Final Project Interactive Graphics
"OPOLI's Drone Adventures"

Giacomo Colizzi Coin 1794538

Lorenzo Diaco 1796658

Matteo Germano 1807599

July 2021

Contents

1 Introduction

The game we have designed and implemented is a one player game. It is set in a futuristic city and the player controls directly a drone, by moving it in the 3D environment. The drone can naturally move forward and backward, to the left and to the right, it can rotate counterclockwise and clockwise and it can rise or descend. The goal consists in the collection of coins up to the threshold (this number varies based on the difficulty) as fast as possible, before the depletion of the lives of the drone (initially, the drone has 3 lives). A life is lost when the drone hits a wall or when the fuel bar drops to 0 (the left red zone indicates that the player is running out of fuel). During the game it is possible to collect also fuel tanks, to fill partially the fuel bar, and hearts, used in order to gain a life for the drone.

2 Libraries used

For a good realization of the project we have used three main libraries, **Three.js**, **Tween.js** and **Ammo.js**. Three.js is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL. It allows the creation of graphical processing unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. Tween.js is a JavaScript tweening engine for easy animations, incorporating optimised Robert Penner's equations. It supports tweening of both numeric object properties and CSS style properties. The API is simple but very powerful, making it easy to create complex tweens by chaining commands. We have used Ammo.js as physical engine, to make more realistic and accurate some behaviors in the game. Ammo.js is a direct port of the Bullet physics engine to JavaScript, using Emscripten. In further detail, we used ENABLE3D, which is an easy implementation that merges Three.js and Ammo.js. In particular we handle collisions with this powerful library. It is very important for the correct implementation of the project because, if the drone collides with something that is not a consumable while having a high speed, it must lose a life. Moreover, also the spawn and collection of coins, hearts and fuel tanks is handled via the physical engine.

3 User Interface

As we can see in Figure 1, the game starts with this interface. The player can choose the color of the drone and the color of the propellers between 5 different colors. The game has three levels of difficulty. The change of difficulty modifies the speed of depletion of the fuel bar, and also the number of coins to be collected in order to win the game. The player can also choose the time of the day between day, afternoon and night and the drone type between the two visible in Figure 1. The change of day time makes possible to use three different environments in the game scene. The models of the two drones are analyzed in a successive section of the report.

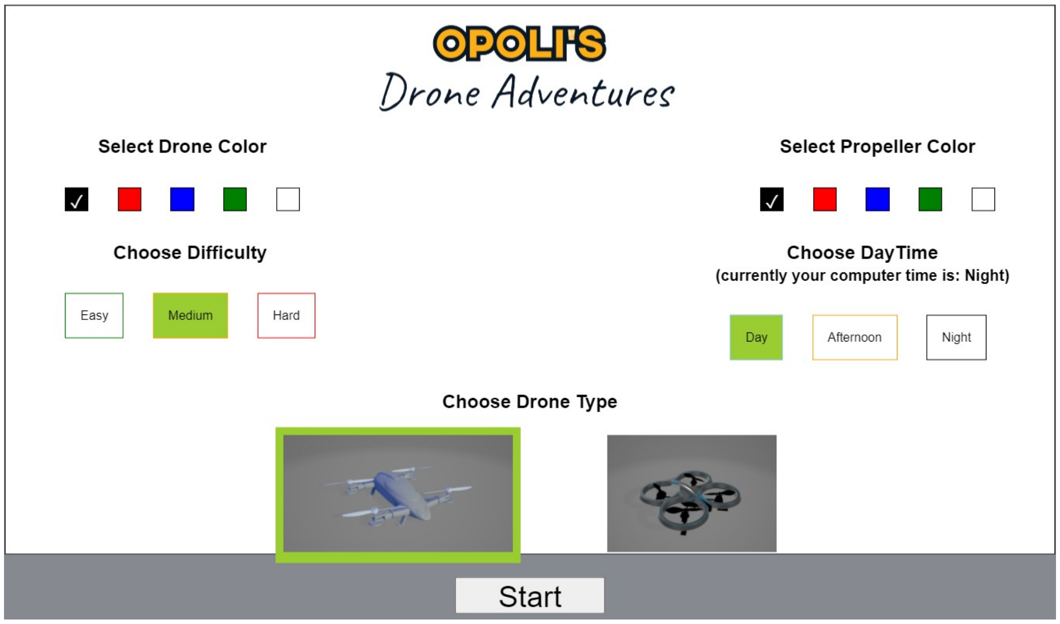

Figure 1: Starting game interface

The start button lets the game begin, along with the music. The game interface is very simple. In the upper right some notifies appear to inform the player about relevant events in the game. To handle these simple messages we have used the javascript library Noty.js, a dependency-free notification library. In the lower right part we set the fuel indicator, by using the javascript library Gauge.js, while in the lower left part of the screen there are all the other relevant buttons and information. Between these ones we can see the current Frames per second (FPS), the time of flight in seconds, the current number of lives of the drone (counted with hearts), the current level of difficulty of the game and last but not least the number of coins collected by the drone. Under these information we have a button for the various sounds of the game, an indicator for the current weather. Once the pointer is locked into the game, only the "ESC" button can make the pointer unlock, and clicking the joystick helper button makes the player re-enter into the game. If the pointer is locked, there is a custom third person camera that can be moved via the mouse in order to follow the drone. It is not influenced by the drone's rotations, but only by its position. In figure 2 we have a preview of the commands page:

- "W" for pitch forward
- "S" for pitch backward
- "A" for yaw left translations
- "D" for yaw right translations
- "Q" for roll counterclockwise
- "E" for roll clockwise
- "Space" for throttling up
- "<" for throttling down

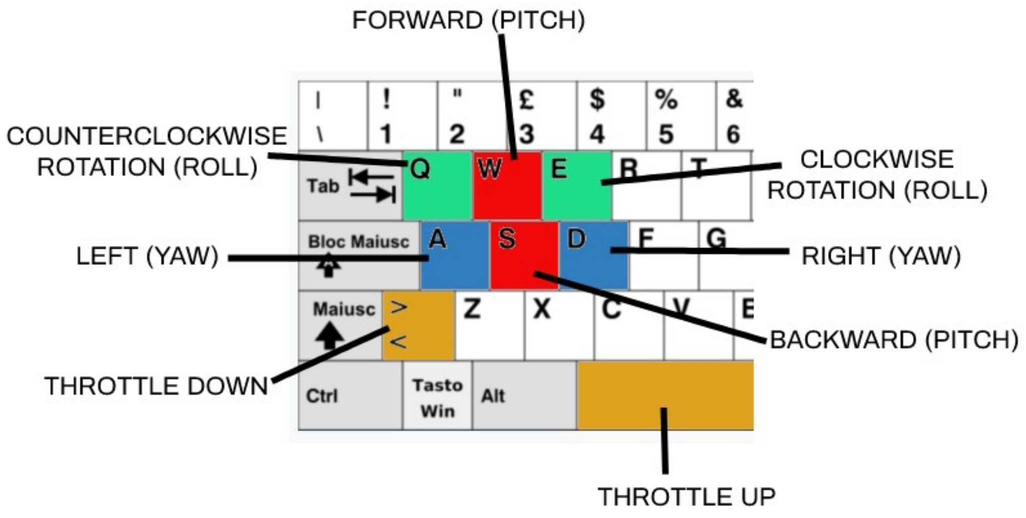

Figure 2: Controls of the movement of the drone

4 City

This futuristic city is the main environment of the game and the drone is free to move around the map. Sometimes and totally random rains can start to fall down. Textures for the grass are been used for some places into the city: these grass fields have an applied bump texture to resemble the appearance of an actual grassy field. The bumpmap textures are visible in Figure 6. Also in some parts of the city there are some neon panels that are always visible, even in the night (the emissive property of these materials is set to 1). Around the city there is a sea created using water normals, to have a more realistic effect.The flow of water is constantly updated in the rendering step. The sea also serves as floor. The scene is wrapped into an atmosphere, which serves instead as ceiling.

4.1 Model

The model of the city is taken from Sketchfab and it is composed by a total of 284.5k triangles and 182.3k vertices. We have modified the original model to fill some holes present in the city model.

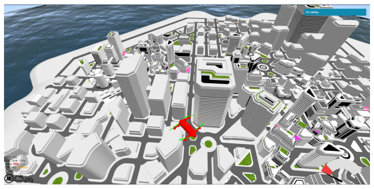
Figure 3: Day

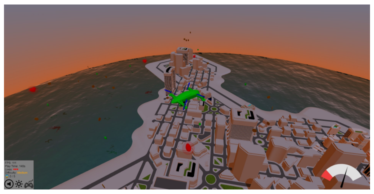
Figure 4: Afternoon

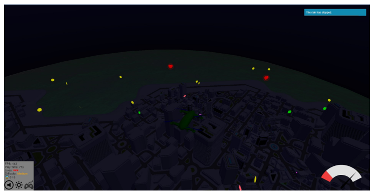
Figure 5: Night

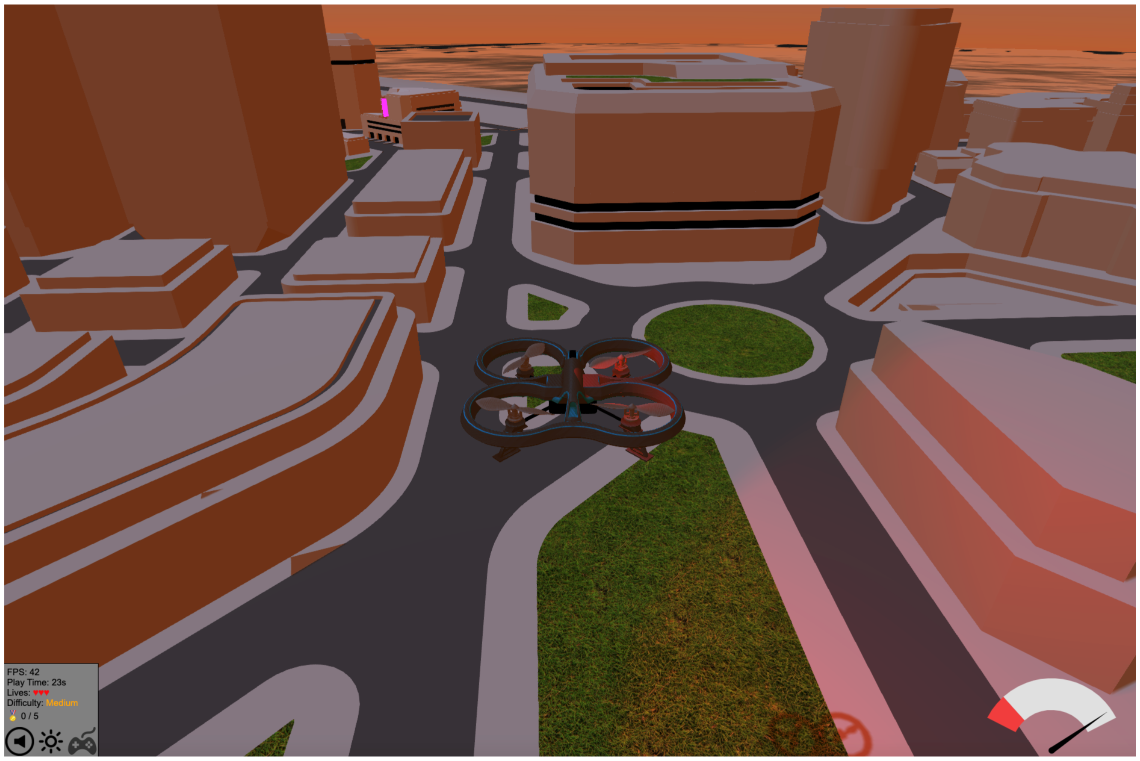

Figure 6: Grass bumpmap textures

5 Drones

We have used two different models for two different drones in the game. Both drones are hierarchical models. They are very complex and composed by a lot of objects. In our case we wanted to access the four propellers of the drone to animate them, and the body in order to rotate it. These accelerate and decelerate when the drone moves up or down. Tween.js was very useful for the realistic implementation of the movement. thanks to this library the animation are very fluid and realistic.

5.1 Models

The model of the first drone is taken from Turbosquid and it is composed by a total of 4148 polygons and 4558 vertices. The model of the second drone is taken from Sketchfab and it is composed by 27.6k triangles and 14.3k vertices. The second drone is more complex in terms of material properties.

6 Consumables

The remaining models are the ones for the coins, for the hearts and for the fuel tanks. They are spawned into the map with a maximum limit of 50 for the coins, and 10 for both the fuel tanks and hearts. Figure 7, 8 and 9 show all the three models.

6.1 Models

The three consumable models are taken by Sketchfab and are clearly more simple than the drone and the city, however they are extremely detailed in terms of textures (especially the coins, which resemble the symbol of Bitcoins). All three have the emissive properties set to 1, in order to make them visible in every time of the day.

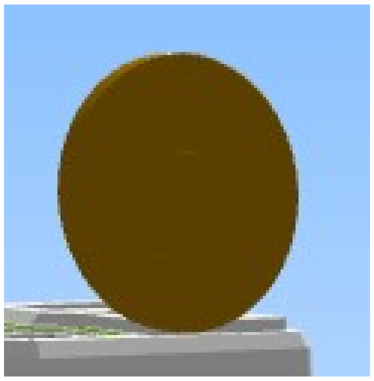
Figure 7: A coin

Figure 8: A heart

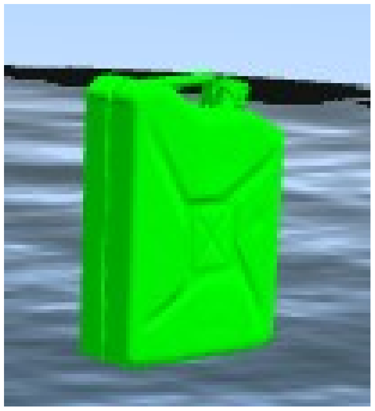
Figure 9: A fuel tank

7 Lights

The scene contains different types of lights. There is an Hemisphere Light, an Ambient Light and a Directional Light to build in a physically correct way the three parts of the day. Their intensity and color vary also depending on whether the rain is present or not, and are shifted in a smooth way via the use of Tween.js. Also, if the rain is active, a flash light is introduced to resemble the behavior of thunders. Finally, if it rains or it is night, a spotlight appears, lightning the objects in front of the drone. It is switched on / off by using Tween.js to reproduce a more realistic behavior. Figure 10 shows the spotlight.

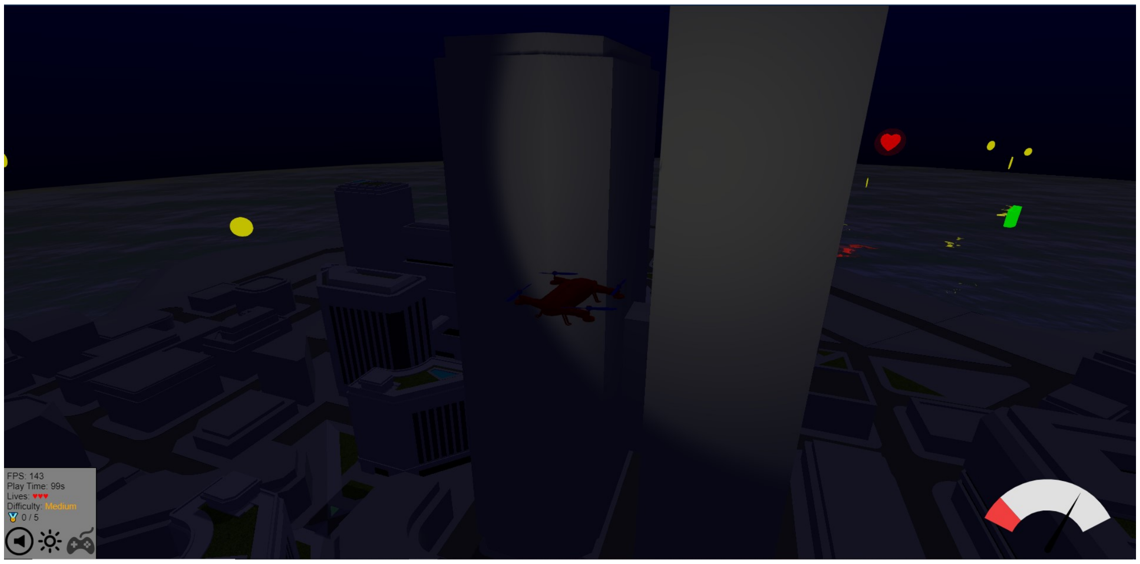

Figure 10: Spotlight of the drone

8 Sounds and effects

If the player clicks the key "U" sounds start/stop in the game. Sounds include music in background, propeller's drone movement sound, death and win sound, consumables collecting sound, rain sound and finally sound of hit again a wall or the floor without the loss of one life (collision).

9 Gameplay

After the selection of the color of the body of the drone and of the color of propellers, the difficulty level and the click of the button start, the game begins. The drone is on the floor at the very beginning. It has to ascend in order to start collecting some coins. If the player sees that the drone is going up with too much speed, the button "<" can be pressed to slow the movement of the propellers in order to start a descent. Opposite commands (such as forward and backward, left and right) are mutually exclusive and clicking them together doesn't change the direction and the rotation of the drone while clicking together non-mutually exclusive commands combine different kinds of movements and rotations. The player can see many kind of objects through the map. Coins must be collected to win the game, in particular 3 coins for the easy mode, 5 coins for the medium mode and 10 coins for the hard mode. We have modified the rotation axis of the coin to reach the current result visible in the game. Fuel tanks must be collected in order to keep the fuel bar filled in order to keep flying. If the bar is totally empty the drone will fall down and it looses a life. Moreover a life is lost also when a collision with a wall happens. The last kind of objects in the game are hearts. They can be collect to restore one life. Every collectable item falls down from the sky and they can stop themselves at random altitudes in a random point in the plain parallel to the floor of the map. The drone's propellers are moved to physically mantain coherence with the drone speed controls, not only while lifting the drone up or down, but also while using the other commands (as in the real world). All is managed to keep physically correct angular velocities, linear velocities and forces by numerical derivation and may randomly be influenced by very low FPS: to resolve this problems a stabilizer is implemented in order to recover any possible problem.

10 Conclusions

We have tried since the very beginning of the project to realize the most accurate animation for the drone. For this reason, it may initially be not so easy the perfect control of the drone. But with some practice it can be perfectly governed and controlled. The animations are obviously a little slowed down by the use of a physical engine and a large scene (in comparison with a light example we initially realized with only the drone and the tweening for its animations), but the overall result in terms of performances on tested average computers are satisfactory.

11 Credits

Complete list of models used

- Drone 1: https://www.turbosquid.com/3d-models/qimmiq-transformer-drone-hybrid-c4d-free/1128800
- Drone 2: https://sketchfab.com/3d-models/rc-quadcopter-e03710ac63a1437ab51723b5c04ad50b
- City: https://sketchfab.com/3d-models/city-grid-block-3488e40ceca846bb9023f894a749c398
- Coin: https://sketchfab.com/3d-models/bitcoin-ff17dab54b044d789d05ae7c8dfb9808
- Heart: https://sketchfab.com/3d-models/heart-in-love-f39ce19b92e246268f4c501b72ea7d0e
- Fuel tank: https://sketchfab.com/3d-models/fuel-tank-midpoly-painted-c82eec8c84c04f10a707d2b3f7e0b727