Project 2 Space Shooter

ZHU Yachao
Guangxi University of Science and Technology
Created:2017-03-06
Last Updated:2017-04-20
Contact:46354184@qq.com

Objectives

In this project, we will build this top-down arcade style shooter. In the foundation section, we will import the assets we need and set our basic project and set the build target for our game. We will then

- set up a basic player game object preparing it to move shot and interact with the other objects in the scene
- set up the camera, lights and background to prepare our scene
- show how to set up game objects attach pre-made components and use our models and artwork in our scene
- write simple custom code in C sharp to capture the player's input and use it to move our player ship after the scene set up
- create laser bolts for our player to shoot
- create asteroid hazard to challenge our player
- write a game controller to control our game, spawn our hazards in waves, count and display our score and manage how we end and when we can restart our game
- add audio effect and a background music track
- finally build and deploy the game to the web

In move advanced lessons, we will use this project for further learning where we will add enemy ships scroll the background and add the star field and more.

Table of Contents

1. Game Setup, Player and Camera

1.1 Setting up the project

We will create a new project, import assets, save main scene and set up basic foundations for our game.

1. Creating a new project 【File | New Project 】
    a. Put this project on the Desktop
    b. Call it Space Shooter
    c. Create project
2. Importing the assets for this project
    a. Download the assets from the learn site and Asset Store
        i. Click【 Asset Store| Top Free | Space Shooter tutorial | Download 】(or Search Space Shooter in the filters bar and click on Space Shooter tutorial)
        ii. Or Download the assets for free from the Asset Store here
    b. Import the complete project
        i. Click the Import button on the Asses Store View
        ii. Import complete project
        iii. Make sure all the files we need for this project is selected, if not click the All button in the button left, then select import to import all these assets[1].
3. Saving the scene[2] and name it Main
    a. 【File | Save Scene 】or【Cmd + S(Win) / Ctr + S(Mac)】 (hot key combination)
    b. Create a folder in the assets directory called _Scenes to hold the scenes
    c. Name the scene Main and save it into _Scenes folder
4. Setting our build target
    a. 【File | Build Settings 】or【Shift + Cmd + B(Mac)/ Shift + Ctrl + B(Win)】
    b. Change the build target to **WebGL** as we will be building and deploying this project to the web
        i. Click on **WebGL** to select the WebGL build target
        ii. Click open download page to download WebGL Support Installer
        iii. Download and install it
        iv. Click Switch Platform
        v. Check the header bar to make sure our build target WebGL is listed in
    c.
5. Setting the build details for our game
    a. Click on Player Settings to select Player Settings from build settings window

[1] Unity will copy these files to our new project, compile any new scripts and import all of the assets. We will be creating this project from scratch using the assets provided. If, however, you are confused by any step, this project does include a _Completed folder which contains the entire project in finished form.

[2] We get a hint to remind us if our scene isn't being saved if we read the header bar. The header bar always lists the scene, project name and build target. Our head bar shows "Untitled –Space Shooter – PC, Mac & Linux Standalone". Untitled is our unsaved scene, Space Shooter is our current project, and the item is our build target

b. Or click【Edit | Project Settings | Player 】 to inspect and change our player settings (With the player setting opens in the inspector, we can change both project-wide settings and platform-specific settings)
c. Click the details link (help button in the gear button left) for more information on Player Settings
d. Setting the aspect ratio(resolution) to 600 by 900
    i. ~~Switch to Settings for PC, Mac & Linux Standalone~~
    ii. ~~Unselect the Default is Full Screen~~
    iii. ~~Set Screen Width = 600~~
    iv. ~~Set Screen Height = 900~~
    v. Click【 Game | Free Aspect | + 】
    vi. Set Width = 600, and set Height = 900

6. Changing the layout and saving it. You can skip this step to the next, if you don't want to.
    a. Change the layout
    b. Click【 Layout | Save Layout | Save 】Save the layout as Space Shooter on the tools bar
7. Switching the Clear Flag of Main Camera to Solid Colour from Skybox
    a. Select and click 【 Hierarchy | Main Camera |】
    b. Select and click 【 Inspector | Camera | Clear Flag | Skybox】, and switch to【Solid Colour】

Readings

1.2 The player GameObject

1. Setting up the player game object
    a. Find the playerShip in the assets folder in the Models directory 【 Project | Assets | Models | vehicle_playerShip 】
    b. Drag and drop vehicle_playerShip model into the Hierarchy view or drag it directly into the scene view. Either way is correct.
    c. Focus or framed Select the vehicle_playerShip game object with the scene view camera to get a better view of the model in our scene. We can do this by either:
        i. Choosing 【 Edit | Frame Selected 】
        ii. Using a hot key F while the pointer is in the scene view
        iii. Double clicking on the game object in the Hierarchy view. This way will also focus the scene view camera
    d. Rename it in the Hierarchy view
        i. Click the game object in the hierarchy
        ii. Type either the return or enter key to enable editing or click on the game object twice slowly to enable editing as well
        iii. Name the game object "Player" and then hit enter or return to confirm that change
    e. Reset the player's transform to make it to be at origin
        i. Find the context sensitive gear menu in the upper right of the transform component
        ii. Select Reset
2. To set up our player game object, we need to add more components that perform specialized functions and we will be creating our own components using simple scripting. We will be moving our ship using physics. Though with an arcade style we need physics to detect collisions between the player and other game objects in the scene. To use physics, we need to add a rigidbody component. Let's
    a. Select the player game object in the Hierarchy view[3], Click on 【 Inspector | Add Component 】 and select 【 Physics - Rigidbody 】[4]
    b. Deselect Use Gravity
3. To detect the collision, the physics engine through the rigidbody needs to know the volume of our objects. We need to know how much space these game objects take up in our game to calculate the collisions. We give this information to the rigidbody by using a cage that we wrap around our game object. This cage defines the volume of that object. The cage is called a Collider. Let's
    a. Click on Add Component button and Select 【 Physics - Capsule Collider 】 this time. This put a simple cage around our game object. This looks like a sphere, that's because the capsule collider is defined by two sphere and the space between them, and we are seeing both two spheres in the same place. Let's change the Capsule Collider's size
    b. Click on 【 Inspector | Capsule Collider | Direction 】 and select the Z Axis. The default orientation for a capsule collider is Up and Down or along the Y

[3] Before doing this, I am going to quickly reduce the view of the reference materials by clicking on the header bars to, so it's easier to see and access Add Component button without scrolling.
[4] This attaches a rigidbody component to our game object. By default, the rigidbody assumes we want to use gravity. We're in space, we don't want to fall out of the game.

Axis. This is to fit a human old object. Our shape is longest along Z Axis. Let's change the direction to Z Axis.

c. Reduce the radius and Increase the height of the Capsule Collider.

4. For a better view, let's change our orientation
   a. For a top-down view, let's Click on the scene view Gizmos and Click on the Y arm[5]
   b. We simply need to choose the values for radius and height that comfortably **fit** the collider to our model.
5. Removing the Capsule Collider and using the Mesh Collider instead. For the purpose of this game, the capsule collider is sufficient. There are other alternative however. If, however, we have a more complex shape that can't be accommodated by any of the primitive colliders and for some reasons it doesn't work by using a compound collider, we can select Mesh Collider.
   a. Click on【 Inspector | Add Component | Physics | Mesh Collider 】[6]
   b. Click on【 Inspector | Capsule Collider 】 Gear button in the help button right and select Remove Component to remove the Capsule Collider
   c. Unselect【 Inspector | Mesh Render 】 to turn off the mesh render for having a better look of the mesh collider
   d. Select【 Inspector | Mesh Collider | Convex 】 Convex to reveal the green lines of the mesh collider that were hidden underneath of the rendered mesh
6. We can see how complex this cage is. Unity must check the position of each triangle in the cage relative to the other colliders in the scene to properly detect the collision. If for whatever reason we use a mesh collider rather than a primitive collider, it's best that we use a simplified mesh. The Mesh Collider holds a reference to the mesh it's using in the Mesh Slot on the component. By default Unity will use the mesh in the Mesh Filter if one is present. We can simply swap this out with a new simplified mesh of our choice. We have supplied a simplified mesh in the Models directory.
   a. Click on【 Project | Assets | Models | vehicle_playerShip_collider 】 to open the Model file and select the mesh asset
   b. Click on【 Hierarchy | Player 】and select the player
   c. Drag the mesh asset into the Mesh Slot【 Inspector | Mesh Collider | Mesh 】[7]
   d. Select【 Inspector | Mesh Render 】 to turn Mesh Render back on
   e. Select 【 Inspector | Mesh Collider | Is Trigger 】to make this is a trigger collider[8]
7. Lastly let's add a engine prefab to our ship to make it a little sizzle. The engines prefab consists of two particle system.
   a. Select 【 Project | Assets | Prefabs | VFX | Engines | engines_player.prefab 】

---

[5] In this way, it's easier to fine tune the shape

[6] There is the Box Collider and Sphere Collider as well. The box collider and sphere colliders are 2 other primitive colliders like the capsule collider, but there is a more complex collider called a Mesh Collider where we can supply the collision mesh ourselves.

[7] Now we can see the substantially simplified mesh being used as a collider. For the purpose of this game, we could use a capsule collider, but this game will be simply enough to absorb the large cost of the mesh collider. So let's leave it as it is.

[8] For this game we don't need to or want to detect for physics collisions, we simply need our collision to trigger an action.

b. Drag and drop this prefab on the player to add it to the player as a child game object

Readings

- Colliders, https://docs.unity3d.com/Manual/CollidersOverview.html
- Compound Collider Section, https://docs.unity3d.com/Manual/class-Rigidbody.html

1.3 Camera and lighting

In this section, we will set up the camera and lighting for our game.

1. Let's reset the camera's transform by using reset on the context sensitive gear menu.
    a. Rest the camera's transform
    b. Rotate the camera to face down by adjusting the rotation on the X axis to 90 degrees
    c. Set the position on the Y axis to 10
2. Let's set up the camera's component
    a. Set the camera's type[9]
        i. Click on `Inspector | Camera | Projection` and select Orthographic[10] as our projection model
    b. Adjust the Orthographic Size to change how much the camera sees
        i. Do any adjustment in the game view
        ii. Adjust the size to see the actual camera output full size
        iii. Set the Size to 10 exactly
3. Let's have the player shape start down at the bottom of the screen[11]
    a. Manipulate directly the camera's transform component
        i. Select the Camera by clicking on `Hierarchy | Main Camera`
    b. Move the camera's position along the Z axis
        i. Click on the Field title of Z
        ii. Drag back and forth until the ship looks good in the scene
        iii. Clean up the value of Z axis and set its value to 5
4. Let's set the camera's background
    a. Find the default background in the Clear Flags setting
        i. `Inspector | Camera | Clear Flags | Skybox`[12]
    b. Use a black background instead of the skybox
        i. Change the Clear Flag to Solid colour by clicking on `Inspector | Camera | Clear Flags | Skybox` to Solid colour[13]
    c. Let's make our background black
        i. Click on the colour box (`Inspector | Camera | Background |`) to bring up a colour picker
        ii. Make our background black by clicking on the bottom of colour picker
        iii. Close the colour picker

---

[9] Our game needs to feel like an upright arcade game. These did not have any perspective, so we will choose orthographic as our projection mode.

[10] Camera will render objects uniformly, with no sense of perspective. Further more information, https://docs.unity3d.com/Manual/CamerasOverview.html

[11] The alternative way is to adjust the player's position instead of the camera's position. But in this case we really want the player's game object to be at origin. We want this for 2 reasons. One, it just feels better for us. And two, it will make certain steps later on in this project easier to manage.

[12] By default, Unity 5 includes a skybox with every scene. The skybox can be found in the "Lighting" tab under "Window/Lighting"

[13] It is worth noting that if our Clear Flags is set to skybox and we have no skybox assigned, Clear Flags will use the background colour instead. This is why, even though we have skybox selected we see either be a procedural horizon or brown depending upon the direction of the Main Camera's rotation.

5. Having no contribution to the lighting from Ambient Light[14]
    a. Select the default “Directional Light” game object in the Hierarchy view and delete it
    b. Remove the skybox by selecting it and deleting it[15]
        i. Click on `Window | Lighting`
        ii. Find the Skybox in the Lighting window’s default tap:`Scene`.
        iii. Click on the gear button to the right (the circle with the dot in the middle) and it will open up an asset picker window
        iv. Slide up and choose None in the asset picker window
        v. Close the Select Material window and Lighting window
6. Let’s light our scene
    a. Create a new Directional light
        i. Click on `Create | Light | Directional Light` in the hierarchy view
    b. Rename this game object Main Light
        i. Select this game object in the Hierarchy view
        ii. Click on it and enter edit status
        iii. Rename it and hit return or enter key
    c. Reset this light’s position to be at origin
        i. Select the Main Light game object in the hierarchy view
        ii. Click on gear menu (`Inspector | Transform`)and select Reset Position
    d. Set this light’s rotation as well
        i. Click on gear menu and select Reset Rotation
        ii. X =20 and Y = -115
    e. Increase the intensity, so that the main light is to be felt like the light from a nearby sun
        i. Click on the field of intensity and drag left and right to adjust the intensity until it gives you a nice hot feeling to the right side of the ship
        ii. Try a value between 1.5 and 2.0[16]
7. To light the other side of the ship we need fill light to fill in the shadows on the far side
    a. Duplicate the main light
        i. Make sure the main light is selected
        ii. Use Edit-Duplicate or use the hot-key combination(Cmd + D/Mac, Ctr + D/Win)
    b. Rename the duplicate game object Fill Light
    c. Set the rotation on the fill light
        i. Select the Fill light in the Hierarchy view
        ii. Click on gear menu (`Inspector | Transform`)and select Reset Rotation
        iii. Grab the fill light and rotate it around on the Y axis to compliment the key light.
        iv. It feels good to set Y equal 125
    d. Reduce the intensity of the Fill Light

[14] Ambient Light uses the Skybox to set its colour values. To make the Ambient Light have no value can by done by either leaving the Ambient Source as Skybox and making sure Skybox is None, or by setting the Ambient Source to Colour and making sure the colour is Black.

[15] The other way to do so is simply select the skybox field and use the delete or backspace key to remove it.

[16] This lighting might seem realistic for some deep space environment but the other side of the ship is far dark for this game. To light this side of the ship we need another light.

        i. Try a value of 1.0 for the Fill Light, then adjust to taste
    e. Change the colour as well
        i. Set R to 128
        ii. Set G to 192
        iii. Set B to 192
        iv. By toggling the Fill Light, we can see how it's lighting that side of the ship
    f. Lastly let's tilt Fill Light down a little in to the scene
        i. Clean up the rotation on the X axis
        ii. Set it to 5
8. Let's add the rim light
    a. Duplicate the fill light
    b. Rename it Rim Light
    c. Turn off the Fill light to focus the Rim Light focus
        i. Select the Fill Light in the Hierarchy view
        ii. Unselect the Fill Light in the Inspector view on the Rim Light right
    d. Reset the transform to move the rotation values
        i. Select the Rim Light in the Hierarchy view
        ii. Click on gear menu (`Inspector | Transform`)and select Reset
    e. Change the colour to pure white
    f. Set the rotation on the Rim Light
        i. X = -15
        ii. Y = 65
    g. Lastly let's drop the intensity down to 0.5
9. Let's organise our scene and keep our lights together
    a. Add a new game object
        i. Click on `Create | Create Empty`
    b. Rename it Lighting
    c. Reset this game object's transform[17]
    d. Drag our lights (Main Light, Fill Light, and Rim Light) into it
    e. Move our lighting parent game object up on the Y axis by 100 units[18]

Readings

- Camera, https://docs.unity3d.com/Manual/CamerasOverview.html
- Camera Inspector, https://docs.unity3d.com/Manual/class-Camera.html

- Lights, https://docs.unity3d.com/Manual/Lights.html
- The Light inspector, https://docs.unity3d.com/Manual/class-Light.html
- Lighting Window, https://docs.unity3d.com/Manual/GlobalIllumination.html
- How do I Make a Skybox, https://docs.unity3d.com/Manual/HOWTO-UseSkybox.html

- Hierarchy, https://docs.unity3d.com/Manual/Hierarchy.html

[17] When we are using empty game objects to organise our hierarchy, those game objects should be at origin with no rotation and scale of 1.

[18] We can move these lights like this, because they are directional lights and directional lights like the entire scene based on their transformed rotation not their position.

1.4 Adding a background

Let's set background to our game.

1. Deactivating our player game object
    a. Select player game object in the Hierarchy view
    b. Deactivate it by unselecting player in the Inspector view
2. Create a Quad to hold our background image
    a. Click on `Create | 3D Object | Quad` in the Hierarchy view
    b. Rename it Background
    c. Reset this game object's transform to make sure it at the origin
        i. Select the Background game object in the Hierarchy view
        ii. Find the Transform component in the Inspector view
        iii. Click on the gear menu on the help button right
        iv. Select Reset
    d. Change the background's transform rotation along the X axis
        i. Set this value to 90 degrees
    e. Remove the Mesh Collider component of the background[19]
        i. Find the Mesh Collider component in the Inspector view
        ii. Click on the gear menu on the help button right
        iii. Select Remove Component
3. Let's add textures to this background
    a. Find the texture
        i. Click on `Assets | Textures` in the Project view
        ii. Find an image named title_nebula_green_dff.tif
        iii. Select this image
    b. Drag and drop it on background in the scene
    c. Use Famed Selected to focus the scene view camera
4. Let's reset the scale of the background
    a. Set x to 17 and y to 34
5. We don't need any lighting at all for our background. Let's change shader[20]
    a. Select the background in the Hierarchy view
    b. Find the tie_nebula-green-dff
    c. Click on the menu on the shader right
    d. Select `Unlit | Texture`
6. The background should behind the game player
    a. Drag the background and adjust along the Y axis
        i. Slide it into a good position by hand or
        ii. Simply set the transform position on the Y axis to -10

Readings

- Lights, http://docs.unity3d.com/Documentation/Manual/Lights.html
- Lights Components, http://docs.unity3d.com/Documentation/Components/class-Light.html
- Materials and Shaders, http://docs.unity3d.com/Documentation/Manual/Materials.html
- Built-in Shader Guide, http://docs.unity3d.com/Documentation/Components/Built-inShaderGuide.html

[19] We don't need this component, so we safely remove it.
[20] Completing this, our background is independent of our lighting system.

- Shader Reference(Advanced), http://docs.unity3d.com/Documentation/Components/SL-Reference.html

1.5 Moving the player

Let's get our player ship moving under our control.

1. Add a new folder to our assets folder
    a. Select Assets folder in the Project view
    b. Use the create menu in the Project view and select folder
    c. Change its name to "Scripts" and hit "enter" key to accept its new name
2. Let's create and add a new script to our player ship
    a. Select the player in the Hierarchy view
    b. Click on `Add Component` button and select new script in the Inspector view
    c. Set the name of the script to PlayerController[21]
    d. Click on `Create and Add` or simply hit return or enter to create and add this script
    e. Drag and drop this script to Scripts folder in the Project view
    f. Open the Scripts folder to view this script and open this script for editing
        i. Select the script and choose open button in the Inspector view
        ii. Or double click on this script
        iii. Or select the player game object in the Hierarchy view and then find the Player Controller component in the Inspector view and click on the gear menu and select Edit Script
3. We will use FixedUpdate[22] function, so that we will be moving our player ship using Physics
    a. Remove all the sample code
    b. Write some code inside PlayerController class

```
public float speed;
private Rigidbody rb;

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


void FixedUpdate(){

        float  moveHorizontal = Input.GetAxis("Horizontal");
        float  moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
        rb.velocity = movement * speed;

}
```

    c. Save this script and return to Unity
    d. Check the Console to make sure there is no error and that everything we wrote has complied properly[23]
    e. Test our code
        i. Save the scene
        ii. Enter play mode

[21] Script names should start with a capital letter and the camel casing of the name makes the name easier to read. It is best to follow the convention of capitalization here.

[22] FixedUpdate function will be called automatically by Unity just before each fixed physic step. All the code we put inside the FixedUpdate function will be executed once per physic step.

[23] If there were an error, we would also see this error in the footer as our recent error or message will always be shown in the footer

iii. Our player moved slowly[24], so we assign a value e.g. 10 to the speed variable[25] in our script

4. We need to constrain the ship within the game area
   a. Leave play mode and return to our PlayerController script
   b. Write some code within FixedUpdate function

```
rb.position = new Vector3 (Mathf.Clamp(rb.position.x,  xMin,  xMax) ,
                           0.0f,
                           Mathf.Clamp(rb.position.z,  zMin,  zMax));
```

   c. Declare Min and Max values at the top of our scripts

```
public float xMin, xMax, zMin, zMax;
```

   d. Put this code into a separate class of their own to clean up it and make it reusable

```
public class Boundary{
    public float xMin, xMax, zMin, zMax;
}
```

   e. Declare a boundary variable inside the PlayerController class

```
private Rigidbody rb;
public Boundary boundary[26];
```

   f. Next update our Clamp code to reflect changes we just made

```
rb.position = new Vector3 (Mathf.Clamp(rb.position.x, boundary.xMin, boundary.xMax) ,
                           0.0f,
                           Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax));
```

   g. Save this code and switch to Unity[27]
   h. Let's switch back to the script to serialize our new class

```
[System.Serializable]
public class Boundary{
    public float xMin, xMax, zMin, zMax;
}
```

5. Let's set the boundary values(the player should be clamped inside the game area)
   a. We can see boundary, if we Look back at our PlayerController
      i. Turn down the gizmo, we can see our properties in an easy to use but very tidy container.
      ii.
   b. Make sure we have the player ship selected in the Hierarchy view
   c. Drag the ship to the edge of the game area and note the values on the X axis
      i. Set xMin to -6
      ii. Set xMax to 6
   d. Do the same thing along the Y axis
      i. Set zMin to -2
      ii. Set zMax to 7[28]
   e. Let's reset the player's transform and enter play mode and() test
6. Let's add some bank or tilt to the player ship when we move from side to side along the X axis

---

[24] This is because input.GetAxis only returns a number between 0 and 1, so our ship's movement was no more than one unit per second.

[25] We can see the speed variable. We can set it and we can change its value in the Inspector view, because we declare it a public variable.

[26] Note the capitalization: Boundary with a capital B is a type as defined by a class name and boundary with a lower b is the name for our reference in the same way that speed is the name for our float variable.

[27] We can't see our new updated properties at all. This is because the new class that we have created is unknown to unity and is therefore not serialized. Serializing is a way of storing and transferring information. Serialization is a complicated issue at this point just understand that unity needs to have properties serialized to view them in the inspector.

[28] We don't really want to our game player access to the entire upper game area. We will need to give the hazards in our game some room to be able to enter the game area, so let's back off a little bit.

a. Switch to the script
b. Create a new variable to hold our tilt value

```
public float tilt;
public float speed;
```

c. Still in FixedUpdate function write

```
rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
```

d. Save the script and return to unity
e. Let's set the tilt value
    i. Adjust the tilt property, until feel good
    ii. Set tilt to 5

Readings

- Quaternion, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion.html
- Quaternion Euler, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion.Euler.html
- Quaternion eulerAngles, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion-eulerAngles.html
- Editor Preferences, http://docs.unity3d.com/Documentation/Manual/Preferences.html
- Visual Studio Integration, http://docs.unity3d.com/Documentation/Manual/VisualStudioIntegration.html

PlayerController.cs

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

[System.Serializable]
public class Boundary{
    public float xMin, xMax, zMin, zMax;
}

public class PlayerController : MonoBehaviour {

    public float tilt;
    public float speed;

    private Rigidbody rb;
    public Boundary boundary;

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

    void FixedUpdate(){

        float  moveHorizontal = Input.GetAxis("Horizontal");
        float  moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
        rb.velocity = movement * speed;

        rb.position = new Vector3 (Mathf.Clamp(rb.position.x, boundary.xMin, bo
undary.xMax) ,0.0f, Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax));
```

```
        rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
    }

}
```

1.6 Creating shots

Let's create shots for our game to shoot

1. First let's deactivate the player
2. Create a new empty game object for our shots as the parent game object
    a. Use the Shift + Command + N on the Mac or Shift + Ctrl +N on the Windows
    b. Rename the empty game object Bolt
    c. Reset the game object's transform to make sure it is at origin
    d. Create a Quad to hold our Visual Effect Image just like we did for our background
        i. Click on `Create | 3D Object | Quad`
        ii. Rename the quad VFX
        iii. Reset it's transform
        iv. Add the VFX game object as a child of Bolt
        v. Change the transform rotation x to 90 to rotate the quad to face up world towards the view of camera
3. Let's make the shot look like a laser bolt
    a. Find artwork for our laser bolt
        i. Open the Textures folder in the Project view
        ii. Find fx_lazer_orange_dff in this folder
    b. Create and assign material[29]
        i. Select the Materials folder in the Project view
        ii. Click on `Create | Materials` with the material selected
        iii. Rename this material 1_fx_bolt_orange
        iv. Click on the context gear menu on Albedo right in the Inspector view to open Select Texture window and select fx_lazer_orange_dff
        v. Or Find fx_lazer_orange_dff image in the Textures folder and drag it onto fx_bolt_orange material and drop it into the Albedo Fields
        vi. Find the 1_fx_bolt_orange material in the Materials folder
        vii. Drag this material onto the VFX/Quad
    c. Change the shader on this material to give us a strong and hot laser bolt
        i. Click on the shader menu on the context gear menu down
        ii. Select `Particles | Additive` or try `Mobile | Particles | Additive`[30]
4. We will be moving these shots with physics and more importantly we will be moving a collider in our game. Both require the rigidbody component.
    a. Select the Bolt game object in the Hierarchy view
    b. Click on `Add Component` and select `Physics | Rigidbody` to add a Rigidbody component to both game object
    c. Deselect Use Gravity[31]
    d. Remove the mesh collider that sticks out so far to the sides beyond the images of laser bolt
        i. Select VFX game object

[29] The other way to do so is to select the texture in the Textures folder and then drag and drop it onto the VFX game object.

[30] In many cases mobile shaders can be used effectively in none mobile games. In general the mobile shaders will more efficient with our game resource budget but in some cases may sacrifice either quality or control. The main control that we will lose by using this mobile shader is the ability to change tint colour which we don't need on our laser bolt with our

[31] As we don't want our shots falling down off the game plane

ii. Find Mesh Collider in the Inspector view to look at it's edge
    1. Select Convex
    2. Unselect Mesh Render
iii. On the mesh Collider component use the context gear menu and click on Remove Component to remove this mesh collider component from the VFX game object

5. let's add the collider component we want for our game to participate in collisions
    a. Click on the parent bold game object
    b. Click on Add Component and select `Physics | Capsule Collider`
    c. Make sure the Bolt game object selected and find Capsule Collider in the Inspector view
    d. Change its direction to Z-Axis by clicking on the gizmo button on Direction right
    e. Adjust Height and Radius of the capsule collider to fit the Bolt
        i. Set Radius to 0.03
        ii. Set Height to 5.5
    f. Select Is Trigger to make this collider a trigger collider
        i. Select Bolt game object and look at Inspector view
        ii. Find Is Trigger within Capsule Collider component
        iii. Click on Is Trigger
6. We now need to write custom logic with the bolt game object
    a. With the Bolt game object selected click on Add Component and choose New Script
    b. Name this script Mover
    c. Drag and drop the Mover script file into the Scripts folder in the Project view
    d. Open this script for editing
    e. Remove the sample code
    f. Write some code within Mover class

```
public float speed;
private Rigidbody rb;

void Start(){
    rb = GetComponent<Rigidbody> ();
    rb.velocity = transform.forward * speed;
}
```

7. Let's same the game object as a prefab, so that our player can shoot many copies or clones of this shot
    a. Drag the bolt game object from the Hierarchy view into the prefabs folder in assets
    b. Set the speed value for our bolt
        i. Select Bolt game object in the Hierarchy view
        ii. Find Mover Component in the Inspector view
        iii. Set speed to 20[32]
    c. Delete our working instance of bolt from the scene, because we only want one of instances of the bolt game object in our scene when our player fires its weapon
        i. Select the Bolt game object in the Hierarchy view
        ii. Delete it
    d. Save the scene
        i. Click on `File | Edit`
        ii. `Or Cmd + S on Mac or Ctrl + s on Windows`
    e. Enter play mode to test

[32] The shots need to go faster than the ship

i. Turn off maximize on play if it's on
ii. Click play button
iii. Test the bolt as we don't have any shooting code, so we can simply drag copies of the prefab into the hierarchy window while the game is running to have a look if they work as expected

Readings

- Material and Shade, http://docs.unity3d.com/Documentation/Manual/Materials.html
- Material, http://docs.unity3d.com/Documentation/Components/class-Material.html
- Build-in Shader Guide, http://docs.unity3d.com/Documentation/Components/Built-inShaderGuide.html
- Shader Reference, http://docs.unity3d.com/Documentation/Components/SL-Reference.html
- Rigidbody, http://docs.unity3d.com/Documentation/Components/class-Rigidbody.html
- Capsule Collider, http://docs.unity3d.com/Documentation/Components/class-CapsuleCollider.html

Mover.cs

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

public class Mover : MonoBehaviour {

    public float speed;
    private Rigidbody rb;

    void Start(){
        rb = GetComponent<Rigidbody> ();
        rb.velocity = transform.forward * speed;
    }
}
```

1.7 Shooting shots

Let's enable our player to shoot them

1. We will need our player game object, so select player and reactivate the player game object
2. We need to edit our PlayerController script
    a. With the player selected use the context sensitive gear menu on the PlayerController component and select Edit Script
    b. Instantiate a copy or clone of the shot prefab when we hit a button or a click mouse[33] during our game play

```
void Update () {

     Instantiate(object, position, rotation);
}
```

3. Create a new empty game object , and rename it Shot Spawn
    a. drag shot spawn on to our player game object and drop it as child[34].
    b. Open the player game object family to check whether you can see the shot spawn in the hierarchy as a child game object
4. We can position the shot spawn as it's a child of the player ship[35]. We want to instantiate shots in front of the player ship
    a. Drag the shot spawn out along its Z Axis until it's in front of the ship
    b. Set z to 1.25
    c. Drag an instance of our bolt prefab into the scene as the child of shot spawn to make a test, and make sure the instance's transform is at origin
    d. Delete our test instance, because we don't want to have a shot in our scene when we start the game
5. Edit and the script

```
private Rigidbody rb;
public Boundary boundary;

public GameObject shot;
public Transform shotSpawn;
public float fireRate;

private float nextFire;

void Update(){

    if(Input.GetButton("Fire1") && Time.time > nextFire){
        nextFire = Time.time + fireRate;
        Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
    }
}
```

6. Save this script and return to Unity
    a. Click on the player game object, so we can inspect it
    b. Open the prefabs folder and drag the Bolt prefab onto Shot reference on the Player Controller component

---

[33] Simply getting an input from a button doesn't require physics and we don't want to wait for a fixed update to fire our weapon. So let's put our code in a update function.

[34] The transform component contains a position as Vector3 and rotation as a quaternion. The transform also includes a scale as a Vector3 which we can ignore for this game. In the inspector the quaternion for rotation is simplified as a quaternion Euler. For more information, visit:
So how do we use this to instantiate a shot? We can use this empty game object's transform as spawn point in our game and his spawn point should move with our player ship. We can think of this is some sort of virtual hard. to attach our weapons to.

[35] Shot spawn's position will be relative to the player ship.

c. Grab the Shot Spawn game object from the Hierarchy into the Shot Spawn reference on the Player Controller component
d. Lastly set our Fire Rate four times a second sounds good and fast
    i. fireRate = 0.25

7. Let's save and play
    a. Our player can now shoot shots
    b. But we have created a problem. We are filling our scene with shot game object clones all flying off to infinity on the Z axis.
    c. In the next section, we will create a boundary to clean up any game objects that leave the game area

Readings

- Input, http://docs.unity3d.com/Documentation/Manual/Input.html
- Input Manager, http://docs.unity3d.com/Documentation/Components/class-InputManager.html
- Time, http://docs.unity3d.com/Documentation/ScriptReference/Time.html
- Input, http://docs.unity3d.com/Documentation/ScriptReference/Input.html
- Input GetButton, http://docs.unity3d.com/Documentation/ScriptReference/Input.GetButton.html
- Instantiating Prefabs at runtime, http://docs.unity3d.com/Documentation/Manual/InstantiatingPrefabs.html
- Instantiate, http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html
- Quaternion, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion.html
- Quaternion Euler, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion.Euler.html
- Quaternion eulerAngles, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion-eulerAngles.html

PlayerController.cs

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

[System.Serializable]
public class Boundary{
    public float xMin, xMax, zMin, zMax;
}

public class PlayerController : MonoBehaviour {

    public float speed;
    public float tilt;

    private Rigidbody rb;
    public Boundary boundary;

    public GameObject shot;
    public Transform shotSpawn;
```

```
    public float fireRate;

    private float nextFire;

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

    void Update(){

        if(Input.GetButton("Fire1") && Time.time > nextFire){
            nextFire = Time.time + fireRate;
            Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
        }
    }

    void FixedUpdate(){

        float  moveHorizontal = Input.GetAxis("Horizontal");
        float  moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
        rb.velocity = movement * speed;

        rb.position = new Vector3 (Mathf.Clamp(rb.position.x, boundary.xMin, bo
undary.xMax) ,0.0f, Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax));

        rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
    }

}
```

2. Boundaries, Hazards and Enemies

2.1 Boundary

1. We will Create a box around our game area
    a. Create a cube game object in the Hierarchy view by using Create menu and select 3D | Cube
    b. Rename it Boundary
    c. Reset its transform to make sure it is at origin
    d. Make this box to trigger a new action when our shots leave this box[36]
        i. Find box collider with Boundary game object selected
        ii. Select Is Trigger
2. We want to place the Boundary game object evenly around the game area
    a. The centre of this game area is defined by Main Camera
        i. Select the Main Camera
        ii. Find Main Camera's transform component in the Inspector view
        iii. The camera's transform on the x plane is 0, 5
    b. Change the transform of the boundary game object the same as main camera's
        i. Select the Boundary game object
        ii. Set the position along Z axis to 5
    c. Change the scale until the boundary is around the scene
3. Let's change the Cubes' scale by using the transform's scale
    a. Set the transform scale X to 15
    b. Set the transform scale Z to 20
    c. Turn off Mesh Renderer
        i. Find the Mesh Renderer component with the boundary selected
        ii. Unselect Mesh Renderer
4. We now need our boundary to do something
    a. Need a script attached
        i. Click on add component with Boundary selected
        ii. Select new script
        iii. Rename this script DestroyByBroundary
        iv. Add this script to boundary
    b. Let's file it in the script folder
        i. Select assets in the project view to see our script in the root level
        ii. Drag and drop it into Scripts folder
    c. Open the script for editting
        i. Open the Scripts folder
        ii. Select DestroyByBoundary
        iii. Open it for editing our boundary game object
        iv. Remove the sample code
5. Our boundary game object's behaviour will be driven by the box collider and that box collider is a trigger. To find out how to script to a trigger collider,
    a. We can search trigger in the document
        i. Type trigger
        ii. Select it
        iii. Use the hot combination key: Cmd + ' on Mac or Ctrl + ' on Windows

[36] We want this box to be a trigger collider

b. We want to destroy the shots as they leave the box collider's trigger value
    i. Let's look at Collider.OnTriggerExit
    ii. The description tells us that OnTriggerExit is called when the Other collider has stopped touching the trigger
    iii. Copy this code and paste it into our code[37]
    iv. Save the script and return to Unity
6. Let's save the scene and test, but before that we need:
    a. Select the boundary game object
    b. Remove the Mesh Render component and Mesh Filter component[38]
    c. Save the scene and test

Readings

- Box Collider, http://docs.unity3d.com/Documentation/Components/class-BoxCollider.html
- Collider, http://docs.unity3d.com/Documentation/ScriptReference/Collider.html
- Collider.OnTriggerExit, http://docs.unity3d.com/Documentation/ScriptReference/Collider.OnTriggerExit.html

DestoryByBoudary.cs

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

public class DestroyByBoundary : MonoBehaviour {

    void OnTriggerExit(Collider other) {
        // Destroy everything that leaves the trigger
        Destroy(other.gameObject);
    }
}
```

[37] Conveniently the sample code does exactly what we want. When the other collider leaves the boundary's trigger volume, we want to destroy the other collider's game object.
[38] Our boundary is set up and in position, so we don't need this component

2.2 Creating hazards

1. Let's create a new game object and rename it Asteroid
    a. Click on Create | Create Empty in the Hierarchy view
    b. Rename it Asteroid
    c. Reset it's transform to origin
    d. Move it up along Z Axis to about 8
2. Let's get some artwork for our Asteroid game object
    a. Open the Models folder in the Assets
    b. Select the first asteroid model and drag it onto the Asteroid parent game object
    c. Make sure asteroid model be a child of asteroid game object
    d. Reset the Model's transform
        i. Click on the button on Asteroid game object left in the Hierarchy view
        ii. Select the Model
        iii. Reset its transform by using the gear menu in the Inspector view
3. Let's set up Asteroid game object's components and logic
    a. Select the parent Asteroid game object and use Add Component button
    b. Add Physics | Rigidbody
        i. Deselect Use Gravity
    c. Add Physics | Capsule Collider
        i. Change the Radius and Height of the collider to more accurately math our model by
        ii. Using the properties on the component
        iii. Changing the shape of capsule collider directly in the scene view
    d. Save the scene and enter play mode
4. Let's give the Asteroid game object some life and make it tumble
    a. Add a new script to Asteroid game object
        i. Use the Add Component button with the asteroid game object selected to create a new script
        ii. Rename it RandomRotator and accept the changes to add this script to asteroid
        iii. Select the Asset folder and file the new script into the Scripts folder
        iv. Open the Script fold and select the RandomRotator
        v. Open this script for editing and remove the sample code
    b. Write code

```
public float tumble;
private Rigidbody rb;

void Start(){
    rb = GetComponent<Rigidbody> ();
    rb.angularVelocity = Random.insideUnitSphere * tumble;
}
```

    c. Save the script and return to Unity
        i. Select the asetoird game object
        ii. Find the RandomRotator component in the Inspector view
        iii. Set Tumble value to 5
    d. Save the scene and enter the play mode to test
        i. We leave it for a short period of time, the asteroid slows to a stop[39]

[39] This is due to a default value on the rigidbody component. The rigibody has two parameter to simulate resistance like air friction, and these are Drag and Angular Drag. Angular drag by default has a small but working value.

ii. Exit play mode
iii. Set angular drag's value to 0

5. We'll need to write code for our two colliders to have any effect
    a. Add a new script to Asteroid game object
        i. With asteroid selected use the add component button to create a new script
        ii. Rename it DestroyBycontact and accept the changes to add this script to asteroid
        iii. Select the Asset folder and file the new script into the Scripts folder
        iv. Open this script for editing and remove the sample code
    b. Type trigger again and search the documentation
        i. Select the trigger and use the hot key to search it
        ii. In this case, we want to destroy the asteroid when the bolt first touches it
        iii. Let's select Collider.OnTriggerEnter and copy it
        iv. Past this code into our script
    c. Edit the code to destroy both laser bolt and asteroid[40]

```
void OnTriggerExit(Collider other) {
    Destroy(other.gameObject);
    Destroy(gameObject);
}
```

    d. Save the script and Enter to play mode
    e. We can see asteroid and boundary game object, but they vanish in the play mode
6. We have a bug. Let's do some simple debugging
    a. In OnTriggerEnter, let's write

```
Debug.Log(other.name);
```

    b. One way of identifying object is with the tag. let's tag our boundary
        i. Select the boundary, in the header of the game object is its tag
        ii. Click on the tag drop down and select add tag
        iii. Let's add a new tag Boundary
        iv. Loo back at our boundary game object
        v. Click on the tag drop down and select boundary to successfully tag this game object as Boundary
    c.
7. Rewrite the destruction code in the DestroyByContact
    a. Add some code

```
if(other.tag =="Boundary"){
    return;
}
```

    b. Save this code and return to unity
    c. Enter play mode and test

Readings

- Random, http://docs.unity3d.com/Documentation/ScriptReference/Random.html

[40] One quick note about destroy is that destroy doesn't immediately destroy the object listed in the parentheses. It marks the object to be destroyed and all of the marked objects are destroyed at the end of the frame. So we can put these destroy lines in any order. It seems wrong if we destroy this game object before we call destroy on the other one. But it doesn't matter what order we mark our objects to be destroyed, as long as they are marked in the same frame.

- Random.value, http://docs.unity3d.com/Documentation/ScriptReference/Random-value.html
- Random.insideUnitSphere, http://docs.unity3d.com/Documentation/ScriptReference/Random-insideUnitSphere.html
- Adding Random Gameplay Elements, http://docs.unity3d.com/Documentation/Manual/RandomNumbers.html
- Rigidbody, http://docs.unity3d.com/Documentation/Components/class-Rigidbody.html
- Rigidbody, http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody.html
- Rigidbody.drag, http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody-drag.html
- Rigidbody.angualDrag, http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody-angularDrag.html
- Capsule Collider, http://docs.unity3d.com/Documentation/Components/class-CapsuleCollider.html
- Collider, http://docs.unity3d.com/Documentation/ScriptReference/Collider.html
- Collider.OnTriggerEnter, http://docs.unity3d.com/Documentation/ScriptReference/Collider.OnTriggerEnter.html
- Tags and Layers, http://docs.unity3d.com/Documentation/Components/class-TagManager.html
- GameObject.tag, http://docs.unity3d.com/Documentation/ScriptReference/GameObject-tag.html

RandomRotator.cs

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

public class RandomRotator : MonoBehaviour {

    public float tumble;
    private Rigidbody rb;

    void Start(){
        rb = GetComponent<Rigidbody> ();
        rb.angularVelocity = Random.insideUnitSphere * tumble;
    }

}
```

DestroyByContact.cs

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

public class DestroyByContact : MonoBehaviour {

    void OnTriggerEnter(Collider other) {
        if(other.tag =="Boundary"){
            return;
```

```
        }
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

2.3 Explosions

Let's add explosion to brighten things up.

1. The first thing we need to do is to add a reference to our explosion visual effect.
    a. Let's open DestroyByContact script
    b. Add some code

```
public GameObject explosion;

void OnTriggerEnter(Collider other) {
    if(other.tag =="Boundary"){
        return;
    }
    Instantiate(explosion, transform.position, transform.rotation);
    Destroy(other.gameObject);
    Destroy(gameObject);
 }
```

    c. Save the script and return to Unity
    d. Create a reference to explosion variable on DestroyByContact script
        i. Find the explosion_asteroid by clicking on Assets | VFX | Explosions in the project view
        ii. Select the Asteroid game object in the Hierarchy view
        iii. Find the DestoryByContact component in the inspector view
        iv. Drag and drop explosion asteroid onto explosion shot on DestroyByContact script
    e. Save the scene and enter play mode
2. Let's include a player explosion when the player and hazard collider
    a. Return to our DestroyByContact script
    b. Let's add code

```
public GameObject explosion;
public GameObject playerExplosion;

void OnTriggerEnter(Collider other)
{
    if (other.tag == "Boundary")
    {
        return;
    }

    Instantiate (explosion, transform.position, transform.rotation);
    if (other.tag == "Player"){
        Instantiate (playerExplosion, other.transform.position, other.transform.rot
ation);
    }

    Destroy(other.gameObject);
    Destroy(gameObject);
}
```

    c. Save this script and return to Unity
    d. Our player needs a tag
        i. Select the Player game object
        ii. Click on tag drop-down menu and select the pre-made tag player[41]
    e. Create a reference to explosion variable on DestroyByContact script
        i. Select the Asteroid game object
        ii. Drag the explosion player on to the player explosion slot on the DestroyByContact component
        iii. Save the scene and play
3. The last improvement we need to make on the asteroid is to get it moving towards the player

[41] There is no need to create a customised tag in this case

a. With the asteroid selected open the Scripts folder
b. Drag the mover script on to the asteroid game object in the inspector
c. Set the speed value to -5[42]
d. Save the scene and enter play mode and test

4. We need to save the Asteroid game object as a prefab
    a. Drag the asteroid game object from the hierarchy view on to the prefab folder in assets
    b. Make sure you have done it correctly
    c. Lastly delete the instance of the asteroid from our scene
    d. Save the scene

Readings

- Instantiate, http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html
- Tags and Layers, http://docs.unity3d.com/Documentation/Components/class-TagManager.html
- GameObject, http://docs.unity3d.com/Documentation/ScriptReference/GameObject-tag.html

DestroyByContact.cs

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

public class DestroyByContact : MonoBehaviour
{
    public GameObject explosion;
    public GameObject playerExplosion;

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }

        Instantiate (explosion, transform.position, transform.rotation);
        if (other.tag == "Player"){
            Instantiate (playerExplosion, other.transform.position, other.trans
form.rotation);
        }

        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

[42] This negative speed will bring the asteroid down the screen towards the player.

2.4 Game Controller

We need to create a game controller to run our game, to spawn our hazard, keep track of and display our score and when our player is destroyed end our game.

1. Create a new game object to hold our game controller logic
    a. Create a new empty game object
    b. Rename it Game Controller
    c. Reset its transform
    d. Using the tag drop-down menu in the header of GameController game object select the pre-made tag GameController
2. With the game object set up, let's write our game logic
    a. With the game controller selected use the add component to create a new script
        i. Rename it Game Controller
        ii. Accept the changes to add this new script to game controller game object
        iii. Select Assets and file the GameController script into the Scripts folder
        iv. Open the Scripts folder and open the GameController script for editing
        v. Remove the sample code
    b. The first thing we will need in the script is a public game object reference to our hazard[43]
    c. Let's write a new function to spawn our waves of hazards
    d. Let's call SpawnWaves function from Start function[44].
    e. What do we want SpawnWaves functions to do? We want to instantiate our hazard at spawnPosition with a spawnRotation applied.
    f. We have three parameters for instantiating and we need a value for each of them
        i. Our object hazard has already been defined as a public variable
        ii. Our spawnPosition is a Vector 3 value
        iii. Our spawnRotation is a quaternion value
    g. The code we write above is below

```
public GameObject hazard;

void Start(){
    SpawnWaves ();
}

void SpawnWaves(){
    Vector3 spawnPosition = new Vector3();
    Quaternion spawnRotation = new Quaternion();

    Instantiate (hazard, spawnPosition, spawnRotation);
}
```

3. Let's look at how we can assign working values to these variables
    a. Save the script and return to Unity
    b. Hazard is a public variable. We can set hazard's value directly in the inspector
        i. Select Game Controller game object
        ii. Find the Game Controller component in the inspector
        iii. Find our asteroid prefab in the prefabs folder
        iv. Drag the asteroid prefab onto the game controller component and drop it onto the hazard slot to create the reference

[43] Our game controller will perform several different tasks. The primary task, however, will be spawning the hazards in our game.

[44] There are some functions that they are called automatically by Unity. But most of the functions that we write we will need to call ourselves or they won't be executed. We want our SpawnWaves function to work for most of

c. We want to be able to set our spawnPostion in the editor as well
    i. Return to the game controller script
    ii. For our spawnPostion let's define a public vector 3 value called spawnValues
    iii. Save this script and return to Unity
d. Let's set spawnvalues' value
    i. Drag asteroid prefab into Hierarchy view temporarily to help visualize this
    ii. Set y to 0
    iii. Set z to 15
    iv. We couldn't use spawn position directly for X[45].

4. Let's return to our script. We will use our spawnvalues to set a random position
    a. Find out more in the document about random
        i. Select the random
        ii. Use hot key: CMD + ' on Mac or Ctrl + ' on Win
    b. Rewrite the line to initialize spawnPosition

```
Vector3 spawnPosition = new Vector3(Random.Range(-spanwValues.x, spawnValues.x),
                                    spawnValues.y, spawnValues.z);
```

    c. Save the script and return to Unity
    d. Move asteroid along x axis to find the edge of our game
        i. It looks like about -8 and 8
    e. Set spanwValues X to 8
5. Last we need to set our spawnRoatation
    a. Back to our game controller script
    b. Let's find our more in the document about quaternion
    c. We will look at quaternion property called quaternion.identity
        i. Quaternion.identiy corresponds with no-rotation of the quaternion
        ii. We will instantiate our hazard with no rotation at all
        iii. Copy quaternion.identity to our script
    d. Rewrite the spawnRotation line

```
Quaternion spawnRotation = Quaternion.identity;
```

6. Save this script and return to Unity to test
    a. Remove our temporary asteroid
    b. Save the scene
    c. Enter play mode and we will get a tumbling asteroid from a random point left to right
    d. Let's enter and exit play mode a few times we can see that our spawn points are random and each new asteroid is spawned a new random location

Readings

- Tags and Layer, http://docs.unity3d.com/Documentation/Components/class-TagManager.html
- GameObject.tag, http://docs.unity3d.com/Documentation/ScriptReference/GameObject-tag.html
- Instantiate, http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html

[45] What we really want on the x axis is a random point. That' is to say a new random value for each new hazard we spawn

- Random, http://docs.unity3d.com/Documentation/ScriptReference/Random.html
- Random Range, http://docs.unity3d.com/Documentation/ScriptReference/Random.Range.html
- Quaternion.identity, http://docs.unity3d.com/Documentation/ScriptReference/Quaternion-identity.html
- MonoBehaviour, http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html

GameController.cs

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

public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;

    void Start(){
        SpawnWaves ();
    }

    void SpawnWaves(){
        Vector3 spawnPosition = new Vector3(Random.Range(-
spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
        Quaternion spawnRotation = Quaternion.identity;

        Instantiate (hazard, spawnPosition, spawnRotation);
    }
}
```

2.5 Spawning waves

Let's spawn wave after wave of hazard instead of one.

1. Open the GameController script and put the block of code into a loop
    a. Open the GameController script
    b. First let's create a public int hazardCount variable to hold the number of times we cycle through the loop[46]

```
public GameObject hazard;
public Vector3 spawnValues;
public int hazardCount
```

    c. Let's put our instantiate code into a loop
        i. Select the code we have written
        ii. click right button of mouse and use Indent selection
        iii. Surround the code block with brackets
        iv. Write the for followed by a set of parentheses containing two semicolons
    d. In the declaration of the for loop, we need to initialize a count and show the condition or stay how long we want to stay in the loop, and then increment our count

```
for (int i = 0; i < hazardCount; i++) {
          Vector3 spawnPosition = new Vector3 (
            Random.Range (-spawnValues.x, spawnValues.x),
            spawnValues.y, spawnValues.z);
    Quaternion spawnRotation = Quaternion.identity;

    Instantiate (hazard, spawnPosition, spawnRotation);
}
```

2. Save this script and switch back to Unity to test
    a. Select the game controller in the hierarchy view
    b. Chang the hazardCount to 10
    c. Let's enter and exit play mode a few times to see what happens[47]
    d. But we don't have wave of enemies
3. What we want our code to do is to wait after spawning each asteroid hazard before spawning the next, so we have a steady barrage of hazards challenge our player
    a. Return to the game controller script
    b. We need a public float variable called spawnWait to hold our wait time value
    c. Where we want to place our wait code(WaitForSenconds(spawnWait)) is at the end of spawn code in the for loop before the code loops back and spawn the next asteroid hazard
    d. But this syntax doesn't work in c#. To have a function that can pause without pausing our entire game, we need to make this function a coroutine and coroutines have some very pacific considerations[48]
    e. For this function to be coroutine, we cannot to return void. We must return IEnumerator

---

[46] We will be able to set this number in the inspector view in Unity.

[47] Very much like when we duplicate the code we are spawning a number of asteroid hazards all at once. Many are destroying each other on the first frame when they overlap and collide. We have simplified our code and we have easy control over the number of hazards that we spawn but we don't have waves of enemies.

[48] A coroutine is like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame. A coroutine is a function that can suspend its execution (yield) until the given given YieldInstruction finishes.

f. Our WaitForSeconcods line must return with yield return new WaitForSeconds
g. Lastly we must explicitly use Startcoroutine(SpawnWaves())

4. To be polite to our new players, it will be a short pause after the game starts
   a. Let's create a new public float variable called startWait
   b. Let's use this as the first line in our new coroutine
      i. Copy the spawnWait line
      ii. Paste it as the first line of the block
      iii. Change the spawnWait to startWait

```
public int hazardCount;
public float spawnWait;
public float startWait;


void Start(){
    StartCoroutine(SpawnWaves ());
}

IEnumerator SpawnWaves(){

    yield return new WaitForSeconds (startWait);

    for (int i = 0; i < hazardCount; i++) {
       Vector3 spawnPosition = new Vector3 (
                    Random.Range (-spawnValues.x, spawnValues.x),
                     spawnValues.y, spawnValues.z);
       Quaternion spawnRotation = Quaternion.identity;

       Instantiate (hazard, spawnPosition, spawnRotation);
       yield return new WaitForSeconds (spawnWait);
    }
}
```

   c. Save this script and return to Unity
   d. We will spawn 2 asteroids every second, So let's set spawn wait to 0.5
   e. Let's set the start wait to 1 giving our players one second to get ready
   f. Save the scene and enter play mode to test[49]

5. We want to create continuous waves of hazards until our player is destroyed and the game is over. We can do this by rapping our instantiate loop into another loop.
   a. Return to the GameController script
   b. Select the loop we have written
   c. Indent that code
   d. Write while loop and set conditional test to true
   e. To create a gap between our waves we need to wait
      i. Write `public float waveWait`
      ii. At the end of while loop write
      `yield return new WaitForSeconds (waveWait);`
   f. Save this script and return to Unity
      i. Let's set wave wait property of the game controller component to 4
      ii. Save the scene and enter play mode to test

6. We will need to write another way of destroying explosion game objects by time
   a. Exit play mode
   b. Select the script folder
   c. Use the create menu in the project

[49] We will get a steady hazard. But when the first 10 asteroids have cleared the screen, there is nothing more for the player to do.

d. Choose C# script to create a new script and rename this script DestoryByTIme
e. With the script selected open it for editing
f. Remove the sample code and write code

```
public float lifetime;

void Start(){
    Destory(gameObject, lifetime)
}
```

g. Save this script and return to Unity

7. Let's update all of our explosion prefabs's component
    a. Select the explosions folder in prefabs VFX
    b. Select asteroid explosion and using the add component button select scripts
    c. Choose DestoryByTime
    d. Set the value of life time to 2
    e. Add a script this way to all the other prefabs in the explosion folder
    f. Save the scene and enter play mode to test

Readings

- Coroutines(Manual), http://docs.unity3d.com/Documentation/Manual/Coroutines.html
- Coroutines(Script), http://docs.unity3d.com/Documentation/ScriptReference/Coroutine.html
- StartCoroutine, http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.StartCoroutine.html
- YieldInstruction, http://docs.unity3d.com/Documentation/ScriptReference/YieldInstruction.html
- Destory, http://docs.unity3d.com/Documentation/ScriptReference/Object.Destroy.html
- Creating and Destroying GameObjects(Manual), http://docs.unity3d.com/Documentation/Manual/CreateDestroyObjects.html
- Mul-Object Editing(Manul), l http://docs.unity3d.com/Documentation/Manual/Multi-ObjectEditing.html

GameController.cs

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

public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;
    public float waveWait;
```

```
    void Start(){
        StartCoroutine(SpawnWaves ());
    }

    IEnumerator SpawnWaves(){

        yield return new WaitForSeconds (startWait);
        while (true) {
            for (int i = 0; i < hazardCount; i++) {
                Vector3 spawnPosition = new Vector3 (Random.Range (-
spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
                Quaternion spawnRotation = Quaternion.identity;

                Instantiate (hazard, spawnPosition, spawnRotation);
                yield return new WaitForSeconds (spawnWait);
            }
            yield return new WaitForSeconds (waveWait);
        }
    }
}
```

3. Scoring, Finishing and Building the Game

3.1 Audio

Now it's time to polish our game. We're going to be adding audio sound effects and music. There are three main audio components in unity. They are audio clips, audio sources and the audio listener. We will concentrate on audio clips and audio sources. Audio clips hold our audio data or sound files. Audio sources play our audio clips in the scene.

1. Let's find our audio clips in assets folder
    a. If select any one of the audio clips, we will see that clips import settings in the inspector
        i. The preview window[50] will display additional information about that clip
        ii. We can see the clips waveform
        iii. We can play or loop the audio clips
        iv. We can get additional information about that audio clip printed at the bottom
    ~~b. Make sure 3D sound has been deselected~~
    c. The audio files that we have in this project are[51]
        i. Three types styles of explosions (Asteroid, enemy and player)
        ii. A background music track
        iii. Two weapons' effects
    d. We need to associate audio clip with our explosion prefabs and our player ship
        i. The explosions need to play their audio clip when they are first instantiated into the scene
        ii. The player ship needs to play its audio clip when the player fires their weapons
        iii. We associate an audio clip with a game object by using an audio source component
        iv. The audio source plays an audio clip
        v. We could add an audio source component to our game objects and then reference an audio clip for that source to play, but there is an easier way to do this. If we drag an audio clip onto a game object, Unity will create a new audio source on the game object and automatically reference the dragged audio clip
2. Let's see how to drag an audio clip onto a gam object or prefab in action
    a. We can drag assets easily from our project view onto game objects in hierarchy
    b. To drag an asset like our audio clip onto a prefab asset which both are in project view, we can change existing project view from using the two column layout to using a single-column layout. This allow us to drag objects from one part of our project view to another without losing focus on our target asset
    c. The first target asset will be the asteroid explosion prefab. The asteroid hazards have a reference to this prefab. When we destroy the asteroid, the asteroid will spawn this prefab. At that point we want it to go bang.
        i. with explosion asteroid prefab selected
        ii. drag the explosion asteroid audio clip onto the game object in the inspector and drop it

[50] If the preview window has been closed, we can drag the preview windows header bar up to reopen it.
[51] In this portion of the space shooter project we will be ignoring the enemy sounds. They are for later optional assignment.

d. We can see that Unity has added an audio source to the prefab asset and the audio clip that we dragged and dropped on it.
    i. We can safely ignore 3D Sound Settings
    ii. The only important setting we need to focus is Play On Awake
    iii. For our explosions to work properly we need to make sure that it is set on[52]
e. We can also drag an audio clip onto a prefab asset without having it visible in the inspector
    i. Select the explosion player audio clip in the Audio folder
    ii. Drag it onto the explosion player prefab asset
f. The player's weapons sound is almost simple
    i. Select the weapon player audio clip
    ii. Drag it into the scene view and drop it onto the player game object
    iii. Select the player game object in the hierarchy view and find Audio Source component
    iv. Deselect the Play On Awake[53]

3. If we won't be playing this sound automatically, how will we play it? We must do this from PlayerController script
    a. Select the PlayerController script and open it for editing
    b. Declare a local member variable to hold the Rigidbody reference by writing at the top of the class

```
private float nextFire;
private AudioSource audioSource;
```

    c. And in void Start() write

```
rb = GetComponent<Rigidbody>();
audioSource = GetComponent<AudioSource> ();
```

    d. Then use the line in Update()

```
Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
audioSource.Play ();
```

    e. Save this script and return to Unity
    f. Save the scene and enter play mode
    g. We have sound effects
4. Let's put the background music track on our game controller
    a. We can find music background in audio folder
    b. Drag music background into our scene view and drop it on Game Controller
        i. This will add a new audio source to game controller with music background as the referenced clip
        ii.
    c. We want this music to start right away and play during the entire game
        i. For this we want to make sure that Play On Awake is selected[54] so it will begin playing on the first frame
        ii. Select loop as well so it the audio runs continuously
    d. Save the scene and enter play mode
    e. Exit the play mode
5. The last step we need to take is to balance the audio. All the clips are currently being played back at full volume and they don't mix well. Let's bring some of these effects down in volume

[52] This way the audio will play automatically when we spawn our explosion
[53] Because we definitely do not want Play On Awake set. Otherwise we would get our weapons sound playing on the very first frame of the game
[54]

a. Select the player game object
b. Bring up the audio source component
c. Chang the volume property to 0.5 to reduce the strength of the player's weapons sound
d. Next select the controller game object and on the audio component reduce the volume to 0.5
e. Save the scene and enter play mode

Readings

- Audio Files, http://docs.unity3d.com/Documentation/Manual/AudioFiles.html
- Audio Clip, http://docs.unity3d.com/Documentation/Components/class-AudioClip.html
- Audio Component, http://docs.unity3d.com/Documentation/Components/comp-AudioGroup.html
- Audio Source, http://docs.unity3d.com/Documentation/Components/class-AudioSource.html
- Audio Listener, http://docs.unity3d.com/Documentation/Components/class-AudioListener.html
- Audio Filter, http://docs.unity3d.com/Documentation/Components/class-AudioEffect.html

PlayerController.cs

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

[System.Serializable]
public class Boundary{
    public float xMin, xMax, zMin, zMax;
}

public class PlayerController : MonoBehaviour {

    public float speed;
    public float tilt;

    private Rigidbody rb;
    public Boundary boundary;

    public GameObject shot;
    public Transform shotSpawn;
    public float fireRate;

    private float nextFire;
    private AudioSource audioSource;

    void Start(){
        rb = GetComponent<Rigidbody>();
        audioSource = GetComponent<AudioSource> ();
    }
```

```
    void Update(){

        if(Input.GetButton("Fire1") && Time.time > nextFire){
            nextFire = Time.time + fireRate;
            Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
            audioSource.Play ();
        }
    }

    void FixedUpdate(){

        float  moveHorizontal = Input.GetAxis("Horizontal");
        float  moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
        rb.velocity = movement * speed;

        rb.position = new Vector3 (Mathf.Clamp(rb.position.x, boundary.xMin, bo
undary.xMax) ,0.0f, Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax));

        rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
    }

}
```

3.2 Counting points and displaying the score

We can destroy our hazards. We should reward our player with points. Let's count and display a score where every asteroid hazards destroyed gives our player 10 points

1. We will start by creating a display for the score by using Unity's UI toolset before we create a score value to feed it with
    a. Create a UI element from the hierarchy's Create menu Hierarchy -> Create -> UI -> Text
    b. Rename the text element Score Text (All UI element must be the child of a canvas to behave correctly)
    c. Make the text color white, so it's easier to see
    d. Set Font Size to 20
    e. Edit placeholder text "Score:"
    f. Display the count text in the upper left of the screen when the game is playing
        i. Click on the button displaying the current anchor preset
        ii. Hold down both the Shift and the Alt keys (Set the pivot and the position based on the new anchor)
        iii. Select the upper left corner button
        iv. Change the Rect Transform(Pos X = 20 and Pos Y = -20)
2. Now to hook up our game logic to the score text.
    a. Open the GameController script
    b. Use name space just as using UnityEngine and System.Collections (`using UnityEngine.UI ;`)
    c. Create a new public text variable called scoreText to hold a reference to the UI text component on our UI Text game object (`public Text scoreText;`)
    d. Create a new private int variable called score to hold the score (`private int score`)
    e. Write the same code in Start Function

```
void Start(){
     StartCoroutine(SpawnWaves ());
     score = 0;
     UpdateScore ();
}
```

    f. UpdateScore();

```
void UpdateScore(){
    scoreText.text= "Score: " + score.ToString ();
}
```

    g. Associate a reference to the Score Text Property by dragging and dropping the CountText game object on to the slot
    h. Save the script and make a test
3. Next we need to call AddScore with an appropriate value whenever the player does something worthy of points… shooting an asteroid for example.
    a. Open up the DestroyByContact script
    b. We need a variable to reference our GameController instance, and we need to call AddScore in that GameController.

```
private GameController gameController
public int scoreValue;
….
GameController. AddScore (scoreValue);
Destroy(other.gameObject);
Destroy(gameObject);
```

    c. Save the script
    d. Select the Asteroid prefab in project view and try to drag our instance of GameController into the script slots.

i. It doesn't work, because our Asteroid is just a prefab
ii. We have to wait until an instance of Asteroid is created and then find an instance of GameController to reference.

4. Reopen DestroyByContact
   a. Create start() function

```
void Start(){
    GameObject gameControllerObject = GameObject.FindWithTag ("GameController");

    if (gameControllerObject != null) {
        gameController = gameControllerObject.GetComponent<GameController> ();
    }
    if(gameController == null){
        Debug.Log("Cannot find 'GameController' script");
    }

}
```

   b.
5. Save and test
   a. Check the Asteroid prefab and set its score value to some positive integer(for example 10)
   b. Enter to play mode
   c. The score should now increment every time an asteroid is destroyed, whether by a shot or a collision with ship

Readings

- Instantiate, http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html
- GameObject.tag, http://docs.unity3d.com/Documentation/ScriptReference/GameObject-tag.html
- UICanvas, https://docs.unity3d.com/Manual/UICanvas.html
- UI Basic Layout, https://docs.unity3d.com/Manual/UIBasicLayout.html
- UI Visual Components, https://docs.unity3d.com/Manual/UIVisualComponents.html
- Interaction Components, https://docs.unity3d.com/Manual/UIInteractionComponents.html

GameController.cs

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

public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;
    public float waveWait;
```

```
    public Text scoreText;
    private int score;

    void Start(){
        StartCoroutine(SpawnWaves ());
        score = 0;
        UpdateScore ();
    }

    void UpdateScore(){
        scoreText.text = "Score:" + score.ToString();
    }

    public void AddScore(int newScoreValue){
        score += newScoreValue;
        UpdateScore ();
    }

    IEnumerator SpawnWaves(){

        yield return new WaitForSeconds (startWait);
        while (true) {
            for (int i = 0; i < hazardCount; i++) {
                Vector3 spawnPosition = new Vector3 (Random.Range (-
spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
                Quaternion spawnRotation = Quaternion.identity;

                Instantiate (hazard, spawnPosition, spawnRotation);
                yield return new WaitForSeconds (spawnWait);
            }
            yield return new WaitForSeconds (waveWait);
        }
    }
}
```

DestroyByContact.cs

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

public class DestroyByContact : MonoBehaviour
{
    public GameObject explosion;
    public GameObject playerExplosion;

    private GameController gameController;
    public int scoreValue;

    void Start(){
        GameObject gameControllerObject = GameObject.FindWithTag ("GameControll
er");

        if (gameControllerObject != null) {
            gameController = gameControllerObject.GetComponent<GameController>
();
        }
        if(gameController == null){
            Debug.Log("Cannot find 'GameController' script");
```

```
        }

    }


    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
            return;
        }

        Instantiate (explosion, transform.position, transform.rotation);
        if (other.tag == "Player"){
            Instantiate (playerExplosion, other.transform.position, other.trans
form.rotation);
        }
        gameController.AddScore (scoreValue);
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

http://www.davidtangness.com/journal/2014/12/24/unity-engine-space-shooter-tutorial

3.3 Ending the game

At this point our game is playable but it runs forever. We need to end the game when the player is destroyed, and give the player an option to restart.

1. We need to create two new labels, one to display game over and the other to indicate what is OK to restart the game
    a. Create two new text game objects
        i. Click on `Create | UI | Text` in the hierarchy view
    b. Rename them: GameOver Text and Restart Text
    c. Make the text color white, so it's easier to see
    d. Edit the placeholder text: Game over and Restart
    e. Adjust the transform to display Restart Text in the upper right screen and GameOver Text in the middle screen
        i. Clikc on the Anchor Presets button
        ii. Hold down both the Alt and the Shift key
        iii. Select the Top Right button
        iv. Change the Rect Transform(Pos X = 20 and Pos Y = -20)
    f. Adjust the transform to display GameOver Text in the center screen
        i. Clikc on the Anchor Presets button
        ii. Hold down both the Alt and the Shift key
        iii. Select the Middle Center button
        iv. Change the Rect Transform(Pos X = 20 and Pos Y = -20)
    g. Change the Restart Text font to 20
    h. Change the GameOver Text font to 25
2. Open the GameController script and add references, states and initialization for these texts
    a. Open the GameController script for editing
    b. Use name space just as using UnityEngine and System.Collections
    ```
    using UnityEngine.UI;
    ```
    c. Create two new public text variables to hold the reference to the UI Text component on our UI Text game object.
    ```
    public Text scoreText;
    public Text restartText;
    public Text gameOverText;
    ```
    d. Create two new bool variables to help us track when the game is over and when it is ok to restart the game
    ```
    private bool gameOver;
    private bool restart;
    ```
    e. We need to set the values for these variable at the Start of the game
    ```
    void Start(){
        gameOver = false;
        restart = false;
        gameOverText.text = "";
        restartText.text = "";

        StartCoroutine(SpawnWaves ());
        score = 0;
        UpdateScore ();
    }
    ```
3. We need to change the state of the game and the content of these labels when the game is over. Then we need to make a function which can be called to end our game
    ```
    public void GameOver(){
        gameOverText.text = "Game Over";
        gameOver = true;
    }
    ```
4. We will use the game over flag to break out of the infinite loop that's spawnning hazards
    ```
    yield return new WaitForSeconds (waveWait);

    if(gameOver){
    ```

```
                restartText.text = "Press 'R' for Restart";
                restart = true;
                break;
            }
```

5. We will now write the code to restart the game when the player press the R key
    a.
```
using UnityEngine.UI;
using UnityEngine.SceneManagement;
 …
   void Update(){
       if (restart) {
           if(Input.GetKeyDown(KeyCode.R)){
               Application.LoadLevel(Application.loadedLevel);
               SceneManager.LoadScene("Main");
           }
       }
   }
```
    b. Save this script and return to Unity
    c. We need to set up our new references that we have written on GameController
        i. Select Game Controller game object in the hierarchy view
        ii. Drag the Restart Text game object onto the restart text slot on the game controller component
        iii. Drag the Game Over Text game object onto the game over text slot on the game controller component
    d.
6. Finally we need to call GameOver() when the player ship is destroyed. We could try to do this on the player game object but we have already written this code somewhere else. Our asteroid hazards already detect collisions with our player and that collision destroys the player game object
    a. Opent the DestroyByContact script for editing
    b. Call the gameController.GameOver() when we destroy the player
```
if (other.tag == "Player"){
    Instantiate (playerExplosion, other.transform.position, other.transform.
rotation);
    gameController.GameOver();
}
gameController.AddScore (scoreValue);
```
    c. Save this script and return to Unity
7. Save the scene and enter playmode
    a. Fantastic
    b. Our game is complete

Readings

- UI System, https://docs.unity3d.com/Manual/UISystem.html
- Rect Transform, https://docs.unity3d.com/Manual/class-RectTransform.html
- UI Text, https://docs.unity3d.com/ScriptReference/UI.Text.html
- Scene Manager, https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html
- Scene Manager LoadScene, https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html

- Application, https://docs.unity3d.com/ScriptReference/Application.html
- Application Quit, https://docs.unity3d.com/ScriptReference/Application.Quit.html

GameController.cs

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

public class GameController : MonoBehaviour {

    public GameObject hazard;
    public Vector3 spawnValues;
    public int hazardCount;
    public float spawnWait;
    public float startWait;
    public float waveWait;

    public Text scoreText;
    public Text restartText;
    public Text gameOverText;

    private bool gameOver;
    private bool restart;
    private int score;

    void Start(){
        gameOver = false;
        restart = false;
        gameOverText.text = "";
        restartText.text = "";

        StartCoroutine(SpawnWaves ());
        score = 0;
        UpdateScore ();
    }

    void Update(){
        if (restart) {
            if(Input.GetKeyDown(KeyCode.R)){
                SceneManager.LoadScene("Main");
            }
        }
    }

    void UpdateScore(){
        scoreText.text = "Score:" + score.ToString();
    }

    public void AddScore(int newScoreValue){
        score += newScoreValue;
        UpdateScore ();
    }

    IEnumerator SpawnWaves(){
```

```
        yield return new WaitForSeconds (startWait);
        while (true) {
            for (int i = 0; i < hazardCount; i++) {
                Vector3 spawnPosition = new Vector3 (Random.Range (-
spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
                Quaternion spawnRotation = Quaternion.identity;

                Instantiate (hazard, spawnPosition, spawnRotation);
                yield return new WaitForSeconds (spawnWait);
            }
            yield return new WaitForSeconds (waveWait);

            if(gameOver){
                restartText.text = "Press 'R' for Restart";
                restart = true;
                break;
            }
        }
    }

    public void GameOver(){
        gameOverText.text = "Game Over";
        gameOver = true;
    }
}
```

DestroyByContact.cs

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

public class DestroyByContact : MonoBehaviour
{
    public GameObject explosion;
    public GameObject playerExplosion;

    private GameController gameController;
    public int scoreValue;

    void Start(){
        GameObject gameControllerObject = GameObject.FindWithTag ("GameControll
er");

        if (gameControllerObject != null) {
            gameController = gameControllerObject.GetComponent<GameController>
();
        }
        if(gameController == null){
            Debug.Log("Cannot find 'GameController' script");
        }

    }


    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Boundary")
        {
```

```
            return;
        }

        Instantiate (explosion, transform.position, transform.rotation);
        if (other.tag == "Player"){
            Instantiate (playerExplosion, other.transform.position, other.trans
form.rotation);
            gameController.GameOver();
        }
        gameController.AddScore (scoreValue);
        Destroy(other.gameObject);
        Destroy(gameObject);
    }
}
```

3.4 Building the game

Our game is complete. Let's build our web player and deploy web player.

1. Open the build settings window
    a. Click on `File | Build Settings` in the tools bar
    b. Select `WebGL` in the building settings window
    c. Customize the details for the build by clicking on `Player Settings`
    d. Add the scenes you want to include when you build your game
        i. Select the scenes you want
        ii. Drag the scenes from project view and drop it onto the upper window of build settings
        iii. Or click `Add Open Scenes` button
    e. Build
        i. Create a new folder inside Space Shooter folder and name it Builds
        ii. With the builds folder selected let's name our build Space_Shooter
        iii. Save
    f. When the build is done, Unity will open the target build folder
2. Open the index.html to play the game locally
3. Upload the game to web host like github

Readings

- Player Settings(Script Reference), http://docs.unity3d.com/Documentation/ScriptReference/PlayerSettings.html
- Player Settings(Manual), http://docs.unity3d.com/Documentation/Components/class-PlayerSettings40.htmlReference
- Build Player Pipeline, https://docs.unity3d.com/Manual/BuildPlayerPipeline.html
- WebGL Player Settings, https://docs.unity3d.com/Manual/class-PlayerSettingsWebGL.html
- WebGL Development, https://docs.unity3d.com/Manual/webgl-gettingstarted.html
- WebGL Browser Compatibility, https://docs.unity3d.com/Manual/webgl-browsercompatibility.html
- Building and running a WebGL project, https://docs.unity3d.com/Manual/webgl-building.html
- Debugging and trouble shooting WebGL, https://docs.unity3d.com/Manual/webgl-debugging.html

4. Extending Space Shooter

4.1 Extending Space Shooter, Enemies, More Hazard, Scrolling BG

4.2 Mobile Development: Converting Space shooter to Mobile