Project 2 Space Shooter

ZHU Yachao
Guangxi University of Science and Technology
Created:2017-03-06
Last Updated:2017-03-10

Objectives

Table of Contents

Project 2 Space Shooter **1**
Objectives **1**
1. Game Setup, Player and Camera **2**
1.1 Setting up the project 2
1.2 The player GameObject 4
1.3 Camera and lighting 7
1.4 Adding a background 10
1.5 Moving the player 12
1.6 Creating shots 15
1.7 Shooting shots 18
2. Boundaries, Hazards and Enemies **20**
2.1 Boundary 20
2.2 Creating hazards 20
2.3 Explosions 20
2.4 Game Controller 20
2.5 Spawning waves 20
3. Scoring, Finishing and Building the Game **21**
3.1 Audio 21
3.2 Counting points and displaying the score 21
3.3 Ending the game 21
3.4 Building the game 21
4. Extending Space Shooter **22**
4.1 Extending Space Shooter, Enemies, More Hazard, Scrolling BG 22
4.2 Mobile Development: Converting Space shooter to Mobile 22

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, 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);
    }

}
```

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 ship35. W 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

2. Boundaries, Hazards and Enemies

2.1 Boundary

2.2 Creating hazards

2.3 Explosions

2.4 Game Controller

2.5 Spawning waves

3. Scoring, Finishing and Building the Game

3.1 Audio

3.2 Counting points and displaying the score

3.3 Ending the game

3.4 Building the game

4. Extending Space Shooter

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

4.2 Mobile Development: Converting Space shooter to Mobile