Interactive Graphics Report

PacTeam
Edoardo Piroli - Luca Faraoni

20th September 2019

Contents

1 — Introduction

For our final project, we have decided to build a 3D version of the traditional Pacman game using the advanced library Three.js.

1.1 How to play

The game presents itself with the camera placed and oriented as in a first person game. The goal is to eat all the balls running inside of a maze, avoiding to be eaten by the ghosts. It is possible to move in each direction and to rotate the camera. The commands are the followings:

- 'W': move forward;
- 'A': move to the left;
- 'S': move backward;
- 'D': move to the right;
- $\leftarrow$: rotate the camera to the left;
- $\rightarrow$: rotate the camera to the right.

There are some bigger balls, the super-balls, which, if eaten, grant Pacman the possibility to eat the ghosts for a certain amount of time. Both balls or ghosts can be eaten exclusively moving forward(using the 'W' command) as pacman's mouth is located in front. Eating ghosts or balls increases the score based on the chosen difficulty. When touched by a ghost, Pacman dies and the game ends.

Before the start of the game, it is possible to change the difficulty in the settings menu, located at the top left corner. There are three different difficulty levels that change some game parameters, according to the following table.

Table 1.1: Difficulty levels parameters

Difficulty level	**N. of ghosts**	**Ghosts' spawn time**	**Super-ball time**	**Balls points**	**Ghosts points**
Easy	4	15s	20s	2	20
Medium	6	10s	15s	5	50
Hard	8	8s	10s	10	100

2 — Models

2.1 Maze

The maze is composed by a floor, many cubes representing the walls and also balls and super-balls. In order to build it as the original Pacman maze, we have used a matrix where each cell's value represents a different object, 1 for walls, 2 for balls, 3 for super-balls.

- *Floor*: the floor is a plane declared as follows

```
var floor = new THREE.Mesh(
    new THREE.PlaneGeometry(205, 211, 10, 10),
    new THREE.MeshPhongMaterial({map: textureFloor})
);

floor.rotation.x = Math.PI / 2;
floor.position.x = 100;
floor.position.z = -103;
```

- *Walls*: walls are formed by a unitary cube which is defined as follows

```
var unique_cube = new THREE.BoxBufferGeometry(5, 50, 5);
var unique_cube_material = new THREE.MeshPhongMaterial({map: textureWall};

//We use the same geometry and material to optimize rendering
var cube = new THREE.Mesh(
    unique_cube,
    unique_cube_material
);
```

  Each cube has its own position based on the cell it belongs to. Moreover there are 2 special cubes, which are defined as follows

```
var cube1 = new THREE.Mesh(
    new THREE.BoxBufferGeometry(5, 50, 5),
    new THREE.MeshPhongMaterial({map: textureTeleport})
);
var cube2 = new THREE.Mesh(
    new THREE.BoxBufferGeometry(5, 50, 5),
    new THREE.MeshPhongMaterial({map: textureTeleport})
);

cube1.position.set(0, 25, -110);
cube2.position.set(200, 25, -110);
```

  Such cubes, as described later, are used to create a portal as in the original game.

- *Balls*: balls are defined as follows

```
var unique_ball = new THREE.SphereBufferGeometry(0.75, 16, 16);
var unique_ball_material = new THREE.MeshPhongMaterial({color: 0xffff00};

//We use the same geometry and material to optimize rendering
var ball = new THREE.Mesh(
    unique_ball,
    unique_ball_material
);
```

  Each ball has its own position based on the cell it belongs to.

- *Super-balls*: super-balls are defined as follows

```
var unique_super_ball = new THREE.SphereBufferGeometry(1.5, 16, 16);
var unique_super_ball_material = new THREE.MeshPhongMaterial({color: 0xffd700};

//We use the same geometry and material to optimize rendering
var ball = new THREE.Mesh(
    unique_super_ball,
    unique_super_ball_material
);
```

  There are 4 super-balls positioned as in the original pacman game.

2.2 Pacman

Pacman is a 3D model imported as follows

```
var pacman_model;
var loader = new THREE.OBJJLoader();
loader.load(
    '3DModels/pacman.obj',
    (object) => {
        object.rotation.y = Math.PI * 50 / 126;
        object.rotation.z = 0.15;
        object.travers((child) => {
            if (child instanceof THREE.Mesh) {
                child.material.color.setHex(0xffff00);
            }
        });
        pacman_model = object;
        pacman_model.position.set(110, 6, -10);
    };
}
```

2.3 Ghosts

There are four different colors for ghosts, light_blue(0x00ffff), red(0xce3025), pink(0xffb8ff), and orange(0xffb852). Each color is associated to a different model, each loaded as follow

```
var GHOST_MODELS = [];
var loader = new THREE.OBJJLoader();

function loadGhost(loader, color) {
    loader.load(
        '3DModels/pacman_ghost.obj',
        (object) => {
            object.scale.set(4,3,4);
            object.travers((child) => {
                if (child instanceof THREE.Mesh) {
                    child.material.color.setHex(color);
                }
            });
            GHOST_MODELS.push(object);
        };
    }
}

loadghost(loader, 0xce3025);
loadghost(loader, 0x00ffff);
loadghost(loader, 0xffb8ff);
loadghost(loader, 0xffb852);
```

To instantiate a ghost, we use a class Ghost that clones a random model and sets a random position among 4 different predefined locations.

2.4 Play

In the homepage there is an object representing the text "PLAY". Such object is clickable and makes the game start.

3 — Lights and Textures

3.1 Textures

In order to load the textures we have used THREE.TextureLoader(); There are different textures for the following objects:

- **Floor**: For the floor we have used a sandy dark texture;
- **Walls**: For the walls we have used a grass-like texture;
- **Portals**: For the portals we have used an image representing a spiral as texture.

3.2 Lights

As for the lights we have chosen to use a very soft ambiental light, with color 0xffffff and intensity 0.3. In the homepage, in order to better display the maze's structure we have added a strong directional light, with same color as the ambiental one and intensity 0.8 which is removed as the game starts(upon a click on the PLAY object). Furthermore we have added a spotlight which moves following pacman, as if he was carrying a torch. The lights are defined as follows

```
var ambientLight = new THREE.AmbientLight(0xffffff, 0.3);
var dirLight = new THREE.DirectionalLight(0xffffff, 0.8)
dirLight.position.set(100, 80, -50);
var spotLight = new THREE.SpotLight(0xffffff, 0.8, 200, Math.PI/4, 1, 2);
```

In order to handle the change in orientation of the spotlight, we have added, as its target an invisible object in front of Pacman.

4 — Animations

The only moving objects are ghosts and Pacman. Pacman is moved by the user through the commands we have described in the introduction. As for the ghosts they are moved autonomously around the maze, although they differ from the ghosts of the original game as they don't follow any pattern but move randomly. In particular ghosts may change directions whenever they find themselves in a cross-road.

4.1 Collisions

In order to handle Pacman's interactions with the environment we have used raycasting. We have defined a raycaster as

```
var raycaster = new THREE.Raycaster();
// Setting the far property to 3 because we are interested only in close collisions.
raycaster.far = 3;
```

In particular at every rendering we cast 8 different rays starting from Pacman's center in the 8 main cardinal-directions. For example here is the code to cast a ray straight in front of pacman

```
raycaster.set(
    pacman.position, new THREE.Vector3(
        Math.sin(-camera.rotation.y),
        0,
        -Math.cos(-camera.rotation.y)
    )
);
```

For every movement-command('W', 'A', 'S', 'D') we use 3 rays to detect any kind of collision. In particular:

- '**W**': We use the North, North-East and North-West directions;
- '**A**': We use the West, South-West and North-West directions;
- '**S**': We use the South, South-East and South-West directions;
- '**D**': We use the East, South-East and North-East directions.

4.1.1 Walls

To avoid that Pacman could pass through walls we check for collisions in the direction Pacman is moving. In particular for each different translation-command('W', 'A', 'S', 'D') we check 3 different rays intersections with the walls' cubes. Moreover for the rotation-commands($\rightarrow, \leftarrow$) we check for only one ray intersection. If any collision is detected, the relative command is skipped.

4.1.2 Portals

When checking for the walls' collisions, if a portal is detected Pacman is teleported to the other one.

4.1.3 Balls

Likewise, for the balls we use 3 different rays intersections. If a ball is detected, pacman eats it(the ball is removed from the scene) and the score is increased. If a super-ball is detected pacman also gains the super-ball power. Whenever a ball is eaten, if it is the last one, the game ends and Pacman has won.

4.1.4 Ghosts