Final Project

Interactive Graphics

Giorgio Leporoni – 1944533

Francesco Pro – 1944191

1. Game goal and commands

Woody is lost in a forest with its broken car. You have to find among the map the pieces to repair the car. After repaired the car drive it through the portal to complete the game.

The commands to play the game are:

- **W, A, S, D:** to move Woody along the forest.
- **E:** to grab objects from the ground.
- **C:** to enter the car after collected all the objects.
- **Z:** to switch between first and third camera person (if allowed by the game difficulties chosen).
- **Q:** to turn on/off the lights (torch, bonfire, car lights).
- **Arrow keys:** to drive the car along the forest.
- **Space bar:** to jump.
- **ESC**: to pause the game and open the menu.

2. Used libraries

To develop the project, we used the following libraries:

- **WebGL** - to render graphics inside the browser.
- **Three.js -** is a js library, that we used to create the game scene, control lights, cameras, materials, objects and to upload third part 3D model

- **Cannon.js** - to model the physics inside the game. It allows to associate to each three.js object a physic shape to model it as a real-world object. In our case we attached a physical shape to each of the modelled object (Woody, trees, car, etc.…), but we will explain it later in the report.
- **Tween.js** - is a js library used for the objects' animation. We used it to animate Woody and the bonfire.

3. 3D Models

We downloaded all the used 3D models from a free-license website: *www.sketchfab.com* and thanks to three.js we were able to upload them.

3.1 Woody

This is the model used for representing the main character "Woody". It is composed of bones that allowed us to change its position during the game. The bones together create a hierarchical model. In fact, there are parts of the body that are influenced by the movement of other parts.

Using tween.js we create different animations for several contexts. In fact, it has a **walking animation**, an **idle animation**, that is the animation when the character is in a fixed position, and at the end another animation for walking when it holds the torch.

Woody can also jump with a specific command, and to do this we simply change its bones positions.

All the bones are identified by a name, with the preimplemented three.js function *getElementByName()* we can access directly to that specific bone and control it with a rotation or a movement in the space. With tween.js for the animation we simply specify the initial position of the bones we want to move, and the final position. Then it creates the movement using interpolation. It is also possible to start and stop the animation in every moment simply calling *start()* and *stop()* or put it in an infinite repetition. In detail for Woody, we start idle animation when it is initialized and stop it when some movement button is pressed, starting the walking animation with or without torch.

3.2 Decorative objects

All the objects shown above are just decorative objects, for those objects we did not implement any animation.

As visible in the game the trees are more than three, we upload these three models at the begin of the game and then when we build the environment, we just clone each model more times to generate a forest, in this way we save a lot of computation. The hedge (last figure above) is used to constraint the forest and don't allow the player to go outside the forest.

3.3 Interactive object

3.3.1 Bonfire

It contains a light that can be turned on or off by Woody pressing a specific button. To emulate a bonfire, we added continuous rotation animation to it.

3.3.2 Car

This is the model of the car we used. Woody can enter or exit it. For it we did not implement any animation, we simply move it in the space based on which movement button is clicked by the player. When the car moves, its anterior wheels rotate in the driving direction and its posterior wheels rotate to simulate the round car effect. To emulate its headlights we inserted a light, that can be turned on or off when Woody is inside it.

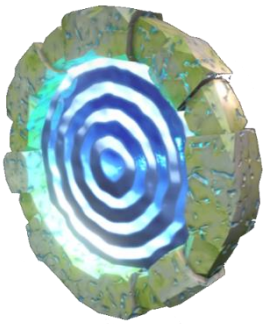
3.3.3 Car pieces

These are the collectable car pieces in order to repair the car. A tank of fuel, a key and a wrench. These objects are inanimate but each one contains a white light that automatically turn itself on when Woody is near one of them, with the aim of highlight the object. We did it with a function that is continuously running inside the render function, and it checks the distance between each object and Woody, if this distance is under a threshold, it turns on the light of the specific object (more than one if Woody is near multiple highlightable objects).

3.3.4 Torch

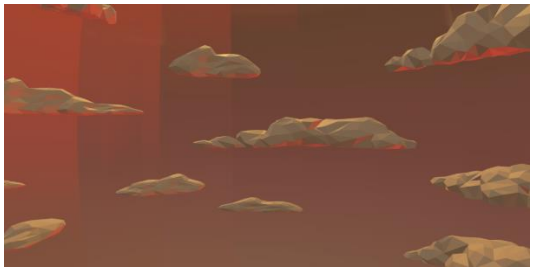
This is the model used for the torch. It is inanimate but used to emulate a torch, it contains a light that when grabbed by Woody can be turned on or off to illuminate the environment. It is held by Woody's hand, following its movement. When in first person camera, moving the mouse is possible to rotate through the vertical axis the torch to change the illuminated area.

3.4 Portal



This is the portal that Woody must hit to complete the game. It contains a white light to illuminates the environments around it.

3.5 World environments



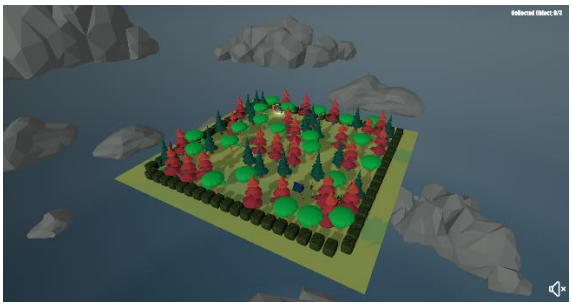
These are screens of the different selectable world ambient. They are three 3D spheres representing the day ambient, the evening ambient and the night ambient. Each one of them is associated with a corresponding light that hold the atmosphere. They are placed in a position and with specific sizes to sorround the forest map and appear like a sky.

All the models mentioned above are initialized at the begin of the game. Some of them are associated to a corresponding CANNON body with different characteristics (sizes, mass, material). To associate a CANNON body to a 3D model, is necessary with the preimplemented cannon functions, create a basic shape (cube, rectangle, sphere) with specific sizes and material. Is not possible to associate directly a cannon body to a 3D model, so to solve this is possible to put equal

both the body and the model positions, in such a way that the position of the 3D model follows the one of the body.

4. Map

We built the environment step by step placing each object in specific position. We started creating the ground plane. Then we placed the decoration objects described above. As said, for the tree we clone the imported model changing each time the position. Then we upload Woody and the remaining objects, highlighting the map constraints with the hedges. The following image show the map:



5. Cameras

To give the user the best experience with the game, we set up different camera views. A third-person camera and a first-person camera, that can be switched using a specific button mentioned in the command section. With three.js we created one generic camera and based on which type of camera the user selected, we just change the position of the generic camera to give the best view. This camera is also used as the view camera when Woody enter in the car, we just change the position of the camera to give the best view. We decided to use just one camera and change dynamically its position to save computation of having multiple camera objects.

The following images show the camera views:

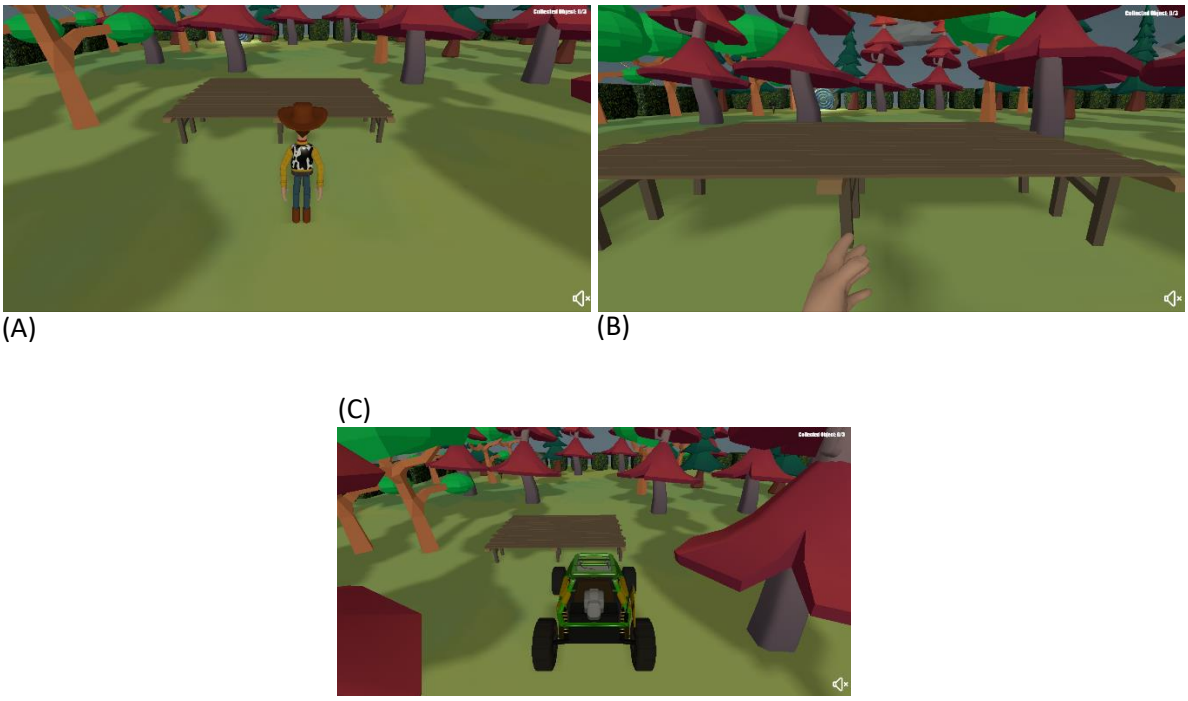


Figure 1: (A) The view in third-person camera (B) The view in first-person camera (C) The view from the car

6. Lights

We inserted 9 lights inside the game. Starting from the Ambiental light, for this purpose we use just one *ambient light* changing its color based on the time part of the day selected in the settings menu. Then we use a *directional light* to simulate the sun and give shadows to the objects.

We added 5 *spotlights* to emulate a cylindrical light for the torch, the headlights car and for the three car pieces. Finally, we added two point *lights* one for the bonfire and one for the portal.

All these types of light are preimplemented by three.js, we just initialize them and put in the correct position after chosen their properties.

As said previously is possible to interact with some of the mentioned lights pressing a specific button (see command section) as for the torch, car headlights and the bonfire. The lights of the three car pieces are turned on/off automatically with the distance from Woody.

When lights are turned off, instead of removing them from the scene, we just set their color to *#000000* saving computation.

Below there are some images that show the lights mentioned above:

7. Shadows

In the menu settings is possible to enable or disable the shadows for the objects that are in the environment map, setting to *true* the *shadowMap* of the *renderer*. Given an object in tree.js we can have two type of shadow settings. The first one is the shadow generated by the object, while the second option that can be set is the shadow that the object could receive on its surface from another object.

These two options can be set by the parameters *castShadow*, to set the shadow generated by the object and *receiveShadow*, to set the shadow generated by another object. They can be set *true* or *false*.

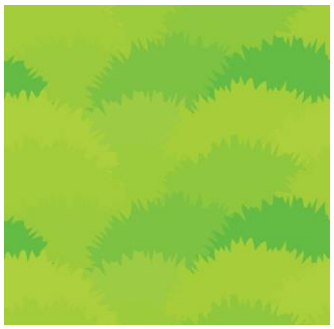
As visible from the image above, those are the shadows generated by the objects because of the directional light.

8. Texture

We applied a simple texture of grass on the ground of the map, to give the appearance of lawn for the forest.

To use textures in tree.js must be imported the *TextureLoader()*. After imported it is possible to use the function *load()* to upload and apply the texture.

The following image shows the texture we applied:



All the textures of the objects are preimplemented in the 3D models imported.

9. Audio

As for real games, we applied a forest soundtrack downloaded from Internet. When the game starts the music start automatically and is possible to mute or unmute it with the corresponding icons at the bottom right corner.

Is possible to apply audio thanks to a set of preimplemented tree.js functions and classes.

The above image highlights the mentioned icons.

10. Game settings

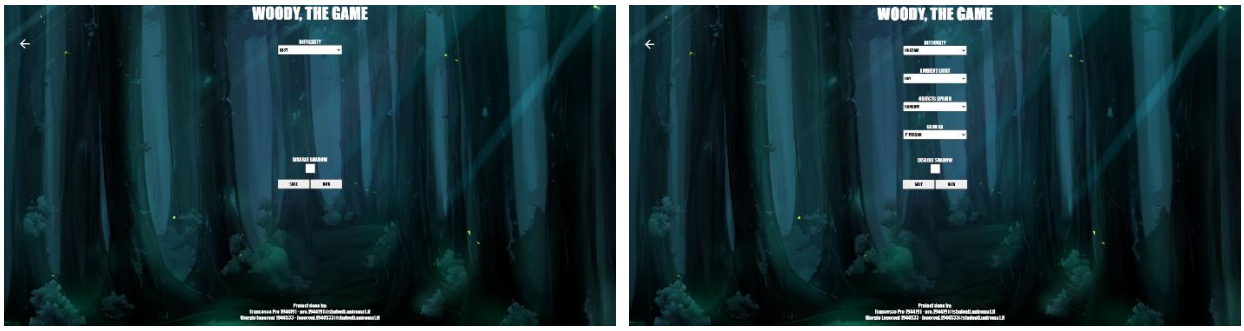
In the menu settings is possible to enable or disable the shadows, in addition is possible to set the difficulties. There are three types of difficulties:

- **EASY** - where there is day light, the player could use both first person camera and third person camera, and the objects that must be collected have fixed positions.
- **MEDIUM** - where there is an evening light, the player could use again both the type of camera but in this case the objects positions are not fixed.
- **HARD** - where the is no light, the player can use only the first-person control and the positions of the objects are not fixed.

There is an additional modality that is:

- **CUSTOM** - where is possible to set manually all the settings mentioned above.

All the game setting are passed from the HTML file to the js file using the local storage variables. So, these variables are set in the HTML and then read from the js file.



On the left image is visible the settings menu when chose a fixed modality, while on the right are visible the settings when chose the *custom* modality.