

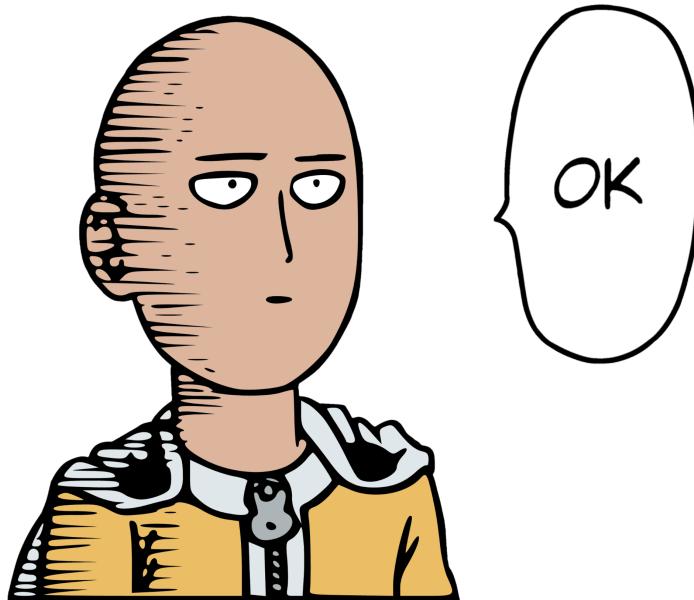
Universidad Nacional  
Autónoma de México



Facultad de Ingeniería

# Simulación de la casa de Saitama

---



Desarrollador:

Mario Alberto Vásquez Cancino

v. 1.0.0

---

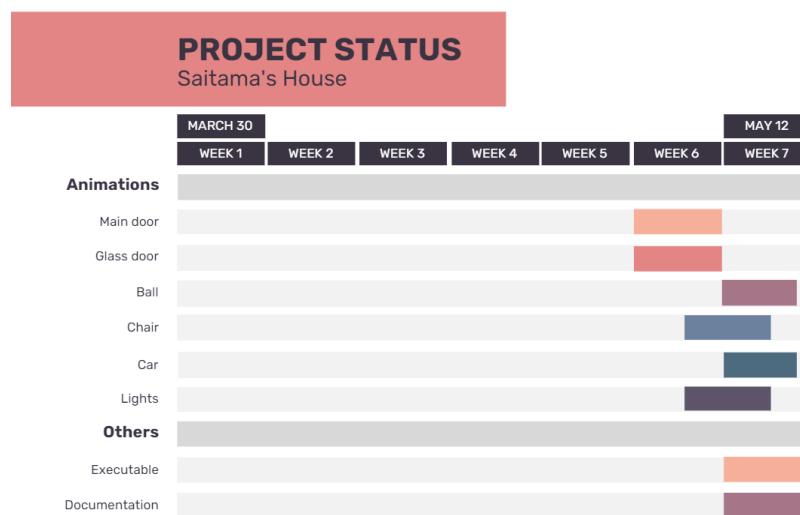
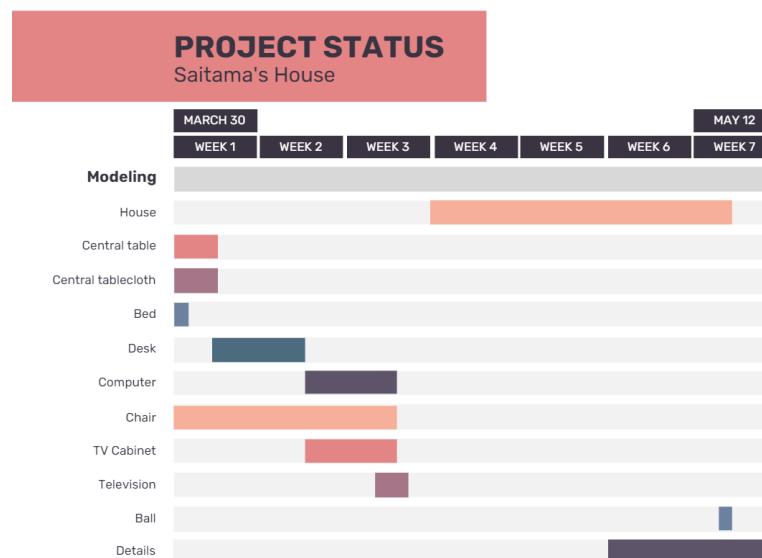
# Índice

<b>Objetivo</b>	<b>2</b>
<b>Plan de Trabajo</b>	<b>2</b>
<b>Alcance del Proyecto</b>	<b>3</b>
<b>Funcionamiento</b>	<b>5</b>
Instalación y Ejecución	5
Código Fuente	5
Estructura	5
ProyectoFinal.sln	5
External Libraries	6
ProyectoFinal	6
Código	7
<b>Conclusión</b>	<b>17</b>

## Objetivo

Ofrecer un recorrido virtual en una sencilla recreación de la casa de Saitama, protagonista de one punch man.

## Plan de Trabajo



## Alcance del Proyecto

Al finalizar este proyecto se debe obtener un programa que permita al usuario recorrer una recreación virtual sencilla de la casa de Saitama, protagonista de la serie One Punch Man.

La recreación virtual interna del cuarto debe parecerse lo más posible a la original, esta debe contar con al menos 7 elementos de la habitación, los elementos seleccionados fueron los siguientes:

- Mesa central
- Tapete central
- Silla
- Escritorio
- Computadora
- Cama
- Televisión
- Mueble de TV



Imagen de referencia del cuarto

La fachada exterior es diferente a la original para poder hacer más ameno el recorrido.



Imagen de referencia de la fachada

También se contará con 5 animaciones, las cuales son:

- Puerta principal abriéndose y cerrándose.
- Puerta de cristal abriéndose y cerrándose.
- Pelota cayendo de la mesa central.
- Silla moviéndose como si alguien se hubiera parado.
- Coche moviéndose afuera de la casa.

Para que el usuario pueda tener una interacción con el programa, se debe desplazarse mediante las teclas W,S,A y D, mover la vista con el mouse, detener y reproducir las animaciones con la barra espaciadora y controlar la luz interna del cuarto.

El proyecto finalizado con su documentación, archivo ejecutable y binarios deberá ser entregado el jueves 12 de mayo de 2022.

## Limitaciones

Debido a que es una recreación, al tiempo de desarrollo y los recursos disponibles el cuarto, las animaciones y elementos modelados no tendrán un parecido al 100% con respecto a los originales, pero se podrá tener un 90% de parecido con lo mostrado en la serie.

## Funcionamiento

### Instalación y Ejecución

Para correr la simulación véase el Manual de Usuario.

### Código Fuente

#### Estructura

El proyecto se realizó con Visual Studio por lo que en la carpeta raíz encontramos los siguientes elementos principales:

📁 External Libraries	30/03/2022 15:40	Carpeta de archivos
📁 ProyectoFinal	30/03/2022 15:41	Carpeta de archivos
📄 ProyectoFinal.sln	30/03/2022 15:41	Visual Studio Solution 2 KB

#### ProyectoFinal.sln

Solución de Visual Studio para manejar el proyecto.

## External Libraries

Librerías ocupadas en el proyecto.

 assimp	30/03/2022 15:41	Carpeta de archivos
 GLEW	30/03/2022 15:40	Carpeta de archivos
 GLFW	30/03/2022 15:40	Carpeta de archivos
 glm	30/03/2022 15:41	Carpeta de archivos
 SOIL2	30/03/2022 15:41	Carpeta de archivos

- **ASSIMP:** Librería que nos servirá para cargar modelos o escenas 3D almacenados en gran variedad de formatos.
- **GLEW:** Es una librería multiplataforma de código abierto escrita en C/C++ la cual nos proveerá de un mecanismo en tiempo de ejecución para determinar cuál de las funciones opengl están soportadas en la plataforma de destino.
- **GLFW:** Librería que nos permitirá crear y administrar las ventanas donde desplegamos los gráficos OpenGL.
- **GLM:** Librería nos brinda acceso a funciones matemáticas comúnmente utilizada en los gráficos 3D
- **SOIL:** Es una pequeña librería para cargar imágenes en 6 tipos de formatos.

## ProyectoFinal

Carpeta que contiene los archivos y recursos principales del proyecto.

 images	30/03/2022 15:41	Carpeta de archivos
 Models	30/03/2022 15:41	Carpeta de archivos
 Release	10/05/2022 22:45	Carpeta de archivos
 Shaders	30/03/2022 15:41	Carpeta de archivos
 SkyBox	30/03/2022 15:41	Carpeta de archivos
 SOIL2	30/03/2022 15:41	Carpeta de archivos
 315021963_Proyecto_Gpo08.cpp	10/05/2022 23:01	Archivo CPP 34 KB
 assimp-vc140-mt.dll	07/04/2019 23:07	Extensión de la aplicacion... 15.705 KB
 Camera.h	31/03/2019 1:51	C Include File 5 KB
 glew32.dll	31/07/2017 21:42	Extensión de la aplicacion... 381 KB
 MainPrueba.cpp	30/03/2022 15:41	Archivo CPP 28 KB
 Mesh.h	08/04/2019 0:21	C Include File 4 KB
 meshAnim.h	15/03/2022 21:49	C Include File 7 KB
 Model.h	08/04/2019 3:15	C Include File 8 KB
 modelAnim.h	15/03/2022 21:49	C Include File 27 KB
 ProyectoFinal\vcproj	03/05/2022 19:57	VC++ Project 11 KB
 ProyectoFinal.vcxproj.filters	03/05/2022 19:57	VC++ Project Filters F... 2 KB
 ProyectoFinal.vcxproj.user	30/03/2022 15:41	Per-User Project Opti... 1 KB
 Shader.h	25/03/2019 3:38	C Include File 4 KB
 stb_image.h	09/01/2019 6:03	C Include File 249 KB
 Texture.h	15/03/2022 21:49	C Include File 3 KB

- **images:** Carpeta de imágenes que se ocuparon en el proyecto.
- **Models:** Carpeta con todos los modelos que se ocupan en el proyecto.
- **Shaders:** Carpeta que contiene todos los shaders para la luz, animaciones, etc.
- **Skybox:** Carpeta con las imágenes para el Skybox.
- **SOIL2:** Librería para cargar imágenes en 6 tipos de formatos.
- **315021963\_Proyecto\_Gpo08.cpp:** Código principal del proyecto.
- **assimp-vc140-mt.dll:** Archivo para manejar la librería ASSIMP.
- **Camera.h:** Archivo de cabecera para controlar la cámara.
- **glew32.dll:** Librería de OpenGL Extension Wrangler.
- **Mesh.h y meshAnim.h:** Archivos de cabecera para manejar las mallas para las texturas de los modelos.
- **Model.h y modelAnim.h:** Archivos de cabecera para manejar los modelos juntos a sus mallas y texturas.
- **Shader.h:** Archivo de cabecera para manejar los shaders.
- **stb\_image.h:** Librería para cargar imágenes hecha por Sean Barrett.
- **Texture.h:** Archivo de cabecera para cargar las texturas del SkyBox

## Código

Abriendo el archivo principal del proyecto (315021963\_Proyecto\_Gpo08.cpp) encontramos:

Importación de las librerías que ocuparemos.

```
#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*/
```

Prototipos de las funciones y variables globales a ocupar.

```
// 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 / 2.0f;
GLfloat lastY = HEIGHT / 2.0f;
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;
    }
}

```

Dentro de la función main encontramos:

Inicialización de GLFW, las opciones de OpenGL y creación de la ventana.

```

// 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);

```

Importación de los modelos.

```
// 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");
```

Inicialización de los KeyFrames para algunas animaciones.

```
//Incialización 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;
}

// Guardando 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;

// Guardando 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 para las luces e inicialización del SKyBox.

```

// Set texture units
lightingShader.Use();
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);

//SKYBOX
GLfloat skyboxVertices[] = {
    // Front Face
    -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,
    // Back Face
    -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,
    // Top Face
    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,
    // Bottom Face
    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,
    // Right Face
    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,
    // Left Face
    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[] =
{
    // Indices that we start from 0!
    0,1,2,
    4,5,6,
    8,9,10,
    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(-1.5f, 0.5f, 0.5f),
    glm::vec3(1.5f, 0.5f, -15.0f),
    glm::vec3(-1.5f, -2.5f, -2.5f),
    glm::vec3(1.5f, -2.5f, -2.5f),
    glm::vec3(-0.4f, -0.4f, -3.5f),
    glm::vec3(-1.7f, 3.0f, -7.5f),
    glm::vec3(1.7f, 3.0f, -7.5f),
    glm::vec3(-1.5f, 2.0f, -2.5f),
    glm::vec3(1.5f, 2.0f, -2.5f),
    glm::vec3(-1.5f, -1.0f, -1.5f),
    glm::vec3(-1.3f, 1.0f, -1.5f)
};

```

Carga de las imágenes del SkyBox con el VAO y 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);

glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLFloat), (GLvoid*)(3 * sizeof(GLFloat)));
glEnableVertexAttribArray(1);

glVertexAttribPointer(2, 3, 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 want 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);
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);
glBindVertexArray(skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
glBindVertexArray(skyboxVAO);

// 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
glEnable(GL_DEPTH_TEST);
projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);

```

En main nos encontraremos con GAME LOOP que es una ciclo que se repetirá hasta que se cierre la ventana para poder refrescar la imagen y actualizar los cambios hechos, en pocas palabras poder movernos en la simulación, ver las luces y animaciones.

Configuración de la luz ambiental, de la luz que posee la cámara y de la luz interna de la casa.

```

// calculate destination of current frame
def currentFrame() { return glnGetFrame(); }
def lastFrame() { return glnGetFrame(); }
def currentPos() { return glnGetPos(); }
lastPos = currentPos();
lastFrame = currentFrame;

// If keyboard/mouse events have been activated (key pressed, mouse moved etc.) and call corresponding response functions
glnGetKeyboard();
glnGetMouse();

// Clear the colorbuffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Set camera
glMatrixMode(GL_MODELVIEW);
glnGetViewMatrix();

// Set lighting
glEnable(GL_DEPTH_TEST);

// Set Model
// Set lighting parameters when setting uniforms/drawing objects
lightingShader.use();
glUniform4fv(uniforms["color_attenuation"], 1, &att);
glUniform4fv(uniforms["color_atten_att"], 1, &atten);
glUniform4fv(uniforms["color_diffuse_att"], 1, &diff);
glUniform4fv(uniforms["color_specular_att"], 1, &spec);
glUniform4fv(uniforms["color_emissive"], 1, &emissive);

// Set light
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLight(GL_LIGHT0, GL_AMBIENT, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_DIFFUSE, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_SPECULAR, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_CONSTANT, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_LINEAR, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_EXP, 0.0f, 0.0f, 0.0f, 1.0f);
glLight(GL_LIGHT0, GL_EXP2, 0.0f, 0.0f, 0.0f, 1.0f);

// Set material properties
glMaterialfv(GL_FRONT, "material_shininess", 16.0f);

// Create camera transformation
glMatrixMode(GL_MODELVIEW);
view = camera.GetViewMatrix();

// Set the uniforms
// Set the uniforms
glUniform4fv(uniforms["model"], 1, &model);
glUniform4fv(uniforms["view"], 1, &view);
glUniform4fv(uniforms["projection"], 1, &projection);

// Pass the matrices to the shader
glUniformMatrix4fv(uniforms["model_mat"], 1, GL_FALSE, glValue_ptr(model));
glUniformMatrix4fv(uniforms["view_mat"], 1, GL_FALSE, glValue_ptr(view));
glUniformMatrix4fv(uniforms["proj_mat"], 1, GL_FALSE, glValue_ptr(projection));

glnDrawTextArray(0);

```

Se dibujan los modelos en su posición específica y con las configuraciones necesarias para que sean transparentes o no y si van a estar en movimiento o fijos.

```

gln::mat4 model(1);
//Carga de modelo
view = camera.GetViewMatrix();
//aaaaaaaaaaaaaaaaaaaaaaa CASA
model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 0);
casa.DrawLightingShader();

model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, 1.0f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 0);
puerta.DrawLightingShader();

//Ventana
model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 0);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 1, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 2, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 3, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 4, 0.0f, 1.0f, 0.0f);
puerta.DrawLightingShader();

glnEnable(GL_BLEND); //Activar la funcionalidad para trabajar el canal alpha
glnBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 1);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 1, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 2, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 3, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 4, 0.0f, 1.0f, 0.0f);
puerta.DrawLightingShader();

glnEnable(GL_BLEND); //Activar la funcionalidad para trabajar el canal alpha
glnBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 1);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 1, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 2, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 3, 0.0f, 1.0f, 0.0f);
glnUniformfv(uniforms["lightingShader_Program", "colorAlpha"], 4, 0.0f, 1.0f, 0.0f);
puerta.DrawLightingShader();

model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 0);
casa.DrawLightingShader();

//Casa
model = gln::mat4();
model = gln::translateModel( gl::vec3(0.0f, -8.5f, 0.0f));
glnUniform4fv(uniforms["model"], 1, &model);
glnUniformfv(uniforms["lightingShader_Program", "activeTransparency"], 0);
casa.DrawLightingShader();

```

```

// Mesa
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(lightingShader);

// Piso
model = glm::mat4();
model = glm::rotate(model, glm::radians(rotp1), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::translate(model, glm::vec3(posP1, posP1, posP1));
plata.Draw(lightingShader);

// Silla
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp2), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
silla.Draw(lightingShader);

// Escritorio
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp3), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
escritorio.Draw(lightingShader);

// Tapete
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp4), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
tapete.Draw(lightingShader);

// Laptop
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp5), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
laptop.Draw(lightingShader);

// Monitor
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp6), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
monitor.Draw(lightingShader);

// Cama
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp7), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
cama.Draw(lightingShader);

// Control IV
model = glm::mat4();
model = glm::translate(model, glm::vec3(0.025f, -0.5f, 0.0f));
model = glm::rotate(model, glm::radians(rotp8), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
controlIV.Draw(lightingShader);

// Curva
model = glm::mat4();
model = glm::translate(model, PointCar + glm::vec3(movX, 0, movY));
model = glm::rotate(model, glm::radians(rotp9), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
curva.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");

// *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_EQUAL); // Change depth function so depth test passes when values are equal to depth buffer's content
SkyboxShader.Use();
view = glm::mat4(1.0f); mat = glm::mat4(Camera.GetInvMatrix()); // 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(cubeVAO);
glActiveTexture(GL_TEXTURE0);
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);
}

/*SWAPBUFFER*/
glDeleteVertexArrays(1, &VAO);
glDeleteVertexArrays(1, &lightVAO);
glDeleteBuffers(1, &VAO);
glDeleteBuffers(1, &EBO);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVAO);
/*SWAPBUFFER*/
// Terminate GLFW, clearing any resources allocated by GLFW.
glfwTerminate();
}

```

Cerrando los buffers ocupados.

```

gBindVertexArray(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_EQUAL); // Change depth function so depth test passes when values are equal to depth buffer's content
SkyboxShader.Use();
view = glm::mat4(1.0f); mat = glm::mat4(Camera.GetInvMatrix()); // 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(cubeVAO);
glActiveTexture(GL_TEXTURE0);
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);
}

/*SWAPBUFFER*/
glDeleteVertexArrays(1, &VAO);
glDeleteVertexArrays(1, &lightVAO);
glDeleteBuffers(1, &VAO);
glDeleteBuffers(1, &EBO);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVAO);
/*SWAPBUFFER*/
// Terminate GLFW, clearing any resources allocated by GLFW.
glfwTerminate();
}

```

Ya fuera del main, encontramos otras funciones que controlan las animaciones e interacción que tiene el usuario con el programa.

Controles de la cámara.

```
// 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);
    }
}
```

Controles para las animaciones, luz del cuarto y cerrar la ventana.

```
// 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); //Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        }
    }
}
```

Controles del mouse para mover la vista de la cámara.

```
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);
}
```

Función main de las animaciones.

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

Función que controla las animaciones de las puertas.

```
// 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;
}
```

Función que anima la silla y la pelota con el método de KeyFrames.

```
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();
            }
        }
        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++;
        }
    }
}
```

Función que controla la ruta de la animación del carro.

```
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;
            }
        }
    }
}
```

---

## Conclusión

Al finalizar el proyecto aparte de aprender los conceptos que vimos durante el semestre mediante la repetición, la metodología en la que está pensado para realizarlo es lo más parecido a desarrollar un proyecto de la vida real en la facultad. El simple hecho de tener que cumplir requisitos, tener fecha de entrega, ir avanzando en el proyecto poco a poco, estructurarlo, hablar con el profesor (cliente) sobre las dudas que surgían, tener una sencilla planeación para realizarlo, documentarlo y sobretodo los problemas que surgen en el camino y que se tienen que solucionar lo más pronto posible para no atrasarse. Es lo que me dejó el desarrollo del mismo.