Virtual, Augmented & Mixed Reality

University of Ioannina, Department of Computer Science & Engineering

Endless Outrun

Contents

Download link:
https://drive.google.com/drive/folders/1u9AnDrJ3glV0idhEYlGaBcfr5btE4J-N?usp=sharing

Quickstart

While using a smartphone, the buttons seem to be misaligned or unresponsive but do work after a few seconds.

Use ALT while moving the mouse around to move the reticle while inside Unity's editor.

The menus are operated using the smartphone's touchscreen, but I have created optional menus that can also be operated with gaze tracking, in the Scenes folder.

Compiling the project to test it on my smartphone takes about 90 seconds so ended up not using Unity Remote.

Colored words in this document and the contents section can be clicked (or ctrl+click) to navigate to the appropriate section quickly.

Introduction

This is an explanatory document, written in order to explain in detail the functions and design of the Android VR application created for this course's assignment. The theme and inspiration for the app is the Synthwave genre of music, a microgenre of electronic music that that draws predominantly from 1980s films, video games, and cartoons. It is also known as Outrun, a term coming from the 1986 driving arcade game "Out Run" by Sega. Since the obstacle avoidance game is meant to go on forever, or until the player loses, I named it *Endless Outrun*.

The game is simple; the player must avoid obstacles and collect coins to improve their score, which is also increased by the total distance travelled. When starting the application, the player is presented with a *Start Menu*, where they can adjust the phone's tilt sensitivity until it is comfortable to control the vehicle, quit, or start the game. The player has 5 lives; colliding with an obstacle removes one life. The obstacles are selected randomly from a pool of five different obstacles, all with a destroyable variant. The player can recognize the destroyable ones by the target they have on them. Moving the reticle on such an object will destroy it. The objects are spawned randomly on set of predetermined spawn points. Coins are also spawned randomly in between obstacles, and improve the player's score when picked up. After losing all their lives, the player is presented with the *Game Over* menu, where they can view their score, readjust the tilt sensitivity, quit the application, or try again.

Assets

3rd Party Assets

The "3rd Party Assets" folder contains asset packages downloaded (for free) from Unity's Asset Store. I have deleted some of the various assets contained in each package that I did not use, in order to decrease the file size. 48 Particle Effect Pack contains particle effects, that I used in the app's particle systems. Azerilo contains several 3D car models, one of which I used as the player's car in the Main Scene. Real Stars Skybox contains images that I used to create the Main Scene's skybox. Most importantly, Google VR SDK contains the files necessary to implement a reticle for gaze tracking and interacting with objects in the scene by looking at them. The folder also contains two audio files, StartCarEngineEffect.wav, which is played at the start of the MainScene, and The Synth Wars.mp3, which is the soundtrack playing on a loop during the MainScene. Both were downloaded from Youtube.

Menu Assets

Menu Assets contains the image used as a background during the Start Menu and Game Over scenes. The same image is implemented as a skybox during the alternate *Start Menu*/*Game*

Over scenes where the user uses the reticle to interact with the UI elements instead of their phone's touch screen. It also includes the fonts used for the UI text elements, Springate and Sulcant, which were downloaded from https://www.1001freefonts.com/.

Obstacles

The Obstacles folder contains the assets that make up the "rays", coins, and obstacles. The obstacles are made of 1x1x1 cubes joined together, in various shapes, joined together. Each cube contains the **#ObstacleDestruction** script, a RigidBody with a very low mass that is set to Kinematic so it doesn't interact with Unity's physics system, with collision detection set to Continuous Speculative which is appropriate for kinematic objects. In addition to all these, the destructible variant of each object also contains the **#ActivateLazerRay** script and an Event Trigger component. The trigger is activated when the reticle is on top of the obstacle, essentially when the player is looking at the obstacle. The trigger activates the scripts responsible for destroying the obstacle and shooting a "ray" from the PlayerCar towards the position of the obstacle. Each cube is painted with a simple Black.mat or Purple.mat material in alternating fashion. The destructible obstacles also use the Target.mat made from the Target.png. Each obstacle variant is saved as prefab for easy instantiation.

The coin prefab is used to instantiate coins on the MainScene. It is a 0.7x0.7x0.7 quad painted with the coin2D.mat made from the coind2D.png. The **#CoinCollision** script is contained within, and whenever a coin is "picked up", the MarioCoinSound.mp3 (found on Youtube) effect is played. Similar to the obstacles, coins are also set to kinematic, with a low mass and Continuous Speculative collision detection.

The LazerRay prefab is used to instantiate the laser rays fired whenever the reticle targets a destructible obstacle. It is a 0.1x-2x0.2 capsule with a deactivated collider, colored with the LazerGreen.mat. It contains the **#MoveTowards** script responsible for moving the capsule towards the destroyed obstacle.

PlayerCar

Contains a PlayerCar prefab of the PlayerCar in the MainScene, as well as the physics material applied to the PlayerCar. More details on the PlayerCar can be found at the **Scenes** section.

Shaders

This folder contains a .shadervariants file used for preloading shaders in the Edit -> Project Settings -> Graphics tab, at the bottom.

Terrain

Inside the Terrain folder are all the assets used to create the ground that the PlayerCar is placed and drives on. The terrain is painted with the Grid.mat, created from the Grid.png which I made using paint3d. The GridMaterial.mat, the terrain asset and the terrain layer are all parts of the Terrain prefab that is instantiated whenever we need to create more ground for the player to drive on.

UI Elements

This folder contains the heart.png, which is a 2D sprite used to represent the PlayerCar's health during the MainScene, and the TiltSensitivity prefab, which is a simple cube container for the **#AdjustTiltSensitivity** script.

Scripts

I prefer keeping my scripts small unless it can't be avoided, instead of cramming everything in a mega script. A lot of the scripts contain only 1-2 methods and a few lines of code.

#ActivateLazerRay

This script is responsible for instantiating a LazerRay prefab and supplying it with the position of the destroyed obstacle that it must move towards to. The script is a component of destructible obstacle prefabs. It is activated by the Even Trigger component of those obstacles, whenever the player moves the reticle on top of the obstacle.

First, the script locates the position of the LazerShooter GameObject by searching for its tag. It is an invisible cube, child of the PlayerCar so it moves along with it, positioned slightly ahead of the PlayerCar. This is the position that the LazerRay will be instantiated, or "fired" from. The Event Trigger specifically activates the shootLazer function, which retrieves the targetPosition variable from the LazerRay prefab and assigns it the position of the obstacle that is getting destroyed. A simple check happens to determine whether the obstacle being fired upon is on the left, right, or in front of the PlayerCar, and the LazerRay is instantiated with the appropriate rotation using Quaternion.Euler. I tried to retrieve the MainCamera's exact rotation to use as the LazerRay's instantiation rotation instead of a 45-degree rotation but couldn't get it to work,

so it looks quite clunky. The script **#MoveTowardsTarget** contained in each LazerRay prefab handles the prefab's movement towards its target.

#AdjustTiltSensitivity

This script allows the user to adjust the speed at which the PlayerCar moves left or right when tilting the phone. In the Awake function, which I used to make sure that this happens before other script activations or other actions, Unity is instructed to not destroy the invisible cube-container of this script whenever a scene is loaded. This way the script's speed variable is preserved between scene changes. The value of the speed variable is adjusted using the adjustSensitivity function, which is attached to a slider at the optionsMenu.

#CoinCollision

This script handles actions related to picking up a coin. First, a coin GameObject must locate GameComponents, an invisible cube which contains the #GameManager and #ScoreTracking scripts. This is necessary since coins don't already exist on the MainScene but are instantiated randomly when the scene is loaded. Thus, it isn't possible to directly add a reference to GameComponents from the Inspector. Whenever a collision with the player is detected, the coin is destroyed, the player's score is increased using **#ScoreTracking**, and the MarioCoinSound.mp3 is played using **#GameManager**.

#ConstantPlayerMovement

This script is responsible for the PlayerCar's movement along the Z axis. The PlayerCar's rotation is disabled using freezeRotation on its RigidBody to prevent any unintended movement. The PlayerCar's speed is set to 25. The current speed can be monitored at the Inspector during gameplay thanks to the public currentSpeed variable. The PlayerCar is moved forward by increasing its current position at the Z axis by constantSpeed, also multiplied by Time.deltaTime in order to be framerate independent.

Initially, I wanted to use Unity's physics system to move the PlayerCar, by modifying its velocity, and applying a braking or accelerating force to keep it at the same level. I have left this code in comments.

#CreateParticlesOnCollision

This is simple script attached to the PlayerCar. It activates the Particle System that is placed at the front of the PlayerCar whenever a collision is detected thanks to the OnCollisionEnter function.

#GameManager

The GameManager script handles core functions of the application. All the obstacle prefabs, both variants are referenced here, and added to an obstaclesList, as well as the coin prefab and an AudioSource that contains the coin sound effect. Each terrain prefab has a length of 1000 units along the Z axis and 50 units along the X axis. The first terrain is already placed on the scene at (-25,0,0). Two invisible walls are placed at (-10,0,0) and (10,0,0), with a length of 1000 units similar to the terrain. Obstacles need to be placed in between (-10,10) along the X axis, and at (0,1000) for the first terrain, then (1000,2000) for the second terrain and so on. Coins also need to be placed similarly. The current terrain that we are operating on is tracked with the terrainN variable. When the MainScene is loaded and the script is activated, the terrainN is initialized at 0 and the functions responsible for spawning obstacles and coins are called, with it as argument.

spawnObstacles uses a for-loop to spawn two randomly selected obstacles at one of three randomly selected spawn points. The spawn points are (-7,0), (0,7), (-7,7) at the X axis, with a simple check to make sure the third one doesn't happen back-to-back. The location on the Z axis is calculated using the i value that is running the for-loop and adding to it an offset of 1000 units multiplied by the current terrain number. Obstacle placement starts at 40 units on the Z axis to give the player some time to get used at the game before presenting them with obstacles.

spawnCoins works in a similar manner, placing coins in between obstacles, since coins start their placement at 30 along the Z axis, and are placed randomly at (-8,8) along the X axis.

In the Update function, we keep track of the PlayerCar's current position, and whether the user presses the X button at the top left of their touchscreen, which translates to an Escape button press. If the X is pressed, the application is terminated. If the player passes the halfway mark of the terrainN they are currently on, terrainN is increased by one, a new terrain is spawned using terrainN in order to calculate the offset needed, and the spawnObstacles and spawnCoins functions are called to populate it with obstacles and coins.

playCoinSound is called whenever the PlayerCar collides with a coin. In hindsight, I could have probably just added an AudioSource with the coin sound in the coin prefab.

#GetStatistics

This script retrieves the latest score and number of coins picked up, in order to display them at the GameOverMenu scene. It searches for the “Statistics” GameObject, which isn’t destroyed on scene load, and contains statistics from the MainScene.

#GyroControlsXAxis

This script implements the PlayerCar’s movement along the X axis when the user tilts their phone on either side. The default speed is 25 units, which the user can change using the slider in the OptionsMenu and the **#AdjustTiltSensitivity** script. In the Start function, the script first searches for the TiltSensitivity GameObject that contains the AdjustTiltSensitivity script and retrieves the speed set by the user. In FixedUpdate, the PlayerCar’s RigidBody position is updated with the input from the phone’s accelerometer.

#HealthManagement

This script keeps track of the player’s health, updates the UI accordingly, and loads the next scene whenever the player runs out of health. The demonstration variable can be deactivated so the MainScene doesn’t terminate when the player runs out of health (for testing purposes). The public Image variables have the heart.png sprite assigned to them from the Inspector.

Whenever a collision is detected, the decreaseHealth function is used to decrease the player’s health and hide one of the heart sprites. Since an obstacle is made of multiple cubes, colliding with one obstacle can register more than one collision and remove more than one heart. To avoid that, the last Z axis coordinate of the last obstacle hit is saved in the lastObstacleZhit variable (all cubes that make up an obstacle have the same Z coordinate), so only one collision is registered (in terms of health removal).

The increaseHealth function can be used to increase the player’s health, but ultimately, I decided against implementing such a mechanic.

#MainMenuManager

This script is attached to the Canvas in the StartMenu and GameOver menu and handles button input from the player. startGame loads the next scene, while quitGame terminates the application.

#MoveTowardsTarget

This script is a component of every Lazer Ray prefab. When a Lazer Ray prefab is instantiated by the **#ActivateLazerRay** script, it is provided with the target position. In the Update method, the "ray" moves towards the target position using Unity's MoveTowards function. It is important that the speed is higher than the PlayerCar's speed so that the ray doesn't fall behind. When the ray reaches the position of the destroyed obstacle it is destroyed.

#ObstacleDestruction

This script is attached to every obstacle, and every cube that makes up that obstacle. It has a reference to the GameComponents GameObject so it can find and access the #GameManager script, so it can use the decreaseHealth function to decrease the player's health. Collisions are detected thanks to the OnCollisionEnter function. When the player collides with a GameObject that contains this script, the GameObject is destroyed. Destroyable objects have the "ReticleDestructible" tag, which means they'll be destroyed by the destroyOnPointerEnter function when the player moves the reticle on top of them.

#QuitWithX

This script enables the player to also quite during the StartMenu and GameOver scenes. It monitors whether the player presses the Escape key in the Update function and terminates the application when they do.

#ScoreTracking

This script keeps track of the player's score and updates the relevant UI elements. It keeps track of the PlayerCar's position, as the player's score increases the further the PlayerCar travels. The font previously used in the rest of the UI, Springate, couldn't display numbers so I had to use TextMesh's Pro default font to display the actual score numbers. The script keeps track of the PlayerCar's position in the Update function, it formats it accordingly, and increases it per coin picked up through the increaseScore function. The score and the number of coins collected are then sent to the #trackStatistics script where they are "saved".

#trackStatistics

This script is where the score and the number of coins collected are saved. It is updated by the **#ScoreTracking** script. Since it isn't destroyed on scene load, this information is available at the next scene.

Scenes

StartMenu

This is the first scene that the player is presented with when starting the application. The MainCamera is looking at a Canvas, which is set to Screen Space – Camera mode and contains all UI elements. Text Mesh Pro text objects are used to create the text of the game's title and the UI buttons.

Pressing the Play button will load the next scene in the hierarchy (1) while pressing Quit will terminate the application. Pressing the Options button will deactivate the MainMenu panel and activate the OptionsMenu panel, where the user can adjust the speed with which the PlayerCar moves along the X axis when the user tilts their phone. The value is changed in the **#AdjustTiltSensitivity** which is placed in an invisible cube, TiltSensitivity. Pressing Back deactivates the OptionsMenu panel and activates the MainManu panel, returning the user to the main menu.

QuitGame is an invisible cube that contains the **#QuitWithX** script which allows the user to also quit by pressing the X at the top left of their screen.

StartMenuWithReticle

This is a copy of the StartMenu scene, but uses prefabs from Google's VR SDK to implement gaze tracking with a reticle. Specifically, the GvrEditorEmulator, GvrEventSystem and GvrReticlePointer are used. GvrEditorEmulator allows the user to move the camera around with the mouse while pressing the ALT or CTRL key. GvrEventSystem handles the events generated by the reticle, while GvrReticlePointer implements the actual reticle. These prefabs allow the user to select buttons by moving the reticle on top of them, by adding an EventTrigger component which is activated on PointerEnter. However, I couldn't get the slider to work with reticle input.

The background image is disabled for this scene. A skybox that uses the background image for all of its 6 sides is used instead.

This is scene exists in the Scenes folder but not in the scene hierarchy. It needs to replace the StartMenu scene, at File -> Build Settings -> Scenes In Build, at index 0.

GameOverMenu

This scene is presented when the player runs out of health. It works in the same manner as the StartMenu Scene. Two text fields that display the score and the coins collected at the previous attempt are also included. The text fields retrieve the score using the **#GetStatistics** script.

GameOverMenuWithReticle

This scene works similarly to the **GameOverMenu** and **StartMenuWithReticle** scenes.

MainScene

This is the main scene of the application, where the actual game is played. A terrain object is placed at (-25,0,0). Its length is 1000 units (along the Z axis) and its width is 50 units (along the X axis). The playable area is enclosed by two cubes, with the same length as the terrain, that have very large mass and form two invisible walls. The walls are placed at (-10,0,0) and (10,0,0). The terrain area outside the playable area is raised and lowered using various terrain brushes.

The PlayerCar is placed at (0,0,3). It is created using a 3rd party asset pack, **Azerillo**. I've edited the car's plate to my initials in English (AV) and my student ID (2651). The PlayerCar's RigidBody has a mass of 15, very low drag and angular drag, with Continuous Speculative collision detection. It contains the #ConstantPlayerMovement, #GyroControlsXAxis, #CreateParticlesOnCollision scripts.

A Particle System, child of the PlayerCar so it moves along with it, is placed right around where the engine is at the front of the vehicle, and is responsible for producing the particle effects on collision.

LazerShooter is an invisible cube, child of the PlayerCar, and the origin point of "lazer rays", created by the **#ActivateLazerRay** script.

The MainCamera is a child of the PlayerCar, and the reticle, implemented by GvrReticlePointer, a child of the MainCamera. A canvas containing the UI elements regarding health and score is also a child of the MainCamera.

Inside the scene are also two invisible cubes. GameComponents which contains the **#GameManager**, **#HealthManagement**, **#ScoreTracking** scripts and three AudioSources that contain the coin pick-up sound, the soundtrack playing on a loop and a car engine starting sound effect. StatisticsTracker contains **#TrackStatistics**. It is a separate object, as it persists through scene loading.

Finally, two prefabs from Google's VR SDK, GvrEventSystem and GvrEditorEmulator also exist in the scene to implement gaze tracking with the reticle.

Bugs

Pressing the UI buttons while running the application on a smartphone is inconsistent. Sometimes it seems like the button's "hitbox" is in a slightly different position although visually

its in the same position. Or maybe the application is lagging, and it takes a few seconds to register/activate input ? I spend a whole day trying to diagnose what was happening but didn't manage to understand why it isn't working as expected.

Sometimes while running the application in Unity's editor, the mouse cursor will be "trapped" in the application and I need to press the Escape button to free it.

After I started using DontDestroyOnLoad to preserve some GameObjects and retrieve their variables later, I sometimes get an error like this: "Serialized object target has been destroyed".

Conclusion

Initially I was planning on creating a more complicated game, but I quickly realized that I would need to create my own assets, since I couldn't find everything I wanted available for free. As I don't have any experience with 3D modelling software, I settled for making what is essentially a reskin of the original game with small tweaks. Even though I started about 2 weeks before the deadline, I was quite anxious about missing it, especially since I would be away from my PC for a few days due to Easter holidays.

I'm not sure how enjoyable and fluid the game feels like, as I do not own a Google Cardboard headset. I usually never use tilt/accelerometer controls in mobile games as I find them too imprecise. While playing the game on my smartphone, I had a hard time controlling the car while also looking around to target obstacles with the reticle. It would have been nice if the player had some kind of different method of input or game pad while playing the game, to implement a proper shooting/ammo system, jumping and sliding. I tried to create a jumping action whenever the player looked down, but I didn't manage to make it work correctly. Sometimes the player would jump while trying to target obstacles far ahead, or it wouldn't be reliable enough as the player would have to look too far downwards.