**Computer Graphics and Game Programming – 508837**
**4th Assignment**
**Advisor: Assoc. Dr. Murat KURT**

# 4th Assignment Report

**05.20.2023**

**By:**
**Mehmet ISIKHAN, 91220001906**

## Table of Contents

# Introduction

This report includes developing processes, of a plane game with the mini planes flying on a part of the New York City developed on Unity game engine. The big maps cause the long loading time and large number of calculation costs, so in this game the map chose as part of the New York City and the planes' sizes are edited according to the flying on the city, in this way the planes can fly inside the buildings. Also, for the limitation of the map ranges a signal system developed and thanks to this signal system the game was born with a story. In the story, one of the roof of the buildings, a young man who living in a container, build a mini airport on the roof, and he controls the planes with his computer with connection to an antenna on the roof. According to the antenna range the planes flying, and he controls the signal level on his screens. But the water tank on the roof causes the low signal problem on his west side. According to the story, at the beginning of the game the character must make a plane (the players should choose one of the planes) and he is flying with this plane. Also, the game allows changing the camera positions both at the beginning of the game and in the game. If the player hits a building or passes the signal borders, the plane start falling and a game over screen showing to the players.

# Plane Game

## 1. Character Animation

For creating the history, the character needed to do something related to the story, so the game start with the character animation, and in this animation, he is doing some things on his computer for the starting to fly. I found the three-dimension character and the other components such as desk monitors chair on the “cgtrader” website ***[1]*** and I added to my project. For the motions of the character, I created an animation object on Unity, and I bring together on the animator screen then I assign this animator object to the character animator component. I found the character animations on the “Mixamo” web page ***[2][3]***. After the character animation, the camera is moving out of the character’s container to the airport. For this camera animation, I created another animation object and animator then I assigned this animator object to the camera as the animator component. In this way, the starting of the game was made, the player must choose a camera position and plane for the flying, but the game should know the animations are over for showing of the camera position selection and plane selection menus. So, the “planeSelectionMenu” script developed for the understanding the finishing of the camera animation (CameraAnimation). After the camera animation the script set active the camera selection menu (cameraCanvas). Also, for closing the camera selection menu there is a function in this script after the camera selected.

***Picture.1: Character Animation***

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

public class PlaneSelectionMenu : MonoBehaviour
{
    [SerializeField] private Animator animator;
    private int cameraAnimationHash;
    [SerializeField] private GameObject cameraCanvas;

    public static bool isCameraSelected = false;

    private void Start()
    {
      cameraAnimationHash =
      Animator.StringToHash("CameraAnimation");
    }

    private void Update()
    {
      AnimatorStateInfo stateInfo =
      animator.GetCurrentAnimatorStateInfo(0);

        if (stateInfo.shortNameHash == cameraAnimationHash)
        {
            // Check if the animation is over
            if (stateInfo.normalizedTime >= 1f
            &&(!isCameraSelected))
            {
                cameraCanvas.SetActive(true);
            }
        }
    }

    public void setCameraSelectionCanvas(){
        isCameraSelected = true;
        if(cameraCanvas.activeSelf){
            cameraCanvas.SetActive(false);
        }
    }
}
```

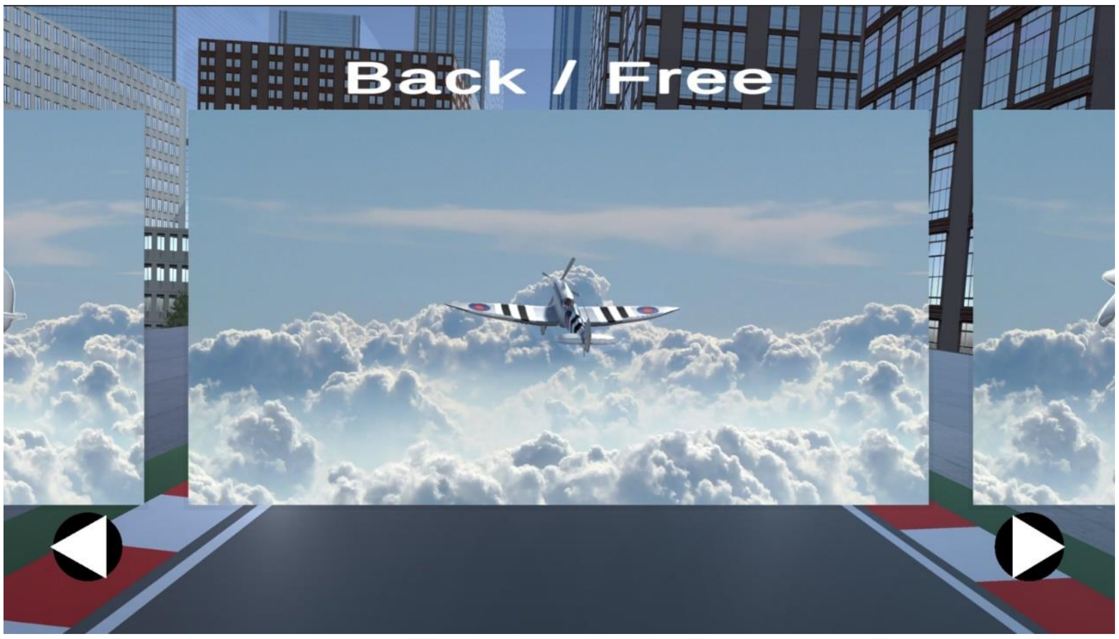
# 2. Selection Menus and Canvas

The selection menus are managed by the scripts, and they are like popup menus on the same scene because of my computer lighting render performance. In the car race game *[4]*, I could not get renders for all scenes and when the moving from one scene to other one, it was causing the shadowing and lighting problems. So, in this reason, I added all menu components in the same scene in this project like the popup menu and I manage these menu components with the scripts.

## a. Camera Selection Menu

After the animation, the first menu is camera position selection menu. As with all menus, there is a "scrollview" for navigating through the camera position options with the arrow buttons. All camera position components are located with the "1350-x" pixel space value, and the arrow buttons change this variable for the navigating. Also, there is a counter for blocking navigating of the outside of the menu. This counter keeps the navigating value so the "scrollview" does not move more when the player came to the last menu component.

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

public class ScrollBarScript : MonoBehaviour
{
    private ScrollRect scrollRect;
    [SerializeField] private Button rightButton;
    [SerializeField] private Button leftButton;
    private int menuCounter = 0;

    private void Start()
    {
        scrollRect = GetComponent<ScrollRect>();

        // Add button click listeners
        rightButton.onClick.AddListener(ScrollRight);
        leftButton.onClick.AddListener(ScrollLeft);
    }
    public void ScrollRight()
    {
        if(menuCounter != -2){
            scrollRect.content.localPosition = 
scrollRect.content.localPosition + new Vector3(-1350.0f,0f,0f);
            menuCounter--;
        }
    }

    public void ScrollLeft()
    {
        if(menuCounter != 2){
            scrollRect.content.localPosition = 
scrollRect.content.localPosition + new Vector3(1350.0f,0f,0f);
            menuCounter++;
        }
    }
}
```



***Picture.2: Camera Selection Menu***

There are five different camera positions the players can choose. Each plane has the same camera positions with different values according to the plane's features. Before starting the game, with the camera selection menu, the players can select the starter camera position also inside the game they can change the camera positions with "V" key for desktop *[5]* and "Cam" button for the mobile devices. According to selection of the camera position, the system makes active the selected camera and the other ones be deactivated.

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

public class SelectedCameraController : MonoBehaviour
{
    [SerializeField] private GameObject free;
    [SerializeField] private GameObject up;
    [SerializeField] private GameObject inside;
    [SerializeField] private GameObject rightSide;
    [SerializeField] private GameObject leftSide;

    private void Start(){
        changeCamera();
    }

    public void changeCamera(){
        switch(PlaneController.selectedCameraValue){
            case 1:
                leftSide.SetActive(false);
                rightSide.SetActive(false);
                inside.SetActive(false);
                up.SetActive(false);
                free.SetActive(true);
                break;
            case 2:
                leftSide.SetActive(false);
                rightSide.SetActive(false);
                inside.SetActive(false);
                free.SetActive(false);
                up.SetActive(true);
                break;
            case 3:
                leftSide.SetActive(false);
                rightSide.SetActive(false);
                up.SetActive(false);
                free.SetActive(false);
                inside.SetActive(true);
                break;
            case 4:
                leftSide.SetActive(false);
                inside.SetActive(false);
                up.SetActive(false);
                free.SetActive(false);
                rightSide.SetActive(true);
                break;
            case 5:
                rightSide.SetActive(false);
                inside.SetActive(false);
                up.SetActive(false);
                free.SetActive(false);
                leftSide.SetActive(true);
                break;
        }
    }

    void OnGUI(){
        Event e = Event.current;
        if (e.isKey)
        {
            switch(e.keyCode.ToString()){

                case "V":
                    changeCamInTheGame();
                    break;
            }
        }
    }

    public void changeCamInTheGame(){
        if(PlaneController.selectedCameraValue != 1){
                PlaneController.selectedCameraValue =
            PlaneController.selectedCameraValue - 1;
            }else{
                PlaneController.selectedCameraValue = 5;
            }
            changeCamera();
    }

}
```

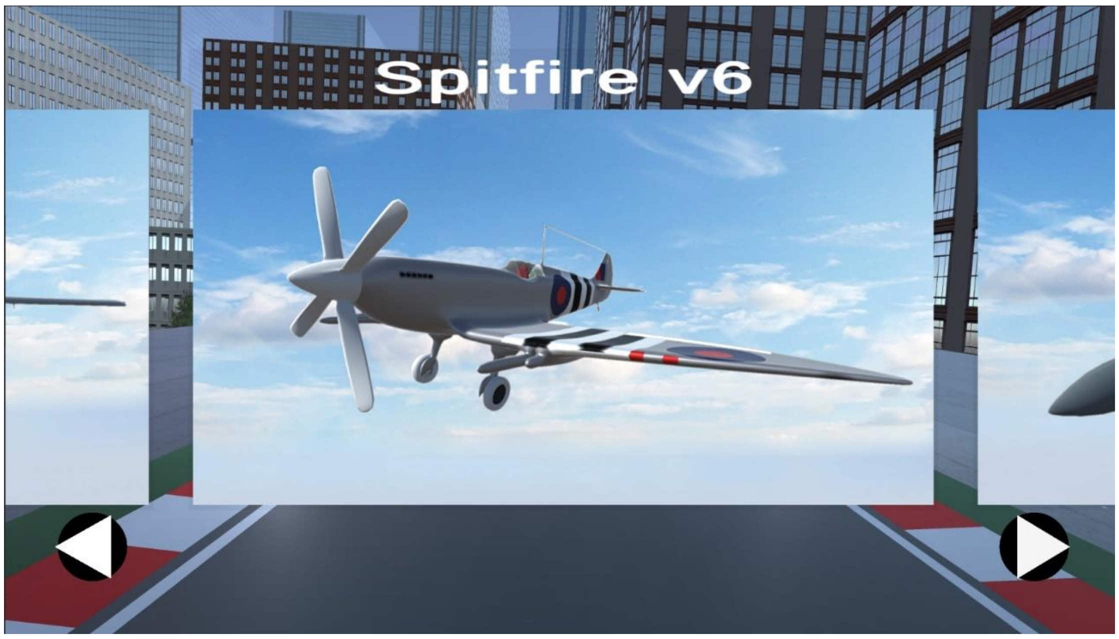
Also, the main selection function included in the “PlaneController” script with the plane selection phase. This situation occurred because of a bug when the plane selection is made before the camera selection phase. After the camera selection phase, the plane selection menu’s object set active after the camera selection menu set deactivated. (The below code belongs to PlayerController script)

```
        public void cameraSelection(int selectedCamera){
            selectedCameraValue = selectedCamera;
            planeSelectionCanvas.SetActive(true);
        }
```

## b. Plane Selection Menu

The plane selection menu phase acts like the camera selection menu for the selection processes. But this script includes the plane canvas components like digital joystick *[6]*, turbo button and camera button for the mobile devices and the signal canvas components for all devices. The “Start” function of the “PlaneController” script checks the input of the device like touchable or not *[7]*. According to this check result, the system decided to be making mobile components (joystick, turbo button and camera button) active or not. After the plane is selected, the plane canvas be active and the game start.

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

public class PlaneController : MonoBehaviour
{
    [SerializeField] private GameObject plane1;
    [SerializeField] private GameObject plane2;
    [SerializeField] private GameObject plane3;
    [SerializeField] private GameObject plane4;
    [SerializeField] private GameObject plane5;

    [SerializeField] private GameObject planeCanvas;
    [SerializeField] private GameObject planeSelectionCanvas;

    [SerializeField] private GameObject joystick;
    [SerializeField] private GameObject nitroButton;
    [SerializeField] private GameObject camButton;
    public static int selectedCameraValue = 0;

    public void Start(){
        if(SystemInfo.deviceType == DeviceType.Handheld){
            joystick.SetActive(true);
            nitroButton.SetActive(true);
            camButton.SetActive(true);
        }else{
            joystick.SetActive(false);
            nitroButton.SetActive(false);
            camButton.SetActive(false);
        }
    }
    public void planeSelection(int selectedPlane){
        switch(selectedPlane){
            case 1:
                plane5.SetActive(false);
                plane4.SetActive(false);
                plane3.SetActive(false);
                plane2.SetActive(false);
                plane1.SetActive(true);
                break;
            case 2:
                plane5.SetActive(false);
                plane4.SetActive(false);
                plane3.SetActive(false);
                plane1.SetActive(false);
                plane2.SetActive(true);
                break;
            case 3:
                plane5.SetActive(false);
                plane4.SetActive(false);
                plane2.SetActive(false);
                plane1.SetActive(false);
                plane3.SetActive(true);
                break;
            case 4:
                plane5.SetActive(false);
                plane3.SetActive(false);
                plane2.SetActive(false);
                plane1.SetActive(false);
                plane4.SetActive(true);
                break;
            case 5:
                plane3.SetActive(false);
                plane4.SetActive(false);
                plane2.SetActive(false);
                plane1.SetActive(false);
                plane5.SetActive(true);
                break;
        }
        planeSelectionCanvas.SetActive(false);
        planeCanvas.SetActive(true);
    }

    public void cameraSelection(int selectedCamera){
        selectedCameraValue = selectedCamera;
        planeSelectionCanvas.SetActive(true);
    }
}
```



***Picture.3: Plane Selection Menu***

## c. The Plane Canvas and Signal Zones

Keeping the plane inside the map is a necessary feature for a plane game. Because, for the plane, the walls or physical components cannot be limitations. My mission was to develop a plane game, and it has to playable on the mobile, so I needed to find something about the borders for the plane. I figured out this problem with the signal range limitation for the plane. There are four different zones for the border limitation. The first zone is the nearest zone to the base, and the signal shows three lines in this zone. The second zone is a bit far away from the base, and the signal shows two lines in this zone. Outside these two zones, the signal shows one line. If the player gets into the danger zones, the signal does not show any signal lines and a danger message about should turn back to the area shows the player. If the player moves and passes the danger zone, it means the player is inside the "endofthemap" zone. The plane shuts down, and the game is over. (Game over message shows to the player as a new scene)

***Picture.4: The Map***

### a) First Signal Zone

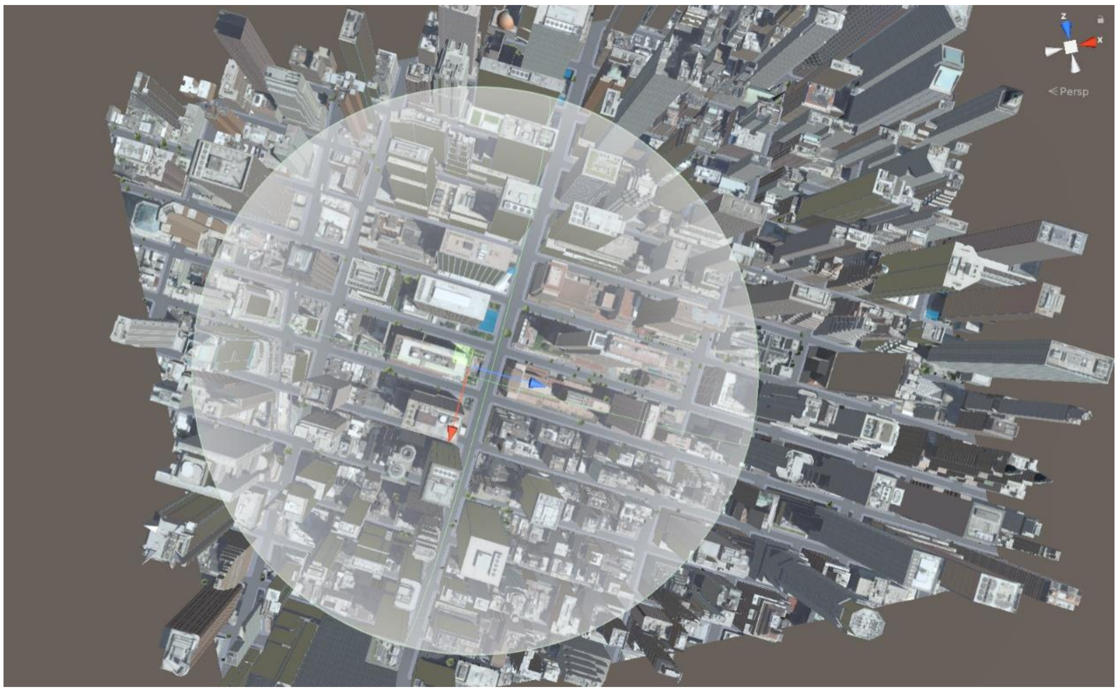
In this zone, the signal line shows three lines. The system triggers when the plane enters or leaves the “collider” zones using the “UnityEvent” system. All zones are “collider” zones for using the signal system. The “isTrigger” boolean variables of all signal zones were assigned a value of “true”. However, for the ready-made asset plane fly scripts, not all colliders are trigger colliders. This situation caused the planes to collide with the collider zones. To address this, I modified the ready-made asset's “SimpleAirPlaneCollider” script for the signal zone colliders. Upon examining the script codes, I noticed that the “collideSomething” boolean variable controls the planes' collision behavior with the colliders. Even if the colliders are set to trigger mode, if this variable is “true”, the plane continues to collide with all colliders except for certain components. These components have been defined by the asset's developer(s), and I added them as exceptions in my signal zones script. Consequently, the signal zones' colliders have been defined as exceptions for collision in the ready-made asset.

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

namespace HeneGames.Airplane
{
    public class SimpleAirPlaneCollider : MonoBehaviour
    {
        public bool collideSometing;

        [HideInInspector]
        public SimpleAirPlaneController controller;

        private void OnTriggerEnter(Collider other)
        {

            if(other.gameObject.GetComponent<SimpleAirPlaneCollide
            r>() == null &&
            other.gameObject.GetComponent<LandingArea>() == null
            && other.gameObject.GetComponent<SignalZones>() ==
            null)
            {
                collideSometing = true;
            }
        }
    }
}
```

In the "SignalZones" script, the collider's trigger when both the plane's colliders inside of them and leaving them. For the first signal zone, when the plane enters the trigger, the activity of the "signal3" canvas game object is set to "true". When the plane leaves the trigger, the activity of the "signal3" canvas game object is set to "false". This way, in the first signal zone, the plane displays the "signal3" canvas object.

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


public class SignalZones : MonoBehaviour
{
    [SerializeField] private GameObject signal1;
    [SerializeField] private GameObject signal2;
    [SerializeField] private GameObject signal3;
    [SerializeField] private GameObject lowSignal;
    [SerializeField] private GameObject turnBack;
    [SerializeField] private GameObject danger;

    [SerializeField] private UnityEvent zoneEnter;
    [SerializeField] private UnityEvent zoneExit;
```

```
void OnTriggerEnter(Collider other)
{
    zoneEnter.Invoke();
}


void OnTriggerExit(Collider other)
{
    zoneExit.Invoke();
}

public void firstZoneEnter(){
    signal3.SetActive(true);
}

public void firstZoneExit(){
    signal3.SetActive(false);
}
```

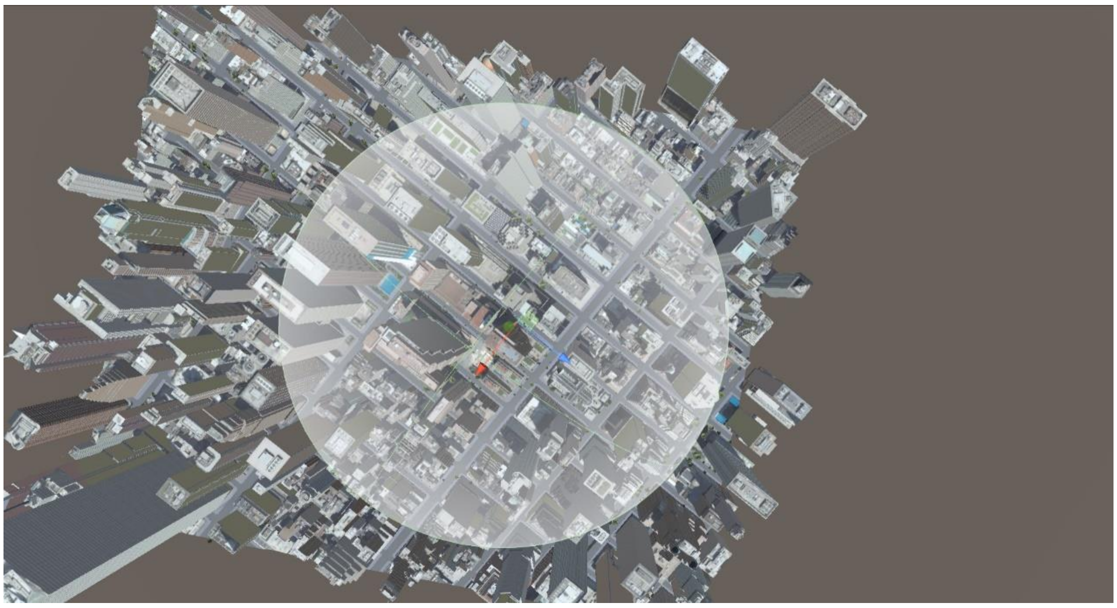


***Picture.5: First Signal Zone***

### b) Second Signal Zone

In this zone, the system shows two signal lines on the “planeCanvas”. For the second signal zone, when the plane enters the trigger, the activity of the “signal2” canvas game object is set to “true”. When the plane leaves the trigger, the activity of the “signal2” canvas game object is set to “false”. This way, in the first signal zone, the plane displays the “signal3” canvas object. But there is an intersection area between the first zone and the second zone. In this intersection area, the system shows three signal lines because the first signal zone does not set the three-signal line canvas object as “false” yet.

```
public void secondZoneEnter(){
    signal2.SetActive(true);
}

public void secondZoneExit(){
    signal2.SetActive(false);
}
```

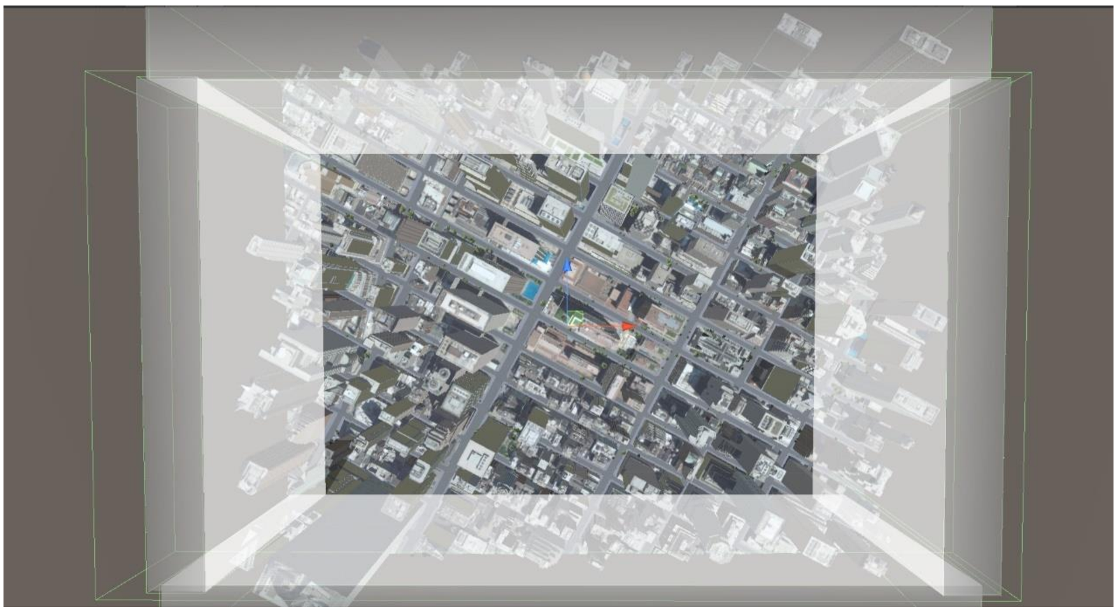

***Picture.6: Second Signal Zone***

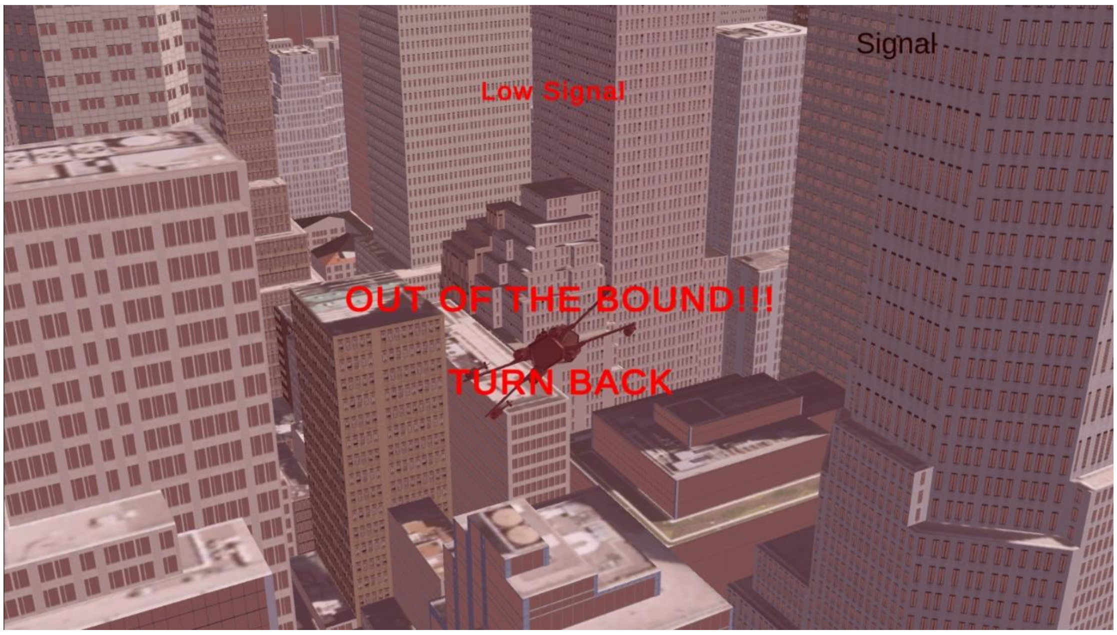
Note: The first signal canvas object is always active until the player gets in the danger zones, so outside the first, second and the danger zones the system shows one signal line to the players.

### c) Danger Zones

These zones are the nearest zones to the end of the map, and the player must not move more away from these zones. When the players get inside of these zones, the system sets all signal lines activity as "false", and it shows three canvas objects for a warning message. These canvas objects include a "low signal" message, a danger message, and a red semitransparent background.

```
    public void dangerZoneEnter(){
        lowSignal.SetActive(true);
        turnBack.SetActive(true);
        danger.SetActive(true);
        signal3.SetActive(false);
        signal2.SetActive(false);
        signal1.SetActive(false);
    }

    public void dangerZoneExit(){
        lowSignal.SetActive(false);
        turnBack.SetActive(false);
        danger.SetActive(false);
        signal3.SetActive(false);
        signal2.SetActive(false);
        signal1.SetActive(true);
    }
}
```


***Picture.7: Danger Signal Zones***



***Picture.8: Danger Signal Zone Message***

### d) Out of the Map

If the player moves more and passes the danger zones. He or She gets would have entered the out of the map zone and the plane starting to fall, and the system change the scene to show a "game is over" message.

GAME OVER !

***Picture.9: Game Over Message***

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.SceneManagement;

public class EndOfTheMap : MonoBehaviour
{

    [SerializeField] private UnityEvent zoneEnter;
    [SerializeField] private UnityEvent zoneExit;
    [SerializeField] private GameObject signal1;
    [SerializeField] private GameObject signal2;
    [SerializeField] private GameObject signal3;

    void OnTriggerEnter(Collider other)
    {
        zoneEnter.Invoke();
    }

    void OnTriggerExit(Collider other)
    {
        zoneExit.Invoke();
    }

    public void outOfZoneEnter(){
        signal1.SetActive(false);
        signal2.SetActive(false);
        signal3.SetActive(false);
        SceneManager.LoadScene(1);
    }
}
```

# 3. Planes

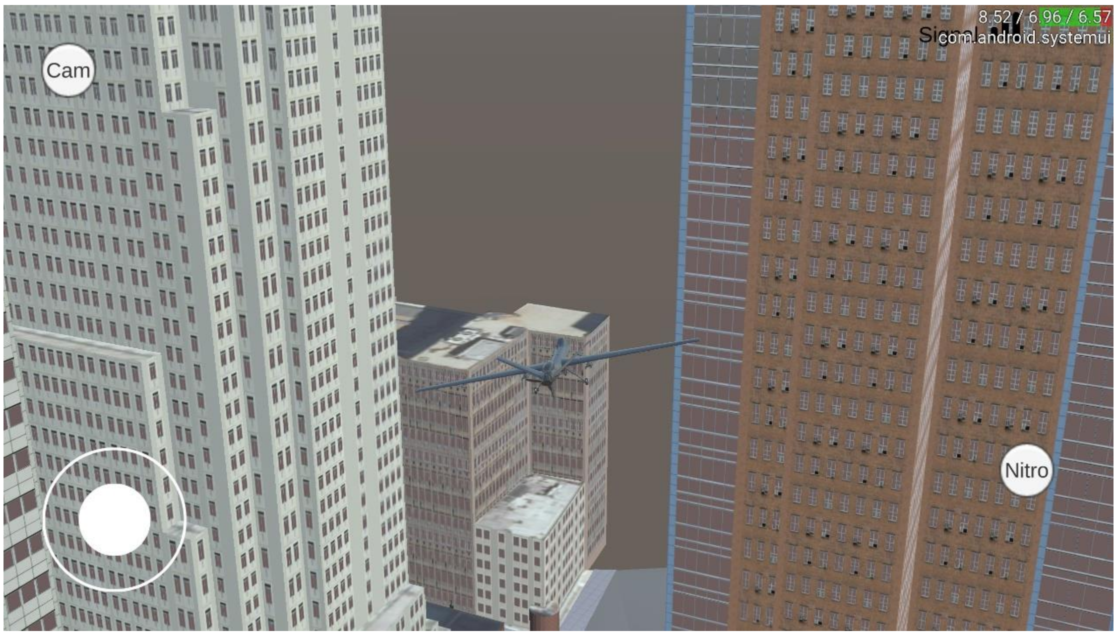
I chose all three-dimensional plane models on the "cgtrader" website *[1]* and I added to my project with resized. As I mentioned before, according to the player's plane selection and camera selection preferences, the system set active the selected plane and camera. I used the ready-made plane asset for plane flying mechanics *[9]* but I changed and modified this asset according to my needs.

The back/free camera is powered by "Cinmemachine" camera asset *[8]* helps to improve camera moving for each plane. In all maps have a main camera with the "Cinemachine Brain" component and each car has "Cinemachine Free Lok Camera" and a follow point. This "Free Look Camera" looks and follows to the "followpoint" element. In this way, the plane moving and the showing of this moving makes it more realistic and intuitive. Also, I used the ready-made plane asset for plane flying mechanics *[9]* as I mentioned before. This asset includes a ready camera script for the planes working with the "Cinemachine" asset. But, for mobile devices, when the player moves the plane with digital joystick or uses the nitro or change camera button, the camera was changed own look position and this situation makes the game unplayable. I developed two scripts for mobile devices. The "MobileCamera" script blocks the "Cinemachine Free Look" camera moving with the touches when the player touches the digital joystick or the buttons. The "JoystickController" checks these touches on the digital joystick and the buttons. If the player touches these components, the system changes "isJoystickActive" boolean variable's value as "true" else its values assign to "false".

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using static JoystickController;
using Cinemachine;

public class MobileCamera : MonoBehaviour
{
    private CinemachineFreeLook freeLookCamera;

    void Start()
    {
        freeLookCamera = GetComponent<CinemachineFreeLook>();
    }
    void Update()
    {
        if (JoystickController.isJoystickActive)
        {
            freeLookCamera.m_XAxis.m_InputAxisName = "";
            freeLookCamera.m_YAxis.m_InputAxisName = "";
        }else{
            freeLookCamera.m_XAxis.m_InputAxisName = "Mouse X";
            freeLookCamera.m_YAxis.m_InputAxisName = "Mouse Y";
        }
    }
}
```

---

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

public class JoystickController : MonoBehaviour,
IPointerDownHandler, IPointerUpHandler
{
    public GameObject nitroButton;
    public GameObject camButton;
    public static bool isJoystickActive;

    public void OnPointerDown(PointerEventData eventData)
    {
        isJoystickActive = true;
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        isJoystickActive = false;
    }

    public void camButtonDown(){
        isJoystickActive = true;
    }

    public void camButtonUp(){
        isJoystickActive = false;
    }

    public void nitroButtonDown(){
        isJoystickActive = true;
    }
    public void nitroButtonUp(){
        isJoystickActive = false;
    }
}
```



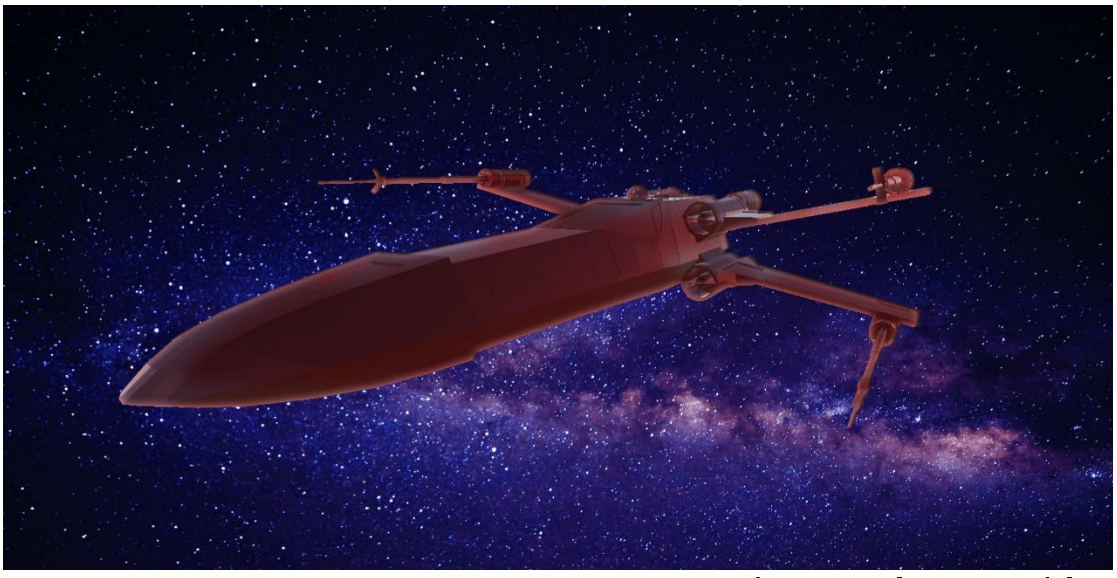
***Picture.10: Android Device Game Play***

***Picture.11: Plane 1 – Spitfire V6***

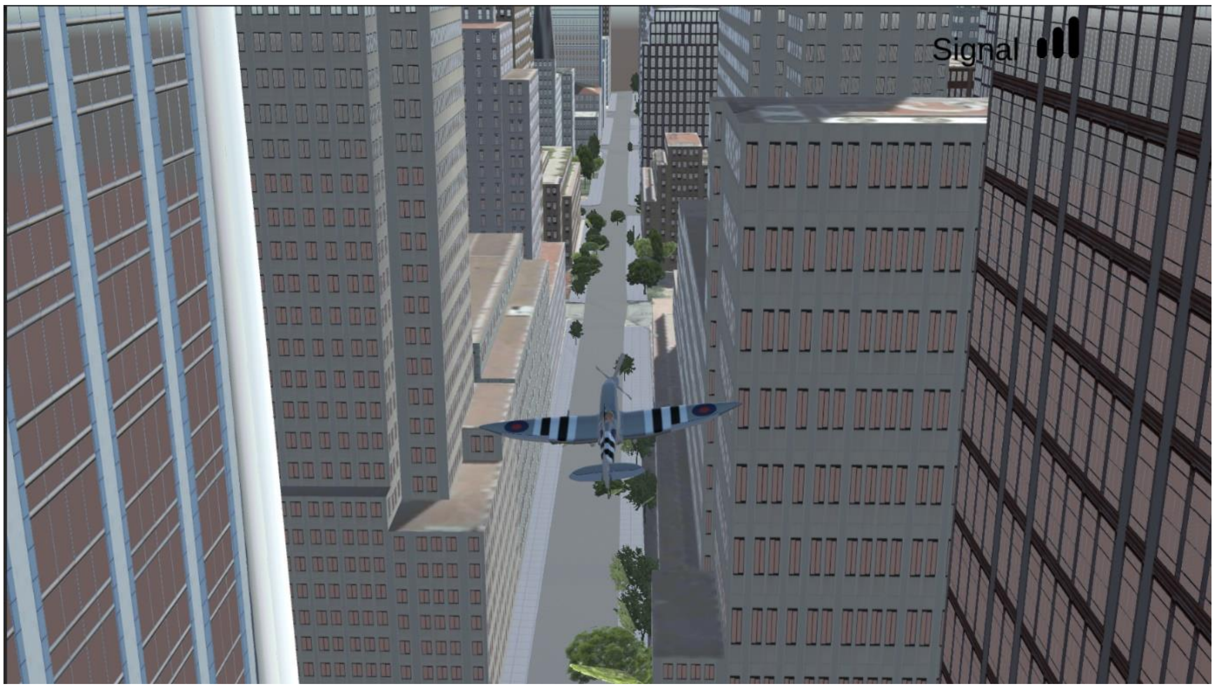
***Picture.12: Plane 2 – Sea Harrier***

***Picture.13: Plane 3 - MQ 9***

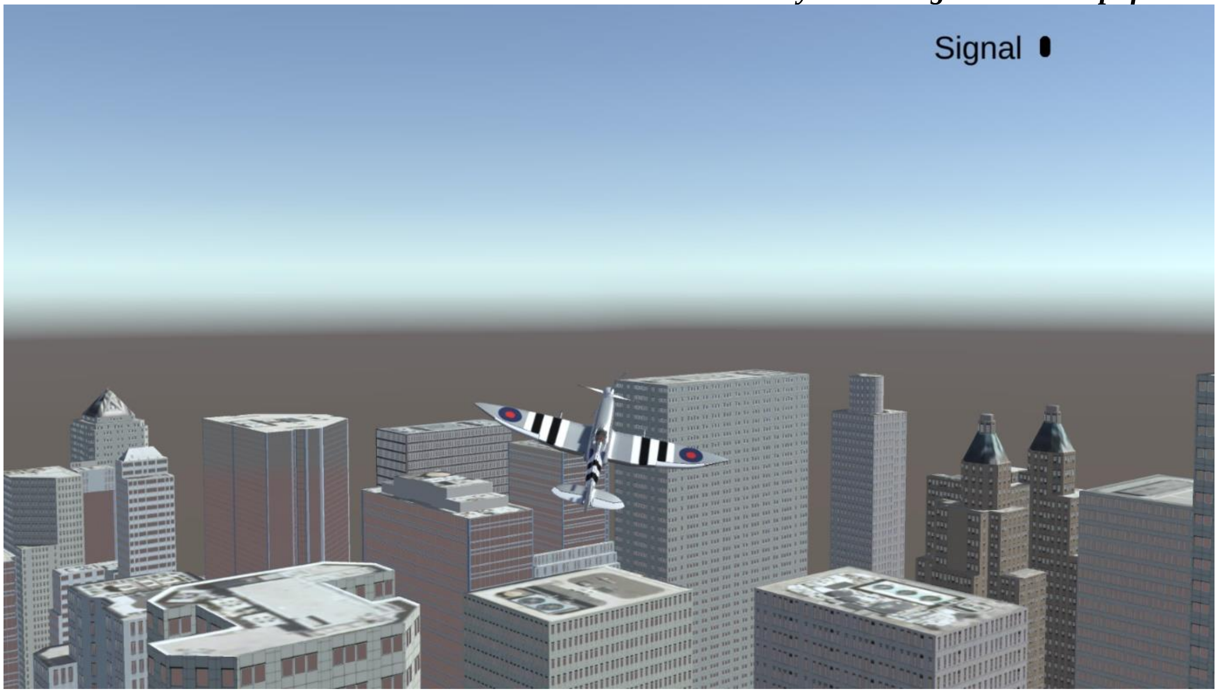
***Picture.14: Plane 4 – Avion***


***Picture.15: Plane 5 - X-Fighter***

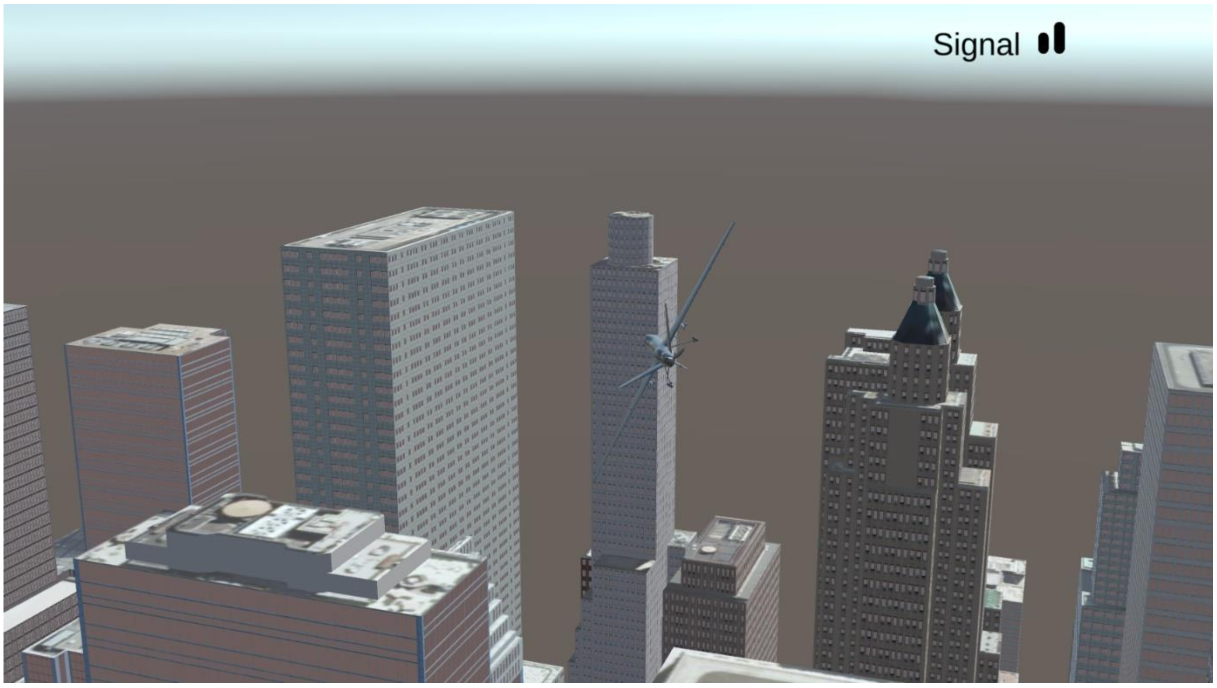
## Screen Shots



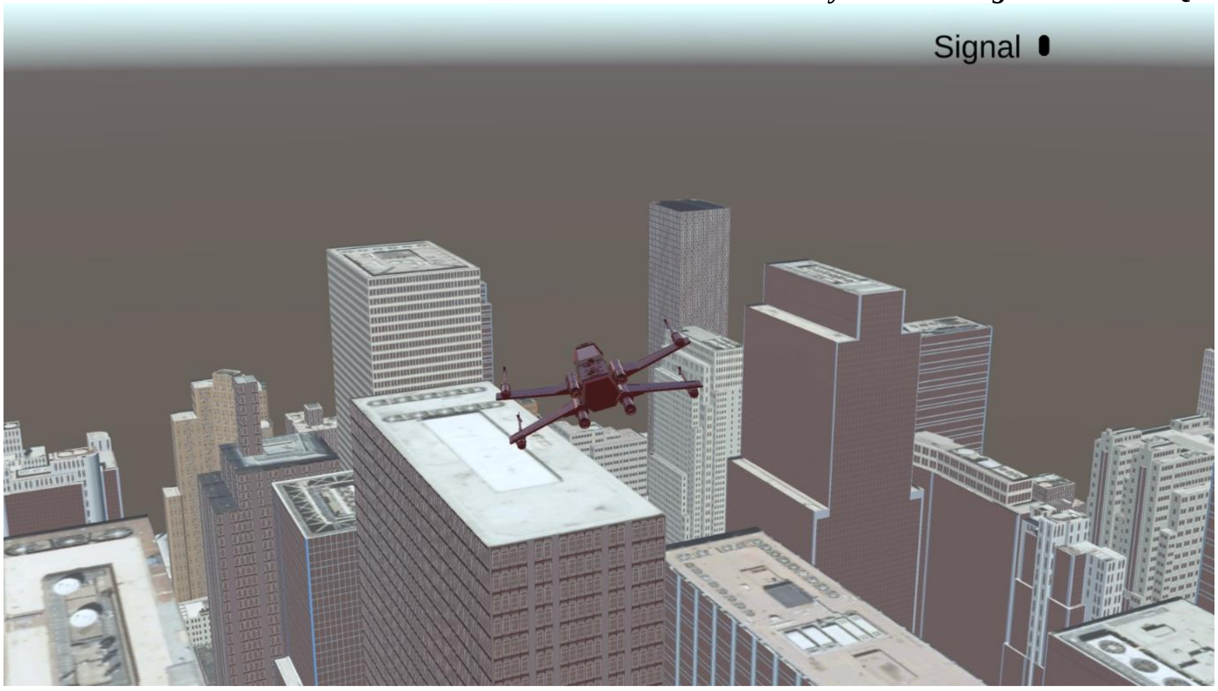
***Picture.16: Game Play – First Signal Zone – Spitfire V6***



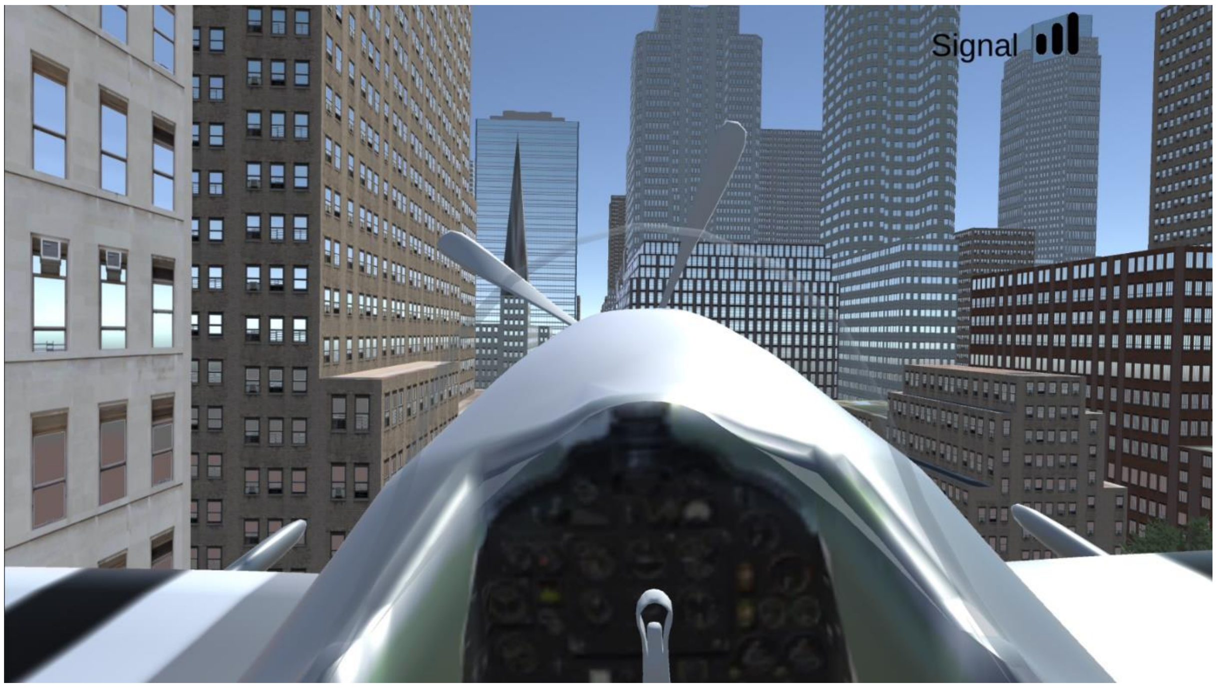
***Picture.17: Game Play – Out of the Signal Zones – Spitfire V6***



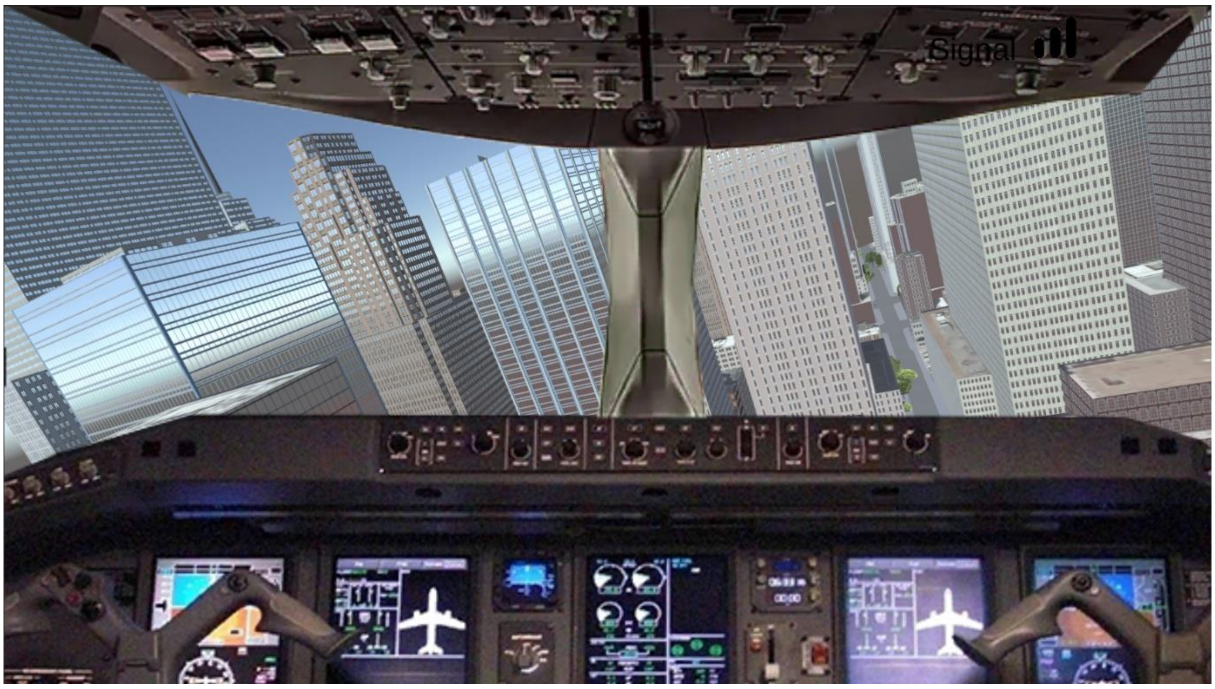
***Picture.18: Game Play – Second Signal Zone – MQ 9***



***Picture.19: Game Play - Out of the Signal Zones – X-Figther***



***Picture.20: Game Play – Inside Camera – Spitfire V6***



***Picture.21: Game Play – Inside Camera – Avion***

## References

1. https://www.cgtrader.com/3d-models
2. https://www.mixamo.com
3. https://www.youtube.com/watch?v=lThBmXYZABA&list=PLsJ3OeB_y-cnesl_v45Jem1p8QC6UGJXE&index=10
4. 91220001906_assignment_3 – report.pdf
5. https://docs.unity3d.com/ScriptReference/Event-keyCode.html
6. https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631
7. https://www.youtube.com/watch?v=Sot6B-EybIs
8. https://docs.unity3d.com/Packages/com.unity.cinemachine@2.9/manual/index.html
9. https://assetstore.unity.com/packages/tools/game-toolkits/simple-airplane-controller-228575
10. https://assetstore.unity.com/packages/3d/environments/roadways/modular-lowpoly-track-roads-free-205188