Carrots Hunt

Interactive Graphics - Final project

Giacomo Acitelli, mat. 1643776
Giorgia Di Pietro, mat. 1751145

Semptember 26, 2021

1 Introduction

The project is a simple game realized using the *Babylon.js* library. The basic idea is allowing the user to move a dummy in a countryside and in the scene there is also a rabbit trying to eat the carrots around the area. The goal of the game is catching as many carrots as possible before the rabbit gets them.
There are three difficulty levels and the user can select one of them from the main menu. In the easy mode there are only five carrots and the rabbit is quite slow, increasing the difficulty, the number of carrots increases too and the rabbit becomes faster. In every level the number of carrots is odd, if the player gets the most of them, he wins. In addition there are two different game modalities, day and night, in which the light is managed differently.
Once the game ends, both in the winning and losing case, the user can chose to play again the same level or come back to the main menu in order to select another setting.

2 Commands and Instructions

The user selects the level of difficulty and the desired mode, then he clicks the play button for starting the game. In order to move the character, you need to use both keyboard and mouse. By pressing the *"w"* letter of the keyboard the dummy walks forward, while he walks back using the letter *"s"*, jumping is enabled by pressing the space bar. Through the mouse, or the arrows of the keyboard, you can change the moving direction of the dummy.
In order to catch a carrot, you just need to cross it, so it disappears and the score panel is updated.

3 Environment

We composed the whole environment exploiting the meshes offered by the *Babylon.js* library or importing objects found online.

3.1 Skybox and ground

In order to simulate the appearance of a sky, we applied six suitable images to the insides of a large skybox cube. So we defined at first a big cube surrounding the scene and then depending on the selected mode we load a day or night skybox countryside images. Then we introduced a ground simulating the grass and we delimited the actual game area creating a rectangular perimeter of walls. We used a physic engine library (*Cannon.js*) to give some physical parameters to all game entities. In this way the dummy is closed inside the perimeter without the possibility to overcome it, the dummy and the rabbit are able to jump on the ground and the fences in the environment.

3.2 Meshes and textures

Focusing on the elements of the scene, we used a lot of different textures to give the game a good graphics. We filled the scene with many typical elements we usually found in a countryside, so we imported different meshes and we took care of each object texture to make everything realistic. In particular some meshes already included a texture or a color, in other cases we handled or refined the look through the texture properties of the specific material (*diffuse*, *bump* and *specular*).
The used meshes are trees, bushes, fences, rocks, carrots, logs and flowers. We simply imported many instances of the elements and made changes on their position, size and orientation for diversifying the environment. The only element created by scratch is the fountain, so we used the example provided in the *Babylon.js* documentation with rotational symmetry using a lathed mesh to create it. Fine particles are recycled to produce a water effect, then we just refined the parameters for managing the flow.

3.3 Lights and Shadows

We used two types of lights, directional light and a hemispheric light.
The directional light is defined by a direction and it is emitted from everywhere in the specified direction with an infinite. This light was useful to simulate respectively sun light and moon light by changing their parameters. The hemispheric light is an easy way to simulate an ambient environment light.
In order to manage the shadows we used the *shadowGenerator()* offered by *Babylon.js*. This function uses a shadow map: a map of our scene generated from the light's point of view. The two parameters used by the shadow generator are the size of the shadow map and which light is used for the shadow map's computation, in our case the directional light. Finally we specify the shadow rendering for each object in the scene with *addShadowCaster(mesh, includeDescendants)*.

4 Game characters

The characters are imported models provided by *Babylon.js*. They are composed by a skeleton, a common structure used to animate models composed of multiple meshes. In particular our models are composed by a hierarchy of bones, each defined by an id, a parent and a transformation matrix. The dummy skeleton is composed of 66 bones with no texture at all[1]. The rabbit hierarchical model is much simpler because it is composed by only 3 bones with already associated textures and colors.
The characters interaction with the elements of the environment are managed attaching to each of them a bounding box defined as mesh parent. The bounding boxes are shaped on the characters size, so when the collisions happen, the interactions are more precise.

4.1 Animations

We implemented the animations of the dummy and the rabbit models by using movements and rotations performed in the render loop cycle. Basically we exploited the structure of the two meshes, in particular firstly we studied the associated bones and with them we managed the different pieces of the bodies. The dummy animation is controlled by the player as explained in the instructions, the rabbit is only able to walk forward and jump the obstacles in order to catch the carrots.

4.1.1 Dummy animation

Before implementing the animation, we set the initial position of the character rotating a bit the arms on different axes in order to make his standing posture more realistic. The dummy animation consists in three main groups: walk forward, walk back and jump.
The basic idea for the walking phase was taking one of the legs as point of reference and using a counter for managing its extension during the run. Of course in order to mimic the natural movement, the upper and the lower legs has different rotations, the feet follows moves and the front leg opens more than the posterior one. So while walking, the dummy moves alternatively back and forth both the legs, the arms and his head.
This pattern is applied for walking forward and back, the only difference is that in the second case the opening range of the legs is smaller because we suppose that the dummy has more difficulty, so he is slower.
The jump animation is implemented with the physics impulse applied to the character and pushing him up, consequently the dummy falls down due to the gravity force.

[1] *Babylon.js* library doesn't provide any particular texture for the dummy character, so we decided to leave his original characteristics.

4.1.2 Rabbit animation

The rabbit animation consists in walking forward and the jumping. It represents the rival, so the user cannot control it, it is an autonomous character that continuously tries to catch the carrots. This mesh is composed by only three bones (head and two feet), so its movement is very basic, but still effective.
The walking mechanism works as for the dummy, basically we use a counter in order to manage the feet opening.
The jump animation is used in order to overcome the obstacles the rabbit could meet during its trajectory.

5 Game structure

The main scene is the countryside, but there are other scenes in order to manage the user interaction during the game.

5.1 Main Menu Scene

Once the user opens the game, the main menu scene is presented. It has its own camera, an *ArcRotateCamera* which acts like a satellite in orbit around a target and always points towards the target position pointing at the center. We defined a hemispheric light because here we only need an ambient one.
The *Babylon.js* GUI library was used to create the interactive user interface, in fact in the menu the user has the possibility to choose the desired settings for the game through *TextBlock*, *StackPanel*, *Simple Button* and *RadioButton* elements. The options that can be selected are the level of difficulty (easy, medium or hard) and the mode (day or night), then start playing. In order to understand the goal and the rules of the game, the user can also access to the instructions scene by pressing the instructions button.

Figure 1: Main Menu Scene

5.2 Instruction Scene

Once the user clicked the instructions button, the instructions scene is presented. It is very simple and explains the goal and all the commands of the game with a simple and intuitive image. We defined again an *ArcRotateCamera* and an hemispheric light for the ambient light.

Figure 2: Instruction Scene

5.3 Loading Scene

Once the user click on the play button from the main menu scene, or every time the game is loaded, the loading scene is rendered. It simply shows a loading text until all the elements of the countryside scene are loaded. We chose to introduce this intermediary scene in order to give a feedback to the user about what is happening.

5.4 Countryside Scene

The main scene is the countryside where the game actually develops. The user can move the dummy character trying to catch the carrots. The environment is composed by several meshes simulating a realistic scenario as trees, bushes, fences, etc. The most important are the carrots because they have a central role in the game. For this reason we decided to highlight them in some way to capture the attention of the player, so we implemented a simple animation that consists in moving them up and down.
As explained before we used two types of lights, a directional one and a hemispheric one. Again there is an *ArcRotateCamera* pointing always on the main characters, so when the player moves, the camera follows the mesh. The camera orientation and its radius parameter can be controlled with the mouse movement. Since the walking direction of the player is linked to the camera orientation, by changing camera orientation we can control the dummy movement.
On the top-right corner of the scene there is a GUI element keeping updated

the score of the game. The information shown are the number of carrots caught respectively by the dummy and the rabbit.

5.5 Win and Lose Scene

The user arrives to the winning scene in case he caught more carrots than the rabbit. This scene is really minimal, it is just composed by *Simple Button* and *TextBlocks* GUI elements. It is used to notify the user about the win and to allow him to play again the same level or come back to the main menu and change the settings of the game.
If the user loses, so the rabbit caught more carrots, the scene is the same as in the previous case, but for the shown text because in this case the user is notified about the failure.

Figure 3: Win Scene

Figure 4: Lose Scene

6 Interactions

6.1 Physics Interaction

We chose *Cannon.js* to simulate the gravity force to the whole scene, then in order to enable it on objects we used physics impostor provided by *Babylon.js*. The two main characters, dummy and rabbit, can interact with the elements of the scene by hurting, moving or going through them. In order to manage the collisions we applied to all of them a box physics impostor specifying the associated mass and the restitution parameter for managing the elastic force due to a collision (*restitution: 0.0* for avoiding bounce). This method is mainly used to prevent a body from entering into another body. In fact in case of blow, if the character mass is much greater than the object one, the latter moves. Otherwise if the character has a smaller mass than the object, he isn't able to overcome it. Furthermore thanks to the physics engine we also avoids the characters pass through objects in the environment while walking.

6.2 Meshes collision interaction

In order to handle the collisions between the characters and carrots, the simplest way is using the *intersectMesh()* function provided by *Babylon.js* taking two parameters: the mesh to be checked and the precision of intersection. This function checks only once, so during the game it is called in the render loop *RegisterAfterRender()*. Since the carrots have to disappear once they have been caught, we dispose them.

7 Sounds

In order to make the gaming experience more enjoyable, we added music both in the background and in specific moments of the game.
Besides the background music in the countryside scene, we added two different sounds each time the dummy and the rabbit take a carrot.
Finally, we added victory and defeat sounds at the end of the game