UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Edición de circuitos de carreras por tramos acoplables para videojuegos de conducción

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Samuel Sáez García

**Tutor**: Roberto A. Vivó Hernando

[Curso 2017-2018]

# Resum

Este treball està centrat en la creació d'una aplicació gráfica de construcció de circuits de carreres orientada a videojocs. L'aplicació permet que l'usuari poda crear circuits seleccionant peces i modificant els paràmetres de les mateixes (longitud, ample, etc) , guardar i carregar circuits prèviament creats, i realitzar una simulació del recorregut del circuit. El projecte s'ha desenrotllat utilitzant la llibreria gràfica FreeGlut(derivada de OpenGL), recolzant-se en la llibreria GLM per a realitzar càlculs matricials. Per al desenrotllament del mateix s'ha seguit una implementació basada en el patró d'estats. Com a metodologia de desenrotllament s'ha seguit una versió individualitzada de la metodologia Scrum.

**Palabras clave:** FreeGlut, videojoc, carreres, aplicació gráfica, patró d'estats, OpenGL, Scrum.

# Resumen

Este trabajo está centrado en la creación de una aplicación gráfica de construcción de circuitos de carreras orientada a videojuegos. La aplicación permite que el usuario cree circuitos de carreras seleccionando piezas y modificando los parámetros de las mismas(longitud,ancho, etc), guardar y cargar circuitos previamente creados, y realizar una simulación del recorrido del circuito. El proyecto se ha desarrollado utilizando la librería gráfica FreeGlut(derivada de OpenGL), apoyándose en la librería GLM para realizar cálculos matriciales. Para el desarrollo del mismo se ha seguido una implementación basada en el patrón de estados. Como metodología de desarrollo se ha seguido una versión individualizada de la metodología Scrum.

**Palabras clave:** FreeGlut, videojuego, conducción, patrón de estados, aplicación gráfica, OpenGL, Scrum.

# Abstract

This thesis is focused on the creation of a race circuit builder application for video games.This application allows the user to build race circuits using different track pieces being able to modify the parameters of this piece(length, width, etc,), save and load circuits and simulate a race in a circuit. This project has been developed using the graphical library FreeGlut(derived from OpenGL), using the GML as support for the matrix operations. Also for implementing this project I have followed the state pattern methodology. As software development methodology I used an individual variant of Scrum methodology.

**Keywords :** FreeGlut, video game, track, race, state pattern, graphical application, OpenGL, Scrum.

# Tabla de contenidos

# 1. Introducción

En los siguientes puntos se ofrece una breve introducción acerca del trabajo realizado. Los diferentes temas tratados aquí se verán desarrollados en profundidad en los próximos capítulos.

## 1.1. Motivación

Puesto que en los últimos 30 años los videojuegos se han ido convirtiendo cada vez más en una de las principales industrias del entretenimiento junto al cine, es una buena idea aprender sobre su desarrollo y sobre las herramientas que se usan para el mismo, ya que cada vez surgen más desarrolladoras de videojuegos y a su vez puestos de trabajo relacionados con los videojuegos.

Mi propia motivación viene de que soy un entusiasta de los videojuegos y disfruto aprendiendo y comprendiendo cómo estos se diseñan y desarrollan. Gracias a todo lo que he ido aprendiendo a lo largo del grado y lo que he aprendido con la realización de este trabajo, he conseguido expandir mi modo de ver los videojuego. Lo que me permite ser más crítico con los mismos y al mismo tiempo apreciarlos de una forma más profesional.

Por otra parte, para acceder a algunos puestos de trabajo en la industria de los videojuegos, se requiere de un proyecto de presentación para demostrar las capacidades de uno mismo.

## 1.2. Objetivos

El objetivo principal de este proyecto es la implementación de una aplicación de creación de circuitos para videojuegos. Dicha aplicación debe de permitir crear circuitos de carreras mediante diferentes piezas(rectas, curvas, rampas, etc.).

Como objetivo adicional, se incluye la simulación del recorrido de dichos circuitos, así como diferentes menús y opciones que permiten configurar la experiencia del usuario en la aplicación, también se incluye la posibilidad de guardar y cargar circuitos previamente creados.

## 1.3. Metodología

Para la implementación del proyecto se ha seguido una metodología ágil de resolución de objetivos, al ser un trabajo individual se han adaptado las características que esta aporta al trabajo colectivo, pero enfocándose al trabajo individual.

Como metodología base se ha utilizado Scrum, metodología ágil grupal e incremental, especializada en entornos complejos con requisitos cambiantes o poco definidos, que consiste en centrarse en pequeños objetivos dentro de un plazo limitado de tiempo, estos objetivos se plantean con regularidad y se asignan a cada miembro del grupo, priorizando el beneficio que se pueda obtener de cada objetivo. Regularmente los miembros del grupo se reúnen para sincronizar el proceso y evaluar los posibles cambios en la planificación.

Tomando las mejores características de esta metodología y modificándolas para adaptarlas al trabajo individual he creado una metodología individual a seguir durante el desarrollo proyecto. La metodología consiste en una planificación diaria(equivalente a las reuniones diarias de Scrum) donde se decide según el estado actual del desarrollo cuál es el próximo objetivo. Estos objetivos se seleccionan según la prioridad de los mismos, enfocándose siempre en un área concreta de la aplicación y estableciendo un pequeño lapso de tiempo para su completación. Una vez se completa el objetivo actual se evalúan los siguientes objetivos y se asigna el más prioritario. En caso de no completar un objetivo a tiempo se evalúan las posibles causas, si fuese necesario replanifica el objetivo para más adelante.

Siguiendo la metodología autoimpuesta creo firmemente que he acelerado el proceso de implementación del proyecto, y he conseguido seguir un avance regular y bien enfocado a los aspectos más importantes del mismo.

## 1.4. Estructura de la memoria

Esta memoria está estructurada en diferentes capítulos en los que se explicaran los distintos aspectos del trabajo realizado, incluyendo información y explicaciones relevantes sobre las herramientas utilizadas.

En primer lugar se presentarán una serie de aplicaciones y videojuegos actuales que cumplen una función similar a la desarrollada en este trabajo.

En seguida revisaremos el análisis del problema a solucionar, y el diseño de la solución a dicho problema.

A continuación describen las herramientas utilizadas para la implementación del trabajo, comenzando por C++ como lenguaje de programación, a continuación explicar las características de la librería FreeGlut y Opengl, también explicarán las virtudes de la librería GLM.

Más tarde se explicará el diseño de una aplicación por estados. Y los bocetos creados para cada uno de los apartados visuales más destacables.

Posteriormente se detalla el trabajo de desarrollo del proyecto detallando las diferentes partes implicadas en este, como pueden ser el desarrollo de la cámara o el diseño de las diferentes piezas.

Seguidamente se revisa los pasos necesarios para la implantación de la solución y las pruebas a usuarios realizadas.

Finalmente se detallarán las conclusiones extraídas de la realización de este trabajo.

## 1.5. Agradecimientos

Quisiera dar mis más sinceros agradecimientos a mi tutor Roberto Vivó por darme la oportunidad de realizar este trabajo tan interesante. También me gustaría agradecer tanto a la comunidad de Stackoverflow como a la comunidad de Humus, por toda la ayuda que aportan a todos los desarrolladores. También agradecer a los compañeros y amigos que se prestaron como sujetos de prueba a la realización de pruebas a usuarios. Y por último agradecer a mis familiares y amigos que me apoyarán mientras realizaba este proyecto.

# 2. Estado del Arte

En este capítulo revisaremos diferentes aplicaciones que tienen características similares a las que este trabajo tiene como objetivo, para de esta forma comparar las diferentes formas de abordar el problema y las soluciones que estas aportan.

Como es poco común encontrar una aplicación únicamente centrada en la creación de circuitos, al menos publicada al público general, pues las desarrolladoras de videojuegos tienen sus propias herramientas internas desconocidas para los ajenos la compañía, la mayor parte de las aplicaciones analizadas a continuación serán videojuegos, más específicamente los creadores de circuitos que estos incluyen.

## 2.1. TrackMania

La saga de videojuegos TrackMania es la saga más reconocida en cuanto al subgénero de creación de circuitos, famosa por su gran variedad de tramos y piezas utilizables para la creación de los circuitos. A continuación analizaremos algunas de sus características, centrándonos específicamente en el título TrackMania United Forever:

**Figura 1**: imagen del videojuego TrackMania United Forever

Las características más interesantes de este título en relación al objetivo establecido en este trabajo son:

1. Constructor mediante piezas bien diferenciadas
2. Piezas poco modificables: las piezas tienen muchos parámetros fijos, orientación, longitud, etc
3. Colocación por cuadrantes, las piezas que forman un circuito se colocan dentro de una cuadrícula ya preestablecida, ocupando los huecos que la pieza necesite
4. Gran variedad de piezas, loopings, saltos, curvas, peraltes
5. La posibilidad de exportar circuitos entre la versión gratuita del videojuego y la versión de pago

Teniendo en cuenta estas características se podría decir que TrackMania United Forever se acerca bastante al objetivo que se desea alcanzar, por lo que se tendrá muy en cuenta durante el desarrollo del mismo para comparar/apoyarse en el diseño de la solución.

## 2.2. Gran Turismo

La saga de videojuegos Gran Turismo es una saga reconocida, por la meticulosa demostración de simulación de físicas de conducción y la gran variedad de vehiculos que esta implementa en cada una de sus entregas. Algunos títulos de la saga incluyen un editor de circuitos, en este caso se analizará el editor de Gran Turismo 6(lanzado el 6 de diciembre de 2013), conocido como Track Path Editor el cual es una aplicación externa al propio juego.


**Figura 2**: imagen del videojuego Gran Turismo 6

Las características más interesantes de este editor son:

1. En lugar de seleccionar piezas, dibujamos directamente el tramo o curva sobre el terreno
2. La posibilidad de modificar cada tramo individualmente, en cuanto a longitud, o curvatura del mismo
3. La interacción entre la aplicación y el videojuego
4. La automatización de todo lo referente a la inclinación de los tramos, pues esta viene dada por el terreno seleccionado
5. Y por otra parte la pérdida de la variedad en cuanto a los tramos se refiere pues solo podemos introducir tramos paralelos al suelo

Este editor, se centra más en la propia modificación de cada tramo, más que en la propia variedad de los mismos. La capacidad de modificar cada tramo es una propiedad interesante que se tendrá en cuenta a lo largo del desarrollo de este trabajo.

## 2.3. Race Track Builder y Bob's Track Builder

Race Track Builder es una aplicación diseñada por Brendon Pywell, centrada puramente en la creación y diseño de carreteras y su entorno. Brendon Pywell también desarrolló Bob's Track Builder, una aplicación similar pero más enfocada a la creación de circuitos de carreras. Dado el objetivo de este trabajo me centraré en la segunda, pues es la que más se acerca al objetivo del trabajo.

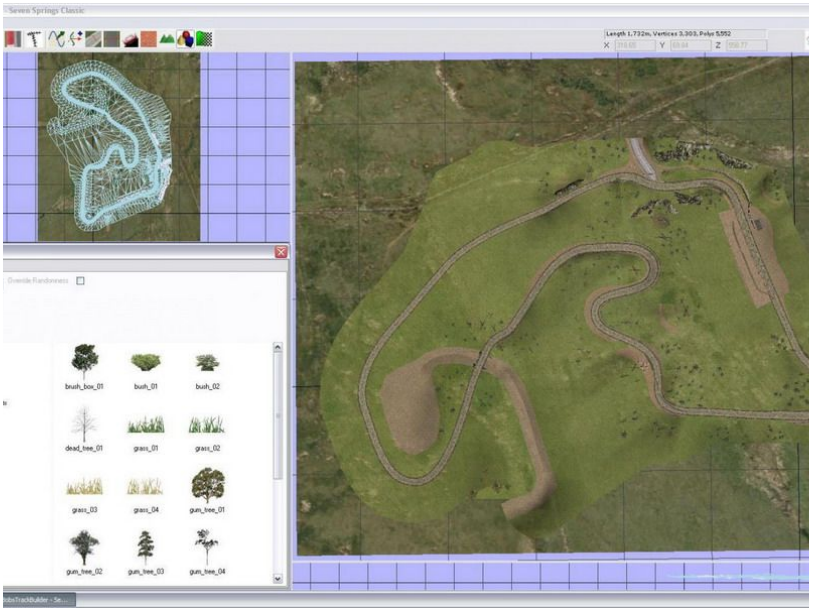

**Figura 3**: imagen de la aplicación Bob's Track Builder

Bob's Track Builder es una aplicación muy compleja, que se podría clasificar como una aplicación de modelado 3D centrada en el diseño de circuitos de carreras, con ella podremos trazar los circuitos modificar el material de los mismos, las físicas del trazado, las inclinaciones y el terreno colindante al circuito, añadir elementos al escenarios como hierbas, árboles, vallas, etc. Carece de la posibilidad de crear tramos tales como loopings o rampas.

De las aplicaciones analizadas esta es la más compleja y la que más posibilidades a la hora de la creación se refiere. Por eso mismo tiene muchos puntos interesantes que servirán de inspiración en la realización de este trabajo. Por otra parte al ser una aplicación tan compleja es muy difícil de utilizar para el usuario medio, esta característica se intentará evitar incluir en la aplicación desarrollada.

## 2.4. Crítica al estado del Arte

Tras analizar las diferentes aplicaciones presentadas anteriormente, uno puede ver que cada una tiene sus puntos fuertes y sus puntos débiles, ya que cada una se tiene que adaptar a un tipo de diseño de circuitos.

No existe una aplicación que abarque todos los aspectos de la creación de circuitos, ni que llegue a todos los públicos. Por un lado las hay excesivamente complejas con multitud de opciones, que son poco intuitivas y alejan al usuario menos avanzado. Y por otro las hay excesivamente sencillas, las cuales no otorgan a los usuarios más avanzados la posibilidades que estos necesitan.

Finalmente no existen apenas aplicaciones que ofrezcan la posibilidad de creación de circuitos fuera del entorno de desarrollo de los videojuegos o no estén incluidas en un videojuego.

## 2.5. Propuesta

En este trabajo se propone diseñar una aplicación que aune y simplifique las mejores características de cada una de las aplicaciones previamente analizadas, es decir se intentará conseguir que la aplicación final tenga las siguientes características:

1. Sencilla de usar, interesa que no sea una aplicación excesivamente compleja
2. Variedad de tramos, incluyendo loopings, rampas, curvas, etc
3. La posibilidad de modificar los diferentes parámetros de cada pieza
4. La posibilidad de lanzar una simulación del recorrido del circuito

5. La capacidad de guardar y cargar circuitos previamente creados, así como diseñar un sistema de guardado que facilite la posibilidad de cargar los circuitos en otra aplicación
6. Que tenga una gran portabilidad
7. Sencilla instalación

De esta forma aunamos y simplificamos algunas de las aplicaciones previamente analizadas, encontrando el hueco que se quiere rellenar con esta aplicación.

# 3. Análisis del problema

En este capítulo analizaremos el problema a solventar, comenzado por el análisis de requisitos donde se analizarán las diferentes requisitos funcionales que deberá cumplir nuestra aplicación. Tras ello se expondrán los distintos casos de uso que harán uso de las funciones descritas como requisitos. Finalmente se analizarán las distintas soluciones posibles y se especificará la solución elegida.

## 3.1. Definiciones

A continuación se definen ciertos términos que aparecen en este capítulo, para facilitar al lector su entendimiento.

**Menú:** interfaz de usuario que brinda diversas opciones de interacción mediante la selección de artefacto visual que clarifique la acción resultado de la misma.

**Estado:** momento particular dentro del tiempo de ejecución de la aplicación, ya sea un momento pausado como podría ser un menú o la representación de alguna otra acción.

**Escena:** conjunto de artefactos visuales que se dibujan en un preciso momento.

**Overlay:** interfaz de usuario que se muestra sobre la escena dibujada permitiendo que se visualice la escena sin problema.

**Cámara:** es la forma en la que se visualiza la escena.

**Path:** camino de puntos en el espacio de la escena.

**Motor gráfico:** conjunto de rutinas diseñadas para el dibujo 2D y 3D así como la representación de físicas.

## 3.2. Requisitos funcionales

La especificación de requisitos tiene como finalidad conocer que se espera obtener de la aplicación, definiendo de manera clara las funcionalidades que presentará la aplicación que se quiere desarrollar.

Siguiendo con lo declarado en el apartado anterior, se buscará realizar una aplicación simple y fácil de usar, que otorgue gran libertad al usuario a la hora de modificar el circuito.

### 3.2.1. Requisitos de la navegación a través de los menús

Los requisitos funcionales que deberán de cumplir los diferentes menús tales como:

1. Se podrá de navegar hacia otro menú o estado
2. De igual forma se deberá de poder volver al menú o estado anterior, ya sea directa o indirectamente
3. El usuario tiene que poder salir de la aplicación a través de la navegación por los menús
4. Debe de existir la opción de guardar el progreso siempre que sea necesario
5. Deben responder con rapidez al input del usuario

### 3.2.2. Requisitos de la creación de circuitos

Los requisitos funcionales de la propia creación de circuitos:

1. Debe ofrecer la posibilidad de añadir y eliminar piezas del circuito
2. Se tiene que poder modificar diversos parámetros de las piezas
3. El usuario podrá de acceder a un menú para navegar a través de otros estados
4. Debe de ser capaz de representar un circuito cargado desde un fichero

### 3.2.3. Requisitos del guardado de circuitos

El guardado de circuitos se realizará a través de acciones determinadas dentro de menús de la aplicación o dicho de otro modo será transparente para el usuario, por lo que no será necesario definir una interfaz de usuario para este apartado.

En cuanto a los requisitos funcionales se requiere que:

1. Se pueda guardar el circuito actual en el estado en el que se encuentre sin necesidad de que este esté terminado
2. Será totalmente transparente al usuario, el usuario no tiene que introducir ningún dato ni tiene la necesidad de indicar el lugar de guardado

3. Debe de almacenarse en ficheros separados para cada circuito
4. El patrón de almacenaje debe de ser de simple lectura, para facilitar la integración con otros proyectos
5. Se intentará ocupar el mínimo espacio de almacenamiento

### 3.2.4. Requisitos de la Simulación

La simulación consistirá en la recreación de un móvil circulando, esta se representará sobre el circuito.

Los requisitos funcionales de este apartado serían los siguientes:

1. La simulación funcionará en el tiempo, dejando que el usuario pueda modificar la velocidad a la que funciona
2. El cálculo del path será automático y transparente al usuario
3. El usuario podrá modificar la cámara con la que se representa la simulación
4. El usuario podrá detener la simulación y cambiar de estado en cualquier momento

## 3.3. Casos de uso

A continuación se detallarán los diferentes casos de uso según cada uno de los estados/menús inicialmente ideados. Se decidió desarrollar cada caso de uso para cada posible pantalla, para centrarse durante el desarrollo en pequeños objetivos a cumplir, para posteriormente ir cohesionando toda la solución.

### 3.3.1. Caso de uso menú principal

El usuario tendrá el siguiente caso de uso cuando se encuentre en el menú principal:

**Comenzar a crear un circuito:** el usuario accede directamente al estado de creacion de circuito, donde podrá comenzar a colocar piezas en su circuito.

**Seleccionar un circuito ya creado:** el usuario accede a una pantalla de selección donde se mostrarán todos los circuitos almacenados.

**Acceder a las opciones:** el usuario accede a una pantalla que le permite seleccionar y modificar diferentes aspectos del funcionamiento de la aplicación.

**Salir de la aplicación:** la aplicación termina su ejecución.

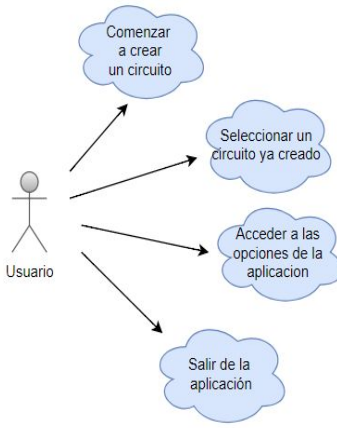


**Figura 4**: diagrama UML. Caso de uso menú principal

### 3.3.2. Caso de uso menú de opciones

El usuario tendrá el siguiente caso de uso cuando se encuentre en el menú de opciones:

**Modificar funciones:** el usuario tiene la capacidad de modificar ciertas funcionalidades de la aplicación desde esta misma pantalla, como por ejemplo activar el modo pantalla completa.

**Guardar cambios y volver:** el usuario vuelve a la pantalla en la que se encontraba antes de acceder a esta pantalla guardando los cambios en el fichero de opciones.

**Volver al estado anterior:** el usuario vuelve a la pantalla en la que se encontraba antes de acceder a esta pantalla sin guardar los cambios.

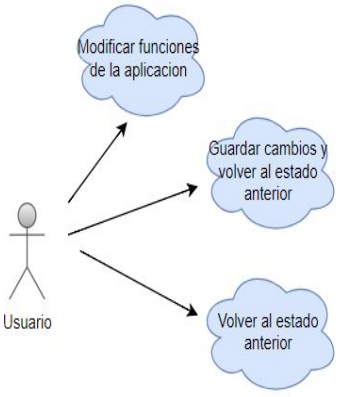


**Figura 5**: diagrama UML. Caso de uso menú opciones

### 3.3.3. Caso de uso creación de circuito

El usuario tendrá los dos siguientes casos de uso cuando esté creando un circuito:

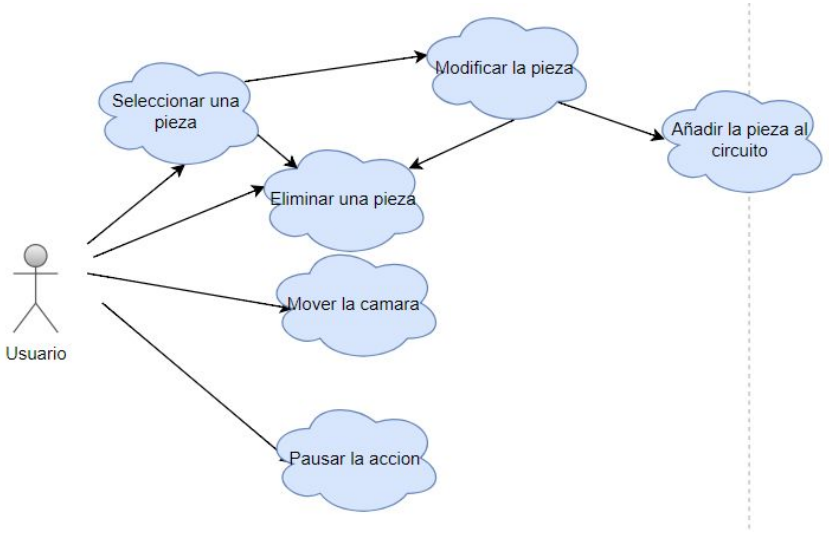


**Figura 6**: diagrama UML caso de uso creación de circuito

En este caso de uso encontramos las siguientes acciones:

**Seleccionar una pieza:** el usuario selecciona una pieza, dicha pieza pasa a estar seleccionada y pasamos a poder realizar la siguiente acción. **Modificar una pieza:** si el usuario ha seleccionado una pieza puede modificar los parámetros de la misma, una vez modificados el usuario puede desencadenar la acción siguiente. **Añadir la pieza al circuito:** donde la pieza ya modificada se añade al circuito en memoria.

**Eliminar una pieza:** El usuario puede eliminar del circuito la última pieza colocada o bien descartar las modificaciones que se han realizado a la pieza seleccionada.

**Mover la cámara:** El usuario puede modificar la posición de la cámara a su antojo.

**Pausar la acción:** El usuario puede pausar la acción lo que desencadenará el siguiente caso de uso.

### 3.3.3.1. Caso de uso pausa

Durante la pausa el usuario se encontrará ante el siguiente caso de uso:

**Iniciar simulación:** el usuario inicia la simulación sobre el circuito actual.

**Reanudar:** el usuario vuelve al caso de uso anterior donde puede continuar con la edición del circuito.

**Menú de opciones:** el usuario accede a la pantalla del menú de opciones.

**Cambiar de circuito:** el usuario accede a la pantalla de selección de circuito.

**Guardar circuito:** se guarda el circuito en el sistema de almacenamiento y se vuelve a la pantalla del menú principal.

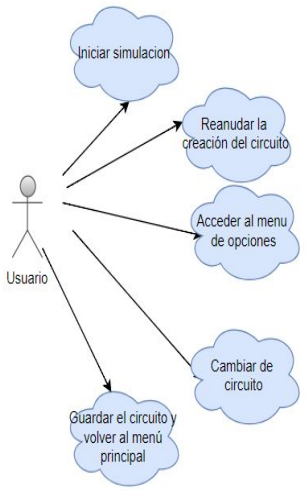


**Figura 7**: diagrama UML. Caso de uso menú pausa

### 3.3.3.2. Caso de uso simulación

Una vez iniciamos la simulación desde el caso de uso anterior accedemos al siguiente caso de uso:

**Modificar parámetros:** el usuario puede modificar parámetros de la simulación.

**Pausar la acción:** el usuario regresa a la pantalla de pausa, es decir regresa al caso de uso anterior.

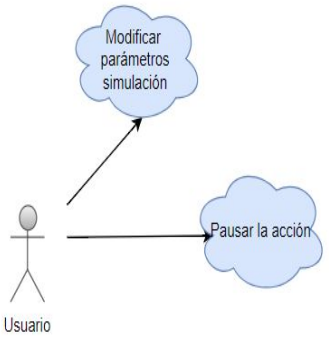


**Figura 8**: diagrama UML. Caso de uso simulación

### 3.3.4. Caso de uso menú de selección de circuito

El usuario tendrá los dos siguientes casos de uso cuando esté seleccionando un circuito:

**Seleccionar un circuito:** el usuario puede seleccionar un circuito de los que le aparecen en pantalla, lo que automáticamente le envía a la pantalla de creación de circuito del circuito seleccionado.

**Volver:** regresa a la pantalla anterior.

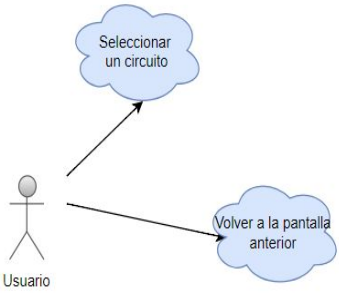


**Figura 9**: diagrama UML. Caso de uso menú selección

## 3.4. Identificación y análisis de las soluciones posibles

En este apartado se identificarán y analizarán las diferentes soluciones posibles, que se pueden elegir a la hora de elaborar una solución que cumpla con los requisitos establecidos.

Al ser una aplicación grafica, la elección más importante y la que más peso lleva a la hora decidir la implementación de la misma, es la elección de una API gráfica o directamente un motor gráfico. Ya que dependiendo de esta elección se deberá de trabajar con el entorno de desarrollo más a adecuado a la misma. Tras una investigación de las virtudes y desventajas de cada uno se seleccionan como posibles candidatos:

1. **OpenGL:** Una API gráfica que lleva siendo vigente desde hace más de 20 años, lanzada en 1992 permite el dibujado de de escenas gráficas complejas a partir de sus más de 200 funciones de dibujado, y es multiplataforma
2. **Vulkan:** es una API desarrollada por los creadores de OpenGL fue anunciada en 2015 y cuenta con unas características muy similares a OpenGL pero con un rendimiento mucho mayor
3. **DirectX:** El competidor directo de OpenGL, es una colección de API orientadas tanto para el dibujo gráfico como para el tratado de audio, el procesamiento del teclado, etc. Esta colección se encuentra disponible exclusivamente para la plataforma de Windows
4. **Unity:** Un motor gráfico reconocido por la sencillez de su uso, este puede utilizar tanto DirectX como OpenGL. Utiliza los lenguajes C#, Javascript y Boo
5. **Unreal Engine:** Un motor gráfico que utiliza DirectX como librería de dibujado por defecto, este se puede forzar a trabajar con OpenGL. Utiliza el lenguaje C++ y uno propio Unreal script

## 3.5. Solución propuesta

La opción adoptada para elaborar la solución es OpenGL, más específicamente el uso de la librería FreeGLut de la cual se darán detalles en un capítulo posterior.

OpenGL ha sido seleccionado, ya que al sopesar la posibilidad de elegir entre trabajar con la propia API gráfica o un motor gráfico y teniendo en cuenta que el objetivo principal de este trabajo es el desarrollo personal y el aprendizaje la elección de un motor gráfico facilitaba demasiado la implementación del proyecto.



Finalmente se ha elegido OpenGL debido a la experiencia previa con el mismo, de lo contrario se habría optado por Vulkan que pese a ser una API muy reciente y aunque cuente con menos información tras la misma tiene mayor potencial que OpenGL. Por otra parte si comparamos OpenGL o Vulkan con DirectX. OpenGL es más conveniente ya que es multiplataforma lo que facilita el desarrollo de una aplicación más portable.

# 4. Diseño de la solución

En este capítulo se presentarán los detalles del diseño, las decisiones sobre los mismos y el proceso de diseño de la aplicación final, en diferentes niveles de detalle.

## 4.1. Arquitectura del Sistema

La arquitectura de software elegida para la implementación de la solución es la arquitectura de estados de juego o también conocido como diseño por patrón de estados.

Este tipo de diseño es utilizado a la hora de desarrollar videojuegos, consiste fundamentalmente en la interacción entre dos componentes principales, los diferentes estados y el motor de estados.

### 4.1.1. Estado

Un estado es una clase que define una serie de funcionalidades tales como el dibujado de la misma, el manejo de eventos, la pausa, la inicialización del mismo, o el cambio a otro estado, también tiene la capacidad de interactuar con el motor de estados.

Por ejemplo: El estado que se corresponda a un menú, contará con la función que dibuje los elementos del mismo, las funciones que se encarguen de procesar la entrada de los periféricos, etc.

Cada estado sólo puede estar ocurriendo una vez, por lo que su implementación se basará en un diseño singleton.

### 4.1.2. Motor de estados

El motor de estados, también conocido como Engine o Game Engine, se trata de un objeto o una clase que contiene la pila de los estados por los que ha pasado la aplicación.

Las funcionalidades que este desempeña y algunas de sus características son las siguientes:

1. Permite modificar la pila de estados añadiendo o eliminando el último estado de la pila
2. Mantiene la aplicación en funcionamiento, una vez se detiene la ejecución del motor de estados la aplicación se detiene
3. Puede llamar a los diferentes métodos de un estados, así como limpiar el estado una vez ha sido eliminado
4. Permite la total navegación entre los estados

Como se puede esperar por las funciones que puede desempeñar, básicamente es una clase con métodos para el control de la pila, esta clase no tiene ningún elemento de dibujado ni ningún modelo.

### 4.1.3. Interacción entre los componentes.

Tanto los estados como el motor de estados interactúan entre sí para ir modificando los diferentes estados en la pila, se elimina el estado de la cima, se añade un estado, etc. Esta interacción se podría resumir con el siguiente diagrama.

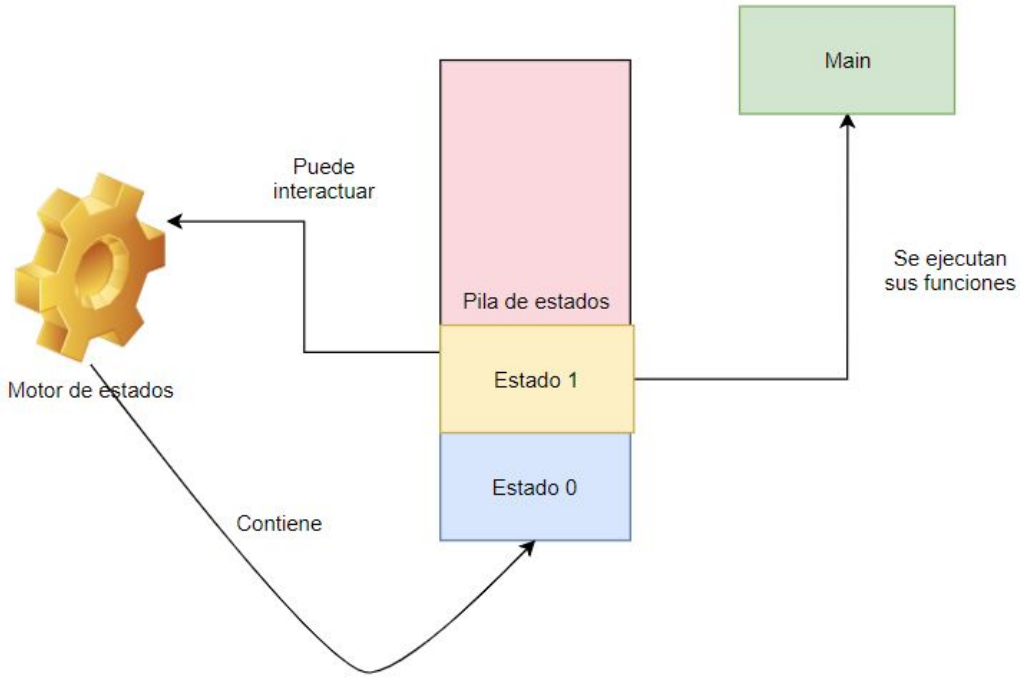


**Figura 10**: diagrama UML interacción componentes

Este tipo de diseño por estados es muy utilizado en videojuegos por la gran versatilidad que aporta, ya que permite diferenciar en qué estado se encuentra la aplicación y a que estados puede acceder en cada momento. Facilitando el diseño de lo que puede ocurrir en cada escena.

Esta forma de interactuar entre los componentes se suele extender a todos los aspectos de un juego, es decir podríamos tener varios motores de estado, cada uno para cada objeto, como por ejemplo para el jugador, en cada uno de ellos se almacenarían los estados correspondientes de dicho objeto en específico.

Para el diseño de esta aplicación se ha utilizado únicamente el concepto global por lo que solo se tendrá un motor de estados que controlará toda la aplicación.

Dejando claro este tipo de diseño queda claro que clases se tendrán que declarar e implementar en el proyecto.

## 4.2. Diseño de clases

Una vez determinada la arquitectura a seguir y los requisitos funcionales de la aplicación podemos determinar que necesitaremos las siguientes clases:

**Tramos**: se necesitará implementar una clase para definir los diferentes tramos que se puedan dibujar, implementando una interfaz o clase padre y haciendo que las variaciones de tramo hereden de dicha clase.

**Cámara**: si se va a implementar una cámara con sus propias funcionalidades será recomendable crear un clase cámara donde describir todos los atributos y funciones que esta requiera, de esta forma se podrán crear diferentes subclases cámara para modificar la funcionalidad de cada una.

**Motor de estados**: para poder crear la pila de estados será necesario implementar una clase que la contenga y maneje todos los cambios entre estados.

**Estados**: será necesario definir una clase interfaz, de la que el resto de estados puedan heredar, de esta forma se podrá implementar diferentes estados, cada uno con diferentes funcionalidades.

**Point3D:** clase de apoyo que se utiliza para simular un punto 3D en el espacio y facilitar el acceso a los atributos x,y,z. En definitiva en simplemente una forma diferente de acceder a un array.

**Entorno**: clase de apoyo utilizada para dibujar el entorno 3D de la escena de creación del circuito.

Si analizamos el siguiente diagrama uml:

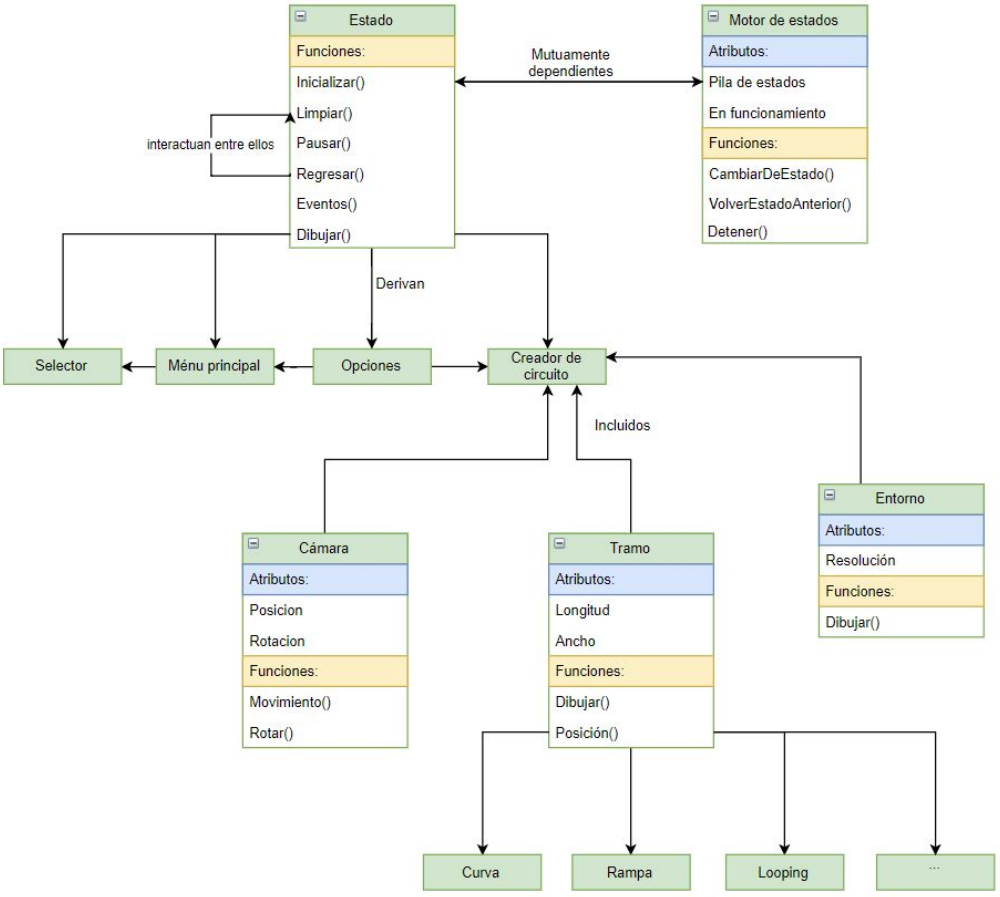


**Figura 11**: diagrama UML de las clases

Las clases **Estado** y **Motor de estados** son mutuamente dependientes pues el **Motor de estados** debe contener la pila de **estados**, y a su vez los **estados** deben de ser capaces de llamar a los métodos de la clase **Motor de estados** para poder navegar entre **estados**. Se inyecta la dependencia por métodos.

A su vez dependiendo del Estado este podría incluir clases como la **Cámara** o las diferentes clases derivadas de **Tramo** o **Entorno**, como podría ocurrir en el estado de creacion de circuito.

En definitiva donde verdaderamente recae el peso en cuanto a interacciones será entre las diferentes variantes de la clase estado y el propio motor de estados.

## 4.3. Diseño gráfico

En este apartado se tratarán las diferentes decisiones tomadas en cuanto al apartado gráfico se refiere, se presentarán bocetos de las diferentes interfaces, efectos, y

modelados que se quieren implementar en la solución final. Se tendrá en cuenta que se quiere seguir un diseño parecido al de un videojuego donde toda la información importante aparece en pantalla de forma práctica e intuitiva.

### 4.3.1. Menús

Siendo un requisito clave que el uso de la aplicación sea sencillo se optará por realizar el siguiente diseño de interfaz para el usuario en los diferentes menús dentro de la aplicación:

En el esbozo que se puede apreciar a la derecha, se puede observar que la primera idea sería crear menús con botones que reflejen claramente cual está siendo seleccionado para que el usuario tenga claro qué opción va a seleccionar.

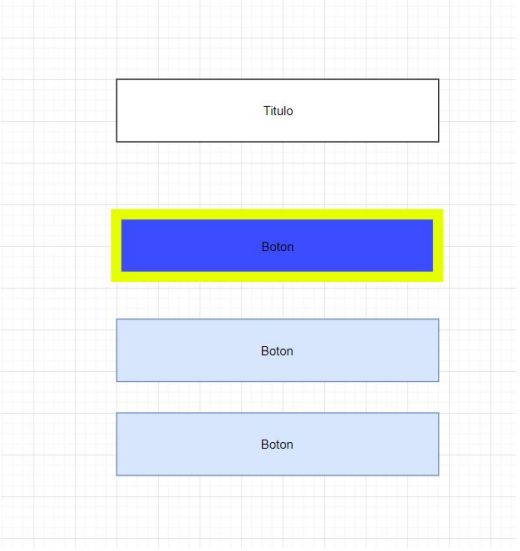


**Figura 12**: esbozo de un posible menú.

Siguiendo el esbozo anterior se diseñarán cada uno de los diferentes menús, teniendo en cuenta las particularidades de cada uno.

#### 4.3.1.1. Menú principal

El menú principal se muestra al iniciar la aplicación, y permite la navegación a través de la misma.

El diseño del menú principal sería muy cercano al diseño presentado de la figura 12, se mostrarían las opciones de nuevo circuito, seleccionar un circuito ya creado, entrar al menú de opciones y salir de la aplicación.

Se resaltaría vistosamente que botón tenemos seleccionado y a través del teclado se podría acceder a las diferentes opciones que se muestran en pantalla.

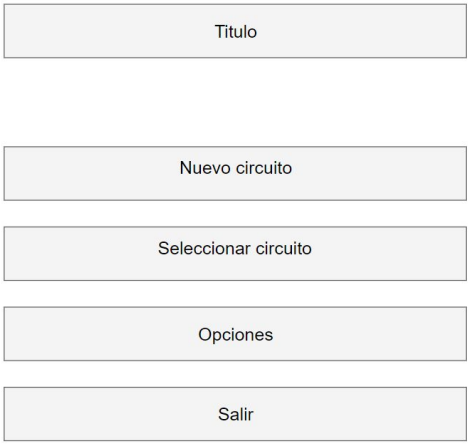


**Figura 13**: diseño del menú principal

#### 4.3.1.2. Menú opciones

El menú de opciones mostrará las opciones de configuración de la aplicación de forma sencilla y clara.

El menú se distribuirá en filas.

La primera mostrará los diferentes niveles de detalle gráfico seleccionables.

La segunda mostrará un check o algún artefacto visual similar para dar la opción de activar y desactivar la pantalla completa.

Finalmente se muestra un botón para guardar y volver al estado anterior.


**Figura 13**: diseño del menú opciones

Este menú podría ser ampliado con submenús y secciones para profundizar más en las diferentes opciones gráficas.

#### 4.3.1.3. Menú selección circuito

El menú de selección de circuito sería algo diferente a los anteriores, pues este tendría que mostrar todos los circuitos almacenados en disco, lo que podría ser imposible debido al espacio finito de dibujado que existe en la pantalla, por dicha razón se ha escogido el siguiente diseño

El diseño consistiría en una parte central en la pantalla donde se representan los circuitos almacenados en disco para resolver el problema del espacio se utilizará una paginación de los mismos, mostrando una cantidad limitada de circuitos por página. Se permitiría avanzar y retroceder de página de manera automática.

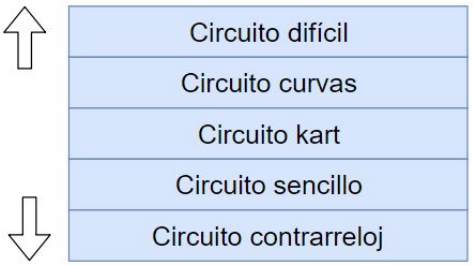


**Figura 15**: diseño de la pantalla de selección

### 4.3.2. Creación del circuito

En este apartado se revisarán los distintos elementos gráficos que forman la escena de creación del circuito y el diseño escogido para cada uno de ellos.

#### 4.3.2.1. Conceptos

**Cámara libre o flotante** es una camará que se mantiene en el aire y permite al usuario moverse libremente por el espacio de dibujo, permitiendo enfocar cualquier punto de la escena.

#### 4.3.2.2. Piezas

Las piezas se representarán como octaedros, aportando volumen al circuito y simulando ser piezas de juguete, estás se representarán cohesionadas y sin fisuras ni juntas.

Cuando el usuario esté seleccionando una pieza se irá dibujando la pieza según los parámetros que el usuario tenga seleccionados, de esta forma el usuario tendrá feedback visual de las modificaciones que está haciendo. Una vez termine las modificaciones la pieza se añade al circuito y se queda dibujada con los últimos parámetros seleccionados.

#### 4.3.2.3. Entorno

El entorno donde se realizará la construcción del circuito simulará una habitación.Se dibujará un cubo gigante que englobe todo el espacio de trabajo y las piezas se colocarán sobre una mesa tridimensional dibujada en el centro.

De esta forma se creará cierta inmersión, haciendo parecer que se está jugando con piezas dentro de una habitación, ayudando a que el usuario pueda disfrutar de una experiencia relajada.

#### 4.3.2.4. Interfaz

El proceso de crear un circuito debe de ser intuitivo y fácil de comprender, para ello se ha optado por un diseño de interfaz simple y minimalista que se mostrará como un overlay durante el proceso de creación. Dicho overlay contará con dos elementos principales:

El primero lo podríamos denominar una cinta de selección, donde el usuario recibirá información sobre el elemento que tiene seleccionado y le permitiría desplazarse entre diferentes elementos.

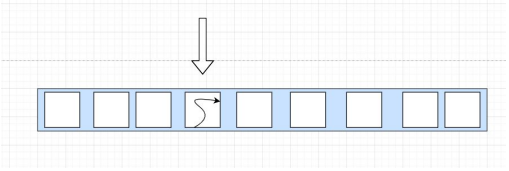

**Figura 16**: esbozo de un elemento selector

El segundo sería un área determinada de la pantalla, donde se mostraría información relevante para ayudar al usuario, tal como sugerencias, teclas que pueda pulsar o las acciones que estas desencadenan. Avisos tales como el exceso de piezas en el circuito, etc.

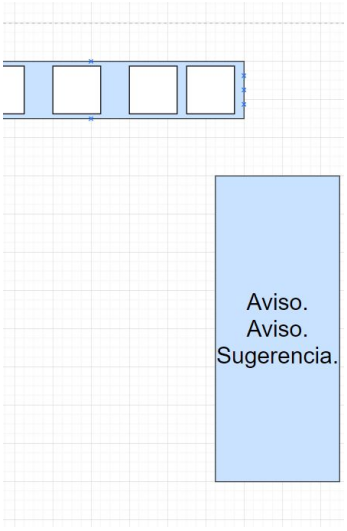


**Figura 17**: esbozo de un tablón de información

Con estos dos elementos de interfaz el usuario recibiría información tanto del estado del circuito como de las acciones que puede realizar. Adicionalmente al ser un overlay el usuario tendría el campo visual libre para poder visualizar el circuito. Para la facilitar visualización del circuito, se definiría una cámara libre para que el usuario tenga libertad para observar el circuito desde cualquier ángulo.

#### 4.3.2.5. Menú pausa

El menú de pausa se mostrará tras ocultar la interfaz, este se mostrará al igual que la propia interfaz como un overlay sobre la escena dibujada el propio esquema del menú será casi idéntico al menú principal el cual se puede observar en la figura 13, solo se añadirán las opciones pertinentes a iniciar la simulación y reanudar la construcción del circuito.

## 4.4. Diseño estructural

En este apartado se describe la estructura de carpetas de la que se hará uso durante la implementación, detallando la separación de cada uno de los diferentes tipos de recursos.

**Carpeta principal**: contendrá el código fuente de la aplicación o el ejecutable una vez compilada y las subcarpetas descritas a continuación.

**Carpeta de guardado**: contendrá los circuitos que se guarden en disco, también tendrá incluida una subcarpeta con el fichero de opciones, que se utilice para cargar la configuración de la aplicación.

**Carpeta de texturas**: contendrá diferentes subcarpetas donde se almacenarán las imágenes, que se usarán como textura para las superficies de cada uno de los estados. De esta forma cada estado tendrá su propia subcarpeta y los recursos de cada uno estarán separados.

Siguiendo esta estructura de carpetas será sencillo mantener organizado cada uno de los diferentes recursos que se van a necesitar para la implementación de este proyecto.

# 5. Tecnología Utilizada

En este capítulo se describen las tecnologías utilizadas durante el desarrollo de este proyecto. Así como el lenguaje de programación, las distintas librerías utilizadas, el IDE donde se ha realizado la implementación del proyecto, y el sistema de control de versiones que se ha usado de soporte durante el desarrollo.

## 5.1. El lenguaje C++

El lenguaje C++ fue lanzado en 1983, apareció como una evolución natural del lenguaje C, con la inclusión del paradigma de la programación orientada a objetos convirtiéndolo en un lenguaje híbrido, pues incluía características tanto del paradigma de la programación estructurada como de la programación orientada a objetos.

Más tarde se le añadieron elementos de otros paradigmas de la programación, creando así un lenguaje multiparadigma.

Algunas de las características más destacables del lenguaje C++ son:

1. Es un lenguaje potente, tiene una gran variedad de tipos de datos, estructuras de control, etc
2. Es un lenguaje dependiente de la máquina donde se realiza la compilación
3. Los ficheros fuente de C++ pueden compilarse en cualquier maquina sin realizar muchos cambios(Gran portabilidad)
4. Es un lenguaje de Alto nivel, que puede manejar funciones de medio-bajo nivel, lo que lo convierte en un lenguaje eficiente
5. La capacidad de usar eficientemente punteros a la memoria
6. Tiene una gran colección de librerías con diferentes funcionalidades
7. Permite la compilación independiente de cada uno de sus módulos

Todas estas características convierten a C++ en un lenguaje extenso, flexible y eficientes y por ello mismo gran parte de las aplicaciones gráficas se basan en este lenguaje.

## 5.2. Librerías Utilizadas

En este apartado revisaremos las diferentes librerías utilizadas para la implementación de este proyecto

### 5.2.1. Conceptos

**Z-Buffer:** Es un buffer de datos donde se distribuye la profundidad de cada uno de los pixeles en pantalla, es usado para reproducir la sensación de profundidad, así como diversos procesos para la eliminación de los puntos que no se ven, ya sea porque están tapados o están fuera del alcance de la visión.

**Pipeline o tubería de renderizado:** En el ámbito de la computación gráfica una pipeline es el conjunto de procesos por el que pasa la escena tridimensional vectorial(el conjunto de vértices y puntos en un espacio 3D) para acabar siendo representado en un espacio 2D(la pantalla).

**Mapeado de texturas**: es el proceso por el cual se establece cómo se sitúa una textura sobre un objeto.

**Rasterización**: es el proceso en el que un objeto vectorial pasa a representarse en un espacio 2D, este proceso forma parte de la pipeline.

**Alpha blending:** es la capacidad de representar objetos con transparencias.

### 5.2.2. OpenGL

OpenGL es una especificación estándar que define una API gráfica para el dibujado 2D y 3D, esta especificación contiene definiciones para más de 250 funciones y multitud de primitivas gráficas simples como triángulos, cubos, esferas. Su principal funcionalidad es facilitar la complicada interacción con las diferentes tarjetas gráficas y proporcionar al programador una API uniforme.

Algunas características de OpenGL son:

1. Contiene varias primitivas básicas de puntos, líneas y polígonos rasterizados. Es decir son primitivas ya procesadas y que han pasado desde una descripción vectorial al propio objeto dibujado, haciendo que para el programador estos pasos sean transparentes

2. Una pipeline modificable de transformación e iluminación
3. Mapeado de texturas
4. Alpha blending
5. Z Buffering

Todas estas características convierten a OpenGL en una API muy potente para el desarrollo de cualquier aplicación gráfica.

A la hora de desarrollar la solución habrá que tener en cuenta el procesamiento del bucle principal de OpenGL y el hecho de que al no ser un motor gráfico, no cuenta con la capacidad de identificar qué vértices forman parte de un objeto concreto, OpenGL simplemente acumula todos los vértices que vayamos a dibujar y los pasa a través de la pipeline. Estas restricciones se tendrán en cuenta durante la implementación de la aplicación.

### 5.2.3. FreeGlut

FreeGlut es una librería que se encarga de agregar portabilidad a OpenGL haciéndose cargo de la inicialización de las ventanas de dibujado, del el control de los eventos de entrada, etc.

Debido a la necesidad de adaptarse a varias plataformas FreeGlut modifica el comportamiento de la pipeline de OpenGL, FreeGlut fija la pipeline, imposibilitando realizar cualquier cambio a los procesos que está ocurren, y simplificando cómputo de la iluminación. Esta característica es un punto a favor y en contra pues simplifica el desarrollo de la aplicación, ya que no es necesario implementar shaders(pequeños fragmentos de código que computan efectos gráficos), pero por otra parte elimina toda posibilidad de implementarlos.

Gracias a la gran portabilidad que FreeGlut aporta facilitará el desarrollo del proyecto y su posible integración en otras plataformas.

### 5.2.4. FreeImage

Cabe destacar el uso de FreeImage una sencilla librería para el procesamiento de imágenes, que se ha utilizado en este proyecto para la carga de texturas desde imágenes de diversos formatos. Facilitando el hecho de no tener que generar un formato de imagen específico, para la lectura de imágenes.

### 5.2.5. GLM(OpenGL mathematics)

La librería GLM(OpenGL mathematics) es una librería matemática especializada en los cálculos matriciales, especialmente en los cálculos orientados a las diferentes transformaciones que puede sufrir una matriz en una implementación gráfica.

A la hora de facilitar la implementación de algunos cálculos esta librería es de gran ayuda.

## 5.3. Visual Studio

Visual Studio es el IDE elegido para la implementación de este proyecto, ha sido elegido por sus numerosas cualidades a la hora de desarrollar proyectos en C++.

1. Cuenta con un gran autocompletado de código que acelera la escritura del mismo
2. Tiene un sistema de depuración muy completo
3. Posee un módulo centrado en el desarrollo gráfico y de videojuegos que ayuda al control de los recursos
4. Está especialmente diseñado para funcionar en windows(plataforma en la cual se desarrolla el proyecto)
5. Facilita la navegación entre los diferentes ficheros abiertos.
6. Cuenta con sistemas búsqueda, reemplazado y refactorización avanzados
7. Posibilidad de mantener entornos de producción separados, por arquitectura o por estado del proyecto
8. Ofrece integración con git para facilitar el control de versiones

Dadas las características del proyecto Visual Studio encaja a la perfección con el mismo. Siendo la herramienta perfecta para la realización del mismo.

## 5.4. Control de versiones con GitHub

Todo proyecto que no sea trivial requiere de un control de versiones, ya sea de forma manual o automática, es necesario tener un historial de los cambios realizados y la posibilidad de volver atrás en cualquier momento, para así poder evitar errores introducidos durante los cambios realizados.

Hay diversas opciones para realizar este control de versiones, puede ser desde local, copiando manualmente los ficheros hasta un control Git utilizando algunos de los conocidos dominios que se pueden encontrar en internet.

¿Que es Git? Git es un software de control de versiones que automatiza todo el proceso, desde saber quien ha hecho las modificación hasta la posibilidad de separar y mezclar diferentes ramas de un desarrollo. Por eso mismo, para llevar un control de versiones en este proyecto, se ha elegido la utilización de Git como sistema de control de versiones.

Una vez elegida la opción de utilizar Git, el abanico de dominios que brindan repositorios online, para el control de versiones es realmente amplio, GitHub, BitBucket, GitLab. De entre todos ellos me he decantado por GitHub por la experiencia personal en su uso, ya que pese a no permitir la creación de repositorios privados sin pagar, al ser este un proyecto académico no era una necesidad la privacidad del mismo.

# 6. Desarrollo de la solución propuesta

En este apartado se detallarán las técnicas más relevantes, que se han utilizado durante la implementación de la solución y se mostrarán los aspectos finales de las escenas desarrolladas.

## 6.1. Conceptos

**Matriz de transformaciones:** matriz 4x4 donde se almacenan todas las transformaciones realizadas, translaciones, rotaciones y escalados.

**Proyecciones:**

Existen dos tipos de proyección de la escena sobre el punto de vista, estos son la proyección en perspectiva y la proyección ortográfica.

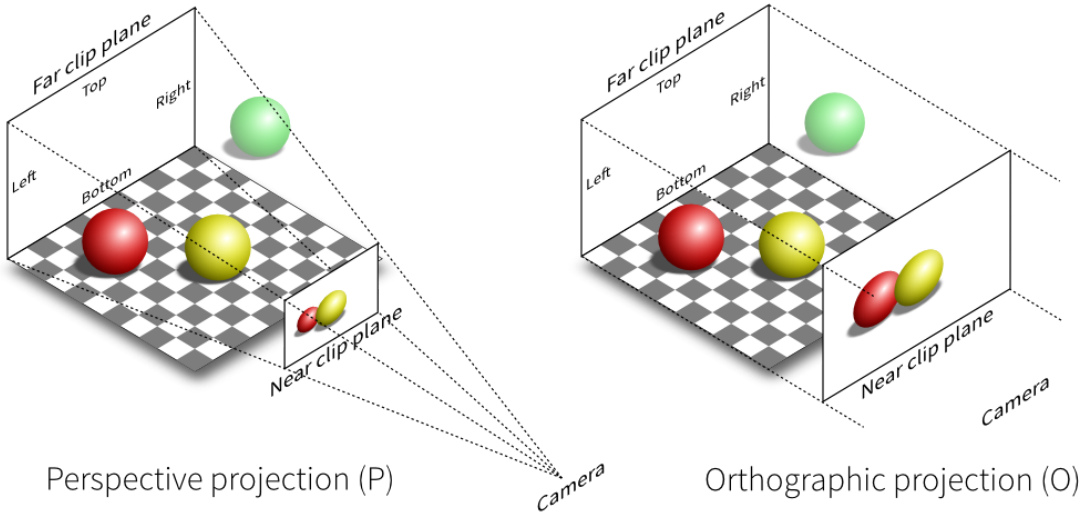


**Figura 18**: Imagen de los distintos tipos de proyección

La proyección en perspectiva que podemos observar a la izquierda en la figura 18, otorga profundidad a la escena y la dibuja simulando la visión de un ojo. Esta se define por parámetros como el ángulo de visión conocido como fov(field of view), el ratio entre la altura y la anchura, y los parámetros de cercanía y lejanía que determinan hasta dónde llega la vista. Este tipo de proyección se utiliza en escenas 3D donde la

demostración de profundidad es importante o bien la acción o movimientos de la cámara se realizan en el espacio 3D.

La proyección ortográfica la cual se muestra a la derecha, carece de profundidad y todos los objetos aparecen dibujados a la misma distancia, haciendo que sea imposible distinguir entre dos objetos del mismo tamaño, cuál se encuentra más cerca o más lejos. Para definir una proyección ortográfica necesitamos definir el plano de la proyección con las medidas que elijamos, en el caso de la computación gráfica se suele definir un octaedro cúbico con unas medidas fijas. Este tipo de proyección se utiliza para representar escenas 2D donde no existe la necesidad de representar una profundidad real.

Concluyendo, la proyección en perspectiva se utilizará para representar una escena 3D y la proyección ortográfica se usará para representar objetos 2D.

## 6.2. Menús e interfaces

La técnica utilizada para el dibujado de los menús y las diferentes interfaces es una técnica estándar, en el ámbito de la computación gráfica, esta consiste en establecer una proyección ortográfica sobre la escena y dibujar los elementos pertenecientes a la interfaz durante el uso de dicha proyección, si se requiere una vez dibujados los elementos 2D en una posición fija, se cambiará a la proyección en perspectiva para dibujar los elementos de la escena 3D.

Siguiendo la técnica comentada generando una proyección ortográfica sobre la que colocar los diferentes elementos, se implementaron las interfaces y los menús utilizados en la aplicación.

### 6.2.1. Menú principal

Tras seguir los pasos establecidos se llegó a implementar la siguiente escena como menú principal:

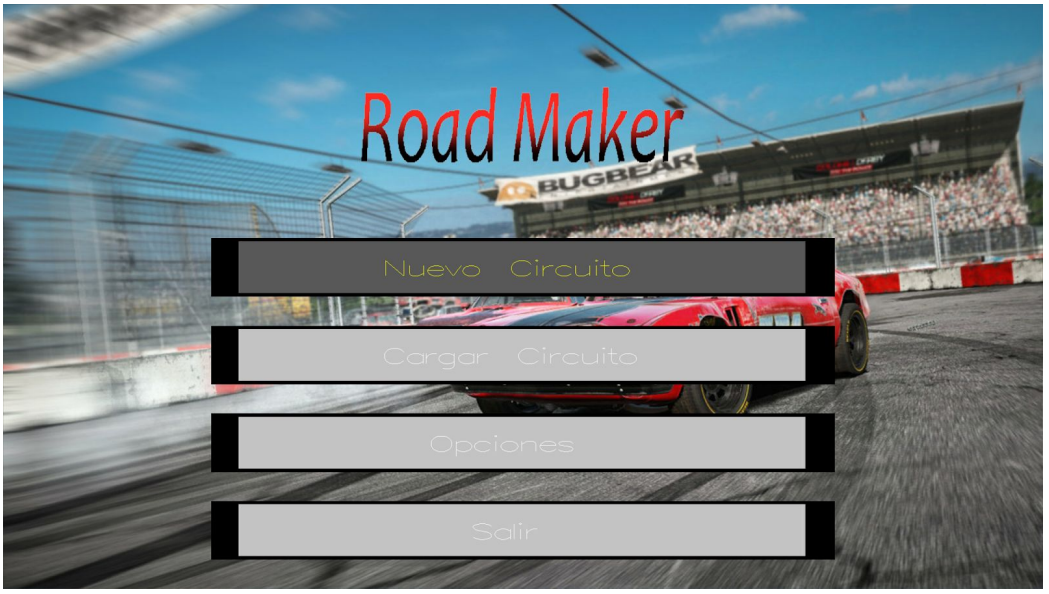


**Figura 19**: Imagen del menú principal implementado

Como se puede apreciar en la figura, se siguió el diseño establecido casi sin modificaciones, se puede observar el botón donde se encuentra el puntero con un color oscurecido, también se puede observar el texto que describe la acción de cada botón. Al pulsar la tecla enter se accede al estado relacionado con el botón seleccionado.

### 6.2.2. Menú opciones

Al igual que con el menú principal se siguieron las técnicas descritas y el diseño original para llegar a implementar la siguiente escena:


**Figura 19**: Imagen del menú de opciones implementado

Al igual que ocurría en el menú anterior se puede observar que se muestra resaltado el botón que se tiene seleccionado, así como la opción gráfica que está establecida actualmente(flecha sobre la opción), también se puede apreciar un pequeño cambio en el diseño, pues originalmente se pretendía dejar un checkbox para alternar pantalla completa y ventana, pero por conseguir mayor claridad visual consideré dejar dos botones de selección. Al pulsar enter se aplican los cambios de la selección actual.

### 6.2.3. Pantalla de carga de circuitos

En esta pantalla tal y como se describió anteriormente se muestran todos los circuitos almacenados actualmente en la carpeta correspondiente. Estos se paginan para poder mostrar un número potencialmente infinito de circuitos, se muestra la página actual y el número total de páginas en pantalla. Al pulsar enter se cargará el circuito y pasaramos a la pantalla de edición del circuito.

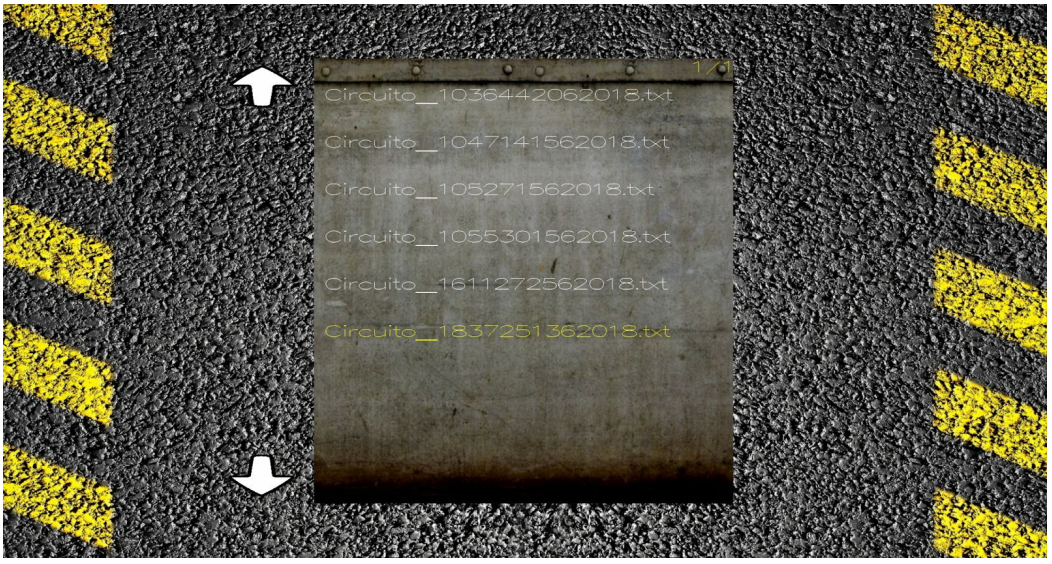


**Figura 20**: imagen de la pantalla de carga de circuitos

### 6.2.4. Interfaz de edición

La interfaz que se muestra durante la creación del circuito se dibuja primero sobre un proyección ortogonal, la cual se cambia a una proyección en perspectiva para el dibujado de los objetos 3D que pueblan la escena.

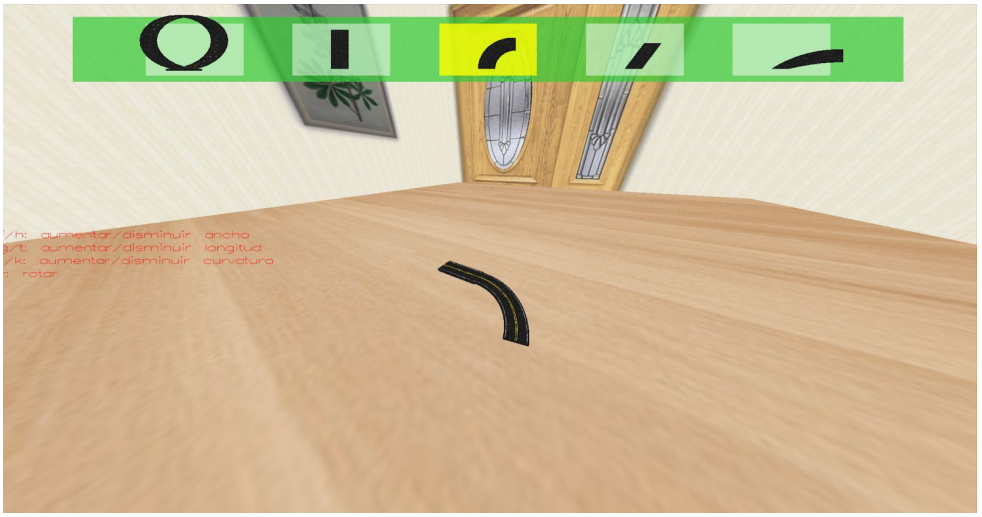


**Figura 21**: imagen de la pantalla de edición de circuitos implementada

Si observamos la figura 21 en ella se muestra la cinta de selección con 5 elementos simultáneos, estos se van desplazando según indique el usuario con las flechas de dirección. La pieza que se muestra en el centro de la misma es la pieza que se puede seleccionar, esta está resaltada por un tono de blanco menos transparente que en las otras piezas. Una vez se pulsa el botón enter la cinta se bloquea y se resalta en color amarillo la pieza seleccionada, a partir de ese momento se pueden modificar los parámetros de las piezas.

A la izquierda de la pantalla se muestran las teclas que se pueden pulsar y los parámetros que estas modifican. Durante la implementación se decidió cambiar la ubicación de los mensajes ya que el ojo humano visualiza las imágenes de izquierda a derecha, al colocar los mensajes a la izquierda se facilita su lectura.

## 6.3. Cámara libre

La implementación de la cámara libre está basada en los ángulos de Euler, ideado por Leonhard Euler consistente en un conjunto de 3 coordenadas, que sirven para determinar la orientación de un sistema de referencia, en este caso el sistema de referencia de la cámara.

Se define de la siguiente forma:

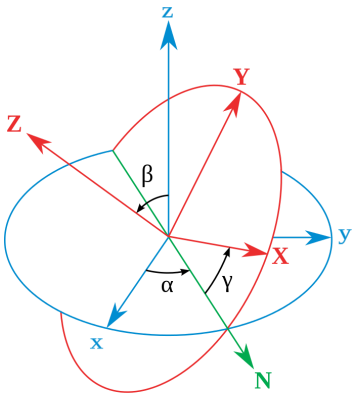


**Figura 21**: imagen representativa ángulos de Euler

Dados los dos sistemas de coordenadas que se muestran en la imagen xzy y XZY se puede calcular la posición del sistema mediante 3 ángulos, estos ángulos se corresponden a la rotación de cada uno de los ejes x, y, z y determinan el lugar hacia el que estará mirando la cámara.

En el ámbito de las cámaras en aplicaciones gráficas a estos ángulos se les conoce como yaw, pitch y roll, que se corresponden con los ejes x, y, z para la construcción de la cámara se utilizarán únicamente los ángulos yaw y pitch, ya que no se requiere de una rotación completa.

Durante la implementación se tuvo en cuenta que el ángulo pitch no fuera ni mayor de 90 grados ni menor de 90 grados, de esta forma se evita que la cámara se vuelque, dejando la escena del revés.

## 6.4. Simulación

Para poder realizar la simulación era necesario construir un conjunto de puntos a través del circuito, para interpolar las posiciones del móvil que representa al vehículo. Dado que OpenGL no es un motor gráfico, este no reconoce los diferentes objetos que se presentan en una escena como tales, los vértices de los mismos pasas a través de la pipeline y se dibujan, y el sistema se “olvida” de los mismos.

Teniendo en cuenta esto último fue necesario añadir 2 atributos nuevos a las clases de los tramos, uno para almacenar la matriz de transformaciones inicial y la final, con esta nueva información podía realizar los cálculos pertinentes para obtener los diferentes puntos necesarios para realizar la simulación. Una vez calculados los puntos, la simulación se realiza interpolando mediante una interpolación simple basada en el tiempo los diferentes puntos que forman el camino, obteniendo así la posición del móvil en el tiempo.

# 7. Implantación y pruebas

En este apartado se explicarán los pasos necesarios para poner en marcha la solución final, y se detallarán las pruebas realizadas con varios usuarios.

## 7.1. Implantación de la aplicación

Gracias a las tecnologías seleccionadas, la aplicación es totalmente portable dentro de la plataforma donde se haya realizado la compilación de la misma, esto ha sido probado durante las diferentes pruebas con los usuarios detalladas en el siguiente apartado.

He preparado un paquete de ejecución, que he dejado subido a mi repositorio(enlace en el apéndice A) con el nombre de ejecutable, en el paquete se encuentran siguiendo la estructura de carpetas descritas en el capítulo 4, los ficheros necesarios para la ejecución de la aplicación, incluyendo las imágenes usadas para las texturas, o el propio archivo .exe junto con los archivos dll de los que hace uso la aplicación.

Gracias a esta estructura, cualquiera que intente ejecutar el archivo .exe desde una máquina windows, será capaz de utilizar la aplicación sin necesidad de instalar ninguna librería ni aplicaciones adicionales.

## 7.2. Pruebas con el usuario

Para determinar si la aplicación respondía tal y como se esperaba, para buscar mejoras en general o mejorar la interactividad, he realizado pruebas con 5 compañeros distintos. A cada uno le he enviado el paquete de instalación que he creado, para comprobar que funcionase perfectamente en todos los dispositivos con windows.

Tras comprobar que no había problemas de incompatibilidad, y que han podido ejecutar la aplicación sin problemas, he procedido a hacerles las siguientes preguntas:

1. ¿La aplicación responde como se espera?
2. ¿Hay alguna interacción que no resulte intuitiva?
3. ¿Puedes recordar fácilmente qué interacciones existen dentro de la aplicación?
4. ¿Incluirías alguna mejora considerada de urgencia crítica?
5. ¿Encuentras algún aspecto de la interfaz mejorable?

De las respuestas obtenidas he podido extraer la siguiente información:

La aplicación responde correctamente en la mayoría de los casos, las interacciones resultan intuitivas aunque me indicaron que un pequeño tutorial haría perfecto el aprendizaje de las mismas, una vez comprendidas las posibles interacciones resultan fáciles de recordar, nadie encontró ningún apartado donde aportar una mejora crítica, que necesitase urgentemente la aplicación.

En cuanto a los aspectos a mejorar de la interfaz hubo un caso esclarecedor y de gran ayuda donde un compañero, que parece daltonismo, más concretamente sufre deuteranopia lo que le impide distinguir correctamente el rojo del verde. Me indicó que, no podía distinguir correctamente las letras, que usaba para describir las acciones posibles durante la creación del circuito. Esto mismo me hizo replantearme distintos apartados estéticos de la aplicación, modificando los colores de varios puntos de la interfaz para evitar situaciones similares en un futuro.

# 8. Conclusiones

A lo largo de la realización de este trabajo he aprendido la complejidad de la implementación de una aplicación real, era la primera vez que implementaba una aplicación desde cero, haciendo cada uno de los pasos de diseños necesarios para desarrollarla.

Como alumno de la rama de computación, ya había estudiado sobre OpenGL y las librerías gráficas, en mi caso tanto durante el estudio de la asignatura de introducción a los sistemas gráficos interactivos en la UPV, donde aprendí las bases de las mismas, como posteriormente durante mi estancia Erasmus en Upsala, donde cursé una asignatura de gráficos avanzados a nivel de máster, donde aprendí técnicas avanzadas como la creación de shaders.

Al ser la primera aplicación que realizaba de una extensión considerablemente mayor a las anteriores, tuve varios problemas relacionados con la implementación gráfica de los diferentes elementos, así como del propio uso de C++, lenguaje que conocía pero del cual no tenía un conocimiento en profundidad. Por ejemplo durante el desarrollo, debido a que unos objetos se estaban creando de forma dinámica, la memoria ram consumida no paraba de crecer desde el comienzo de la ejecución, motivo por el cuál tuve que modificar el comportamiento de varias funciones para evitar este problema. Este tipo de problema era totalmente nuevo para mí pues durante el grado se estudia en mayor profundidad Java o Python, donde nunca ocurrió algo similar.

Debido a la envergadura de la aplicación a desarrollar y la importancia de la misma, era la primera vez que utilizaba un repositorio Git de manera más profesional, indicando los cambios realizados y pautando el siguiente paso a realizar. Lo que al final me ayudó a centrarme y a mantener ordenado el proyecto.

A título personal me ha encantado descubrir y aprender cómo realizar una aplicación gráfica, buscar información sobre las tecnologías que se usan en su diseño, adaptando las que encontraba más interesantes a mi proyecto. El simple hecho desarrollar todo el apartado gráfico de la misma desde zero ha resultado emocionante. También he podido comprender la gran complejidad matemática que se esconde tras cada videojuego y he comenzado a ver las pautas a seguir para desarrollar un motor gráfico.

Como amante de los videojuegos y aspirante a desarrollador de los mismos, me ha encantado realizar este trabajo, pues como se puede apreciar desde el comienzo de esta memoria he tratado de enfocar el desarrollo de la aplicación como si de un videojuego se tratase, con los diferentes elemento que los hacen tan interactivos e intuitivos.

# 9. Mejoras futuras

En este último apartado repasare las mejoras y las ampliaciones que me hubiese gustado añadir al proyecto.

**Funciones online**: la inmensa mayoría de aplicaciones de hoy en día, cuenta con alguna función online, siendo una de las más demandadas la posibilidad de compartir información, sería interesante añadir una plataforma de intercambio donde los usuarios de la aplicación puedan subir sus circuitos completados, añadir comentarios y valorarlos.

**Mejorar el sistema de almacenamiento**: no solo se podría cambiar el modo en el que se almacenan los ficheros, como por ejemplo añadiendo una base de datos. También se podría crear un formato de almacenamiento propio del sistema para incrementar la seguridad.

**Añadir un tutorial**: pese a que la aplicación final ha resultado muy intuitiva, estaría bien añadir un pequeño tutorial scriptado donde se muestre al usuario las distintas acciones que puede realizar.

**Crear un motor gráfico**: se podría crear un pequeño motor gráfico con funciones básicas de cara a mejorar las posibilidades de ampliación del proyecto, como la inclusión de físicas o la mejora de rendimiento.

**Más variedad de piezas**: un aspecto que siempre es mejorable en este tipo de aplicaciones es la cantidad de contenido, añadiendo algunas piezas más o añadiendo la posibilidad de cubrir las piezas como si de un túnel se tratase, etc. Mejoraría la calidad del producto.

**Autocompletado del circuito**: se podría añadir un sistema inteligente de recomendación de piezas, que a su vez fuese capaz de completar el circuito siguiendo un estilo similar al que el usuario estaba desarrollando.

**Desarrollo móvil**: teniendo en cuenta que el mercado móvil, es el mayor mercado de consumo de aplicaciones actual, sería interesante trasladar el desarrollo a las plataformas móviles, gracias al diseño elegido no sería difícil trasladar el sistema a un dispositivo móvil.

**Mejoras generales:** tal y como se comenta en el capítulo 7 durante las pruebas con usuarios han surgido algunas posibles mejoras menores que implementar, sería interesante realizar un estudio más exhaustivo, para localizar puntos flacos mediante el feedback de los usuarios.

# 10. Referencias


Trackmania United Forever. *Nadeo*. Información obtenida de:

http://es-maniaplanet.nexway.com/juegos/otros/trackmania-united-forever-721838.html

Gran Turismo 6. Sony. Información obtenida de:

https://www.gran-turismo.com/es/products/gt6/

http://www.gran-turismo.com/es/gt6/manual/#!/coursemaker/application01

Race Track Builder y Bob's Track Builder. Brendon Pywell. Información obtenida de:

http://www.racetrackbuilder.com/

http://www.bobstrackbuilder.net

OpenGL. Khronos Group. Información obtenida de:

https://www.opengl.org/

FreeGLUT. Pawel W. Olszta. Información obtenida de:

http://freeglut.sourceforge.net/

Vulkan. Khronos Group. Información obtenida de:

https://www.khronos.org/vulkan/

Patrón de estados. Game Programming Patterns Información obtenida de:

http://gameprogrammingpatterns.com/contents.html

C+. Bjarne Stroustrup. Información obtenida de:

http://www.cplusplus.com/

Información general obtenida de:

https://www.wikipedia.org/

## 10.1. Imágenes externas utilizadas

Todas mencionadas en este apartado pertenecen a sus respectivos dueños y son utilizadas en este trabajo a modo informativo y sin la intención de apropiarse de ellas.

Figura 1. Obtenida de:

http://es-maniaplanet.nexway.com/juegos/otros/trackmania-united-forever-721838.html

Figura 2. Obtenida de:

https://www.gran-turismo.com/es/products/gt6/

Figura 3. Obtenida de:

http://www.bobstrackbuilder.net

Figura 18. Obtenida de:

https://www.script-tutorials.com/webgl-with-three-js-lesson-9/

# A. Apéndice: Código fuente.

Dado la gran extensión de código que conforma la aplicación desarrollada, en lugar de poblar esta memoria con páginas y páginas de código, dejo a continuación el enlace al repositorio donde se realizó el desarrollo, para que aquél lector interesado en comprender mejor ciertas partes de la implementación pueda satisfacer su curiosidad.

https://github.com/Obiwanko/TFG/tree/Master