Technical Manual

Peach's Castle

TEACHER: CARLOS ALDAIR ROMAN BALBUENA
317222049-MENDOZA ANAYA ALDAIR ISRAEL
GROUP: 06

1-INTRODUCTION

The project aims to recreate Peach's castle in a virtual environment using the OpenGL framework. The main purpose of the project is to bring together the integral learning elements of the CGEIHC subject (Computer Graphics and Interactive Elements in Humanities and Sciences) and to demonstrate the efficient use of practical and theoretical skills.

Project objectives:

1. Planning and development: Carry out the planning and development of the project in accordance with the established requirements.
    - Select a façade and a space that can be real or fictitious as a reference for the 3D recreation in OpenGL.
    - Present reference images of the selected spaces, showing the objects to be recreated virtually.
    - Objects should be as close as possible to their reference image, both in appearance and setting.
2. User manual: Create a user manual in PDF format that provides clear instructions on how to use the Peach's castle virtual environment. The manual should contain screenshots illustrating each element explained.
3. Technical manual: Prepare a technical manual that includes the schedule of activities carried out for the completion of the project, together with evidence of the collaborative tools used for the development of the software.
    - Show evidence of the use of collaborative software platforms, such as Jira, Trello, Git or Github, with progress specifications in repositories.

The specific objectives of the project to recreate Peach's castle in a virtual environment are as follows:

1. Capture the look and feel of Peach's castle: The main goal is to create accurate and detailed 3D models that faithfully represent the architecture, structural elements and visual details of Peach's castle. This involves recreating the shapes, sizes, colours and textures of each element of the castle in a realistic way.
2. Capture the atmosphere of the castle: Peach's castle has a distinctive and enchanting atmosphere that is reflected in its design, lighting and setting. The aim is to convey that sense of magic and enchantment in the recreated virtual environment. This may include appropriate lighting to highlight specific areas of the castle, the use of visual and particle effects to create a magical atmosphere, and the design of landscapes or surrounding elements to complement the experience.
3. Recreate the objects present in the castle: In addition to the castle itself, there are other iconic and recognisable elements within Peach's world, such as the throne, curtained windows, flags, staircases and other decorative objects. The goal is to virtually recreate these objects and details present in Peach's castle as faithfully as possible, paying attention to their proportions, colours and distinctive features.
4. Achieving interaction and exploration: The project can also include the possibility to interact with the virtual environment of Peach's castle. This involves allowing users to explore different areas of the castle, interact with objects, open doors, activate mechanisms and discover hidden surprises. The aim is to provide an immersive and entertaining experience for users.

Project proposal
Façade to be recreated: The aim is to recreate the façade of Peach's castle, based on the Super Mario 3D Land game. The castle will be represented as a square building with four towers at the corners and a larger tower at the top. It will be located on a rectangle that simulates a small hill. To create a realistic environment, approximately 12 trees will be added. around the castle.

Objects to be recreated:
In addition to the façade of the castle and its interior, several iconic objects and characters from the game will be recreated. These include:

- Toads: Toad characters, Peach's faithful followers, will be modelled and textured. These characters will be distributed around the castle environment, adding life and movement to the scenery.
- Goomba: The classic Super Mario enemies, the Goombas, will also be recreated. This little character will be present in the environment, walking in front of the castle.
- Question mark block: The iconic question mark block will be modelled and textured. This block is known to contain coins, power-ups and other prizes. It will be strategically placed within the environment so that users can interact with it.
- Throne: A throne made of wood, gold and velvet will be recreated and placed in a prominent place in the castle.
- Vase: A decorative vase will be modelled and textured and will be located in one of the interior areas of the castle.
- Fire Flower: The iconic Fire Flower, a special power-up in the Super Mario series, will be modelled and textured, only it was decided to take the Super Smash Bros Melee version. This flower will be available in the outdoor environment so that users can contemplate its unappreciated movement in the games.
- Kart: A kart, the vehicle used by Mario and his friends in Mario Kart races, will be modelled and textured. This object will be present in the environment, adding a fun and playful touch to a route outside the castle. As well as another stationary one located in the castle for more detailed contemplation.

This proposal seeks to faithfully recreate the elements and atmosphere of Peach's castle, providing an immersive and recognisable experience for Super Mario fans.

2-PLATFORMS AND TOOLS USED

For the development of the castle environment, several working platforms were used to implement the project. The main platforms used are described below:

- Microsoft Visual Studio:

The Microsoft Visual Studio Integrated Development Environment (IDE) was used as the main tool for developing software and applications. This IDE offers a wide range of functionalities and tools that facilitate programming and the manipulation of graphics and images.

- OpenGL:

Use was made of OpenGL, which is an API (application programming interface) used to manipulate graphics and images. Although OpenGL is a specification developed and maintained by the Khronos Group, it is necessary to implement these specifications for the functions to operate correctly. In the project, we worked with the OpenGL libraries written in C, which allow their use in other programming languages.

- Maya:

Maya was used as the main modelling software in the project. It is a widely used 3D modelling application that allows to create and edit three-dimensional models with great detail and realism. With Maya, it was possible to design and shape the elements of the castle and the objects/characters that belong to this universe.

- IbisPaintX:

IbisPaintX was used as texture editing software in the project. It is a digital painting application that allows you to create and modify textures with a wide range of tools and effects. With IbisPaintX, the models of Mario's world were brought to life by applying detailed textures.

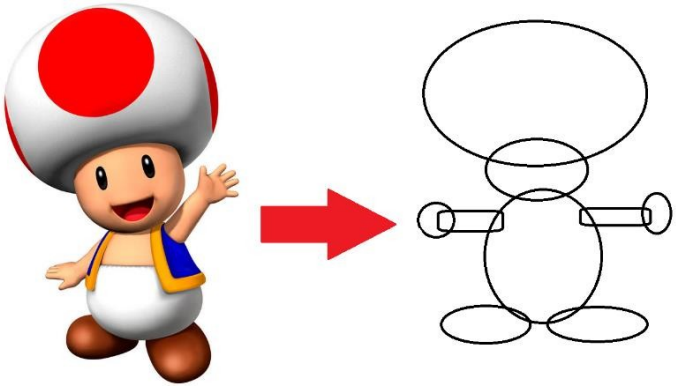
- Models Resource:

The Models Resource website was an important source for models and textures used in the project. Models Resource is an online platform that offers a large collection of resources, such as 3D models and textures, that can be downloaded and used in design and development projects. Thanks to Models Resource, a variety of high quality models and textures could be accessed to enrich the castle environment.

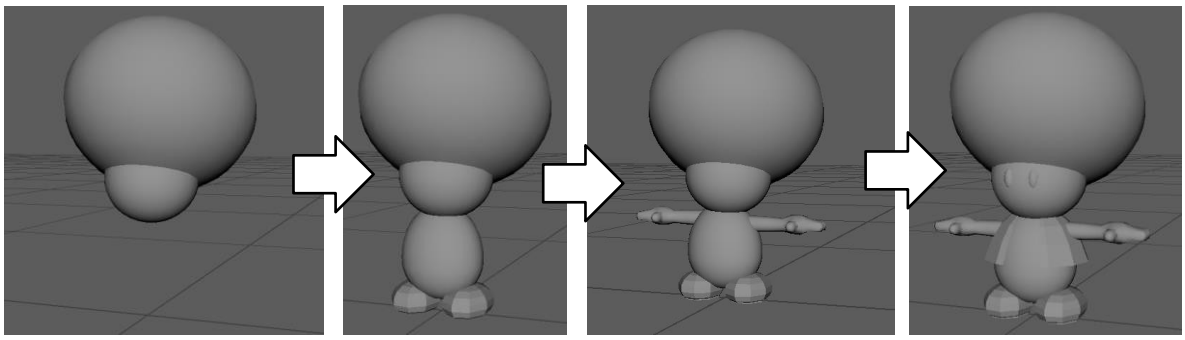
By employing these platforms and programmes in the Peach Castle project, an effective and detailed implementation was achieved, allowing the elements of Mario's world to be accurately modelled, textured and enriched with resources available in Models Resource.

3-3D Modelling Process

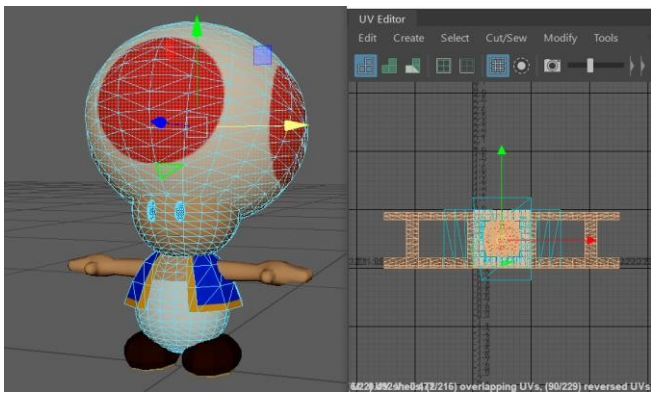
Modelling in Maya is a creative and technical process that brings ideas and concepts to life in the form of 3D models. It begins with the planning and conceptualisation of the objects to be created. This involves defining the basic shape, proportions and specific details of each component that will make up the object.



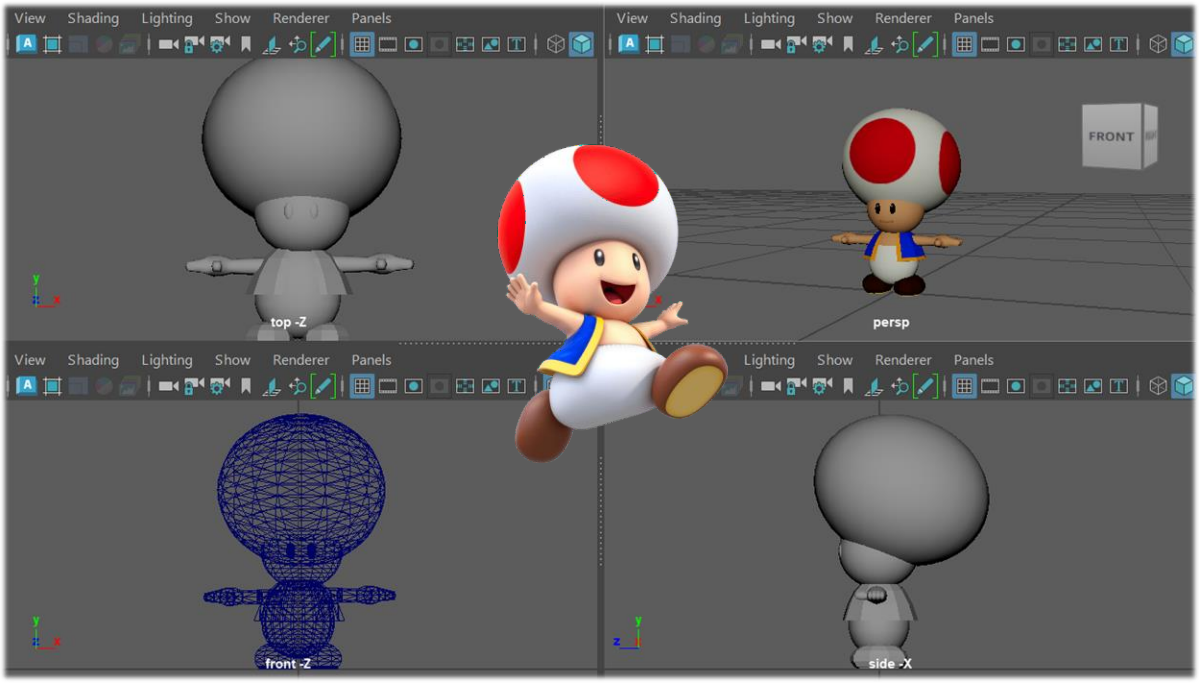
Once the basis of the design was established, the geometry was created using the modelling tools available in Maya. During the modelling process, surface subdivision techniques, extrusion, smoothing and transformation using faces, edges and vertices were applied for the details of the object.



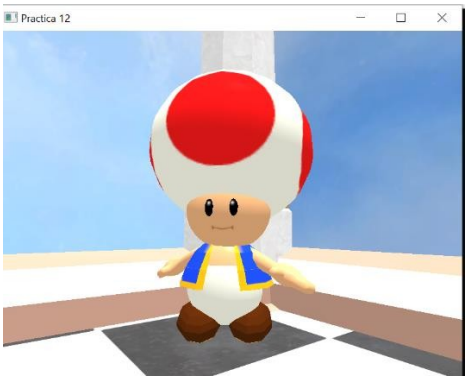
We then look to choose the right image to texture the object and give the object fidelity to the selected design, so we work with texture mapping so that each face is being painted in the best possible way.



Once finished with the object, it is analysed from all possible perspectives and with the three available views, to determine if it can be optimised or with the object you have is more closely resembles the desired object/character. There were cases where I decided to remove faces of the object that were not visible from the outside, making it easier to load.

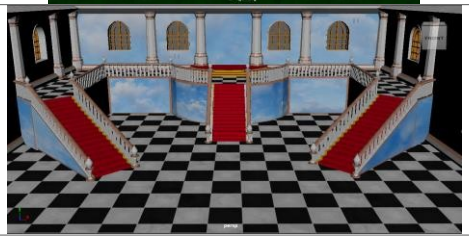


Finally we integrate the model into our scenario within Visual Studio, checking that the generated and/or downloaded model matches the environment we are generating.



4-GALLERY AND DETAILS OF MODELS IN MAYA

Peach Castle

Image reference	Model recreated in Maya	Number of triangles
		**42, 655 triangles**
		

Toad

Image reference	Model recreated in Maya	Number of triangles
		**3, 400 triangles**

Goomba

Image reference	Model recreated in Maya	Number of triangles
		**1, 669 triangles**

Question Block

Image reference	Model recreated in Maya	Number of triangles
		**44 Triangles**

Throne

Image reference	Model recreated in Maya	Number of triangles
		**4,518 Triangles**

Vase

Image reference	Model recreated in Maya	Number of triangles
		**192 Triangles**

Flower

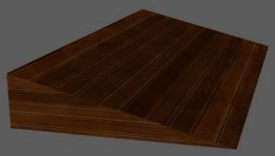
Image reference	Model recreated in Maya	Number of triangles
		**192 Triangles**

Kart

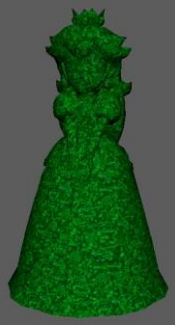
Image reference	Model recreated in Maya	Number of triangles
		**4, 036 Triangles**

4.1 ADDITIONAL MODELS USED

4.1.1 Modelling

Model	Model in Maya	Number of triangles per model
Shrub with flowers		1,848 triangles
Painting		98 triangles
Framework		32 triangles
Star		796 triangles
Question Empty block		44 triangles
Ramp		8 triangles
Windows		372 triangles (186 each half)
Glass translucent		16 triangles

4.1.2 Downloaded

Model	Model in Maya	Number of triangles per model	Re texturing
Tree		1,340 triangles	YES
Shrub with form of Peach		3,419 triangles	YES
Mushroom		466 triangles	NO
Fly Guy		2,316 triangles	NO
Parachutes		410 triangles	NO
Peach		3,755 triangles (1,888 without the kart)	Only the kart changed colour
Windows		3,828 triangles	NO
Paratroopa		3,416 triangles	NO
Doors		3,888 triangles	Y E S

The virtual environment of Peach's castle is using an amount of 116,839 triangles in its composition.

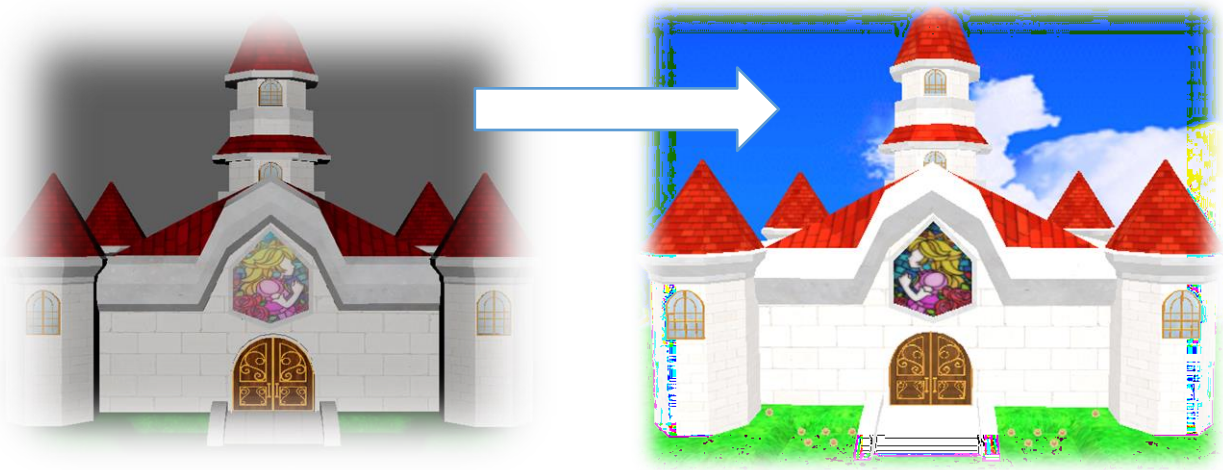
5-LIGHTING

By configuring the overall lighting, the aim was to generate lighting that would bring out the colours and details of the castle, creating a cheerful and festive atmosphere. Different light intensities and tones were experimented with to achieve the desired effect, ensuring that the lighting highlighted the architectural details and brought out the vivid colours present in Peach's castle.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, -0.5f, -0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.9f, 0.9f, 0.9f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
```

By adjusting the overall lighting parameters, it was possible to create lighting that gives a sense of warmth and joy, bringing an attractive and vibrant visual appearance to the castle's virtual environment.

It is important to note that the choice of colourful and cheerful lighting is primarily aimed at enriching the user experience and immersing the user in a magical and enchanting atmosphere, in keeping with the style and theme of Peach's castle.



6-ANIMATIONS

During the OpenGL animation process, we will take advantage of the following techniques and tools:

- Geometric transformations: We will use transformation matrices to achieve translation, rotation and scaling movements of the objects in space. These transformations will be applied to the vertices of the models to achieve changes in their position and shape.
- Timing control: To control the speed and rhythm of the animation, we will implement mechanisms to increase or decrease values that will affect the elapsed time between frames. This will allow us to create real-time animations and adjust the duration and speed of the movements according to our needs.
- Mathematical functions: By incorporating mathematical functions in OpenGL animation, we can expand our creative possibilities and achieve more interesting and dynamic results.

Screenshots of the animations can be found at the end of the document in the section **"Results and visual examples"**.

6.1 Simple animations

- **Doors and windows:**

The animation of the doors and windows of Peach's castle adds an element of interactivity when the P or V key is pressed. The doors move in a rotating motion on the Y-axis, starting from a closed position of 0 degrees to an open position of 90 degrees towards the inside of the castle. On the other hand, the windows also move in a rotational movement on the Y-axis, but with a wider range, from 0 degrees to 120 degrees and outwards. Both animations are triggered by a boolean that determines when to rotate, and the animation cannot be interrupted until it has finished rotating inwards or outwards.

```
//Ventanas//////////////////////////////
if (keys[GLFW_KEY_V])
{
    AnimVentana = true;
}
if (AnimVentana)
{
    if (Ventana)
    {
        if (RotVentana < 120)
        {
            RotVentana += 2;
        }
        else
        {
            Ventana = false;
            AnimVentana = false;
        }
    }
    if (!Ventana)
    {
        if (RotVentana > 0)
        {
            RotVentana -= 2;
        }
        else
        {
            Ventana = true;
            AnimVentana = false;
        }
    }
}
```

```
//Puerta/////////////////////////
if (keys[GLFW_KEY_P])
{
    AnimPuerta = true;
}
if (AnimPuerta)
{
    if (Puerta)
    {
        if (RotPuerta < 90)
        {
            RotPuerta += 0.2;
        }
        else
        {
            Puerta = false;
            AnimPuerta = false;
        }
    }
    if (!Puerta)
    {
        if (RotPuerta > 0)
        {
            RotPuerta -= 0.2;
        }
        else
        {
            Puerta = true;
            AnimPuerta = false;
        }
    }
}
```

```
model = glm::translate(model, glm::vec3(-0.428f, 12.109f, 1.559f));
model = glm::rotate(model, glm::radians(-RotVentana), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana1.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.446f, 12.109f, 1.559f));
model = glm::rotate(model, glm::radians(RotVentana), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana2.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(1.362f, 1.417f, 5.059f));
model = glm::rotate(model, glm::radians(-RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta2.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.495f, 1.417f, 5.059f));
model = glm::rotate(model, glm::radians(RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta1.Draw(lightingShader);
```

To achieve a sense of realism and weight in the door animation, the opening speed of the windows has been adjusted to be faster than the door. This creates an interesting visual contrast and highlights the difference in size and material between the doors and windows. The intention is to convey a sense of solidity and robustness in the door, while the windows provide a lighter and more agile opening.

- **Toads jumping:**

The animation of the jumping Toads adds an element of activity to Peach's castle environment. In this animation, two Toads will be jumping while slightly raising and lowering their arms on the Z-axis. To achieve this effect, two variables are used: one variable "IDLE" and another variable "

RotBrazo". These variables increase and decrease at different rates, which creates a harmonic and synchronised movement of the Toads.

The "IDLE" variable controls the jumping motion, making the Toads move up and down on the Y-axis. By increasing and decreasing this variable, the effect of continuous and rhythmic jumping is achieved. On the other hand, the "RotBrazo" variable controls the movement of the Toads' arms on the Z-axis. As this variable changes, the arms of the Toads move up and down in a smooth and coordinated manner. The use of different speeds for the "IDLE" and "RotBrazo" variables adds dynamism and naturalness to the animation, avoiding repetitive and monotonous movements.

```
//Toad//////////////////////
if (sentidoIDLE)
{
    IDLE += 0.0006;
    if (IDLE > 0.025)
    {
        sentidoIDLE = false;
    }
}
if (!sentidoIDLE)
{
    IDLE -= 0.0006;
    if (IDLE < 0)
    {
        sentidoIDLE = true;
    }
}
if (sentidoBrazo)
{
    RotBrazo += 0.4;
    if (RotBrazo > 20)
    {
        sentidoBrazo = false;
    }
}
if (!sentidoBrazo)
{
    RotBrazo -= 0.4;
    if (RotBrazo < 0)
    {
        sentidoBrazo = true;
    }
}
```

```
//Toad
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(-1.247f, 0.609f + (IDLE * 1.7), -0.619f));
model = glm::rotate(model, glm::radians(38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(lightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(RotBrazo), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(lightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-RotBrazo), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.247f, 0.576f + IDLE, -0.619f));
model = glm::rotate(model, glm::radians(38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(IDLE * 800), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(lightingShader);
```

In order to avoid doing combined transformations manually, we chose to make the Toad with hierarchy, so that the torso receives the jump and transmits it to the arms, while the arms do the Y-transfer and Z-rotation.

- **Goomba walking**

The Goomba walking animation adds a touch of movement and personality to Peach's castle environment. In this animation, the Goomba moves from side to side on the X-axis while rotating his head and lifting his feet slightly on the Z-axis, following his characteristic walk. To achieve this effect, increments and decrements in specific variables are used. These variables control the horizontal displacement of the Goomba on the X-axis, the rotation of his head and the movement of his feet on the Z-axis which are controlled with booleans. An additional detail is added to ensure that the character's feet appear to step firmly on the floor and not through it. To achieve this, the vertical position of the Goomba's feet is checked during its movement on the Z-axis, when the rotation of the foot causes its position on the Z-axis to exceed the level of the floor, its value is set to 0. This ensures that the Goomba's foot is in contact with the floor and does not cross the surface.

```
//Goomba
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(rotCabezaGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaCabeza.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(2 * rotPieDerGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaPieDer.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(2.992f + CaminataGoomba, -0.041f, 12.026f));
model = glm::rotate(model, glm::radians(2 * rotPieIzqGoomba), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
GoombaPieIzq.Draw(lightingShader);
```

```
if (SentidoGoomba == false)
{
    CaminataGoomba += 0.002f;
    if (CaminataGoomba > 0.647f)
    {
        SentidoGoomba = true;
    }
}

if (SentidoGoomba == true)
{
    CaminataGoomba -= 0.002f;
    if (CaminataGoomba < -0.936f)
    {
        SentidoGoomba = false;
    }
}
```

```
if (RotSentidoGoomba == false)
{
    rotCabezaGoomba += 0.3f;

    if (rotCabezaGoomba < 0)
    {
        rotPieDerGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba > 0)
    {
        rotPieIzqGoomba = rotCabezaGoomba;
    }

    if (rotCabezaGoomba > 10.0f)
    {
        RotSentidoGoomba = true;
    }
}
```

```
if (RotSentidoGoomba == true)
{
    rotCabezaGoomba -= 0.3f;
    if (rotCabezaGoomba < 0)
    {
        rotPieDerGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba > 0)
    {
        rotPieIzqGoomba = rotCabezaGoomba;
    }
    if (rotCabezaGoomba < -10.0f)
    {
        RotSentidoGoomba = false;
    }
}
```

- **Dancing flowers:**

Mario's fire flower animation adds a fun and dynamic touch to Peach's castle environment. These flowers perform a side-to-side rotational movement on the Y-axis, while simultaneously increasing their position on the Y-axis when their rotation is 0 degrees, creating a jumping effect.

```
//Flor//////////////////////////////////////////
if (FlorSentido)
{
    FlorGiro += 4;
    if (FlorGiro > 99)
    {
        FlorSentido = !FlorSentido;
    }
}
if (!FlorSentido)
{
    FlorGiro -= 4;
    if (FlorGiro < -99)
    {
        FlorSentido = !FlorSentido;
    }
}
```

```
if (FlorGiro > 0)
{
    FlorBrinco = 0.025 * (100- FlorGiro) / 100;
}
if (FlorGiro < 0)
{
    FlorBrinco = 0.025 * (100 + FlorGiro) / 100;
}
```

```
//Flores
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5.350f, 0.480f + FlorBrinco, 6.050f));
model = glm::rotate(model, glm::radians(-3 * FlorGiro / 20), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Flor.Draw(lightingShader);
```

- **Fly Guy flying back and forth**

For the FlyGuy animation, a lateral movement is created on the X-axis as the character tilts to the side to which it is moving. The lateral movement is achieved by modifying the FlyGuy's X-axis position, moving it from one side to the other, the direction being controlled with a boolean. At the same time, a tilt is applied to the corresponding side using a rotation on the Y-axis. This combination of lateral movement and tilt creates the sensation that the FlyGuy moves dynamically and fluidly, while keeping the view to one side of the castle.

The propeller in the FlyGuy's head is animated by a recycled variable that controls its rotation. This variable is gradually increased from 0 to 360 degrees, allowing the propeller to perform a full rotation. Once it reaches 360 degrees, the process restarts, creating a continuous and constant rotation effect. Additionally, a slight up and down flight motion can be added using an auxiliary. This auxiliary variable is added or subtracted from the FlyGuy's Y-axis position, generating a smooth up and down motion that simulates its flight.

```
//Flyguy
if (SentidoFlyGuy)
{
    TrasladoFlyguy += 0.01;
    if (RoatFlyGuy < 20.0)
    {
        RoatFlyGuy += 0.5;
    }
    if (TrasladoFlyguy > 2)
    {
        SentidoFlyGuy = false;
    }
}
if (!SentidoFlyGuy)
{
    TrasladoFlyguy -= 0.01;
    if (RoatFlyGuy > -20.0)
    {
        RoatFlyGuy -= 0.5;
    }
    if (TrasladoFlyguy < -2)
    {
        SentidoFlyGuy = true;
    }
}
```

```
//FlyGuy
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(8.162f , 2.224f + (auxilar1 / 2000), 0.0f +TrasladoFlyguy));
model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(0.0f, 0.0f,1.0f));
model = glm::rotate(model, glm::radians(RoatFlyGuy), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
FlyGuy.Draw(lightingShader);
modelaux = model;
model = modelaux;
model = glm::rotate(model, glm::radians(Rotacion2*10), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Helices.Draw(lightingShader);
```

- **Rotating star**

For the animation of the star, the same auxiliary variable mentioned above is used to achieve a slight up and down movement. This variable is added to or subtracted from the Y-axis position of the star, creating a smooth and constant floating effect. In addition to the vertical movement, a slow rotation about the Y-axis is applied via a rotation variable. This variable is gradually updated in each animation frame, allowing the star to rotate continuously and steadily on its own axis. The combination of the upward and downward movement together with the slow rotation on the Y-axis creates a fluid and enchanting animation for the star.

```
//Estrella
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 8.884f + (auxilar1 / 2000), 0.0f));
model = glm::rotate(model, glm::radians(Rotacion2), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Star.Draw(lightingShader);
```

- **Toad waving**

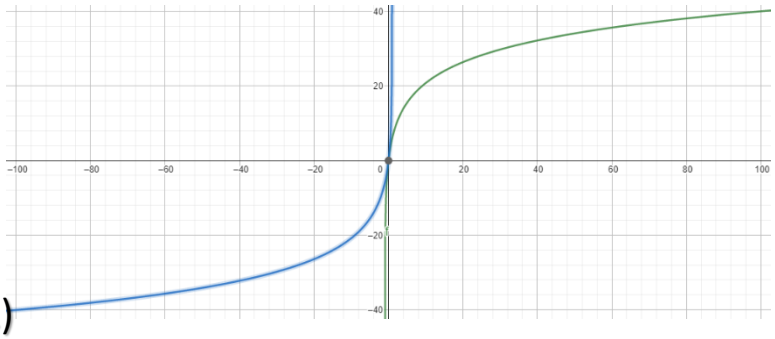
For the animation of Toad waving, a rotation on the X-axis is used which can be increased and decreased. Both Toad's body and his arm perform this rotation to simulate the waving gesture. The arm rotation variable used with the previous Toads is reused to keep pace with all the Toads in the throne room.

```
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(0.0f, 11.663f, 1.151f));
model = glm::rotate(model, glm::radians(RotBrazo - 10), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(lightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.059f, 0.165f, 0.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(4 * RotBrazo - 20.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(lightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.645f, 1.151f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(lightingShader);
```

6.2 Complex Animations

- **Toad waving his arms**

The animation of Toad performing the alternating arm-crossing motion is characterised by a gesture in which he rotates his torso while extending one arm forward and the other arm moves backward. This movement is repeated intermittently, creating a dynamic and energetic visual effect. To achieve this animation, a technique based on an auxiliary variable ranging from -100 to 100 is used. This variable is key to determining the position and rotation of Toad's arms and torso. The logarithm function is used to control the movement of the arms, so that when the variable is positive, a positive logarithm function is applied, and when the variable is negative, a negative logarithm function is used. This allows the arms to move fluidly and in harmony with the rest of Toad's body.



During the animation, the arm in the forward position is held for a brief moment extended forward, before making the change of position. This effect is achieved by taking advantage of the properties of the logarithm, which creates a pause in the movement at certain specific points.

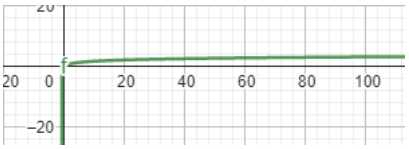
An interesting detail of this animation is that, when the maximum point of rotation is reached, regardless of whether it is to the right or to the left, there is a slight increase in the overall movement of the arms and torso. This accentuates the gesture performed by Toad, adding a touch of expressiveness to this movement.

```
if (Sentido1)
{
    auxilar1 += 2;
    if (auxilar1 > 99)
    {
        Sentido1 = false;
    }
}
if (!Sentido1)
{
    auxilar1 -= 2;
    if (auxilar1 < -99)
    {
        Sentido1 = true;
    }
}
if (auxilar1 > 0)
{
    Giro = 20 * log10(auxilar1 + 1);
    Brinco = 0.025 * (Giro / 40);
}
if (auxilar1 < 0)
{
    Giro = -20 * log10(-auxilar1 + 1);
    Brinco = 0.025 * (-Giro / 40);
}
```

```
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(5.716f, 2.029f + (Brinco * 1.7), -5.042f));
model = glm::rotate(model, glm::radians(-38.0f + (Giro / 4)), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CuerpoToad.Draw(lightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(Giro), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoDer.Draw(lightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.044f, 0.175f, 0.0f));
model = glm::rotate(model, glm::radians(-25.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(Giro), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
BrazoIzq.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5.716f, 2.029f + Brinco, -5.042f));
model = glm::rotate(model, glm::radians(-38.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(Brinco * 800), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Zapatos.Draw(lightingShader);
```

- **Paratroopa flying in and out of the frame**

The Paratroopa starts quickly out of the frame and then slows down to maintain a steady, but slightly forward progression. Before completing this cycle, it makes a complete 180-degree turn and moves steadily but slowly towards the return frame. This effect is achieved by using a logarithmic function for the Paratroopa's path. As the variable X increases, it is checked to see if it has reached a certain value to increase the rotation value and thus achieve the 180 degree turn back into the box. Once X reaches its maximum value, the Paratrooper gradually decreases its value of X, allowing the sequence to be repeated over and over again. In addition, the same auxiliary variable is reused in the same way as the star and the Fly Guy so that it rises and falls slightly to achieve a nice flying effect as it performs the described path.



$$f(x)=1.5\log_{10}(x+1)$$

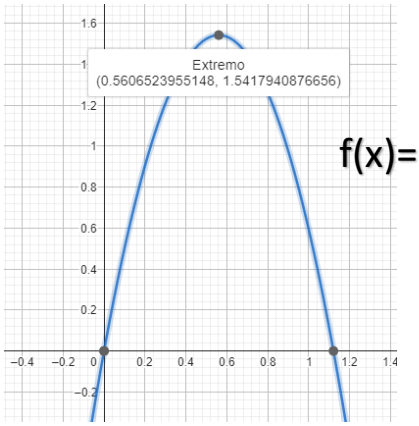
```
//Paratroopa/////////////////////////////////////////
if (Sentido2)
{
    VueloParatroopa = 1.5 * log10(auxilar2 + 1);
    auxilar2 += 0.09;
    if (auxilar2 > 79.91)
    {
        RoatParatroopa += 0.81;
    }
    if (auxilar2 > 99.91)
    {
        Sentido2 = false;
    }
}
if (!Sentido2)
{
    if (VueloParatroopa > 0.3)
    {
        VueloParatroopa -= 0.01;
    }
    if (VueloParatroopa < 0.3 && VueloParatroopa>0)
    {
        VueloParatroopa -= 0.0005;
    }
    if (VueloParatroopa < 0.3 && VueloParatroopa < 0)
    {
        RoatParatroopa = 0;
        auxilar2 = 0;
        Sentido2 = true;
    }
}
```

```
//Paratroopa
model = glm::mat4(1);
modelaux = model;
model = glm::translate(model, glm::vec3(3.422f, 1.257 + (auxilar1 / 2000), -4.794f + VueloParatroopa));
model = glm::rotate(model, glm::radians(RoatParatroopa), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.9f, 0.9f, 0.9f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Paratroopa.Draw(lightingShader);
modelaux = model;
model = modelaux;
model = glm::translate(model, glm::vec3(-0.061f, 0.085f, -0.042f));
model = glm::rotate(model, glm::radians(RotBrazo * 4), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
AlaDerecha.Draw(lightingShader);
model = modelaux;
model = glm::translate(model, glm::vec3(0.061f, 0.085f, -0.042f));
model = glm::rotate(model, glm::radians(-RotBrazo * 4), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
AlaIzquierda.Draw(lightingShader);
```

- **Mushroom inside the block**

The mushroom is shot from a question block and is launched upwards following the rules of physics. As it rises, it loses speed until it reaches its highest point and then begins to fall. During this process, the cube will scale, becoming smaller, while another smaller cube will become larger to represent the empty question block. When the mushroom returns to its original position, the cubes return to their original positions as well.

This animation is achieved by using the formula Y = 5.5 * X - 0.5 * 9.81 * X^2, which represents the vertical movement of the mushroom under the influence of gravity. While the Power-Up is in flight, it will make a Z-rotation to give the rotation more dynamism. This animation is activated by pressing the C key once and cannot be interrupted. It can only be restarted when the whole sequence is finished.



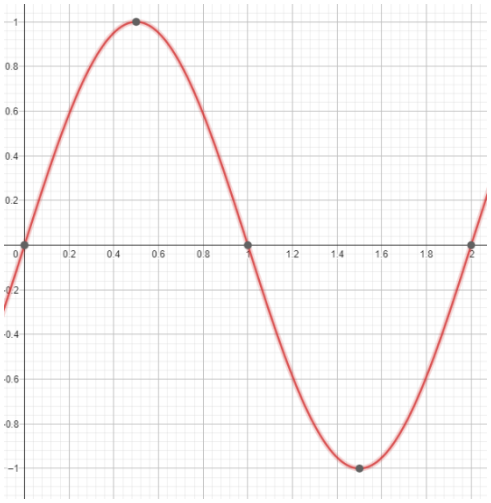

```
//Cubo-Champiñon//////////////////////////////////////////////////////////
if (keys[GLFW_KEY_C])
{
    AnimCubo = true;
}
if (AnimCubo)
{
    if (EstadoCubo)
    {
        if (Golpe < 0.3)
        {
            Golpe += 0.04;
        }
        else
        {
            EscalaVacia = 1.00;
            EscalaCubo = 0.5;
            EstadoCubo = !EstadoCubo;
        }

    }
    if (!EstadoCubo)
    {
        auxilar3 += 0.006;
        RoatChampi += 2;
        TiroVertical = 5.5 * auxilar3 - 0.5 * 9.81 * auxilar3 * auxilar3;
        if (Golpe > 0.05)
        {
            Golpe -= 0.05;
        }
        if (auxilar3 > 1.12)
        {
            EscalaCubo = 1.00f;
            EscalaVacia = 0.5f;
            auxilar3 = 0.0f;
            RoatChampi = 0.0f;
            TiroVertical = 0.0f;
            EstadoCubo = !EstadoCubo;
            AnimCubo = false;
        }
    }
}
```

```
//Cubo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + Golpe, 4.719f));
model = glm::scale(model, glm::vec3(EscalaCubo, EscalaCubo, EscalaCubo));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cubo.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + Golpe, 4.719f));
model = glm::scale(model, glm::vec3(EscalaVacia, EscalaVacia, EscalaVacia));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Vacio.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.425f, 1.404f + TiroVertical + Golpe, 4.719f));
model = glm::rotate(model, glm::radians(RoatChampi), glm::vec3(0.0f, 0.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Champi.Draw(lightingShader);
```

- **Mario Tanooki flying**

Mario rises and falls as he moves his tail up and down, which is achieved by a function Sen(x) that determines the rise and fall of his height. This function creates a smooth and fluid effect on Mario's vertical movement. To avoid filling up memory the variable X is reset to 0 so that the cycle continues without interrupting the animation.

In addition to the tail movement, an auxiliary variable is added to make Mario turn slightly from side to side, adding a more natural effect to the character's flight. This subtle sideways movement complements Mario's ascent and descent, creating a more dynamic and realistic sense of flight. Tanooki's suit gives Mario the ability to fly, and this animation highlights that characteristic.



f(x)= sin(x*3.1416)

```
//Mario Tanooki

Tanooki += 0.01;
if (Tanooki > 2)
{
        Tanooki = 0.0f;
}
VueloAltura = 1 * sin(Tanooki* 3.1416);
```

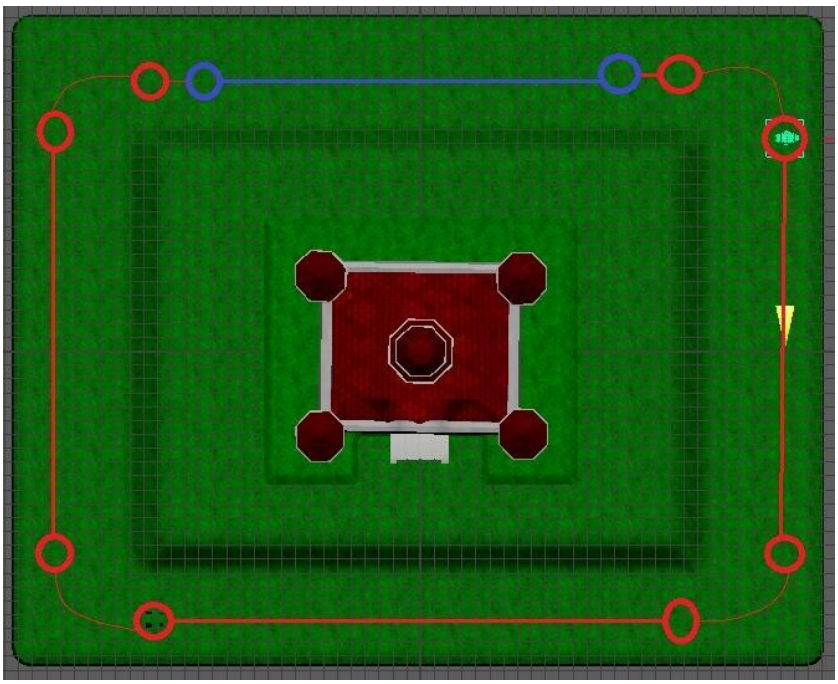
```
//Mario Tanooki
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.201f+ VueloAltura/2, -3.490f));
model = glm::rotate(model, glm::radians(180.0f+10 * (auxilar1 / 100)), glm::vec3(0.0f, 1.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mario.Draw(lightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 11.201f + VueloAltura/2, -3.490f));
model = glm::rotate(model, glm::radians(180.0f + 10 * (auxilar1 / 100)), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(20.0f*(Viento /3)), glm::vec3(1.0f, 0.0f, 0.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Colita.Draw(lightingShader);
```

- **Princess Peach driving a go-kart**

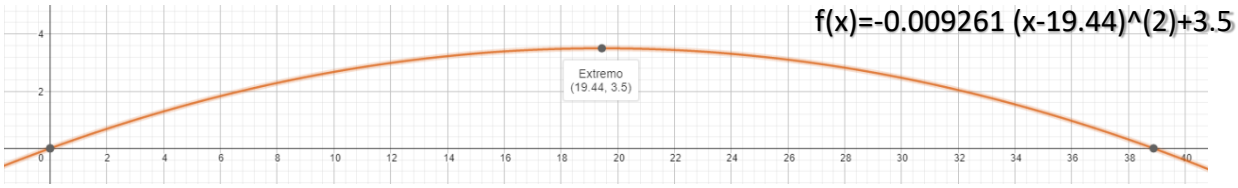
The animation of Peach in the go-kart was the most complex of this project. She goes all around the castle in a well-established circuit. As she progresses, she encounters a ramp at the back of the castle, which propels her into the air.

When Peach rises into the air, she deploys a parachute to maintain a steady flight, similar to what is seen in Mario Kart 8. The parachute allows her to maintain a smooth and stable trajectory in the air as she enjoys the flight. Once he returns to the ground, he stows his parachute and continues his journey, repeating the circuit over and over again.

This complex animation is achieved through a series of predefined routes. A total of eight routes are marked. It starts with route 1, which consists of a straight line. Once this path is completed, path 2 is activated, which involves a turn while decreasing the increment in distance X and increasing the increment in Z while increasing a rotation value to make the car more realistic in terms of its path. Then, route 3 is activated, and so on, until the square is completed and you start again.



However, on track 5, the animation becomes even more interesting. Peach not only moves forward in a straight line, but also upwards on the Y-axis. A parabola formula is used as a function of the X variable to create the effect of upward flight. During this part of the ride, Peach performs an acrobatic stunt by turning 380 degrees, adding a touch of excitement and style to his flight.



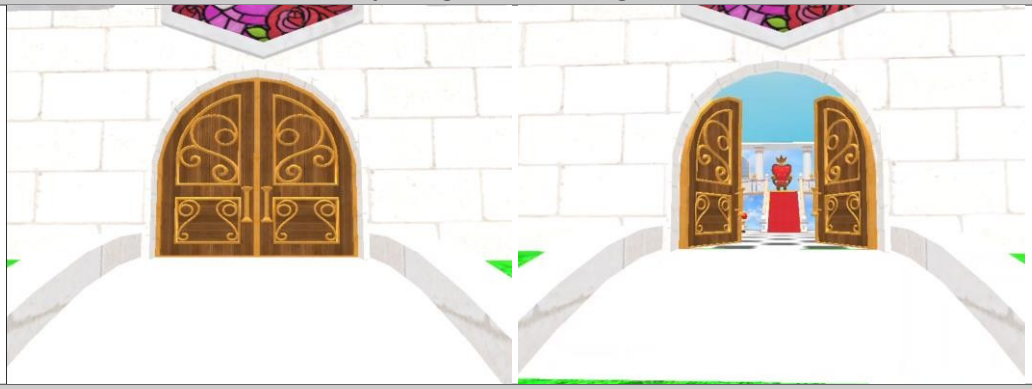
When Peach completes the circuit all values are reset to the value they were when the program was initialised to ensure that it repeats the loop in exactly the same way.

During flight, the parachute is gradually scaled so that it is visible. In addition, the parachute rotates rapidly and slightly from side to side to simulate the wind. Once the Y value returns to 0, the parachute is stowed and the kart resumes the initial rotations of track 5 to continue the circuit.

7-Results with visual examples

In this section, we will present the results obtained and visual examples of the animations implemented in our programme using OpenGL. Here you will be able to appreciate the final result of this project and visualise the animations described above in a concrete way.

Screenshots
Complete Virtual Space

Opening of the castle gates

Prancing Toads


Panting

ParaTroopa coming out of the painting

Kart Stationary

Trhone, flowers y vases

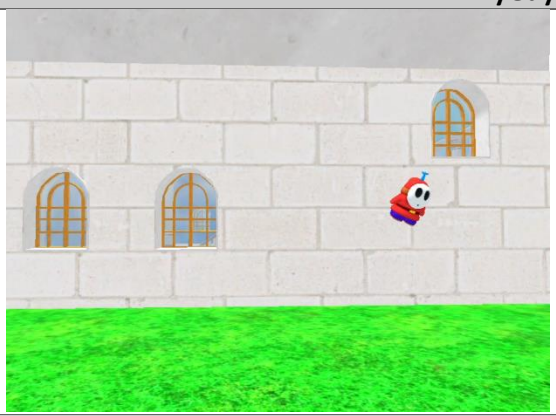
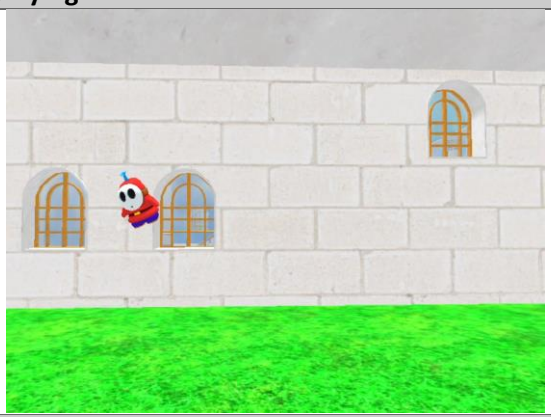
Translucent Window

Toad Waving his hands


Goomba walking


Dacing Flowers

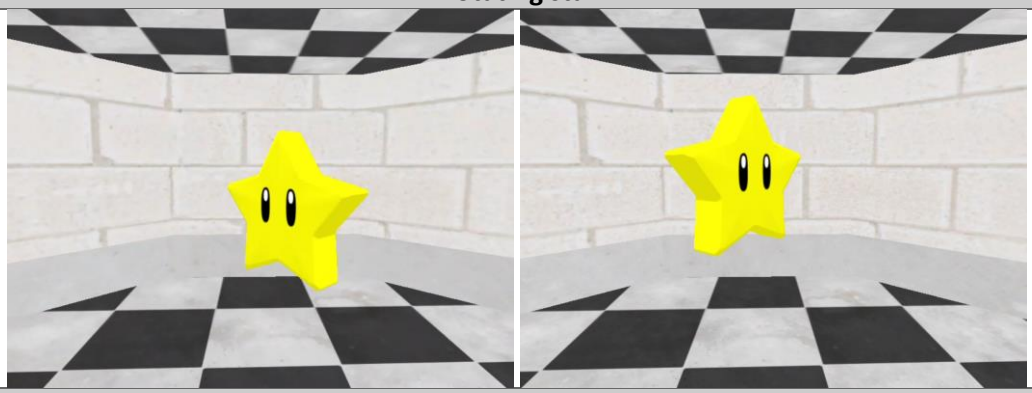

FlyGuy flying

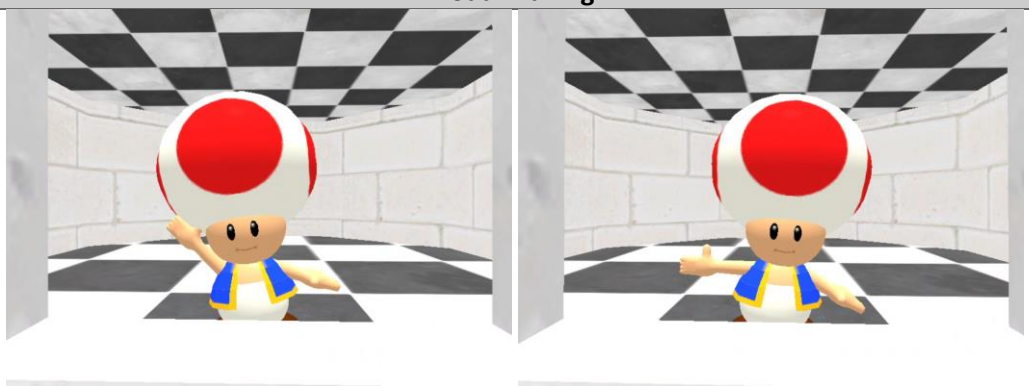
Vertical Mushroom


Windows

Rotating star



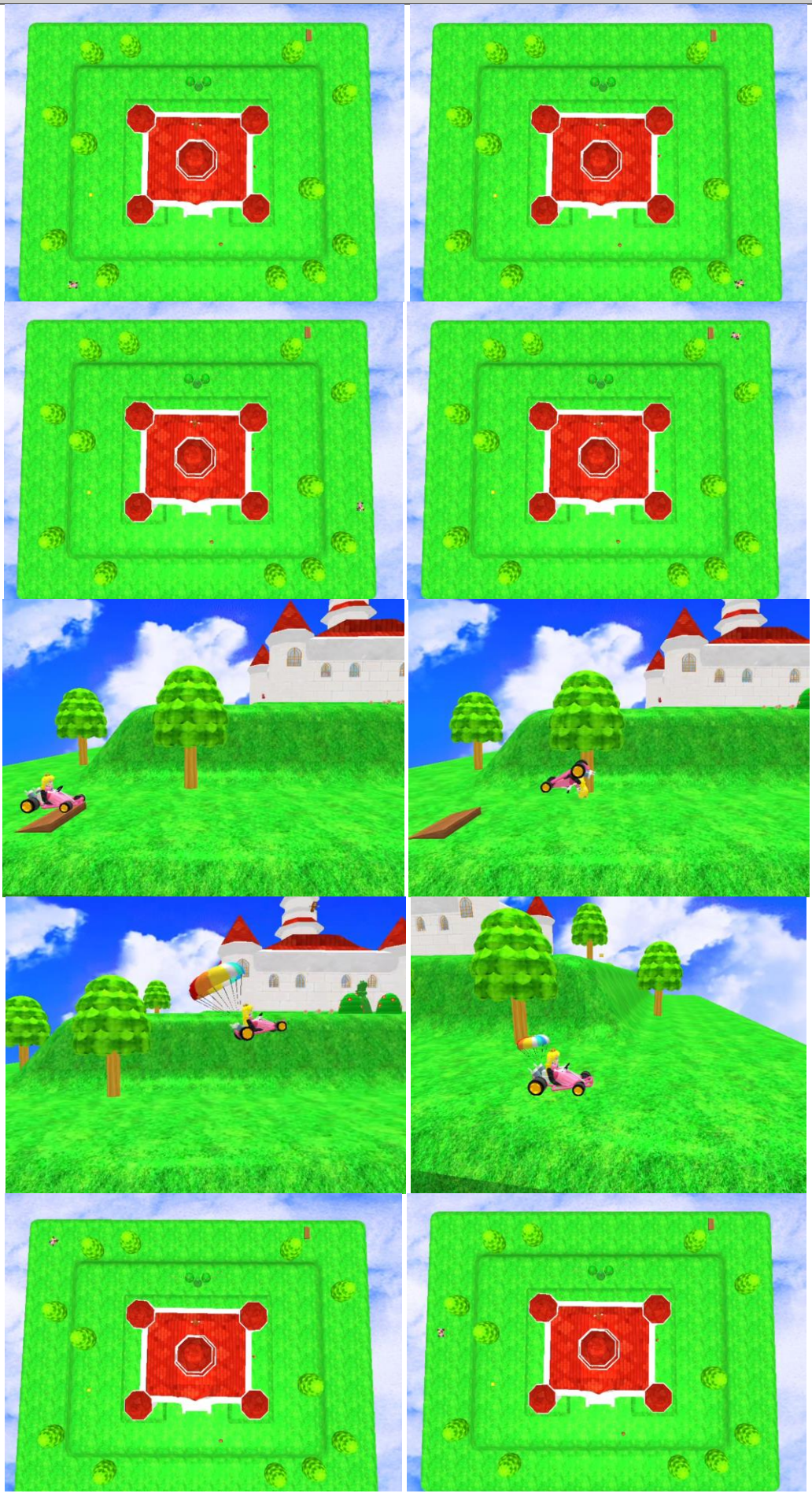
Toad Waving



Mario Tanooki

Shurbs

Peach's tour



8-CONCLUSIONS

In this technical manual, we have explored the fundamentals of OpenGL and how to apply them to create an impressive castle inspired by the world of Mario. Through the use of the 3D modeling, lighting, and animation techniques we have covered, we have successfully recreated the charm and magic of Mario games.

During the modeling process, we utilized tools like Maya to create detailed 3D models of different castle elements, including towers, walls, and Peach's iconic stained-glass window. Furthermore, we modeled representative objects and characters from this world to add an even more lively touch.

Regarding lighting, we applied techniques such as ambient, diffuse, and specular lighting to accentuate the architectural details of the castle and create a realistic atmosphere. This allowed light to properly reflect on surfaces and generate shadows that contribute depth and realism to the scene.

Animation has been a key aspect in bringing the castle to life. By implementing smooth animations, we have made characters move, jump, and interact with their surroundings.

Throughout the manual, we have provided visual examples and detailed explanations of the necessary steps to achieve this final result. Personally, this has been an enjoyable experience from start to finish, finally pursuing my passion and getting excited every time I completed a model, animated it, combined everything learned in class, and applied it to create something that I still can't believe I've accomplished, something I enjoy seeing and sharing.

9- GITHUB LINK

To access all this information or retrieve a copy of the project, please visit the following repository. Aldair20's Repository:

https://github.com/Aldair20/317222049_PROYECTOFINAL2023-2_GPO06/tree/main/ProyectoFinal/Models