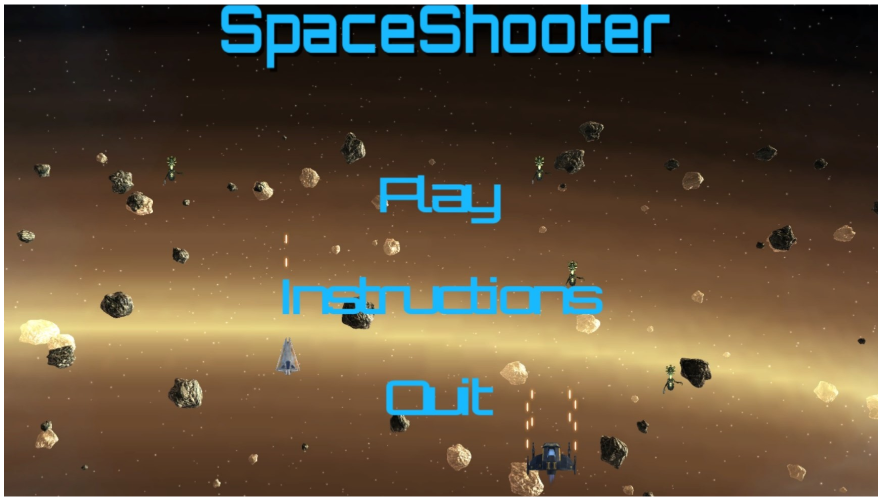
Game Developer Expertise Test

Artemis Georgakopoulou
art.geo@gmail.com
30/07/2019
SpaceShooter Game



Part 1Game Overview - Introduction

SpaceShooter is a simple 2D Game I created, based on Unity's SpaceShooter Tutorials. The tutorials mainly helped on focusing on the important points, in order to not waste any time while creating the game.
Some of the 3D Models/Materials/Textures I have used have been either downloaded from this tutorial, or from the AssetStore. I have designed/programmed the Level Design and Game Logic.
Audio clips have also been downloaded from the Web.

Summary Plot of the Game

The whole game consists of 2 Levels and a Main Menu.
MainMenu
Firstly, after executing the application we can see the Main Menu Panel, which is basically a snapshot of a scene I designed, deactivated the script components and saved as "MainMenu" Scene with buildIndex = 0. The Buttons of the panel play Animations, which I designed myself, once high-lightened or pressed.

Play Button initializes the GamePlay and loads the first level, which is another Scene named “Main”.
Quit Button quits the game and ends the application.
Instructions Button deactivates the MainMenu Panel and activates Instructions Panel, where we can see the instructions of how to move the player Ship, gain points and win the game.

Level1
This is a simple level, where ship moves through Space while using either (W,A,S,D) or arrow keys to navigate. Keys (Q,E) rotate the ship, so that player can fire in multiple angles. (Space) Key fires laser bullets and smashes incoming asteroids. When player reaches 300Points (10/asteroid), Win Panel loads with a custom Animation and asks player if he/she wishes to -Continue to next Level, -Retry same Level, or -Quit game.

Level2
Difficulty increases on Level2, as the incoming asteroids consist of 2 types: the small ones and the bigger/faster ones. When player hits the bigger he/she receives 15 points. Additionally, shooting aliens appear in waves trying to kill the player. Player receives 20Points/Alien Kill. Once player reaches 400Points he/she can update the space ship into a bigger one with more fire holes and faster fire rate. Once he/she reaches 1000Point he/she wins the game and is presented with the options to: -Retry whole Game, -Quit.

<u>Folders</u>
-Animation: contains .anim files, all custom made.
-Audio: contains audioclips assets, all downloaded from web.
-Fonts: contains fonts assets, all downloaded from web.
-Imported: contains files imported from downloaded .unitypackage files
-Materials / -Models / -Textures: contains materials, models and textures of Unity’s tutorial

-Prefabs: contains modified downloaded and custom made Prefabs.
-Resources: contains one prefab file, for easy instantiation.
-Scenes: contains all 3 Scenes of the Game.
-Scripts: contains all the Scripts.

Scripts
-BGScroller: In the Hierarchy the Background GO consists of the main Plane, plus 2 more extension Planes. This script moves the background Tiles on the z axis to create the illusion of moving forward through space.
-Boundary: A Boundary Cube hold this script in order to destroy useless GameObjects from the scene.
-DestroyByContact: GOs like the Asteroids/Aliens hold this script in order to get destroyed appropriately
-DestroyByTime: Destroys GO after a specific amount of time.
-EvasiveManeuver: Attached on the Alien prefab, controls the "Random" movement of the alien.
-GameController: Controls the core of the game, the enemy waves and status of the game.
-GameOver: Controls the GameOver Panel.
-GamePaused: Have not completed in using this script - Unused
-MainMenu:Controls the MainMenu Panel.
-*MoveDiagLeft*: *Have not completed in using this script - Unused*
-*MoveDiagonal*: *Have not completed in using this script - Unused*
-Mover: Controls the forward movement of the Aliens.
-MoverAsteroids: Controls the movement of the Asteroids
-PlayerController: Controls the movement of the Player
-PlayerStats: Holds the score of the player
-RandomRotator: Randomly rotates Asteroids
-UpdateShip: Updates the ship when the corresponding Button gets pressed.
-WeaponController: Fires the bullets
-WinMenu: Controls the WinMenu Panel

Code contains comments for further details. Configurations settings (enemy/ship/asteroids speed, points etc set up so that game play makes sense and are corresponding to each other.

Part 2 - Questions

-MipMapping

When enabled on texture import settings, MipMaps are smaller versions of textures, used when rendering textures far from the camera. This allows textures in the distance to appear smoother and also improve rendering performance giving the GPU smaller textures to render. That way, if the renderer would struggle with a 2k texture, instead it can attempt to use one of the smaller MipMap versions.

-Transformation Matrices

Unity Transformation Matrices store object's position, scale and rotation (x,y,z,w)(w as the real part and x, y, z as imaginary parts) in two vectors and a Quaternion, which basically signifies their rotation/direction in 3D space and distinguishes depth. In order to stored all that information in a single matrix, we need 4x4 dimensions.

-Stencil Buffer

The depth buffer helps us compare depths of objects to ensure they occlude each other properly. But there's also a part of the buffer reserved for "stencil operations". This part of the depth buffer is commonly referred to as stencil buffer. Stencil buffers are mostly used to render only parts of objects, while discarding others. The stencil buffer is also used by Unity internally for the deferred graphics pipeline, so if we do deferred rendering, some limitations are applied.

-GPU Rendering/Performace Optimization

Examples:

--Use frustum culling, Only render what the camera sees.

--Use Compressed textures or texture MipMaps to decrease the size of textures.

--Use LOD and per-layer cull distances. Render according to a set-up Level of Detail and make objects visible at a specific distance from camera.

--Use Occlusion Culling to reduce the amount of visible geometry and draw-calls in cases of complex static scenes with lots of occlusion.

-MVC design Pattern

The Model-View-Controller pattern (MVC) splits the software into three major components: Models (Data CRUD), Views (Interface/Detection) and Controllers (Decision/Action).

Models store values. In a game related example the model of the player would store their total and current health points, their speed, attack power and so on. Models do not perform any calculations or decisions, nor do they interact with other units or components. They just handle data.

The View handles inputs and events and is in charge of graphical representation or objects in my concept. The View also is the only part of the MVC unit that communicates with other scripts outside the unit itself.

The Controller handles information, makes calculations and decisions with it and sends it to the model or returns it to the view.

As a game grows in size and complexity, it gets harder and harder to test new stuff or check if the old things still work after any modifications. The problem with Unity is that it is object oriented and that it is often impossible to test something if those objects don't get instantiated by the game. Unity's unit tests allow faking objects to test something, though this functionality is very limited.

With an MVC design pattern calculations and mechanics can be separated from the objects in the game and facilitate testing.

-Coroutines

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.

-Start() vs Awake()

Awake is called when the script instance is being loaded. It is used to initialize any variables or game state before the game starts. It is called only once during the lifetime of the script instance.

Start is called on the frame when a script is enabled just before any of the Update methods are called the first time. Like the Awake function, it is called exactly once in the lifetime of the script.

However, Awake is called when the script object is initialized, regardless of whether or not the script is enabled. Start may not be called on the same frame as Awake if the script is not enabled at initialization time. The Awake function is called on all objects in the Scene before any object's Start function is called. This fact is useful in cases where object A's initialization code needs to rely on object B's already being initialized; B's initialization should be done in Awake, while A's should be done in Start.

-Abstract Classes vs Interfaces

In an abstract class, we can create the functionality and that needs to be implemented by the derived class. An abstract class can actually have functionality in it, which child classes take advantage of. The interface allows us to define the functionality or functions but cannot implement that. The derived class extend the interface and implement those functions. An interface only acts as a contract for the functions in a class, so no functionality in interfaces at all.

In our game, we use abstract classes when we have multiple items that share some base functionality, but that base functionality is not enough to be an object itself. Interfaces are useful when we want to refer to several disparate objects by a common attribute.

-Canvas Group

The Canvas Group Component allows us to adjust specific properties of the UI object that it is attached to and all of its children at once, rather than having to adjust these properties individually. The order of the UI objects in Hierarchy is a common issue of this component.

-Exit time on animation state transition

When exit time is ticked,if the animation is completed then the next animation is played. For example, a character might have a “idle” state and a “moving” state, once the one animation reaches exit time, the next one plays.