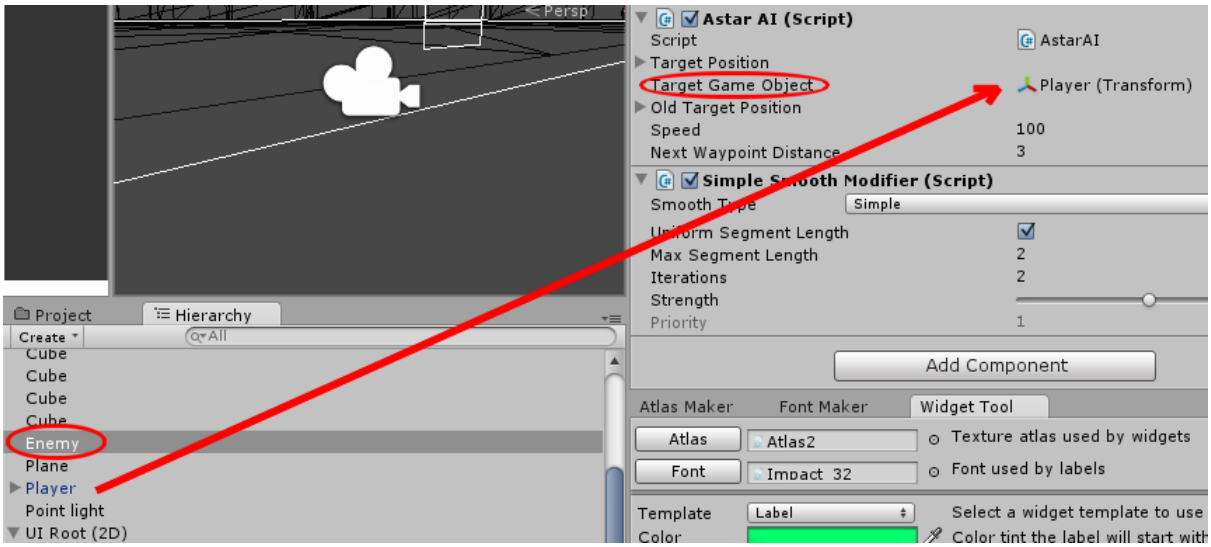
7.

8. Add the code to rebuild the path after that.

```
        // call again
        Invoke("checkForTargetObject",3);

    }
    public void buildNewPath()
    {
        Debug.Log("Making new path");
        seeker.ReleaseClaimedPath();

        // start a new path
        seeker.StartPath (transform.position,targetGameObject.position, OnPathComplete);
        oldTargetPosition = targetGameObject.position;

    }
    public void OnPathComplete (Path p) {
        Debug.Log ("Yey, we got a path back. Did it have an error? "+p.error);
```

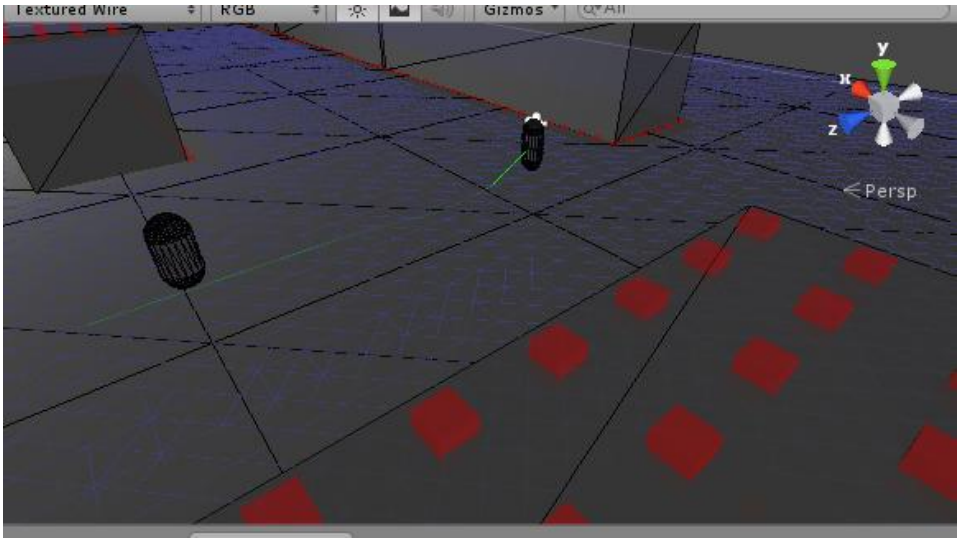
9.

10. F7 to compile and test for errors/typos.
11. Go back to Unity and from the Heirarchy in the AI scene, select the Enemy and drop the Player into the newly exposed Target Game Object. Actually oldTargetPosition should really be private – change it if you want.



12.

13. Run the game now and move around with the Scene tab open as well. Every 3 seconds the Enemy will update its path to find you.

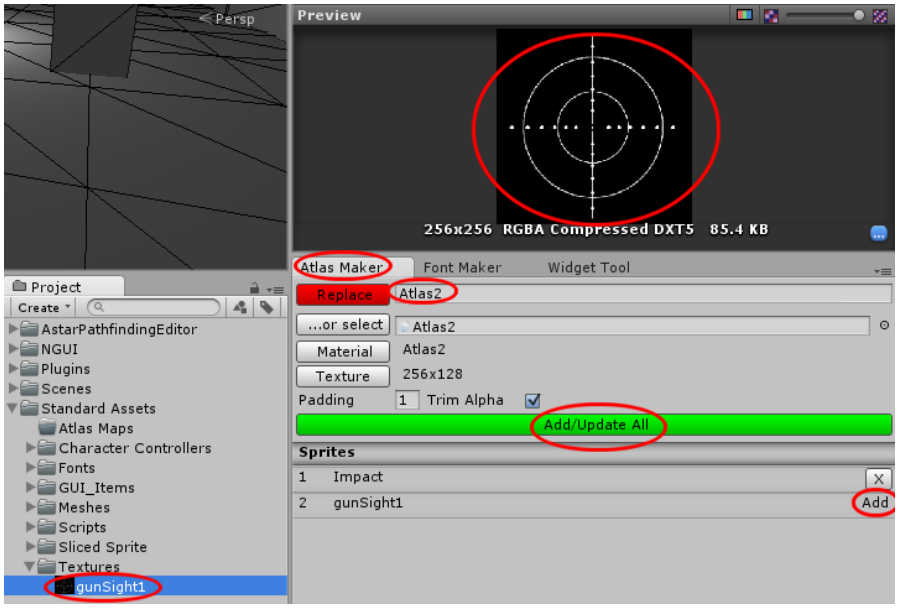


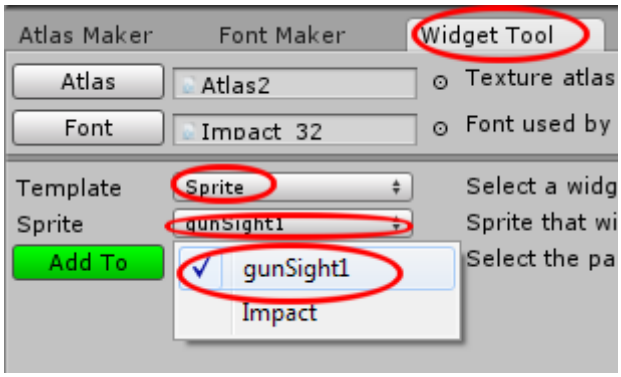
14.

15. Challenge – with your awesome programming skills expose a variable of type float called AIThinkTimer and change the speed of the callback to checkForTargetObject to use this, then in the Inspector change it to 5 seconds, then 2 seconds.

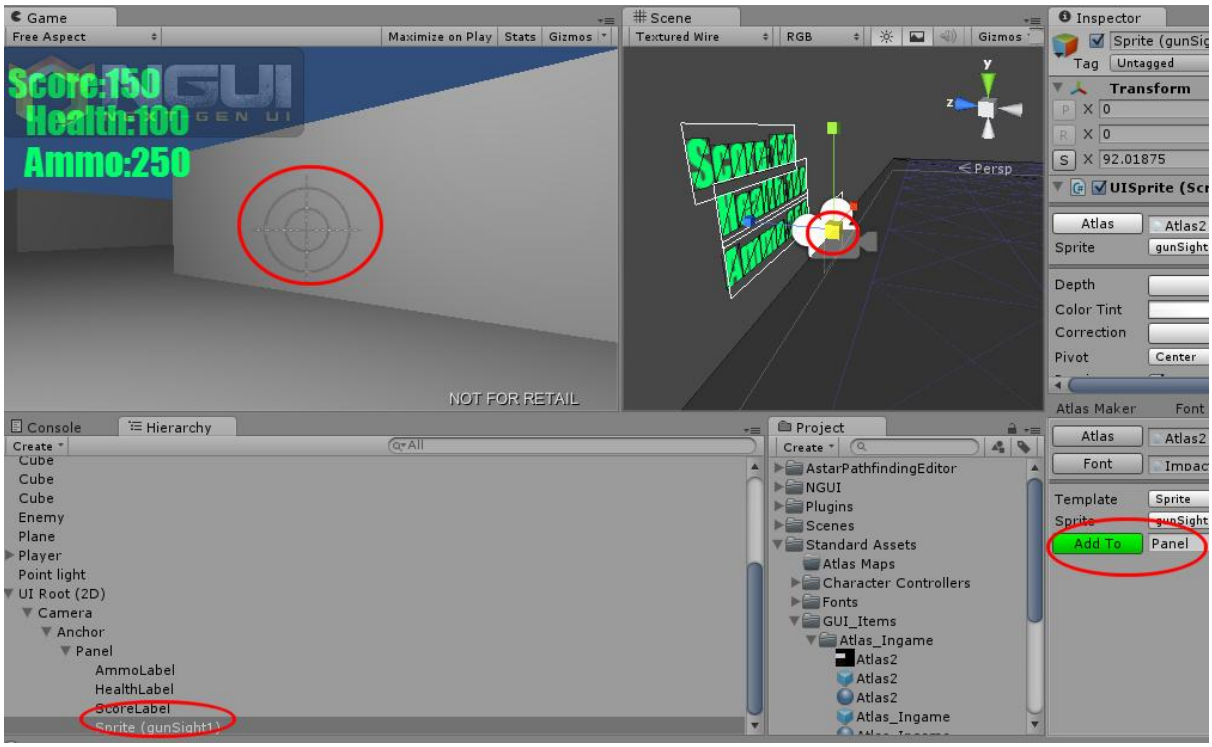
Adding a gun and firing at enemy – reducing ammo

Add the gunsight first.

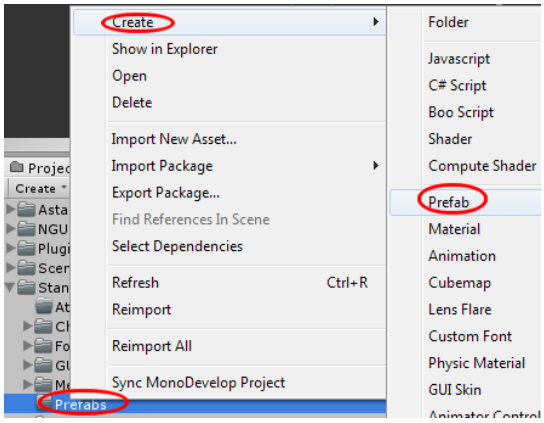
1. This is an NGUI component – load the gunsight.png into textures
2. It needs to be added to the texture atlas so with it selected, click on Atlas Maker, and scroll down to see it's a candidate to be added.
3. Click on the green add button.
4. 

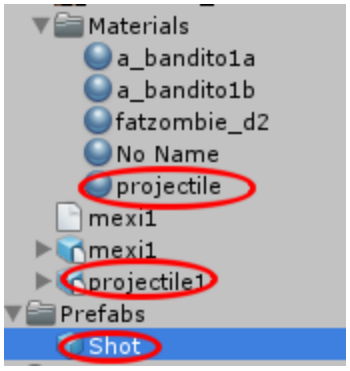
5. In the Widget tool, select Sprite as the template and then hit the dropdown to select the newly added gunsight sprite.
6. 

7. Click on Add to Panel and move it to the centre of the screen. Resize to suit.

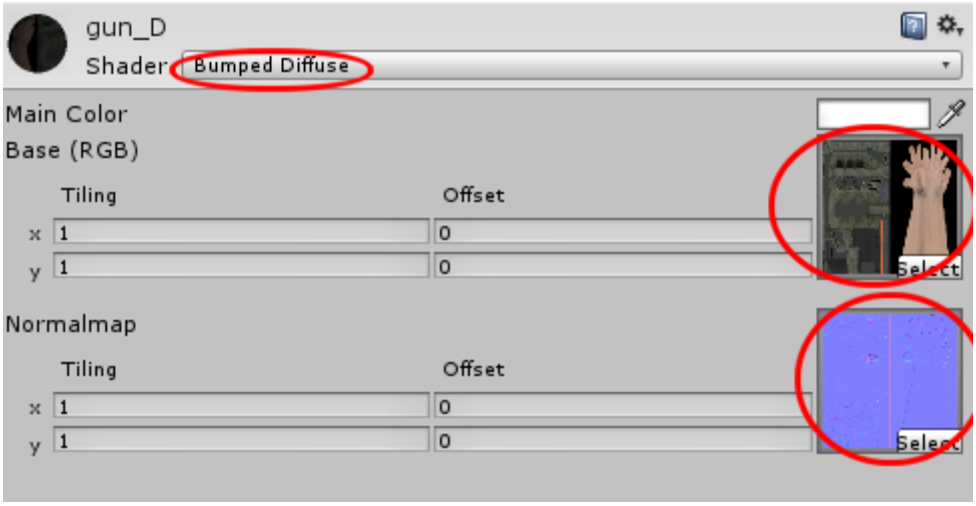
8. 

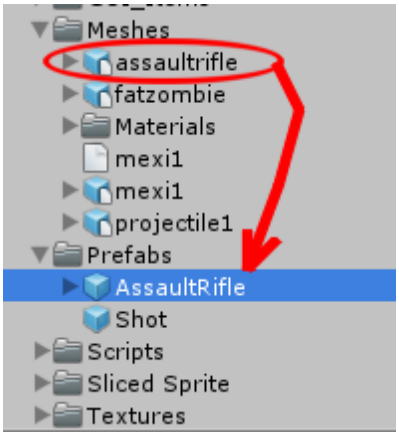
Next we need to add a projectile. By making it a prefab when can spawn it from the player.

1. Import projectile into meshes
2. Make a folder called Prefab and create an empty one called Shot.
3. 

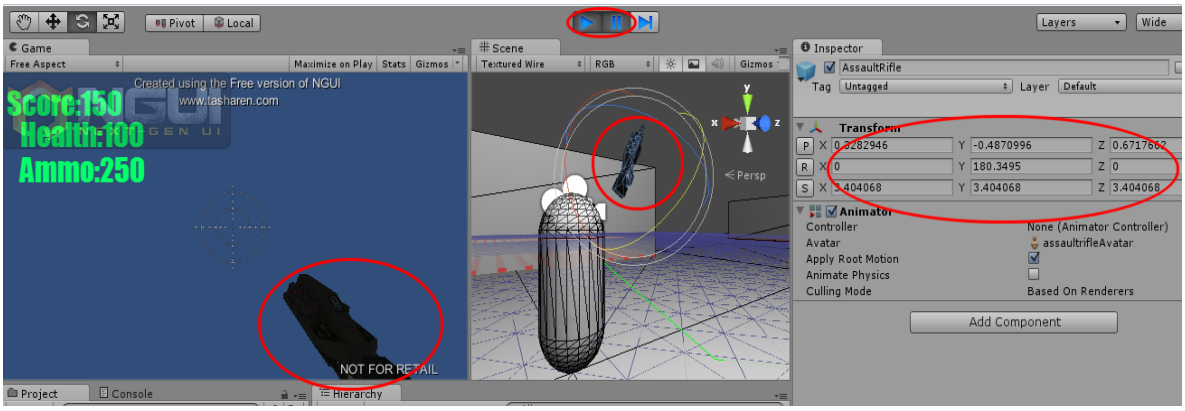
4. Drag the projectile prefab into it. Put materials into the Meshes folder under Materials to clean it up. Put the projectile into Meshes.
5. 

Now we add a gun. We'll attach it the player, and add an empty gameobject to the nozzle to fire the project from.

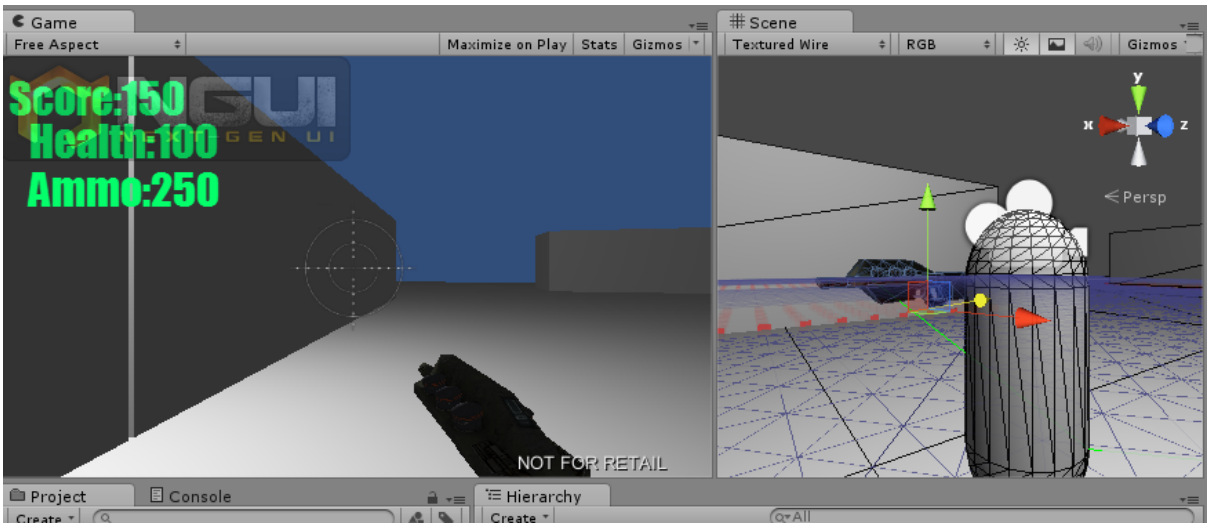
1. Import assaultrifle.fbx into Meshes.
2. Right click on it to select dependency and update the textures.
3. Make sure the material is Bumped Diffuse and has two textures in place.
4. 

5. Make a Prefab called AssaultRifle and add the assaultrifle model to it.
6. 

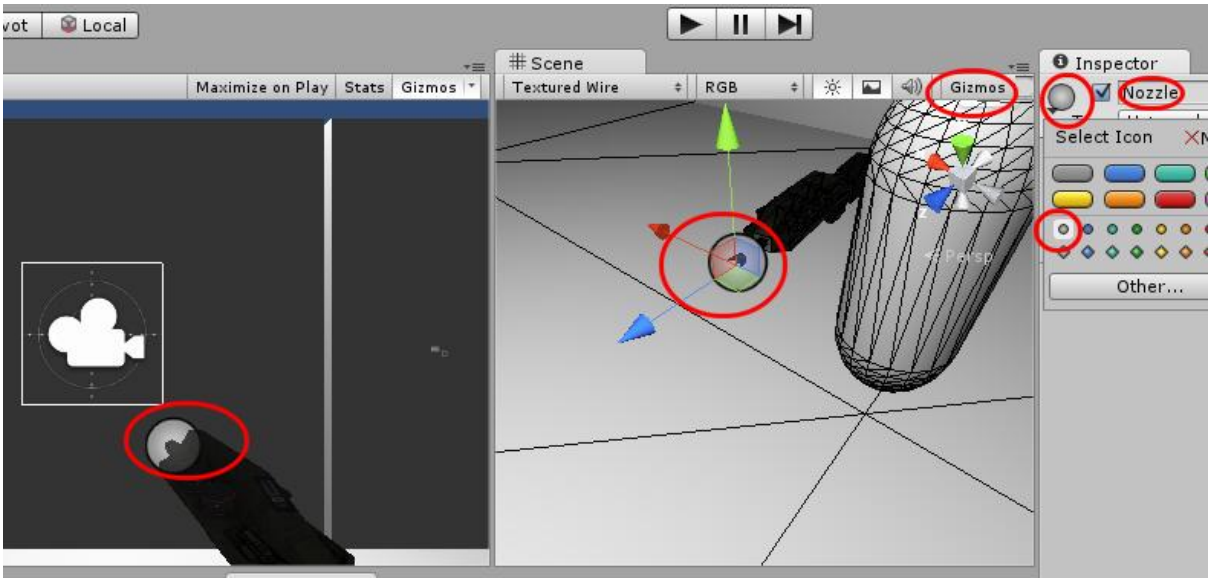
7. Find the Main Camera under the Player in the scene and drop the AssaultRifle prefab on it. Reset it's coords to 0,0,0 on X,Y,Z and 0,0,0 on A,B,C (rotation). Scale to about 4.
8. Click on the rifle, change to local and put in the shot as you like.



9. Running it, it should look like a Dalek :D

10. 

11. Create an Empty Game object – call it Nozzle, attach to Rifle and displace just like you did for the gun, but put it on the nozzle. Use a small 3D gizmo to help you place it.

12. 

We need to add some firing code now. This will be done via the player controller.

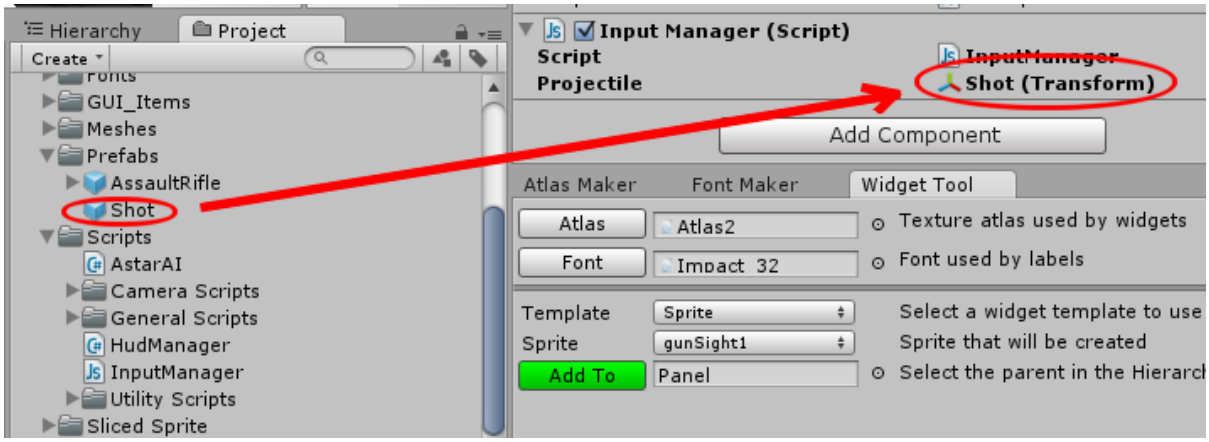
1.Create a Javascript script called InputManager

2. Open in MonoDevelop and add this code.

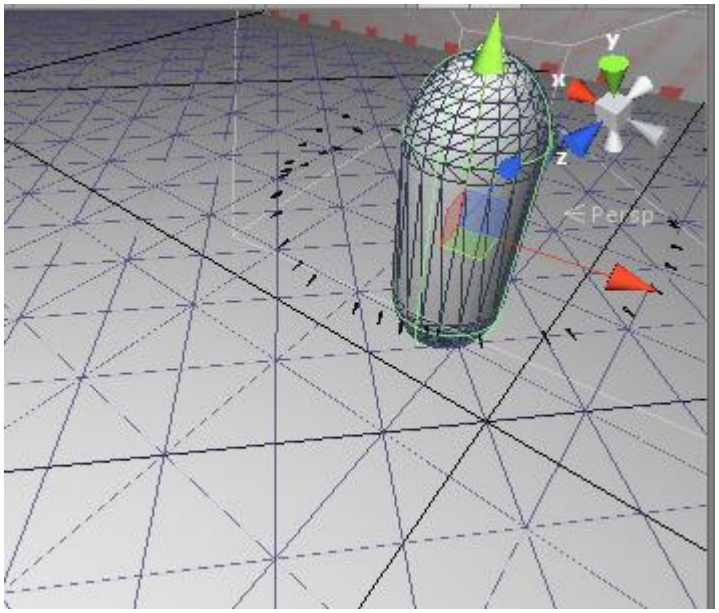
```
var projectile : Transform;

function Start () {

}

function Update () {

    if (Input.GetButtonDown("Fire1"))
    {
        // Instantiate the projectile at the position and rotation of this transform
        var gunNozzle : GameObject;
        gunNozzle = GameObject.Find("Nozzle");

        var clone : Transform;
        clone = Instantiate(projectile, gunNozzle.transform.position, gunNozzle.transform.rotation);

    }
}
```

3. Drop script onto Player.

4. Drag the Shot prefab into the Projectile transform slot.



5. Run the game and spin around firing. You will see projectiles spawn wherever the gun nozzle was.



Next we need to add some code to launch the project and kill it off over time.

1. Create a Javascript file called projectileController
2. Add this code.

```
var projectile : Transform;
var bulletSpeed : float;
function Start () {

}

function Update () {

    if (Input.GetButtonDown("Fire1"))
    {
        // Instantiate the projectile at the position and rotation of this transform
        var gunNozzle : GameObject;
        gunNozzle = GameObject.Find("Nozzle");

        var clone : Transform;
        clone = Instantiate(projectile, gunNozzle.transform.position, gunNozzle.transform.rotation);

        // Add force to the cloned object in the object's forward direction
        clone.gameObject.AddComponent(Rigidbody);
        clone.rigidbody.mass = 0.1;
        clone.rigidbody.useGravity = false;
        clone.rigidbody.AddForce(clone.transform.forward * bulletSpeed);

        // attach script
        clone.gameObject.AddComponent("projectileController");
    }
}
```

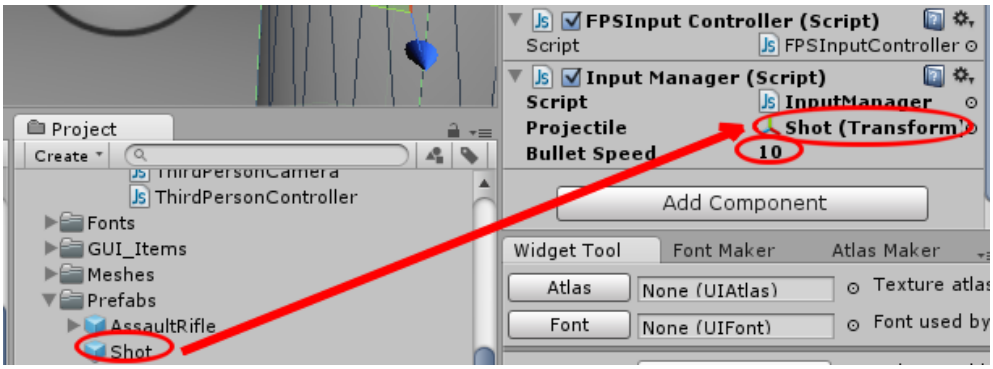
3.

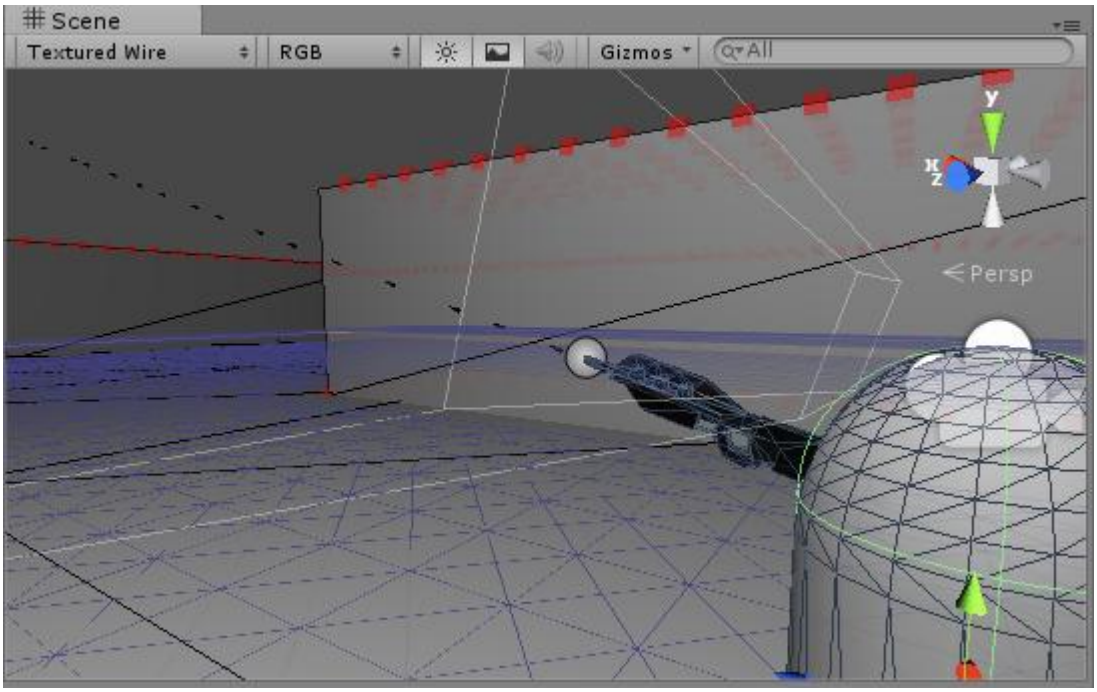
4. Add it to the Player.
5. Create a new Javascript called projectileController.
6. Open in MonoDeveloper and add this code.

```
var timer : float;

function Start () {

    // set timer to 3 seconds of life
    timer = 3;
}

function Update () {
    // tick over timer and kill after duration
    timer -= Time.deltaTime;

    if (timer < 0)
    {
        // kill off bullet
        Destroy (this.gameObject);
    }

}
```

7.

8. In Unity click on the Player and under the InputManager script add 10 to bullet speed.
9. Drag the Shot prefab to the projectile slot.

10. 

11. Now test in game. Shoot into the sky and you should see the bullets delete along their path. Know issue – player speed needs to be added to bullets.
12. 

Finally add this code to keep track of the ammo.

1. Open InputManager in MonoDeveloper.

2.

```
#pragma strict
using UnityEngine;
using System.Collections;

public class HudManager : MonoBehaviour {

    public int score = 0;
    public int ammo = 255;

    // Use this for initialization
    void Start () {

        UILabel c = GameObject.Find("ScoreLabel").GetComponent<UILabel>();
        c.text = "Score: "+score;
        c = GameObject.Find("AmmoLabel").GetComponent<UILabel>();
        c.text = "Ammo:"+ammo;
    }

    // Update is called once per frame
    void Update () {

        if (Input.GetButtonDown("Fire1") && ammo > 0)
        {
            UILabel c = GameObject.Find("AmmoLabel").GetComponent<UILabel>();
            if (ammo > 0) ammo -= 1;
            c.text = "Ammo:"+ammo;
        }
    }
}
```

3. Run now and your ammo goes down as you fire. Know issue – can keep firing when ammo is zero.

Coming up in Session 8.

Add player speed to bullets and stop firing when ammo is spent.

Pickup ammo, loot and health

Enemy gets shot – modifying enemy health

Enemy fires back – modifying player health

Enemy melee – modifying player health

Player melee – modifying enemy health

Spawning new enemy

Finding the exit

Timer / Best score

State management – title, game, game over.