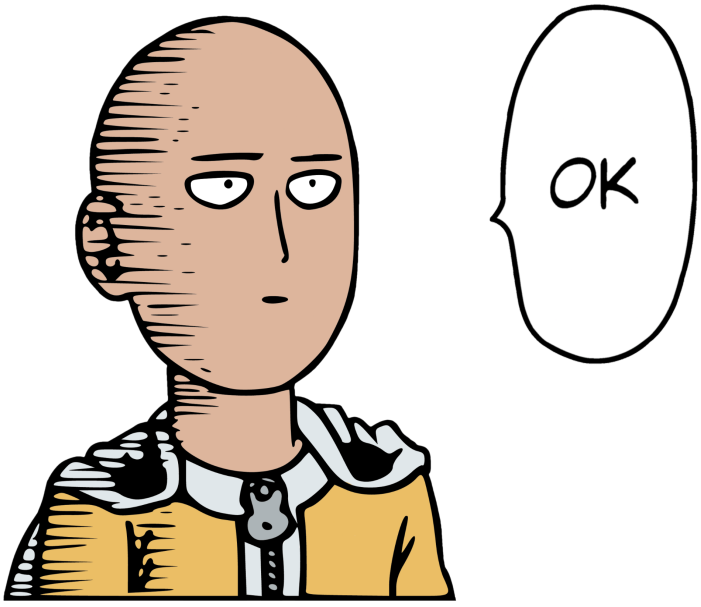
Universidad Nacional Autónoma de México

Engineering Faculty

Saitama's House Simulation



Developer:

Mario Alberto Vásquez Cancino

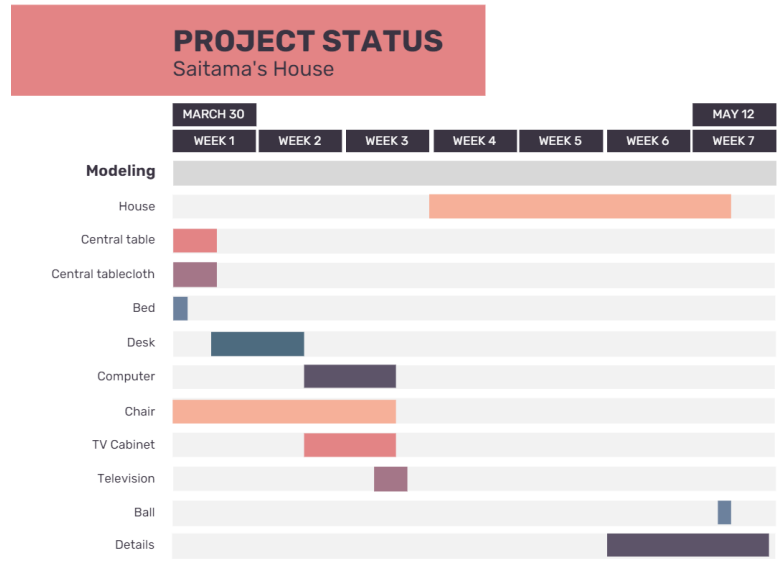
v. 1.0.0

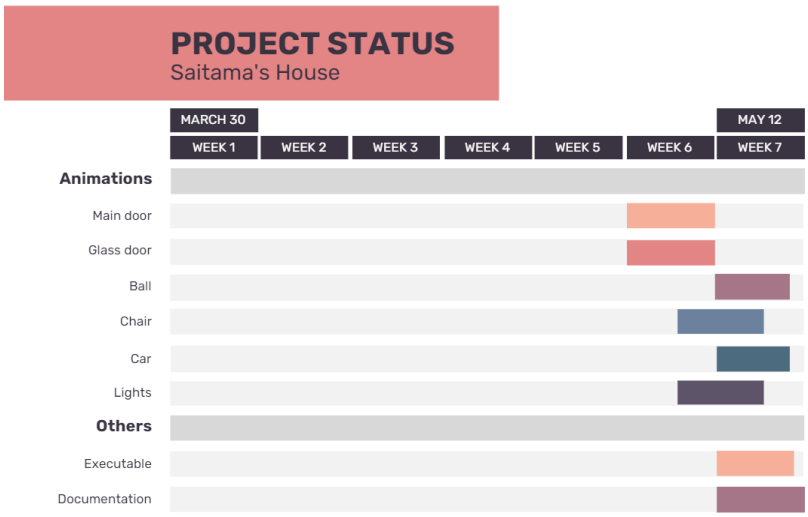
Index

Objective

To offer a virtual tour in a simple recreation of saitama's house, the main character of one punch man.

Work Plan





Project Scope

At the end of this project, a program should be obtained that allows the user to walk through a simple virtual recreation of Saitama's house, the main character of the One Punch Man series.

The internal virtual recreation of the room should be as similar as possible to the original, it should have at least 7 elements of the room, the selected elements were the following:

- Central table
- Central tablecloth
- Chair
- Desk
- Computer
- Bed
- Television
- TV cabinet

Room reference image

The exterior is different from the original in order to make the tour more enjoyable.

Ireference image of the outside

There will also be 5 animations, which are:

- Main door opening and closing.
- Glass door opening and closing.
- Ball falling from the central table.
- Chair moving as if someone had stood up.
- Car moving outside the house.

In order for the user to interact with the program, the user must move around with the W, S, A and D keys, move the view with the mouse, stop and play the animations with the space bar and control the room's internal light.

The completed project with its documentation, executable file and binaries must be delivered by Thursday, May 12, 2022.

Project Limitations

Due to the fact that it is a recreation, the development time and the resources available in the room, the animations and modeled elements will not have a 100% resemblance with respect to the originals, but it will be possible to have a 90% resemblance with what was shown in the series.

Operating

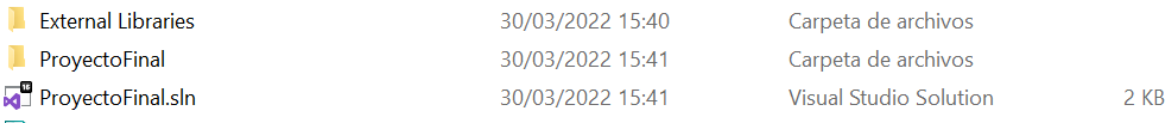
Installation & Execution

To run the simulation see the User's Manual.

Source Code

Structure

The project was made with Visual Studio so in the root folder we found the following main elements:

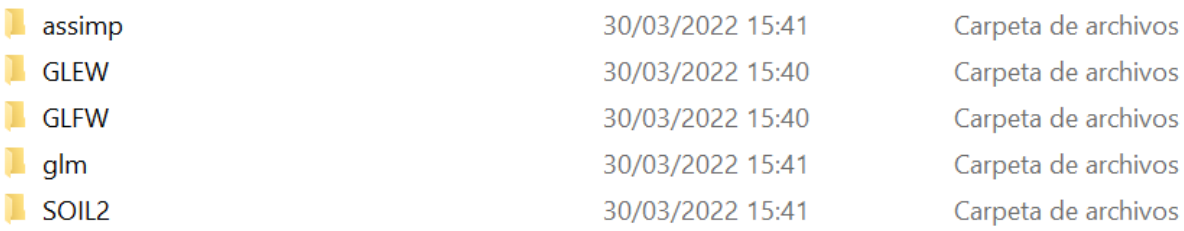


ProyectoFinal.sln

Visual Studio solution to manage the project.

External Libraries

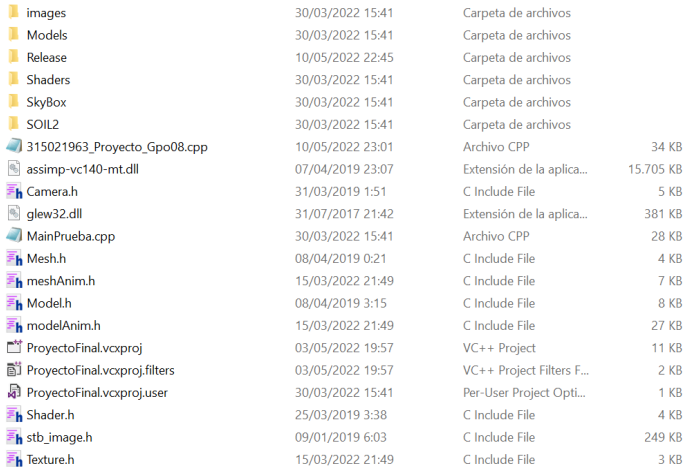
Libraries used in the project.



- **ASSIMP:** Library that will be used to load 3D models or scenes stored in a variety of formats.
- **GLEW:** It is an open source multiplatform library written in C/C++ which will provide us with a runtime mechanism to determine which of the opengl functions are supported in the target platform.
- **GLFW:** Library that will allow us to create and manage the windows where we display OpenGL graphics.
- **GLM:** Library provides access to mathematical functions commonly used in 3D graphics.
- **SOIL:** It is a small library to load images in 6 types of formats.

ProyectoFinal

Folder containing the main files and resources of the project.



- **images:** Folder of images used in the project.
- **Models:** Folder with all the models used in the project.
- **Shaders:** Folder that contains all the shaders for the light, animations, etc.
- **Skybox:** Folder with the images for the Skybox.
- **SOIL2:** Library to load images in 6 types of formats.
- **315021963_Project_Gpo08.cpp:** Main code of the project.
- **assimp-vc140-mt.dll:** File to manage the ASSIMP library.
- **Camera.h:** Header file to control the camera.
- **glew32.dll:** The OpenGL Extension Wrangler Library.
- **Mesh.h and meshAnim.h:** Header files to handle the meshes for the model textures.
- **Model.h and modelAnim.h:** Header files to handle the models together with their meshes and textures.
- **Shader.h:** Header file to handle the shaders.
- **stb_image.h:** Library to load images made by Sean Barret.
- **Texture.h:** Header file to load SkyBox textures.

Code

Opening the main project file (315021963_Project_Gpo08.cpp) we find:

Import of the libraries to be used.

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h" /*SKYBOX*/
```

Prototypes of the functions and global variables to occupy.

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void animacion();
void animPuertas();
void animKeyFrame();
void animacionCarro();

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(0.0f, 1.0f, 4.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;

// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni(-6.13f, -0.5f, 4.55f);
glm::vec3 PosIniPel(2.0f, 1.12f, 0.0f);
glm::vec3 PosIniCar(-43.0f, -0.4f, 25.0f);
bool active = true;
bool luces = false;

// Variables animaciones
float rotPuerta = 0;
bool abiertaPuerta = false;
float movCristal = 0;
bool abiertaCristal = false;

//Animación del coche
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;

bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
```

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,5.0f,0.0f)
};

glm::vec3 Light1 = glm::vec3(0);

// Deltatime
GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
GLfloat lastFrame = 0.0f;   // Time of last frame

// Keyframes
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotSilla = 0;
float posXPel = PosIniPel.x, posYPel = PosIniPel.y, posZPel = PosIniPel.z, rotPel = 0;

#define MAX_FRAMES 9
int i_max_steps = 50;
int i_curr_steps = 0;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotSilla;
    float rotInc;

    float posXPel;      //Variable para PosicionX
    float posYPel;      //Variable para PosicionY
    float posZPel;      //Variable para PosicionZ
    float incXPel;      //Variable para IncrementoX
    float incYPel;      //Variable para IncrementoY
    float incZPel;      //Variable para IncrementoZ
    float rotPel;
    float rotIncPel;

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 2;         //introducir datos
bool play = false;
int playIndex = 0;
```

```
void resetElements(void)
{
    posX = KeyFrame[0].posX;
    posY = KeyFrame[0].posY;
    posZ = KeyFrame[0].posZ;

    rotSilla = KeyFrame[0].rotSilla;

    posXPel = KeyFrame[0].posXPel;
    posYPel = KeyFrame[0].posYPel;
    posZPel = KeyFrame[0].posZPel;

    rotPel = KeyFrame[0].rotPel;

}

void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps;

    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotSilla - KeyFrame[playIndex].rotSilla) / i_max_steps;

    KeyFrame[playIndex].incXPel = (KeyFrame[playIndex + 1].posXPel - KeyFrame[playIndex].posXPel) / i_max_steps;
    KeyFrame[playIndex].incYPel = (KeyFrame[playIndex + 1].posYPel - KeyFrame[playIndex].posYPel) / i_max_steps;
    KeyFrame[playIndex].incZPel = (KeyFrame[playIndex + 1].posZPel - KeyFrame[playIndex].posZPel) / i_max_steps;

    KeyFrame[playIndex].rotIncPel = (KeyFrame[playIndex + 1].rotPel - KeyFrame[playIndex].rotPel) / i_max_steps;

}
```

In the main function we find:

GLFW initialization, OpenGL options and window creation.

```
// Init GLFW
glfwInit();

// Create a GLFWwindow object that we can use for GLFW's functions
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Saitama's House", nullptr, nullptr);

if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}

glfwMakeContextCurrent(window);

glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

// Set the required callback functions
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

// GLFW Options
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

// Set this to true so GLEW knows to use a modern approach to retrieving function pointers and extensions
glewExperimental = GL_TRUE;
// Initialize GLEW to setup the OpenGL Function pointers
if (GLEW_OK != glewInit())
{
    std::cout << "Failed to initialize GLEW" << std::endl;
    return EXIT_FAILURE;
}

// Define the viewport dimensions
glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

// OpenGL options
glEnable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Import models.

```
// Setup and compile our shaders
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");

Model Casa ((char*)"Models/Casa/Casa.obj");
Model puerta ((char*)"Models/Casa/Puerta2.obj");
Model marcoVen ((char*)"Models/Casa/MarcoVen.obj");
Model cristalVen ((char*)"Models/Casa/CristalVen.obj");
Model puertaCrisA ((char*)"Models/Casa/PuertaCrisA.obj");
Model puertaMarA ((char*)"Models/Casa/PuertaMarcA.obj");
Model puertaCrisC ((char*)"Models/Casa/PuertaCrisC.obj");
Model puertaMarC ((char*)"Models/Casa/PuertaMarcC.obj");
Model cerca((char*)"Models/Casa/fenceFinal.obj");

Model mesa ((char*)"Models/Mesa/Mesa.obj");
Model tapete ((char*)"Models/Tapete/Tapete.obj");
Model escritorio ((char*)"Models/Escritorio/Escritorio.obj");
Model silla ((char*)"Models/Silla/Silla.obj");
Model laptop ((char*)"Models/Laptop/Laptop.obj");
Model cama ((char*)"Models/Cama/Cama.obj");
Model muebleTV ((char*)"Models/MuebleTV/Cabinet.obj");
Model tv ((char*)"Models/TV/TV.obj");
Model controlTV ((char*)"Models/TV/Control.obj");
Model pelota ((char*)"Models/Pelota/Pelota.obj");

Model carro ((char*)"Models/Carro/Carro.obj");
```

Initialization of KeyFrames for some animations.

```
//Inicialización de KeyFrames
for (int i = 0; i < MAX_FRAMES; i++)
{
    KeyFrame[i].posX = 0;
    KeyFrame[i].incX = 0;
    KeyFrame[i].incY = 0;
    KeyFrame[i].incZ = 0;
    KeyFrame[i].rotSilla = 0;
    KeyFrame[i].rotInc = 0;

    KeyFrame[i].posXPel = 0;
    KeyFrame[i].incXPel = 0;
    KeyFrame[i].incYPel = 0;
    KeyFrame[i].incZPel = 0;
    KeyFrame[i].rotPel = 0;
    KeyFrame[i].rotIncPel = 0;
}

// Guargando animacion KEYFRAMES
KeyFrame[0].posX = posX;
KeyFrame[0].posY = posY;
KeyFrame[0].posZ = posZ;
KeyFrame[0].rotSilla = rotSilla;

KeyFrame[0].posXPel = posXPel;
KeyFrame[0].posYPel = posYPel;
KeyFrame[0].posZPel = posZPel;
KeyFrame[0].rotPel = rotPel;


// Guargando animacion KEYFRAMES
KeyFrame[1].posX = -4.13f;
KeyFrame[1].posY = -0.5f;
KeyFrame[1].posZ = 3.0f;
KeyFrame[1].rotSilla = 135.0f;

KeyFrame[1].posXPel = 2.4f;
KeyFrame[1].posYPel = 1.02f;
KeyFrame[1].posZPel = 0.0f;
KeyFrame[1].rotPel = -30.0f;
```

Lightning Shader for lights and SKyBox initialization.

```
// Set texture units
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);

/*SKYBOX*/
GLfloat skyboxVertices[] = {
    // Positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    -1.0f,  1.0f, -1.0f,
     1.0f,  1.0f, -1.0f,
     1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
     1.0f, -1.0f,  1.0f
};


GLuint indices[] =
{  // Note that we start from 0!
    0,1,2,3,
    4,5,6,7,
    8,9,10,11,
    12,13,14,15,
    16,17,18,19,
    20,21,22,23,
    24,25,26,27,
    28,29,30,31,
    32,33,34,35
};

// Positions all containers
glm::vec3 cubePositions[] = {
    glm::vec3(0.0f,  0.0f,  0.0f),
    glm::vec3(2.0f,  5.0f, -15.0f),
    glm::vec3(-1.5f, -2.2f, -2.5f),
    glm::vec3(-3.8f, -2.0f, -12.3f),
    glm::vec3(2.4f, -0.4f, -3.5f),
    glm::vec3(-1.7f,  3.0f, -7.5f),
    glm::vec3(1.3f, -2.0f, -2.5f),
    glm::vec3(1.5f,  2.0f, -2.5f),
    glm::vec3(1.5f,  0.2f, -1.5f),
    glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

Loading of SkyBox images with the VAO and EBO.

```
// First, set the container's VAO (and VBO)
GLuint VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);
/*glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);*/

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
// Normals attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
// Texture Coordinate attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glBindVertexArray(0);

// Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same for the light object (also a 3D cube))
GLuint lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO's data already contains all we need.
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// Set the vertex attributes (only position data for the lamp))
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0); // Note that we skip over the other data in our buffer object (we don't need the normals/textures, only positions).
glEnableVertexAttribArray(0);
glBindVertexArray(0);


//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.jpg");
faces.push_back("SkyBox/left.jpg");
faces.push_back("SkyBox/top.jpg");
faces.push_back("SkyBox/bottom.jpg");
faces.push_back("SkyBox/back.jpg");
faces.push_back("SkyBox/front.jpg");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
/*SKYBOX*/
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
```

In main we will find GAME LOOP which is a cycle that will repeat until the window is closed in order to refresh the image and update the changes made, in short to be able to move in the simulation, see the lights and animations.

Setting the ambient light, the light in the camera and the internal light of the house.

```
// Calculate deltatime of current frame
GLfloat currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;

// Check if any events have been activiated (key pressed, mouse moved etc.) and call corresponding response functions
glfwPollEvents();
DoMovement();
animacion();

// Clear the colorbuffer
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//OPTENGL
// Depth options
glEnable(GL_DEPTH_TEST);
//OPTENGL


//Load Model
// Use corresponding shader when setting uniforms/drawing objects
lightingShader.Use();
GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);


// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);


// Point light 1
glm::vec3 lightColor;
lightColor.x = Light1.x;
lightColor.y = Light1.y;
lightColor.z =  Light1.z;

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.0orf);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.075f);

// SpotLight
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), camera.GetFront().x, camera.GetFront().y, camera.GetFront().z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"),1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.12f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.0orf);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));


// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);

// Create camera transformations
glm::mat4 view;
view = camera.GetViewMatrix();

// Get the uniform locations
GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

// Pass the matrices to the shader
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));


glBindVertexArray(VAO);
```

The models are drawn in their specific position and with the necessary configurations for them to be transparent or not and if they are going to be moving or fixed.

```
glm::mat4 model(1);

//Carga de modelo
view = camera.GetViewMatrix();

// ******************** CASA ********************
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Casa.Draw(lightingShader);

// Puerta Entrada
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-7.16f, 3.05f, 9.6f));
model = glm::rotate(model, glm::radians(rotPuerta),glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puerta.Draw(lightingShader);

// Ventana
// Marco
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
marcoVen.Draw(lightingShader);
// Cristal
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 0.5);
cristalVen.Draw(lightingShader);
glDisable(GL_BLEND);  //Desactiva el canal alfa
// Para que no cambie el color el canal alpha a los demas objetos
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);

// Puerta de Cristal
// Marco que se queda cerrado
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertaMarc.Draw(lightingShader);
// Cristal que se queda cerrado
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 0.5);
puertaCrisC.Draw(lightingShader);
glDisable(GL_BLEND);  //Desactiva el canal alfa
// Para que no cambie el color el canal alpha a los demas objetos
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
// Marco que se abre
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::translate(model, glm::vec3(newCristal, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertaMarA.Draw(lightingShader);
// Cristal que se abre
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::translate(model, glm::vec3(newCristal, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 0.5);
puertaCrisA.Draw(lightingShader);
glDisable(GL_BLEND);  //Desactiva el canal alfa
// Para que no cambie el color el canal alpha a los demas objetos
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
// ******************** CASA FIN ********************

// Cerca
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
cerca.Draw(lightingShader);
```

```
// Mesa
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(lightingShader);
// Pelota
model = glm::mat4(1);
//model = glm::translate(model, glm::vec3(0.025f, 1.02f, 0.0f));
model = glm::rotate(model, glm::radians(rotPel), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(posXPel, posYPel, posZPel));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
pelota.Draw(lightingShader);
// tapete
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
tapete.Draw(lightingShader);
// Escritorio
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
escritorio.Draw(lightingShader);
// Silla
model = glm::mat4(1);
//model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(rotSilla), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
silla.Draw(lightingShader);
// Laptop
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
laptop.Draw(lightingShader);
// Cama
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
cama.Draw(lightingShader);
// Mueble TV
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
muebleTV.Draw(lightingShader);
// TV
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
tv.Draw(lightingShader);
// Control TV
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
controlTV.Draw(lightingShader);

// Carro
model = glm::mat4(1);
model = glm::translate(model, PosIniCar + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
carro.Draw(lightingShader);


glBindVertexArray(0);


// Also draw the lamp object, again binding the appropriate shader
lampShader.Use();
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");
```

Closing busy buffers.

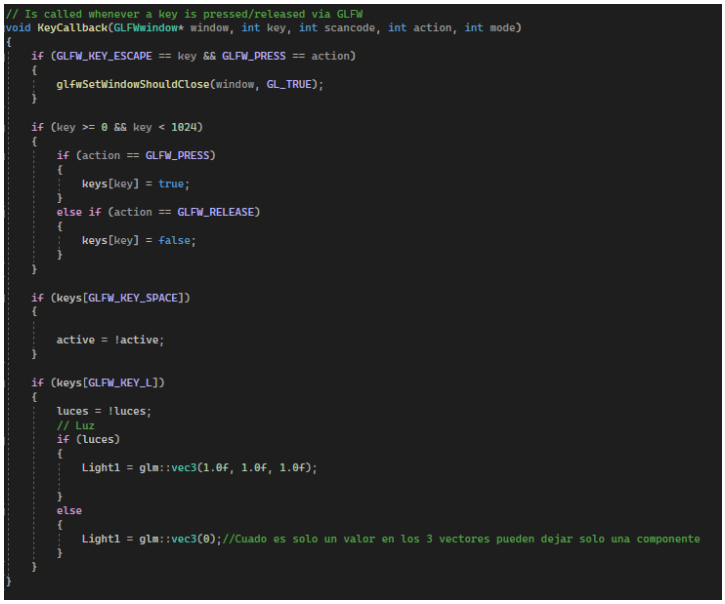
```
    glBindVertexArray(0);


    // Also draw the lamp object, again binding the appropriate shader
    lampShader.Use();
    // Get location objects for the matrices on the lamp shader (these could be different on a different shader)
    modelLoc = glGetUniformLocation(lampShader.Program, "model");
    viewLoc = glGetUniformLocation(lampShader.Program, "view");
    projLoc = glGetUniformLocation(lampShader.Program, "projection");


    /*SKYBOX*/
    // Also draw the lamp object, again binding the appropriate shader
    lampShader.Use();
    // Get location objects for the matrices on the lamp shader (these could be different on a different shader)
    modelLoc = glGetUniformLocation(lampShader.Program, "model");
    viewLoc = glGetUniformLocation(lampShader.Program, "view");
    projLoc = glGetUniformLocation(lampShader.Program, "projection");


    // Draw skybox as last
    glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes when values are equal to depth buffer's content
    SkyBoxshader.Use();
    view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any translation component of the view matrix
    glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

    // skybox cube
    glBindVertexArray(skyboxVAO);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glBindVertexArray(0);
    glDepthFunc(GL_LESS); // Set depth function back to default
    /*SKYBOX*/

    // Swap the screen buffers
    glfwSwapBuffers(window);
}

/*SKYBOX*/
glDeleteVertexArrays(1, &VAO);
glDeleteVertexArrays(1, &LightVAO);
glDeleteBuffers(1, &VBO);
glDeleteBuffers(1, &EBO);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVBO);
/*SKYBOX*/
// Terminate GLFW, clearing any resources allocated by GLFW.
glfwTerminate();


return 0;
}
```

Outside the main, we find other functions that control the animations and interaction that the user has with the program.

Camera controls.

```
// Moves/alters the camera positions based on user input
void DoMovement()
{

    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);

    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);


    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);


    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);


    }

}
```

Controls for animations, room light and closing the window.



```
// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }

    if (keys[GLFW_KEY_SPACE])
    {

        active = !active;
    }

    if (keys[GLFW_KEY_L])
    {
        luces = !luces;
        // Luz
        if (luces)
        {
            Light1 = glm::vec3(1.0f, 1.0f, 1.0f);

        }
        else
        {
            Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
        }
    }
}
```

Mouse controls to move the camera view.

```
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go from bottom to left

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

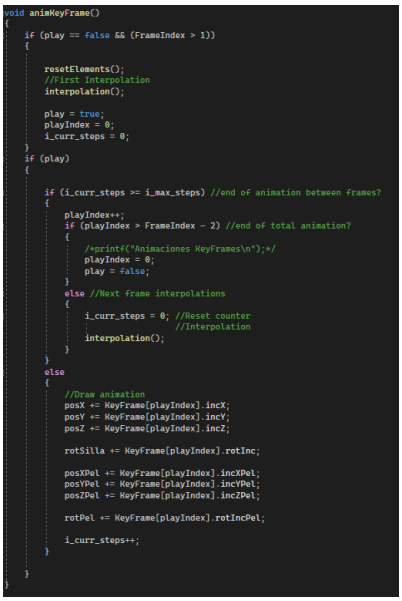
Animation's main function.

```
// Funcion que controla todas las animaciones
void animacion()
{
    if (active)
    {
        animPuertas();
        animKeyFrame();
        animacionCarro();
    }
}
```

Function that controls door animations.

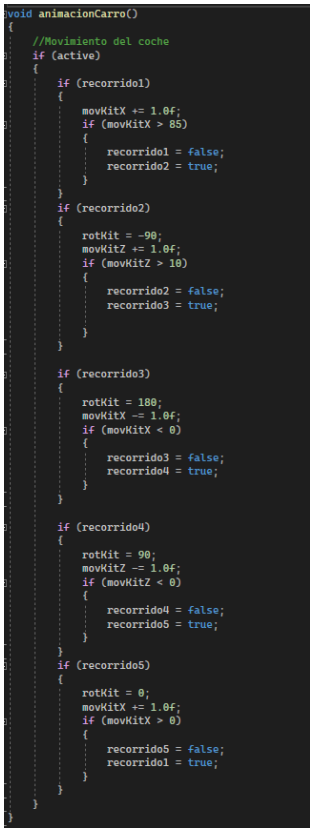
```
// Animacion de la puerta de entrada
void animPuertas()
{
    // Puerta Entrada
    if (rotPuerta > -90 && !abiertaPuerta)
        rotPuerta -= 1.0f;
    else
        abiertaPuerta = true;
    if (rotPuerta <= 0 && abiertaPuerta)
        rotPuerta += 1.0f;
    else
        abiertaPuerta = false;

    // Puerta Cristal
    if (movCristal > -5.0 && !abiertaCristal)
        movCristal -= 0.1f;
    else
        abiertaCristal = true;
    if (movCristal < -0.1 && abiertaCristal)
        movCristal += 0.1f;
    else
        abiertaCristal = false;
}
```

Function that animates the chair and the ball with the KeyFrames method.



```
void animKeyFrame()
{
    if (play == false && (FrameIndex > 1))
    {

        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    if (play)
    {

        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                /*printf("Animaciones KeyFrames\n");*/
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                                  //Interpolation
                interpolation();
            }
        }
        else
        {
            //Draw animation
            posX += KeyFrame[playIndex].incX;
            posY += KeyFrame[playIndex].incY;
            posZ += KeyFrame[playIndex].incZ;

            rotSilla += KeyFrame[playIndex].rotInc;

            posXPel += KeyFrame[playIndex].incXPel;
            posYPel += KeyFrame[playIndex].incYPel;
            posZPel += KeyFrame[playIndex].incZPel;

            rotPel += KeyFrame[playIndex].rotIncPel;

            i_curr_steps++;
        }

    }
}
```

Function that controls the way of the car's animation.



```
void animacionCarro()
{
    //Movimiento del coche
    if (active)
    {
        if (recorrido1)
        {
            movKitX += 1.0f;
            if (movKitX > 85)
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
        {
            rotKit = -90;
            movKitZ += 1.0f;
            if (movKitZ > 10)
            {
                recorrido2 = false;
                recorrido3 = true;

            }
        }

        if (recorrido3)
        {
            rotKit = 180;
            movKitX -= 1.0f;
            if (movKitX < 0)
            {
                recorrido3 = false;
                recorrido4 = true;
            }
        }

        if (recorrido4)
        {
            rotKit = 90;
            movKitZ -= 1.0f;
            if (movKitZ < 0)
            {
                recorrido4 = false;
                recorrido5 = true;
            }
        }
        if (recorrido5)
        {
            rotKit = 0;
            movKitX += 1.0f;
            if (movKitX > 0)
            {
                recorrido5 = false;
                recorrido1 = true;
            }
        }
    }
}
```

Conclusion

At the end of the project, apart from learning the concepts that we saw during the semester through repetition, the methodology in which it is thought to carry it out is the most similar to developing a real life project in the faculty. The simple fact of having to known the requirements, having a delivery date, advancing the project little by little, structuring it, talking with the professor (client) about the doubts that arose, having a simple planning to carry it out, documenting it and above all the problems that arise along the way and that have to be solved as soon as possible to avoid delays. This is what the development of the project left me with.