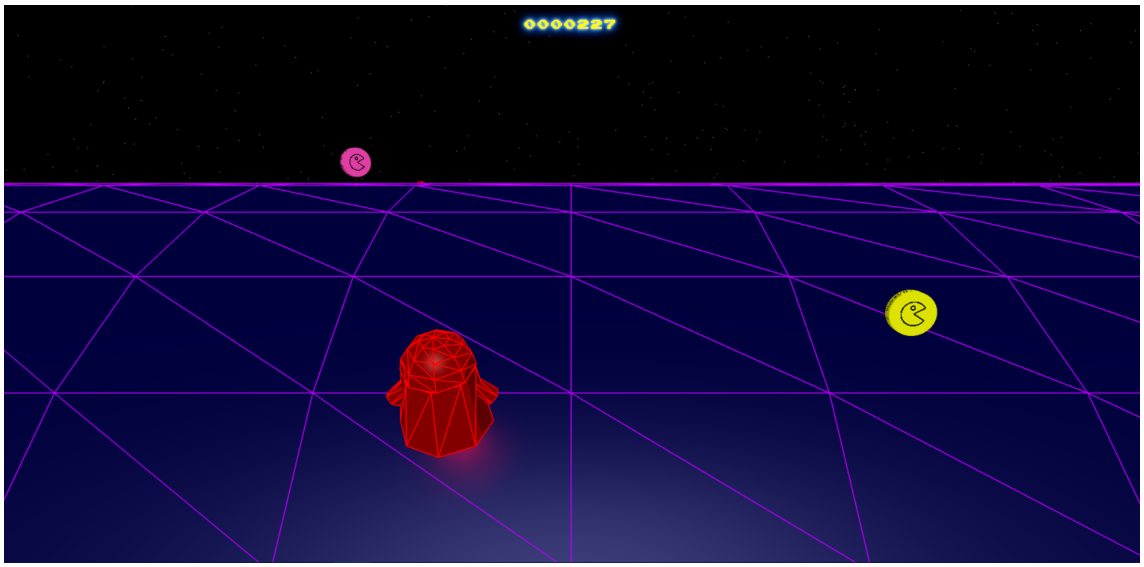
Engineering in Computer Science

Interactive Graphics

GHOST-MAN



Daniele Buonadonna 1855047 - Umberto di Canito 1851685

Contents

Chapter 1

Overview

1.1 Introduction

The game *Ghost-man* is a platform game based on the notorious game *Pacman*. In the following chapter, all the preliminary information will be listed, such as the libraries used, and the main idea of the game. Basically, the user play as one of the ghost of the *Pacman* game, and run in order to collect as much coin as possible. The more coin he gets, the higher the score will be. If the coin collected hasn't the same color of the ghost, the player lose. On this basis, there are three different modalities (*normal, hard* and *crazy*), for which the difficulty grows. In addition, in the *crazy* mode, the color of the ghost (aka the user character) can change randomly. How all these functionalities are implemented is listed in the following sections.

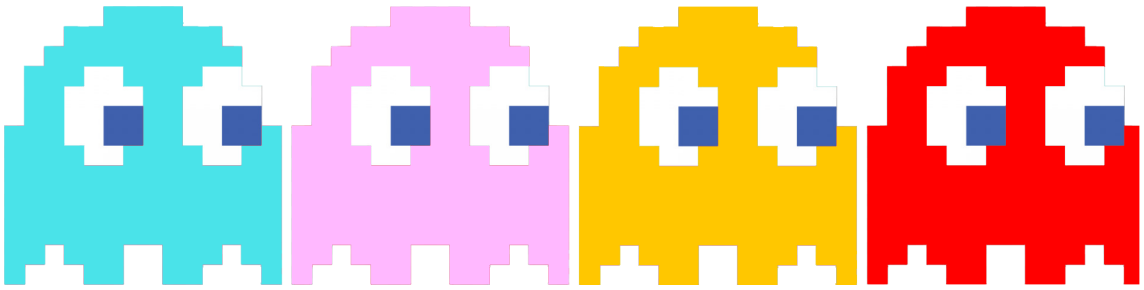


Figure 1.1: The ghosts/characters

The game includes the usage of textures, audio files, hierarchical models, lights and animations, using principally the *three.js* libraries.

1.2 HTML pages

The structure of project has been built in such a way as to have a logical subdivision of the individual components of the rendering, i.e. creating appropriate files for each individual object and functionality. More in particular, the HTML pages are:

- `index.html`, used to manage the homepage of the game
- `chooseColor.html`, used to manage the choose of the character
- `scoresPage.html`, used to manage the page of scores obtained during the all modalities of the game

- `ingame.html`, used to manage the rendering of the game. This is the most important one that was used as core page for all the game.

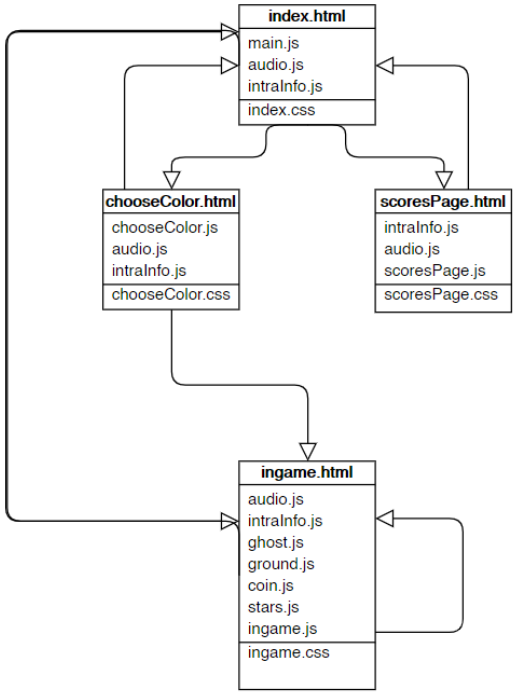


Figure 1.2: Flowchart of HTML pages relationship and JS/CSS scripts

1.3 Data management

As shown in figure 1.2, there is a defined data flow between the pages, in order words, there is the necessity of managing persistent information. This job is done by the file named *intraInfo.js*. This file makes use of the session storage of the browser in order to maintain all that information that is considered as persistent, as the scores, the current character chosen by the player, the player name itself, etc.

1.4 Libraries

All the figures and components are made with a javascript library called `Three.js` [3] [1], that is a cross-browser JavaScript library based on WebGL and application programming interface (API) used to create and display animated 3D computer graphics in a web browser.

Three.js allows the creation of graphical processing unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible due to the advent of WebGL.

Chapter 2

The main scene

2.1 Building the scene

In order to properly build the main scene, the one in which the player will actually play, we need to give first a little insights of what is rendered. The ghost is floating above one of four different lines. These lines are just four positions over a cylinder which is rotation on his axis. So, in this way, we have two different and independent models, the ghost and the ground. The other model present is the coin, which is spawned dynamically and with random position (over the fixed four lines) and random color. The camera use a perspective point of view, in order to focus only on the central four lines of the cylinder. In the scene more than one light is present: one which light up all the scene, making the color on the shape of the object clearer to the user, and one which is inside the ghost, creating an area on the ground illuminated of the color of the character, simulating the halo of the ghost. Both the ghost and the ground present another object above them, simulating the polygon mesh of the object. This is used to better perceive the representation of the lines on the ground, where the ghost can move.

2.2 The ghost model

The ghost model is composed in the following way: there are two parallel hierarchical models, one which is the *body* itself of the ghost, and the other is the *wire body*, the one that simulate the polygon mesh of the object. These models have the same positions and rotation. Each of them can be described as shown in the figure 2.1.

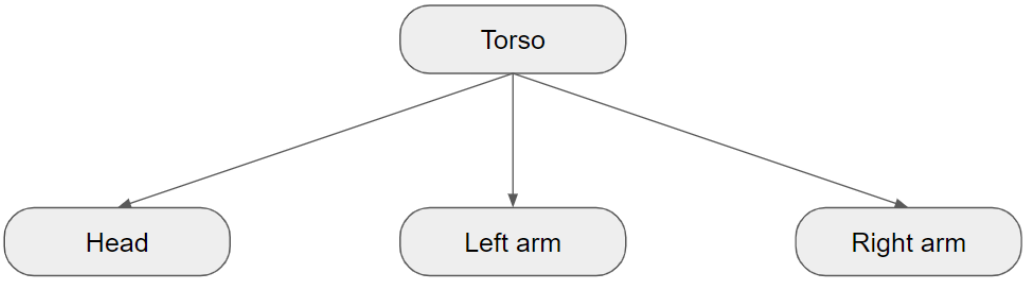


Figure 2.1: Hierarchical model of the ghost

One problem to overcome in this case, is the fact that the polygonal mesh's lines will *flash*, due to the overlap on the z-axis with the polygon of the other model. This problem is solved easily by *three.js*, setting the parameter *polygonOffset*, *polygonOffsetFactor* and *polygonOffsetUnits*, for the material representing the body, that part that should not be seen over the mesh's lines. The differences between the material used are clear in the following code fragment reported:

```
var matWireBody = new THREE.LineBasicMaterial
                  ({color: colorGhost,
                    linewidth: 10 });
var matBody = new THREE.MeshToonMaterial
                  ({color: colorBodyGhost,
                    polygonOffset: true,
                    polygonOffsetFactor: 1,
                    polygonOffsetUnits: 1});
```

This logic is applied to each part of the tree model, in order to avoid this problem. This logic makes possible to create a basic animation for the ghost, since each part can move dependently from the torso.
There are other models that are related somehow to the ghost. The *crazy* mode is going to change the color of the ghost. The player can know the future color by looking at a token that descend upon the ghost. The token is a model created using the *three.js* object called *PlaneGeometry*, to which is applied a texture representing one of the possible four ghost (the one shown in figure 1.1). This token needs to move along the x-axis with the ghost, but should not implement the same animation of the torso (up and down alternatively), so it can not be a child of the *torso* node. As previously said, the ghost has an halo. This is halo is simulated by a spotlight set inside his body and which is pointing towards the ground. The spotlight changes color with the ghost, when the change happens. Also, this light should make the same movement of the torso, otherwise it will appear like the halo is fixed whereas the body of the ghost is moving.

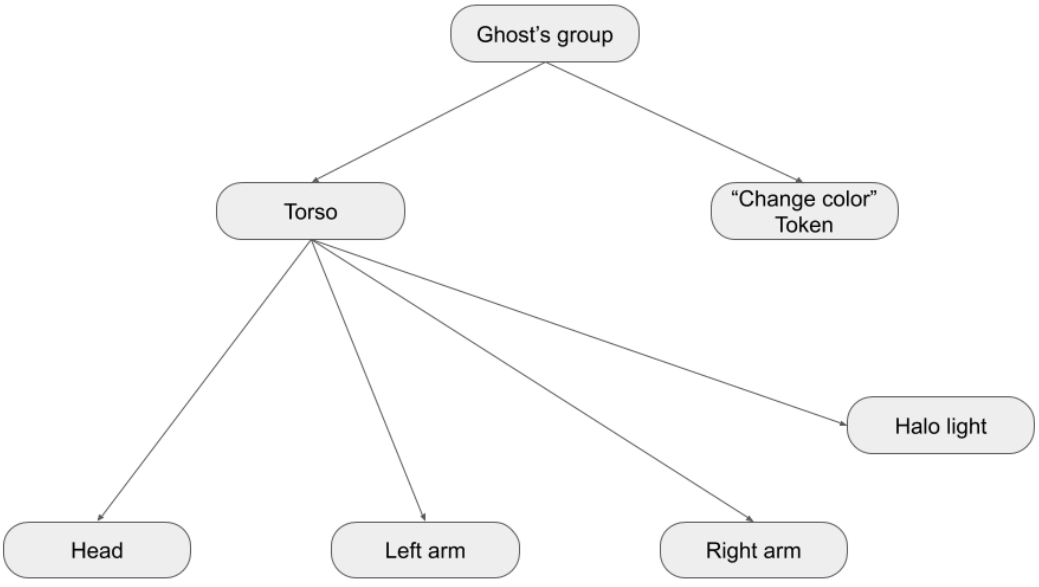


Figure 2.2: Final hierarchical model of the ghost

The initial hierarchical model presented in figure 2.1 can be now updated to a

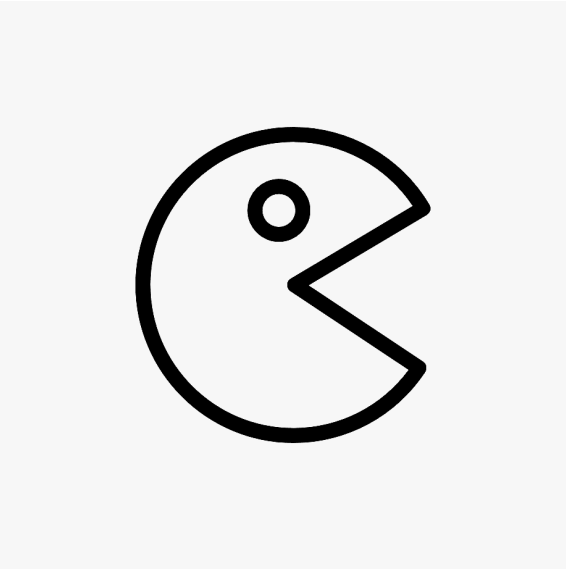


Figure 2.3: Texture applied on each coin

more complex one, with the usage of a *group*, as root. The *three.js* object *Group* is used specifically in order to implement this kind of model. The new hierarchical model concerning the ghost and all his aspects is as shown in figure 2.2

2.3 Ground and coins

2.3.1 Ground implementation

One of the most important object present into the scene is the ground that represent the rotating track on which the main character (ghost) moves and on which the entire game phase develops. Its main structure is composed by two objects that in practice are two cylinders. The first one is a `THREE.CylinderBufferGeometry` (basically a cylinder) and the second one is a `THREE.WireframeGeometry`, used to build a cover to the main cylinder and to outline the 4 tracks on which you can move the ghost. So the implementation of full ground is in the `ground.js` file:

```
var geometryGround = new THREE.CylinderBufferGeometry( radiusTop,
    radiusBottom, height, radialSegments, heightSegments);
var materialCylinderGround = new THREE.MeshToonMaterial({
    color: 0x000044,
    polygonOffset: true,
    polygonOffsetFactor: 1,
    polygonOffsetUnits: 1});
meshGround = new THREE.Mesh( geometryGround,materialCylinderGround
    );

// horizontal rotation of ground, coins
meshGround.rotation.z = Math.PI/2;
scene.add(meshGround);

var geometryGroundWireframe = new THREE.WireframeGeometry(new THREE
    .CylinderBufferGeometry(radiusTop, radiusBottom, height,
    radialSegments, heightSegments));
var materialGroundWireframe = new THREE.LineBasicMaterial( { color:
    0xBD00FF, linewidth: 0 } );
```

```
var meshNeonGround = new THREE.LineSegments(geometryGroundWireframe
    , materialGroundWireframe);
meshGround.add(meshNeonGround);
```

The texture applied on each coin is the one showed in figure 2.3.

2.3.2 Coin implementation

Another important object for the game is the coin, that represent the reward of the user to reach a good score. As for the ground, also the coin has a cylender geometry but with different characteristic, since it has a small `height` parameter and also a `texture` application. About the geometry it is a `THREE.CylinderBufferGeometry` with a `THREE.MeshToonMaterial` for the material. In particular, since the cylinder is composed geometrically by three side and we needed to apply a texture only on 2 of 3 sides we declared two material types in which in one there is a texture, composed by an image and in the other one only a color property. The full implementation is in the `coin.js` file:

```
var texture = new THREE.TextureLoader().load('../images/
    pacmanTexture.png')
texture.minFilter = THREE.LinearFilter;
var geometryCoin = new THREE.CylinderBufferGeometry( 0.12, 0.12,
    0.05, 30, 1);
var materialCoinBase = new THREE.MeshToonMaterial({color: colorCoin
    , side:THREE.DoubleSide, map: texture});
var materialCoinSide = new THREE.MeshToonMaterial({color: colorCoin
    });
materialCoinBase.minFilter = THREE.LinearFilter;
var meshCoin = new THREE.Mesh(geometryCoin, [materialCoinSide,
    materialCoinBase, materialCoinBase]);
```

2.3.3 Coin management

The coin are generated, added and removed to the scene dynamically. Each coin can be generated in just one of the four tracks and can have one color. Initially we though to generate these parameters randomly at run-time, although, in order to avoid to impact on the performances, we used a different method. In the initial page, where the computation done by the machine are at minimum, 200 random number are generated. The first one hundred are the first one-hundred position in which the coin will spawn, and the second hundred are the index of the color they are going to have. Since, after one hundred different coin, hardly the player can remember the sequence, these values are reused. In case of game restart, these sequence are updated using mathematical computations instead of generating totally new sequences. After the 6-th attempt from the player to restart the game, the sequence are generated randomly again, and the process in repeated.
Regarding the adding to the scene of the coin, each coin is added as child of the ground (cylinder) in order to follow the rotation. The coins are added in a position behind the cylinder, in such a way that the player cannot see them appear suddenly. When the coin pass over the ghost position and are, in fact, outside the camera view, the model is removed from the scene. In order to be able to do these step, each coin as an id, which is just an incremental number. In this way the hierarchical model of the ground as root has, approximately, five or six leafs. When a new one

enter the tree, the oldest is removed. In conclusion, it is been used a dynamically changing hierarchical model.

2.4 Camera and light

All the scene is seen by the user as a camera that is positioned fixed on the side of the cylinder, elevated, so as to frame the four main tracks on which the main character (the ghost) moves in order to take the coins. Going into details, the camera is implemented in the `ingame.js` file:

```
var camera = new THREE.PerspectiveCamera( 50, window.innerWidth /
    window.innerHeight, 0.1, 1000 );

camera.position.x = 0;
camera.position.y = 3.5;
camera.position.z = 6;
camera.lookAt(0,3,0);
```

About the lights, in addition to ghost spotlight previously introduced there is another one, used to radiate some light on the ground, character and scene that is a directional light type. As for the camera, it is implemented in the `ingame.js` file:

```
var light = new THREE.DirectionalLight(0xffffff,0.9)
light.position.y = 4
light.position.z = 10

var mainLightTarget = new THREE.Object3D()
mainLightTarget.position.y = -1

scene.add(mainLightTarget)
light.target = mainLightTarget
scene.add(light);
```

2.5 Background stars

One of the main goal was to make the theme of the game in the space. To do this initially we thinked about a static image representing the stars. However the effect was not really good and for this reason we think to use a dynamical approach. Since the Three.js has some problems with the use of .gif images we found an interesting way to generate stars into a scene buildt by TK from *redstapler.co* [2].

```
// this creates the geometry, with 6000 vertices (stars)
starGeo = new THREE.Geometry();
    for(let i=0;i<6000;i++) {
        star = new THREE.Vector3(
            Math.random() * 600 - 300,
            Math.random() * 600 - 300,
            Math.random() * 600 - 300
        );
        star.velocity = 0;
        star.acceleration = 0.02;
        starGeo.vertices.push(star);
    }
```

```
// this apply a texture that in practice is an image representing a
    white point, previously drawed with a photo software
let sprite = new THREE.TextureLoader().load( '../images/star.png' )
   ;
sprite.minFilter = THREE.LinearFilter;
let starMaterial = new THREE.PointsMaterial({
    color: 0xaaaaaa,
    size: 0.2,
    map: sprite
});

stars = new THREE.Points(starGeo,starMaterial);
scene.add(stars);

// this animate the stars
starGeo.vertices.forEach(p => {
    if(p.velocity >=150){
        p.velocity += p.acceleration
        p.x -= p.velocity;

        if (p.x < -200) {
            p.x = 200;
            p.velocity = 0;
        }
    }

});
starGeo.verticesNeedUpdate = true;
stars.rotation.x +=0.001;
```

Chapter 3

Animations

3.1 Ghost animation

The ghost follow two kind of animation. One that is done constantly and one that happen only when it moves between the tracks on the ground.

3.1.1 Persistent animation

In each instant the ghost is doing the following movements:

- Arms move forward and backward;
- Head is shaking left and right;
- The body is moving up and down, imitating the floating.

This is done maintaining a boolean flag and a threshold for the rotation of the arm. This threshold is currently equal to 0.5 radiant and once is reached, the boolean flag is set at his opposite value. On this basis, it is possible to understand easily if the arms must currently move forward on backward, if the head must rotate to the left or to the right and if the body must go up or down. Below the code representing the main step of the animation of the ghost.

```
if(armsDirection && meshArm1.rotation.x<0.5){
    meshArm1.rotation.x += 0.025
    meshArm2.rotation.x -= 0.025
    meshWireArm1.rotation.x += 0.025
    meshWireArm2.rotation.x -= 0.025
    meshHead.rotation.y += 0.020
    meshWireHead.rotation.y += 0.020
    meshWireBody.position.y += 0.0035
    meshBody.position.y += 0.0035

}
else if(armsDirection)
    armsDirection = false

if(!armsDirection && meshArm1.rotation.x>-0.5){
    meshArm1.rotation.x -= 0.025
    meshArm2.rotation.x += 0.025
    meshWireArm1.rotation.x -= 0.025
    meshWireArm2.rotation.x += 0.025
```

```
    meshHead.rotation.y -= 0.020
    meshWireHead.rotation.y -= 0.020
    meshWireBody.position.y -= 0.0035
    meshBody.position.y -= 0.0035
}
else if(!armsDirection)
    armsDirection = true
```

The code presented here is executed at render time. Please notice that since the hierarchical structure of the ghost the movement of the body is inherited by all the others parts, included the internal spotlight.
There is also another animation, that is the one performed by the token that point to the color the ghost will have. This token is constantly rotating on the y-axis.

3.1.2 Changing track animation

Initially, when the ghost changes track, it will simply appear suddenly on the next position, instead of performing an animation showing the movement. This was motivated by the fact that when the speed of the game is at his maximum, the position of the ghost (aka the *current track*) was not the same as the *exact* position (aka the *position displayed*), when the ghost is moving and this can create a problem if a coin is passing by. In order to perform this animation, it is necessary to use an high speed translation, for which this particular case will happen very rarely.
With these in mind, we decide that the best choice was to perform a very fast animation, completed in just ten steps. Since the translation of *changing the track* means just translating of a total of 1.2 on the x-axis, the animation is done in the following way: the animation start at the click of a directional arrow or A/D keys. This will invoke this function, passing the direction as parameter:

```
function moveGhostX(direction){
    if(direction == "left"){
        if(currentTrackGround>0 && !isMoving){
            isMoving = true
            movingRight = false
            currentTrackGround--
        }
    }else{
        if(currentTrackGround<3 && !isMoving){
            isMoving = true
            movingRight = true
            currentTrackGround++
        }
    }
}
```

When *isMoving* is set at *true*, the render function can perform the following steps:

```
if(isMoving)
{
    if(deltaMovementX <= 1.2){
        deltaMovementX += 0.12
        if(movingRight)
            translateGhostX(0.12)
        else
            translateGhostX(-0.12)
    }else{
```

```
        deltaMovementX = 0
        isMoving = false
    }
}
```

As shown, the single steps are just a tenth of the total movement, meaning that the animation is computed in ten steps, that is quite a low number. The results are still satisfactory, although, since the high speed the game arrives and the little distance that the movement require.

3.2 Coin animation

The coin animation is simpler with respect to the two performed by the ghost. The following cycle is invoked in the render function and work over the array *coins* which store all the current existing coins in the scene.

```
coins.forEach((obj) => {
    obj.rotation.y = meshParent.rotation.x
    obj.rotation.x += 0.021
});
```

More specifically, the line

```
obj.rotation.y = meshParent.rotation.x
```

is need exclusively to orient all the coin in the same direction (toward the camera) and leave the direction fixed, accordingly with the rotation of the ground. Whereas the line

```
obj.rotation.x += 0.021
```

is necessary to implement the spin of the coin over the x-axis. In this way the object is not immobile and all the scene seems more dynamic.

Bibliography

[1] Three.js. Three.js documentation. URL https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene.

[2] TK. Cool space warp background effect with three.js. URL https://redstapler.co/space-warp-background-effect-three-js/.

[3] Wikipedia. Three.js. URL https://en.wikipedia.org/wiki/Three.js.