UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

Technical manual

Student	Account number
Barrios López Francisco	317082555
Mendoza Anaya Aldair Israel	317222049
Páez López Didier Marcelo	317224108

Theory Group: 04

Semester 2023-2

Grade: _____

Content

Project objectives

1. Develop an interactive virtual environment with elements from the everyday life of various fictional characters.
2. Implement geometry and texturing of at least 10 elements of everyday life, including vehicles, food and drink, flora and fauna, architecture and industrial elements, and inhabitants.
3. Include basic animation, complex animation and keyframe animation in the elements of the scenario, and use a development methodology that includes a storage system, proposal document and sketch/sketch with the design of the scenario to be created.
4. Allowing the user to roam the stage with a third-person camera and an isometric camera, as well as spotlighting that reflects the day/night cycle and spotlights that can be switched on and off with the keyboard.

5. The project must be optimised to run on different hardware configurations without problems. The graphic load, the number of elements on the stage and the number of lights active at all times must be taken into account.

Presentation

The project "Diorama of everyday life" is an interactive virtual environment that allows the user to explore a scene with elements characteristic of the everyday life of various fictional characters. This project will be delivered in teams of two or three people and consists of several elements to be included in the scenario.
In order to use this project, it is important to follow the steps below:

6. Execute the exe file delivered with the project.
7. Walk around the stage using the 3rd person camera linked to a plane parallel to the XZ plane representing the floor. It is also possible to switch to an isometric camera which will allow the stage to be shown in its entirety and allows the camera to be zoomed in and out.
8. Interact with stage elements using the keyboard. Basic animations can be initiated on the stage elements, as well as interacting with the Spotlight type lights that simulate lights that we can manipulate in everyday life.
9. Observe the change in skybox textures, reflecting the day/night cycle of the virtual environment.

In terms of development methodology, a GithHub repository was used in which versions of the project were stored to demonstrate how the development progressed. A project proposal on the Google Classroom platform and a sketch of the scenario design were also delivered.

Introduction

In the field of computer graphics, the ability to model three-dimensional environments and virtual objects has been fundamental to creating immersive and realistic digital worlds. In the framework of this final project, we have developed a three-dimensional environment that includes a hut, a room and seven objects inside the room. The main objective of this project was to apply the knowledge acquired in computer graphics and to demonstrate our ability to create and manipulate virtual elements in a three-dimensional space.
In our project we have recreated a diorama with the main attraction of "Peach's Castle". Peach's Castle is an iconic place in the Mario Bros. video game universe. It appears mainly in the Super Mario series of platform games. The castle is the residence of Princess Peach, one of the main characters in the series.
The project was not only limited to depicting this castle and various objects in the diorama, but the team worked hard to give a meaning to each of the pieces in the diorama.

our recreated environment, which can be seen in a section of the user manual, as well as each animation we present.
Throughout this documentation, we will present the steps and processes we have followed to achieve our goals, from the creation and loading of three-dimensional models to the implementation of rendering techniques and real-time object manipulation.

Development methodology.

The purpose of implementing a methodology in a software development project is to establish a structured and organised approach to guide the software creation process.
The main objective of this project has been to apply the principles and techniques of graphics programming to create a convincing and visually appealing virtual environment. The accurate representation of our diorama, as well as the layout and interaction of objects in the environment, were key aspects of our approach.
We have created and placed various objects within the environment, each with their own unique characteristics and behaviours. All of this has been done using the versatile and flexible capabilities of OpenGL.
Scrum was used as a software methodology model, thanks to its flexibility, adaptability, effective management of time and priorities, and above all for the continuous improvement that we would have thanks to the use of this agile methodology.
For a better understanding of the development methodology used in this project, please refer to the "Project Plan" document, which will be attached to this document, where the progress of the project can be seen in detail.

Quality control

For the project the quality control is based on two main aspects, that the object has been modelled with respect to the proposed objects and that it is well optimised for its subsequent loading with the help of OpenGL. As the practices of semester 2023-2 progressed, as well as the theory classes with the technical explanations, the quality control increased in terms of its requirements. The first criterion that was added to consider the model suitable for the project was that it was well textured so that when it was included in the final product it would be visually attractive to the viewer. The next criterion added was the correct use of ambience, i.e. that it was sensibly and appropriately lit. Finally, the criterion of animating with meaning was considered, in case the object had an animation within the scenario, the context of why the animation will appear in the final product must first be proposed and justified.

Meeting deadlines and budget

In order to complete the project in accordance with the requirements set out from the beginning, a timetable was followed, which included the fulfilment of tasks, taking into account the possible

delays or deviations that could occur during the project and thus take the necessary measures to manage these situations and avoid a high impact on the project.

Continuous improvement

An important feature of the selected model is based on the best continuous, being a project that we will be adding objects to the diorama we proposed, and we will not have to discard the ones we have already made, the selected methodology is perfect for this project and every time a model was made or downloaded from any platform that provide useful 3D models to be able to include them within the selected room.

Each time a model was made and obtained from a platform, it was scaled, moved or rotated within the scenario to give it meaning and an aesthetically pleasing result for the end user.

The project was carried out in teams, and communication was an extremely important aspect for its successful completion. During the theory classes, as well as through a group on the social network "WhatsApp", we were able to maintain fluid and direct communication between the participants. In addition, we used the social network "Discord" to maintain active communication with the teacher, who acted as a client, which allowed us to ask questions and share comments and doubts.

Scope of the project

The scope of the project is to design a diorama depicting a scene with characteristic elements of everyday life of various fictional characters, with prior approval. The set must include correctly textured geometric elements, with at least 10 elements typical of everyday life, such as vehicles, food, flora, fauna, architecture, industrial elements, inhabitants, and 1 or 2 additional characters depending on the size of the team. In terms of lighting, light-emitting elements such as luminaires or lamps should be included on the stage, with spot lights that turn on and off simultaneously following a day/night cycle determined by the students. The skybox should also reflect the change of textures in the day/night cycle. In addition, spotlights should be added that can be switched on and off via the keyboard, simulating lights that can be manipulated in everyday life.

Students should use their 3D modelling and design skills to recreate the objects accurately, based on the images provided. In addition, they are expected to pay attention to the specific details and features of each object, as well as the overall atmosphere and style of the environment.

The final result of the project will be a three-dimensional representation of the selected space, where the recreated objects are as close as possible to the reference images and the desired atmosphere is adequately reflected.

It is important to note that the project focuses mainly on the visual and aesthetic recreation of the selected spaces and objects, using OpenGL as a tool to achieve this.

Project constraints

Availability of reference images: There may be limitations on the availability of high quality and detailed reference images for our diorama, space and selected objects. This could hinder accuracy and fidelity in the 3D recreation.

Complexity of the selected diorama and space: If the chosen diorama and space are very complex in terms of architecture, structural details or specific features, it may require a higher level of skill and effort to recreate them faithfully in the 3D environment.

Available resources and time: The project may have limitations in terms of available resources, such as the time allotted to complete the project, the processing power of the computer, or the availability of software and tools needed for OpenGL authoring.

Level of object detail: If the reference objects are very detailed or have intricate features, it can be challenging to recreate them virtually in OpenGL with a high level of fidelity and realism.

Knowledge and skills of the team: The level of skill and knowledge of the students in OpenGL and 3D modelling may affect the quality and complexity of the recreation. Limitations in terms of technical knowledge could influence rendering techniques and the implementation of advanced visual effects.

Legal limitations: If you use copyrighted reference images, you must comply with applicable laws and regulations. This may require obtaining permissions or using appropriately licensed or public domain images to avoid legal problems.

Once the main limitations of the project had been identified, we proceeded to analyse them and tried to find a balance in order to recreate the environment correctly.

To avoid the problems of complexity in the diorama, which goes hand in hand with the level of detail of the objects, it was decided to carry out the modelling personally by the team, however; This led us to another limitation, which highlights the knowledge and skills of the students to model the selected 3D spaces, as some models presented a greater challenge to the knowledge acquired until the time of the realization, we chose to resort to platforms that provide models freely, always respecting the legal limitations, being a school project and non-profit, we can use these models and adapt them to our scenario, always taking into account the aspects mentioned in the quality control.

Working platforms

Requirements

Requirements	Description
Visual Studio 2022	Visual Studio 2022 is the programme in which we can enter the solution designed for the project and interact with it.
OpenGL	OpenGL is a library required to run the program properly, you must use the version 3.3 onwards.

Modelling software	The project used the 3D design tools Blender, Maya and 3ds MAX, which can be useful if you want to visualise a model of the project individually.
Software for the creation of textures for models	To develop the project, the GIMP software was used to create, edit and export textures that will be useful for our project.

Modelling software

To create the models for the project, Maya modelling software was used to create, manipulate and export the selected 3D models.

Maya is a 3D modelling and animation software widely used in the entertainment industry, film production, video games and visual effects creation. Developed by Autodesk, Maya offers a wide range of tools and functionalities that allowed to create detailed 3D models.

It is important to mention that in order to use this software correctly we must have a computer with the minimum hardware specifications provided by the developer, which are:

CPU: 64-bit, multi-core Intel® or AMD® processor with SSE4.2 instruction set Apple Mac models with the M-series chip are supported in Rosetta 2 mode.

Graphics hardware: See the following pages for a detailed list of recommended systems and graphics cards:

Hardware certified for Maya

RAM: 8 GB RAM (16 GB or more recommended)

Disk space: 4 GB free disk space for installation

Pointing device: Three-button mouse

Software for the creation of textures for models

In this project, GIMP was used to edit and create textures that would later be assigned to the models made.

GIMP (GNU Image Manipulation Program) is a free and open source image editing software. With GIMP, users can perform a wide range of image manipulation and editing tasks, from simple colour adjustments and retouching to the creation of complex illustrations and advanced compositions.

In order to be able to use the GIMP software we also have some minimum requirements to be able to run it correctly on a computer, these are:

Operating system: GIMP is compatible with several operating systems, including Windows, macOS and Linux. It is recommended to use the most up-to-date version of the operating system to ensure best performance and compatibility.
Processor: A processor with at least 1 GHz speed is recommended for smooth operation of GIMP.
RAM: GIMP requires at least 2 GB of RAM for optimal performance. However, 4 GB or more is recommended for working with higher resolution images and performing more complex operations.
Disk space: GIMP requires at least 350 MB of free hard disk space for installation. Additional space should be considered for storing images and files created or edited with the software.
Screen resolution: GIMP works properly at a screen resolution of at least 1024x768 pixels. However, a higher resolution will allow a more comfortable and detailed display of images and tools.

Library to run our environment

As mentioned throughout the technical manual, we will use OpenGL for the correct loading of our environment.
OpenGL is a powerful graphics API that allows developers to create interactive applications and games with high-quality 2D and 3D graphics. Its portability, efficiency and performance capabilities make OpenGL a popular choice in the graphics and gaming industry.

Development environment

To develop the project we made use of an IDE (Integrated Development Environment), this was Visual Studio 2022.
Visual Studio 2022 is the latest version of Microsoft's software development suite. It is a widely used integrated development environment (IDE) that provides advanced tools and features for creating applications for various operating systems, platforms and devices.
Visual Studio 2022 is a powerful and versatile IDE that gives developers the tools they need to create quality applications across platforms and programming languages. Its feature set, integration with cloud services and focus on productivity make it a popular choice among software developers.
Minimum requirements for the working environment.

- ARM64 or x64 processor; quad-core or higher recommended. ARM 32 processors are not supported.
- 4 GB of memory, minimum. There are many factors that affect the resources used, 16 GB of RAM is recommended for typical professional solutions.
- Windows 365: Minimum 2 vCPU and 8 GB RAM. 4 vCPU and 16 GB RAM recommended.
- Hard disk space: minimum of 850 MB and up to 210 GB of available space, depending on the features installed; typical installations require between 20 to

50 GB of free space. It is recommended to install Windows and Visual Studio on a solid state drive (SSD) to improve performance.

- Video card that supports a minimum screen resolution of WXGA (1366 x 768); Visual Studio will work best with a resolution of 1920 x 1080 or higher.
    - The minimum resolution assumes that the zoom, DPI setting and text scaling are set to 100 %. If it is not set to 100%, the minimum resolution must be adjusted accordingly. For example, if you set the Windows display setting "Scale and Layout" on the Surface Book, which has a physical display of 3000 x 2000, to 200%, Visual Studio would see a logical display resolution of 1500 x 1000, meeting the minimum requirement of 1366 x 768.

Once we have installed the necessary software to be able to develop our project, we can start developing it.

In order to be able to load the models correctly with the help of OpenGL we will first make use of some external libraries that will be linked through the Visual Studio 2022 work window.

GLEW (The OpenGL Extension Wrangler Library):

GLEW is an external library used in OpenGL application development. Its main function is to facilitate the management of OpenGL extensions on different platforms. GLEW provides a unified interface for dynamically loading and using OpenGL extensions, allowing developers to access advanced OpenGL features and functionality not available in the base implementation. GLEW simplifies the management of OpenGL extensions and allows programmers to take full advantage of the API's capabilities.

GLFW (Graphics Library Framework):

GLFW is an open source library that provides an application programming interface (API) for creating and managing windows, OpenGL contexts and event handling in graphics applications. Its main function is to abstract the creation and management of windows and graphics contexts across different platforms, which facilitates the development of cross-platform OpenGL applications. GLFW also offers additional features, such as keyboard and mouse input detection, event handling, multiple monitor support and timer management. GLFW simplifies window creation and event handling in graphics applications.

glm (OpenGL Mathematics):

glm is an open source mathematical library designed specifically for use with OpenGL. It provides a wide range of mathematical functions and classes that are useful in the development of 3D graphics and rendering applications. glm provides functionality to perform vector calculations, matrices, transformations, projections, intersections and other mathematical operations common to 3D graphics. This library is inspired by the GLSL (OpenGL Shading Language) syntax and provides an intuitive and easy-to-use interface for performing mathematical calculations in the context of OpenGL.

SOIL2 (Simple OpenGL Image Library):

SOIL2 is an external library used to load, manipulate and save images in OpenGL applications. It provides functions and utilities for loading textures in common image formats, such as JPEG, PNG, BMP and TGA. SOIL2 simplifies the process of loading textures into OpenGL, allowing developers to efficiently load images and apply them to 3D objects in their graphics applications.

assimp (Open Asset Import Library):

assimp is an external library that provides functions and tools for importing and processing 3D models in different formats into graphics applications. Its main function is to facilitate the import of 3D models from a variety of common formats, such as OBJ, FBX, Collada, DirectX, among others. assimp takes care of loading geometry, textures, materials and other data related to 3D models, allowing developers to use external models efficiently and easily in their applications. In addition, assimp offers additional functionalities such as mesh optimisation, scene node manipulation and animation management. It simplifies the process of importing and processing 3D models in graphics applications.

irrKlang

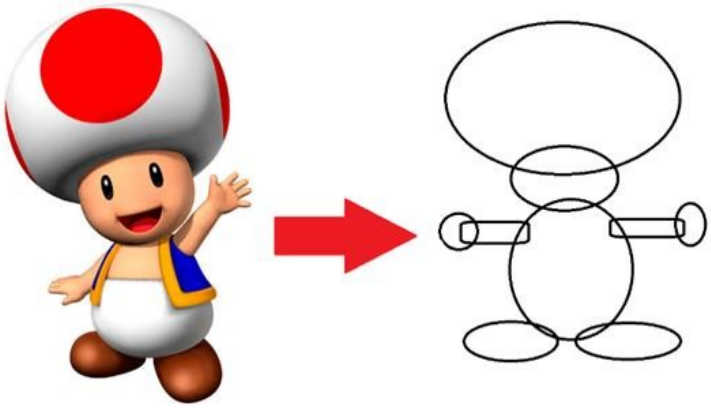
IrrKlang is a high-performance cross-platform audio library designed to provide an easy-to-use solution for sound playback in applications and games. With its wide compatibility, it allows developers to efficiently implement sound effects, background music and any other type of audio in their projects.

In addition, it offers advanced features such as 3D sound positioning support, enabling an immersive experience by simulating the spatial location of sound effects. Thanks to its efficient architecture, IrrKlang delivers optimal performance even on resource-constrained systems, making it an ideal choice for applications and games on platforms such as Windows, macOS, Linux and mobile devices.

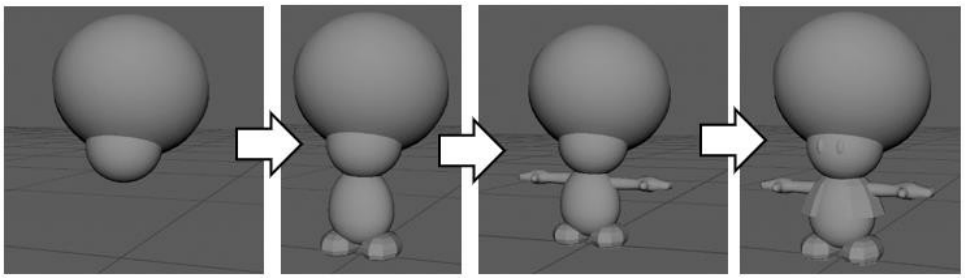
CREATIVE PROCESS

To begin the project, we started with the creative process of animation in real time, which consists first of modelling, followed by texturing, the animation if any, and finally the rendering of the model in the proposed scenario.

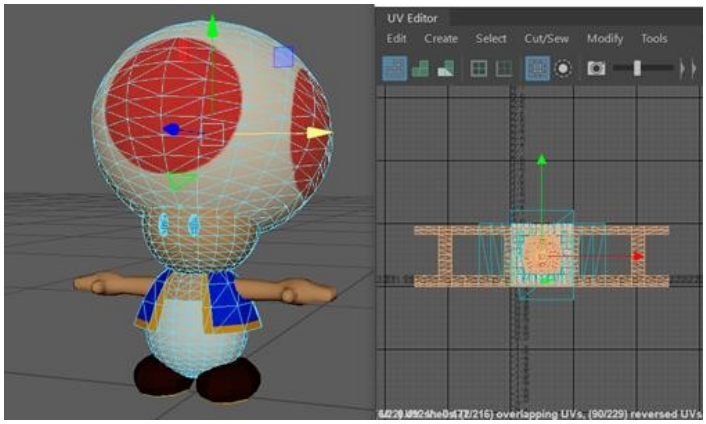
Modelling in Maya is a creative and technical process that brings ideas and concepts to life in the form of 3D models. It begins with the planning and conceptualisation of the objects to be created. This involves defining the basic shape, proportions and specific details of each component that will make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



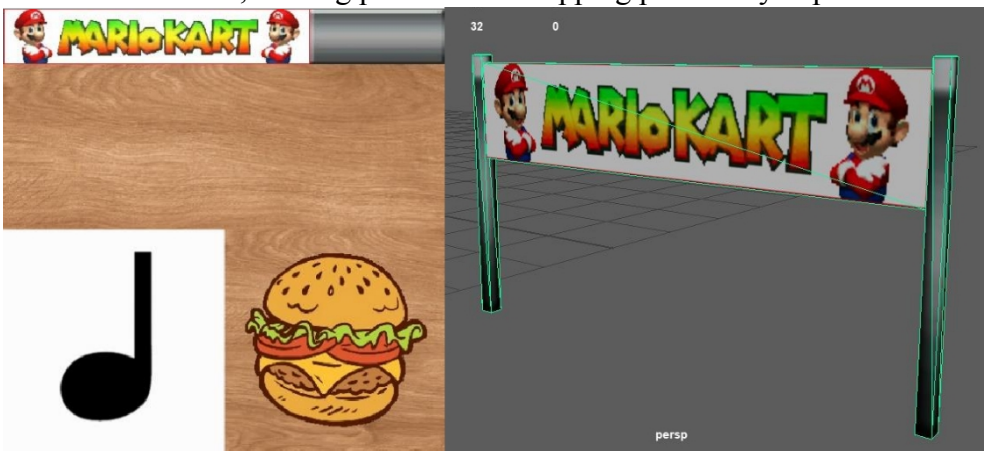
Once finished with the object, it is analysed from all possible perspectives and with the three available views, to determine if it can be optimised or with the object you have is more closely resembles the desired object/character. There were cases where it was decided to remove faces of the object 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.

Primitives:

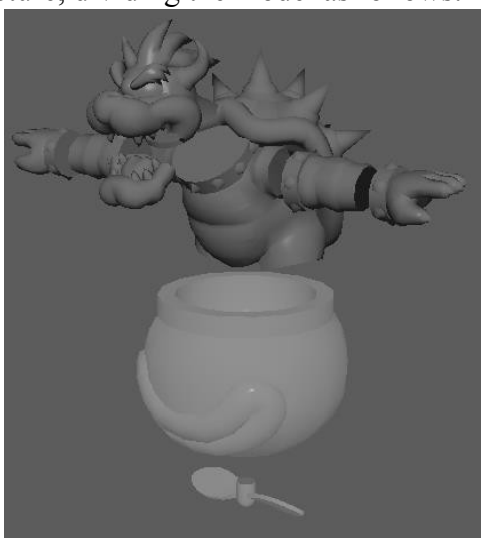
The project also included objects drawn from primitives declared in code. More specifically, the hamburger sign and the sign at the start of the track.

Within the same arrangement of vertices and indices, the triangles needed to create these figures are drawn. Using the concept of texturing seen in class, a single image is used to texture these figures, so the mapping was even more meticulous, recreating one cube for the musical note cube, two cubes and two planes to make the goal and two cubes for the hamburger sign. To obtain the locations of the vertices, the models were created in Maya, exporting them without texturing, as we are only interested in the .obj file, as there are the coordinates of each vertex, making possible the mapping previously explained.



AVATAR

For our project set in the world of Mario Bros, we have chosen Bowser as the main avatar, using the model seen in New Super Mario Bros Wii. To build the avatar in OpenGL, we adopted a hierarchical structure, dividing the model as follows:



- The Koopa Clown acts as the main element, being the father.
- Bowser's propellers and torso are his children.
- Bowser's torso is the father of the arms and head.

- The head is the parent of the jaw.
- The arms are the parents of the forearms.
- The forearms are the parents of the hands.

To achieve the animation, we have used an incrementable that resets its values, allowing the propellers to rotate constantly, justifying their ability to move even at high altitudes. For Bowser's body, we have used sine functions that give a natural rotation to each part of his body.

In addition, we have added an additional animation to recreate his roar. To achieve this, we extended the value by which the sine function is multiplied, which causes each part of his body to rotate further backwards, creating the illusion of him letting out a roar. As it reaches this ideal position, it starts a counter that sets a flag. When the flag is active, we reset the values at an interval to make the model shake, simulating the trembling of the roar. At the same time, an additional sine function causes its head to rotate from side to side, adding dynamism to the roar. During this process, we play audio containing Bowser's roar, bringing the scene to life.

Once the counter reaches a certain value, the flag is deactivated and the avatar continues its path in the sine function to return to its initial position. The animation returns to an "IDLE" state with a smaller amplitude in the sine function. This completes the cycle of the roar.

RECORRIDO

3D camera

The avatar is linked to a 3D camera which is based on the free camera provided and used throughout the semester.

Camera.h

This file handles both the 3D camera and the isometric camera. Starting with the 3D camera, the void function ProcessKeyboard(Camera_Movement direction, GLfloat deltaTime) is modified to restrict the position of the free camera:

```
GLfloat velocity = this->movementSpeed * deltaTime;
if (!isometric) {
    if (direction == FORWARD) this->position += this->front * velocity;
    if (direction == BACKWARD) this->position -= this->front * velocity;
    if (direction == LEFT) this->position -= this->right * velocity;
    if (direction == RIGHT) this->position += this->right * velocity;
    if (this->position.y < 2.0f) this->position.y = 2.0f;
    if (this->position.y > 40.0f) this->position.y = 40.0f;
    if (this->position.x < -20.0f) this->position.x = -20.0f;
    if (this->position.x > 20.0f) this->position.x = 20.0f;
    if (this->position.z < -15.0f) this->position.z = -15.0f;
    if (this->position.z > 30.0f) this->position.z = 30.0f;
}
```

When the isometric camera is not active, the camera position is modified. The movement of the camera is restricted by conditionals so that the character does not leave the boundary of the world and does not cross the floor.

Within this file, a function is generated that returns the horizontal mouse position, whose value ranges from 0 to 360°.

```
GLfloat getYaw() { return this->yaw; }
```

MainTest.cpp

In order to link the camera to the character, the character will receive the position of the camera within the translation. When rotating the camera, the avatar must rotate along with the camera. For this, it is known that, when moving the camera, a circle is generated by which the avatar must adjust to the camera. For this, two variables are generated:

```
BowserCameraX = 1.75f * glm::cos(glm::radians(camera.getYaw()));
BowserCameraZ = 1.5f * glm::sin(glm::radians(camera.getYaw()));
```

As the avatar is drawn by hierarchy from the cup, this is the one where the camera will be positioned.

```
model = glm::translate(model, glm::vec3(camera.GetPosition().x + BowserCameraX,
    camera.GetPosition().y + (auxilar1 / 2000) - 1.0f, camera.GetPosition().z + BowserCameraZ));
model = glm::rotate(model, glm::radians(- camera.getYaw() + 90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
KoopaClown.Draw(lightingShader);
```

With this, the character turns when the camera turns, and the character is always seen looking straight ahead.

Isometric camera

Camera.h

For the isometric camera, we started by defining the following variables:

```
// variables camara isometrica
bool isometric = false;
GLfloat iso_right, iso_up;
glm::vec3 iso_position;
GLfloat iso_zoom = 2.0f;
```

In turn, the following functions are declared:

```
bool getIsometric() { return this->isometric; }
GLfloat getIsoZoom() { return this->iso_zoom; }
void setZoom(GLfloat zoom) { this->iso_zoom = zoom; }

void setIsometric(GLfloat iso) { this->isometric = iso; }
```

The first function defines whether the isometric camera is active or not. The second one gets the zoom value which will modify the viewing area of the diorama. The third function modifies the zoom, because the keyboard control is done externally.

Within the function ProcessKeyboard(Camera_Movement direction, GLfloat deltaTime), the camera is moved sideways with WASD. Limits are set for each side to prevent the camera from moving out of the diorama's field of view.

```
else {
    if (direction == FORWARD) {
        iso_up += 0.5f;
        if (iso_up >= 20.0f) iso_up = 20.0f;
    }
    if (direction == BACKWARD) {
        iso_up -= 0.5f;
        if (iso_up <= -25.0f) iso_up = -25.0f;
    }
    if (direction == LEFT) {
        iso_right -= 0.5f;
        if (iso_right <= -20.0f) iso_right = -20.0f;
    }
    if (direction == RIGHT) {
        iso_right += 0.5f;
        if (iso_right >= 20.0f) iso_right = 20.0f;
    }
    iso_position = glm::vec3(iso_right, iso_up, iso_right);
}
```

In order for the 3D camera to save the camera state when switching cameras, the mouse movement is only used when the isometric camera is deactivated:

```
if (!isometric) {
    this->yaw += xOffset;
    this->pitch += yOffset;
}
```

Within the glm::mat4 function GetViewMatrix(), the camera view is obtained, depending on which camera is active:

```
glm::mat4 GetViewMatrix()
{
    if (isometric == false) return glm::lookAt(this->position, this->position + this->front, this->up);
    else return glm::lookAt(iso_position, iso_position + glm::vec3(1.0f, 0.0f, -1.0f), glm::vec3(0.0f, 1.0f, 0.0f));
}
```

Finally, in order for the models to be drawn in the isometric camera, each model matrix is modified to perform two rotations.

```
glm::mat4 ConfIsometric(glm::mat4 model) {
    model = glm::rotate(model, glm::radians(45.0f), glm::vec3(1.0f, 0.0f, 0.0f));
    model = glm::rotate(model, glm::radians(35.2644f), glm::vec3(0.0f, 0.0f, 1.0f));
    return model;
}
```

MainTest.cpp

The isometric camera uses the orthogonal projection. Within the while cycle, you define which projection is to be used:

```
if (!camera.getIsometric()) {
    projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
else {
    projection = glm::ortho(-camera.getIsoZoom(), camera.getIsoZoom(), -camera.getIsoZoom(), camera.getIsoZoom(), -30.0f, 40.f);
```

Within the same file, I is used to activate the isometric camera, and U is used to deactivate it. At the same time, when the up arrow is activated, the zoom is increased to 2.0 (the maximum zoom), and when the down arrow is pressed, the zoom is decreased to 30.0f (all visible scenery).

```
if (keys[GLFW_KEY_I]) camera.setIsometric(true);
if (keys[GLFW_KEY_U]) camera.setIsometric(false);
if (keys[GLFW_KEY_DOWN]) {
    camera.setZoom(camera.getIsoZoom() + 0.5f);
    if (camera.getIsoZoom() >= 30.0f) camera.setZoom(30.0f);
}
if (keys[GLFW_KEY_UP]) {
    camera.setZoom(camera.getIsoZoom() - 0.5f);
    if (camera.getIsoZoom() <= 2.0f) camera.setZoom(2.0f);
}
```

As mentioned, each model uses the ConfIsometric(model) function to correctly draw each model in the isometric view.

```
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
```

LIGHTING

For this section, two distinct environments are considered: day and night. Inspired by Super Mario Maker 2, the lighting characteristics of both versions were used to create the lighting in the castle environment.

During the day, it was chosen to simulate the presence of a sunlight beam by means of a directional light with direction values of (0, -1, -1). This light will provide a main illumination coming from a specific direction. On the other hand, the ambient lighting was set to a value of (0.9, 0.9, 0.9) to achieve a colourful and cheerful atmosphere during the day. At this stage, no objects will glow, and all objects will be illuminated in a colourful way to match the daytime illumination. In addition, the skybox will use images of a sky with clouds to reflect the appearance of the daytime environment.

In contrast, during the night, it was decided to slightly reduce the intensity of the colours and emphasise them in bluer tones. To achieve this, the ambient lighting was adjusted to (0.6, 0.6, 0.8) during the night. The directional light was set to (0.0, 0.0, 0.0) because at night there are no direct sunlight rays. Instead, the illumination will depend mainly on spot lights that will be seen only during this period.

Three Point lights will be used to highlight specific elements of the night environment:

- The first Point light will be located at the top of the castle and will be present regardless of the state of day or night. This light shall provide a white illumination that simulates starlight, but shall not be as intense as sunlight during the day. To achieve this effect, values of (1.0, 1.0, 1.0) will be used for the diffuse and specular, while the linear and quadratic components will be set to values close to zero to allow for a wider range of light.
- The second Point light will be placed above the Purohuesos grill to simulate the presence of a lit fire. This light will have a more limited range radius, so its linear and quadratic components will be set to 5 and 3, respectively. The colour emitted by this light will be red, using the values (1.0, 0.0, 0.0), to simulate the colour of the grill fire.
- For the last Point light, it is placed on the master sword, which will use a cyan light with the values (0.0, 1.0, 1.0), and the blade will emit a cyan light on the blade side. When the corresponding animation is activated, it will light a

SpotLight to illuminate both the sword and the stone, as shown in The Legend of Zelda games, and will be deactivated when the animation is finished.

To achieve these changes in lighting and skybox, a counter will be used which, when reaching its maximum value, will reset the counter and change the boolean indicating whether it is day or night.

Two Skybox vectors are used, one for day and one for night.

```
//Texturas para SkyBox
vector<const GLchar*> facesDias, facesNoche;

facesDias.push_back("SkyBox/right.tga");
facesDias.push_back("SkyBox/left.tga");
facesDias.push_back("SkyBox/top.tga");
facesDias.push_back("SkyBox/bottom.tga");
facesDias.push_back("SkyBox/back.tga");
facesDias.push_back("SkyBox/front.tga");

facesNoche.push_back("SkyBox(Noche)/right.tga");
facesNoche.push_back("SkyBox(Noche)/left.tga");
facesNoche.push_back("SkyBox(Noche)/top.tga");
facesNoche.push_back("SkyBox(Noche)/bottom.tga");
facesNoche.push_back("SkyBox(Noche)/back.tga");
facesNoche.push_back("SkyBox(Noche)/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(facesDias);
```

This will also activate the appropriate lighting values for each state, turning the Point lights on when switching to the night state and off when switching to the day state. In addition, when the boolean is set to "true" (daytime state), the skybox will be illuminated with images of clouds in a clear blue sky, while when set to "false" (nighttime state), images of the starry sky will be used.

```
if (EstadoDia) GLuint cubemapTexture = TextureLoading::LoadCubemap(facesDias);
else GLuint cubemapTexture = TextureLoading::LoadCubemap(facesNoche);
```

To change the state between day and night, a code structure like the following is used:

```
LaHora += deltaTime;
if (EstadoDia)
{
    if (LaHora > 59.72f)
    {
        LaHora = 0.0f;
        EstadoDia = false;
        PointlightTrue = 1.0f;
        ambiental1 = 0.6f;
        ambiental2 = 0.8f;
        directionalAmbiental = 0.0f;
        PosteLuzB = 0.0f;
        PosteLuzRG = 1.0f;
        SoundEngineNigth->play2D(bocina2, false);
    }
}
```

Point Lights On/Off

According to the previous code structure, certain variables are modified which correspond to the cooking of lights for the poles, in order to avoid drawing a large number of Point Lights but simulating that the lights are there. The poles are made up of the pole as such, and the stars that simulate the spotlights.

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), PosteLuzRG, PosteLuzRG, PosteLuzB);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.9f, 0.9f, 0.9f);
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
model = glm::translate(model, glm::vec3(8.854f, 2.559f, 27.501f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Estrellitas.Draw(lightingShader);
```

In addition, there are lights that illuminate the star, the grille, and the sword. The definition of each Point Light takes as arguments for the diffuse and specular components the modified variables in the day/night cycle, thus achieving the on/off effect.

```
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 0.0f * PointlightTrue, 1.0f * PointlightTrue, 1.0f * PointlightTrue);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.0f * PointlightTrue, 1.0f * PointlightTrue, 1.0f * PointlightTrue);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.054f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.0192f);
```

Switching SpotLights on/off

To switch the SpotLights on and off, keys 2 and 3 are used:

```
if (keys[GLFW_KEY_2]) {
    Reflectores = 1.0f;
}
if (keys[GLFW_KEY_3]) {
    Reflectores = 0.0f;
}
```

For the reflectors, the technique of "cooked lighting" was used in the reflector space that is illuminated when the SpotLight is switched on. For this, the model was separated into two parts, one consisting of the metal part of the reflector and one corresponding to the spotlight of the reflector. By means of variables, the direction, ambient, diffuse and specular components are modified, and with this the focus of each reflector is drawn.

```
if (Reflectores>0) {
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, 0.0f, 0.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 2.0f, 2.0f, 2.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.9f, 0.9f, 0.9f);
}
if (Reflectores < 1.0) {
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), 0.0f, -directionalAmbiental, -directionalAmbient
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), ambiental1, ambiental1, ambiental2);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.2f, 0.2f, 0.2f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.1f, 0.1f, 0.1f);
}
model = glm::mat4(1);
if (camera.getIsometric()) model = camera.ConfIsometric(model);
model = glm::translate(model, glm::vec3(8.213f, 0.595f, 21.701f));
model = glm::scale(model, glm::vec3(-1.0f, 1.0f, 1.0f));
glUniform4f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0f, 1.0f, 1.0f, 0.1f);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Foco.Draw(lightingShader);
```

The definition of each SpotLight is based on the reflector variables (because the light of the sword depends on other variables explained below). With this, when 2 is pressed, the SpotLight will be drawn, and with 3 it will not.

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].position"), SpotLightPositions[1].x, SpotLightPositions[1].y, SpotLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].direction"), SpotLightDirections[1].x, SpotLightDirections[1].y, SpotLightDirections[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].ambient"), Reflectores, Reflectores, Reflectores);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].diffuse"), Reflectores * 0.1f, Reflectores * 0.1f, Reflectores * 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].specular"), 1.0f, 0.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].linear"), 0.1f);//0.09
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].quadratic"), 0.01f);//0.032
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].outerCutOff"), glm::cos(glm::radians(15.0f)));
```

Isometric camera lighting

Because the isometric camera is based on two-axis rotations of the models, the lights must also be adjusted to be displayed correctly. This is achieved by passing the

position of the Point Lights, and the position and direction of the Spot Lights to the ConfIsometric(model) function.
We start by defining the position matrix of each Point Light:

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(6.766,0.670,24.985),
    glm::vec3(0.0f, 9.751, 6.016),
    glm::vec3(0,1.0,19.892)
};

// Positions of the Spot lights
glm::vec3 SpotLightPositions[] = {
    glm::vec3(6.766f, 5.0f, 24.985f),
    glm::vec3(8.099f, 1.002f, 21.799f),
    glm::vec3(-8.099f, 1.002f, 21.799f)
};

// Directions of the Spot lights
glm::vec3 SpotLightDirections[] = {
    glm::vec3(6.766f, 5.0f, 24.985f),
    glm::vec3(-8.099f, 2.909f, 3.455f),
    glm::vec3(8.099f, 2.909f, 3.455f)
};
```

The lights in their definition take the values of the previous arrangements. When the isometric camera is deactivated, the positions are retained:

```
if (!camera.getIsometric()) {
    projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
    pointLightPositions[0] = glm::vec3(6.766, 0.670, 24.985);
    pointLightPositions[1] = glm::vec3(0.0f, 9.751f, 6.016f);
    pointLightPositions[2] = glm::vec3(0, 1.0, 19.892);

    SpotLightPositions[0] = glm::vec3(6.766f, 5.0f, 24.985f);
    SpotLightPositions[1] = glm::vec3(8.099f, 1.002f, 21.799f);
    SpotLightPositions[2] = glm::vec3(-8.099f, 1.002f, 21.799f);

    SpotLightDirections[0] = glm::vec3(0.0f, -1.0f, 0.0f);
    SpotLightDirections[1] = glm::vec3(-8.099f, 2.909f, 3.455f);
    SpotLightDirections[2] = glm::vec3(8.099f, 2.909f, 3.455f);
}
```

When the isometric camera is active, the positions are modified:

```
else {
    projection = glm::ortho(-camera.getIsoZoom(), camera.getIsoZoom(), -camera.getIsoZoom
    glm::mat4 model(1);
    model = camera.ConfIsometric(model);
    pointLightPositions[0] = glm::vec3(model * glm::vec4(6.766, 0.670, 24.985, 1.0f));
    pointLightPositions[1] = glm::vec3(model * glm::vec4(0.0f, 9.751f, 6.016f, 1.0f));
    pointLightPositions[2] = glm::vec3(model * glm::vec4(0, 1.0, 19.892, 1.0f));

    SpotLightPositions[0] = glm::vec3(model * glm::vec4(6.766, 5.0f, 24.985, 1.0f));
    SpotLightPositions[1] = glm::vec3(model * glm::vec4(8.099f, 1.002f, 21.799f, 1.0f));
    SpotLightPositions[2] = glm::vec3(model * glm::vec4(-8.099f, 1.002f, 21.799f, 1.0f));

    SpotLightDirections[0] = glm::vec3(model * glm::vec4(0.0f, -1.0f, 0.0f, 1.0f));
    SpotLightDirections[1] = glm::vec3(model * glm::vec4(-8.099f, 2.909f, 3.455f, 1.0f));
    SpotLightDirections[2] = glm::vec3(model * glm::vec4(8.099f, 2.909f, 3.455f, 1.0f));
}
```

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.

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

Simple animations

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: an "IDLE" variable and a "RotBrazo" variable. These variables are incremented and decremented at different speeds, 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.

In order to avoid doing combined transformations manually, we chose to make the Toad hierarchical, 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. First the Goomba will rotate his head from side to side without any other movement; when the user presses the corresponding key, the Goomba moves from side to side on the X-axis while rotating his head and slightly lifting his feet on the Z-axis, following his characteristic walk. To achieve this effect, we use increments and decrements in specific variables and a boolean that determines when to activate the animation. 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.

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 Z-axis, while simultaneously increasing their position on the Y-axis when their rotation is 0 degrees, creating a hopping effect. This is done by increments and decrements that control both the rotation and the hopping of the flower, while its direction is being controlled by a boolean.

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 X-axis position of the FlyGuy, 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 is restarted, 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.

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.

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.

Piranha plant biting

The animation of the piranha plant begins with its initial static position inside the pipe. Using a variable increment and decrement on the Y-axis, the piranha plant performs a smooth movement inside the pipe. Meanwhile, by pressing the corresponding button, the piranha plant rises and falls on the Y-axis, while its stem changes size to simulate a stretch. During this displacement, the piranha plant performs

its iconic mouth opening and closing animation. The implementation of this animation is based on the use of variables to control different aspects. A Y position variable is incremented and decremented to make the piranha plant move along its path. At the same time, a scale variable is adjusted so that the stem appears to be stretching as the plant moves. In addition, a counting variable is used that varies from 0 to 1 and is rapidly reset. Each time this reset occurs, the rotation of the piranha plant changes from 0 to 45 degrees and vice versa. When it finishes its run and returns to the pipe it will remain static as at the beginning of the description so that the user can re-activate its animation.

Bob-omb marching

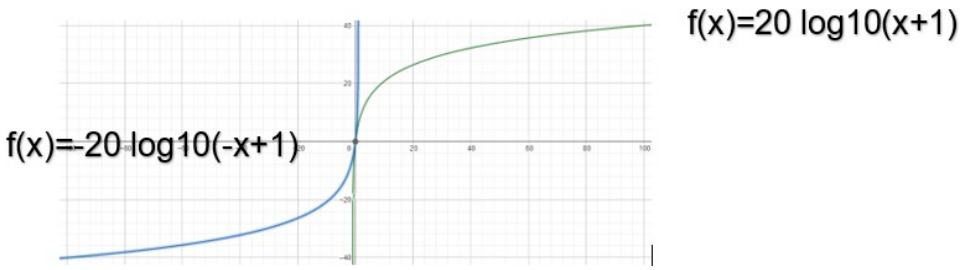
The bob-omb is initially in a static, stationary position until the user activates its animation by pressing the corresponding button. Once the animation starts, the bob-omb's hand begins to rotate on the Z-axis while the legs go up and down. At the same time, the bob-omb's head rotates on the Y-axis to give more dynamism to the movement.

To achieve this animation, Boolean-controlled incrementing and decrementing variables are used to affect the position of the bob-omb's feet and head. Also, a knob is used that increments from 0 to 360 and then restarts its cycle. When the animation time reaches its limit, the bob-omb returns to its initial position and becomes static again, waiting for the user to activate the animation once more.

Complex Animations

Toad waving his arms

The animation of Toad performing the alternate 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 the change of position is made. This effect is achieved by

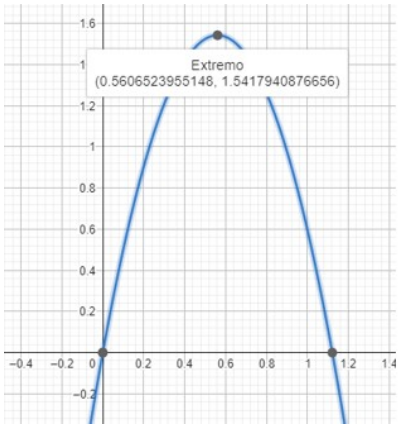
by exploiting the properties of the logarithm, which creates a pause in the movement at certain specific points.
An interesting detail of this animation is that, when the maximum point of rotation is reached, regardless of whether it is to the right or to the left, there is a slight increase in the overall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

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. In addition, a shader was used to make the painting it passes through move, giving a jelly effect, recreating the animation and logic of the Mario 64 frames, always using the position of the paratroopa's flight as a reference.

Mushroom inside the block

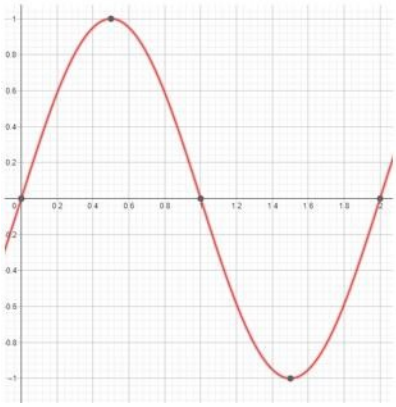
The mushroom is shot from a question block and is launched upwards following the rules of physics. As it rises, it loses speed until it reaches its highest point and then begins to fall. During this process, the cube will scale, becoming smaller, while another smaller cube will become larger to represent 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 more dynamism to the spin. 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.



$$f(x) = 5.5\,x - 0.5 * 9.81\,x^2$$

Mario Tanooki flying

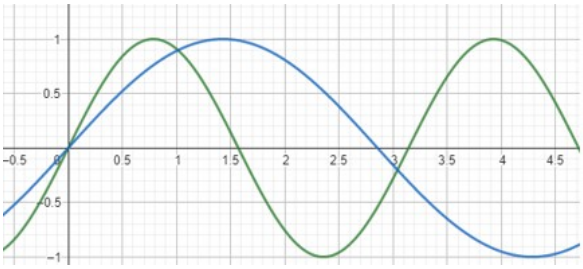
Mario rises and falls as he moves his tail up and down, which is achieved by a function Sen(x) that determines the rise and fall of his height. This function creates a smooth and fluid effect on Mario's vertical movement. To avoid filling up memory the variable X is reset to 0 so that the cycle continues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



$$f(x) = sen(x * 3.1416)$$

Luigi playing with a Nintendo Switch

In the animation of Luigi playing with his Nintendo Switch, the character is inside the castle while having fun with the console. While playing, Luigi moves his arms and head from side to side to simulate using the console's gyroscope. His feet do not rotate, but move slightly up and down to simulate the effect of his arms swinging. To achieve this effect, two sine functions of x are used. Each variable increases at a different speed, causing Luigi's arms and head to rotate at different rates. However, because they are built in a hierarchy, this creates a more fluid and realistic movement in the animation of Luigi playing.



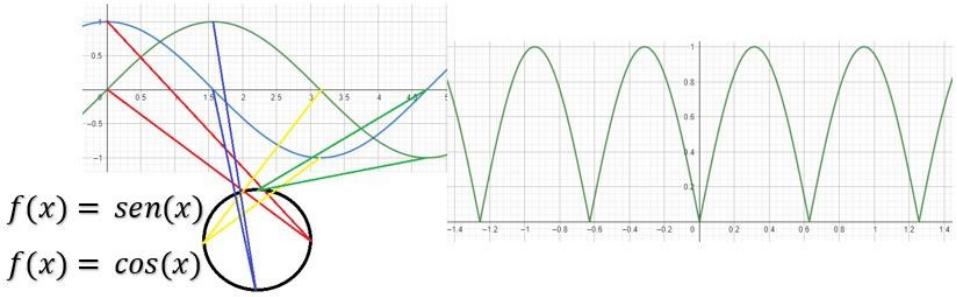
$$f(x) = sen(x)$$

$$f(Y) = sen(Y)$$

The result is that Luigi appears to be immersed in his game, moving his arms and head dynamically as he enjoys the gaming experience with his Nintendo Switch. The slight movement of his feet up and down adds an extra touch of realism and naturalness to the overall movement.

Chilly Willy escaping from Chain Chomp

In the Chilly Willy escaping Chain Chomp animation, the character Chilly Willy will be driving a Mario Kart vehicle around a Mario-shaped bush. Behind him, a Chain Chomp will follow the same route as he opens and closes his mouth and does little hops characteristic of Mario Kart games. This will generate a chase in circles around the Mario-shaped bush. To achieve this animation, sine and cosine functions are used on each object involved. These functions are used to exploit the principle of a circle. Both formulas are given the same values, but generate different results. By applying this transformation to the X and Z coordinates of the objects, a circle is drawn in the plane. For the Chain Chomp jumps, only a sine function is used. The variable used in this function only varies from 0 to a value resulting in 0, avoiding negative values and thus generating the characteristic jumps. This approach ensures that the jumps are maintained in an upward and downward trajectory. Finally, to make the objects rotate on the Y-axis according to their position, an auxiliary variable is used. When this variable reaches its maximum value, it is multiplied by a number that results in 360 degrees. As 0 and 360 degrees are equivalent, the resetting of the values is not noticed and the animation repeats over and over again.



$$f(x) = sen(x)$$

$$f(x) = cos(x)$$

This makes for a captivating animation in which Chilly Willy escapes from the Chain Chomp as they circle around the Mario-shaped bush. The characters' characteristic jumps and movements add dynamism and fun to the animation.

Cheep Cheep swimming

To achieve this effect, a sine function is used to determine the upward and downward movement of the fish. This sine function creates a smooth, undulating trajectory, creating the sensation of swimming in the water. The Cheep Cheep rises and falls in harmony with this function, imitating the movements of a swimming fish. In addition, the fins of the Cheep Cheep Cheep also

are animated and move according to the same sine pattern. Because they are built in a hierarchy, the fins follow the general movements of the Cheep Cheep, giving the illusion that it uses them to swim and adding to the realism of the animation.

Princess Peach dancing with coins

8 coins will be presented in a circle and will rotate rapidly, similar to the Chilly Willy animation. As the coins rotate, Peach will be performing a constant 360-degree rotation while moving in an elliptical motion, simulating a dance.

The principle of the formula for drawing a circle using sine and cosine functions is reused. However, in this case, the formula is repeated continuously, but the value within the sine function is changed. Due to the cyclic and continuous nature of this function, it does not matter whether the value is positive or negative. This allows all the coins to travel around the same circle, but each in a different position, thus forming the ring of coins. Peach will follow the same principle, but on one of the axes, the distance between the points will be smaller to create the ellipse shape. This will give a more dynamic and stylised movement to her dancing in the coin ring. For Peach's rotation, the same auxiliary variable mentioned above will be used, similar to Chilly Willy's animation. However, in this case, the speed of the variable will be increased so that Peach's spins are not completed at the same speed as the coins move along the path. This will add an interesting visual effect and differentiate Peach's movements from the rest of the animation.

Pure Bones making burgers

Puro Huesos will be presented in front of a grill holding a spatula. While holding the spatula, Puro Huesos will move his arm in a hierarchical manner, at the same time as a hamburger is thrown into the air using the properties of a vertical shot. The burger will fall back onto the grill and Puro Huesos will place the spatula back underneath the burger to throw it again. There will be a wait time between each start of the animation to give the impression that it is cooking the burgers.

To achieve this animation, a simple animation will be used for the Pure Bones arm and a more complex animation for the hamburger. An incrementable will be used to control the timing, and when it reaches a certain point, the rotation of the arm and forearm will be increased or decreased to simulate the motion of turning the burger on the grill. The animation of the burger will serve as a sort of "flag" to indicate when the vertical flip should be performed. Once the burger falls, it will again be used as a "flag" to indicate to the arm when to come down and repeat the process. This continuous cycle of burger throwing and flipping of Puro Huesos will create a fun and dynamic animation that simulates the process of cooking burgers.

Koopas waiting for their burger

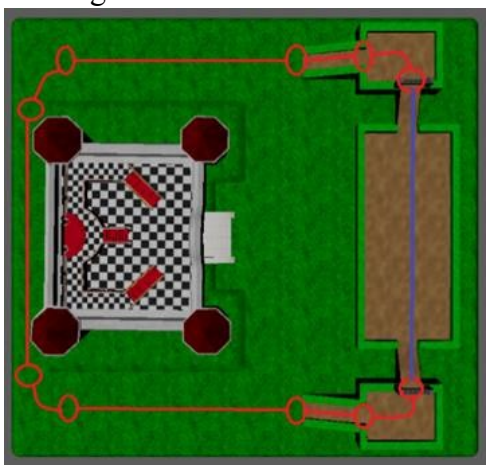
three Koopas will be added in front of Puro Huesos. Each Koopa will be rotating their body and waving their arms happily, simulating that they are waiting for their burger and generating a customer atmosphere for Puro Huesos.

To achieve this, each Koopa will be built in a hierarchical fashion, so that the shell and arms can rotate and generate fluid, joyful movement. A sine function cycle will be used to control these movements, giving them a sense of joy and excitement as they wait for their burger. This addition of the Koopas creates an ambience

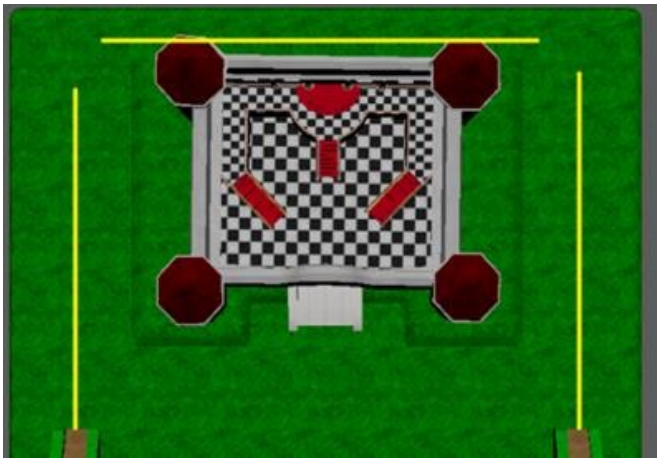
lively and festive on the scene, bringing life and dynamism to the Puro Huesos environment as he prepares some burgers.

Driving a go-kart

In this animation, several karts will be driving around a circuit around the castle. Each kart will perform different actions, such as going around the corresponding curves, going up and down ramps in the garden, and flying with gliders over the garden, using elements characteristic of the Mario Kart 8 games.

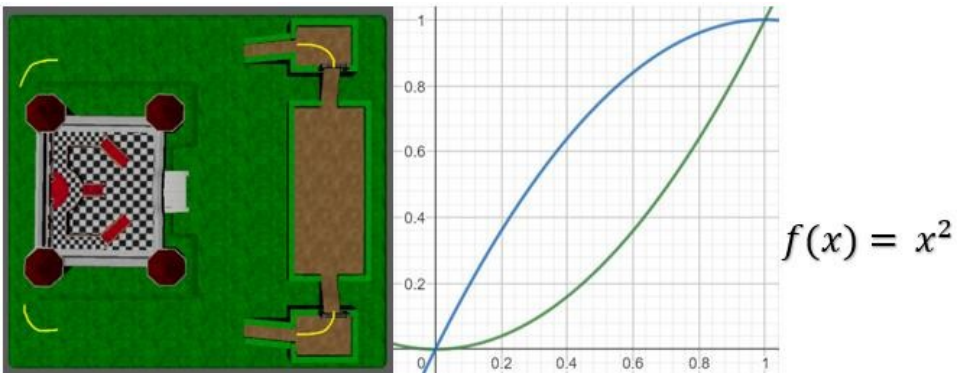


This animation is the most complex, as a 10-flag system has been developed that will allow the karts to traverse the entire circuit smoothly. To achieve this, three main routes have been established which consist of increasing or decreasing the X and Z values for the straight sections of the circuit.



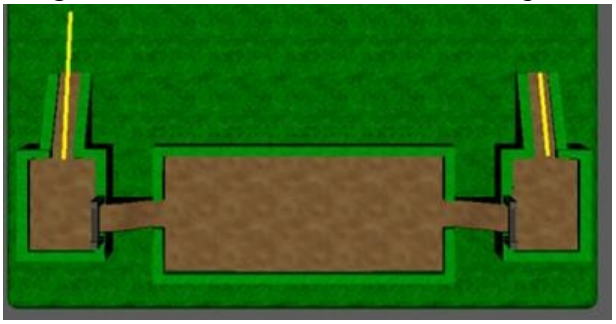
$$f(x) = 1 - (x - 1)^2$$

Four flags will be used to make smooth turns on the circuit. For this, the property of quadratic functions has been employed, using the formula x^2 and 1-(x-1)^2. These functions are applied to the X and Z coordinates, respectively, allowing the kart to gradually decrease its speed when approaching a flag, and then gradually increase its speed after passing it, thus achieving smooth and realistic curves.

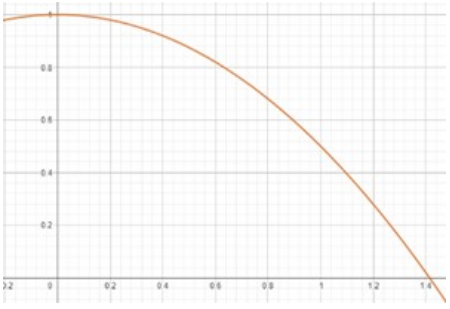


During these speed and direction changes, the kart's rotation will be adjusted to simulate the turns it makes on the track, creating a more authentic driving feel.

In addition, special flags have been added to simulate climbs and descents on the track. When approaching a climb flag, the Z value will be increased while also increasing the Y coordinate to simulate the kart going up a ramp in the garden. At the end of the ramp, the rotation will be adjusted again to indicate that the kart has completed the climb.

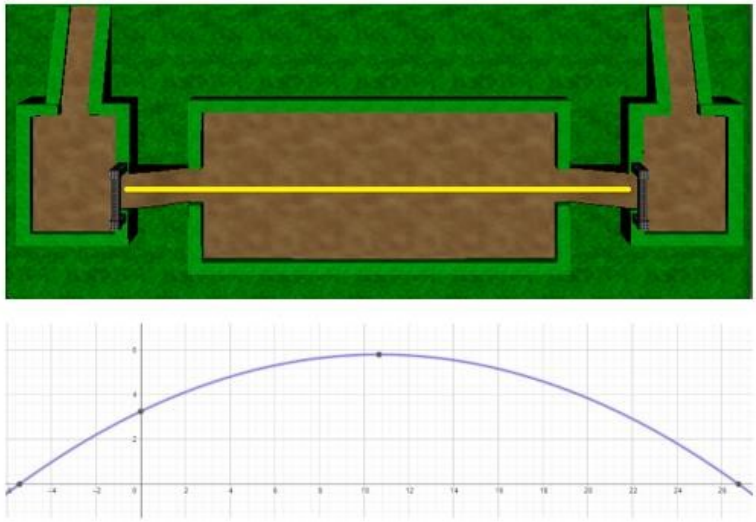


On the other hand, when approaching a downhill flag, it was chosen to make the kart descend while continuing to move forward rapidly in the Z-coordinate and descending in the Y-coordinate using the free fall formula under the influence of gravity. During this descent, the kart will rotate slightly to simulate a suspension in the air and, upon reaching Y = 0, the initial positions will be restored to simulate the impact with the ground and continue the ride.



The last flag will be used to simulate the use of gliders in Mario Kart 7 and 8. The kart will perform a 380-degree turn to simulate a stunt and then open its parachute to fly. During the flight, the kart will move along the X-axis, while the Y-axis will follow a parabolic trajectory to simulate the sensation of flight. At the end of the flight, additional degrees will be adjusted to complement the kart's landing effect.

It is also mentioned that in one of the corner flags the scale of the parachute is added to 0 to simulate that it is stored in the kart.



$$f(x) = 2.283 - 0.02233919\,x^2 + 0.47640569\,x + 0.96004303$$

To liven up the environment, a total of four karts have been added to the circuit. These karts represent characters from different franchises, such as Mulan, SpongeBob SquarePants, Goku and Ladybug. Each kart has been built hierarchically to allow the wheels to turn freely and follow the position of the kart. Similarly, the parachute will adjust at the rear of the corresponding kart at the appropriate time, creating a dynamic and realistic visual effect.

With all these actions combined, a complex and exciting animation will be achieved, in which the karts go around the circuit around the castle, performing movements and actions typical of the Mario Kart 7 and 8 games.

Keyframe animation

The master sword

When the Master Sword animation is activated, a series of actions will be performed to bring this effect to life. First, a SpotLight will light up, illuminating both the sword and the stone in which it is stuck. This light will highlight the details of the sword and create a mysterious atmosphere around the stone. The sword will then rise from its initial position to start a circle around the stone. During this path, the sword will rotate on its Y-axis to add a dynamic and eye-catching visual effect. Once the sword has completed its path and has returned to its original position, it will gradually descend and then plunge back into the stone. This action adds a dramatic element to the animation, simulating the return of the sword to its resting place. Finally, at the end of the animation, the SpotLight illuminating the sword and the stone will be switched off, which will contribute to create a mysterious and enigmatic atmosphere in the scene.

To achieve this, the KeyFrames are stored in a text file. When the program is executed, it starts by reading the text file using an implemented function:

```
void readFile() {   // funcion para leer archivo que contiene los keyframes
    int i = 0, linead = 0;
    std::string indice, valor, content;
    int index;
    float value;
    std::ifstream fileStream("KeyFrames.txt", std::ios::in);

    if (!fileStream.is_open()) {
        printf("Failed to read %s! File doesn't exist.", "KeyFrames.txt");
        content = "";
    }
```

Within this function, each line of the file is read and the value for each axis in position and rotation is extracted:

```
std::string line = "";
while (!fileStream.eof()) {
    std::getline(fileStream, line);

    if (linead < 10) indice = line.substr(9, 1);
    else indice = line.substr(9, 2);
    index = std::stoi(indice);
    switch (i) {
    case 0: //posicion en X
        if (linead < 10) valor = line.substr(19, line.size() - 21);
        else valor = line.substr(20, line.size() - 22);
        value = std::stof(valor);
        KeyFrame[index].posX = value;
        std::cout << "KeyFrame[" << index << "].posX = " << value << ";" << std::endl;
        i++;
        break;
```

The Switch iterates over 4 cases: when reading the position in X, Y, Z or the rotation of the object, care is taken to correctly extract each substring. This is achieved with the help of the index variables, and the i counter.

Within the code there were already functions to perform linear interpolation for the animation, and to reset the variables to the values of the first frame.

Finally, the activation of the animation with 1 and the reset of the animation with 0 were added.

```
if (keys[GLFW_KEY_1]) {
    if (reproduciranimacion < 1) {
        if (play == false && (FrameIndex > 1)) {
            espadaLuz = 1.0f;
            resetElements();
            //First Interpolation
            interpolation();
            play = true;
            playIndex = 0;
            i_curr_steps = 0;
            reproduciranimacion++;
            printf("\n presiona 0 para habilitar reproducir de nuevo la animación'\n");
            habilitaranimacion = 0;
        }
        else play = false;
    }
}
if (keys[GLFW_KEY_0]) {
    if (habilitaranimacion < 1) reproduciranimacion = 0;
}
```

SpotLigth linked to animation

To make the animation look better, the first Spot Light declared is activated when the KeyFrame animation starts and remains activated until the animation ends.

This is seen in the variable swordLight, which takes the value of 1 to be assigned as the ambient and diffuse component of the light.
Within the animateKeyFrame(void) function, when the last Frame is reached, the variable swordLight will be set to 0.

```
void animateKeyFrame(void) {
    //Movimiento del objeto
    if (play) {
        if (i_curr_steps >= i_max_steps) {//end of animation between frames?
            playIndex++;
            printf("playindex : %d\n", playIndex);
            if (playIndex > FrameIndex - 2) {//end of total animation?
                printf("Frame index= %d\n", FrameIndex);
                printf("termina anim\n");
                espadaLuz = 0.0f;
                playIndex = 0;
                play = false;
            }
            else {//Next frame interpolations
                i_curr_steps = 0; //Reset counter
                interpolation();
            }
        }
```

Dictionary of variables for animations

Functions	Explanation
main	The main function is the entry point to a C++ program, the code establishes the environment and initial settings for a graphics program using GLFW and OpenGL. A window is created, libraries are initialised and shaders and 3D models are loaded. Then, it enters a main loop where rendering and interaction with the user.
animation	This function declares all the conditions that are present in each object to perform an action when activating or deactivating a variable.
KeyCallback	This function defines that when pressing the declared key within the function will perform the corresponding action defined in each step.
MouseCallback	The way in which the mouse will act when inside the window and the actions it will take when doing any of the following are coded. interaction with it.
DoMovement	This code block defines how the code is to act. in case a key is pressed, and as the name says, it will generate movement.

Variable	Type	Functionality
LaHora	float	Counter used to measure the duration of the day and night during the execution.
StatusDay	bool	Flag that indicates to the programme in which state of the day the place is located (day or night).
PointlightTrue	float	Multiplier that will help to switch on and off the pointlights.
environmental1 environmental2	float	Values that help change the colour used in the models to make them look like they are in the daytime or in the evening. at night.
directionalEnvironmental	float	Variable that helps to set during the day or remove during the night the directional of the ambient to greater realism.
PosteLuzRG PosteLuzB	float	Values that help to change the colour used in the light pole models to make them look like on or off.
swordLight	float	Variable that helps to illuminate the sword blade during the day.

Reflectors	float	Variable that increases or decreases the illumination of the reflectors depending on whether they are switched on or off.
VariableRadioTurns VariableRadioTurns VariableRadioBrincos	float	Auxiliary variables that will constantly increase and help the movement of chillywilly and chain chomp.
TranslationXChilly TranslationZChilly	float	Calculate the X and Z position of chilly Willy's car using the circumference formula.
TranslationXChainChomp TranslationZChainChomp	float	Same functionality, but for the chainchomp with its own variable.
TraslacionYChainChomp	float	Calculate the height of the chain chomp hops using the sine function.
VariableSet VariableHeadLuigi	float	Auxiliary variables that will help Luigi move his arms and head at different rates.
MovementSetting MovementHeadLuigi	float	They calculate the rotation of Luigi's arms and head at different rates.
VariablePeachTurns	float	Spin variable that helps to calculate the position and spin of Peach
TranslationXPeach TranslationZPeach	float	Calculate the X and Z position for Peach to rotate in an oval inside the coins.
VariableCurrenciesDrafts	float	Variable that helps to calculate the position of the coins and their rotation.
TraslacionXCoins TraslacionZCoins TraslacionXCoins1 TraslacionZCoins1 TraslacionXCoins2 TraslacionZCoins2 TraslacionXCoins2 TraslacionXCoins3 TraslacionZCoins3 TraslacionXCoins4 TraslacionZCoins4 TraslacionZCoins4 TraslacionXCoins5 TraslacionZCoins5 TraslacionXCoins5 TraslacionXCoins5 TraslacionXCoins6 TraslacionZCoins6 TraslacionXCoins7 TranslacionZMonedas7	float	It calculates the position of each coin so that each one has its own position, but follows the same route, thus creating a ring made of coins that go around Peach in the same route.

RotCaparazon	float	Incrementable and decrementable that controls the direction of movement of koopas.
senseHeart	bool	Determines the direction and limit of rotation of the koopas for their movement.
CheepFloat	float	Auxiliary variable that will increase or decrease the height of the cheep cheep.
HeightNadado	float	Calculate the Y position of the cheep cheep for your swim inside the bag.
FinsCheep	float	Variable that increments and decerments to rotate the cheep cheep flaps.
IDLE	float	Variable incrementable y decrementable that controls all static animations.
senseIDLE	bool	Flag controlling the idle animation direction
RotBrazo	float	Variable incrementable y decrementable that controls the rotation of the arms of all.
senseArm	bool	Flag controlling the direction of rotation of the arms of the all
Giro	float	Calculates the position of a toad to rotate using the property of the logarithmic function.
Brinco	float	Variable that controls the small height that the toad gains when it performs its animation of moving the toads. arms.
auxiliar y1 auxiliar y2 auxiliar y3	float	Incrementable/decrementable variables that support certain animations with their number intervals, each at its own pace.
Sense1	bool	Flag controlling the direction of auxiliary 1.
Sense2	bool	Flag controlling the direction of the auxiliary 2.
RoatParatroopa	float	Variable that controls the rotation of the paratrooper to give direction to whether it enters or exits the frame.
FlightParatroopa	float	Controls the position of the paratrooper as it makes its way in and out of the paint.
Rotation2	float	Variable that increases cyclically to control the rotations of the propellers present in the diorama.
Times	float	Variable that helps the animation shader to create the swell effect in the painting of the castle.
Reduce	float	Variable that controls the movement of the swell so that it only moves when the paratrooper comes into contact with the frame.

Hamburger	bool	Flag indicating the start of the hamburger launch animation.
Wait for	float	Counter that will indicate when to advance to the next stage of the purohuesos animation.
SentidoHamburger	bool	Flag indicating the direction of the purobone arm holding the spatula.
roatAnteBrazo	float	Variable that controls the rotation of its arm.
FlightTimeBurger	float	Auxiliary variable measuring the time of flight of the burger.
HamburFlight	float	Value that calculates the height of the burger using the vertical draft formula.
AnimationGoomba	bool	Flag controlling the Goomba walk animation.
WalkGoomba	float	Incrementable/decrementable variable that controls the distance the Goomba travels.
SentidoGoomba	bool	Flag that controls the direction in which the goomba moves.
rotHeadGoomba rotFootDerGoomba rotFootIzqGoomba	float	Incrementable/decrementable variables controlling the rotations of each part of the goomba's body while walking
RotSentidoGoomba	bool	Controls the direction in which the goomba rotates its feet and head.
RoatFlyGuy TransferFlyguy	float	Variables that control the translation and tilt of the flyguy as it flies.
SentidoFlyGuy	bool	Flag controlling the flyguy's direction of flight
RoatPump RoatWrench RoatPumpIzqPump pieDerPump	float	Variables that control each part of the bobomba to rotate the head and key while moving the feet to simulate the bob's characteristic gait. omb.
TiempoanimaiconBomba	float	Incrementable that controls the duration of the bobble run.
MarchaBomba	Bool	Flag that controls which foot goes up and which foot comes down to simulate the bob omba's gait.
AnimacionBomba	Bool	Flag indicating when to activate the pump animation.
piranaPlant	float	Incrementable/decrementable variable that moves up or down the piranha plant to simulate entering and exiting its tube.
PiranaplantSense	bool	Determines the direction in which the plant is directed.
timeBite	float	A counter that measures the time you hold your mouth in one position.

AnguloMorida	float	Variable that changes the angle of the mouth from 0 to 45 degrees, which depends on the counter before mentioned.
AnimPlantaPriaña	bool	Flag that controls when the piranha plant animation is activated.
LadyBugmovKitX LadyBugmovKitZ LadyBugmovKitY GokumovKitX GokumovKitZ GokumovKitY BobEsponjamovKitX BobEsponjamovKitZ BobEsponjamovKitZ BobEsponjamovKitY MulanmovKitX MulanmovKitZ MulanmovKitY	float	Variables that control the X,Y and Z positions of each of the karts, which can be incrementable/decrementable or formulas. To identify which value corresponds to which, the character name is used at the beginning of each variable.
LadyBugrotKit		
LadyBugGyroExtreme GokurotKit GokuGyroExtreme BobEsponjarotKit BobEsponjaGyroExtremeM ulanrotKit MulanGyroExtreme	float	Variables for each racer that control the degree of turning on the Y and X axes to give more realism to the handling of the cars.
LadyBugDeltaTime GokuDeltaTime SpongeBobSeltaTime MulanDeltaTime	float	Auxiliary used in curves to make turns more natural by switching them to two different formulas, which will reduce the speed on one axis and increase it on the other.
LadyBugrun1 LadyBugrun2 LadyBugrun3 LadyBugrun3Up LadyBugrun3Up LadyBugrun4 LadyBugrun5 LadyBugrun5 LadyBugrun6 LadyBugrun6Down LadyBugrun6Down LadyBugrecorrido7 LadyBugrecorrido8	bool	Flags that mark the end and start of a course, which together make up the complete circuit, each of which handles a different logic for the corresponding car to perform certain actions before moving on to the next course. The reason why each character has its own track, but using the same base, is so that they initialise at different points and do exactly the same thing. action to give more dynamism and realism to the scenario.
LadyBugaOpening LadyBugClose GokuaOpening GokuClose SpongeBobOpening BobSpongeClose	bool	Flags indicating when each character's parachutes are scaled to 1 or 0 at times when the chariot is suspended in the air and lands back on the ground.

LadyBugXscale LadyBugYscale LadyBugZscale GokuXscale GokuYscale GokuZscale SpongeBobXscale SpongeBobXscale SpongeBobYscale SpongeBobYscale MulanXscale MulanYscale MulanYscale MulanZscale	float	Variables that change the scale of each parachute, which will scale from 0 to 1 and vice versa depending on the character's flag.
LadyBugDescend GokuDescend SpongeBobDescend MulanDescend	bool	Flag that activates the decrement function at its rotation to achieve the effect of angle change due to the descent of the flight.
Rotation1	float	Variable that is constantly increasing to rotate the wheels.
SentidoViento	bool	Flag indicating the direction in which the wind is moving.
Wind	float	Rapidly increasing and decreasing variable, which causes the parachutes to move from side to side simulating that a wind is present.
Door	bool	Flag indicating in which state the door is in, whether closed or open.
AnimGate	bool	Flag indicating whether the animation is active and move the doors depending on their status.
RotGate	float	They indicate the angle at which the doors are to be opened or closed.
AnimCubo	bool	Flag indicating whether the question block animation is activated or not.
StatusCube	bool	Flag indicating whether the yellow textured cube or the dark textured cube is used.
EscalaCubo EscalaVacia	float	Variables that scale the yellow and dark cubes to show one and hide the other.
RoatChampi	float	Variable that controls the angle of rotation of the mushroom when it is in the air.
TiroVertical	float	Formula that calculates the Y-position of the mushroom to achieve the vertical throw effect.
Strike	float	Variable that moves up and down a little to simulate the little jump the cube makes when you hit it.

time	float	Auxiliary variable measuring flight time, which is used to calculate mushroom height.
FlorBrinco	float	It calculates the position of the flower depending on its rotation in order to simulate the jumps the flower makes.
FlorGiro	float	Variable incrementable y decrementadle that determines the angle of rotation of the flower.
FlowerSense	bool	Flag indicating direction of rotation
FlightHeight	float	Formula that calculates by the use of sines the mario Tanooki's height position with an auxiliary variable.
Tanooki	float	Steadily increasing auxiliary variable to support the calculation of Mario Tanooki's height

Audio

irrKlang is a sound engine and a high-level audio library that offers numerous advantages for the implementation of audio in your Computer Graphics project. Due to previous research it was decided to use this library. So we implemented in the code the library loading, the declaration of variables that will contain the implemented audios and we placed their playback without loop inside the while, the day or night music will be played in the while.

will be in the day state change logic, in which apart from changing the lighting values, it will play the music corresponding to the state change. While Bowser's roar will only be when the R key is pressed and Bowser's animation is turned off, so that he can turn it on and play the audio to bring the scene to life.