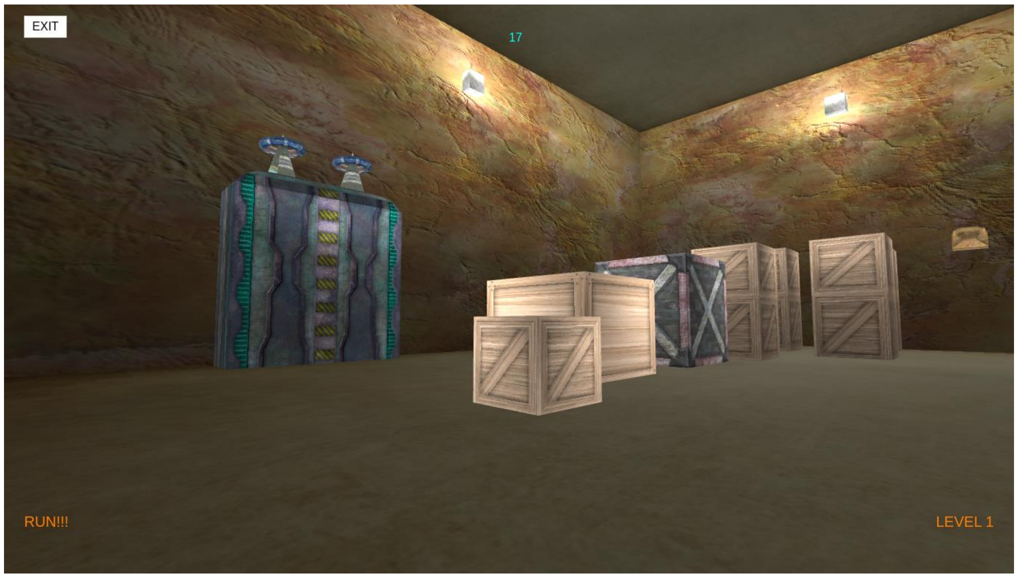
Seminarska Naloga
za predmet RAČUNALNIŠKA GRAFIKA

<Fleeting Time>

Ivan Nikolov - 63190378



Januar 2021

Introduction

This 3D FPS browser video game was developed using WebGL 2 and JavaScript. The game starts with a welcome screen where the basic instructions and a start button are available. The goal of the game is to get to the finish and do the task before the time runs out. The player moves using WASD, space keyboard keys and the mouse, the E key is used for doing the tasks.

Realization

Components of the game:

- SPACE
- CAMERA
- PLAYER
- LIGHT
- MODELS (walls, boxes, floor, celling, lamps...)
- BASIC PHYSICS
- BASIC ANIMATIONS
- LEVELS
- STARING SCREEN AND TEXT

Space

The space where the player can move is restricted in the internal space of one cuboid that represents the room. The floor, celling and the walls are realized using basic blender models. As noted above the player moves using WASD and space for translation and the mouse for rotation. The forward direction is calculated based on the current rotation. The typical starting position is x = 0, y = 0, z = 0, for the local coordinate system of the camera.

Camera

The camera as node in the scene is implemented in the class CameraNode, it has a reference to a projection camera that represents the field of view of the player. When the player moves / rotates all the needed transformations are applied to the camera node. At first the rotation of the camera caused some glitches (separate rotations along the x and the y axis caused rotation along the z axis). I solved this problem by keeping a special array for the rotation angles and then a creating a quaternion form them (quat.fromEuler). For the movement to work correctly the initial rotation along the x axis is set at -90°. Also, the axis of the local system of the camera are different from the global coordinate system (the y and z axis are swapped), so some minor changes were needed to the original code from the examples on the laboratory exercises.

Player

In the code there is not a special player class but, the player is represented as a bounding box (AABB) around the camera with minimum that is [-2, -2, -2] and maximum of [2, 2, 2]. This value worked best for the movement and the collision detection. The player (camera node) has velocity, maximum velocity, acceleration, friction with the surface, current height and staring height (used for the collision detection and the simulation of gravity). The player has to fix the generator before finishing the level. The generator is fixed by pressing E multiple times (determined in the level object).

Models (objects)

The objects in the scene were created using the basic meshes in Blender, most of the objects are cubes with applied textures. With the textures we get 'the dessert bunker' look of the scene. Some objects (two parts of the generator and the finish banner are animated and glow).

Light

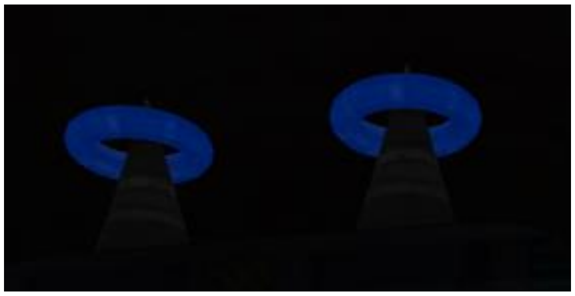
In the scene there is one light that is either on or off. The light reflects on the wall giving the illusion that there are multiple lights used. At the beginning of each level the lights are off (on low power) and the generator has to be fixed in order to turn them on. When the light is off the ambient, diffuse and specular color are set to darker shades.

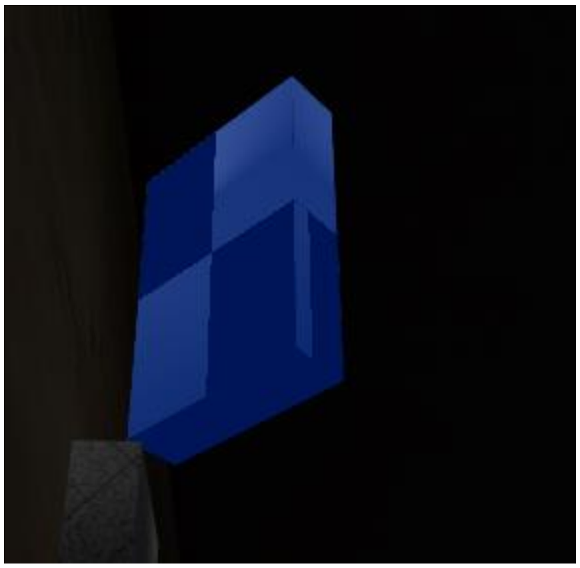
Basic physics

I tried to use a physics engine but had some problems with setting up the bounding boxes and decided that only collision detection and basic gravity is needed. So, I used the 17th example from the exercises and added some functions for the gravity and collision detection. The parameters for the bounding boxes are taken from the node attributes given by the gltf file and scaled by the node scale / 2. When the player jumps a constant vector of [0, 0, jumpValue] is added to the acceleration. The gravity works if the player is higher than his default height for that surface and pulls the player down the z-axis (local coordinate system). The game has weak gravity that is not realistic but works good with the nature of the game.

Basic animations

Three models have basic animations. The two rings on top of the generator have scalation and rotation applied to them, and the finish banner has only rotation. The animated objects have the uniform uAnimated in the fragment shader set to 1 which means a special glowing effect is applied to them (a mixture of the texture with blue color that changes depending on the time).





Levels

The parameters needed for the levels (END_TIME, FINISH_COORD...) are given in a special object in the Levels.js file, the levels index starts at 0 and increments as long there are available levels. I planned on adding different scenes for each level but, I did not have the time to make the Blender models.

Starting screen and text

The staring screen is a separate html page that has a link with the name 'Play' and takes us to the main application. Also, the basic controls and instructions are showed on the staring screen. Throughout the gameplay special messages are showed depending on the situation.

Possible improvements

The application could be improved with fixing the bugs and glitches:

- In rare cases the rotation of the camera does not work correctly, and the camera is 'thrown' along the axes.
- When restarting the player is sometimes placed in midair
- On some edge cases the collision detecting fails

In addition, new features could be added, for example: more tasks, moving obstacles, new scenes.

Conclusion

With this seminar I gained new knowledge on computer graphics from a practical perspective and gained insight on how game engines are made. Also, I learned the basics of JavaScript and web programming.

In the near future, I am planning to add new features and fix the glitches in the game.