Final Project Report

Antonino Iaria 2082145

An Interactive Graphics Assignment

Sapienza Università di Roma

September 24, 2023

1 Introduction

On a high level, the solo-made project, consists in a mini-game. The project is an endless game in which the player should make his/her avatar cross as many streets as possible. The game ends when the avatar is run over by a car or a truck. The controls are pretty simple, they allow the user to move the avatar in 4 different directions.

2 Used Environment Description

I decided to use the well-known WebGL library named *Three.js* for its ease of use and for its huge available documentation. This library indeed was used pretty much to perform everything, from creating new models, to manage lights and textures.
In addition to this powerful tool, I also used HTML and CSS languages to make the game run on web browser. In order to load correctly the textures and the models, I used Microsoft's extension *Live Preview* inside *Visual Studio Code.*

3 External Tools

Level generation algorithm was not developed by me, but instead I re-wrote it from a sample model found on the internet (here is the link to the template).
The base model I started from included this already-written tasks:
- Trees generation.
- Cars and trucks models and their generation.
- Road/asphalt models and their generation.
- Grass terrain mesh and its generation.
- Avatar basic movements.
- Vehicles basic movement.

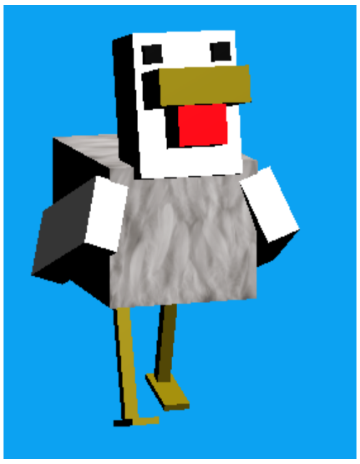
4 Technical Aspects

This section will cover more deeply the scripts I wrote and the models I designed to develop the final project.

4.1 3D Models

At the very beginning, the game only offered a ”boxed” avatar of a chicken, so I decided to re-design it from scraps as well as other 3 extra characters.

4.1.1 Chicken Model



First playable game avatar.

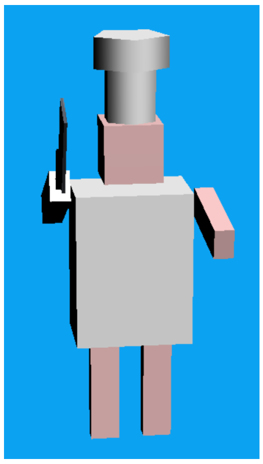
As it is probably noticeable, the chicken I designed is pretty similar to the one from the famous *Minecraft* videogame. This model is made of different geometries, all being boxed ones of different sizes. To make animations easier to develop, I also nested some models into single groups. For example:

- The head group includes the head, the eyes, the beak and the wattle geometries.
- Each leg includes the foot and the leg geometries.

The chicken also has a feather texture attached to its body only.
Avatar’s animation will:

- Rotate the whole head group.
- Rotate and move its wings.
- Rotate its legs.

4.1.2 Chef Model



Second playable game avatar

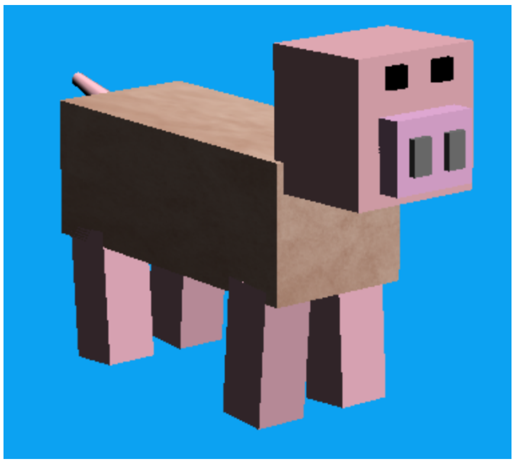
I added this model to the game because I though it would be fun and enjoyable for the players.
As before, this model is made of different geometries, all being boxed ones except for his hat. The character has two grouped geometries:

- The chef's head includes his head, the hat base and the hat top part.
- The chef's right arm includes his right arm, the knife's grip and its blade.

Furthermore, The hat is made of two cylinder geometries. The lower part is a hexagonal base cylinder while the upper one is a pentagonal base cylinder.
I preferred giving the hat this look because I though it reminded better the iconic *toque*. The blade of the knife has a metal-like texture.Avatar's animation will:

- Rotate the whole head group.
- Rotate its left arm.
- Rotate its right arm group.
- Rotate and move its legs.

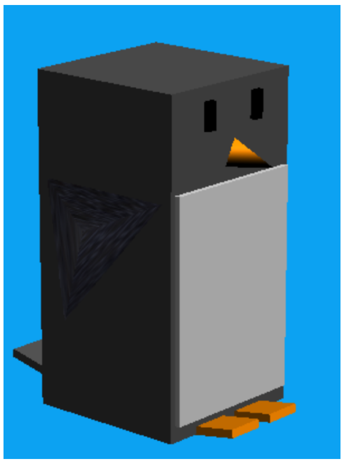
4.1.3 Pig Model


Third playable game avatar.

Following the shape of the chicken model, I also developed this third character. As before all of its limbs are box geometries except for its tail, which is a truncated cone. To simplify animation development, I nested its limbs as it follows:
- Head has a group that includes the head, the eyes, the nose and its holes geometries.
- Legs have been cross-grouped, meaning that left-front one is grouped with right-back and vice-versa.

The pig has also a texture attached to its body only. The avatar's animation will:
- Rotate and move its legs.
- Rotate its head group.
- Rotate and move its tail.

4.1.4 Penguin Model


Fourth playable game avatar.

This is the last model I designed. It's made of few limbs all of them being box geometries except for its beak, which is a cone geometry. The eyes and the beak are nested into a single group named groupedHead. The penguin has its own texture attached to its wings. The avatar's animation will:

- rotate its feet.
- Rotate its tail.
- Rotate and move its wings.

5 Game's Logic

This section will cover what and how game's mechanics work. To give a better understanding, I divided the analysis on the 3 HTML pages. The first two are indeed in the same subsection since the menu does not contain anything relevant concerning about 3D graphics and animations.

5.1 Menu and Character Selection Pages

The menu is very simple and indeed has no relevant 3D graphics elements. From here, the user can play the game or select its favourite character. The default character is the chicken.
Once landed to the character selection page, the user can pick one of the available avatar. In this page, the selected avatar will spin and perform its animation.
Three.JS models have been written in the *mcsript.js* file, and the *characters.js* was written to manage all the logic behind this page. The script allows the avatar to be generated in the pages and allows also to perform their animation. Each character has its own *AnimationUp* function.
By default, the first loaded character is the chicken. The script creates four objects and groups them into a single model. I chose to line them up at a distance of 50 from the x axis, so the chicken is in position 0 the chef in position 50, the pig in position 100 and the penguin in position 150. When the user change viewed character using the buttons, the previous avatar stops spinning and the camera is shifted to the next model.
The *functions.js* file ensure that the renderer and camera are always up-to-date with the current size of the browser window. Without it, the avatars' preview would have a poor quality definition.

5.1.1 Symmetric Animation

At the beginning of the project, the animation was divided into two functions: *AnimationUp* and *AnimationDown*. The idea was that to call the first function as long as needed, and then recall the second function to "reverse" it.
This approach worked great on the character selection page, because the "jump" movement called the two animations function depending on the movement of the avatar.
On the other hand, things got more complicated when I started dealing with the play page

instead. For this reason I developed a ”Tic-Toc” approach, to obtain this ”symmetric animation”.
Another advantage of this approach is that I was able to change animation length by simply modifying the two variables *ticMax* and *ticMin*. This aspect was indeed quite important, because the animation duration differs from the two pages (character selection and play).

Once the user press the button *Play*, the script saves what characterthe player has chosen, in order to load it in the play page.

5.2 Play Page

The majority of the logic behind this page is done by the *script.js* file.
The avatar will perform a simple animation, which is the same in character selection page, but quicker. During the development I though it was not a good idea to make the jump duration slower, since the interaction would have been too slow and hard to enjoy. So, I preferred to keep a quick and short jump/animation movement and speed up the animation the avatar does. The only disadvantage to this is that the movement is limited to the short duration of time, so in other words the animation may seem to move the avatar not so much.
I also tried making vehicles' wheels spin, however there were several performance issues related to this animation and probably with the generation, so in the end I preferred to keep them static.
I also added some textures to the trees and their trunks, however once again the performance were too bad and made the game unplayable.
Jump animation is triggered by pressing one of the four movements buttons and it lasts for 250 milliseconds.
The script generates 9 ”stripes” in front of the avatar, and it randomly selects to add a road, or a grass terrain to each of them. If it's a road, there might be a car or a truck one. Vehicles will follow the same direction for each lane. Vehicles' speed will be the same for each lane but there are different speed which the script randomly selects.
Whenever the avatar jumps to the next lane, the player's score will increase of 1 point.
The game keeps generating new lanes until the avatar is hit by a vehicle. In this case, the game ends and the player can retry by pressing the red button. Once the button appear, indeed, the player is not able to make movements anymore.
When the avatar is run over a vehicle, it will be upside down and will spin whenever is hit.
When the user presses the retry button, the game will reset the avatar position, the map and player's score.
Since some models have different height, I introduced an adjustment value that trim avatar's jump height to be right above the terrain.

6 User Interaction

All the basic interactions are done by pressing the buttons. Some of them call html pages, while other ones indeed recall some events from JavaScript code. Regarding avatar's movement, it's possible to also use keyboard arrows instead of the buttons.
Whenever the user wants, he/she can return to the main menu by pressing the button on the top.

7 Textures

At the end I decided to only include four different texture, all being assigned to avatar models. As I mentioned before, I also tried associating some of them to the tree's leaves and trunks but the game performance was unpleasant. The problem was that once the avatar did a jump, the new lane generation (if included some trees) would halt the game for about a second.
Avatar's textures have all been compressed and colour-bit-depth reduced to be as tiny as possible.