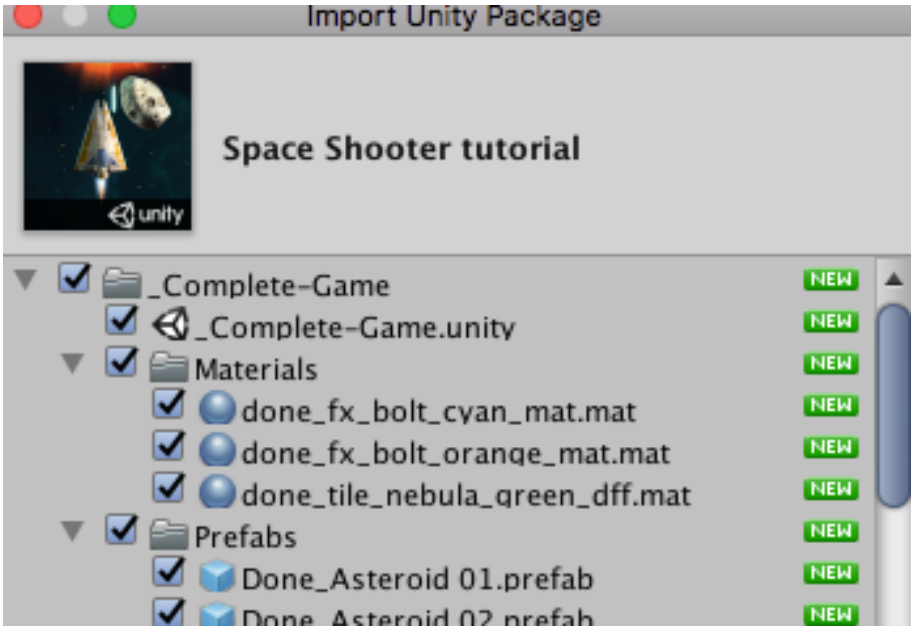
Unity3D - SpaceShooter 1.0

A Top-down arcade space shooter game built for WebGL and MacOS X

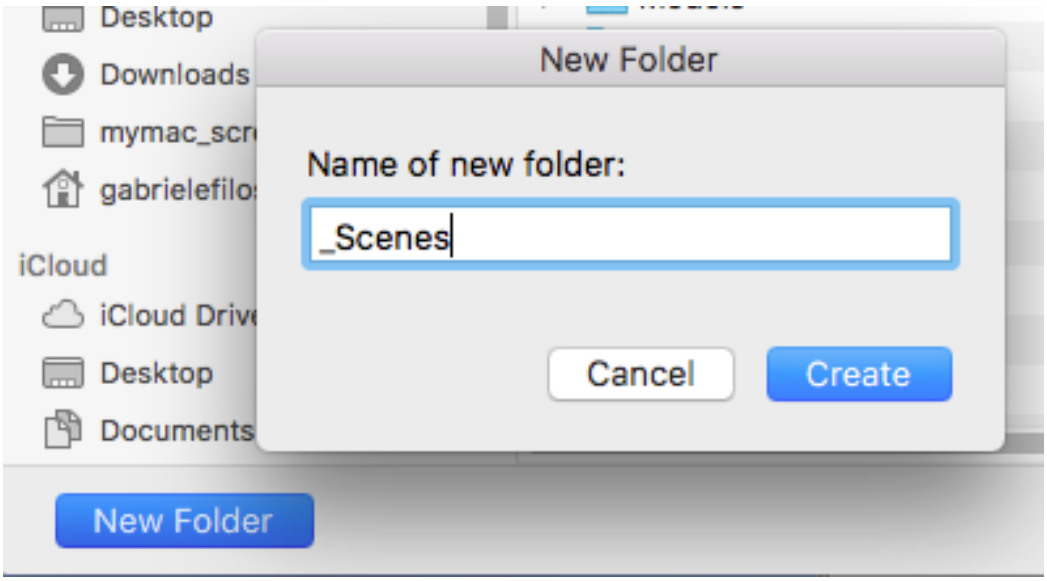
Get Started

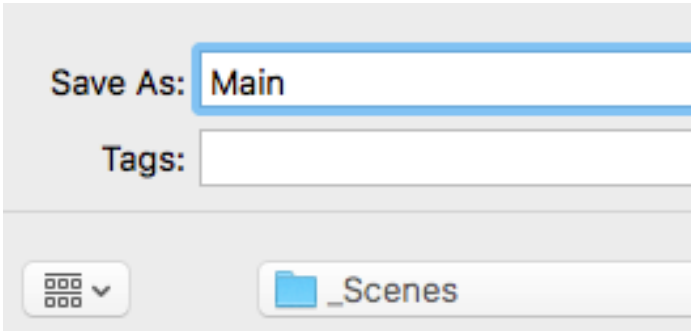
- Under Unity 5.x create a 3D project “SpaceShooter”
- In the Asset Store search, download the Assets Package (free)



Click *All* and then the *Import* buttons.

- Click *File > Save Scenes*. Call the scene “Main” and save it to a new folder called *_Scenes* in the Assets folder





- Click *File > Build Settings*, set the WebGL

You may require to download the WebGL support package

and then install it by double click the file
UnitySetup-WebGL-Support-for-Editor-5.6.1p1
After installation you need to restart Unity Editor.
WebGL is a cross-platform, royalty-free web standard for a low level 3D graphics API based on OpenGL ES.

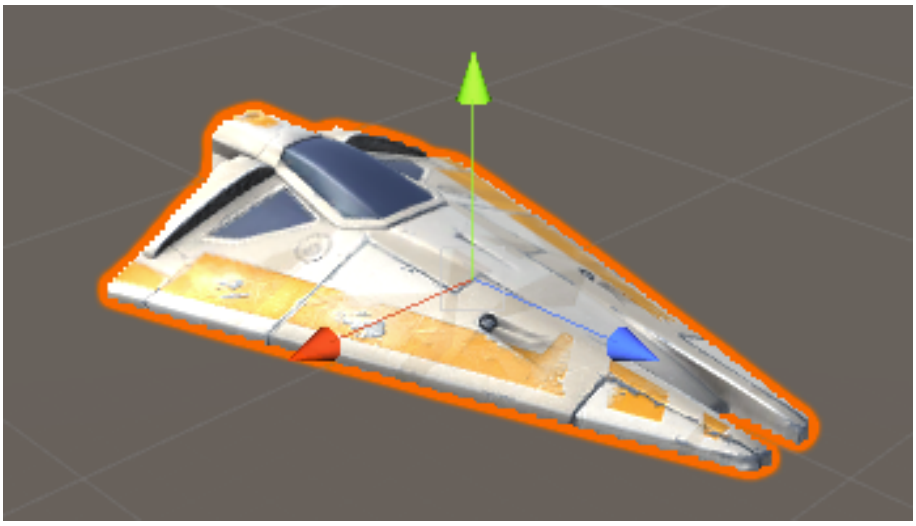
- Under *File > Build Settings*, set the WebGL and click *Switch Platform* button, then *Player Settings*
- Under Resolution and Presentation tab, set the screen width and height to 600 and 900, respectively

The Player gameobject

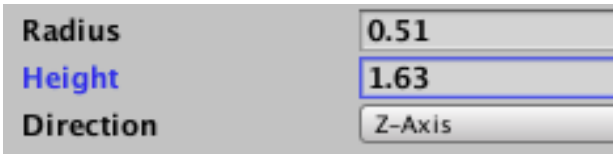
- From the *Assets/Models* folder, select the playerShip vehicle model

drag it in the Hierarchy window and then rename it “Player”.

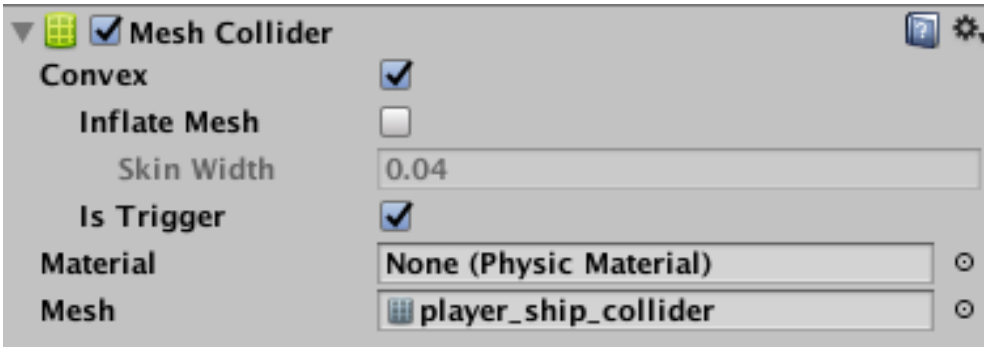
- Double click on it to get the Scene camera focus on the ship



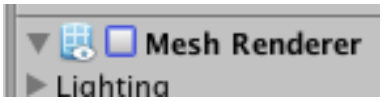
- Reset the object's transform to origin
- Add a *Rigidbody* component to the Player's game object. Uncheck *gravity*, otherwise the object will fall downward when the game starts
- Add a *Capsule Collider* component
- Move the Gizmos in order to get a top-down view of the Player (click the y axes)
- Adjust *Radius* and *Height* of the Collider referred to the z direction



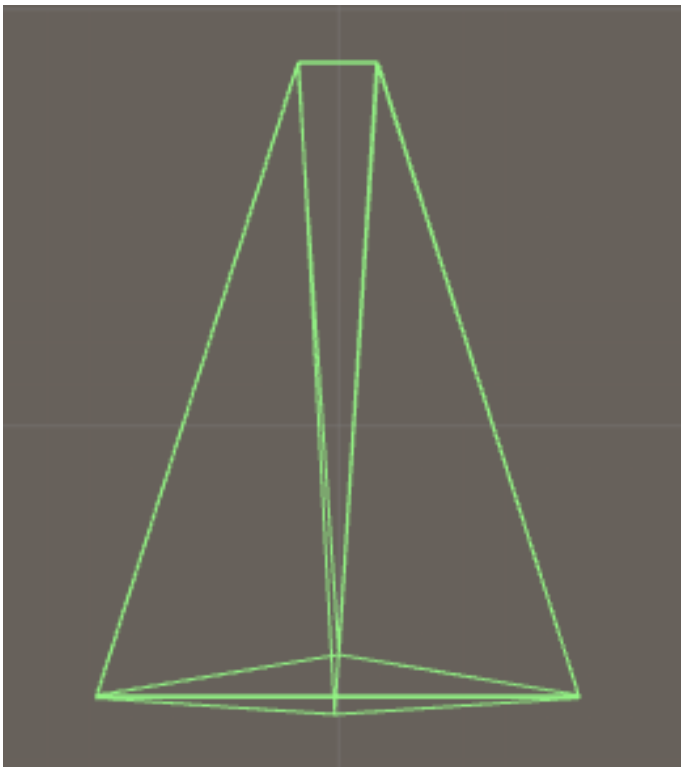
- Another option is to use the *Mesh Collider*. But first remove the *Capsule Collider*. Check both *Convex* and *Trigger* box, and for the Mesh, use the simplified version available in the vheicle model (*player_ship_collider*), instead of *vheicle_playerShip*, the Mesh used by the Mesh Renderer



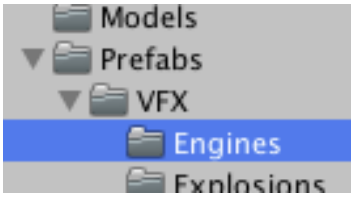
Uncheck the *Mesh Renderer*



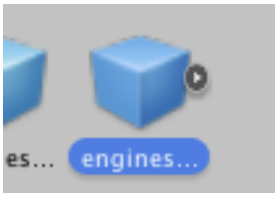
to take a look at this simplified Mesh Collider



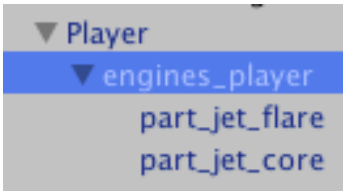
- Open the *Assets/Prefabs/VFX/Engines*



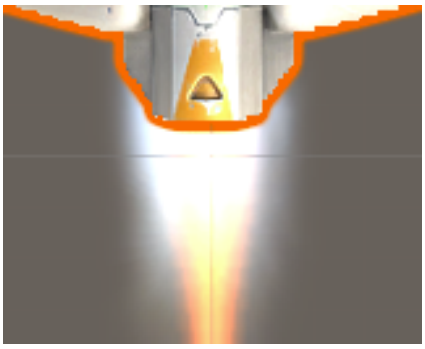
Select the Player in the Hierarchy window, then drag the *engines_player*



on top of the Player gameobject.

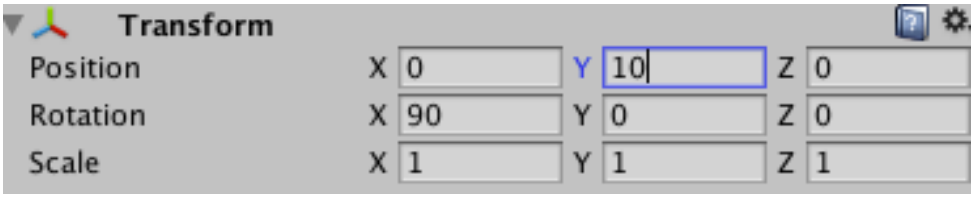


This stuff add the particle effect of the propeller

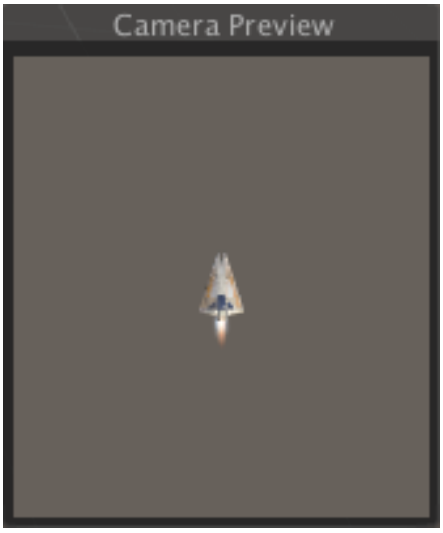


Adjust Light and Camera

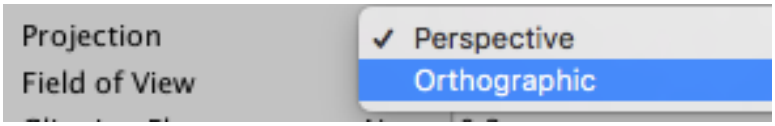
- Select the Main Camera object and reset the transform
- rotate the camera downward and move the camera upward by 10 units



In this way we obtain the desired top-down view

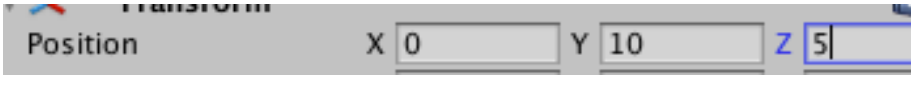


- Now set the Main Camera Projection type to Orthographic.



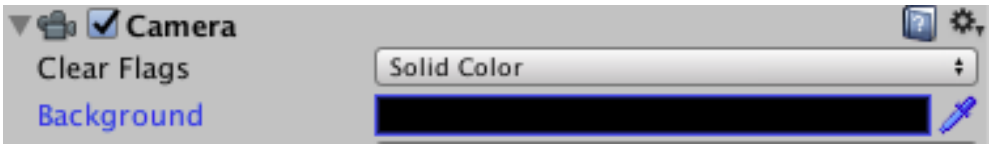
The adjust the Size

and the Z position

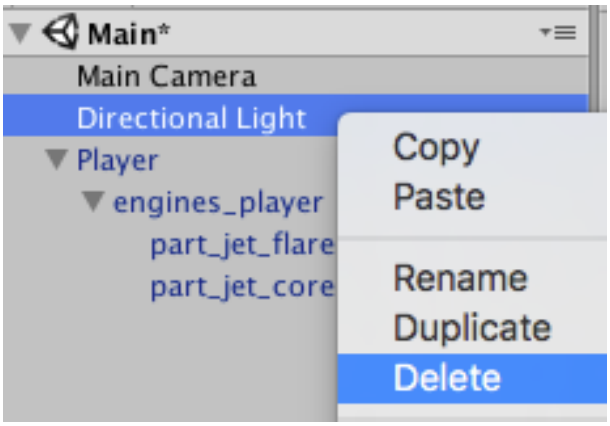


Note that we preferred to move the Camera position, not the Player, in order to have the Player always at the space origin.

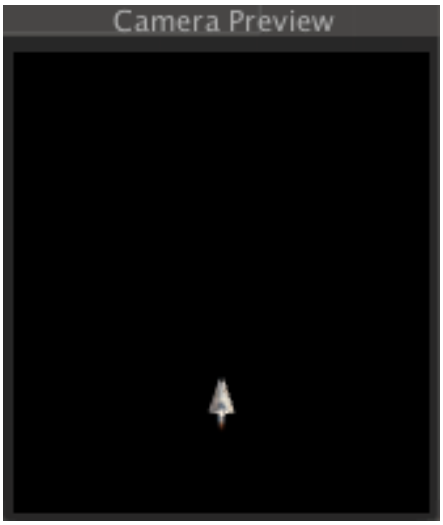
- To set the black space background, set Clear Flags to Solid Color black.



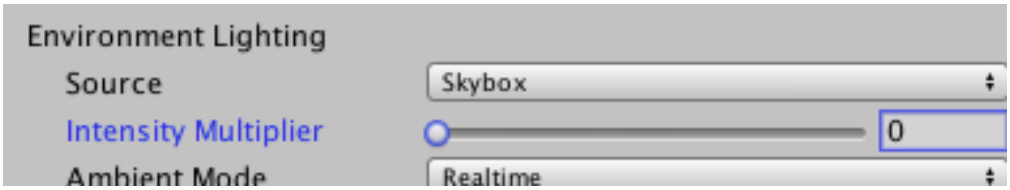
Also remove the Directional Light from the Hierarchy window

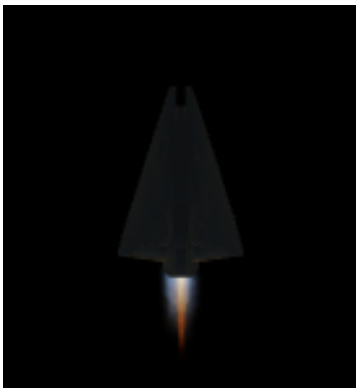


and here you are

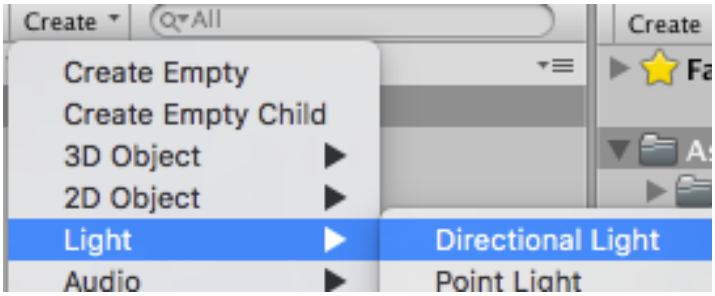


- Now we want to add a lighting of our objects in the scene.
- First of all, we have to remove the ambient diffused light set by default. Click *Window > Lighting > Settings* and under the *Scene* tab, dim to dark the environmental light (*Intensity Multiplier* slider to 0)

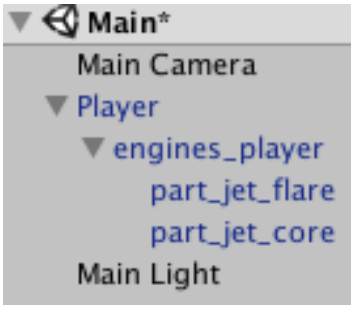




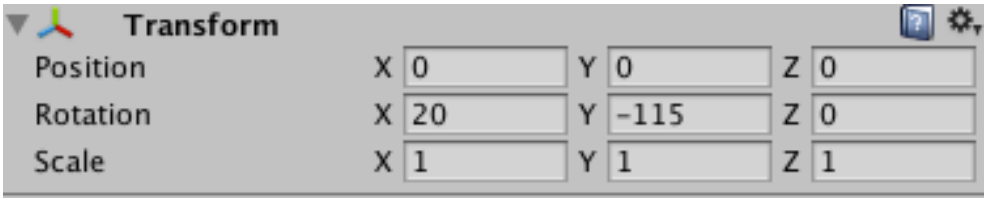
- Now create a *Directional Light* object in the Project window



and call it “Main Light”



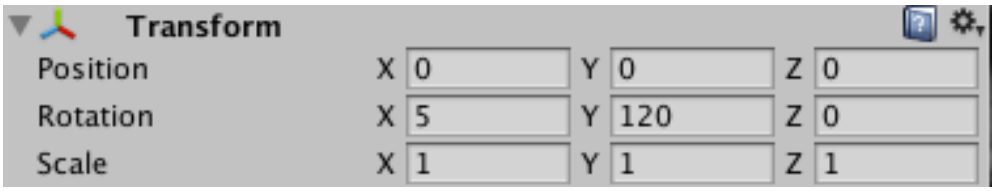
Reset transform of the Main Light object and the adjust the orientation of the light



and also the intensity

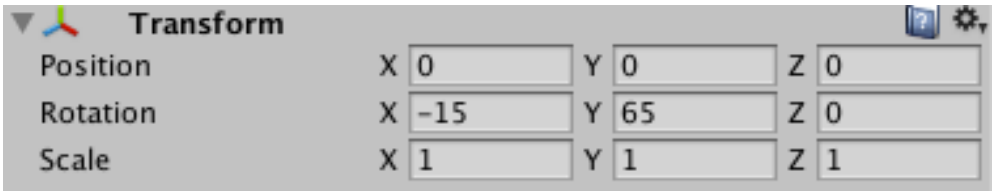
at your choice

- We notice the dark side of the ship is too dark. Let’s create a second source of light by duplicating the Main Light. Rename it “Fill Light” and apply the following transform



color and intensity.

- In order to enhance the ship's edges in the dark side, let's add a third source of directional light, the "Rom Light".



Note: when adjusting a light source, it is worthwhile to switch on-off the other sources, just to assess the contribution of the different sources

- To organise better the Hierarchy window, create an empty game object, rename it "Lighting" and drag all lighting objects on top of it.

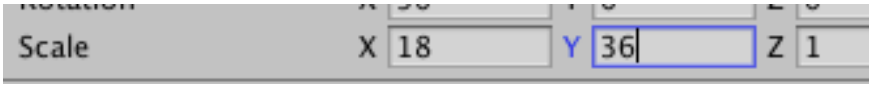
It is a good idea to move this group away from origin, just to clean the gizmos at the origin (this will not affect the lighting effects). Then set position Y=100.

The Background

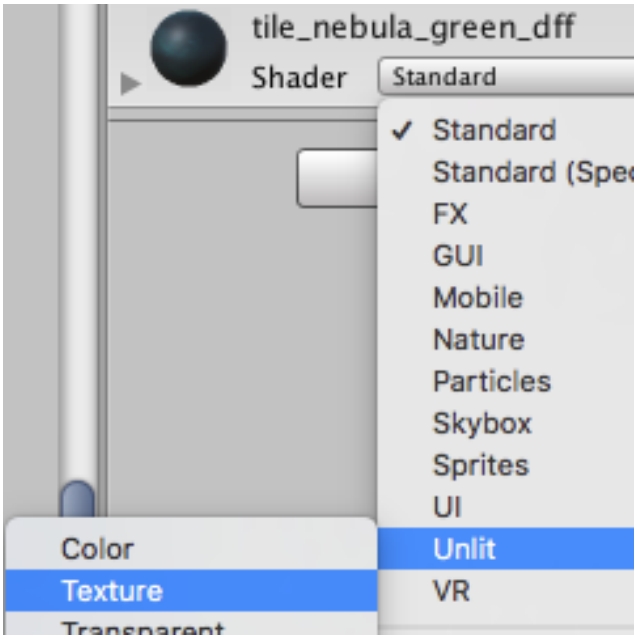
- Switch to the Scene view
- Add a *Quad* gameobject and rename it “Background”
- Reset transform and rotate 90 degrees around x. Now it becomes visible in the Game scene
- Remove the Mesh Collider (we don’t need it)
- Drag the *Default-Material* to the Mesh Renderer (*Element 0* slot)
- In the Toolbar, select the “hand” button

- From the *Assets/Textures* pick up the nebula texture (*tile_nebula_green_dff*) and drag on top of the Background
- Now we have to scale there Background in order to fill the entire Game view. Look at the image properties of the nebula texture

Then, in order to prevent image distortion, we need to constraint the y scale to be twice the x scale.
Scale the Background x up to fill the Game view. We find that 15 is fine. Then set the y scale to 30.



We realise that the background is still too dark. One solution is to change the Shader of the texture from Standard opaque to *Unlit/Texture*

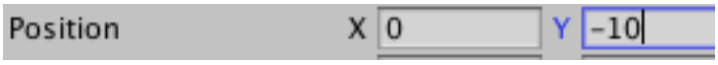


and the difference is quite remarkable.

- We notice a problem, the ship is buried in the middle of the Background.

This is because both are positioned at the origin.

To solve the issue shift the Background -10 in the y direction.



Moving the Player

- Under Assets root folder create a new folder "_Scripts"
- Select the Player object and add it a *New Script* C# component named "PlayerController"
- Drag the script into Scripts folder
- Edit *PlayController.cs*
- Add the callback function *FixedUpdate*(). It will be called right before any new physics calculation
- Inside *FixedUpdate* use *Input.GetAxis* which returns the value of the specified virtual axis. The value will be in the range -1...1 for keyboard. X and Z input values are then used to impress the velocity of the Player's Rigidbody, referenced through the *GetComponent* method in the Start() init function.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour {

    public Rigidbody rb;
    public float speed;

    // Use this for initialization
    void Start () {
        rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void FixedUpdate () {
        float horizontal = Input.GetAxis ("Horizontal");
        float vertical = Input.GetAxis ("Vertical");
        Vector3 movement = new Vector3 (horizontal, 0.0f, vertical);
        rb.velocity = speed * movement;
    }
}
```

- In order to prevent the Player ship to move outside the game area, apply suitable constraints to the X and Z position by using the *Mathf.Clamp* function

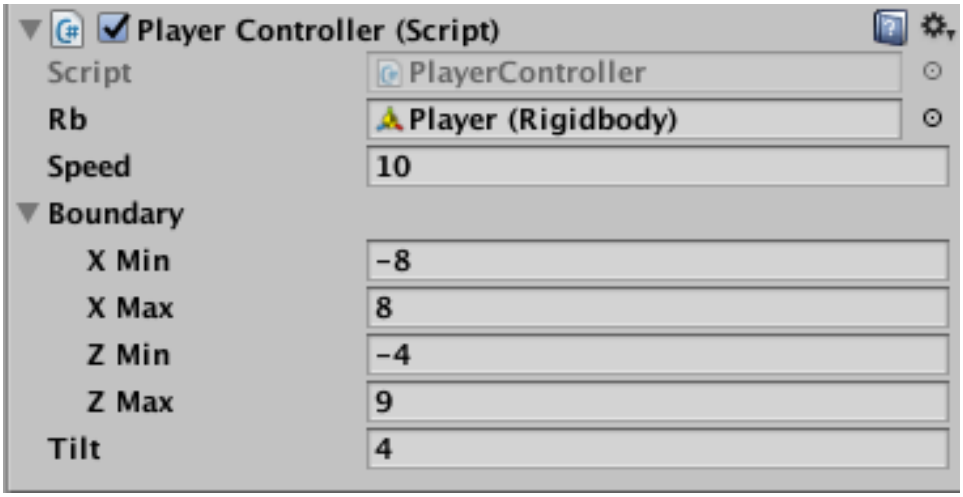
```
        Vector3 movement = new Vector3 (horizontal, 0.0f, vertical);
        rb.position = new Vector3( Mathf.Clamp(rb.position.x, -5.0f, 5.0f), 0, Mathf.Clamp(rb.position.z, -5.0f, 10.0f));
    }
```

The region is bounded by public numbers accessible from the Unity editor using a serializeable class called Boundary

```
 5 [System.Serializable]
 6 public class Boundary {
 7     public float xMin, xMax, zMin, zMax;
 8 }
 9
10 public class PlayerController : MonoBehaviour {
11
12     public Rigidbody rb;
13     public float speed;
14     public Boundary boundary;
15
16     // Use this for initialization
17     void Start () {
18         rb = GetComponent<Rigidbody>();
19     }
20
21     // Update is called once per frame
22     void FixedUpdate () {
23         float horizontal = Input.GetAxis ("Horizontal");
24         float vertical = Input.GetAxis ("Vertical");
25         Vector3 movement = new Vector3 (horizontal, 0.0f, vertical);
26         rb.position = new Vector3(
27             Mathf.Clamp(rb.position.x, boundary.xMin, boundary.xMax),
28             0,
29             Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
30         );
31     }
32 }
```

- Add a tilt rotation to the ship as a function of the X velocity component by using the *Quaternion.Euler*

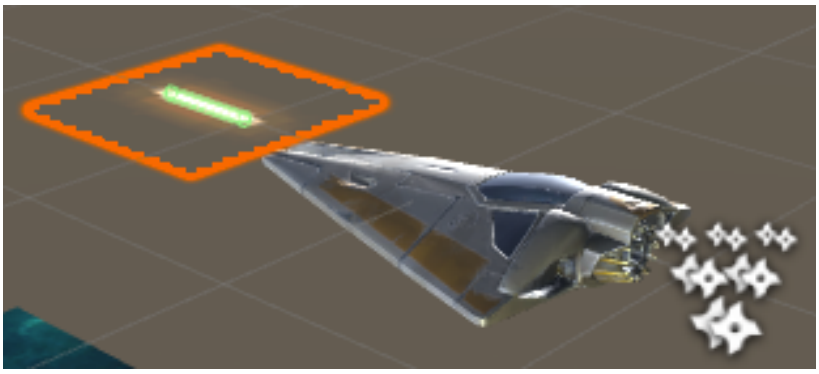
```
27         rb.position = new Vector3(
28             Mathf.Clamp(rb.position.x, boundary.xMin, boundary.xMax),
29             0,
30             Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
31         );
32
33         rb.rotation = Quaternion.Euler (0.0f, 0.0f, -tilt * rb.velocity.x);
34     }
```



Fire shots

- Instantiate is the correct method to clone a game object or a game object's component, inheriting the initial position and rotation.
- To use Instantiate in scripts is equivalent to use Duplicate in the Unity editor.
- Instantiate is most commonly used to instantiate projectiles, AI Enemies, particle explosions or wrecked object replacements
- Switch to Unity editor
- Create a new empty game object "Shot Spawn" and drag into the Player's family. You can think this object as a weapon where to load bolts
- Search for *Done_Bolt* Prefab object, attach it to the Player by drag on top of *Slot Spawn*. Rename as "Bolt"

- Now move the Bolt in front of the ship



by adjusting the *Shot Spawn*'s position

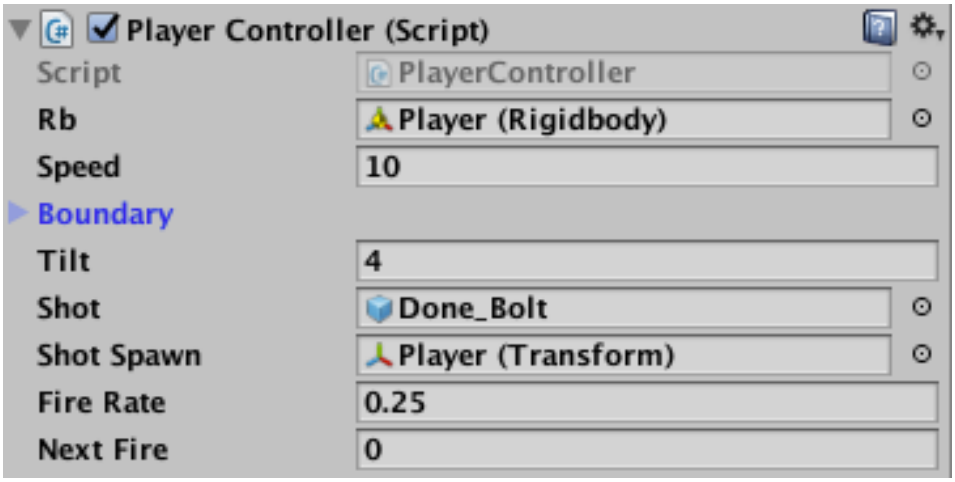
- Delete the Bolt object (we used it just to find the correct Z offset for the *Shot Spawn*)
- In PlayerController.cs, let's add the following piece of code

```
22    // Update is called once per frame
23    void Update () {
24        if (Input.GetButton ("Fire2")) { //this is the Alt key
25            if (Time.time > nextFire) {
26                Instantiate (shot, shotSpawn.position, shotSpawn.rotation);
27                nextFire = Time.time + fireRate;
28            }
29        }
30    }
```

In this code we have created two empty references, one for a GameObject (shot) and one for a Transform of a GameObject (shotSpawn). Once per frame

we check a suitable time has elapsed (fireRate) before to coning a new shot GameObject having the Transform position and Quaternion.Euler rotation provided by shotSpawn.

- Save and turn to Unity editor.
- Drag the *Done_Bolt* Prefab on top of the shot slot of the *PlayerController* component
- Drag the Player gameobject on top of the shotSpawn slot
- Set the Fire Rate to 0.25 in order to fire four shot once a second



- Save and run the game

- The ship fires all the time. Now we want fire only when pressing a button on the keyboard. To do that test the *Input.GetButton* function in the script as follows

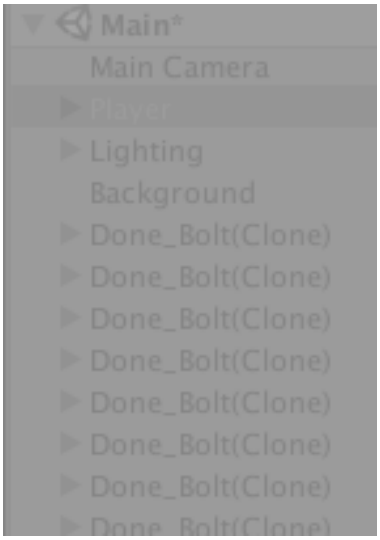
```
22      // Update is called once per frame
23      void Update () {
24          if (Input.GetButton ("Fire1")) {
25              if (Time.time > nextFire) {
26                  Instantiate (shot, shotSpawn.position, shotSpawn.rotation);
27                  nextFire += fireRate;
28              }
29          }
30      }
```

In order to see which physical button corresponds to “Fire2”, go to *Edit > Project Settings > Input* (that is the so called Input Manager). In this case the key is the left Alt

- Save and run the game

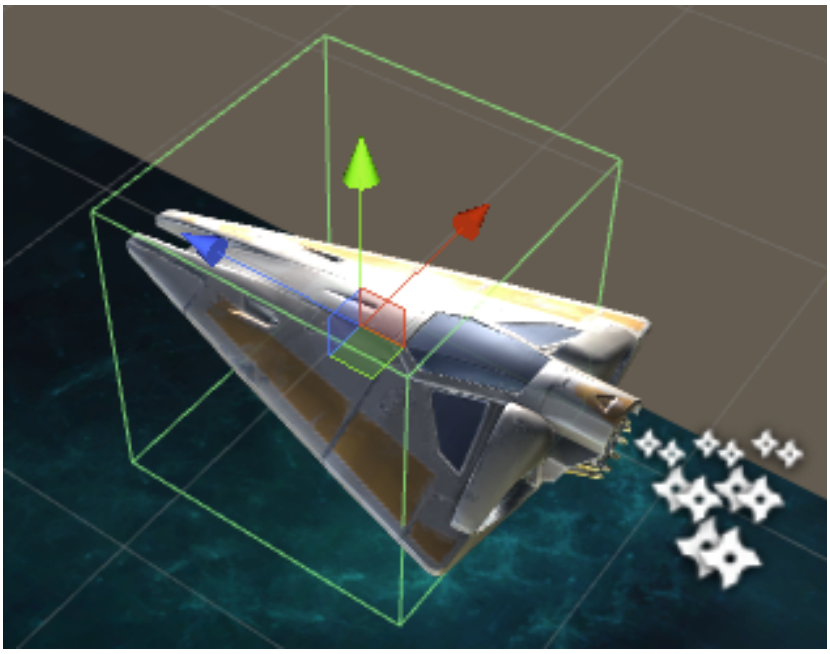
Boundary box

You might notice that for each fired shot a new *Done_Bolt* GameObject is instantiated and lives forever, eroding processing resources

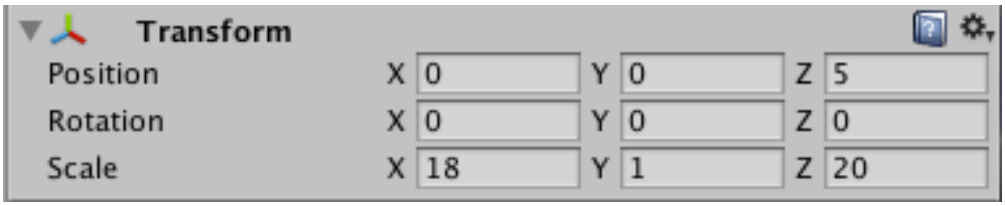


We need a way to remove them when they leave the game area. There are a number of different ways to address the problem. We want to create a box which contains the game area. Any objects would be destroyed as they collide with the box walls.

- Create a new *Cube* GameObject
- Rename it as “Boundary” and reset its transform
- Focus the scene’s camera on *Boundary*

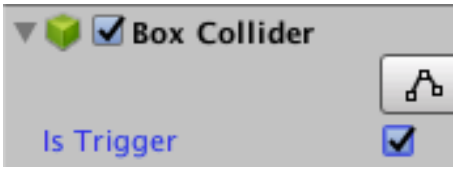


- Switch to the Game view
- Center the Boundary to Z=5, then stretch the X scale of over the entire game area. Do the same for the Z axis (it should be twice the Main Camera size)



- Turn the *Mesh Renderer* off

- Enable “Is Trigger” in the *Box Collider* component

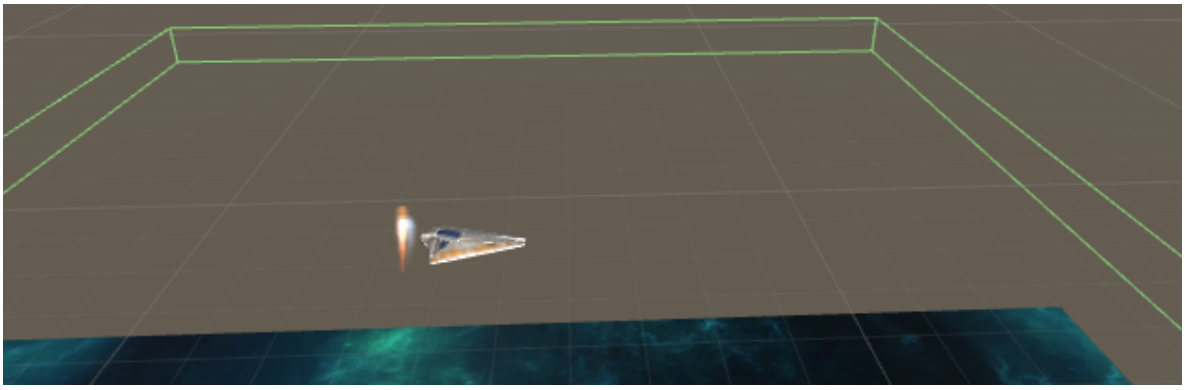


- Add a new script “DestroyByBoundary” and move it in _Scripts folder
- Edit *DestroyByBoundary.cs*

Type the word trigger, select it and see the API references
We want to destroy objects as their Collider stop touching the Boundary box walls.
Then we need the *Collider.OnTriggerExit*

```
4
5  public class DestroyByBoundary : MonoBehaviour {
6
7      void OnTriggerExit(Collider other) {
8          // Destroy everything that leaves the trigger
9          Destroy (other.gameObject);
10     }
11 }
```

- Remove the *Mesh Filter* and the *Mesh Renderer* components of the *Boundary* GameObject because they are not used
- Switch to "2 by 3" layout and run the game. You will see the bold disappear when they touch the Boundary walls



Add Hazards

we are going to create an asteroid object.

- Create a new empty game object "Asteroid"
- From the *Assets/Models* folder, select the *prop_asteroid_01* model and drag it over the *Asteroid* to make it a child object
- Select the *Asteroid* GameObject (not the child !)
- Double click to get the Scene camera focus on the asteroid
- Reset the transform to world space origin, then move at a convenient distance from the ship, let's say Z=8
- Add a *Rigidbody* component to the *Asteroid*. Uncheck *gravity*, otherwise the object will fall downward when the game starts
- Add a *Capsule Collider* component. Adjust *Radius* and *Height* of the Collider to match the *Asteroid* model shape

- Select the *Asteroid* object and add it a *New Script* named "RandomRotator"
- Drag the script into _Scripts folder, then edit it
- In the *Start*() function impress a initial random angular rotation to a public Rigidbody object. The *Random* class can do the job providing a random Vector3 inside the unity sphere. Another public property, called *tumble*, would allow to adjust the magnitude of the rotation

```
 5 public class RandomRotator : MonoBehaviour {
 6
 7     public Rigidbody rb;
 8     public float tumble;
 9
10     void Start () {
11         rb = GetComponent<Rigidbody>();
12         rb.angularVelocity = Random.insideUnitSphere * tumble;
13     }
14 }
```

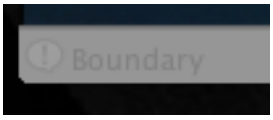
- Save and turn to Unity editor
- Drag the *Asteroid* gameobject on top of the *Rb* slot
- Set the tumble parameter to 5
- Save and run the game
- You may notice that the asteroid rotation gets slower as time passes. To prevent this effect set the Angular Drag parameter of the Rigidbody component to 0
- Save and run the game
- Now we want the asteroid be destroyed by bolt collision
- Select the *Asteroid* object and add it a *New Script* named "DestroyByContact"
- Drag the script into _Scripts folder, then edit it
- A good solution is to use the *OnTriggerEnter*() callback. When another object collider hit the Asteroid both got destroyed

```
5 public class DestroyByContact : MonoBehaviour {
6
7     void OnTriggerEnter(Collider other) {
8         Destroy (other.gameObject);
9         Destroy (gameObject);
10    }
11 }
```

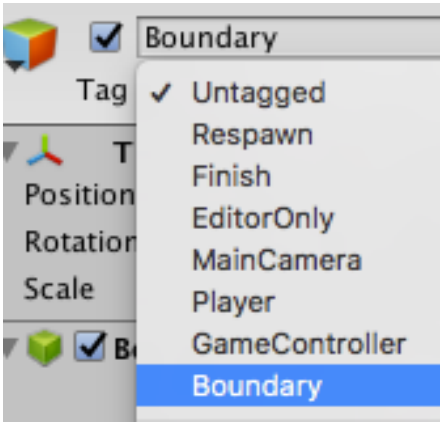
- Save and run the game
- You may notice that the asteroid disappears without any clear explanation. Let's add a debug line to find out the problem

```
7     void OnTriggerEnter(Collider other) {
8         Debug.Log (other.name);
9         Destroy (other.gameObject);
```

- Save and run the game. In the bottom-left corner of the editor we read the reason of Asteroid destruction is it collides with the *Boundary* GameObject



- Then we want to attach a tag "Boundary" for the *Boundary* GameObject



and then test for the colliding object tag in the *DestroyByBoundary.cs*

```
7     void OnTriggerEnter(Collider other) {
8         if (other.tag == "Boundary") {
9             return;
10        }
11        Destroy (other.gameObject);
12        Destroy (gameObject);
13    }
```

- Save and run the game. Great!

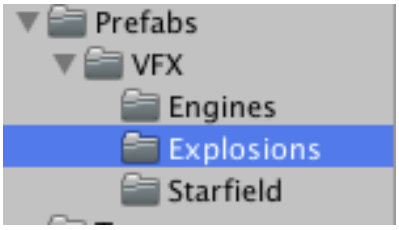
Add Explosion Effects

When the Asteroid is destroyed by boundary collision, we want to instantiate an

explosion effect. We have some ready to use in the Prefabs folder.

```
 5 public class DestroyByContact : MonoBehaviour {
 6
 7     public GameObject explosion;
 8
 9     void OnTriggerEnter(Collider other) {
10         if (other.tag == "Boundary") {
11             return;
12         }
13         Instantiate (explosion, transform.position, transform.rotation);
14         Destroy (other.gameObject);
15         Destroy (gameObject);
16     }
17 }
```

- Save and turn to Unity editor
- Drag the *Assets/Prefabs/VFX/Explosions/explosion_asteroid* gameobject



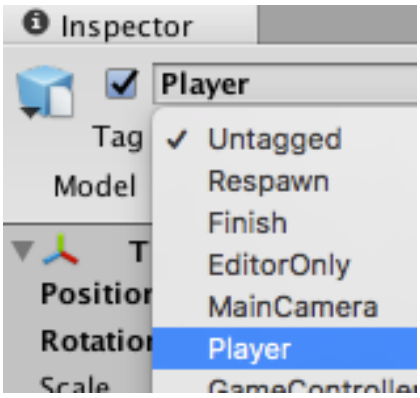
on top of the *explosion* empty slot

- Save and test the game.

However, if the Player ship hit the asteroid, we would like to have the explosion of the ship in addition to the asteroid explosion.
To achieve this, let's change the code a little bit. As usual, use a tag to properly identify the other collider as the *Player* GameObject

```
 5 public class DestroyByContact : MonoBehaviour {
 6
 7     public GameObject explosion;
 8     public GameObject playerExplosion;
 9
10     void OnTriggerEnter(Collider other) {
11         if (other.tag == "Boundary") {
12             return;
13         }
14         Instantiate (explosion, transform.position, transform.rotation);
15         if (other.tag == "Player") {
16             Instantiate (playerExplosion, other.transform.position, other.transform.rotation);
17         }
18         Destroy (other.gameObject);
19         Destroy (gameObject);
20     }
21 }
```

In the Inspector view of the Player, let's add the tag "Player"



Moreover, drag the *Assets/Prefabs/VFX/Explosions/explosion_player* on top of the *playerExplosion* empty slot

Done!

Moving the asteroid

- Now we want the asteroid move at constant speed backward along the Z axis
- Select the *Asteroid* object and add it a *New Script* named "Mover"
- Drag the script into _Scripts folder, then edit it

```
5 public class Mover : MonoBehaviour {
6     public float speed;
7     Rigidbody rb;
8
9     void Start () {
10        rb = gameObject.GetComponent<Rigidbody> ();
11    }
12
13    void FixedUpdate () {
14        Vector3 movement = Vector3.back;
15        rb.velocity = speed * movement;
16    }
17 }
```

- Save and run the game
- Now the *Asteroid* GameObject is completed. We need to make it a Prefab. Drag the *Asteroid* object to the Prefabs folder. Then delete the *Asteroid* object from the Project window.
- Save scenes and project.

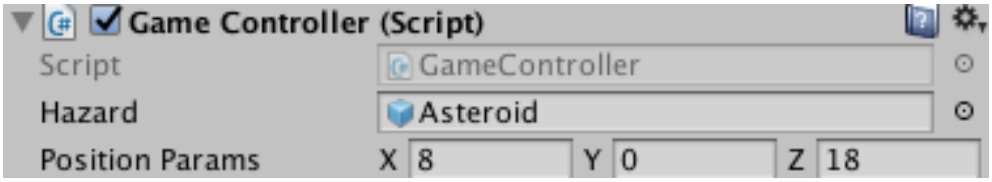
The Game Controller

With this new GameObject we want to spawn the hazards, print the scores, restart the game. These are control functions which do not need Transforms, Renderers or Colliders, just scripts.

- Create the GameObject and call it “Game Controller”, reset its transform, tag it as the “GameController”, add a new script called “GameController”.
- Open the script and create two public reference, one for the hazard object and one for the parameters controlling the initial position of the hazard.
- In the Start() callback we have to call a function we are going to define, called SpawnWaves. This function will instantiate one hazard object.

```
5 public class GameController : MonoBehaviour {
6
7     public GameObject hazard;
8     public Vector3 positionParams;
9
10    // Use this for initialization
11    void Start () {
12        SpawnWaves ();
13    }
14
15    void SpawnWaves () {
16        Vector3 position = new Vector3 (Random.Range(-positionParams.x, positionParams.x), positionParams.y, positionParams.z);
17        Quaternion rotation = Quaternion.identity;  //no rotation
18        Instantiate(hazard, position, rotation);
19    }
20 }
```

- Save.
- From the Unity editor, fill the empty hazard slot with the *Prefabs/Asteroid*
- By inspecting the Player’s position when moving the ship around the play area, we can easily realise that X=8 and Z=18 are good choices for the position parameters to pass to the Game Controller



- Save and run the game
- In order to spawn much more hazards per unit of time we need to move ahead. The following version of the code implements a delay function and some program cycle like the wait and for loops

```
5 public class GameController : MonoBehaviour {
6
7     public GameObject hazard;
8     public Vector3 positionParams;
9     public float nextFire;
10    public int hazardCount;
11
12    // Use this for initialization
13    void Start () {
14        StartCoroutine (SpawnWaves());
15    }
16
17    IEnumerator SpawnWaves () {
18        yield return new WaitForSeconds (4);
19        while (true) {
20            for (int i = 0; i < hazardCount; i++) {
21                Vector3 position = new Vector3 (Random.Range (-positionParams.x, positionParams.x), positionParams.y, positionParams.z);
22                Quaternion rotation = Quaternion.identity;  //no rotation
23                Instantiate (hazard, position, rotation);
24                yield return new WaitForSeconds (nextFire);
25            }
26            yield return new WaitForSeconds (4);
27        }
28    }
```

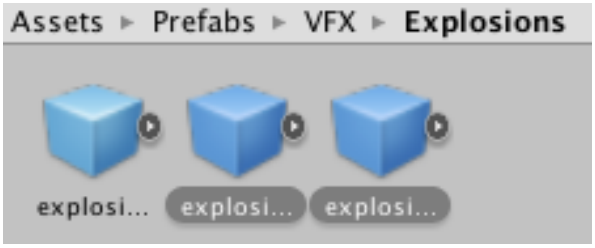
You can also test this simplified version

```
30      void Update () {
31          if (Time.time > nextFire + 0.5f) {
32              Vector2 v2 = Random.insideUnitCircle;
33              nextFire = Time.time + v2.y;
34              Vector3 position = new Vector3 (v2.x * 8.0f, 0.0f, 18.0f);
35              Quaternion rotation = Quaternion.identity;
36              Instantiate (hazard, position, rotation);
37          }
38      }
```

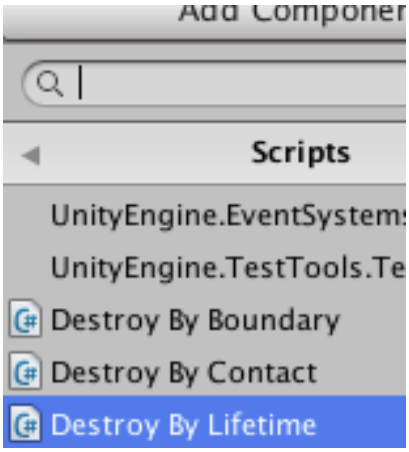
- When running the game, you may notice that the explosion objects accumulate into the scene. Until now we have used a DestroyByContact and a DestroyByBoundary mechanisms to get rid of objects. None of them can be used for explosions. So let's create a new script to the *explosion_asteroid* to handle explosion object destruction by time

```
 5 public class DestroyByLifetime : MonoBehaviour {
 6
 7     public float lifetime;
 8     // Use this for initialization
 9     void Start () {
10         Destroy (gameObject, lifetime);
11     }
12 }
```

In the Unity editor, select both the *explosion_player* and *explosion_enemies*



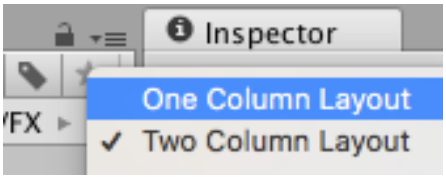
and from the Inspector window, add them jointly the same script *DestructionByLifetime*



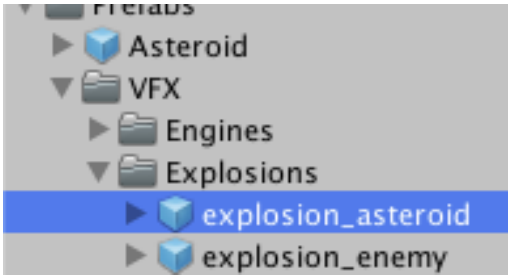
A good value for the lifetime parameter is 2 s.

Audio

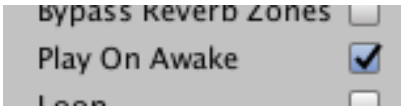
- Select the One column view of the Project window.



- Select the *explosion_asteroid* Prefab object



- Add a Audio *Source* component
- Drag the *explosion_asteroid* audio clip into the audio source's reference slot.
- Check the "Play On Awake" setting is on. This is because we want the audio clip be played when the *explosion_asteroid* object is instantiated.



- Do the same with the *explosion_player* Prefab and audio clip
- Select the *weapon_player* audio clip and drag on top of the *Player* GameObject. This will create the Audio Source automatically with the referenced audio clip
- For the *weapon_player* we want the *Play On Awake* be unchecked. Instead we need this sound be played each time a new shot is fired
- To do that, open the PlayerController script and play the audio clip when a new shot is spawned

```
21      private Rigidbody rb;
22      private new AudioSource audio;
23
24      // Use this for initialization
25      private void Awake () {
26          rb = GetComponent<Rigidbody>();
27          audio = GetComponent<AudioSource>();
28      }
29          |
30      // Update is called once per frame
31      void Update () {
32          if (Input.GetButton ("Fire2")) { //this is the Alt key
33              if (Time.time > nextFire) {
34                  Instantiate (shot, shotSpawn.position, shotSpawn.rotation);
35                  audio.Play ();
```

- Select the *music_background* audio clip and drag on top of the *Game*

Controller GameObject. This will create the Audio Source automatically with the referenced audio clip
- Set both the *Play On Awake* and the *Loop* checkboxs.
- Now we ave all the active Audio Sources in place at full volume. Reduce from 1 to 0.5 the Volume for the Player weapon and for the background music.

Score text GUI and logic

We want to add some GUI textual element to print the total score.

- Create a new empty object “Score Text”
- Add a *GUI Text* component
- It is worth to note that GUI objects lie in the so called Viewport space which is (0,0) in bottom-left corner and (1,1) in top-right corner, as opposed to the Screen space which is measured in pixels (600x900 in our example).
- Set the transform position to (0,01,0). It holds in the Viewport space
- Set Text value to “Score Text”
- Set “Pixel Offset” to X=10, Y=-10. It holds in the Screen space
- Open the *GameController* script
- Add a public reference to the GUI Text “scoreText”, a private int variable to store the score, a function to update the text of scoreText and a new public method GameController.AddScore(int scoreValue) to be called from any instance of a DestroyByContact.
- To do that edit the *DestroyByContact* script and add a public int to assign a score to add for any hazard object destroyed by contact. Moreover, we need a reference to the GameController instance to call the AssScore method. We can’t simply declare a public GameController object, because our hazard objects (for example Asteroids) are Prefab, and a prefab would not accept any object reference to be dragged on its components. This is because a prefab is a template for object instances, not object instances. So, the correct way is to exploit the GameController’s tag from within DestroyByContact. A tag can be used to identify a game object. This way

```
5  public class DestroyByContact : MonoBehaviour {
6
7      public GameObject explosion;
8      public GameObject playerExplosion;
9
10     private GameController gameController;
11     public int scoreValue;
12
13     void Start() {
14         //we search for the GameObject instance which is tagged as "GameController"
15         GameObject gameControllerObject = GameObject.FindWithTag ("GameController");
16         if (gameControllerObject != null) {
17             gameController = gameControllerObject.GetComponent<GameController> ();
18             if (gameController == null) {
19                 Debug.Log ("Cannot find the GameController instance");
20             }
21         } else {
22             Debug.Log ("Cannot find a GameObject with 'GameController' tag");
23         }
24     }
25
26     void OnTriggerEnter(Collider other) {
27         if (other.tag == "Boundary") {
28             return;
29         }
30         Instantiate (explosion, transform.position, transform.rotation);
31         if (other.tag == "Player") {
32             Instantiate (playerExplosion, other.transform.position, other.transform.rotation);
33         }
34         Destroy (other.gameObject);
35         Destroy (gameObject);
36
37         gameController.AddScore (scoreValue);
38     }
39 }
```

- Get back to the Unity editor
- Drag the *Score Text* object onto the Score Text empty slot of the Game Controller
- Select the *Assets/Prefabs/Asteroid* and set the Score Value to, let's say, 10. This means that each time a asteroid is hit and destroyed by contact, the Score increases by 10 units.
- Save the scene, the project and test.
- Now we want to add two more GUI text, one for the game over and one for the restart game.
- Create an empty GameObject called "Display Text", reset the transform, then drag the Score Text object into

- Create a new GUI Text object "Restart Text", and place it on top-right corner of the Viewport space, by setting the transform position to (1,1,0)
- Set Anchor to "Upper right" and Alignment to "Right". Set "Pixel Offset" to X=-10, Y=-10

Game Over and Restart GUI and logic

- Create a new GUI Text object "Game Over Text", and place it at the center of the screen, by setting the transform position to (0.5,0.6,0)
- Set Anchor to "middle center" and Alignment to "center"
- Move the *Game Over Text* object into the *Display Text* parent container
- Edit the *GameController* script and create a reference for the additional GUI

Text we have just created. Add a boolean *gameover*

```
16      public GUIText restartText;
17      public GUIText gameoverText;
18      private bool gameover;
```

- Reset all this stuff in the Start() function. Add a public method called "GameOver" in which the *gameover* flag is turned true and the "Game Over" text is displayed on the screen. We can give the user the chance to restart the game

```
48      public void GameOver() {
49          gameover = true;
50          gameoverText.text = "Game Over !";
51          restartText.text = "Press 'R' to restart the game";
52      }
```

The Update() function can now make use of the gameover flag in order to stop cloning new hazards when the game is over, and waiting for the user press of a key to restart the game

```
28      void Update () {
29          if (!gameover) {
30              if (Time.time > nextFire + 0.5f) {
31                  Vector2 v2 = Random.insideUnitCircle;
32                  nextFire = Time.time + v2.y;
33                  Vector3 position = new Vector3 (v2.x * positionParams.x, positionParams.y, positionParams.z);
34                  Quaternion rotation = Quaternion.identity;
35                  Instantiate (hazard, position, rotation);
36              }
37          } else if (Input.GetKeyDown(KeyCode.R)) {
38              //Application.LoadLevel(Application.loadedLevel); //this is an obsolete method
39              SceneManager.LoadScene (SceneManager.GetActiveScene().name);
40          }
41      }
```

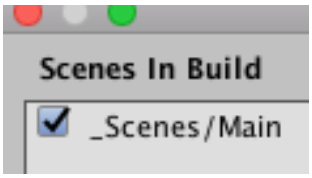
When the user press 'R' on the keyboard the scene is reloaded and the game start over. Note that, to make use of the SceneManager class you need to import the *UnityEngine.SceneManagement*.

- Save and get back to the Unity editor
- Drag the *Restart Text* and the *Game Over Text* objects onto the corresponding empty slots of the *Game Controller*.

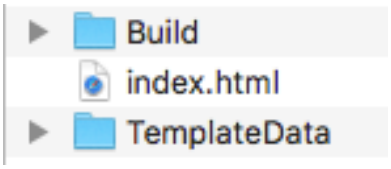
Build the Game

We want to build the Space Shooter game for a WebGL platform, i.e. it can run in a internet browser, either locally or globally on the internet.

- go to *File > Build Settings*
- Select the WebGL Build Platform
- Click "Switch Platform"
- Click "Add Open Scenes" button, or from *Assets/_Scenes* manually drag the scene(s) you want to build into the window



- check the *Player Settings*, then click the *Build* button
- When prompted to save the build, create a new subfolder "Builds" in the project folder, along side *Assets*, etc., and save as "Space_Shooter".
- The underscore is required because we are deploying for a WebGL, that means the build name will be part of a URL, where empty spaces are not allowed.
- In the build folder you will find the html to play the game from within a web browser



The index.html is the file to index the program location in the local filesystem file:///Users/gabrielefilosofi/UnityProjects/SpaceShooter/Builds/Space_Shooter/index.html
However, if you want to play the game from any other network client, copy the Build folder in a reachable server, and provide the html to the player's client.

Add more asteroids

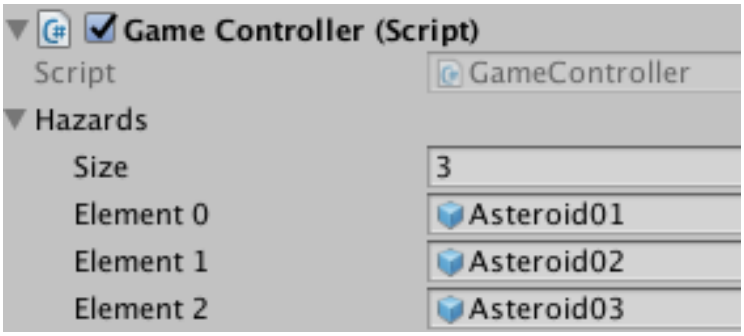
- Actually we have three different models of asteroid, but we have used only one.
- Now we are going to use all of them.
- Drag a Assets/Prefabs/Asteroid into the Hierarchy window, duplicate it and rename Asteroid02, duplicate it and call Asteroid03.
- Drag Assets/Models/prop_asteroid_02 on top of Asteroid02 and remove the prop_asteroid_01
- Drag Assets/Models/prop_asteroid_03 on top of Asteroid03 and remove the prop_asteroid_03
- For Asteroid02 and Asteroid03, adjust the Capsule Collider Axis, Height and Radius
- Drag Asteroid02 and Asteroid02 into the Assets/Prefabs folder
- Rename Asteroid prefab as Asteroid01, just to keep the naming convention
- Now we have three different kind of hazards, then we have to change the logic
- Edit the GameController script
- Change the hazard type from GameObject GameObject[]

```
8     public GameObject[] hazards;
```

Then instantiate hazard picked up from the array at a random index

```
GameObject hazard = hazards[Random.Range(0, hazards.Length)];
Instantiate (hazard, position, rotation);
```

- Save and switch to the Unity Editor. Here we have to specify the Size of the Hazards array and populate the reference slots with the three different asteroid tamplates from the Prefabs folder

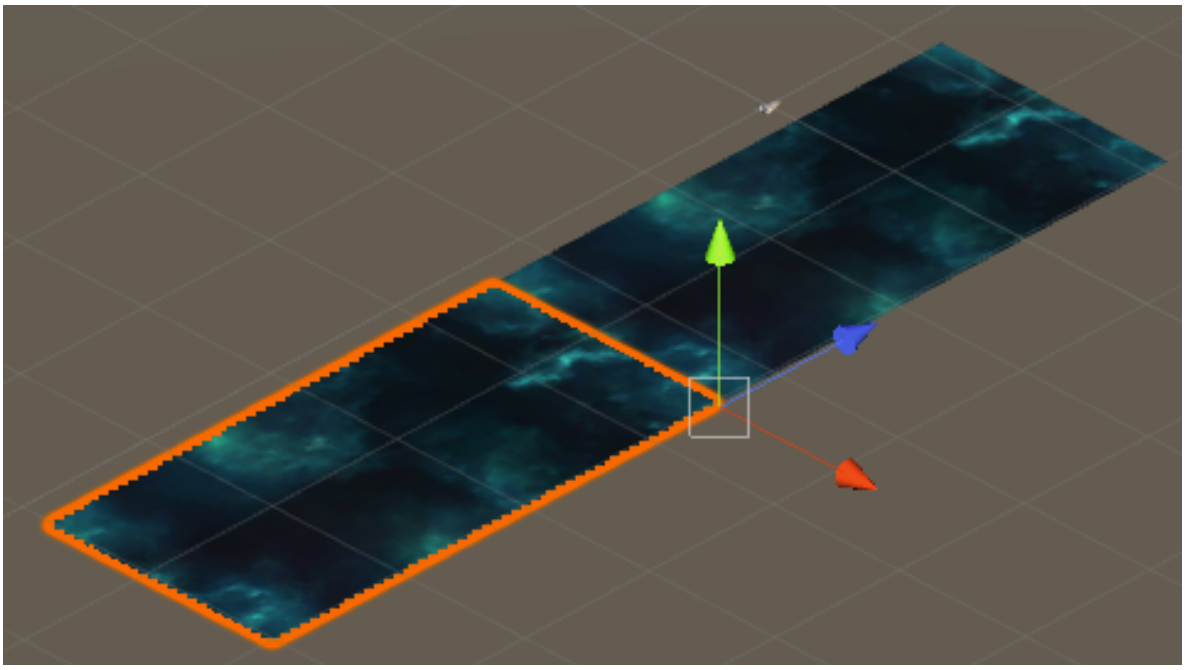


- Save and run.

Star field and scrolling Background

- The background is a static image.
- We want to add a sort of kind a animated star field.
- Drag *Assets/Prefabs/VFX/StarField* into the Hierarchy window
- Save and test
- The *StarField* object is made out of two distinct particle systems, the smaller stars and the bigger stars. They move at different speed resulting in a parallax visual effect
- If you expand the *StarField* object and select one of the two particle systems, a bunch of parameters open up in the Inspector window. If you click the *Open Editor* button you can detach the control panel from the component and move around

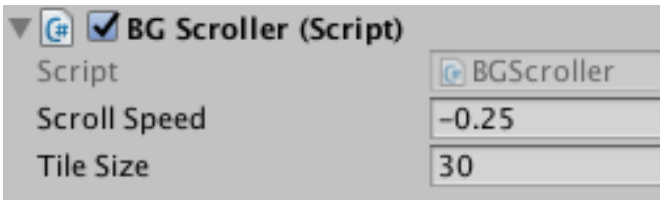
- Now we want to make the background image scrolling slowly downward the Z axis. We can accomplish this by rotating the Background image using a script. But first we want to attach a shifted copy to the original image in order twice the height
- Make a copy of the Background object and drag it into the original one, to make it a child. This is because we will attach the rotation script to the parent object only, and this will move both



- On the parent Background add a new script “BGScrolled”.
- In this script we are going to use *Mathf.Repeat(a,b)* which is similar to the modulo operation *mod(a,b)*, but with a and b floating point numbers.

```
 5 public class WeaponController : MonoBehaviour {
 6
 7     public GameObject shot;
 8     public Transform shotSpawn;
 9     public float fireRate;
10     public float fireDelay;
11     private new AudioSource audio;
12
13     void Start () {
14         audio = GetComponent<AudioSource> ();
15         InvokeRepeating ("Fire", fireDelay, fireRate);
16     }
17
18     void Fire ()
19     {
20         Instantiate (shot, shotSpawn.position, shotSpawn.rotation);
21         audio.Play ();
22     }
23 }
```

- Turn back to Unity editor and set the *scrollSpeed* and the *tileSize*. The *scrollSpeed* has to be negative (the scroll direction is downward) and smaller than the smallest speed of the StarField’s particle systems. The *tileSize* is equal to the Y value of the Background’s transform Scale, that is 30



Enemies

Now we want to add enemies.

- Create a new empty GameObject and rename it “Enemy Ship”
- Give to the object a suitable position to work with, for example (0,0,9)
- Add the 3D model to the *Enemy Ship* object. You can use *Assets/Models/vheicle_enemyShip*. Drag it onto the object to make it a child
- Rotate the *veichle_enemyShip* child by 180° around the Y axis
- Add the engine to the *Enemy Ship* object. You can use *Assets/VFX/Engines/engines_enemy*
- Add a *Rigidbody* component, to allow the ship to participate to physical interaction, and provide object’s velocity. Uncheck the “Use Gravity” box
- Add a *Sphere Collider* component. Set the Center and Radius to match the vehicle shape. Check the “Is Trigger” box
- Select the *Assets/Prefabs/VFX/Explosions/explosion_enemy* Prefab template, add it a *Audio Source* loaded with eh *Assets/Audio/explosion_enemy* audio clip
- Select the *Enemy Ship* object, add it a *DestroyByContact* script, then populate the *Explosion* and the *Player Explosion* empty slots with the enemy’s and player explosion Prefabs, respectively. Set the Score Value to 20.
- Add a empty script called “WeaponController”

- Apply the tag “Enemy” to *Enemy Ship*, to all *Asteroids* prefabs and to the *Bolt Enemy* prefab. This tag will be used to into the script ..

- Into *Assets/Prefabs/* create a folder Asteroids and move there all asteroid prefabs. Into *Assets/Prefabs/* create a folder Bolts and move there all bolt prefabs

- Select the *Enemy Ship* object, add it the *Mover* script (the same already used for the asteroids)
- Select the *Enemy Ship* object, add it a new *EvasiveManeuver* script,

```
public class EvasiveManeuver : MonoBehaviour {

    public float dodge;
    public float smoothing;
    public float tilt;
    public Vector2 startWait;
    public Vector2 maneuverTime;
    public Vector2 maneuverWait;
    public Boundary boundary;
    private float targetManeuver;
    private float currentSpeed;
    private Rigidbody rb;

    void Start () {
        rb = GetComponent<Rigidbody> ();
        StartCoroutine (Evade());
    }

    IEnumerator Evade ()
    {
        yield return new WaitForSeconds (Random.Range(startWait.x, startWait.y));

        while (true) {
            targetManeuver = Random.Range (1, dodge) * -Mathf.Sign(transform.position.x);
            yield return new WaitForSeconds (Random.Range(maneuverTime.x, maneuverTime.y));
            targetManeuver = 0;
            yield return new WaitForSeconds (Random.Range(maneuverWait.x, maneuverWait.y));
        }
    }

    void FixedUpdate () {
        float newManeuver = Mathf.MoveTowards(rb.velocity.x, targetManeuver, Time.deltaTime * smoothing);
        rb.velocity = new Vector3 (newManeuver, 0.0f, rb.velocity.z);
        rb.position = new Vector3
        (
            Mathf.Clamp (rb.position.x, boundary.xMin, boundary.xMax),
            0.0f,
            Mathf.Clamp (rb.position.z, boundary.zMin, boundary.zMax)
        );
        rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
    }
}
```

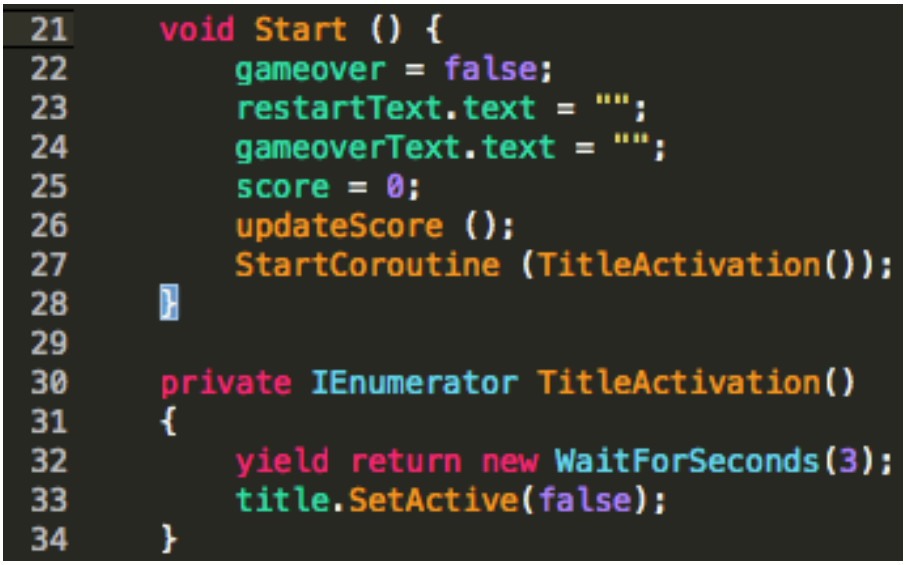
You can make the enemies tracking the Player's position. To do that you need to get the reference of the Player GameObject within the EvasiveManeuver script. This script is a component of a Prefab (the Enemy Ship), so you cannot use a public GameObject and fillll it from the Inspector window, because Prefab objects are created at runtime. The solution is to search the Player GameObject by tag.

```
    private GameObject player;

    void Start () {
        player = GameObject.FindGameObjectWithTag ("Player");
        rb = GetComponent<Rigidbody> ();
        StartCoroutine (Evade());
    }

    IEnumerator Evade ()
    {
        yield return new WaitForSeconds (Random.Range(startWait.x, startWait.y));

        while (true) {
            if (player != null) {
                targetManeuver = player.transform.position.x;
            }
```

The Title image

- Switch to the Scene view

- Add a *Quad* gameobject and rename it “Title”
- Reset transform and rotate 90 degrees around x. Now it becomes visible in the Game scene
- Move at (0,0,5)
- Remove the Mesh Collider
- Drag the Default-Material to the Mesh Renderer (*Element 0* slot)
- Create the title image, *title_01.tiff*. For example, go to https://textcraft.net and download the PNG image, then export as TIFF
- Drag the image into *Assets/Textures*
- Drag *Assets/Textures/title_01* onto Title object
- Adjust poising, for example (0,-4,12), and Scale, for example (10,2,1)
- Set the Shader to *Unlit/Texture*
- Open the GameController script
- add a public GameObject reference *title*, then create the appropriate logic to turn off the title after few seconds. You can use StartCoroutine



```
21      void Start () {
22          gameover = false;
23          restartText.text = "";
24          gameoverText.text = "";
25          score = 0;
26          updateScore ();
27          StartCoroutine (TitleActivation());
28      }
29
30      private IEnumerator TitleActivation()
31      {
32          yield return new WaitForSeconds(3);
33          title.SetActive(false);
34      }
```

- From the Unity editor drag the *Title* object onto the title empty slot of the Game Controller