KeyScape - Interactive Graphics Exam 2017/2018

Lucia Rodino' - 1562837

1 Introduction

This project has been done using WebGL and BabylonJS for the "Interactive Graphics" exam (Professor: Marco Schaerf) of the academic year 2017/2018.

My goal was to build a game similar to the games that I like the most respecting the requirements of the exam.

Indeed, as I really like "Final Fantasy", the game takes place in nature, in the middle of the mountains, where the player has to explore this landscape in order to find a magic key.

The main character has been trapped in this place, structured like a maze, by a criminal organization of scientists that is making an experiment on the survival of the human race. The magic key is a gate to the real world and it is the only way to escape from this maze. Indeed, when the player finds the key, he wins the game. This objective is not so easy to reach, cause in the whole land there are little rabbits which are actually robot bombs that explode if the player gets too close to them. Because of that, the character has to pass carefully near those rabbits because if they explode the game will be over.

Due to the fact that the aim of this game is to find the magic key to escape from the maze where you're trapped, I chose to name my game "KeyScape".

The project code is on GitHub and the game is playable through GitHub Pages.

2 General Aspects

2.1 Libraries

As I said at the beginning of this document, I chose BabylonJS to develop my project.

BabylonJS is an open source 3D engine based on WebGL and Javascript. It is the the most popular framework for building 3D games cause it provides built-in functions to help you implement common 3D functionality more quickly. For example, it has an implemented collision detection that is really useful for developing games. Personally I've used two libraries in my game:

- **"babylon.custom.js"** that is the base library necessary if you want to build 3D games using BabylonJS

- **"babylon.objFileLoader.js"** that is the library necessary if you want to import meshes or scenes.

BabylonJS provides also a complete documentation which helps you a lot, explaining the basic features of this framework.

2.2 HTML pages

In order to play game through the browser, HTML pages are needed. The main HTML page is the **index.html** page, that is the starting page of the game. In my project, in order to give a visible structure to the game, I've divided it into multiple html pages:

- **"index.html"** : the beginning page connected through a button with the menu page;
- **"menu.html"**: the menu page with buttons that let you choose if starting the game or read the instructions or see the credits;
- **"instruction.html"**: the instruction page where I explain the game's rules;
- **"credits.html"**: information about this project;
- **"gameIntro.html"**: the page just before the real game, where there is an introduction to the game, and two possible options that the player can choose (I'll explain those options later in the report) which are passed to the game through the local storage of the browser;
- **"main.html"**: the main page where the game is playable;
- **"win.html"**: the page visible when the player wins;
- **"loss.html"**: the page visible when the player loses.

2.3 Credits

The soundtrack used in my project has been downloaded from youtube, its name is "The maze runner" and it is property of *TheGringoMedia*.

2.4 Introduction to the technical aspects

Now on in this report, I'm going to explain all the technical aspects of my project and how I've realized each of these aspects, starting from the environment, continuing with the models, imported or created by myself, the animations, the victory and the game over and finally the user interactions.

3 The environment

A realistic environment would have sky, uneven terrain, shadows, fog and different lighting effects. I've chosen to build my realistic environment, developing all those aspects.

3.1 Sky

I've added a simulated sky to my scene using a skybox. This is created using a large standard box, and defining the *skyboxMaterial* with a **reflection texture** of six images, one for each face of the cube.

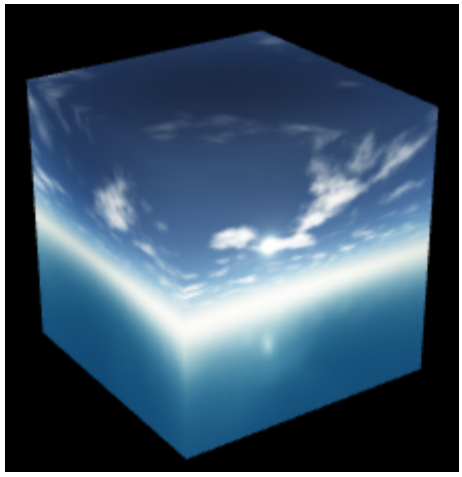


Figure 1: skybox with texture

3.1.1 Sky image Texture

Six images are necessary to create a skybox, all of them should be square and of the same size which is best as a power of 2, eg 1024px x 1024px. The name for each image should have a common part followed by a position given by _px, _nx, _py, _ny, _pz or _nz corresponding whether it is on the positive (p) or negative (n) x, y or z axis. BabylonJS has some examples of skybox image textures in the Playground folder on GitHub. For my game I've chosen two different image textures for the skybox: the *TropicalSunnyDay* and the *nebula* as you can see in figure 2.

(a) TropicalSunnyDay (b) nebula

Figure 2: The BablonJS' skybox textures used in my game (a)TropicalSunnyDay skybox texture and (b)nebula skybox Texture

Moreover, when it's nighttime in my game, I've decided to add **fog** to my scene using the BabylonJS **fogMode** in order to obtain a more realistic reduced visibility if it's nighttime.

3.2 Uneven terrain

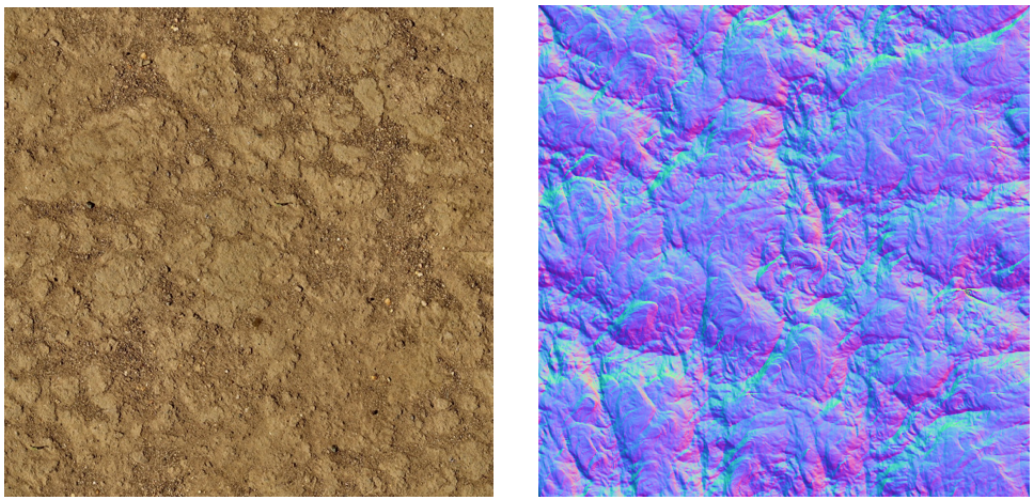
The best way to generate a realistic uneven terrain is using **height maps** and the **terrain material**. Now I'll explain how I've used height map and terrain material to crate my landscape.

3.2.1 Terrain Material

The terrain material works with at least 4 **textures**:

- 3 Diffuse textures (required);
- 3 Bump textures (not required);
- 1 Mixmap texture (required), which represents the intensity of each diffuse texture according the channels R (red), G (green) and B (blue).

In my case I've used both the 3 **diffuse texture** and the 3 **bump textures** to make my terrain more realistic, cause the diffuse texture give to the terrain the "color" desired and the bump texture make it more realistic and not flat, as you can see in figure 3.



(a) ground diffuse texture (b) ground bump texture

Figure 3: An example of pair of textures used for my terrain (a)diffuse texture and (b)bump texture

For what concerns the 3 diffuse textures, I've used one for the grass and two for the mountains: the one in the image above and another darker than this one.

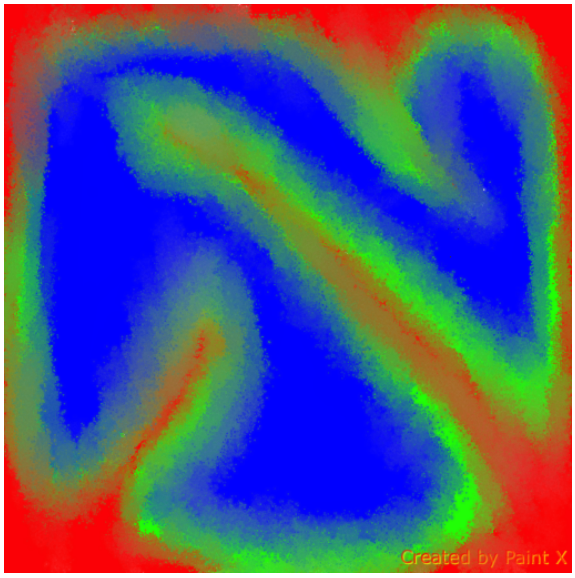


Figure 4: Mixmap texture

Finally, after having chosen the 3 diffuse and bump textures for my game, I've put them together using the **mixmap texture**, which represents the intensity of each diffuse texture. I made my **mixmap texture**, that you can see in this figure to the left, using *Paint X Lite*.

In other words, the Mixmap texture mixes the 3 diffuse textures thanks to the color channels RGB.

In my case, the blue color is for the grass texture, the green color is for the first texture of the mountains and the red color is for the second darker texture representing the top of the mountains.

Only with the terrain material, the ground is still flat; this is the reason why I chose to use a height map.

3.2.2 Height Map

The best way to generate realistic grounds is using height maps.

A height map is simply a greyscale image. This greyscale pixels of such an image provides the values for the height of the ground. I made my **height map**, that you can see in the figure to the right, using *Paint X Lite*.

Figure 5: Height map

This image has been used to generate my ground, using the different variants of gray of the picture. Indeed, this image is the elevation data for my ground. Each pixel's color is interpreted as a distance of displacement or "height" from the "floor" of the mesh. So, the whiter the pixel is, the taller the mountain will be.

Finally I've created my ground from this height map and I've applied to this ground the terrain material described above, and I've obtained my final ground as you can see in figure 6.

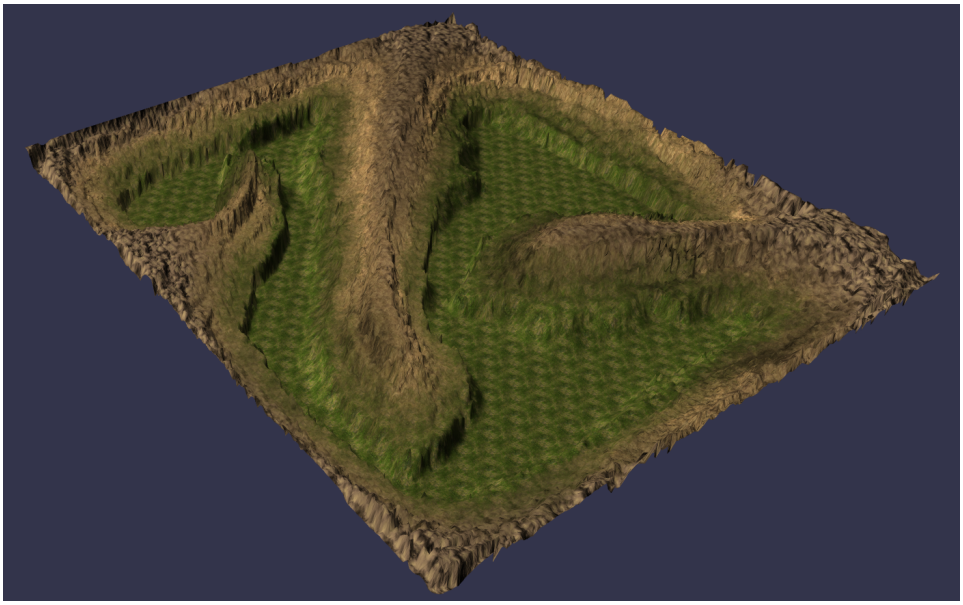


Figure 6: My final ground obtained combining the terrain material and the height map

3.3 Lights

Lights are used to affect how meshes are seen, in terms of both illumination and color.

BabylonJS provides four types of light in its library:

- The Point Light: a light defined by an unique point in world space. The light is emitted in every direction from this point.
- The Directional Light: is defined by a direction. The light is emitted from everywhere in the specified direction, and has an infinite range.
- The Spot Light: is defined by a position, a direction, an angle, and an exponent. These values define a cone of light starting from the position, emitting toward the direction.
- The Hemispheric Light: is defined by a direction. It is an easy way to simulate an ambient environment light.

I chose to use three of these four types of light:

- one Hemispheric light to simulate the ambient light; I've set the intensity to 0.8 if it's daytime and to 0.2 if it's nighttime;
- two Directional light to generate the shadow of my main character (I'll explain the shadows later);
- one Spot light to illuminate the key and generate its shadow.

4 Models and animations

I've imported 3 different meshes in my game: a key, some rabbits and the main character. I've made the key by myself using blender and I've imported the main character and the rabbits from the BabylonJS' library. Now I'll describe each of these models and their animations, starting from the key, than the rabbits and finally the main character.

4.1 The key

I've made the key using Blender and I've applied to the key a gold material using a color **texture**.

Figure 7: Two screens of my key in the blender editor

To export it in *.babylon* extension, I've downloaded the export plugin from the BabylonJS' GitHub repository; than I've exported it and imported into my game.

I chose the final position of the key, that is the victory position and I made a simple loop animation of the key, rotating on both x and y axis. I've decided to not

scale the key because, first of all, it is more visible to the player and because it is not a real key, but it represents the magical gate to the real world. It is placed in a cone of light with its shadow.

4.2 The rabbits

I've downloaded the rabbits from the BabylonJS website, and I've imported them with the textures already implemented in the downloaded model. Those rabbits are simple hierarchical models.

There are four or more rabbits in my game, and each one of them is placed in a specific position and animated. The animation is quite simple cause the rabbit moves only the paws as if he is jumping; each rabbit moves straight on and than turn around and move straight on to the opposite direction. The animation function is repeated in a loop as if the rabbits are guarding something.

4.3 The main character

The main character have been downloaded, as the rabbits, directly from the BabylonJS website. It is a complex hierarchical model.

I've made two animations of the main character: walk animation and run animation.

I've divided both the animations into six phases to make them as realistic as possible. The animations are made using the skeleton of the main character in a way that if I move a parent bone like the upper leg, all the children bones move with the parent, respecting the hierarchical architecture.

For what concerns the **walk animation**, the character moves the upper and lower legs, both the two parts of the feet, the upper and lower arms, the head and the torso.

For what concerns the **run animation**, the character moves the same bones as the walk animation plus the hands and two phalanx of each finger cause, when a person runs, usually clench his fist.

In order to correctly pass from an animation to another, I've put a starting phase which sets all bones from where they are to a base position before the animation starts.

Finally, the simplest animations are the two for turning right and left cause the character basically rotate him around. However, the interesting part of these animations is the **position of the camera**.

4.3.1 The camera and the character

I've used an **ArcRotateCamera** in my game, I've set the locked target, that is always updated, as the position of the character, so the camera follows the character. However, locking the target is not sufficient to make the camera turning around with the character.

Indeed, I've made the camera rotate with the character in a way that the player does not have to care about the position of the camera, cause the camera will see what the player sees. To do that, I update a value that I've called *alpha* in the turn left and right functions of the character, so that the camera rotate her view of the same angle as the rotation of the character.

4.3.2 shadows

The shadows in BabylonJS can be generated dynamically with a **ShadowGenerator**. This function uses a shadow map: a map of the 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. To generate the character' shadows I've inserted two directional lights and two **ShadowGenerator**. Than I pushed the character's mesh into the two **ShadowGenerator** and finally I've allowed the ground to receive shadows.

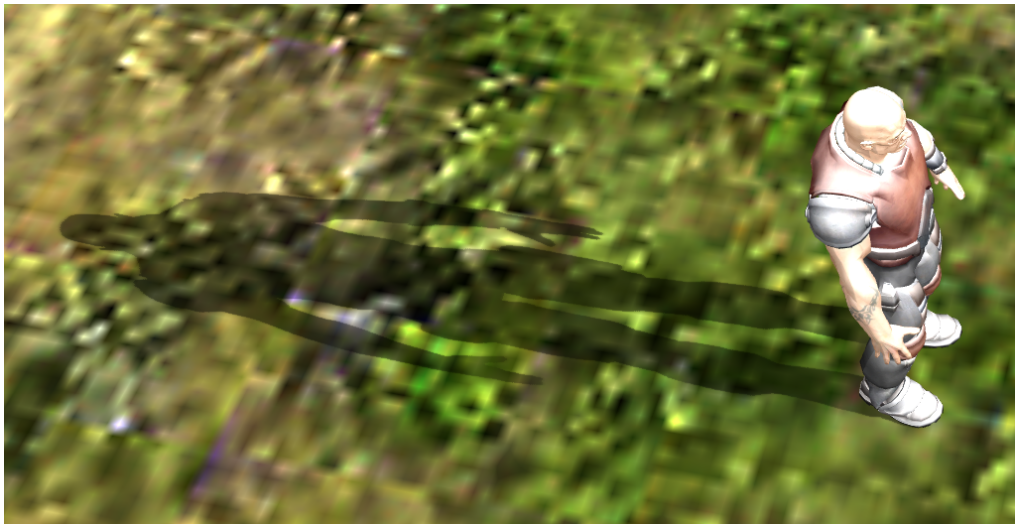


Figure 8: The main character's shadow in my game

4.3.3 Gravity

Finally I've inserted gravity in my game, in a way that, if we move the main character towards the mountains, he won't be able to scale them simply walking. With BabylonJS you can apply gravity to the scene by defining **scene.gravity** as a vector with the gravity desired on the y axis. Finally, I've applied gravity to the character too, after having created the mesh.

4.4 Interactions between meshes

The main character interacts with the other two meshes with two possible actions. If the character gets close to the rabbits the game will be over and if the character gets

close to the key, the game ends and the player wins. The functions which check the distances between the character and the other meshes are repeated in every render step and if the distance is less than the fixed win distance for the key or the fixed loss distance for the rabbits, the player wins or loses.

5 User Interaction

There are some user interactions in my game.

5.1 Main character movement

First of all, the user can control the main character using the arrow keys on the keyboard: if the player presses the up arrow key, the walk animation starts; moreover if he press the up arrow key plus the shift key, the run animation starts. Furthermore, if the left or right arrow key is pressed, the character turns to the left or to the right, with the camera. Finally, the user can choose to end the game, pressing the "esc" key on the keyboard.

5.2 The camera

When the user turns the character to the left or to the right, the camera follows the rotation about the same degrees and in the same direction, as I've explained before; anyway the user can turn around the camera or set it in the position that he prefers, using the mouse, and than the camera will turn with the character from that fixed user's position.

5.3 Daytime or nighttime

The user can choose to play as if it is daytime or nighttime, by checking a checkbox at the beginning of the game. By default it's daytime, so the skybox is the *"TropicalSunnyDay"* and the lights have an high intensity, but if the user checks the night checkbox, the skybox will be the *"nebula"* skybox, the lights will have lower intensity and the *fogmode* will be present in the game.

5.4 Difficulty

The user can also decide the difficulty of the game, choosing the value of a bar at the beginning of the game. The bar goes from "0" (beginner mode) to "3" (expert mode). In the beginner mode, there are 4 rabbits and the loss distance from them will be of "20". Each difficulty step adds one rabbit to the 4 rabbits and increases of 5 the loss distance. Therefore, in the expert mode, there are 7 rabbits and the loss distance will be of "35".

6 References

[1] GitHub repository -> `https://github.com/MarcoSchaerfCourses/keyscape`
[2] Game Demo -> `https://marcoschaerfcourses.github.io/keyscape/`
[3] BabylonJS WebSite -> `http://www.babylonjs.com`