Down The Veins!

Interactive Graphics Final Project

27 June 2021

Federica Cocci
1802435

Alberto Coluzzi
1814786

1 Introduction

This project has been developed for the Interactive Graphics course during the Academic Year 2020/2021.
It is a game implemented in JavaScript and HTML using Three.js library which allows us to create and display animated 3D computer graphics in a web browser using WebGl.
We have been inspired by the present situation about the Covid-19 pandemic: "Down the Veins!" is a game that wants to reproduce the inside of the human veins when the vaccine is inoculated. We imagine a micro-doctor who wanders around veins to pick up masks, hand sanitizers and vaccine doses to fight the enemy: the SARS-CoV-2 virus.

2 Rules of the game and controls

The goal of the game is to defeat all the viruses using the collected objects.
"Down the Veins!" starts in a maze where we are the micro-doctor and we have to collect the greatest possible number of supply in 1 minutes and 30 seconds.
We have to go around the maze using the common WASD controls in conjunction with Q and E for controlling the rotation, to give a more retro look.
While we explore the maze we need to stock up on some useful resources before the timer ends, like:

- Dose of vaccine: it will be used to fight the virus.
- Mask: using this in the arena will allow you to be immune from the virus for 5 seconds.
- Hand sanitizer: using this in the arena will allow you to increase your velocity for 5 seconds.

At the end of this timer, all the walls of the maze collapse and in this big arena we have to fight against some viruses.
In this stage of the game we can utilize the objects, that we grabbed before to give us some powers.

The keys that you can press in the game are:

- W: moving forward
- A: moving to left
- S: moving to right
- D: moving backward
- Q: rotate to left
- E: rotate to right
- Z: use the dose of vaccine
- C: use the hand sanitizers
- X: use the mask
- R: change the camera (third person - first person)
- ESC: pause the game

3 Environment

To develop our game we have used Three.js using the source code at https://threejs.org/build/three.module.js whose revision is the number 129.
The animations have been implemented using the Three.js animation object.

4 Tools and Models

Our models have been imported from Blender where, for all of them, we have created their textures and their structures.
In Blender we have built 6 models and 3 of them are, in particular, hierarchical models: the player, the virus and the syringe. Here there are their hierarchical tree representations:

Figure 1: Tree of the player

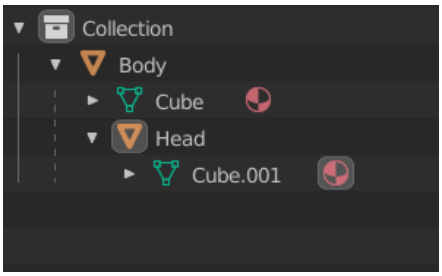

Figure 2: Tree of the syringe

Figure 3: Tree of the virus

We use GLTFLoader of Three.js to load the models exported from Blender.
It is important to point out that the hierarchical model of the player has been expanded with the syringe that will be the child of LowerLeftArm. This expansion has been made directly in the JavaScript code using Three.js function "add" called over the children of the player.

Below we can see all the models:

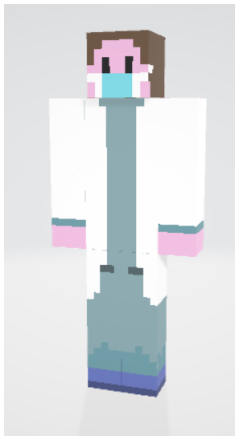

Figure 4: Player

Figure 5: Virus

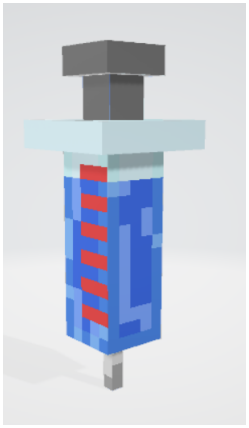

Figure 6: Full Syringe

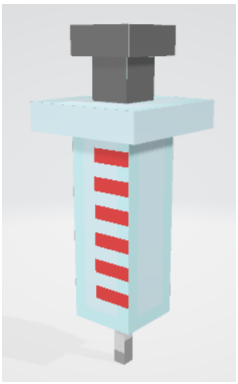

Figure 7: Empty Syringe

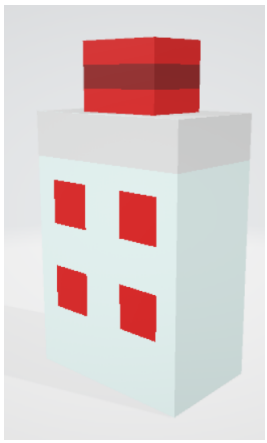

Figure 8: Gel

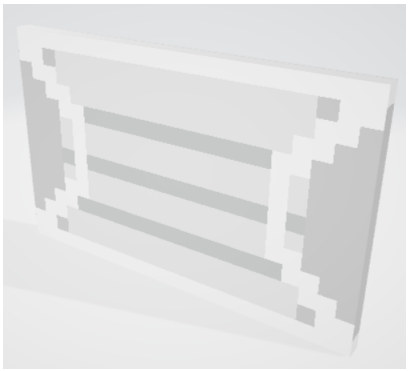

Figure 9: Mask

Figure 10: Vaccine

The full syringe and the empty syringe share the model but the texture which is different to give the effect of vaccine absence.

5 Technical aspects

5.1 FSM

We have used a Finite State Machine with the following states:

- State 0: menu
- State 1: maze
- State 2: arena
- State 3: pause

- State 4: tutorial page
- State 5: final result (there can be two finale pages: one if we win, one if the viruses kill us; if we finish the doses of vaccine then no message will be displayed till we die).

5.2 Light and Textures

In the game we apply to the walls and to the ground a color texture and a bump texture; for the objects and the player we use a color texture that we made in blender. For the lights we have spread 36 point-lights on a 1200x1200 grid outrunning one from the follower of 200 on x-coordinates and y-coordinates. These 36 lights have the burgundy color
There is also a global light which is white.

5.3 Animations

These three following animations have been implemented using only Three.js and its keyframe modality with animation clips (then they are distributed over time).

- Stab virus animation
- Walking player legs animation
- Walking player arms animation

These other animations have been built using directly changing the position:

- Walking player animation
- Chasing virus

5.4 Environment collisions

We implemented the player collision detection, using the ray system.
This system is able to give us the normal of the faces in front of the player.
We use this normals to compute the direction if the player is colliding with the wall.
We used this formula that makes the player able to slide against the wall and not passing through it.

$$FinalDirection = InitialDirection - FaceNormal * (InitialDirection \cdot FaceNormal);$$

5.5 Objects generations

To spreading the objects we have created an array of forbidden positions. Initially this array is composed by the areas of the maze covered from its walls (because of course is useless position objects there as the player can reach them); this array is then expanded on the way with the positions of the already spread objects. Also the initial position of the player is excluded and then is integrate in this array.

5.6 Integration of the sound

We have integrated the sound of the heart bit and the sound starts after having clicked the start button of the menu.

6 Implemented interactions

6.1 Interactions player - objects

6.1.1 In the maze

In this context, because the only thing is collecting the objects, we have written a function which depends also from the distance used to compute if the player is enough close to the object to pick up it. In this function, we have used compared squared distances for a computational reason; we use the position of the player and the position of the object and when the player is enough close to the object, this one is set to invisible and the counter of caught objects is incremented of one.

6.1.2 In the arena

In this part of the game the player has to face the virus, to give the virus a more responsive feeling the virus has three function to move and attack the player.

- The first function detectionVirus update all the virus object and gives them the chasing tag if they are less then 80 unit of distance from the player and once spotted they can run after the player until the player puts 140 unit of distance from them.
- The second function update all the position of the viruses, if they have the chasing tag active, the virus rotates in the direction of the player and chase him with a lower speed than the player, so it's possible to escape.
- The third and last function is called contact with virus and removes health to the player if is colliding with a virus.

6.2 Interactions user - game

6.2.1 Using keyboard

We used two type of interaction with keyboard:

- The first type of interaction awaits that the player remove his finger from the button for stopping the action, this is used for example for the movement.
- The other type of interaction is where the player press the key and the action is performed once and immediately.

6.2.2 Using mouse

We used the Mouse event only to make possible the interaction with the main menu, it's possible through the raysystem of Three.js that gives the programmer the ability to project a ray and know what object interacts with it.

6.3 Interactions user - menu

In the menu there are two bottoms: START and TUTORIAL. The first one allows you to start and play the game, the second one opens a window with the recap of the buttons to press (from this window you exit pressing ESC).
The interaction in the menu is made through the mouse: if the mouse is detected while is interacting with the START button then the state of the FSM is 1 so the game is loaded; if the mouse intersects the TUTORIAL button than we display a window with the instruction and when the ESC key is pressed, we make disappear the message and the state of the FSM is changed to 0 so we can return to the initial page of the menu.

7 Conclusion

It has been a beautiful adventure lasted only one week: a full immersion in JavaScript, Three.js and HTML.
We have gained lots of new notions about Interactive Graphics and developing this game has been a challenge that we have won with pleasure and pride.