Interactive Graphics
Final Project:

In the hall of the (ice) mountain king

Andrea Catalano
1968239

SAPIENZA
UNIVERSITÀ DI ROMA

Introduction

The main idea of this project is to realize a classical puzzle, in which the player has its movements prolonged until an obstacle or a border is encountered. The goal of the player is to use these obstacles in order to find a route toward the goal, usually found at the extreme opposite of the room. This simple implementation can also be the basis of a much complex system of rooms interconnected, be it serially (the player has to solve one room in order to get into another) or in parallel (e.g. two rooms one above the other, where the player can fall from the upper one, finding themself in an otherwise unreachable spot).

Framework adopted & external additions

Foundation stone of this project it's threejs library, by which means a neat graph scene has been implemented, with all its elements. On top of the core three.js I also used PointerLockControls.js, a useful threejs library to read inputs from keyboard.

All the models present in the scene have been handcrafted by myself, together with animations and collision map. The textures instead, together with the audio and the inspiration for the starting/ending menus, have been taken from various places on the internet.

Code structure

The code is structured in:
- 1 starting html page, containing the main menu
- 1 main html page, containing the actual game
- 1 ending html page, containing the victory screen and options to return to main menu or to replay
- 2 css files to help with html presentation
- 1 main js file, in which the context is generated, and then passed to the scene manager
- 1 scene manager js file, whose job is to set, manage and update the scene; in here the 3 model files are called
- 3 model js files (1 for the room, 1 for the player and 1 for the rocks), containing both data and functions to manage single-handedly their respective models
- 9 textures, either colour, normal and displacement
- 1 audio file to be reproduced during the game
- 2 background images to be shown in the menus
- 1 icon file
- obviously the needed libraries, cited before

Implementation

1. Event handling: first of all, the code sets and binds in the main.js the listeners to all the events we're interested to detect, namely the resize of the window and keyboard input. In the main we also handle the calls to scenemanager.js that will handle the events.

2. Scene initialization: thanks to threejs the code first creates the renderer, camera and lights needed by the scene, attaching to the camera the audio listener. It also sets the functions that will be called whenever the corresponding event will arise.

3. Model placement in the scene: With our scene ready, we set the desired models in it.

First, the model of the room, that is nothing else than the interior of a rectangular cuboid; its walls are textured with a combination of colour, displacement and normal textures, representing cold rocky walls, while the floor is created by applying both a colour and a displacement texture (almost not relevant for obvious reasons) to give an impression of ice; the ceiling is not rendered, since the camera is above it, facing its external face (and we're rendering only the inner ones); on the wall facing the camera there is a door, to quickly identify the goal of the player; to further help them locate the area in which they have to arrive, at the feet of said door lies a green square.

Once created the room and placed in the scene, we place the rocks that will obstacle the player; for the sake of diversity we have two kind of rocks, that actually differs only by the textures applied to them (both have a combination of two colour - displacement textures). The rocks are placed with the aid of a for cycle and a given map, that will also be helpful in computing the allowed movement of the player.

With our scenario set, we introduce the player model. It is a hierarchical model composed by 30 nodes, modelling a humanoid figure. The player model is the only model in the scene that doesn't have any texture applied.

4. Player movement animations:
The player moves with the classic WASD configuration, and their movement is fixed until it collides with either a wall or a rock.

When they are moving, it performs a "wiggle" animation, waving its arms in an attempt to keep balance. The animation starts naturally with a first quick phase where they raise their arms from the rest position, then waving it until they're safe and still again, lowering the arms to the rest position again.

When a movement is started, another animation is performed, and that's the backward "kick" the player has to perform in order to begin sliding on the ice. One leg flexes, while the other hastily runs backward, starting the desired movement.

When the player starts going towards a direction different with respect to the one they are facing, they steer toward the new movement direction. The animation is completed even if a collision is detected before its natural conclusion (i.e. the distance covered is too small to complete the animation naturally). In this way our player will always face the direction in which they are moving (or just moved).

5. Collision and win detection: Since the position of each rock is already known, the detection of collisions with rock is easily made by computing the distance (radial) of the player and each rock, comparing it with a threshold. Similarly the player is restricted in an area (delimited by the walls) by checking at each step that its position is not outside the bounds. The win condition is the presence of the player in a 2.25x2.25 area right in front of the exit door, computed as above.

6. Menus implementation: A starting menu introduces the game, making the player conscious of the controls and its goal, and awaiting for them to be ready to play. An ending menu communicates the achieved victory to the player, offering them to play again or to return to the main menu.

Direct response to the requests:

1. Hierarchical models: the player model has been implemented, composed of 30 nodes.

2. Light and textures: one ambient light and one point light adorn the scene, and all static objects have at least a colour and a displacement texture each.

3. User interaction: the user interaction is enacted by the capability of the human player to control their avatar in game, moving it around and trying to get to the goal.

4. Animations: several animations are implemented on the player, and are discussed in detail above. No library has been used to compute the animations.

Conclusions

Despise the consistent difficulties, both technical and personal, I had fun in making this project. Seeing a feature successfully implemented after hours of trials is always a great feeling, and overall being able in the end to develop a game, however simple, is and will always be a satisfaction. Most importantly I developed and honed my skills with higher level graphical programming than simple WebGL, and I'm overall satisfied with my work and the course.