Run 4 Freedom
Final Project for the Interactive Graphics course

Ivan Fardin 1747864
Samuele Olivieri Pennesi 1753295
Francesco Ottaviani 1759720

September 27th, 2020

Contents

1 Abstract

This document represents a technical report and a user manual about the *Run 4 Freedom* game, made by three students of the MSc of *Engineering in Computer Science at Sapienza University of Rome*.

Run 4 Freedom is an endless runner game developed as part of the *Interactive Graphics 19-20* course. It immerses the player in an escape scenario in which he/she has to run as fast as possible by driving a car or moving his/her legs in order to avoid being arrested by the pursuing cops.

2 Environment development

For developing this project we used ***Three.js***[1], a JavaScript 3D library with the aim to create an easy to use, lightweight, 3D library with a default *WebGL* renderer.

WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on *OpenGL ES*, exposed to ECMAScript via the HTML5 Canvas element.

So, *Three.js* is a cross-browser Javascript library used to create and display animated 3D computer graphics in a web browser with less effort than the "basic" *WebGL*.

In order to show a performance monitor, we used the *stats.js* module while to handle the settings menu during the gameplay the *dat.GUI* both offered by *Three.js*.

3 Physics engine

A core component of a game is its physics engine that attempts to simulate real world physics.

There are several Javascript physics engines such as *Physijs*, *Cannon.js*, *Oimojs* and *Ammo.js* but we decided to use ***Ammo.js*** (a porting of the famous and used *Bullet* library) because it is constantly updated and has better performance than the others even if perhaps it's a little more complicated than the others.

[1] https://threejs.org/

So, with the physics engine we are able to move our objects as in the real world, detect and perform collisions among them.

Its implementation in the game started with the creation of the physics world that consists of the setting of gravity and the algorithms to use for rigid bodies simulation and collisions simulation and detection. Then, we bound each 3D animable model of the game with a **box** that represents its **mass** and **volume**. Each box size has been fine-tuned via a long manual process to better feature the model's volume while the mass value of each model was taken from the web to be aligned with the real world. Bounding boxes are also used for the terrain in order to place 3D models on it and for the new jersey barrier 3D model to be consistent with the logic of the game.

The relationship between the physics world and the 3D graphics world is indeed represented by the bounding box of each model. This means that at every tick of the physics world, each 3D model is drawn according to its bounding box position that represents its mass and volume.

Collisions between the 3D models are set using a collision filtering configuration and are detected via a predefined *Ammo.js* function.

Every rigid body has a bitwise mask collision group and collision mask. The **collision group** represents the collision identity group of the rigid body while the **collision mask** represents other collision identity groups that should be collided with. So, let us assume we have two bodies A and B, collision between them can only occur if a bitwise AND operation between the collision mask of A and the collision group of B is anything but zero and viceversa.

Since we are particularly interested in the contact between the police model and the player one, we check for that at every tick of the physical world after setting up a **collision pair** callback.

Last but not least use of *Ammo.js* is the setting of **lateral borders** in order to limit user movement on the road and not off-road because otherwise, the player could have exploited it to "cheat", going always straight without running into obstacles. Like for the terrain, borders are implemented via invisible boxes.

4 User interaction

The main component of a game is of course the user interaction and in order to implement that we followed the principle that it must be **easy to use**.

After this initial interaction, the user starts to play and control his/her selected model via keyboard arrows or wasd.

These inputs are handled by an Input Manager that receives events according to the keyboard pressure.

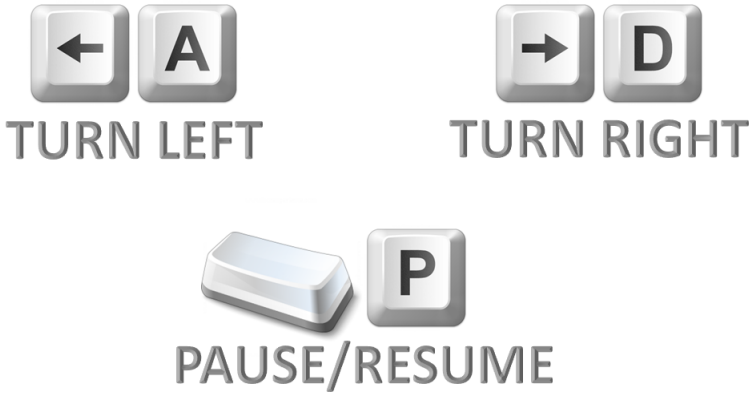
In each moment while the user is playing, he/she can pause it via spacebar and/or change some game settings via a panel: the audio, the intensity, and the position of the sunlight.

5 Main Menu

The first user interaction occurs in the main menu in which via an easy user interface the user selects in which **scenario** to play and the **car**.

For each scenario there is the possibility to choose between 4 different cars.

In addition to the management of game components, the user can click on the tutorial of the game before starting to play, to know the driving controls and the commands to pause the game.



As you can see in the image, there is **no button to accelerate or slow down by design**. This is part of the idea of making the controls easier

and of the difficulty of the game that grows as you play by speeding up the player's model without the ability to brake or decelerate. The speed of the player's car raises automatically second after second, until it reaches a maximum limit. Of course, if we didn't set a limit for the player's speed, the game would have become unplayable, because higher is the speed, more difficult is to avoid the obstacle cars.

When the user has made all his choices, and therefore all the game components have been set, the play button is finally shown to allow the user to start the game.

6 Game

Once the scenario and the car are chosen through the main menu, only the corresponding 3D models are loaded to reduce the initial loading time (e.g., if the *Country desert scenario* is chosen, buildings like the stadium or Starbucks are not loaded, because they are set to appear in the *City* scenario).

After the static buildings are loaded, the player can press the play button, and in this moment dynamic models are loaded. The user is immersed in an escape run and starts playing by controlling his/her selected model via *keyboard arrows* or *WASD keys* (see image above).

The goal of the game is to **avoid obstacles** (whatever is the chosen scenario) and avoid being reached by the pursuing police to achieve the longest possible escape.

For **performance** reasons, it was not possible to keep a large number of models inside the scene, so we decided to translate surpassed models by the player, instead of adding new ones. Basically, when the player goes beyond a certain 3D model, this one is shifted of a constant value and becomes visible only when the player is near "enough" to it.

In order to achieve a good **balance between the variety of the landscape and performance**, we selected a small range of 3D models based on the number of polygons and MB size.

Apart from physics, 3D models are treated differently, according to the group they belong to (they can be static or dynamic).

Static models (all buildings and road elements) are simply shifted of a constant value when the player goes beyond them. This means that the environment around the player is repeated over and over (fixed pattern).

Dynamic models (all cars except the one controlled by the player) are not only shifted like the previous ones, but also they are given in a random

X coordinate among predefined ones, so that they **are not respawned always in the same lane**. This makes the game less repetitive, and, of course, more difficult.

The possible coordinates in which the cars can spawn are different according to the chosen scenario. For example, in the *Country desert*, cars can spawn only in two different X coordinates, because we have a total of two lanes for the road, while in the city there are 4 different X coordinates, because we have a total of 4 lanes (2 for each direction of travel).

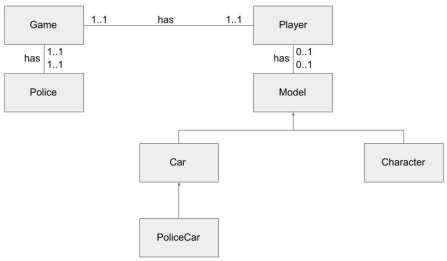
The car controlled by the player and the police car are exceptions: they spawn always in the same position.

The X coordinates have been chosen with attention: they are "near enough" among them, so that the player can't stay between two of them without running into an obstacle car.

Since the scenarios have a different number of lanes, we decided to use more cars as obstacles in the Highway and a lower number for *City* and *Country desert*. An excessive number of cars in an environment like *Country desert*, for example, would have made the game unplayable, simply too difficult, because the player wouldn't have enough space to move between the obstacles and go on.

The velocity is the same for all the obstacles cars, and it is intentionally much lower than the max speed which the player's car can reach.

These inputs are handled by an Input Manager that receives events according to the keyboard pressure.



7 Other menus

In each moment while the user is playing, he/she can pause it by pressing the spacebar or P keys (see image above) and/or change the audio of the game via a default panel in the top right.

The **pause menu** contains two buttons to resume the game (restarting the scene that was blocked until now) and return to the main menu. The user can resume the game also by pressing the same keyboard key used to

pause it.

When the player is caught by police, the **game over menu** is shown. It's very similar to the pause one, but it also has the information about the final score of the current game played by the user and the highest score of all games. Eventually, there are two buttons to restart immediately with the same game components and to return to the main menu.

8 Lights

In the game there is just a type of light: directional and it's used to reproduce the sun.

The **daylight** has been implemented to **change during the day** according to the daytime. In order to realize that, we assume each hour lasts a few seconds and light intensity and position is relative to the daytime. In addition, also the **sky color** changes according to the daytime by setting the *WebGLRenderer* color using a different gradient of the sky color.

In this way, there is a good alternance between day and night that makes the game more involving and realistic.

9 Textures

Textures are ubiquitous in the game. Each 3D model has at least one that is implicitly loaded with the Gltf model. We customized some 3D models textures about billboards and ads models.

An explicit usage of the textures has been used to realize the infinite terrain and road (over which the player runs), the clouds and the road crosswalks.

10 Game mechanics

Whatever environment is chosen, the player has to avoid the obstacles. Due to performance, it was not possible to maintain a high number of models inside the scene, and this led us to translate models, instead of adding new ones. Basically, when the player goes beyond a certain model, this one is shifted of a constant value. Also, a model becomes visible only when the player is near "enough" to it. Again, this choice was imposed from the performance issue.

Apart from physics, the obstacles are treated differently, according to the group they belong to (they can be static or dynamic). Static models are

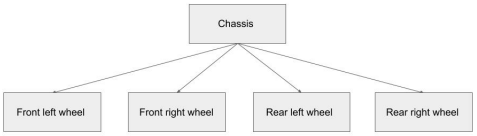
simply shifted of a constant value when the player goes beyond them. This means that the environment around the player is repeated over and over. When we talk about dynamic obstacles, we mean the cars which are not controlled by the player. They are not only shifted like the buildings, but also they are given a random x coordinate among predefined ones, so that they are not respawned always in the same lane. This makes the game less repetitive, and, for sure, adds a bit more difficulty. The possible coordinates in which the cars can spawn are different according to the environment chosen. For example, in the desert environment cars can spawn only in two different x coordinates, because we have two lanes, while in the city there are 4 different x coordinates, because we have 4 lanes (2 for direction of travel). The car controlled by the player and the police car are exceptions: they spawn always in the same position. The x coordinates have been chosen with attention: they are "near enough" among them, so that the player can't stay between two of them without running into an obstacle car.

Once the environment and the car/character are chosen through the main menu, only the corresponding models are loaded. For example, if the environment "Country" is chosen, buildings like the stadium or starbucks are not loaded, because they are set to appear in the City environment. After the static buildings are loaded, the player can press the play button, and in this moment dynamic models are loaded. Since the environments have a different number of lanes, we decided to use more cars as obstacles in the highway and a lower number for city and country. An excessive number of cars in an environment like country, for example, would have made the game unplayable, simply too difficult, because the player wouldn't have enough space to move between the obstacles and go on.

The velocity is the same for all the obstacles cars, and it is intentionally much lower than the max speed which the player's car can reach.

11 Hierarchical models

In the game almost every 3D model is a hierarchical one. The models we focused on were both the ones animated by the player and the AI. The car hier-



archical model has the following structure

12 Animation

The 3D model animations exploit the structure of the hierarchical models. So, for moving a car, we don't simply apply a force to it but we also rotate its wheels according to current object speed. It's important to underline that we initially found a lot of car models, but in many of them, the references for the 4 wheels were absent (they were probably designed to be static models), or in other cases there was just a reference for all the wheels. We discarded this last category of cars too in order to exploit as much as possible the movement of each single wheel.

13 3D models

All models in the game are taken from Sketchfab(https://sketchfab.com/) and are free. We performed a long research to find appropriate models with the game context and that are lightweight (in terms of polygons and Mb size) at the same time. We found different type of models according to the environments used in the game: city, desert, highway. The most complex scenario is the city, and a great effort was made in order to create a good looking environment with both buildings (like houses, shops, offices and stadium) and street elements, like vertical drive signs, dumpsters, and traffic lights). We also manually edited some models, like the sidewalk (city environment) and the jersey (highway environment), to extend them on the z axis. In this way, instead of cloning them n times, we just keep 2 copies of the model and when the players surpass the first copy, we translate it right after the second one. The goal was to obtain the effect of the "never ending" element.

14 Sounds

All the sounds of the game are taken from youtube videos with free license of use.

15 Link to used models

Buildings

City

building1 3.7k tri
- office 1.7k tri
- la cantine 867 tri
- boulangerie 604 tri
- starbucks 1.5k tri
- stadium 1.9k tri
- arena 898 tri
- coffee 1.6k tri
- pizza 100 tri
- cotton club 3.2k tri
- office2 863 tri
- apartment 1.4k tri
- house 508 tri
- dumpster 1.1k tri
- bench 2.9k tri
- trash can 132 tri
- sidewalk 503 tri
- menu
- stop sign 100 tri
- speed sign 30 9 tri
- tree 1.3k tri
- fence 1.3k tri
- traffic light 1.7k tri
- road sign 104 tri
- table 362 tri

Highway

fence 1.3k tri
- jersey barrier 804 tri
- bilboard ad 420 tri

Desert / Country

stores 712 tri
- wooden house
- abandoned shopmall 1.9k tri
- motel 1.6k tri
- cinema 712 tri
- house 1k tri
- old rusty car 2.5k tri
- bush 472 tri
- bus stop 684 tri
- thumbleweed 600 tri

Cars

bmw i8 18.4k tri
- lamborghini aventador 61.8k tri
- tesla model s 18.2k tri
- audi r8 25.3k tri
- bmw e30 21.7k tri
- range rover 40k tri
- low poly small car 7.3k tri
- old version taxi 1.2 k
- low poly car 6.4k tri
- dodge viper 3.2k tri
- nissan gt 50k tri
- police car 15.7k