Name of the Game: “Catch the dogs”

Interactive Graphics Project a.y.2021/2022

Philip Wright *- Matricola 795153*

“Master’s degree in Artificial Intelligence and Robotics” at La Sapienza, University of Rome

Summary

Introduction

The scope of this project is to create a 3D game using WebGL; we used “three.js” library, one of the most popular Javascript libraries for displaying 3D content on the web. The game is in first person, so the player can move on the field using his keyboard and he can also look around using the mouse. The scope of this game is to catch all dogs.

The game finishes in winning mode, if the player catch all dogs, but he can lose in case he goes too far from the starting point and he “falls” down.

The first page shows the rules and the instructions, as it is shown in the next Figure:

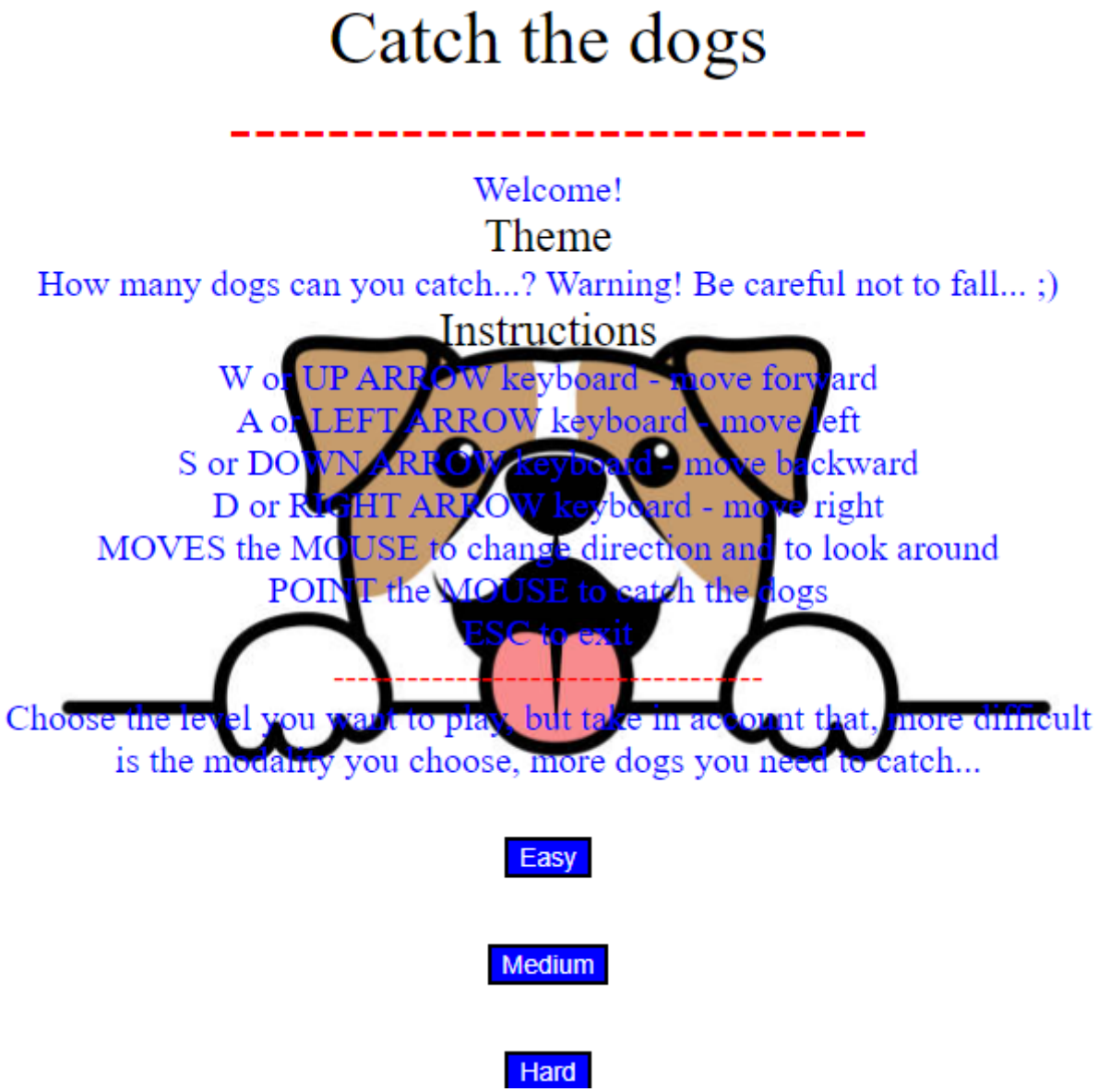

Figure 1 – First web page of the game ("index.html")

There are three different levels: ***easy***, ***medium*** and ***hard*** as requested for the project. The difficulty is related to the numbers of dogs to catch.

Libraries used

Three.js

Three.js is a JavaScript library and an API that is used to create and display animated 3D computer graphics in a web browser without relying on proprietary browser plugin: this is possible due to the advent of WebGL. Three.js runs in all browsers supported by WebGL 1.0. The version that has been used is the release number 119, "r119" and some of the main features of this API are:

- Scenes: add and remove objects at run-time;
- Cameras: perspective, orthographic, cube, array and stereo;
- Geometry: planes, cubes, spheres, torus;
- Objects: meshes, particles and more;
- Materials: basic, Lambert, Phong, smooth shading, texture and more;

- Lights: ambient, directional, point, spot and shadows;
- Data loaders: binary, image, JSON and scene.

The three.js GitHub repository is full of ready to use useful examples like the loaders, so we used different loaders that will be mentioned in this report.

PointerLockControls.js

PointerLockControls implements the inbuilt browsers Pointer Lock API. It provides input methods based on the movement of the mouse over time, rather than the absolute position of the mouse cursor in the viewport to have more sophisticated and smooth effects.

It gives access to raw mouse movement, locks the target of mouse events to a single element, eliminates limits on how far mouse movement can go in a single direction and removes the cursor from view. It is the best choice for first person 3D games; therefore, this is the reason why we chose it for this project.

MTLLoader.js, MtlObjBridge.js and OBJLoader2.js

All these libraries are needed to load external objects: in our case to make the trees. In particular, the Material Template Library format (MTL) or .MTL File Format is a companion file format to .OBJ that describes surface shading (material) properties of objects within one or more .OBJ files. The material can be added to the object using "MtlObjBridge.js" library. Finally, OBJLoader2 is a loader for the OBJ file format in which it is possible also to set the position, the rotation, scale, etc. into the scene.

The scene

Camera and Controls

The camera used is a perspective camera, and it is given to the PointerLockControls. There is a listener for the buttons that allow the player to interact with the scene; basically, for this game, simply to move the player. We also used a class named raycaster is used for mouse picking (working out what objects in the 3d space the mouse is over) amongst other things; in fact, we used this to "catch" the dogs.

Light

There is just one HemisphereLight that simulates the daily sunny light, in a fixed position.

Floor

The floor is a PlaneBufferGeometry. Its texture has been downloaded and it has been repeated in both coordinates to fill it.

Dogs

Dogs are made respecting one of the requirements to realise this project. In fact, all the parts are connected as hierarchical connection among them. The parts connected are the body, that is the "father" in the structure we have followed, and in which all are other parts are connected as shown in Figure 2.

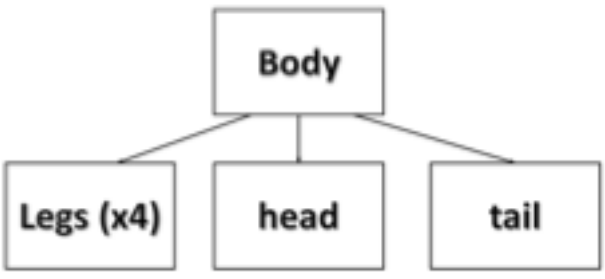

Figure 2 - Hierarchical structure of the dog. Arrow starts from the child and points to its parent

Instead of showing four boxes to represent the four legs of a dog, the figure shown “X4”. In the code these four components are represented by 2 legs and 2 arms; of course, a dog has no arms, but we followed the example given during the lessons and especially the one that we have seen to make the second homework in this academic year. The idea it is only to distinguish the two front legs (“arms”) and the backs one (“legs”).

Since the code has different rows to implement this part, we decided to create a dedicated javascript file, named “dogs.js”. This can be seen as an external function called from all the different javascript files representing the three different levels.

All the parts of the dogs (body, legs, etc.) are cubes and are called in the main javascript file (i.e., “easy_level.js”) that merge all in a single element, the dog. What must be taken in account are the number of cubes that has to be set in the “dog.js” file, that is related to the numbers of link connected to the body; in our case this number is equal to 7. If we want to add a new part, such as 2 ears, for example, we need to follow the commented code inside “dogs.js”, but we have also to increase this number to 9 (7 plus 2 ears, so 9).

The skin of the dogs is made by applying a texture on the six faces of the cubes; in addition, an image of a face of a dog in one single face of the cube of the head element has been added to give a cuter look; the obtained result is the following:

Figure 3 – Dog

In the javascript levels there is, an array named “dog_scene” that has been added in the general definition of the variables; so that, instead of looking every time in the scene looking for all the dogs’ “elements” to be animated, it is possible to simply call the related object of this array.

For the animation we used some additional libraries; in fact, the animation is controlled by a simple function that rotate the legs, front and back ones, and translate the position of the dogs on the space at a regular sequential time. We also changed the position of the legs of the dog respect to his body to avoid any different situation from the reality.

Trees

To make the game more realistic, we decided to add some objects in the scene; to do that, we used a software named Blender, in which it is possible to create a 3D object from scratch and to export the object in OBJ format.

Following a screenshot during the realisation of the tree using Blender software:

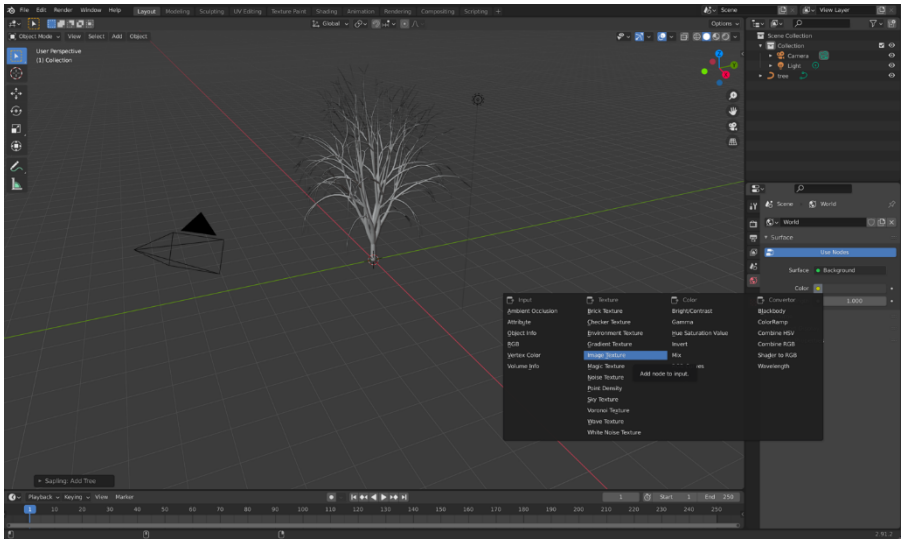

Figure 4 – Making the tree using Blender software

The tree has been downloaded in obj format. In this format the files that describe the visual aspects of the polygons are stored in external .mtl files. More than one external MTL material file may be referenced from within the OBJ file. The .mtl file may contain one or more named material definitions. However, in this case we have just one .mtl file related to some texture image to make the tree looks great.

After that the tree was realised in Blender, a function has been made in the code to import both the object and the material; so that, it is possible to call that function passing the position in the space to easily make few trees in different positions. Note that also in this case it is possible to make a random loop, as it was done to set the position of dogs, but few trees are enough in pre-defined positions.

The following figure shows how a tree looks like in the game.

Figure 5 - Tree

Global Aspects

Technical details

The scene requires some seconds to be ready especially when “hard level” is selected in which a huge amount of dogs will be positioned randomly in the scene.

To avoid the player to not moving randomly in the space, an “end of the game” has been added if we “falling down” or better if the position is major or the ground dimension. This is due by the fix dimensions that has to be set for the ground, so two possible solutions are available: adding a “wall” to not allow to move on, but

we decided that can be more funny to end the game if the player pass this part: so we take in account the position of the player and if this position is reached the heigh position in the space will be automatically increased with negative values and the game will be over.

Pause

The user can pause the game by pressing ESC. Initially we thought to also pause the time, but since we do not take in account the time in this game, this part of the code was commented. Since the focus will be lost in this modality it is possible to use the cursor of the mouse to catch the dog/s: so it was leaved as a possible trick of the game.

Levels

The game is presented in three levels: ***easy***, ***medium*** and ***hard*** as requested for the project. The difficulty is related to the numbers of dogs to catch.

Conclusions

The aim of the project was to make a browser game that meets some requirements. The hierarchical models representing the dogs as well as the different levels and the animation that was done for the dogs both to match the requirement of the project, but also to make the game more dynamic.

In addition, more features have been added to make the game nice to play and funny as well.

References

Starting point - https://threejs.org/

Starting point with some tutorials related - https://threejsfundamentals.org/

A useful guide - https://github.com/tweenjs/tween.js/blob/master/docs/user_guide.md

Additional information - https://get.webgl.org/

To make an object to be imported (in our case the three) - https://www.blender.org/

Add-on three for Blender - https://docs.blender.org/manual/en/latest/addons/add_curve/sapling.html

Texture to make the tree more realistic - https://ambientcg.com/view?id=Bark008