Interactive Graphics: Project Report

Matteo Matera 1795339
Luca Polenta 1794787
Michela Proietti 1739846
Sofia Santilli 1813509

June 6th, 2021

SAPIENZA
UNIVERSITÀ DI ROMA

Contents

1 Abstract **3**

2 Chosen Environment **3**

3 Implementation **4**

3.1 Code Structure and File Organization 4

3.2 Camera 4

3.3 Lights 4

3.4 Textures 5

3.5 Geometry - Hierarchical Model 6

3.6 Rendering The Scene 7

3.7 Animations and Collision Handling 7

4 Levels of Play: Characteristics and Differences **8**

4.1 Level 1: Easy 8

4.1.1 Animations of the Main Character in Level 1 8

4.2 Level 2: Medium 9

4.2.1 Animations of the Main Character in Level 2 9

4.3 Level 3: Hard 10

4.3.1 Animations of the Main Character in Level 3 10

5 User Manual: How to Play & When the Player Loses **11**

6 Conclusion **12**

1 Abstract

A web version of the famous mobile game "Crossy Road" has been implemented as the final project of the interactive graphics course. The game features different difficulty levels, several environments, various user customizations in the main menu and different animations. Some aspects are based on the original game, while other features are the brainchild of programmers. The project was implemented using the advanced library "ThreeJS". The animations were implemented manually in WebGL via javascript.

2 Chosen Environment

ThreeJS[1][2] is a Javascript library for 3D web computer graphics based on WebGL. The latter is a low-level implementation that allows OpenGL to be executed on the browser, while ThreeJS is a high level implementation of OpenGL that hides the pragmatic and time consuming coding of gl programs, vertex-shader, fragment shader, buffers, projections and rendering. To include all its features, it has been imported via website references. The main features of this library that have been used are:

- THREE.Scene[3] - it allows to set up what and where is to be rendered by ThreeJS. It sets up the gl programs, data for buffers, vertex shader and fragment shader (that are individually specified every time an object3D is added to it), so this is where are placed objects, lights and cameras.
- THREE.Camera[4] - it allows to specify the view volume of the scene. It must be associated with a certain projection, which in this case is the PerspectiveCamera[5] projection.
- THREE.WebGLRenderer[6] - it is responsible for linking the shaders with the gl programs and setting up the 3D graphics in the canvas element. It is called to render by the rendering function.
- THREE.Object3D[7] - it is the base class for most objects in ThreeJS. It is a key component as it provides a set of properties and methods for manipulating objects in 3D space. Some of the most important properties are:
  - modelViewMatrix: it is passed to the shader for computing the position of the object.
  - position: a Vector3 representing the object's local position.
  - rotation: a Vector3 representing the object's local rotation in radians (Euler angles).
  - scale: object's local scale.
- THREE.Group[8] - it is almost identical to an Object3D and its purpose is to make working with groups of objects easier. One of its properties is the set of "children", which allows you to create a parent-child relationship between various groups and/or structures. This property is fundamental because through groups it is possible to define a hierarchical model[13][14] among the objects present in the environment.
- THREE.OBJLoader[9] - it is a loader for .obj resource. The OBJ file format is a simple data-format that represents 3D geometry in a human readable format as the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. It was used to import most of the 3D elements present in the various levels of the game.
- THREE.GLTFLoader[10] - it is a loader for glTF resources. glTF (GL Transmission Format) is an open format specification for efficient delivery and loading of 3D content. It can include different assets (textures, additional binary data and so on) in different formats. A glTF asset may deliver one or more scenes, including meshes, materials, textures, skins, skeletons, morph targets, animations, lights, and/or cameras. It was used to import the main characters of each level.

3 Implementation

3.1 Code Structure and File Organization

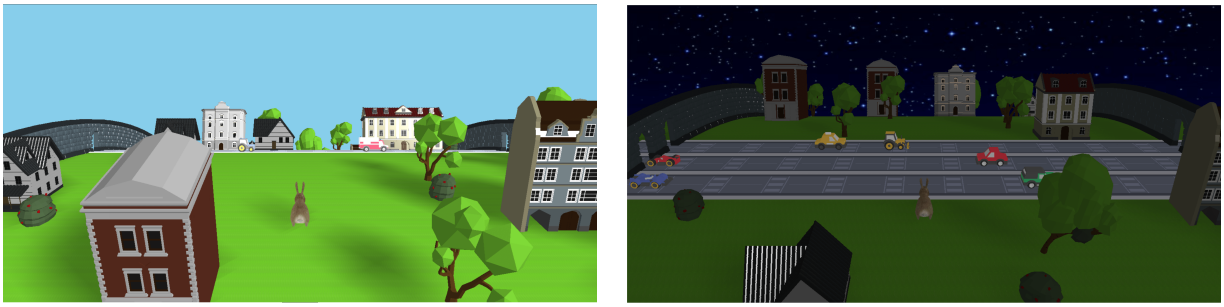
The source code has been divided into 3 main folders (assets, css, and js) and the HTML files are located in the macro folder that contains the whole project. The HTML are 3: main.html, which launches a menu to allow the user to select some modes and the level to play; gameover.html which is called when the player loses and allows him to return to the menu or to try again; and finally game.html which contains the canvas in which the game is loaded when requested by the main.html and with the settings chosen by the user.
Inside the assets folder we can find two subfolders that are "textures" and "models", which contain respectively the textures used in the world and all the preconfigured and imported models. Instead, in the css folder there are the files that define the aesthetics of the various html pages. Finally, in the js folder there are the files in which all the features of the game are implemented through ThreeJS.
Once the main calls the game.html file, the latter calls the corresponding javascript game.js which defines the game world and through a series of cascaded javascript, imports and sets the whole game.

3.2 Camera

ThreeJS provides a large set of cameras that can be chosen according to the needs of each project. In the current case, the PerspectiveCamera[5] has been selected. It uses perspective projection that is designed to mimic the way the human eye sees. It is the most common projection mode used for rendering a 3D scene. In each level, the camera is positioned slightly behind the main character so that you have a third person perspective of what the character is about to face. Finally, the characteristic parameters of the camera are fov, aspect, near and far. They have been set as follows: fov=35; aspect=window.innerWidth/window.innerHeight; near=0.1; far=3000.
Finally, for a design choice, it is the world that moves in front of the camera to show what the user must see at that moment and therefore during the game it never changes angle or position, except when the user loses the game: in fact when the main character dies, the camera is rotated and it frames the sky before showing the gameover screen.

3.3 Lights

There are two types of light implemented: ambient light and directional light. Both can be implemented using the ThreeJS THREE.AmbientLight[15] and THREE.DirectionalLight[16] functions.
The ambient light globally illuminates all objects in the scene equally. This light cannot be used to cast shadows because it doesn't have a direction. The light has been set to white (0xfffff) and the intensity varies depending on whether the game is being played during the day or at night: during the day it is 1, otherwise it is 0,5. In addition there is also a directional light. This light will behave as if it were infinitely distant and as if the rays produced by it were all parallel. This light can cast shadows by setting the camera parameters. It generates a depth map of the scene and the model used for the shadows was the Phong model. The parameters set are: mapSize which defines the width and height of the shadow map; the camera projection, which was set up as an orthographic projection using THREE.OrthographicCamera[17]; and the values of this projection including near, far and fov. In this way objects behind other objects from the point of view of light will be in shadow. During the day its intensity is 1, otherwise it is 0,15. So, since it is also active at night, it creates a slight shadow effect even at night. This effect has not been removed to make the objects show their three-dimensionality and can be justified by the presence of the moon. The effects obtained are the following:



(a) Day Light (b) Night Light

Figure 1: Graphic Differences Between Day and Night in Level 1

3.4 Textures

The textures used in this project are both color textures and bump map textures. The management of the textures took place partly automatically and manually: the models imported via OBJLoader[9] did not require any changes, while the characters imported via GLTFLoader[10] and some elements of the environment were handled manually.
The first programmed color texture concerns that of the floor on which the main character plays. In the first level, it is obtained as a color texture deriving from a jpg image of a lawn. Since the lawn image was very realistic and not too inherent in the stylized look of the game environment, it was only used to create subtle veins on the lawn that remain a stylized homogeneous green color. An explanatory image is shown in the chapter of lights. ThreeJS's MeshPhongMaterial[11] function was used to generate this texture: this function generates the material using a non-physically based Blinn-Phong model for calculating reflectance. Instead, in the second level it is obtained through a bumpMap that simulates the effect of snow. To do this, ThreeJS's MeshStandardMaterial[12] function was used. It allows you to define a base color on which to then apply the bump texture. The base color is a light gray (0x8b8b8b) which will then be further lightened by the lights and the bump map has been set to be 100% visible. The effect obtained is the following:

Figure 2: Bump Map of the World in Level 2

Furthermore, as regards the texture of the glTF models, ThreeJS's MeshStandardMaterial[12] function was used. It was also used in this case because it is suitable for managing the complex texture to be applied to characters made up of numerous parts. In addition, it is also adequate to correctly manage how the external lights must affect the character's appearance: during the day they must affect 40-50% (varying from character to character), while at night 0%. This feature was set to prevent the main character from being over-lit during the day or under-lit at night.

3.5 Geometry - Hierarchical Model

The ThreeJS library allows to build geometries in a simple way. On the other hand, when an object is composed of several structures, it is possible to enclose them in a THREE.Group[8]. Through the "children" property of the Groups it is possible to create a hierarchical model[13][14] of all the elements present in the environment. The main feature of the hierarchical model is that by changing a property of a given group, such as position or rotation, that property will also affect all child groups and their children in turn. The main objects of the world is the ground on which the character and all the other objects are placed and it is a cylinder obtained from the THREE.CylinderGeometry[18] function. Furthermore, some of its properties vary according to the environment, such as the fact that in the first level a texture is applied to simulate the lawn, while in the second level a bump texture is applied to simulate snow, and similarly for the third level. Instead most of the objects placed on it were downloaded[22][23][24][25] from the internet and imported into the hierarchical model via THREE.OBJLoader[9] or via THREE.GLTFLoader[10].
In general, the hierarchical model of the scene that contains all these elements is structured as follows:

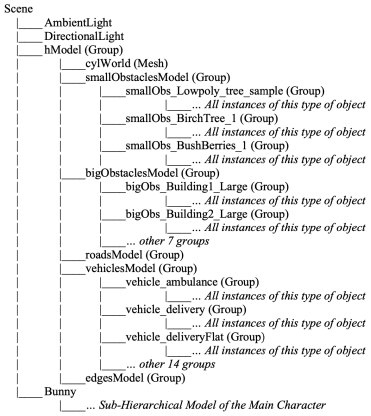
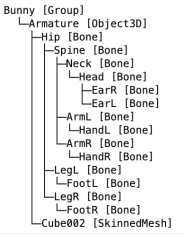


Figure 3: Hierarchical Model of the Scene

This hierarchical model is specific to the first level, but it assumes a different structure in the other levels. In fact, each level has different objects, so some leaf nodes that enclose all the instances of a certain type of object are created and added to the hierarchical model only when that objects are present in the world. For example, in the second level we will find a group that contains all the instances of a type of snow-covered tree, while in the first level this group is not present. In addition, the node containing the main character also varies according to the level chosen because each level has a different character. In addition, the main character of the first two levels has a more complex structure than the objects loaded with THREE.OBJLoader[9]. It is organized as an hierarchical model and it is loaded via THREE.GLTFLoader[10]. In the case of the first level, the rabbit has the following structure:

(a) Bunny



(b) Bunny Structure

Figure 4: Hierarchical Structure of Characters - Particularly of the Level 1 Character

The bunny is made up of a series of Bones[20] to which a SkinnedMesh[21] is linked. A bone is a part of a Skeleton[19] and they are almost identical to a blank Object3D[7] . The skeleton in turn is used by the SkinnedMesh[21] . The latter is a mesh where the shape of the Mesh is deformed by the bones. Therefore, by modifying a property of a Bones, such as position or rotation, it also affects all the other parts connected as its children and through this it was possible to animate the animal during the game. In addition, in the second and third level a bear and a man have been imported and animated respectively via glTF and they have the same logical structure as the rabbit, although they have more structural components.

3.6 Rendering The Scene

Once the user sets the preferences in the main menu and the level is selected, the world is generated while the user can view a loading screen while waiting until the level is fully loaded. Loading the world begins with creating the environment in which the main character interacts. We start with the creation of the ground, which is the lateral surface of the cylinder. Then the lights are added to the scene. Finally, the elements above the cylinder are generated, including walls, static obstacles and dynamic objects. The characteristics of all objects, such as position, rotation, and scale vary according to the object itself so that all appear in the correct size, position and rotation. The generation of the walls takes place along the 2 lateral perimeters. Bridges can be spawned when there are roads or walls otherwise. The other objects instead vary in position randomly each time the level is reloaded so as to have always different challenges. The obstacles themselves then change from level to level according to the theme of the same. At the end of this step, the main character is inserted, which varies according to the chosen level. During the generation of any object, some control function are performed that verify the absence of objects in the area where you want to make the object appear. This allows to avoid the overlapping of the objects and slightly reduces the computation as it ignores the generation of some elements. Just in case two vehicles are about to overlap, the second generated is repositioned. In addition, no items are generated in the main character's initial position.

3.7 Animations and Collision Handling

In the game there are several animations both automatic and manually activated by the interaction with the user. An automatic automation is about the obstacles that can kill the main character. In the first level they are represented by cars in motion, in the second they are snowmans or trains in motion and so on. Their animation is obtained by calling iteratively a function in the render function that translates all the cars in the game by 0.3 along the x axis in the direction of travel of the road. When a car reaches the edge of the playing field, it is repositioned at the start of its lane and starts its animation again. Colliding with a car or being hit by it because you were already on the road involves the automatic game over. The change of position of the vehicles and the checking of collisions with the cars are done within the render in game.js . Instead, an animation that is activated manually is the one that concerns the movement of the main character. It can move forward or sideways, but it cannot go back. Each time the player moves forward, the world, that is the cylinder, is rotated and with it all the other objects rotate through the use of the hierarchical model. On the other hand, when the player moves sideways, the world is not rotated and the main character first rotates in the direction it wants to go, then its movement animation takes place and finally rotates again to return to facing forward. Furthermore, regardless of the direction the character is made to move, the main character performs an animation. It varies character-by-character and will be better illustrated in the next chapter. In addition, the animation of the movement of the main characters is not performed when they are near a static obstacle that prevents their passage, such as a building or a tree. These static obstacles vary from level to level and will be better illustrated later, while the checks to avoid crossing static obstacles are carried out by comparing the distance between the position of the main character and the position of these obstacles (all in the global coordinates computed through the hierarchical model).

Finally, there are controls that prevent the user from performing multiple movements at the same time: therefore, if an animation is in progress, any button the user will press to perform a further movement, will have no effect.

4 Levels of Play: Characteristics and Differences

The game features 3 levels that have different difficulties, settings and characters. Each level can be played both day and night and this setting can be chosen in the game's main menu. Furthermore, the animations of the main characters also vary from each other as each character has different characteristics. In addition, the world is infinite: its structure is a rotating cylinder and once a spin is completed the player can continue playing as long as it does not lose. Each step forward adds one point to the score. Furthermore, there are spinning coins in the game that if taken add 10 points to the score. Finally, sound effects and background music were also included in the game. The original Crossy Road game music was not found and therefore a secondary music from the Super Mario Galaxy game was selected. The song was set to start over when it ends. Instead, a simple sound effect occurs upon the character's death. This sound is the same in all levels and occurs before moving on to the gameover page. Finally, they are present in the specific sound effects for each level that will be illustrated in the following paragraphs.

4.1 Level 1: Easy

The easy level features a setting very similar to the style of the original Crossy Road. The character is a rabbit who has to cross roads busy with different types of cars (17 types in total). Roads can have up to 3 consecutive lanes in both directions. In additions, the static obstacles are 3 types of trees and 9 types of buildings of different sizes. Their presence and position is chosen randomly at each restart of the level. This additional feature has been added to create new variety and challenges for the player who will use the game.

4.1.1 Animations of the Main Character in Level 1

The animation of the bunny in level 1 was obtained by changing the rotation and position parameters of the rabbit in webGL and it corresponds to a jump in which all its limbs are animated: the front legs tend to go forward, while the hind legs tend to go backwards. The ears are also animated: during the ascent they bend backwards as if pushed by the wind and then during the descent they return to their original position. At the same time the rotation of the body varies by leaning upwards during the jump and returning to the starting position during the descent. Furthermore, if the player attempts to move forward in the world, the rabbit jumps, but does not change his position because the world rotates below him. If, on the other hand, it moves sideways, in addition to jumping, the value along the x axis of the body also changes by a certain positive or negative quantity compared to if it moves to the right or to the left. Before running the animation, the rabbit rotates in the direction it needs to jump and stays in that direction when the jump is complete, just like in the original game. The animation of the rotation does not happen gradually for a design choice: animating the rotation would have resulted in a slow animation in its entirety and the character could have died too easily when crossing the streets. Instead speeding up the animation would have resulted in too sudden a movement that doesn't give the player enough time to fully appreciate it. Furthermore, even in the original game there is no rotation animation, so it was not implemented for reasons of design and fidelity to the original game. In addition, however, a sound effect has been added when the rabbit jumps in any direction. This sound is very reminiscent of a stylized videogame jump, therefore sufficiently themed with the main character. Finally, some frames of the animation can show how the body varies during the jump:

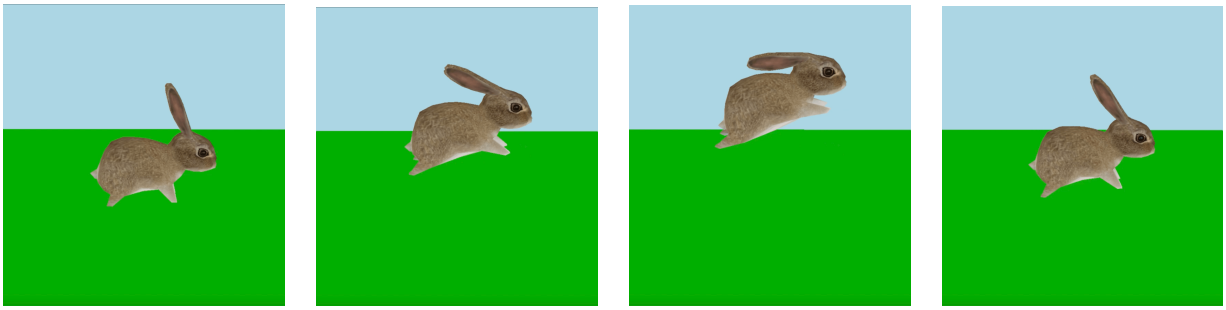


Table 1: Frames of The Rabbit Jump

4.2 Level 2: Medium

The medium level has a snowy setting. The snowy effect is achieved through a bump texture that has been applied to the world. The main character is a bear that has to cross either railways or snow-covered roads on which two snowmen glide. This particular type of road can have up to 3 consecutive lanes in both directions. In addition, the bear may encounter ice sheets full of obstacles and on which he will slide. As for the static obstacles, they are snow-covered trees, benches, woods, rocks, gifts and candies. Their presence and location are chosen randomly each time the level is restarted. This functionality has been added to create new varieties and challenges for the player who will be using the game.

4.2.1 Animations of the Main Character in Level 2

The animation of the bear in level 2 was obtained by changing the rotation and position parameters of the character in webGL as for the level 1. When the user presses one of the appropriate keys, the bear performs a walk by alternately moving its hind and front legs. If the user chooses to continue sideways, the bear rotates and turns in the direction it needs to go and at the end of the animation it rotates again to return facing forward. On the other hand, when the user makes the bear move forward, it does not rotate and continues straight, while at the same time the world and all the objects on it are made to turn. Some explanatory frames of this animation are:

Table 2: Frames of The Bear Walk

Also in this level the rotation of the bear has not been programmed when it changes direction for the same reasons explained in the previous paragraph. In addition, a sound effect has been added during the walk of

the bear that recalls the sound of footsteps on the snow, therefore themed with the environment of the level. In addition, when the bear encounters the ice sheets, it loses its balance and slips. Therefore, during the whole path characterized by the ice sheets, it slides forward. To do this, the world continues to spin without the user pressing any key. However, the user can choose to slide left or right to avoid obstacles. Pressing the side arrows or one of the other corresponding buttons on the keyboard leads to a movement along the respective direction by a certain amount. To move in the same direction you have to take the corresponding button again. When the ice sheets run out, the bear returns to its original position and then the player can starts walking again in the next steps. A further caveat that has been added in the code is that there will be no roads with moving obstacles immediately after the end of the ice sheet. This has been added to prevent the bear from colliding with a moving obstacle and losing the game. In addition, the bear does not stop sliding exactly when the ice sheets run out, but one step later. This feature has been incorporated to add realism to the context as it is as if the bear still holds some force which made it slide. Furthermore, if in the strip of land after the last ice sheet there are obstacles and the bear takes them, then it dies. Also this feature has been inserted to have realism in the scene as it is as if it were taking at full speed an obstacle that would lead to death. Some explanatory frames of this animation are:

Table 3: Frames of The Bear Glide

4.3 Level 3: Hard

The hard level is set in a monstrous world that is reminiscent of Halloween. The world has a slime green color and the main character is a little girl who has to avoid obstacles. This world is slippery, so the main character will slip all the time once the first forward step is taken. In this level the roads are made of rock and are crossed by gelatinous monsters. The world is also littered with various static obstacles, including trees without leaves, Halloween pumpkins, tombstones, graves and similar items. Furthermore, their presence and position are chosen randomly at each restart of the level to always have new challenges. In this level, hitting a static obstacle leads to game over like hitting a monster because they are taken in speed.

4.3.1 Animations of the Main Character in Level 3

The animation of the girl in this level was obtained with the same technique of the other two levels. Since the world is slippery, in this level the girl does not walk and will remain seated all the time. Once the up arrow or the corresponding keyboard key is pressed it starts sliding uncontrollably and the player can only move the side arrows or the corresponding keyboard keys to slide left and right and dodge obstacles. When the buttons corresponding to the lateral movements are pressed, the child slides to the corresponding side by tilting the body and moving the arms up and down. This movement was designed to simulate a little girl

having fun as she tries to keep her balance during the slide. This animation takes place on both sides with slight differences on the movement of the arms. Furthermore, in this level there are no sound effects because the main character does not walk like in the other two levels. Some explanatory frames of this animation are:

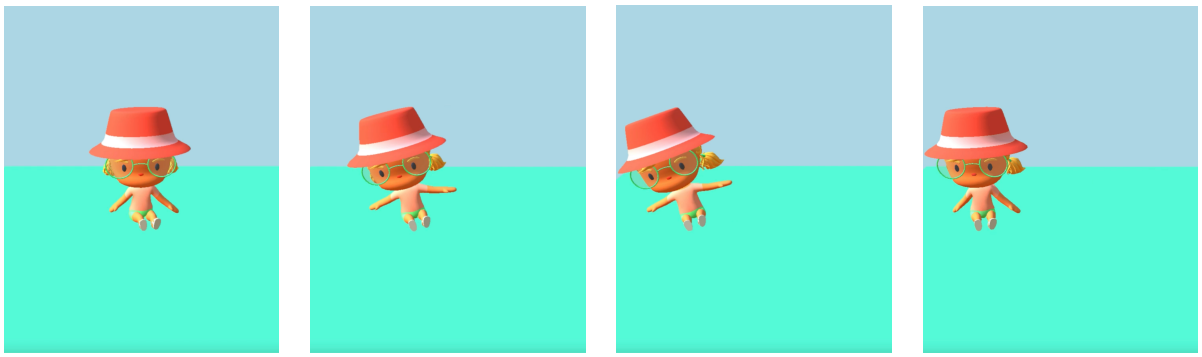


Table 4: Frames of the Girl Sliding to the Left (Right From the Player's Point of View)

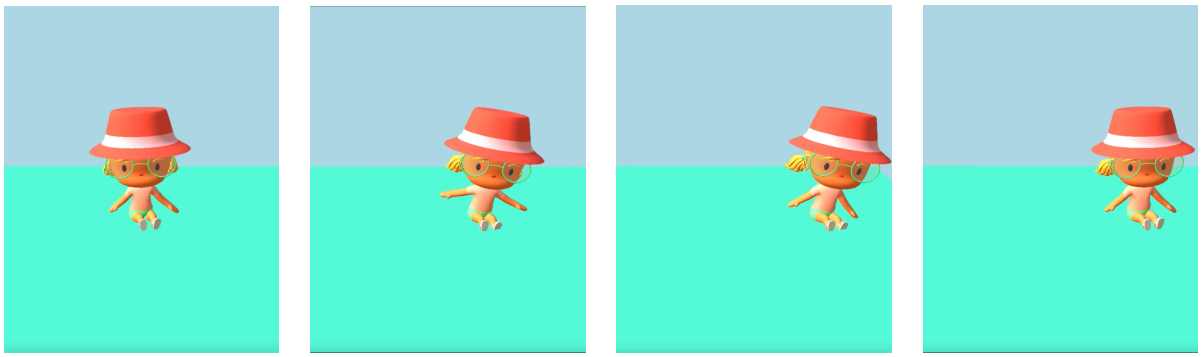


Table 5: Frames of the Girl Sliding to the Right (Left From the Player's Point of View)

5 User Manual: How to Play & When the Player Loses

When starting the game, the user can interact with the main menu using the mouse to choose some game settings, such as whether to set it during the day or at night, and the desired level. There is also a tutorial button that displays a guide to the buttons to press during the game.
The player can control the main character of the game in two ways:

- W, A and D (S is disabled because there is no going back).
- $\leftarrow$, $\uparrow$ and $\rightarrow$ ($\downarrow$ id disabled because there is no going back).

These buttons are valid for both walking and sliding in level 2. During the game and when the player loses, there are two buttons on the screen: one allows to restart the game while the other allows to return to the main menu to change levels and/or settings.
The player accumulates points as it progresses in the world. Each step forward adds a point, while the side

steps add no points. Another way to add points is to take the spinning coins in the game that add 10 points. The player can lose in various ways and they vary from level to level:

- Level 1: In this level the player can only die when he is run over or thrown into a moving obstacle. Static objects only block passage, but do not kill.
- Level 2: In this level the player can die either by being run over or thrown into a moving obstacle, or by sliding on the ice and hitting an obstacle on it. He can die even if there are objects at the end of the slippery path and it hits them because this type of collision is considered to be hit at speed. This additional death type has been added to maintain some realism in the game.
- Level 3: In this level the player can die by hitting any obstacle, whether it is moving or not. This is because if the obstacle is moving then it is considered as a hit, while if it is static it is considered to be hit in speed and therefore realistically the character would die.

6 Conclusion

The final result of this project is reasonably acceptable: it provides a fluid and interactive experience for the player, combined with an easy to use and always original environment due to the random generation of the setting and obstacles. Compared to the original game, some simplifications have been made, such as the absence of water to cross, but these shortcomings have been replaced with additional challenges and animations, such as those in which the player slips and must dodge further deadly objects. In general, all the graphic and technical features of the project are characterized by a theoretical background to better understand the most flexible, modular and feasible way to implement common properties such as the generation of the hierarchical model, obstacles, lighting and anti-overlap or collision controls. Furthermore, all the requirements have been met as there are:

- A complex hierarchical model, which is appropriately used throughout the game to apply changes to the world;
- Lights and textures: two lights of different types and various textures of various types, including color, for example on objects or on the world in the first level, and bump map, for example on the world of the second level;
- User interaction that allows to change the difficulty level and the day and night setting. The camera does not interact directly with the user, but can vary according to the user, i.e. when the main character dies;
- Animations created manually and of various kinds that also exploit the hierarchical model.

Every aspect of the gaming experience has been carefully curated and the overall result appears to be excellent as some select external test players have also reported good reviews regarding playability and interaction with the game. To finally conclude this report, the proposed project appears accurate in its initial inspiration and original in the introduction of new peculiar characteristics.

References

[1] Three.js : https://threejs.org/

[2] Three.js Fundamentals: https://threejsfundamentals.org/

[3] Scene in Three.js : https://threejs.org/docs/#api/en/scenes/Scene

[4] Camera in Three.js : https://threejs.org/docs/#api/en/cameras/Camera

[5] Perspective Camera in Three.js : https://threejs.org/docs/#api/en/cameras/PerspectiveCamera

[6] WebGLRenderer in Three.js : https://threejs.org/docs/#api/en/renderers/WebGLRenderer

[7] Object3D in Three.js : https://threejs.org/docs/#api/en/core/Object3D

[8] Group in Three.js : https://threejs.org/docs/#api/en/objects/Group

[9] OBJLoader in Three.js : https://threejs.org/docs/#examples/en/loaders/OBJLoader

[10] GLTFLoader in Three.js : https://threejs.org/docs/#examples/en/loaders/GLTFLoader

[11] MeshPhongMaterial in Three.js : https://threejs.org/docs/#api/en/materials/MeshPhongMaterial

[12] MeshStandardMaterial in Three.js : https://threejs.org/docs/#api/en/materials/MeshStandardMaterial

[13] Hierarchical Model - article n.1 - in Three.js : https://www.programmersought.com/article/55513469364/

[14] Hierarchical Model - article n.2 - in Three.js : https://www.programmersought.com/article/74993687094/

[15] Ambient Light in Three.js : https://threejs.org/docs/#api/en/lights/AmbientLight

[16] Directional Light in Three.js : https://threejs.org/docs/#api/en/lights/DirectionalLight

[17] Orthographic Camera in Three.js : https://threejs.org/docs/#api/en/cameras/OrthographicCamera

[18] Cylinder Geometry in Three.js : https://threejs.org/docs/#api/en/geometries/CylinderGeometry

[19] Skeleton in Three.js : https://threejs.org/docs/#api/en/objects/Skeleton

[20] Bone in Three.js : https://threejs.org/docs/#api/en/objects/Bone

[21] SkinnedMesh in Three.js : https://threejs.org/docs/#api/en/objects/SkinnedMesh

[22] Site number 1 for downloading GLTF models: https://creazilla.com/sections/3-3d-models/tags/649-gltf?page=1

[23] Site number 2 for downloading OBJ models: https://free3d.com/it/3d-models/obj

[24] Site number 3 for downloading OBJ models: https://sketchfab.com/tags/obj

[25] Site number 4 for downloading GLTF and OBJ models: https://opengameart.org/users/quaternius