Day and Night (Interactive Graphics Final Project)

Salvatore Cognetta 1874383, Daniele Appetito 1916560

Introduction

This project was created for the Interactive Graphics course at Sapienza Università di Roma. In this project we were tasked to use hierarchical models to create an animation. Lights and textures were also required in the models. Finally we had to implement some user interaction (i.e. switch lights on or off, change colours, move viewpoint , etc.).

We decided the best way to approach this project's requirements was to create a game (which we decided to call "Day and Night").

Day and Night is a simple platforming game where a character has to jump on various platforms to get to the final goal. However, some of the platforms are visible only in the daytime, while others only at nighttime. Thus the player must switch from day to night as he/she traverses the level in order to find the right platforms to jump to.

Environment

For this project we used babylon.js in javascript for the majority of the functions; from the UI to the mesh imports, as well as the sound. The majority of models were downloaded from royalty-free websites. All the sources for assets can be seen in the list below:

List of imported files and their sources:

- The main character(Figure 1) model was downloaded from "cgtrader.com", then was slightly altered in order to create a hierarchical model (limbs were separated and labelled as children nodes of the torso, this was all done in blender).

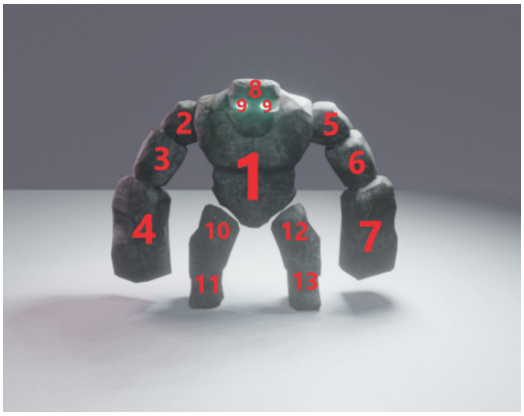

Figure 1: Image of the rock golem model showing all the nodes. Node 1 is the main parent node, the rest of the nodes follow from it in hierarchical order (i.e 2-3-4 and 10-11 are two different hierarchies with the same parent node, 1).

- The lamppost model (Figure 2) was imported from "sketchfab.com", then was altered in blender to remove the circular base. This model was also made hierarchical in blender.

Figure 2: Image of the lamppost model showing the hierarchical nodes.

- The flag model (Figure 3) was imported from "sketchfab.com" and later separated into three hierarchical nodes in blender.

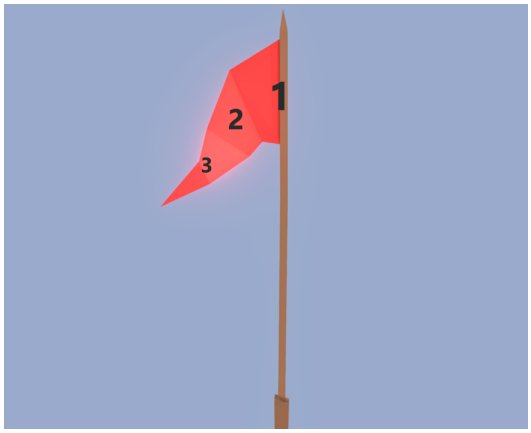

Figure 3: Image of the flag model showing the hierarchical nodes.

- All the music and sound effects were downloaded from "freesound.org". This includes: "$background.wav$", "$footsteps.wav$", "$ground_impact.wav$", "$jump.wav$", "$light_switch.wav$", "$menu_background_music.wav$", "$menu_select.wav$", and "$win.wav$".

- Finally all the images for the platform textures and backgrounds were found from quick google image searches, with the exception of the main menu background ("*temp_menu_image.jpg*") and the controls menu image("*info.jpg*") (which were both created using paint 3D).

Libraries and Tools

As stated in the beginning of the previous section, we used babylon.js for our project and thus the Libraries we used were all parts of said environment. More specifically we used: "babylon.js" for our basis, "babylonjs.loaders" to load 3D models into our scenes, "babylon.gui" to create interactive user interfaces, and finally we used "ammo.js" as our physics engine.

Technical aspects

Babylon.js is a real time 3D engine using a JavaScript library to display 3D graphics in a web browser via html. We used their official "how to" page to help us implement most of the functions used in this game.

Scenes

We created 3 different scenes; a Main Menu from which you can choose what level to play and look at the controls, a tutorial level that teaches you the game mechanics, and a level 1 scene where you must simply get to the goal.

To create a scene we used the "BABYLON.scene(engine)" function which, given an initial engine ("BABYLON.engine") as input creates the basis for a 3D scene. The camera is set up similarly with "BABYLON.FreeCamera". These functions are used in all the aforementioned scenes.

Meshes

The platforms were made using a function that creates a box-shaped mesh "BABYLON.MeshBuilder.CreateBox", these were then scaled and subsequently moved to desired positions. The floor and walls of the levels were created in the same manner. For the day/night platforms we implemented a boolean that would trigger the rendering of certain meshes during the day, and others during the night. The imported models mentioned under the "Environment" section of this report were added using a similar command "BABYLON.Sceneloader.ImportMesh". Some models had to be saved as gltf files, this was done through the "export as gltf" option in blender.

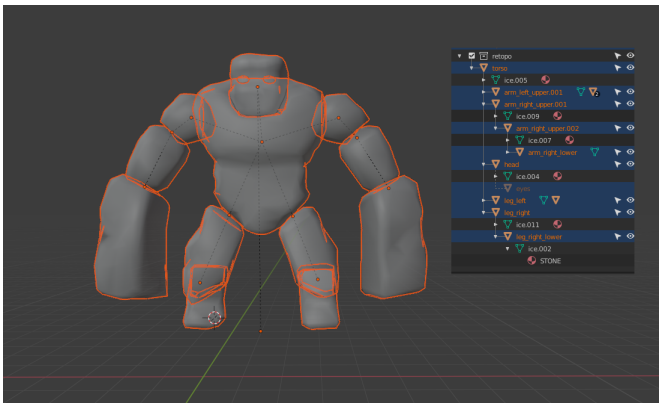

Figure 4: Blender model of the stone golem showing all the separate nodes and their hierarchies

Animations

For the animations we used key frames ("createAnimationKeys()"), and babylon.js interpolated the position of the models at each frame automatically. We had to make a different animation for each node separately. To create a ""full" animation action we grouped up all the node animations in "BABYLON.AnimationGroup" functions, which we initialised at a later notice.

Sounds

The sound system for this game was simply implemented by using the "BABYLON.Sound" function. Once the sounds were imported they were then initialised inside the scene-creation functions and used in different contexts.

Physics

As stated earlier, we used the ammo.js physics engine, which is not fundamental to the game but we wanted to test out its features. In particular for when the hero reaches the flag goal, and finishes the level or tutorial, we created 50 balls (placed in random positions in the level) that fall from the sky, to which we added "physics impostors". In fact, to allow interaction between objects, the physics engines uses an impostor, which is a simpler representation of a complex object, to which we add some properties such as mass and restitution (the amount of force the body will "give back" when colliding). The impostors are added also to all the platforms and the hero mesh, doing so we have a physics interaction between the created balls and the environment (hero plus the platforms).

Collision

For the collisions we used the Babylon "checkCollisions" property of a mesh imported or created in the framework. In particular for each platform and each imported object, if this property is set to true, two meshes cannot move if there is a collision detected. We used the built-in function "moveWithCollisions" for moving the mesh and checking "collisionable" objects in that direction, if so the mesh is blocked and there is no movement. This type of collision, however, has a problem. In fact the collision is detected right in the center of the meshes, and for this reason we had to create a bounding box around some models (the hero, the flag and the lamppost). The type of bounding box used is the AABB - axis aligned bounding box - which can be seen in debug mode as a transparent box around the mesh. Using this bounding box around the character we can use the "intersectMesh" built-in function of Babylon and check if there is an intersection between a platform/model and the main character. For example to prevent the main character from intersecting with an upper platform while jumping, we check if there is a collision between the AABB bounding box of the hero and the upper platform; if this collision is detected the character stops moving. The same is done for lateral collisions.

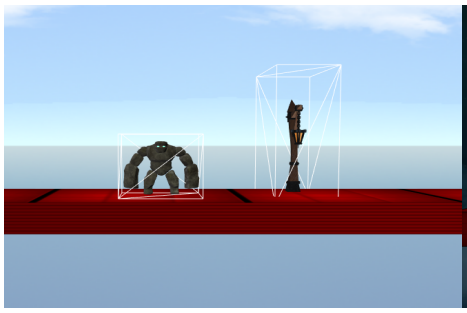

Figure 5: Bounding box around the hero and the lamppost

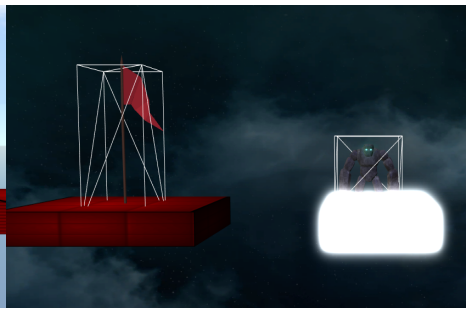

Figure 6: Bounding box around the hero and the flag goal

This type of collision detection is used also in the tutorial, when the hero pass through the lamppost is fired a popup with all the needed information, and also while reaching the goal flag. In fact there are two AABB bounding box also around the lamppost and the flag goal, and the intersection between these two and the hero AABB is done inside the checkLampTutorial and the checkGoal functions.

Interactions

Controls

For the controls of the hero(stone golem) model we created a big if statement where once a specific keyboard input is detected, a corresponding action is

carried through(such as moving left or right), as well as an animation. Once the key is released or another is pressed the state changes thus also changing the animation. The light switch control was done in a similar fashion.

Lights

Both the tutorial and the first level have 2 sets of lights, one Hemispheric light ("BABYLON.HemisphericLight") and one directional light("BABYLON.DirectionalLight"). The former is always on and is used to better illuminate all the models. The latter changes intensity when the game goes from day to night, on top of that, the colour changes from a dim yellow (during the day) to an intense dark blue (during the night). As an added effect, a glow layer was added to the scene with "BABYLON.GlowLayer". This function makes emissive materials glow, meaning that if an item's texture can be set to emissive, it will glow, in our case this function was used to light up the golem's eyes and the lantern.

UI

We created pause menus for the levels (Figure 7) to allow the user to change the volume of the music and effects, as well as adding buttons for a "controls" screen and for returning to the main menu. These user interfaces were created using "BABYLON.GUI" functions and simply disabling the background scene once the menus were called (in order to truly pause the game). The "GUI" functions were used to create the gray overlay as well as the text-blocks("GUI.TextBlock"), sliders ("GUI.Slider"), and the buttons ("GUI.Buttons").

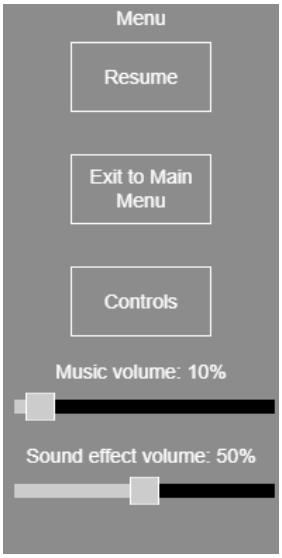

Figure 7: Example of a pause menu in the game