UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO COMPUTADORA

GRUPO 3

Prof.: Ing. José Roque Román Guadarrama

PROYECTO FINAL

MARTE

Alumno:

BRITO SEGURA ANGEL

Manual Técnico

SEMESTRE 2021-2

Fecha de entrega: Agosto 12, 2021

INGENIERÍA EN COMPUTACIÓN

GEOMETRÍA

Los elementos implementados dentro del ambiente virtual fueron por carga de modelos, por lo que el texturizado se realizó con el programa de Blender.

En este programa se exportaron los modelos en formato OBJ o FBX y en el archivo ***Marte.cpp*** se declaró por cada modelo su variable tipo Model:

```
Model Eva_M;
Model Cuca_M;
Model FinnJake_M;
Model ClonO_M;
Model Yoda_M;
Model Droide1_M;
Model Droide2_M;
Model R2D2_M;

Model Curiosity_M;
Model Perseverance_M;
Model Rocas_M;
Model TieDV_M;
Model TieB_M;
Model TieD_M;
Model AT_M;

//Stormtroppers
Model CuerpoC_M;
Model BrazoCI_M;
Model BrazoCD_M;
Model PiernaCI_M;
Model PiernaCD_M;

//Naves
Model InterceptorJedi_M;
Model SpeederBike_M;
Model Pod_M;

Model Mont_M;
Model Camino_M;
Model Basura1_M;
Model Basura2_M;
Model Basura3_M;
```

En la función *main()* de dicho archivo se realizó la carga de los mismos:

```
Eva_M = Model();
Eva_M.LoadModel("Models/EVA.obj");

Cuca_M = Model();
Cuca_M.LoadModel("Models/cucaracha.obj");

ClonO_M = Model();
ClonO_M.LoadModel("Models/Stormtrooper.obj");

CuerpoC_M = Model();
CuerpoC_M.LoadModel("Models/Stormtrooper-Cuerpo.fbx");

BrazoCI_M = Model();
BrazoCI_M.LoadModel("Models/Stormtrooper-Brazo-Izquierdo.fbx");

BrazoCD_M = Model();
BrazoCD_M.LoadModel("Models/Stormtrooper-Brazo-Derecho.fbx");

PiernaCI_M = Model();
PiernaCI_M.LoadModel("Models/Stormtrooper-Pierna-Izquierda.fbx");

PiernaCD_M = Model();
PiernaCD_M.LoadModel("Models/Stormtrooper-Pierna-Derecha.fbx");

InterceptorJedi_M = Model();
InterceptorJedi_M.LoadModel("Models/InterceptorJedi.fbx");

SpeederBike_M = Model();
SpeederBike_M.LoadModel("Models/SpeederBike.obj");

Yoda_M = Model();
Yoda_M.LoadModel("Models/yoda.obj");

Droide1_M = Model();
Droide1_M.LoadModel("Models/droide-batalla.fbx");
```

Finalmente se colocaron dentro del escenario en la posición deseada:

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(200.0f, 0.0f, 180.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
model = glm::rotate(model, -115 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TieB_M.RenderModel();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-70.0f, 0.0f, 250.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TieD_M.RenderModel();

// Vehículo terrestre AT-AT
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(40.0f, 35.0f, 200.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
model = glm::rotate(model, -120 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AT_M.RenderModel();

//Curiosity
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-190.0f, -1.0f, 70.0f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Curiosity_M.RenderModel();

// CUBOS DE BASURA
//Lado derecho visto de frente de Wall-E
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(275.0f, 0.0f, 55.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Basura1_M.RenderModel();
```

En la siguiente captura de pantalla podemos observar algunos de los modelos cargados en el ambiente virtual:

AVATAR

El avatar es el personaje de Wall-E, con el cual el usuario podrá interactuar mediante el teclado. Para respetar la jerarquía, se utilizaron 4 modelos adicionales para separar cada parte del objeto:

```
Model Avatar_M;
Model BrazoD_M;
Model BrazoI_M;
Model PieD_M;
Model PieI_M;
```

Posteriormente en la función *main()* se realiza la carga de dichos modelos texturizados:

```
Avatar_M = Model();
Avatar_M.LoadModel("Models/Wall-E.obj");
BrazoI_M = Model();
BrazoI_M.LoadModel("Models/Brazo-Derecho-Frente.obj");
BrazoD_M = Model();
BrazoD_M.LoadModel("Models/Brazo-Izquierdo-Frente.obj");
PieD_M = Model();
PieD_M.LoadModel("Models/Pie-Derecho.fbx");
PieI_M = Model();
PieI_M.LoadModel("Models/Pie-Izquierdo.fbx");
```

Y a la hora de renderizarlos se hace uso de la matriz auxiliar:

```
//Brazo derecho Wall-E
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f + mainWindow.getMovAvatarX(), 0.0f + mainWindow.getMovAvatarZ(), 4.8f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BrazoD_M.RenderModel();

//Brazo izquierdo Wall-E
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f + mainWindow.getMovAvatarX(), 0.0f + mainWindow.getMovAvatarZ(), -12.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BrazoI_M.RenderModel();

//Pie derecho Wall-E
model = modelaux;
model = glm::translate(model, glm::vec3(3.6f, 2.4f, -6.5f));
model = glm::rotate(model, -10 * (posXrobot + posZrobot) * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); //Rotación de sus engranes
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
PieD_M.RenderModel();

//Pie izquierdo Wall-E
model = modelaux;
model = glm::translate(model, glm::vec3(3.6f, 2.4f, 6.5f));
model = glm::rotate(model, -10 * (posXrobot + posZrobot) * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); //Rotación de sus engranes
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
PieI_M.RenderModel();
```

Para activar la animación del teclado se tuvo que modificar el objeto Window, agregando como atributos públicos las banderas de control:

```
Window::Window(GLint windowWidth, GLint windowHeight) {
    width = windowWidth;
    height = windowHeight;
    prendeLuz = false; //Sin Luz
    verShow = true; //Con luz
    mueveYoda = false; //Inicia sin movimiento
    mueveR2D2 = false; //Inicia sin movimiento
    activaAnimacionWallE = false; //Activar animación Wall-E
    reseteaAnimacionWallE = false; //Resetear animación Wall-E
    activaAnimacionSpeeder = false; //Activar animación Speeder Bike
    reseteaAnimacionSpeeder = false; //Resetear animación Speeder Bike
    movimientoAvatar[0] = 0.0f; // X
    movimientoAvatar[1] = 0.0f; // Z
    for (size_t i = 0; i < 1024; i++) {
        keys[i] = 0;
    }
}
```

Y en el método *manejaTeclado()* de dicho objeto se agregaron más condiciones:

```
//Mover brazos del avatar con las flechas
if (key == GLFW_KEY_LEFT && theWindow->movimientoAvatar[0] >= -5.0) {
    theWindow->movimientoAvatar[0] -= 1.0; // X--
}
else if (key == GLFW_KEY_LEFT && theWindow->movimientoAvatar[0] < -5.0) {
    theWindow->movimientoAvatar[0] -= 0.0; // X--
}

if (key == GLFW_KEY_RIGHT && theWindow->movimientoAvatar[0] <= -1.0) {
    theWindow->movimientoAvatar[0] += 1.0; // X++
}
else if (key == GLFW_KEY_RIGHT && theWindow->movimientoAvatar[0] > -1.0) {
    theWindow->movimientoAvatar[0] += 0.0; // X++
}

if (key == GLFW_KEY_DOWN && theWindow->movimientoAvatar[0] < -5.0 && theWindow->movimientoAvatar[1] >= -3.0) {
    theWindow->movimientoAvatar[1] -= 1.0; // Y--
}
else if (key == GLFW_KEY_DOWN && theWindow->movimientoAvatar[0] < -5.0 && theWindow->movimientoAvatar[1] < -3.0) {
    theWindow->movimientoAvatar[1] -= 0.0; // Y--
}

if (key == GLFW_KEY_UP && theWindow->movimientoAvatar[0] < -5.0 && theWindow->movimientoAvatar[1] <= -1.0) {
    theWindow->movimientoAvatar[1] += 1.0; // Y++
}
else if (key == GLFW_KEY_UP && theWindow->movimientoAvatar[0] < -5.0 && theWindow->movimientoAvatar[1] > -1.0) {
    theWindow->movimientoAvatar[1] += 0.0; // Y++
}

if (key == GLFW_KEY_I && action == GLFW_PRESS && contadorAnimacionWallE % 2 == 0) {
    theWindow->activaAnimacionWallE = true;
    contadorAnimacionWallE = contadorAnimacionWallE + 1;
}
else if (key == GLFW_KEY_I && action == GLFW_PRESS && contadorAnimacionWallE % 2 != 0) {
    theWindow->activaAnimacionWallE = false;
    contadorAnimacionWallE = contadorAnimacionWallE + 1;
}
if (key == GLFW_KEY_R && action == GLFW_PRESS && contadorResetWallE % 2 == 0) {
    theWindow->reseteaAnimacionWallE = true;
    theWindow->reseteaAnimacionSpeeder = true;
    contadorResetWallE = contadorResetWallE + 1;
    contadorResetSpeeder = contadorResetSpeeder + 1;
}
else if (key == GLFW_KEY_R && action == GLFW_PRESS && contadorResetWallE % 2 != 0) {
    theWindow->reseteaAnimacionWallE = false;
    theWindow->reseteaAnimacionSpeeder = false;
    contadorResetWallE = contadorResetWallE + 1;
    contadorResetSpeeder = contadorResetSpeeder + 1;
}
```

La animación de Wall-E es básica al ocupar dos transformaciones geométricas y se está ocupando un efecto de sonido:

```
if (mainWindow.activaAnimacionWallE) {
    if (reproduceW) {
        sonido->play2D("media/wall-e.mp3", false); //Efecto de sonido
    }
    reproduceW = false;
    if (posXrobot <= 100.0f && posXrobot > -65.0f && adelanteX == 1) {
        posXrobot -= 0.01*deltaTime;
        spotLights[1].SetPos(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot));
    }
    else if (posXrobot <= -65.0f && posXrobot > -66.0f && adelanteX == 1) {
        if (posZrobot > -20.0f && adelanteZ == 1) {
            giro = 1;
            posZrobot -= 0.01*deltaTime;
            spotLights[1].SetPos(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot));
            spotLights[1].SetFlash(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot), glm::vec3(0.0f, 0.0f, -1.0f));
        }
        else {
            adelanteZ = 0;
            arriba = 1;
        }

        if (posZrobot < 20.0f && adelanteZ == 0) {
            giro = 2;
            posZrobot += 0.01*deltaTime;
            spotLights[1].SetPos(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot));
            spotLights[1].SetFlash(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else
        {
            adelanteZ = 1;
            abajo = 1;
        }
```

```
        if (!adelanteZ && rotaHeli < 90) {
            posZrobot += offsetPos * deltaTime;
            rotaHeli += offsetHeli * deltaTime;
        }
        if (adelanteZ && rotaHeli > 90 && rotaHeli < 180) {
            posZrobot -= offsetPos * deltaTime;
            rotaHeli += offsetHeli * deltaTime;
        }

        if (rotaHeli > 180) {
            rotaHeli = 0;
        }
    }
}
else if (mainWindow.reseteaAnimacionWallE) {
    reproduceW = true;
    posXrobot = 0.0f;
    posZrobot = 0.0f;
    spotLights[1].SetPos(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot));
    spotLights[1].SetFlash(glm::vec3(-1.0 + posXrobot, 8.5f, 0.1 + posZrobot), glm::vec3(-1.0f, 0.0f, 0.0f));
}
```

A continuación se muestran capturas de dicha animación:

RECORRIDO

Como se utiliza el teclado para cambiar de cámara, se tuvo que agregar el atributo de cámara al objeto *Window*, así como se crean otras condiciones para el método asociado:

```
void Window::ManejaTeclado(GLFWwindow* window, int key, int code, int action, int mode) {
    Window* theWindow = static_cast<Window*>(glfwGetWindowUserPointer(window));

    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS) {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

    if (key == GLFW_KEY_1) {
        theWindow->camara = 1;
    }
    if (key == GLFW_KEY_2) {
        theWindow->camara = 2;
    }
    if (key == GLFW_KEY_3) {
        theWindow->camara = 3;
    }
```

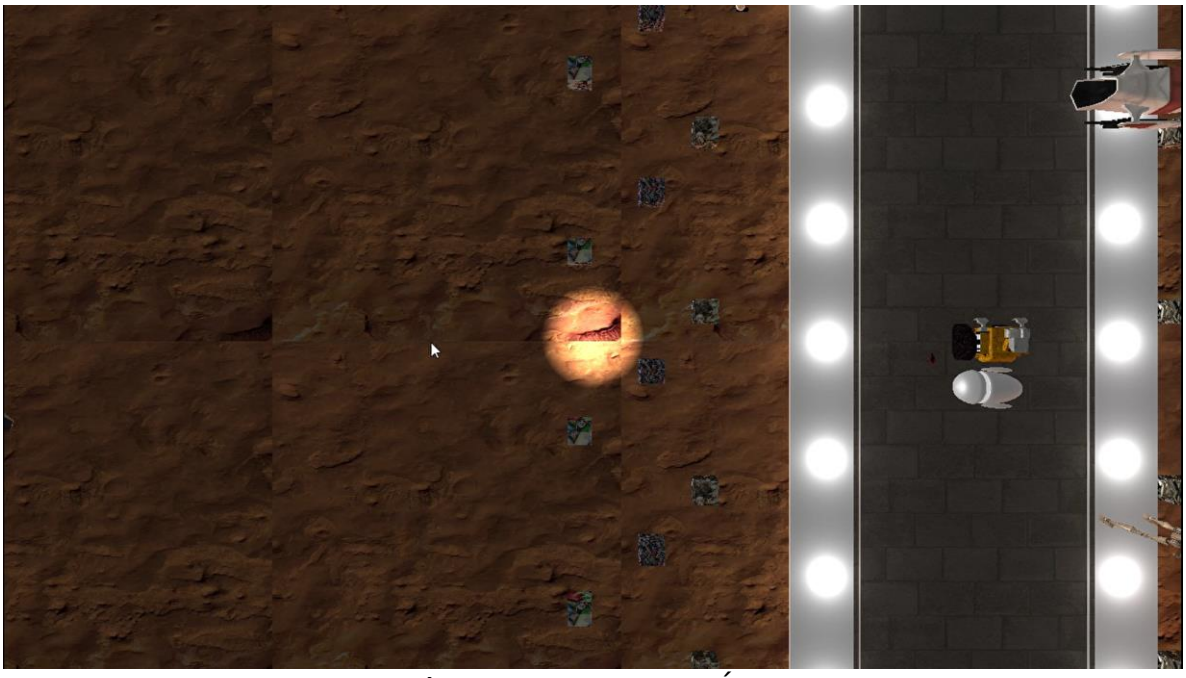
En la función *main()* se hace el manejo correspondiente de la cámara:

```
if (mainWindow.getCamara() == 1) {

    camera.keyControl(mainWindow.getsKeys(), deltaTime);
    camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());
    camera.setCameraDirection(camera.getCameraDirection());
    camera.setCameraPosition(camera.getCameraPosition());
}
else if (mainWindow.getCamara() == 2) {
    camera.setGiro(0,0);
    switch (giro)
    {
    case 1:
        camera.setCameraPosition(glm::vec3(0.0f + posXrobot, 10.0f, 20.0f + posZrobot));
        camera.setCameraDirection(glm::vec3(0.0f, 0.0f, -1.0f));
        break;

    case 2:
        camera.setCameraPosition(glm::vec3(0.0f + posXrobot, 10.0f, -20.0f + posZrobot));
        camera.setCameraDirection(glm::vec3(0.0f, 0.0f, 1.0f));
        break;
    default:
        camera.setCameraPosition(glm::vec3(25.0f + posXrobot, 10.0f, 0.0f + posZrobot));
        camera.setCameraDirection(glm::vec3(-1.0f, 0.0f, 0.0f));
    }
}
else if (mainWindow.getCamara() == 3) {
    camera.setCameraPosition(glm::vec3(0.0+camera.getCameraPosition().x, 100.0f, 0.0f+camera.getCameraPosition().z));
    camera.setCameraDirection(glm::vec3(-1.0f, 0.0f, 0.0f));
    camera.setGiro(0, -90);
    camera.keyControlXZ(mainWindow.getsKeys(), deltaTime);
}
```

A continuación se muestran las tres cámaras posibles, con lo cual el usuario puede recorrer el escenario:

Cámara 1 - LIBRE

Cámara 2 - PLANO PARALELO A XZ



Cámara 3 - PLANO AÉREO

ILUMINACIÓN

Para las luminarias, se ocuparon luces de tipo *PointLight* a continuación se muestra la declaración de algunas:

```
pointLights[53] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -240.0f, 1.5f, 25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;

pointLights[54] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -240.0f, 1.5f, -25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;

pointLights[55] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -260.0f, 1.5f, 25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;

pointLights[56] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -260.0f, 1.5f, -25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;

pointLights[57] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -280.0f, 1.5f, 25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;

pointLights[58] = PointLight(1.0f, 1.0f, 1.0f, //color blanco
    1.0f, 1.0f, //coeficientes
    -280.0f, 1.5f, -25.0f, //posición dentro del escenario
    0.3f, 0.2f, 0.1f); //ecuación de segundo grado para la atenuación ax2 + bx + c
pointLightCount++;
```

Para prenderlas (noche) y apagarlas (día) será de acuerdo al cambio en el Skybox:

```
// ------------------------------------------ CAMBIO DE SKYBOX -------
// skybox de día
if (playSol)
{
    skybox.DrawSkybox(camera.calculateViewMatrix(), projection);
    shaderList[0].UseShader();
    shaderList[0].SetPointLights(pointLights, 0);
}

// Proyección de skybox de noche
if (playLuna)
{
    skyboxNight.DrawSkybox(camera.calculateViewMatrix(), projection);
    shaderList[0].UseShader();
    shaderList[0].SetPointLights(pointLights, pointLightCount);
}
```

Como se puede observar, el Skybox cambiará de acuerdo a este ciclo:

Para el show de luces, es necesario agregar al objeto *Window* las banderas de control, así como en el método *ManejaTeclado()* agregar las condiciones necesarias:

```
if (key == GLFW_KEY_C && action == GLFW_PRESS && contadorColor % 2 == 0) {
    theWindow->prendeLuz = true;
    contadorColor = contadorColor+1;
}
else if (key == GLFW_KEY_C && action == GLFW_PRESS && contadorColor % 2 != 0) {
    theWindow->prendeLuz = false;
    contadorColor = contadorColor+1;
}

if (key == GLFW_KEY_H && action == GLFW_PRESS && contadorShow % 2 == 0) {
    theWindow->verShow = false;
    contadorShow++;
}
else if (key == GLFW_KEY_H && action == GLFW_PRESS && contadorShow % 2 != 0) {
    theWindow->verShow = true;
    contadorShow++;
}
```

De acuerdo a esta bandera, se inicia o apaga el show de luces:

```
if (mainWindow.getPrendeLuz()) {
    colorLuzW = glm::vec3(1.0f, 1.0f, 0.0f); //Encender luz

    colorLuzF1 = glm::vec3(1.0f, 0.0f, 1.0f); //color morado
    colorLuzF2 = glm::vec3(0.0f, 1.0f, 1.0f); //color azul verdoso
    colorLuzF3 = glm::vec3(1.0f, 0.5f, 1.0f); //color lila

    if (luzFZ1 < 80.0f && luzFX1 == -60.0f) {
        luzFZ1 += 0.1 * deltaTime;
        if (luzFZ2 > 35.0f) {
            luzFZ2 -= 0.1 * deltaTime;
        }
        else {
            luzFX2 += 0.1 * deltaTime;
        }
        luzFX3 -= 0.1 * deltaTime;
        luzFZ3 += 0.1 * deltaTime;
    }
    else if (luzFX2 > -60.0f) {
        luzFZ1 -= 0.11 * deltaTime;
        if (luzFX1 > -70.0f) {
            luzFX1 -= 0.1 * deltaTime;
        }
        luzFX3 += 0.18 * deltaTime;
        luzFZ3 -= 0.15 * deltaTime;
        luzFX2 -= 0.1 * deltaTime;
    }
    else { //Valores iniciales
        luzFX1 = -60.0f;
        luzFZ1 = 40.0f;
        luzFX3 = -70.0f;
        luzFZ3 = 50.0f;
        luzFX2 = -50.0f;
        luzFZ2 = 50.0f;
    }
}
else { //Apagar luces
    colorLuzW = glm::vec3(0.0f, 0.0f, 0.0f);
    colorLuzF1 = glm::vec3(0.0f, 0.0f, 0.0f);
    colorLuzF2 = glm::vec3(0.0f, 0.0f, 0.0f);
    colorLuzF3 = glm::vec3(0.0f, 0.0f, 0.0f);
    //Valores iniciales
    luzFX1 = -60.0f;
    luzFZ1 = 40.0f;
    luzFX3 = -70.0f;
    luzFZ3 = 50.0f;
    luzFX2 = -50.0f;
    luzFZ2 = 50.0f;
}
```

A continuación se muestra el show de luces, así como se puede prender la luz de Wall-E al mismo tiempo:

ANIMACIÓN

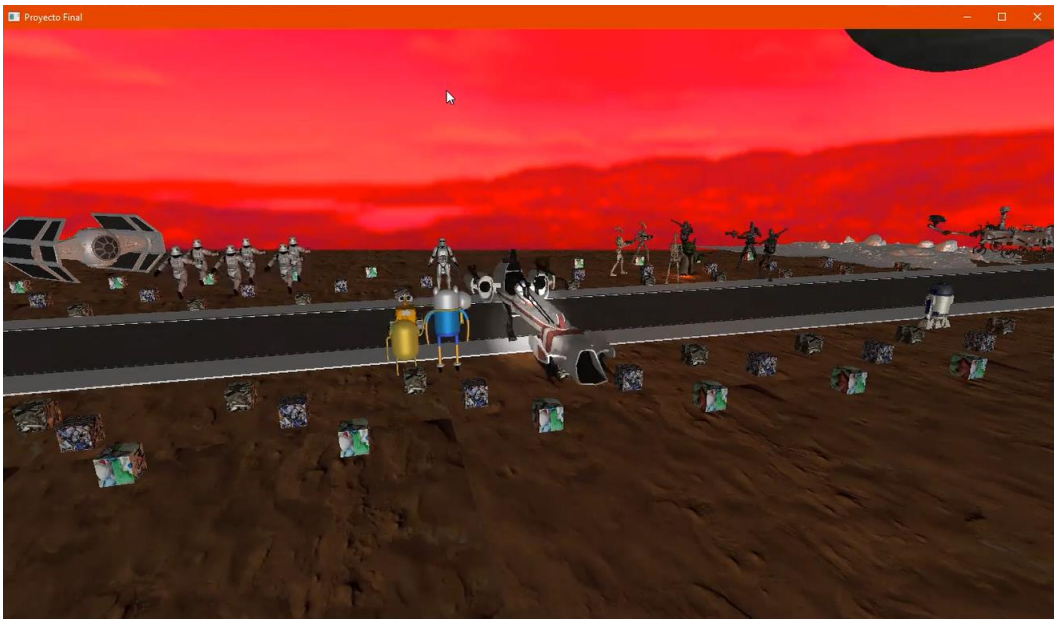
Para las animaciones básicas, se tiene que agregar atributos y métodos al objeto *Window*, así como agregar las condiciones necesarias para el método *ManejaTeclado()*:

```
if (key == GLFW_KEY_Y) {
    theWindow->mueveYoda = true;
}

if (key == GLFW_KEY_N) {
    theWindow->mueveR2D2 = true;
}
```

```
if (key == GLFW_KEY_P && action == GLFW_PRESS && contadorAnimacionSpeeder % 2 == 0) {
    theWindow->activaAnimacionSpeeder = true;
    contadorAnimacionSpeeder = contadorAnimacionSpeeder + 1;
}
else if (key == GLFW_KEY_P && action == GLFW_PRESS && contadorAnimacionSpeeder % 2 != 0) {
    theWindow->activaAnimacionSpeeder = false;
    contadorAnimacionSpeeder = contadorAnimacionSpeeder + 1;
}
```

La primera animación es la del Speeder Bike:

```
model = glm::mat4(1.0);
if (posYspeeder > 0.0f && adelanteY == 0) {
    model = glm::translate(model, glm::vec3(-40.0f, -5.0f + posYspeeder, -60.0f + posZspeeder));
    model = glm::rotate(model, -180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else if (posYspeeder < 20.0f && adelanteY == 1 || posYspeeder == 0.0f) {
    model = glm::translate(model, glm::vec3(-40.0f, -5.0f + posYspeeder, -60.0f + posZspeeder));
}
model = glm::scale(model, glm::vec3(15.0f, 15.0f, 15.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SpeederBike_M.RenderModel();

if (mainWindow.activaAnimacionSpeeder) {
    if (reproduceS) {
        sonido->play2D("media/star-wars-dogfight.mp3", false); //Efecto de sonido
    }
    reproduceS = false;
    if (posYspeeder > 0.0f && adelanteY == 0) {
        posYspeeder -= 0.5 * sin(0.05);
        posZspeeder -= 0.5 * cos(0.05);
    }
    else {
        adelanteY = 1;
    }

    if (posYspeeder < 20.0f && adelanteY == 1) {
        posYspeeder += 0.5 * sin(0.05);
        posZspeeder += 0.5 * cos(0.05);
    }
    else
    {
        adelanteY = 0;
    }
}
else if (mainWindow.reseteaAnimacionSpeeder) {
    reproduceS = true;
    posZspeeder = 0.0f;
    posYspeeder = 0.0f;
}
```


Proyecto Final


Proyecto Final


Proyecto Final

La segunda es la de R2-D2:

```
if (mainWindow.getMueveR2D2()) {
    if (reproduceR) {
        sonido->play2D("media/r2d2_sms.mp3", false); //Efecto de sonido
    }
    reproduceR = false;

    if (posYr < 10.0f && rotaRY == 90.0f) {
        posYr += 0.1f*deltaTime;
    }
    else if (rotaRY > -90.0f && posZr == 40.0f) {
        rotaRY -= 0.3f*deltaTime;
    }
    else if (posZr < 80.0f) {
        posZr += 0.1f*deltaTime;
    }
    else if (posYr > -1.0f) {
        posYr -= 0.1f*deltaTime;
    }
    else if (rotaRZ < 5.0f) {
        rotaRZ += 0.2f*deltaTime;
    }
    else if (posZr < 120.0f) {
        posZr += 0.1f*deltaTime;
    }
    else if (rotaRY < 0.0f) {
        rotaRY += 0.2f*deltaTime;
    }
    else {
        mainWindow.setMueveR2D2(false); //Termina animación
    }
}
else {
    reproduceR = true;
    rotaRY = 90.0f;
    rotaRZ = 0.0f;
    posZr = 40.0f;
    posYr = -1.0f;
}
```


Proyecto Final


Proyecto Final

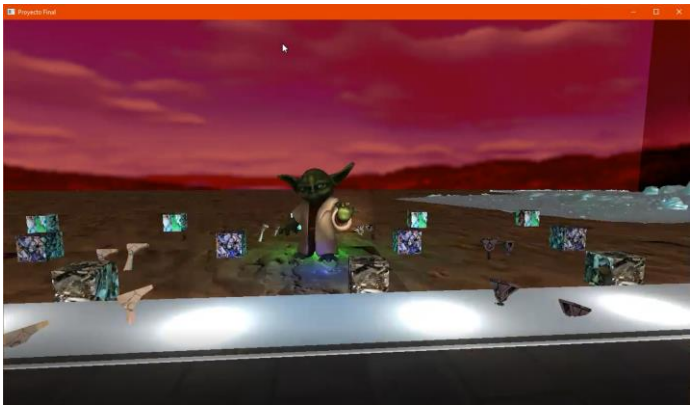

Proyecto Final

Adicionalmente, se tiene una animación en Yoda en conjunto con un show de luces:

```
if (mainWindow.getMueveYoda()) {
    if (reproduceY) {
        sonido->play2D("media/yoda-effects.mp3", false); //Efecto de sonido
    }
    reproduceY = false;
    colorLuzY1 = glm::vec3(0.0f, 0.0f, 0.0f); //apagada
    if (rotaY < 90.0f) {
        rotaY += 0.2f*deltaTime;
        luzYX2 -= 0.1*deltaTime;
        luzYX3 += 0.1*deltaTime;
        posYdroide += 0.2*deltaTime;
        colorLuzY2 = glm::vec3(0.5f, 1.0f, 1.0f); //cyan
        colorLuzY3 = glm::vec3(0.5f, 1.0f, 0.5f); //amarillo claro
    }
    else if (rotaY < 180.0f) {
        rotaY += 0.2f*deltaTime;
        luzYZ2 += 0.04*deltaTime;
        luzYZ3 -= 0.04*deltaTime;
        rotaD -= 0.35f*deltaTime;
        colorLuzY2 = glm::vec3(0.33f, 0.67f, 1.0f); //cyan claro
        colorLuzY3 = glm::vec3(0.33f, 1.0f, 0.33f); //verde acuoso
    }
    else if (rotaY < 270.0f) {
        rotaY += 0.2f*deltaTime;
        luzYX2 += 0.1*deltaTime;
        luzYX3 -= 0.1*deltaTime;
        posYdroide -= 0.2*deltaTime;
        colorLuzY2 = glm::vec3(0.16f, 0.34f, 1.0f); //cyan fuerte
        colorLuzY3 = glm::vec3(0.16f, 1.0f, 0.16f); //verde claro
    }
    else if (rotaY < 360.0f) {
        rotaY += 0.2f*deltaTime;
        luzYZ2 -= 0.04*deltaTime;
        luzYZ3 += 0.04*deltaTime;
        colorLuzY2 = glm::vec3(0.0f, 0.0f, 1.0f); //azul
        colorLuzY3 = glm::vec3(0.0f, 1.0f, 0.0f); //verde
    }
    else {
        mainWindow.setMueveYoda(false); //Termina animación
    }
}
else {
    reproduceY = true;
    colorLuzY1 = glm::vec3(1.0f, 0.27f, 0.0f); //color naranja rojizo
    colorLuzY2 = glm::vec3(0.0f, 0.0f, 0.0f); //apagada
    colorLuzY3 = glm::vec3(0.0f, 0.0f, 0.0f); //apagada
    rotaY = 0.0f;
    posYdroide = 0.0f;
    rotaD = 0.0f;
}
```


Proyecto Final


Proyecto Final


Proyecto Final

Las animaciones complejas son cíclicas y no requieren uso de teclado:

I. Interceptor Jedi & Estrella de la Muerte

```
if (baja) {
    desaparece = 1.0f; //Aparece estrella de la muerte
    if (posYnave > 110.0f) {
        posYnave -= 0.02f * deltaTime;
    }
    else {
        giraI = true;
    }
}

if (giraI) {
    desaparece = 1.0f; //Aparece estrella de la muerte
    baja = false;
    if (rotacionN1 < 90.0f) {
        rotacionN1 += 0.2f *deltaTime;
        posXnave += 0.01f *deltaTime;
    }
    else if (rotacionN2 > -90.0f) {
        rotacionN2 -= 0.2f *deltaTime;
        posXnave += 0.01f *deltaTime;
    }
    else
    {
        avanza = true;
    }
}

if (avanza) {
    desaparece = 1.0f; //Aparece estrella de la muerte
    giraI = false;
    if (posXnave < 75.0f) {
        posXnave += 0.1f *deltaTime;
    } else if (posXnave < 80.0f) {
        if (reproduceE) {
            sonido->play2D("media/explosion.wav", false); //Efecto de sonido
        }
        reproduceE = false;
        desaparece = 0.0f;
        posXnave += 0.05f *deltaTime;
    } else {
        vuelta = true;
    }
}
```

```
if (vuelta) {
    desaparece = 0.0f;
    avanza = false;
    if (reproduceSW) {
        sonido->play2D("media/interceptor-jedi.mp3", false); //Efecto de sonido
    }
    reproduceSW = false;
    if (rotacionN1 < 180.0f) {
        rotacionN1 += 0.2f *deltaTime;
        posXnave += 0.1f *deltaTime;
    }
    else {
        irse = true;
    }
}

if (irse) {
    desaparece = 0.0f;
    vuelta = false;
    if (angulo < 100.0f) {
        angulo += 1.0f * deltaTime; //ángulo de inclinación
        posZnave = tan(angulo * toRadians); //arriba y abajo
    }
    else { //Regresar a valores iniciales
        irse = false;
        reproduceE = true;
        reproduceSW = true;
        desaparece = 1.0f;
        baja = true;
        posXnave = -150.0f;
        posYnave = 180.0f;
        rotacionN1 = 0.0f;
        rotacionN2 = 0.0f;
    }
}
```


Proyecto Final


Proyecto Final


Proyecto Final

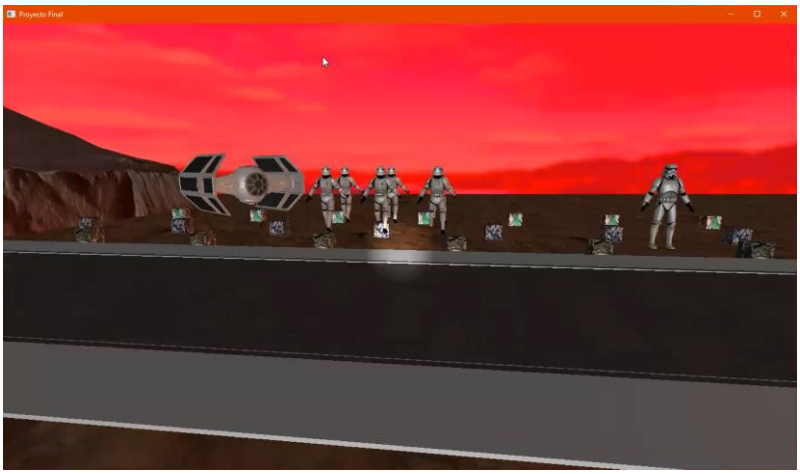
II. Stormtroopers

```
if (avanzaS) {
    if (iniciaR) {
        if (rotacionDerecha > -30.0f) {
            rotacionDerecha -= 0.02f * deltaTime;
            rotacionIzquierda += 0.02f * deltaTime;
        }
        else {
            regresaR = true;
            iniciaR = false;
        }
    }
    if (regresaR) {
        if (rotacionDerecha < 30.0f) {
            rotacionDerecha += 0.02f * deltaTime;
            rotacionIzquierda -= 0.02f * deltaTime;
        }
        else {
            iniciaR = true;
            regresaR = false;
        }
    }
    if (posZclon < -25.0f) {
        posZclon += 0.01f * deltaTime;
    }
    else {
        vueltaS = true;
    }
}
```

```
if (vueltaS) {
    avanzaS = false;
    if (iniciaR) {
        if (rotacionDerecha > -30.0f) {
            rotacionDerecha -= 0.02f * deltaTime;
            rotacionIzquierda += 0.02f * deltaTime;
        }
        else {
            regresaR = true;
            iniciaR = false;
        }
    }
    if (regresaR) {
        if (rotacionDerecha < 30.0f) {
            rotacionDerecha += 0.02f * deltaTime;
            rotacionIzquierda -= 0.02f * deltaTime;
        }
        else {
            iniciaR = true;
            regresaR = false;
        }
    }
    if (rotacionS1 > -180.0f) {
        rotacionS1 -= 0.2f *deltaTime;
    }
    else {
        regresa = true;
    }
}
```

```
if (regresa) {
    vueltaS = false;
    if (iniciaR) {
        if (rotacionDerecha > -30.0f) {
            rotacionDerecha -= 0.02f * deltaTime;
            rotacionIzquierda += 0.02f * deltaTime;
        }
        else {
            regresaR = true;
            iniciaR = false;
        }
    }
    if (regresaR) {
        if (rotacionDerecha < 30.0f) {
            rotacionDerecha += 0.02f * deltaTime;
            rotacionIzquierda -= 0.02f * deltaTime;
        }
        else {
            iniciaR = true;
            regresaR = false;
        }
    }
    if (posZclon > -80.0f) {
        posZclon -= 0.01f * deltaTime;
    }
    else {
        vuelveOrigen = true;
    }
}
```

```
if (vuelveOrigen) {
    regresa = false;
    if (reproduceDV) {
        sonido->play2D("media/darth_vader.mp3", false); //Efecto de sonido
    }
    reproduceDV = false;
    if (iniciaR) {
        if (rotacionDerecha > -30.0f) {
            rotacionDerecha -= 0.02f * deltaTime;
            rotacionIzquierda += 0.02f * deltaTime;
        }
        else {
            regresaR = true;
            iniciaR = false;
        }
    }
    if (regresaR) {
        if (rotacionDerecha < 30.0f) {
            rotacionDerecha += 0.02f * deltaTime;
            rotacionIzquierda -= 0.02f * deltaTime;
        }
        else {
            iniciaR = true;
            regresaR = false;
        }
    }
    if (rotacionS1 < 0.0f) {
        rotacionS1 += 0.2f *deltaTime;
        rotacionS2 -= 0.4f * deltaTime;
    }
    else {
        subeBrazos = true;
    }
}

if (subeBrazos) {
    vuelveOrigen = false;
    if (rotacionBrazoD > -180.0f) {
        rotacionBrazoD -= 0.55f * deltaTime;
        rotacionBrazoI -= 0.5f * deltaTime;
    }
    else {
        terminaS = true;
    }
}
                if (terminaS) {
                    subeBrazos = false;
                    if (rotacionBrazoD < 0.0f) {
                        rotacionBrazoD += 0.55f * deltaTime;
                        rotacionBrazoI += 0.5f * deltaTime;
                    }
                    else { //Regresar a valores iniciales
                        terminaS = false;
                        rotacionBrazoD = 0.0f;
                        rotacionBrazoI = 0.0f;
                        avanzaS = true;
                        reproduceDV = true;
                    }
                }
```


Proyecto Final


Proyecto Final


Proyecto Final

III. Eva

```
if (iniciaE) {
    if (rotaEX < 90.0f) {
        rotaEX += 0.2f*deltaTime;
    }
    else {
        giraE = true;
    }
}

if (giraE) {
    iniciaE = false;
    if (rotaEZ < 90.0f) {
        rotaEZ += 0.2f*deltaTime;
        posXeva -= 0.05f*deltaTime;
    }
    else {
        enfrenteW = true;
    }
}

if (enfrenteW) {
    giraE = false;
    posZeva -= 0.05f*deltaTime;
    if (rotaEX < 180.0f) {
        rotaEX += 0.2f*deltaTime;
    }
    if (posZeva < 11.0f) {
        if (reproduceWE) {
            sonido->play2D("media/wall-e-eve.mp3", false); //Efecto de sonido
        }
        reproduceWE = false;
        if (posZeva < -14.0f) {
            if (reproduceD) {  //Efectos de sonido
                sonido->play2D("media/directive.mp3", false); //Efecto de sonido
            }
            reproduceD = false;
            vueltaEW = true;
        }
    }
}
```

```
if (vueltaEW) {
    enfrenteW = false;
    if (rotaEX > 90.0f) {
        rotaEX -= 0.2f*deltaTime;
        posZeva -= 0.01f*deltaTime;
    }
    if (rotaEY > -180.0f) {
        rotaEY -= 0.2f*deltaTime;
        posZeva -= 0.01f*deltaTime;
    }
    if (posXeva < 10.0f) {
        posXeva += 0.05f*deltaTime;
    }
    else {
        regresaE = true;
    }
}
```

```
if (regresaE) {
    vueltaEW = false;
    if (rotaEY > -270.0f) {
        rotaEY -= 0.2f*deltaTime;
    }
    if (posZeva < 30.0f) {
        posZeva += 0.05f*deltaTime;
        posXeva += 0.01f*deltaTime;
    } else if (rotaEY > -360.0f) {
        rotaEY -= 0.2f*deltaTime;
        posXeva -= 0.05f*deltaTime;
    }
    else { //Regresar a valores iniciales
        regresaE = false;
        posXeva = 0.0f;
        posZeva = 30.0f;
        rotaEX = 0.0f;
        rotaEY = -90.0f;
        rotaEZ = 0.0f;
        iniciaE = true;
        reproduceD = true;
        reproduceWE = true;
    }
}
```

Adicionalmente, el ciclo del Sol y las Lunas de Marte se implementó mediante la técnica de KeyFrames:

```
void resetElementsLuna(void){
    LunaX = 0;
    LunaY = 0;
}

//Funcion de Interpolacion general para Luna
void interpolationLUNA(int playIndex) {
    KeyFrameLuna[playIndex].XPlus = (KeyFrameLuna[playIndex + 1].X - KeyFrameLuna[playIndex].X) / i_max_steps;
    KeyFrameLuna[playIndex].YPlus = (KeyFrameLuna[playIndex + 1].Y - KeyFrameLuna[playIndex].Y) / i_max_steps;
}

//FUNCION QUE DEFINE LA ANIMACION DE LA LUNA
void animaLuna(void)
{
    //Movimiento del objeto
    if (playLuna)
    {
        //primer interpolacion
        if (playIndexLuna == 0 and i_curr_steps == 0)
            interpolationLUNA(playIndexLuna);

        if (i_curr_steps >= i_max_steps) //fin de un frame
        {
            //le sumo uno al indice
            playIndexLuna++;
            if (playIndexLuna > FrameIndexLuna-1)//fin de la animacion
            {
                playLuna = false;
                playSol = true;
                playIndexLuna = 0;
                resetElementsLuna();
                i_curr_steps = 0;
            }
            else //Siguiente frame
            {
                i_curr_steps = 0; //Reset
                //Interpolation
                interpolationLUNA(playIndexLuna);

            }
        }
        else
        {
            //Animacion
            LunaX += KeyFrameLuna[playIndexLuna].XPlus;
            LunaY += KeyFrameLuna[playIndexLuna].YPlus;
            i_curr_steps++;
        }

    }
}
```

AUDIO

Se utilizó la librería de **irrKlang** para el sonido ambiental:

```
irrklang::ISoundEngine* sonido = irrklang::createIrrKlangDevice();

if (!sonido) {
    printf("Error al iniciar el audio\n");
    return -1;
}

sonido->play2D("media/air.mp3", true); //Reproducir sonido de fondo -Soundtrack-

//Banderas para efectos de sonido
bool reproduceW = true, reproduceS = true;
```

Los efectos de sonido están presentes en cada una de las animaciones mostradas, excepto la del Sol y las Lunas. Dichos efectos se activan con banderas:

```
bool reproduceE = true;
bool reproduceSW = true;

//vistas
bool flag = true;
int giro=0;

// Stormtrooper
GLfloat rotacionDerecha = 0.0f;
GLfloat rotacionIzquierda = 0.0f;
GLfloat rotacionBrazoD = 0.0f;
GLfloat rotacionBrazoI = 0.0f;
GLfloat rotacionS1 = 0.0f;
GLfloat rotacionS2 = 0.0f;
float posXclon = -100.0f;
float posZclon = -80.0f;
float posYclon = 12.0;
bool iniciaR = true;
bool regresaR = false;
bool avanzaS = true;
bool vueltaS = false;
bool regresa = false;
bool vuelveOrigen = false;
bool terminaS = false;
bool subeBrazos = false;
bool reproduceDV = true;

//Eva
GLfloat anguloE = 0.0f;
float posYeva = 0.0f;
float posXeva = 0.0f;
float posZeva = 30.0f;
GLfloat rotaEX = 0.0f;
GLfloat rotaEY = -90.0f;
GLfloat rotaEZ = 0.0f;
bool iniciaE = true;
bool giraE = false;
bool enfrenteW = false;
bool vueltaEW = false;
bool regresaE = false;
bool reproduceD = true;
bool reproduceWE = true;

//Yoda
GLfloat rotaY = 0.0f;
float posYdroide = 0.0f;
GLfloat rotaD = 0.0f;
bool reproduceY = true;
```

A continuación se listan los enlaces de donde se obtuvieron los modelos, librería de audio y audios utilizados:

- Modelos:
  - ✓ https://sketchfab.com/3d-models/jedi-star-fighter-0b641c2f2b854f1f9ae7f2a731e44dbd
  - ✓ https://sketchfab.com/3d-models/star-wars-barc-speeder-bike-56ff0b8c18744664aa121db5d2039eb3
  - ✓ https://sketchfab.com/3d-models/dikwrnkrt9ts-stormtrooper-ecf2ebbb6880485f936047929b33fe57
  - ✓ https://sketchfab.com/3d-models/wall-e-modeling-c64a10b13e2a45efb98f9780553a2140
  - ✓ https://sketchfab.com/3d-models/wall-e-eve-91ba6b0cdb9d45da9cb7c014d348aff3
  - ✓ https://sketchfab.com/3d-models/finn-and-jake-4d739e8a98234e70b89e274ded543b28
  - ✓ https://sketchfab.com/3d-models/death-star-star-wars-699e5f635f7a4b21a4c0d2e3f5b68324
  - ✓ https://sketchfab.com/3d-models/space-vehicle-4b9b78c0a72b45fe9ea9b3920b49c906
  - ✓ https://sketchfab.com/3d-models/sgp24-yoda-master-03424fd66c6c4516a24194e4065b6b2b
  - ✓ https://sketchfab.com/3d-models/r2d2-lowpoly-907605d4ecb74bd98b9c97c0e160f2f7
  - ✓ https://sketchfab.com/3d-models/cockroach-f46e0f9d7c99466994806b5b893e6550
  - ✓ https://sketchfab.com/3d-models/battle-droid-51341f3ba62a4d76a9216b4c36d673c5
  - ✓ https://sketchfab.com/3d-models/eyegee-droid-f6b41f2f3efe4143a51a2dfe901d7bb9
  - ✓ https://sketchfab.com/3d-models/mars-olympus-mons-53679864b41a43b0ae0459ce99d6adf0
  - ✓ https://sketchfab.com/3d-models/road-template-4d07393f253c4777ac66c7ec2887e599
  - ✓ https://sketchfab.com/3d-models/star-wars-escape-pod-744b53ababd640c089b9db03b02c968d
  - ✓ https://sketchfab.com/3d-models/mars-perseverance-rover-mastercam-z-r-l-6aea5060d2764b4fb866b2be997aa1bd
  - ✓ https://sketchfab.com/3d-models/curiosity-rover-full-3d-model-no-texture-files-7be1f1a5cf2948be83e04fd8d6cc0b1a
  - ✓ https://sketchfab.com/3d-models/perseverance-rover-made-in-tinkercad-mars-2021-4900faf1f1ae41b6ba30d21aeace6586
  - ✓ https://sketchfab.com/3d-models/darth-vaders-tie-advanced-x1-83654f360e1e4c72b716a2a60ed09031
  - ✓ https://sketchfab.com/3d-models/tie-bomber-dab029b751bd4cc293b66e3cc013e3a0
  - ✓ https://sketchfab.com/3d-models/star-wars-tie-defender-be33664c62f94508bbb77666475b23cd
  - ✓ https://sketchfab.com/3d-models/at-at-star-wars-eb4d795cac2e425db505ab69d4ebe922

- Imágenes para el Skybox:
https://www.pngegg.com/en/png-oeswi

- Librería de audio:
https://learnopengl.com/In-Practice/2D-Game/Audio
https://www.ambiera.com/irrklang/tutorials.html
https://www.ambiera.com/irrklang/downloads.html

- Audios introducidos:
  - **Soundtrack:** https://youtu.be/UmccvCkflpU
  - Interceptor Jedi: https://youtu.be/e9lapdvLSGw
  - Eva: http://www.moviesoundclips.net/sound.php?id=158
  - Wall-E: http://www.sonidosmp3gratis.com/download.php?id=15430&sonido=wall%20e%20eva
  - Speeder Bike: http://www.sonidosmp3gratis.com/download.php?id=16180&sonido=star%20wars%20dogfight
  - Yoda: http://www.sonidosmp3gratis.com/download.php?id=15673&sonido=yoda
  - Darth Vader y R2-D2: http://www.tonosfrikis.com/buscar/melodias/star_wars