Interactive graphics

Final project report

Spring 2020

Instructor:

Prof. Shaerf Marco

Made by:

Pietro Manganelli Conforti

1754825

Index

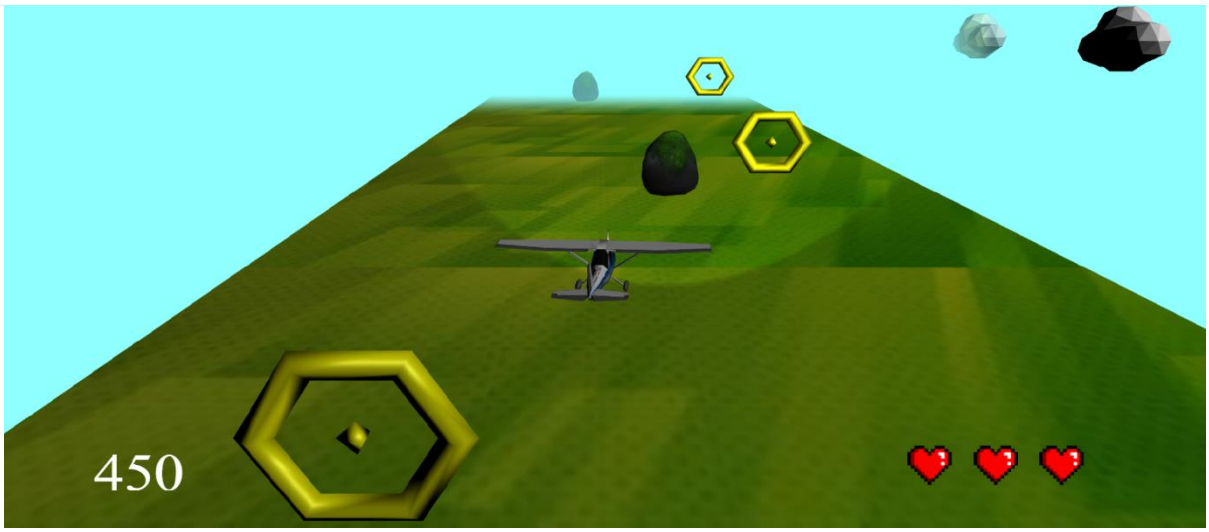
Introduction

The game is an obstacle run where the player controls a plane with the keyboard and follow a bounded path where the plane can move freely while avoiding obstacles and collecting different object to increase the final score.

Everything was implemented using three.js as base library, with GLTFloader.js to import the various models. The menu is done in HTML while the logic and the style with JavaScript and CSS.

Gameplay

The player during the game must maximize its score by collecting coins and survive by avoiding obstacles as trees and rocks. There are three possible difficulty levels, each related to a different plane, initial velocity, maximum velocity, number of lifes and score values for different events. The difficulty level can be chosen in the settings accessible by using the menu and pressing the relative button. Here as well as these levels also the soundtrack, fog presence and no collision mode can be chosen.



The game is developed through three main stages: one where the coins are generated and animated, they follow different pattern and have to be collected by the player. In the next section a group of obstacles is spawned with random sizes and positions (trees and rocks) and the player must

avoid them while proceeding. Then in the third part there is a moving obstacle (a giraffe) which is animated and moves towards the player position and a rotating hearth. This last one can increase the actual life count or if already maxed out increase the total score, of an amount which depends on the difficulty chosen. Both the giraffe and the hearth are animated, the first the player encounter is the giraffe which will try to catch the player by following its actual position while the second one is positioned behind it in the middle of two statues (which are collidable obstacles) rotates around its vertical axis. After those three stages the global light (which is a diffuse light source) behind the player reduce its intensity and change its colour, together with the background and the fog, simulating the night. So, the light source (a spotlight) controlled by the player located in the front of the chosen plane model permits to continue the fly enlightening the obstacle of the course.

After the ending of the three stages together with the light changes the velocity is increase making the game harder. This repeat until a maximum velocity is reached (which is different for each plane, in relation to the difficulty). The score as well increase after completing the three stages and each time it increases is in relation to the difficulty. After the player completes the three level the cycle repeats endlessly the last life is lost, making the ending screen appears with the points scored.

Game menu, settings, commands

The HTML page loads four different div elements (5 counting also the end screen, alternating their values of style. display from block to none) and when opening the page display a menu initialized by the *init_starting_menu()* function. Here the player can choose to: start the game with the default settings, choose the preferred setting pressing the relative button or see how to play by pressing the "command" button.

Settings

Here the user can interact with the settings to choose:

- The difficulty level, which are the three different modalities relative to a big plane (easy modality), a normal plane (medium difficulty) or a paper plane (hardest one). Each of them corresponds to different velocities and score evaluations of elements in the game, as we will see in the next section.
- The audio soundtrack, which is the "*BigCarTheft.mp3"* track in the audio folder and can be enabled/disabled in this setting section,
- The fog presence or not, that makes the game less artifact by making the elements gradually becoming more visible from the background
- The no collision mode, which is a modality where the player cannot die because the collision flag is set to false and so he can freely explore the game.

Commands section

Going back from the settings the menu is enabled again and the commands section can be seen. The plane moves by using the WASD buttons on the keyboard to go up, down, left and right.

To understand how the plane moves we have to take into consideration that it has always a base value for the velocity in relation of the difficulty level chosen. It indicates how much the controlled model will move in relation to its direction, so the angle the model has respect to the vertical

and horizontal axis, global variables angleX and angleY in the code. The angleY is not initialized to 0 but -pi/2 because the plane goes in the negative z direction. When pressing a control button on the keyboard, for example A to go left, the angleY increase (or decrease with D) and so the components of the velocity along that direction. In fact, in the code is implemented (in the function *movemesh()*) with the formula

*mesh.position.z += Math.sin(angleY) * vel;*

*mesh.position.x += Math.cos(angleY) * vel*;

so angleY correspond to a greater or lower lateral movement respect to the x and z axis (in reality its specular respect to the initial value of -pi/2 so the plane by going left pressing A decrease angleY and increase the cosine components on the x and decrease the sine in the z, the opposite going right). This is done also for the W and S button for the angleX so the next y position of the controlled model.

To make the player follow the underlying plane the movement is bounded to -50, 50 for the x axis and 5, 40 for the y axis. When the plane translates at each animation frame also the model rotates coherently , this always in respect to angleY and angleX. Not only the next position is bounded to the discussed limits but also the maximum and minimum angleX and angleY value, so the plane can't rotate on itself (in fact the position is only related to those angle values and if not bounded it could go backward). The spotlight attached to the model follow the same logic and points in front of the plane for the X and Z target value and to make it more effective (in the night phase is recommended to use it) its Y target value is from the sin(angleX) divided by 5 (so the light target is more constrained inside the scene). The light, as explained in the commands section can be turned on and off by pressing L.

While moving together with the light, the model and the rotation of the model, also the camera position and the health-sprite one changes, keeping the first behind the model and the seconds on the bottom right of

the screen. Those last ones are updated each time a collision occurs, in fact they change colour when a life is lost or gained.

Controller

To have a better interaction with the game has been implemented a more responsive controller than the one which results from a listener on the press or release of the WASD key value (implemented in the first versions of the game). This because there is a noticeable delay if the action to be executed has to be done multiple times while the button remains pressed(as in our case, for example if an obstacle has to be avoided). To avoid this problem, at first a map with each key value of the pressable buttons is initialized to the relative function (for example *87:goup(), 83:godown(), 65:goright(), 68:goleft()).* When a button is pressed which has a key value inside the map a setInterval function is called which repeat the related function in the map each 30milliseconds. So, the function repeats until a key is released calling a function which clear the interval associated to the keyup event and so stopping the execution of the movement function. Aside from this controller, also two event listeners to turn on/off the light and to pause the game are implemented.

Collision detection system

As we said the difficulty levels are three and each corresponds to a different plane model. Each plane has an own maximum number of lifes displayed on the bottom right corner of the screen. When the position of the model which the player controls enter inside the box of an obstacle a collision occurs and then one life is lost (executing the function *remove1life()* in the code). Doing so the life counter decrease and the corresponding health sprite becomes black, after three blinks of both the plane model and the health sprite. During this blinking, the player cannot collide with other obstacle, resulting into two seconds of invulnerability where he can move the plane into a safer position. The collision system works by checking at each frame if the player position is contained in one

of the three.js box3 structures, which are allocated for each collidable object. The life-decreasing collision are for the threes, where crowns and logs have different boxes, and excludes the small logs on the bottom of the crown, the rocks, the statue behind the giraffe, the giraffe. All these boxes are created with the function: *THREE.Box3().setFromObject(object)*, and sometimes adjusted by expanding them by a scalar. All those elements are pushed inside a list of collidable object which have to result into a life loss when touched. This is checked inside the function *checkCollision()* once per frame. Together with this kind of collisions also ones with score-rings or with the hearth cube can occur. For the rings another list takes into account the rings created and the relative boxes, when one is touched by the plane the score increase of an amount relative to the difficulty and the ring is removed from the scene. Rings are not fixed but animated with different patterns and are spawned in random position inside a bounded area which the plane can cross. After the rings movement also the corresponding bounding boxes have to be updated with the same translation (this happen also for the giraffe's box). Together with the score-increasing collision(rings) and the life-decreasing ones(obstacles) also the hearth can be picked to recover a life. This is done through the *gain1life()* function in the code, which increase the life counter by one and recolour the respective life sprite. To avoid the waste of resources, all these informations about objects and boxes kept in their respective list are cleared after completing the stage.

Difficulty levels and the score

The score is an HTML element displayed on the bottom left of the screen. It is created in the HTML file and its attributes are defined in the main.css file with z-axis is set to 1. In this way when changed the attribute style.display = none to block it remains on the top of the screen to show the score. It cannot decrease and maximizing it is the purpose of the game. When the game ends the final score is displayed in big at the center of the

screen, near a restart button which refresh the page. Each difficulty level has its own scores values for each event, as well as life counter and velocities. Score rewards are explained in the following table:

Difficulty level	Model type	Lifes & Maxlifes	Ring score	*Light change score*	*Extra life score*
Easy	*Big*	*5*	*100*	*500*	*1000*
Normal	*Normal*	*3*	*150*	*800*	*1500*
Hard	*Paper*	*1*	*200*	*1200*	*1500*

-The model type is the loaded model of a plane which can be chosen to play with.

-The lifes are the sprites which are loaded on the bottom right of the screen, they become black after a collision and at zero the game ends.

- The ring score is how much the score increase after a collision with a ring, which result in the disappearing of the ring.

- the light score change is how much the player is rewarded for completing the three stage of the game, together with the increase of velocity and the change of the light (day/night and viceversa).

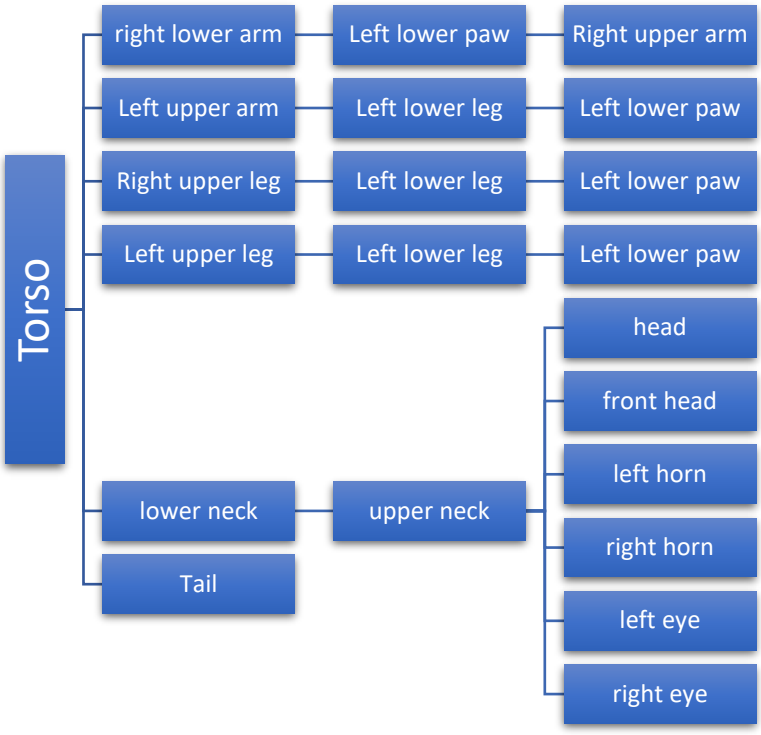
- the extra life score are the score-points obtained after colliding with the hearth box while having all the lifes. This because each plane has as life limit its starting life value. For the paper plane, which has only one life and so doesn't need to recover them, taking the hearth-cube result always into score points.

Moreover, the velocity which start at 1.0, 2.0 or 2.5 and max out to 2.0, 2.5 or 3.5 for the easy, medium and hard mode, is increased after ending the three section of the game of 0.25.

Models & animations

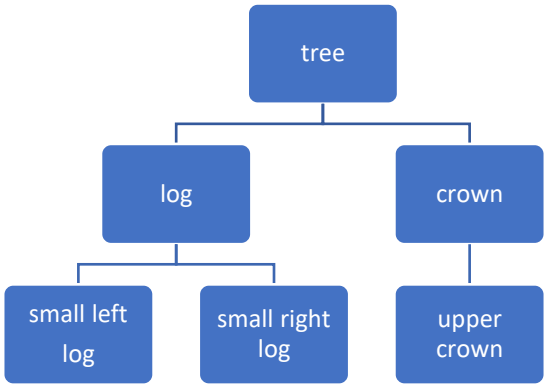
The giraffe

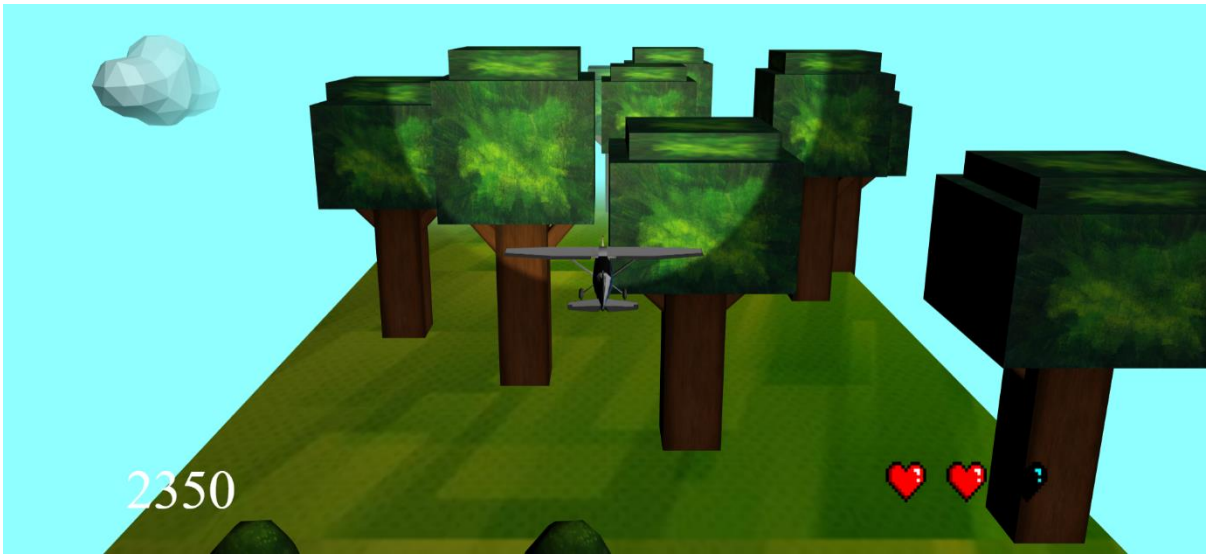
During the initialization in the init() function is created a hierarchical model of a giraffe, which will be next cloned each time the player survives and exit the forest and before the hearth cube. The model is of an animal tall enough so that the player cannot fly over it but must turn on its side. The animal is created in the function *create_giraffe()* and has a bounding box to check the collisions with it. Its animation consists by moving the legs increasing the angle of the upper part by a certain angle. When it reaches its maximum (+/- *Math.PI/12*) inverts the direction of the rotation. To track the player position, the giraffe also rotate respect to the y-axis, when the player is not in front of the giraffe (in a certain range). Then, by using a setInterval function the giraffe move translating by a *delta_x* and *delta_z* value until it reaches the player.



When its z position is almost at the same level of the player it stops, clearing the interval function used to move. Texture used are two, one for the body and one for the front of the head, and for the paws, eyes, horns, and tail have been used different colours. Because its hierarchical structure, when animated moves coherently as a single element.

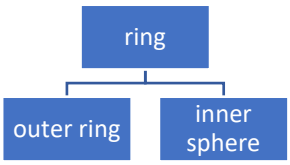
The trees As the other models trees are at first created in the *init()* function and next cloned when needed. Are composed as explained in the figure and associated to a bounding box to check eventual collision. Are principally in the forest section of the game, where a variable (now set to one hundred) indicates how many have to be created.In particular, the position is random but to preserve a good playthrough there is a fixed delay between each creation. The size is random too so the





forest section can be played both in the top or in the bottom of the scene, making it more flexible to different kind of playstyle. The textures are two, one for the crown and the upper crown and one for the logs. The material chosen is *THREE.MeshPhongMaterial*, because gave the best visual result during the night phase.

The rings are created in the first phase of the game, are composed by an outer ring, done with *THREE.TorusGeometry()*, and an inner sphere, done with *THREE.SphereGeometry()*. Together with a yellow material they compose a single object which follow different pattern, basing on the respective creation index. They start fixed from a random position inside the area which the player can cross, with horizontal and vertical velocity equal to zero. Next after 3 rings the horizontal velocity change so they start to animate along the x-axis, stopping before the player arrive to make them easier to pick. Next also the vertical velocity change, so the pattern of the animation become more unpredictable moving along the x and y axis, always remaining bounded into the same limits the player has. When touching a bound, the relative velocity of the ring is inverted giving the impression of a bounce. In the ring section, also rocks are spawned in front of the player with a little displacement, to encourage the movement while trying to reach the rings.

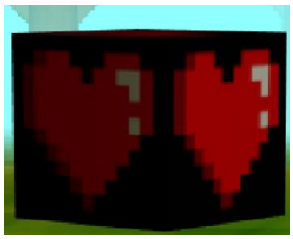


Other models rocks, clouds, the three plane models and the statues behind the giraffe are all models imported in the function *loadModels()* by

the *THREE.GLTFLoader(pathToTheGLTFfile)* element. They as the previous

models are first created and set to the right proportion and next cloned each time when needed. For the planes model when chosen together with the specific velocities is set also the initial position of the plane, to orientate the model respect to the scene.

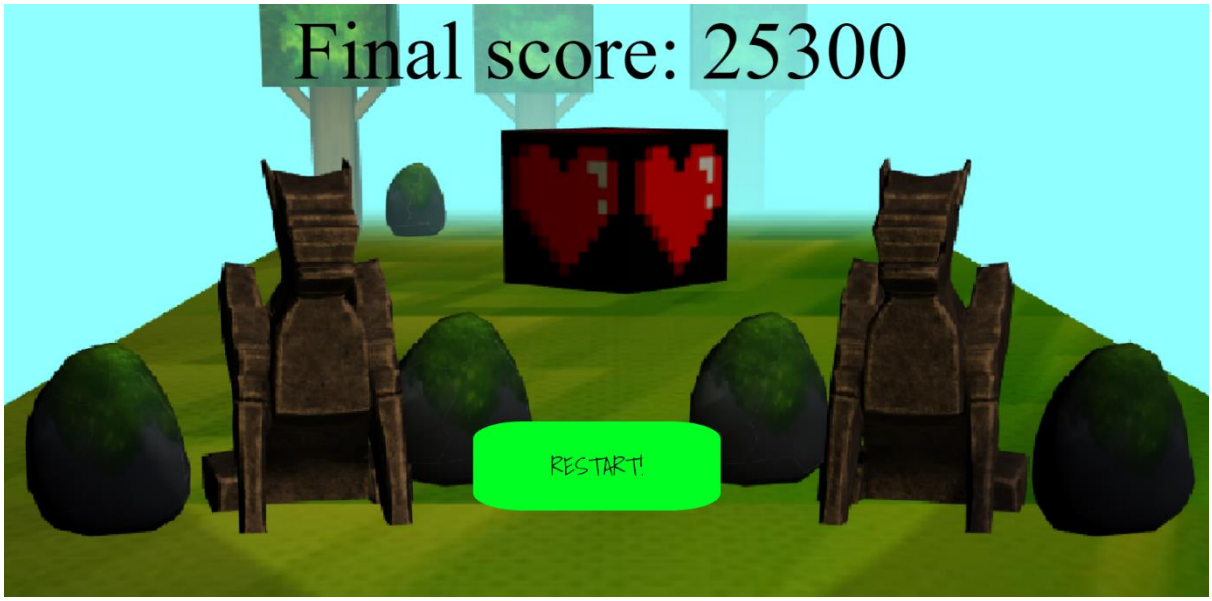
The Life sprites are loaded by using the *THREE.TextureLoader(texturePath),* and created by doing first *new THREE.SpriteMaterial(map)* for the material, passing as a map the texture loaded, and next *new THREE.Sprite(spriteMaterial)*. Finally, are added to an array which will be used to change the colour coherently to the lifes



counter. In fact, they switch in case of a collision the red value from 1 to 0, by doing *sprite_ico_array[lifes_counter].material.color.r=0.0* . The same texture is used for the health-cube which gives one life when picked.

CONCLUSIONS

This project aimed to understand some of the possibility that the WebGL technologies gives through also the use of the three.js library, in order to create an interactive game, hoping also to be enjoyable to play. Actually, the project is hosted in the GitHub page at the link: *https://sapienzainteractivegraphicscourse.github.io/final-project-airplanes/*.



REFERENCES

- https://threejs.org/
- https://discoverthreejs.com/book/contents/
- https://discourse.threejs.org/

3DModels from sketchfab.com :

- https://sketchfab.com/3d-models/cougar-statue-bc6268a7916d48729f9afac0f1eb0cd9
- https://sketchfab.com/3d-models/polygon-runways-sea-duck-plane-9696ba99137c42b69e0b58bbbdde7be6
- https://sketchfab.com/3d-models/low-poly-plane-76230052903540e9aeb46b7db35329e4
- https://sketchfab.com/3d-models/paper-plane-20009f11e69b44cfa47f91ee61d3b849

- https://sketchfab.com/3d-models/mossy-rock-83a63b3fb482442d9fa55ac0585a52fb

The health sprite:

- https://www.deviantart.com/emeraldplaysmcraft/art/Pixel-Heart-487067067

Soundtrack:

- https://freemusicarchive.org/music/Jason_Shaw/Audionautix_Tech_Urban_Dance/TU-BigCarTheft