universidad de león

Escuela de Ingenierías I. I.

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Desarrollo de videojuego con Unity y Reinforcement Learning: TRON

Autor: Samuel Lebrero Alonso

Tutor: Lidia Sánchez González

UNIVERSIDAD DE LEÓN

Escuela de Ingenierías I.I.

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

ALUMNO: Samuel Lebrero Alonso

TUTOR: Lidia Sánchez González

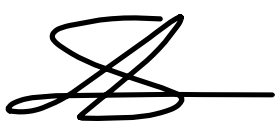
TÍTULO: Desarrollo de videojuego con Unity y Reinforcement Learning: TRON

CONVOCATORIA: Diciembre, 2021

RESUMEN:

En este trabajo se desarrolla un videojuego inspirado en la película de ciencia ficción *TRON* en el que se han empleado varias tecnologías diferentes que son habituales en la industria del videojuego como Unity y Blender; y además se explora la idea de emplear machine learning en vez de las técnicas usuales para crear las inteligencias artificiales que emplea un juego y el proceso a seguir para entrenarlas.

Palabras clave: Videojuego, Inteligencia artificial, Unity, Machine learning, Reinforcement learning

Firma del alumno:	**VºBº Tutor:**
	

I. Tabla de contenido

II. Tabla de Ilustraciones

III. Glosario

C

C++: Lenguaje de programación orientado a objetos basado en el lenguaje C., 21

Computer-generated imagery (CGI): Imágenes, efectos especiales o personajes generados mediante ordenador, 8

F

FPS: Fotogramas por segundo., 26

Framework: Es un tipo de software que ofrece un entorno de trabajo estandarizado en el que trabajar., 21

G

Goal Oriented Planning Action (GOAP): Es un tipo de IA en la que un agente tiene uno o varios objetivos a cumplir y buscará crear una secuencia de acciones disponibles para cumplirlos., 12

H

Hit box: Elementos dentro de una aplicación que delimitan y controlan colisiones entre elementos. Suelen encontrarse especialmente en videojuegos., 30

I

Inteligencia Artificial (IA): Es la simulación de inteligencia humana en una máquina., 9

M

Máquinas de estado finitos: Es un tipo de diseño de IA en el que dado un determinado estado del entorno el programa actuará de una forma concreta. Si el entorno varía el programa podrá cambiar de estado y actuar de manera diferente., 12

P

Plugins: Programa informático que modifica o agrega nueva funcionalidad a otro programa., 22

1 Introducción

En esta memoria de Trabajo de Fin de Grado se desarrolla un videojuego con Unity denominado *TRON* en el cual se incluye la aplicación de técnicas de machine learning como el aprendizaje por refuerzo para desarrollar dinámicas de juego de los enemigos de forma que su comportamiento esté entrenado en vez de programado.

1.1 MOTIVACIÓN

Este proyecto nace de unir dos de mis aficiones: los videojuegos y la ciencia ficción. La película *TRON* aúna ambas.

TRON [1] es una película de ciencia ficción del año 1982 en la que el protagonista Kevin Flynn es teletransportado dentro de un ordenador donde tendrá que interactuar y enfrentarse con diferentes programas para lograr escapar. El estilo de *TRON* es claramente distinguible de otras del mismo género y es además una de las primeras películas en utilizar imágenes generadas por ordenador (*Computer-generated imagery*, CGI).

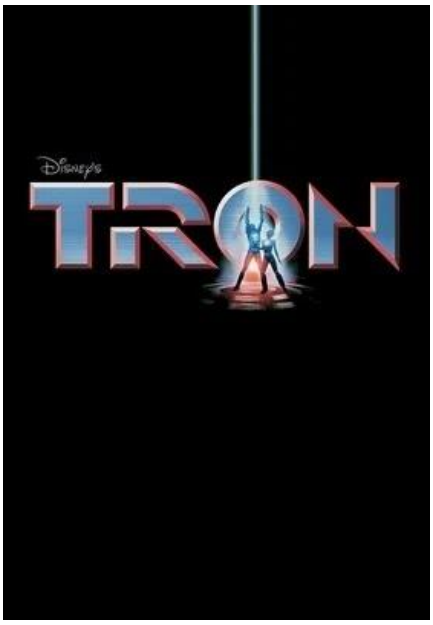


Figura 1.1 Póster de TRON (Fuente: TRON, Películas de YouTube)

En la película hay una escena en la que el protagonista está escapando en una de las "Motos de Luz" cuando sus captores le cortan el paso y se enfrentan utilizando

las motos. Esta escena icónica fue convertida a un videojuego llamado GLTRON, al que jugaba, más horas de las que debiera, allá por el 2002.

Este videojuego queda a día de hoy algo obsoleto y es injugable debido a problemas de compatibilidad con los ordenadores actuales, por lo que aprovechando que otro tema que me interesa es el *machine learning*, decidí aprender y desarrollar empleando herramientas actuales, una versión propia de este juego en la que la Inteligencia Artificial (IA) de los enemigos está entrenada y no programada.

1.2 OBJETIVOS

El objetivo principal del proyecto consiste en aprender a utilizar diferentes herramientas básicas que se emplean en desarrollo de videojuegos y programar una versión sencilla y funcional del juego.

Teniendo en cuenta esto, se pueden especificar los objetivos que lo componen:

- Conocer los motores de videojuegos que hay y elegir uno en el que conseguir un manejo básico del mismo.
- Conocer las herramientas de modelado existentes y elegir una en la que conseguir un manejo básico.
- Aprender el lenguaje de programación correspondiente al motor.
- Aplicar *machine learning* en las IA del juego para entrenar el comportamiento de los enemigos.
- Diseñar y desarrollar una primera versión funcional del juego.
- Evaluar el funcionamiento del juego creado.
- Elaborar el proyecto software asociado al desarrollo de la aplicación.

1.3 METODOLOGÍA

Durante la realización de este proyecto se ha empleado una metodología ágil basada en sprints. Un sprint es el periodo de trabajo en el que se organiza el desarrollo y durante el cual se determinan y desarrollan las partes del proyecto. Los sprints tienen una duración de 2 semanas y consisten en las siguientes fases:

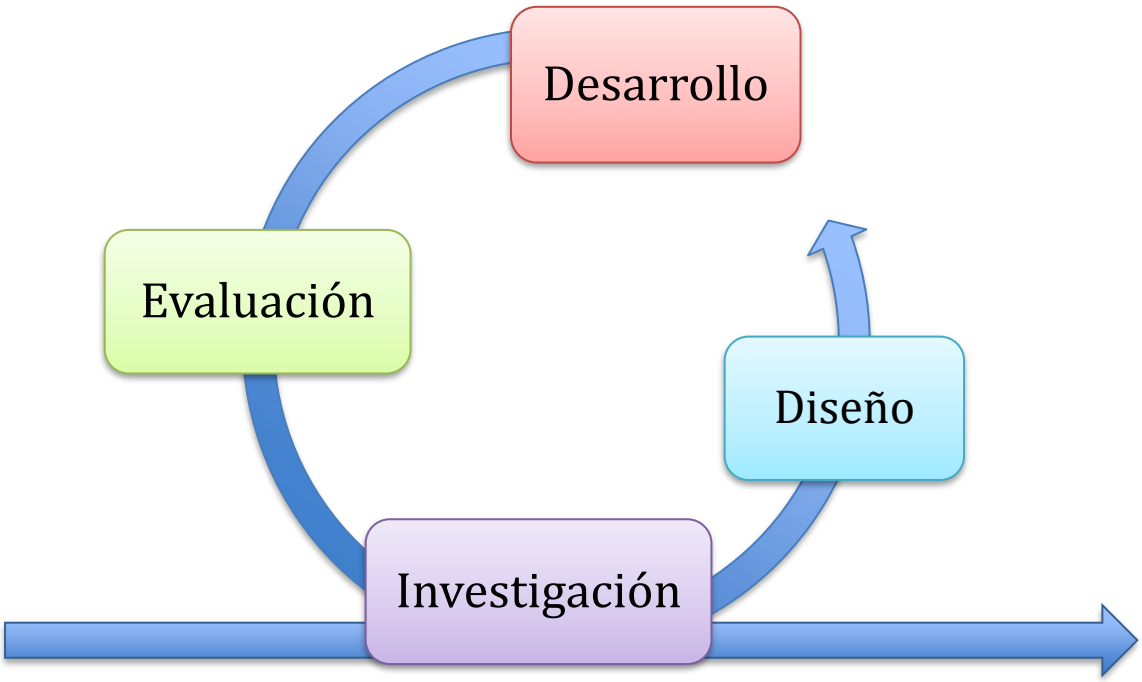


Figura 1.2 Fases de un Sprint (Fuente: Propia)

1. Primero se realiza una fase de investigación donde nos informamos de las tecnologías disponibles y diferentes posibles métodos de realizar la tarea a mano.
2. A continuación, diseñamos una solución que cumpla con nuestros requisitos en base a los conocimientos adquiridos en la fase de investigación.
3. Una vez hemos decidido el diseño pasamos al desarrollo de la solución.
4. La última fase del sprint consiste en asegurar que la solución implementada funciona correctamente en el sistema.

1.4 ESTRUCTURA DEL TRABAJO

En este apartado se detalla cómo se ha dividido el documento y qué se puede esperar de cada apartado:

1. Introducción:

Este apartado consiste en una breve presentación del proyecto, explicando la motivación y objetivos del mismo, así como la metodología empleada y el formato de este documento.

2. Estudio del problema:

En este apartado se explica el contexto de las tecnologías disponibles para el desarrollo del proyecto y los motivos que han hecho que se elijan las empleadas en el trabajo.

3. **Gestión de proyecto software:**

En este apartado se presentan las diferentes consideraciones realizadas al diseñar y planificar el proyecto, y la organización del mismo.

4. **Solución:**

En este apartado se detalla el proceso de desarrollo del proyecto software.

5. **Evaluación:**

En este apartado se evalúa el funcionamiento del programa realizado, tanto a nivel de código mediante pruebas unitarias y de usuario como realizando pruebas de rendimiento y experiencia de usuario.

6. **Conclusión:**

En este apartado se detallan las habilidades adquiridas al final del proyecto y se mencionan los problemas encontrados durante el desarrollo y un futuro del desarrollo.

2 Estudio del problema

A continuación, se presenta el contexto de realización del trabajo y se realiza un análisis de las diferentes tecnologías y herramientas disponibles, además de especificar cuales se van a emplear y los motivos por los que se han elegido.

2.1 EL CONTEXTO DEL PROBLEMA

Los videojuegos se han convertido durante estas últimas décadas en el mayor medio de entretenimiento que existe, superando al cine, la televisión y la música, entre otros. Parte de este impulso se debe a la aparición de los smartphones, los cuales han permitido que los usuarios puedan jugar a juegos en cualquier momento y lugar; mientras se espera al bus o a que empiece una clase.

Con esta popularización también se ha visto el auge de los juegos independientes, juegos normalmente desarrollados por equipos pequeños, siendo el ejemplo más prominente Minecraft. Los videojuegos independientes han traído consigo nuevas formas y sensibilidades al medio, creando o estableciendo géneros nuevos o también revisando otros géneros clásicos como puedan ser los juegos arcade.

En lo respectivo a la IA, los videojuegos son programas donde en gran cantidad de ocasiones partes del sistema tienen que actuar de formas complejas y tener comportamientos determinados para cumplir ciertas funciones o crear desafíos para el jugador.

De un tiempo a esta parte los avances en *machine learning* han dado lugar al desarrollo de herramientas que son capaces de aprender a realizar tareas complejas y de una forma flexible.

Esta tecnología bien puede ser aplicada a los videojuegos donde la necesidad de crear agentes que sean capaces de realizar comportamientos complejos dentro de diferentes entornos es algo usual.

Las formas de solventar la flexibilidad requerida que se suelen emplear en el desarrollo de IA clásicas son Máquinas de estado finitos o GOAP (lo cual es una extensión de lo anterior); soluciones que se basan en establecer los diferentes comportamientos que puede tener y crear condiciones que decidan qué comportamiento se ejecuta. Otra solución puede ser emplear algoritmos que simulan un tipo de comportamiento.

Figura 2.1 OpenAI vs OG (Fuente: OpenAI Five Finals, Twitch)

La aplicación del aprendizaje automático en la IA de los agentes de un videojuego permite dotarles de un comportamiento complejo y que reaccionen de una forma interesante para con el jugador. Por ejemplo, OpenAI entrenó una IA para jugar a Dota2 [2] que fue capaz de ganar a jugadores profesionales; o el futuro *Hello Neighbor 2* el cual utilizará aprendizaje por refuerzo de forma activa para que el enemigo vaya aprendiendo a partir de las acciones de los jugadores haciendo de esa forma que la jugabilidad sea interesante en cada partida [3].

2.2 ESTADO DEL ARTE

El desarrollo de videojuegos varía en sus formas en gran medida. Los grandes estudios de videojuegos como Ubisoft o Sony tienden a crear superproducciones, juegos grandes y complejos, con muchos sistemas diferentes, gráficos detallados, etc; involucrando a cientos de personas en el desarrollo: diseñadores de niveles, de enemigos, de misiones; artistas gráficos, programadores, desarrolladores de motores gráficos, etc; y grandes campañas de marketing. En el lado opuesto del espectro nos encontramos la escena independiente, con estudios pequeños y recursos limitados, que normalmente se centran en crear juegos más cortos y simples, buscando ideas más originales, con un aspecto visual o jugabilidad más interesante para ganar la atención del público. También podemos encontrar una gran variedad de estudios en algún punto intermedio entre estos, estudios

independientes algo más grandes y con proyectos más ambiciosos; o empresas grandes como Ubisoft creando juegos más pequeños y contenidos.

Fuera de estos tipos de desarrollos podemos encontrarnos con pequeños experimentos y prototipos. La mayoría de este tipo de juegos suelen ser pruebas de concepto de una idea de un desarrollador, tomas de contacto con el desarrollo de videojuegos o resultados provenientes de una GameJam, eventos donde el objetivo es desarrollar un juego en 2 o 3 días ciñéndose al tema propuesto en el evento. Proyectos creados en estos eventos han llegado a desarrollarse en juegos completos y tener éxito.

Aunque los estudios más grandes, y raramente alguno pequeño, suelan emplear motores gráficos propios, los motores gráficos disponibles al público son una norma en la industria, especialmente en la escena independiente donde la mayoría de juegos que podemos encontrar usa alguno, especialmente Unreal Engine y Unity.

El uso de machine learning en los videojuegos se puede encontrar principalmente en experimentos y aplicaciones sobre juegos ya existentes, entrenando agentes para que aprendan a jugar y así descubrir estrategias nuevas o mejores, como en el proyecto de AlphaZero en el ajedrez.

Además, se pueden observar ejemplos en herramientas de generación de contenido donde esta tecnología se emplea para generar nuevos elementos en el juego. [4]

También existen ejemplos donde los enemigos toman decisiones de este modo, como el anteriormente mencionado *Hello Neighbor 2* o *Galactic Arms Race* [5]. Sin embargo, no es una práctica habitual en la industria. Conseguir comportamientos complejos mediante machine learning es complicado, y las estrategias usuales como máquinas de estados finitos proporcionan suficiente versatilidad y son más sencillas de crear.

3 Gestión de proyecto software

En este apartado se presentan las diferentes consideraciones realizadas al diseñar y planificar el proyecto, y la organización del mismo.

3.1 ALCANCE DEL PROYECTO

3.1.1 Definición del proyecto

Este proyecto pretende desarrollar un videojuego que se asemeje visualmente a la película TRON y cuyas mecánicas estén basadas en escenas de la misma. El juego estará pensado para un solo jugador. Las IAs que se encuentren en el juego emplearán algún tipo de machine learning para definir su comportamiento.

El programa creado es completamente funcional y se puede obtener y usar mediante un simple ejecutable.

3.1.2 Estimación de tareas

En este proyecto se ha optado por una metodología ágil en la que debido a las múltiples posibilidades para afrontar una misma tarea los sprints incluyen a su vez la fase de investigación, pudiendo así decidir qué solución se decide implementar y facilitando si hubiese que modificarla en el futuro.

3.2 PLAN DE TRABAJO

3.2.1 Identificación de tareas

A continuación, se listan las diferentes tareas en las que se ha dividido el proyecto:

1. Creación de los modelos 3D necesarios
2. Investigación sobre la tecnología
   2.1. Diseño y Desarrollo
3. Creación del entorno de ejecución
   3.1. Investigación sobre la tecnología
   3.2. Diseño y Desarrollo
   3.3. Pruebas
4. Programación de las mecánicas básicas

4.1. Aprendizaje
4.2. Diseño y Desarrollo
4.3. Pruebas
5. Creación de las IA
5.1. Investigación
5.2. Diseño y Desarrollo
5.3. Entrenamiento
5.4. Pruebas
6. Creación de la Interfaz
6.1. Investigación
6.2. Diseño y Desarrollo
6.3. Pruebas
7. Mejorar la visualización y funcionamiento del videojuego
8. Investigación del sonido
9. Diseño y Desarrollo del sistema de sonido
9.1. Pruebas
10. Creación de texturas
10.1. Pruebas

3.2.2 Planificación de tareas

Se ha decidido que los sprints durarán 2 semanas cada uno, donde la primera se centrará en investigar las tecnologías disponibles y como realizar la tarea seleccionada; y la segunda se centrará en el desarrollo de la solución decidida.

Las tareas: *Creación de los modelos 3D necesarios* e *Investigación sobre la tecnología*; se estima que se podrán completar en un mismo sprint teniendo en cuenta los conocimientos previos del desarrollador. Sin embargo, las tareas: *Diseño y Desarrollo del sistema de sonido* y *Creación de texturas*; se prevén que no lleven mucho tiempo dado que son aspectos más opcionales para nuestra aplicación.

Por otro lado, las tareas: *Programación de las mecánicas básicas* y *Creación de las IA*; se extenderán en 2 sprints cada una al subdividirlas en partes más simples.

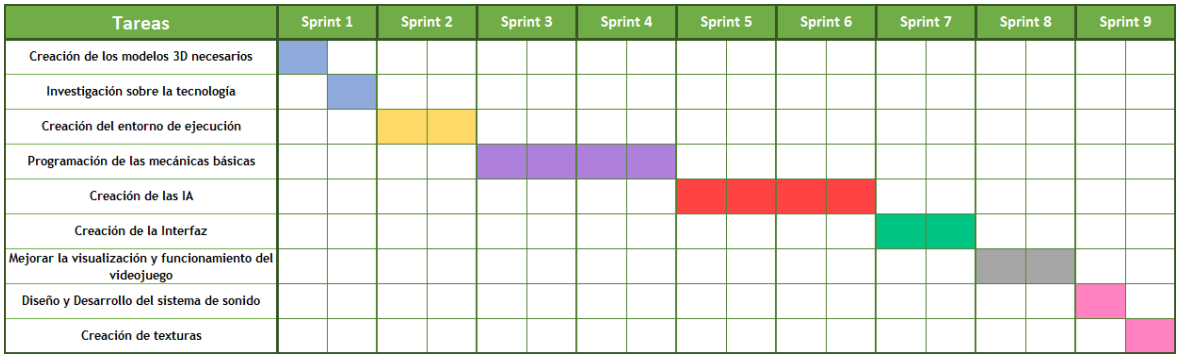


Figura 3.1 Diagrama de Gantt de las tareas en sprints (Fuente: Propia)

3.3 GESTIÓN DE RIESGOS

En este punto se van a analizar qué problemas pueden surgir durante el desarrollo del proyecto y cómo afectarían al desarrollo del proyecto en caso de producirse.

3.3.1 Identificación de riesgos

Teniendo en cuenta la naturaleza de este proyecto, los riesgos a los que podremos encontrarnos serán problemas técnicos o sobre la organización del proyecto:

- **Riesgos técnicos:** Se considera un riesgo técnico las situaciones donde se pueda producir un problema debido a:
  - Problemas de hardware.
  - Problemas de compatibilidad entre librerías.
- **Riesgos de organización:**
  - Retraso del desarrollo con respecto a la planificación.

3.3.2 Análisis de riesgos

En los siguientes puntos se analizan los riesgos valorados en el apartado anterior, señalando los problemas que puede causar al desarrollo y cómo afrontarlos en caso de que se produzcan.

Problema	Consecuencias	Solución
Problemas de hardware	- Retraso del desarrollo de la aplicación. - Pérdida de datos y/o documentación.	Detener el desarrollo y concentrar esfuerzos en solucionar los problemas encontrados, ya que el equipo es necesario para llevar a cabo el proyecto.
Problemas de compatibilidad entre librerías	- Modificación del diseño software. - Impedimento para emplear tecnologías seleccionadas.	Búsqueda de una solución al problema de compatibilidad mediante otra tecnología o abandonar las herramientas y librerías seleccionadas en favor de otra opción completamente diferente.
Retraso del desarrollo con respecto a la planificación	- Algunos de los objetivos pasan a estar fuera del alcance del proyecto.	Rehacer la planificación de acuerdo a los nuevos plazos disponibles.

Tabla 3.1 Análisis de riesgos.

4 Solución

En este apartado se explica la aplicación desarrollada, los requisitos que ha de cumplir y los diferentes procesos por los que se ha pasado durante el desarrollo y sus soluciones.

4.1 DESCRIPCIÓN DE LA SOLUCIÓN

Se ha desarrollado un juego en el cual el jugador toma el papel del protagonista, controlando la "Moto de Luz" y peleando en una arena similar a la de la escena en la película. La moto avanzará de forma constante mientras el jugador decide cuándo girar intentando aguantar sin chocar contra las paredes o los muros generados por las "Motos de Luz", enemigas y la suya propia. Otro objetivo del jugador es destruir a los enemigos cortándoles el paso o encerrándolos con su muro.

En este proyecto los enemigos son controlados por inteligencias artificiales entrenadas mediante aprendizaje por refuerzo. Los agentes serán capaces de obtener observaciones del entorno y decidirán acorde como actuar; en este caso la decisión será en qué dirección girar: izquierda o derecha. El hacer que las IAs funcionen de esta manera permitirá que su comportamiento sea más similar al de una persona jugando, aunque sea más complicado conseguir que aprendan comportamientos complejos.

Si el jugador choca en cualquier momento pierde, pero si sobrevive más tiempo que los enemigos, sea esto que los 3 enemigos choquen contra algo, gana. Cuando el jugador gane o pierda la partida acabará.

Cuando no se esté ejecutando el juego o se encuentre en pausa, el jugador se encontrará en alguno de los diferentes menús del programa, desde los que podrá ir al juego, configurar los ajustes gráficos o salir del juego.

4.2 TECNOLOGÍAS UTILIZADAS

4.2.1 Modelado 3D

- **Maya**

Maya [6] es un programa de creación de gráficos 3D y animación propiedad de Autodesk. Es uno de los referentes en lo que a herramientas de modelado 3D se refiere y se trata de un programa muy potente y que ofrece un equipo completo de herramientas muy pulidas para facilitar su uso al usuario. También permite hacer simulaciones detalladas empleando su motor físico Bifrost y hacer renders usando su motor propietario Arnold.

Maya requiere de una licencia de pago para su uso, lo cual ha hecho que no se elija para este proyecto.

- **Blender**

Es uno de los programas de modelado 3d más extendidos y utilizados, gracias en gran parte a que es un software gratuito. Blender [7] ofrece una gran cantidad de herramientas a la par de otros programas de modelado de pago como Maya. También cuenta con una gran comunidad lo que se traduce directamente en accesibilidad, con una gran cantidad de tutoriales [8] disponibles de forma gratuita y plugins creados por terceros.

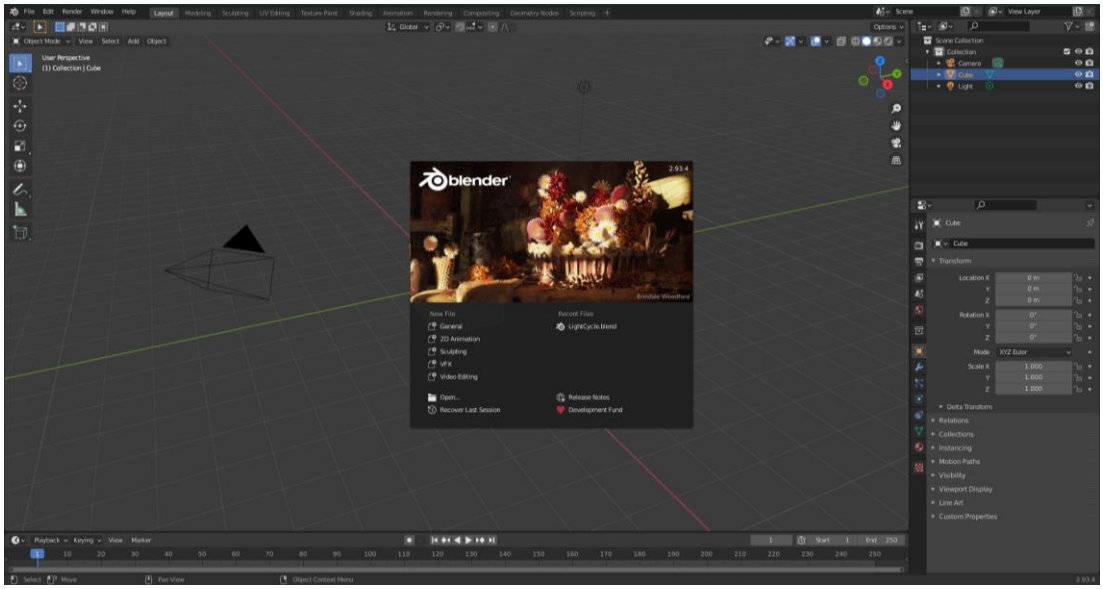


Figura 4.1 Blender (Fuente: Propia)

Teniendo en cuenta estos aspectos, especialmente que no requiera licencia de pago, se ha elegido Blender como herramienta de desarrollo para los modelos 3D necesarios para TRON.

4.2.2 Motores de Videojuegos

Un motor de videojuegos es un framework de trabajo diseñado para crear videojuegos. Proporcionan diferentes herramientas para desarrollar un juego y se maneja la renderización de los gráficos, los sistemas de físicas y gestión del juego con el ordenador. Los motores pueden ser completamente propietarios o pueden ser comerciales y requerir algún tipo de licencia de pago o gratuita.

Éstas son las opciones que se han barajado:

- **Unreal Engine**

Unreal Engine [9] es uno de los mejores motores gráficos disponibles con licencia gratuita, y ofrece un set de herramientas muy completo y potente. La licencia de Unreal es gratuita hasta que la aplicación que hayas creado empleando Unreal genere más de 1 millón de dólares.

Unreal Engine destaca por su capacidad para renderizar gráficos detallados y realistas fácilmente lo cual es un gran aliciente a la hora de elegirlo como motor de desarrollo para un juego detallado y muchos juegos de renombre han sido desarrollados usando este motor.

El lenguaje de programación que emplea este motor es C++ y es un motor muy portable entre diferentes plataformas.

Unreal cuenta con numerosos tutoriales y cursos en su página web, aparte de la documentación de la propia aplicación.

Unreal Engine también es usado en otras industrias como por ejemplo el cine y la televisión, donde se aprovechan sus capacidades para mostrar escenas y simulaciones detalladas a tiempo real. [10]

Unreal es un motor gráfico muy potente y que permite construir proyectos detallados y de gran envergadura, sin embargo, no se ha elegido para este trabajo debido a que su barrera de entrada es algo más alta que la otra opción, dados los orígenes del juego no se requiere de un renderizado detallado o realista y, por último, no dispone de un paquete nativo para realizar *machine learning*.

- **Unity**

Unity es un motor de videojuegos que permite el desarrollo de aplicaciones para prácticamente cualquier plataforma. Está considerado un motor muy versátil y ligero que es empleado en el desarrollo de toda clase de videojuegos, además de uno de los más sencillos de usar gracias a su interfaz simple. La licencia de Unity es gratuita si los ingresos que produces con los programas creados son inferiores a 100.000 dólares anuales [11].

Unity ofrece una gran cantidad de herramientas al usuario de base y una gran variedad de plugins tanto propietarios como de otros usuarios de la comunidad.

Unity no cuenta con tanta potencia de renderizado de base y conseguir un acabado realista y requiere más trabajo que si se empleasen otros motores. Sin embargo, debido a que se trata de un motor muy personalizable los resultados pueden variar mucho en apariencia.

Unity es uno de los motores más recomendados para empezar en el desarrollo de videojuegos por varios motivos: su interfaz sencilla, Unity cuenta con una plataforma de aprendizaje propia donde se pueden encontrar muchos cursos y tutoriales [12]; y emplea como lenguaje de programación C#.

Se ha optado por Unity para el desarrollo de este proyecto por ser simple y de fácil acceso, por la cantidad de contenido y documentación disponible de forma gratuita, y porque emplea C# el cual es un lenguaje de programación bastante sencillo.

4.2.3 Machine Learning

El machine learning es una técnica que consiste en hacer que un programa pueda aprender a realizar una función sin que esta esté programada explícitamente en el mismo. Para conseguir esto se suelen emplear redes neuronales.

Las redes neuronales generan una respuesta a un problema en base a la información que recibe.

Una red neuronal en informática consiste en una red de neuronas artificiales dividida en 3 tipos de capas neuronales [13]:

- Capa de entrada: las neuronas de esta capa reciben los datos del problema a resolver y comienzan su procesamiento, propagando sus resultados hacia las capas siguientes.
- Capas ocultas: son las más abundantes, estas capas continúan procesando y propagando los datos que reciben de las neuronas de la capa de entrada. Cada capa continua el flujo enviando sus resultados a la siguiente capa oculta, hasta que llegan a la capa final.
- Capa de salida: es la última capa de la red neuronal. Las neuronas de esta capa reciben los datos que han sido procesados multitud de veces en las capas ocultas y genera una respuesta al problema.

Para conseguir que la respuesta de la red se corresponda a lo que se quiere que haga, lo que se hace es entrenar la red para modelarla acorde al problema. En general, podemos dividir las aplicaciones de aprendizaje automático en tres tipos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Aprendizaje supervisado utiliza datos etiquetados y se entrena el clasificador para lograr reconocerlos. Se conoce cuando el sistema estima correctamente una respuesta y se va corrigiendo su procesado de la información inicial para conseguir responder correctamente.

Aprendizaje no supervisado emplea datos sin etiquetar por lo que no sabe qué respuestas son correctas o no, así su objetivo suele ser encontrar patrones información que el usuario desconoce.

Para este proyecto se ha decidido emplear aprendizaje por refuerzo para controlar los enemigos del juego. El aprendizaje por refuerzo consiste en ofrecer a nuestra red neuronal una serie de observaciones que indican qué cosas conoce sobre el entorno y sobre sí mismo, y que ésta nos diga qué tiene que hacer el agente dentro de sus acciones posibles.

Aprendizaje por refuerzo se encuentra en un punto intermedio entre aprendizaje supervisado y no supervisado. Por una parte, la red neuronal sabe cuando está haciendo algo bien o mal, pero las decisiones que toma afectan a su vez a la información que recibe por lo que también desconoce como se modifica la misma,

pudiendo generar así estrategias y comportamientos que son desconocidos para el programador.

Para determinar y orientar el comportamiento del agente se emplea un sistema de recompensas que el sistema intentará maximizar. La idea es que la recompensa aumenta cuando el agente se comporta como se desea indicándole así a la red que va por el buen camino.

Al entrenar la red recibirá como valores de entrada las diferentes observaciones, devolverá una serie de acciones que hará el agente y se le devolverá el valor de la recompensa actual que le indicará si va por buen camino o no.

El aprendizaje por refuerzo permite crear IAs capaces de responder a entornos dinámicos y que son capaces de comportamientos complejos.

Se han tenido en cuenta dos opciones a la hora de elegir la herramienta para el aprendizaje de los enemigos:

- OpenAI Gym

OpenAI Gym [14] es un conjunto de herramientas orientado al desarrollo y entrenamiento de inteligencias artificiales a través del aprendizaje por refuerzo empleando Python. OpenAI es una herramienta muy potente como ha demostrado a través de varios experimentos, en especial el de Dota2.

A pesar de la clara eficacia de esta tecnología y su potencia, no se ha empleado en este proyecto por dos motivos: los agentes tienen un comportamiento simple, y la implementación con Unity es más complicada.

- ML-Agents

ML-Agents [15] es un paquete oficial de Unity que ofrece utilidades para entrenar agentes definidos en Unity a través de Python. A partir de su API de Python permite realizar diferentes métodos de aprendizaje: por refuerzo, imitación, neuro evolución, entre otros.

El que sea un paquete oficial de Unity que cumple con todas las necesidades que nos podemos encontrar a la hora de crear un agente en nuestro juego, y que su instalación sea sencilla ha hecho que nos decantemos por esta librería para realizar el proyecto.

4.3 EL PROCESO DE DESARROLLO

4.3.1 Análisis

El análisis del proyecto nos sirve para examinar claramente cada uno de los puntos que engloban nuestro proyecto.

En este proyecto no hay distinción de tipos de usuarios, se define al usuario como: "jugador". El sistema interactuará siempre de la misma forma con el jugador.

La parte jugable será siempre la misma, y su única variación será el modelo de IA que controla a los enemigos y que podrá ser elegido antes de empezar a jugar. La selección de la IA servirá también como selector de dificultad, teniendo algunos modelos que supongan un mayor desafío al jugador.

4.3.2 Definición de requisitos

A continuación, se listan los diferentes requisitos que ha de cumplir el sistema:

4.3.2.1 Requisitos funcionales

1) El jugador tiene que poder iniciar el juego cuando quiera una vez abierto el programa.
2) El jugador tiene que poder configurar el juego.
3) El jugador ha de poder seleccionar la resolución del juego.
4) El jugador ha de poder seleccionar la calidad de gráficos del juego.
5) El jugador ha de poder seleccionar si el juego se muestra en pantalla completa o en modo ventana.
6) El jugador ha de poder modificar el volumen del audio.
7) El jugador ha de poder controlar la "Moto de Luz" pudiendo girar a izquierda y derecha según decida.
8) El jugador ha de poder pausar el juego.
9) El jugador ha de poder resumir el juego si este está pausado.
10) El jugador ha de poder ver los controles.
11) El jugador ha de poder cerrar el juego.
12) El jugador ha de poder cerrar la aplicación.
13) El jugador perderá la partida si su "Moto de Luz" es destruida.

14) El jugador ganará la partida en el momento que el último enemigo es destruido.
15) Las "Motos de Luz" han de moverse constantemente hacia adelante.
16) Las "Motos de Luz" han de girar con un ángulo de 90º.
17) Las "Motos de Luz" tienen que dejar tras de sí un muro.
18) Las "Motos de Luz" tienen que destruirse cuando choquen contra una pared o un muro.
19) La cámara seguirá al jugador en todo momento.

4.3.2.2 Requisitos no funcionales

20) El juego debe mantenerse por encima de 30FPS.
21) Las cargas entre escenas no deben ser superiores en duración a 10 segundos.
22) La interfaz ha de ser sencilla y fácil de navegar.
23) La configuración de gráficos ha de mantenerse entre ejecuciones.
24) El programa será un ejecutable y no necesitará instalación alguna.

4.3.3 Especificación de requisitos

Se especifican las funcionalidades que tiene que cumplir el sistema para con el usuario y como están estructuradas en el sistema.

- El jugador navegará desde el menú principal hacia el resto de las funcionalidades y será el centro de la navegación. Salir de cualquier menú tendrá que devolver al usuario al menú principal, con la excepción del menú de pausa que será un paso intermedio para salir del juego sin que este acabe de forma normal.
- Los requisitos jugables serán más importantes, dado que el juego es la principal funcionalidad del programa.

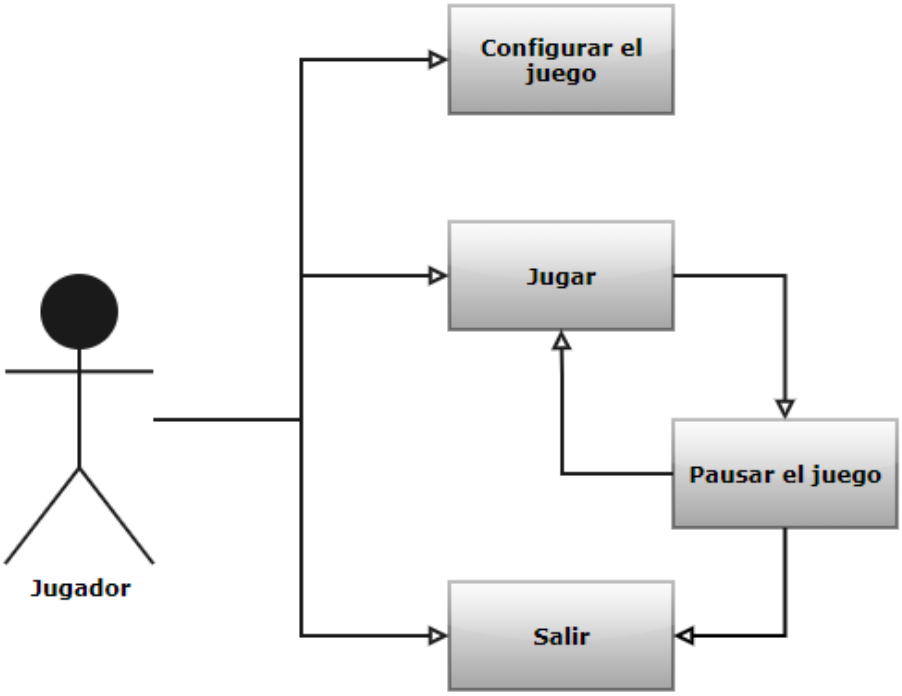


Figura 4.2 Esquema base de acciones que puede realizar el jugador (Fuente: propia)

4.4 DISEÑO

4.4.1 Diseño de sistema

En la figura 4.3 se muestra un diagrama de cómo está dividido el sistema y que diferentes componentes actúan en cada momento de la ejecución. Como se puede ver se han diferenciado dos grandes partes que se consideran independientes en la ejecución:

- Los menús son la interfaz del sistema y el medio para desplazarse en la aplicación, pudiendo cambiar entre los diferentes menús disponibles o decidir lanzar el juego, donde los menús dejan de tener efecto alguno hasta que este se detenga, ya sea porque se ha pausado o la partida ha finalizado por algún motivo. La navegación en la aplicación es la principal funcionalidad de los menús, pero también cumplen otras funciones: las opciones sirven para modificar el aspecto gráfico del juego y los controles informan al usuario como controlar el juego.
- El juego es la parte más importante de la aplicación y aunque solo tiene dos componentes propios la funcionalidad de los mismos es la más

compleja. El jugador es el componente más importante y el único que interactúa con los menús. Los enemigos por su parte solo actúan en el juego y no pueden ser modificados de ninguna manera por el jugador.

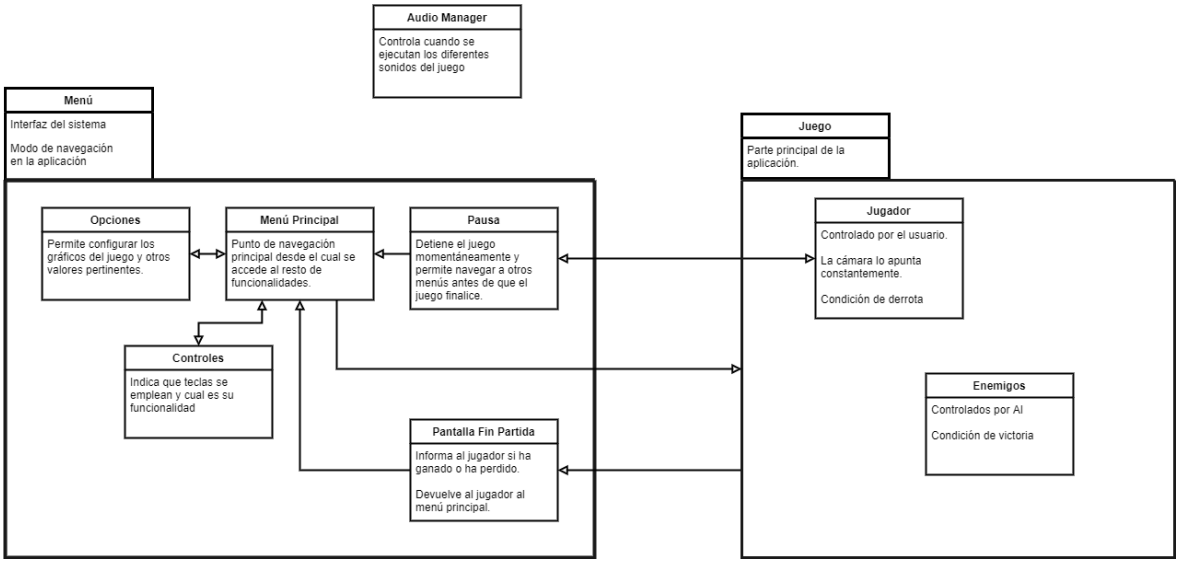


Figura 4.3 Diagrama de relación del jugador con las diferentes partes del sistema y entre ellas. (Fuente: Propia)

- El administrador de audio es una parte no fundamental del sistema que actúa independientemente de como se encuentre la aplicación. Se encarga de manejar los efectos de audio y la música del juego.

Otra parte del diseño es como está planteada la sensación de juego o kinestesia, y el cómo esté planteada puede cambiar completamente como se ve un juego. En este proyecto se quiere crear un juego arcade donde la sensación de velocidad, los reflejos y la intuición a la hora de maniobrar sean el principal foco de la experiencia.

4.4.2 Implementación

En este punto se detalla el desarrollo del sistema diseñado y sus componentes, comentando el diseño, proceso y solución de problemas encontrados durante el desarrollo.

Durante el desarrollo del proyecto se han empleado diferentes tecnologías aparte de las ya mencionadas y se hablará de ellas cuando sea pertinente durante las explicaciones.

Cabe también comentar que este proyecto ha sido programado en C#, ya que es el lenguaje que emplea Unity, utilizando como IDE de programación VisualStudio 2019, un entorno de desarrollo propiedad de Microsoft y que permite trabajar con varios lenguajes y tecnologías como Azure, Python, C++, y Unity, entre otros.

Las explicaciones se presentarán en el orden en el que se han realizado, creando así una cronología del desarrollo.

4.4.2.1 Modelado "Moto de Luz"

Se ha empezado el desarrollo creando el modelo 3D de la "Moto de Luz" de TRON. Para ello se ha empleado la herramienta Blender donde mediante el uso de primitivas y modificadores sencillos, se ha dado forma a la "Moto de Luz".

Figura 4.4 Modelo final en Blender (Fuente: Propia)

Figura 4.5 Modelo 3D de la "Moto de Luz" (Fuente: Propia)

4.4.2.2 Creación del entorno

Empleando el editor de Unity se ha creado una simple arena con forma de caja sin tapa en la que se llevará a cabo tanto el entrenamiento como el juego en sí.

Se han empleado formas básicas para crear la arena y se ha aumentado el tamaño de las hit box para asegurarse de que los agentes no se puedan salir del entorno.

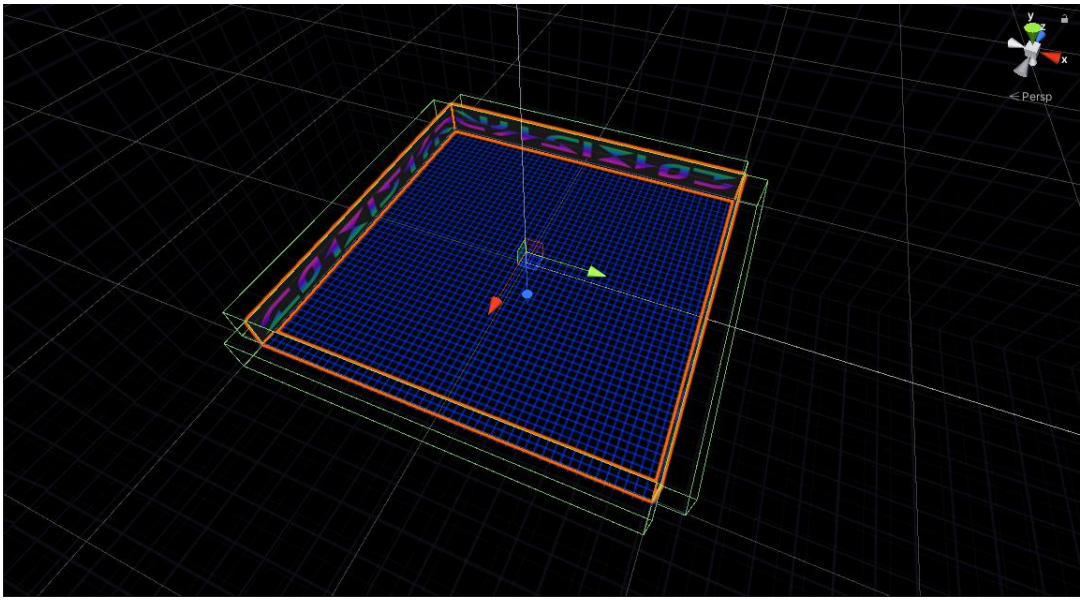


Figura 4.6 Entorno en el que se desarrolla el juego (Fuente: Propia)

4.4.2.3 Movimiento de los agentes

El movimiento de las "Motos de Luz" es la parte fundamental de toda la jugabilidad. Las "Motos de Luz" tienen varias condiciones en cuanto a su movimiento:

1) Siempre se están moviendo hacia adelante. Las "Motos de Luz" no pueden frenar ni acelerar de forma intencionada en ningún momento.
2) Cuando una moto gira lo hace creando un ángulo recto. Los giros se realizarán instantáneamente y no habrá punto intermedio entre giros, solo se modificará la rotación en incrementos de 90º.
3) Se mueven siempre en una línea recta. Las motos solo pueden cambiar su dirección cuando realizan un giro.

Este código será común tanto para el jugador como para los enemigos, ya que ambos se mueven exactamente igual.

Para conseguir este comportamiento se ha empezado por hacer que el agente siempre avance hacia adelante. Para esto primero tenemos que determinar qué es "adelante":

Se ha establecido que se entenderá "adelante" como el vector en el eje Z en sentido positivo del objeto, el cual está denominado en Unity como el vector "*forward*" que se traduce como el vector normalizado de un objeto que apunta en la dirección Z y que por convención se considera adelante.

Una vez establecido esto, podemos hacer que el objeto modifique de forma constante su posición en la dirección de ese vector. Consiguiendo así a todos los efectos que el objeto se mueva en la dirección de ese vector local.

Una vez que el agente se mueve pasamos a programar los giros.

Para conseguir que los agentes realicen el giro de 90º, la rotación ha de ser instantánea ya que cualquier rotación intermedia haría que no se formase un ángulo recto cuando se implemente la generación del muro, debido a que la moto siempre está en movimiento. Por tanto, en vez de modificarse de forma gradual la rotación del agente, el valor de la rotación cambiará en el instante que se produzca el giro y no habrá ninguna otra modificación.

Este comportamiento se ha programado de la siguiente manera: primero se distingue en qué sentido se quiere rotar, izquierda o derecha; entonces se modifica el valor del vector "*rotation*" que indica hacia donde está mirando el objeto en +/-90º.

```
void Update()
{

    // Checking for turning inputs
    if (Input.GetKeyDown(KeyCode.RightArrow))
    {
        turn = 1;
    }

    else if (Input.GetKeyDown(KeyCode.LeftArrow))
    {
        turn = -1;
    }
}

// Update is called once per frame
void FixedUpdate()
{
    if (turn == 1)
    {
        this.transform.Rotate(this.transform.up, turnAngle);

        turn = 0;
    }
    else if (turn == -1)
    {
        this.transform.Rotate(this.transform.up, -turnAngle);

        turn = 0;
    }
    this.transform.Translate(0, 0, movementSpeed * Time.deltaTime);
}
```

Figura 4.7 Código de movimiento de los agentes (Fuente: Propia)

Juntando estos dos comportamientos hemos conseguido, en principio, imitar el movimiento de la película original. Sin embargo, al testear el movimiento podemos observar un problema, no en el comportamiento esperado, sino en la sensación que produce al jugar. Ciertamente el agente se mueve en líneas rectas y con giros de 90º, pero la rotación se produce desde el centro del objeto, haciendo que los giros parezcan menos apurados una vez que se realizan. Este detalle provoca una sensación de descontrol al jugar.

La forma de solucionar este problema ha sido modificar el punto de giro de la moto, introduciendo un pequeño desfase que hace que gire más adelante, cerrando así ese espacio se dejaba antes y eliminando la sensación de poco control.

4.4.2.4 Generando un muro

Después del movimiento esta funcionalidad es la más importante del juego, ya que toda estrategia y amenazas para el jugador nacen de esta mecánica.

El muro se tiene que ir generando de forma constante detrás de la "Moto de Luz" y seguirá el recorrido que efectúe ésta perfectamente. Esta funcionalidad se ha implementado de la siguiente manera:

El muro ha de ir exactamente por donde ha pasado la moto, por lo que dentro del código de movimiento añadimos la capacidad para ir creando instancias del muro que se van generando en la posición del agente y que al desplazarse este quedan visibles.

El muro está formado por dos tipos de partes: las zonas rectas y las esquinas. Mientras el agente se mueva en línea recta se irán instanciando en el juego zonas rectas, que son piezas más alargadas y que permiten reducir la cantidad de objetos que se van generando.

En el momento en que se gira se instancia en vez de una zona recta una esquina, para conseguir un muro limpio, porque si se instancia la zona recta justo en el giro al ser alargado sobresaldrá un poco y no quedará un muro limpio y que fluya. Es también importante porque puede dar problemas de jugabilidad tener esos salientes en un juego donde la movilidad es tan limitada.

```
    // Update is called once per frame
    void FixedUpdate()
    {
        if (turn == 1)
        {

            GameObject wall = Instantiate(turnPrefab, this.transform.position,
this.transform.rotation);
            generatedWall.Add(wall);

            this.transform.Rotate(this.transform.up, turnAngle);

            turn = 0;
        }
        else if (turn == -1)
        {

            GameObject wall = Instantiate(turnPrefab, this.transform.position,
this.transform.rotation);
            generatedWall.Add(wall);

            this.transform.Rotate(this.transform.up, -turnAngle);

            turn = 0;
        }
        else
        {

                GameObject wall = Instantiate(wallPrefab,
this.transform.position, this.transform.rotation);
                generatedWall.Add(wall);


        }

        this.transform.Translate(0, 0, movementSpeed * Time.deltaTime);

        if (generatedWall.Count > 500 && dynamicWall)
        {
            GameObject wall = generatedWall[0];
            generatedWall.Remove(wall);
            Destroy(wall);
        }

    }
```

Figura 4.8 Código de movimiento con generación de muro (Fuente: Propia)

4.4.2.5 Ganar/Perder

Ahora que contamos con un agente funcional pasamos a establecer cuando un agente es destruido, lo que ocurrirá cuando choque contra algo. Para ello nos aprovechamos de funciones y eventos predeterminados de Unity.

A los agentes se les ha añadido un componente “Character controller” el cual contiene un colisionador que permite detectar cuando el agente ha chocado contra algo. Utilizando esto podemos añadir una función que detecta el evento de que el agente se choque contra algo y tratarlo.

Cuando choque destruiremos en primer lugar el muro asociado a ese agente. Para ello guardamos en una lista los objetos que el agente va instanciando y los destruimos cuando se llama a la función. A continuación, simplemente hacemos que el agente se destruya a sí mismo.

Una vez que podemos sacar de la escena a los diferentes agentes cuando mueren, podemos determinar las condiciones de victoria y derrota que se emplearán en el juego:

- El jugador ganará si cuando el último enemigo es destruido, él continúa vivo.
- El jugador perderá en caso contrario.

Al acabar la partida el juego se detendrá y se mostrará al jugador una pantalla que indicará si ha ganado o perdido desde la cual volverá al menú principal.

```
private void OnTriggerEnter()
{
    Debug.Log("Player: HIT");

    FindObjectOfType<AudioManager>().Stop("Motor");

    Explode();
}


public void Explode()
{
    foreach (GameObject w in generatedWall)
    {
        Destroy(w);
    }

    FindObjectOfType<PauseMenu>().Lose();

}
```

Figura 4.9 Código destrucción del jugador y mostrar pantalla de derrota (Fuente: Propia)

```
/// <summary>
/// Called when the agent's collider enters a trigger collider
/// </summary>
/// <param name="other">The trigger collider</param>
private void OnTriggerEnter(Collider other)
{
    isAlive = false;

    if (isTraining)
    {
        AddReward(-30f);

        //if (enemyAgent.isAlive) { enemyAgent.AddReward(5f); }
    }

    if (!isTraining )
    {
        GameObject[] agents = GameObject.FindGameObjectsWithTag("AI");

        int agentsAlive = 0;

        foreach (GameObject ag in agents)
        {
            if (ag.GetComponent<LightCycleAgent>().isAlive)
            {
                agentsAlive++;
            }
        }
        if (agentsAlive < 1) {
            FindObjectOfType<PauseMenu>().Win();
        }
    }

    Explode();

    if (isTraining)
    {
        //enemyAgent.EndEpisode();

        EndEpisode();
    }
    else
    {
        explosion.Play();
        GameObject cycle = gameObject.transform.GetChild(0).gameObject;
        cycle.SetActive(false);
    }
}
```

Figura 4.10 Código destrucción de los enemigos, fin de episodio de entrenamiento y mostrar de pantalla de victoria (Fuente: Propia)

4.4.2.6 Cámara

La cámara es el componente que controla qué y cómo se muestra la escena en la pantalla, por lo que es uno de los elementos más importantes a la hora de crear

juego. El cómo se vea la acción y el qué se ve, puede hacer que un juego se sienta de maneras muy distintas, mejorando o empeorando la experiencia.

Para que el jugador pueda ver y tomar decisiones mientras controla su "Moto de Luz" correctamente este ha de poder ver de una forma clara dónde se encuentra, hacia dónde se está dirigiendo y qué tiene a su alrededor. Esto en un principio podría solucionarse con una cámara que muestre en una vista de planta la arena y el jugador desde esa posición tendrá siempre un control y visión de la situación excelente. Sin embargo, uno de los aspectos que se intentan conseguir es dar una sensación de velocidad, por lo que una cámara cenital no cumple con nuestras necesidades.

La decisión final ha sido emplear una cámara en tercera persona que siga constantemente al jugador y que acentúe la sensación de velocidad. Para conseguir esto se han tenido que hacer ciertos compromisos con las necesidades mencionadas en el anterior párrafo, reduciendo la capacidad del jugador para ubicarse a sí mismo en la arena y la visión de la que dispone, haciendo así que pierda de vista en ciertos momentos a los enemigos y no sepa donde se encuentran. Estas limitaciones pueden parecer grandes, pero son necesarias para conseguir las sensaciones de juego que se desean, en las que estos posibles problemas pasarán desapercibidos.

En orden de crear esta cámara hemos empleado el paquete de Unity Cinemachine.

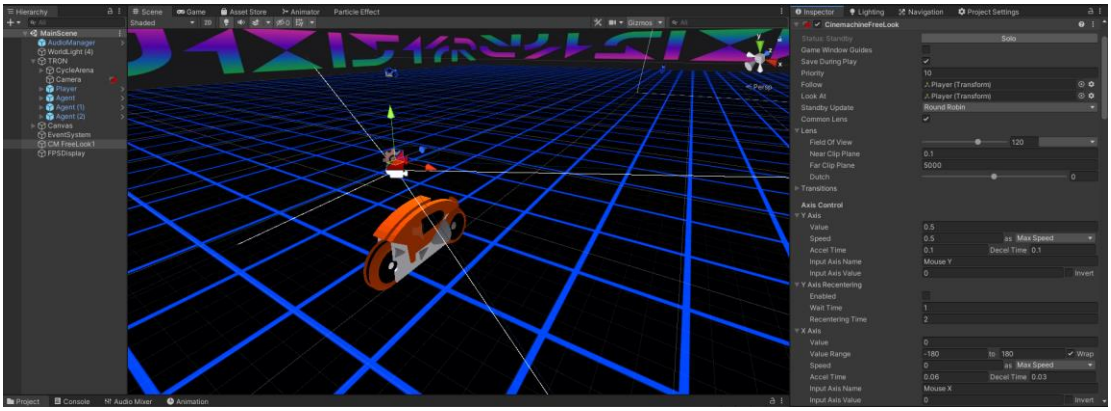


Figura 4.11 Configuración de cámara con Cinemachine (Fuente: Propia)

Cinemachine es un conjunto de herramientas para crear cámaras dinámicas, inteligentes y hacerlo de una forma sencilla sin tener que emplear código. Cinemachine ofrece varios tipos de cámaras predeterminadas a crear y a las que

añadir funcionalidades complejas modificando simplemente los diferentes atributos y valores de configuración.

4.4.2.7Diseño de la IA

Los agentes que sean controlados por la máquina se comportarán de forma similar a como pueda actuar un jugador. Para ello se ha decidido que la forma en la que se comporten de base sea que aprendan a sobrevivir y manejar un agente para evitar obstáculos y aguantar lo máximo posible, forzando así al jugador a que se arriesgue intentando atraparlo para ganar.

La IA tendrá unas capacidades similares a las del jugador, sabrá qué tiene delante y qué tiene a derecha e izquierda. Además de esto se probará a añadir otras observaciones y se comprobará cómo modifican el comportamiento y cómo son de eficaces.

Para crear la IA se ha empleado el paquete ML-Agents que permite en conjunción con su paquete de Python realizar aprendizaje por refuerzo y entrenar así una IA que aprenda a evitar obstáculos.

El código de movimiento, generación de muro y destruir es igual tanto como en los enemigos como en el jugador, pero ahí acaban las similitudes. Para entrenar un agente a través de ML-Agents primero tenemos que adaptar el código que controla los enemigos. El paquete trae consigo una clase propia llamada “Agent” de la que heredará la nuestra. Esta clase proporciona los métodos necesarios para realizar el aprendizaje, en las que especificar qué ve nuestro agente, y qué acciones puede realizar acorde a las instrucciones que reciba de la red neuronal; también proporciona funciones para declarar cuándo se ha acabado un episodio de entrenamiento y especificar qué recompensas se añaden.

Como observaciones se crearán tres sensores que mirarán respectivamente hacia adelante, derecha e izquierda hasta cierta distancia que denominaremos radio de visión; y le indicarán si hay algo o nada.

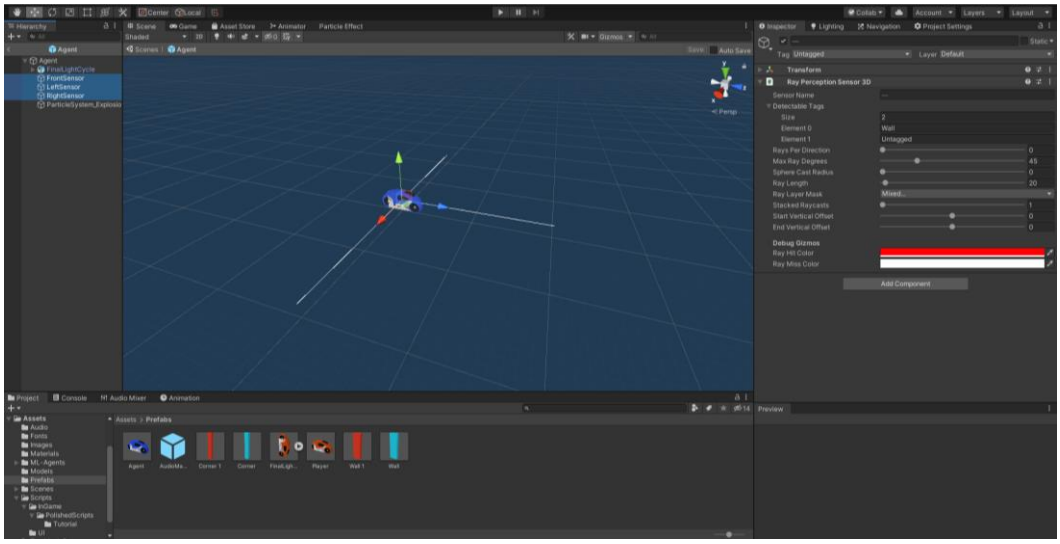


Figura 4.12 Agente por defecto con sensores (Fuente: Propia)

Las acciones que puede realizar son las mismas que puede realizar el jugador, el agente podrá decidir si girar y en su caso hacia qué lado, o no hacer nada y continuar recto.

Un episodio de entrenamiento terminará cuando el agente sea destruido y para que aprenda se penalizará que choque contra algo fuertemente y se recompensará de forma continua que se mantenga con vida.

```
/// <summary>
    /// Initialize the agent
    /// </summary>
    public override void Initialize()
    {
        startPosition = this.transform.position;
        startRotation = this.transform.rotation;

        explosion = GetComponentInChildren<ParticleSystem>();

        cycleArena = GetComponentInParent<CycleArena>();

        onCooldown = true;
        Invoke("RefreshCooldown", cooldown);


        // Normal gameplay
        if (!isTraining)
        {
            MaxStep = 0;
        }
    }
```

Figura 4.13 Inicializar el agente (Fuente: Propia)

```
    /// <summary>
    /// Reset the agent when an episode begins
    /// </summary>
    public override void OnEpisodeBegin()
    {
        ResetAgent();
        explosion.Stop();
    }

    /// <summary>
    /// Resets the agent to its original position and rotation
    /// </summary>
    private void ResetAgent()
    {
        this.transform.position = startPosition;
        this.transform.rotation = startRotation;

        Explode();

        isAlive = true;

        onCooldown = true;
        Invoke("RefreshCooldown", cooldown);
    }
```

Figura 4.14 Devuelve al agente a su estado original al comienzo de un episodio de entrenamiento (Fuente: Propia)

```
    /// <summary>
    /// Collect vector observations from the enviroment
    /// </summary>
    /// <param name="sensor">The vector sensor</param>
    public override void CollectObservations(VectorSensor sensor)
    {
        /*
         *  The first approach is to leave all the observations to the children
sensor.
         *
         *  Expected result: Learn to survive similar to the heuristic.
         *
         *  Rewards: [-30 if it crashes, +0.01/0.02s] The reward only increases as
long as it survives.
         *
         *  Result: Works as intended.
         *
         *
         *
         */
        //

        sensor.AddObservation(0f);
    }
```

Figura 4.15 Observaciones que hace el agente (Fuente: Propia)

```
    /// <summary>
    /// Called when an action is recieved from either the player input or the
neural network
    ///
    /// vectorAction[i] represents:
    /// Index 0: Turn Left, Right or No-Turn ( 1 Left, 2 Right, 3 No-Turn)
    ///
    /// </summary>
    /// <param name="vectorAction">The actions to take</param>
    public override void OnActionReceived(float[] vectorAction)
    {
        float turnDir;

        switch (vectorAction[0])
        {
            case 1:
                turnDir = -turnAngle;
                break;
            case 2:
                turnDir = turnAngle;
                break;
            case 3: //Don't turn
            default:
                turnDir = 0f;
                break;
        }
        //If it's alive
        if (isAlive)
        {
            //If there is no cooldown the Cycle can turn
            if (turnDir != 0f && !onCooldown)
            {
                this.transform.Rotate(this.transform.up, turnDir);

                GameObject wall = Instantiate(turnPrefab,
this.transform.position, this.transform.rotation);
                generatedWall.Add(wall);

                //Refresh the cooldown
                onCooldown = true;
                Invoke("RefreshCooldown", cooldown);
                //vectorAction[0] = 0;
            }
            else
            {
                GameObject wall = Instantiate(wallPrefab,
this.transform.position, this.transform.rotation);
                generatedWall.Add(wall);
            }
            //Always moving forward
            this.transform.Translate(0, 0, movementSpeed * Time.deltaTime);
            if (generatedWall.Count > 500 && dynamicWall)
            {
                GameObject wall = generatedWall[0];
                generatedWall.Remove(wall);
                Destroy(wall);
            }
        }
        if (isAlive && isTraining)
        {
            AddReward(0.01f);
    }
```

Figura 4.16 Acciones a realizar según que devuelva la red neuronal (Fuente: Propia)

Una vez editado el código y añadido a nuestro agente, se le añadirán de forma automática otros dos scripts que se encargarán de definir las observaciones y las acciones que recibe y devuelve la red neuronal, así como de la comunicación con Python.

4.4.2.8 Entrenamiento

Una vez que hemos configurado la IA para que sea capaz de emplear ML-Agents, pasamos al entrenamiento.

Para realizar el entrenamiento se han usado:

- Python el cual se encarga de llevar a cabo el entrenamiento conectándose con Unity a través del paquete ML-Agents,
- Pytorch que es un marco de trabajo de Python para machine learning requerido por ML-Agents,
- Y Conda para manejar los entornos y la instalación de librerías.
- Se ha empleado también CUDA para usar la tarjeta gráfica en vez de la CPU a la hora de entrenar.

Antes de lanzar el entrenamiento desde la consola deberemos editar un archivo *.yaml* que contendrá la configuración de la red neuronal para entrenar y donde se puede diferenciar la configuración entre diferentes comportamientos que se hayan determinado en nuestro programa. Tras unas cuantas pruebas iniciales se ha decidido mantener la configuración por defecto y solo variar la duración del entrenamiento si los resultados tras la duración base son buenos.

En Unity hemos creado una escena dedicada al entrenamiento, donde tenemos 9 entornos idénticos para que entrenen los agentes más rápido.

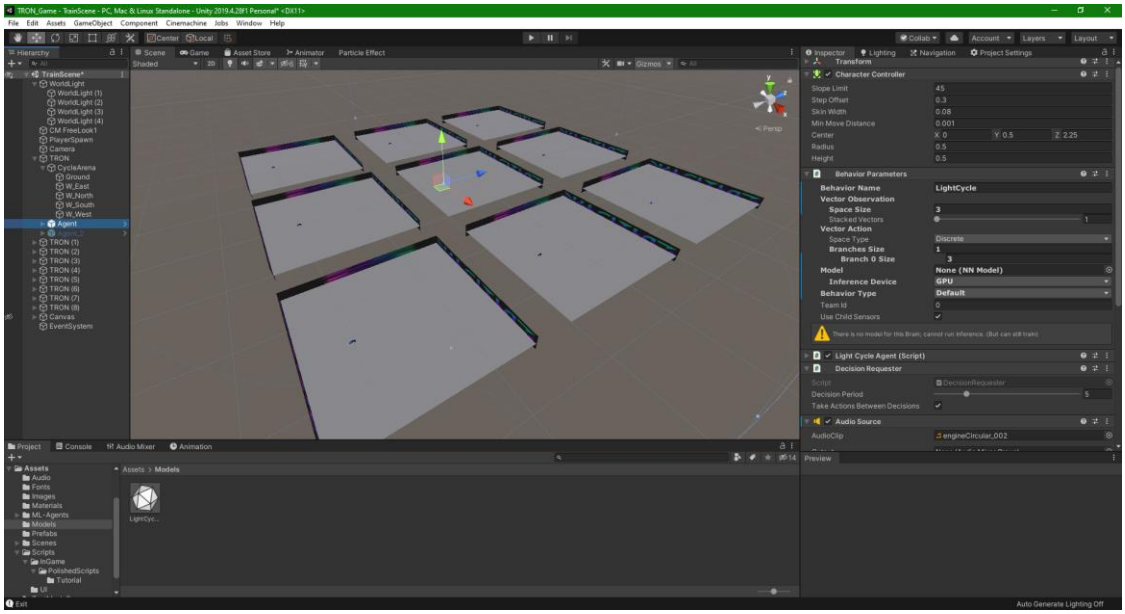


Figura 4.17 Escena de entrenamiento en Unity (Fuente: Propia)

Una vez ejecutado ML-Agents en Python, este se quedará escuchando en un puerto a la espera de que se ejecute Unity. Al lanzar Unity, ML-Agents intenta conectarse a ese puerto y cuando lo hace empieza a entrenar.

Figura 4.18 ML-Agents esperando a que se inicie el entrenamiento (Fuente: Propia)

Cuando acabe, se habrá generado un modelo de red neuronal con formato compatible con Unity que podemos asignar a nuestros agentes y un sumario de los

resultados que podemos observar utilizando Tensorboard, una herramienta de visualización de métricas dirigido a machine learning.

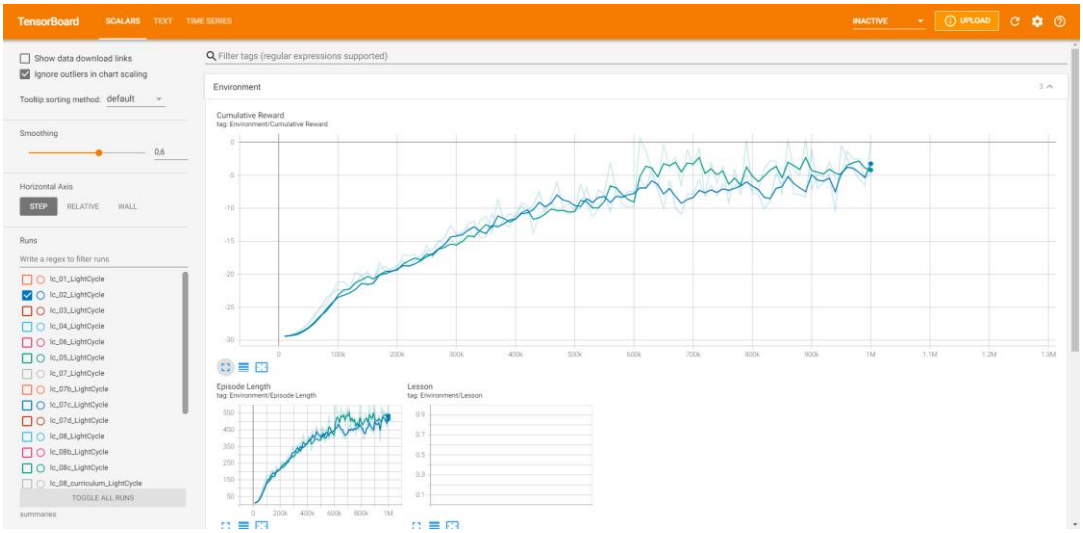


Figura 4.19 Gráficas de 2 entrenamientos vistos en Tensorboard (Fuente: Propia)

4.4.2.9 UI: Menú Principal, Opciones y Controles

Cuando se abre la aplicación no se lanzará el juego directamente, sino que nos encontraremos con un menú principal desde el cual podremos decidir si jugar, modificar la configuración del juego o salir de la aplicación.

El menú no es una parte esencial del sistema, pero ha de ser cómodo y sencillo de navegar. Se ha diseñado un menú simple para asegurar que cumplimos estos requisitos y no tener problemas de navegación.

En el desarrollo del apartado UI se ha empleado el paquete TextmeshPro que facilita la personalización y ajuste de la interfaz gráfica, por ejemplo: añadir gradientes y efectos a los textos o personalizar la fuente que se emplea y su apariencia.

Figura 4.20 Etiqueta editada con TextmeshPro (Fuente: Propia)

Consistirá en una imagen de fondo que tenga relación o estética de TRON, y una lista vertical con las diferentes opciones de navegación: "Jugar, Opciones, Controles y Salir", seleccionando "Jugar" cambiará la escena y empezará el juego como tal y "Salir" cerrará la aplicación; las otras dos opciones ocultarán la lista actual y mostrarán su respectiva información de la misma manera que el menú principal.

Figura 4.21 Menú principal del juego (Fuente: Propia)

El menú de opciones tendrá varios elementos que modificar y que cambiarán la forma en la que se ve el juego acorde a los cambios realizados a través de utilidades propias de Unity.

Figura 4.22 Menú de opciones del juego (Fuente: Propia)

El menú de controles mostrará cuales son los inputs de las diferentes acciones que tiene el jugador mientras juega.

Figura 4.23 Menú de controles del juego (Fuente: Propia)

4.4.2.10 UI: Menú Pausa

Una vez que el usuario pasa al juego los menús dejan de importar y toma importancia la jugabilidad. Pero aun así tiene que existir una manera de detener o salir del juego sin tener que acabar la partida, para comodidad del jugador. Ahí es cuando entra en juego el menú de pausa.

Al abrir el menú de pausa el juego se detendrá completamente y se le mostrará al jugador dos opciones:

- Resumir el juego, opción que reanudará el juego de inmediato y el usuario podrá seguir jugando.
- Salir al menú principal, opción que cerrará el juego y devolverá al jugador al menú principal.

Figura 4.24 Menú de pausa (Fuente: Propia)

4.4.2.11 Efectos de sonido y Música

Estos últimos puntos tratan puntos que aún importantes en un videojuego en este proyecto tienen menos presencia.

Para la música y efectos de sonido se ha creado un controlador de audio simple que controla qué sonidos se reproducen y cuándo lo hacen. Manteniendo así la música constante tanto en el menú como en el juego, pero desactivando los efectos de sonido de las motos en el menú principal.

Como nota adicional en este punto: la música que se ha empleado ha sido creada empleando la demo web de MuseNet [16], una red neuronal profunda que puede generar cortas composiciones musicales en diferentes estilos empleando varios instrumentos.

4.4.2.12 Texturas

Al igual que el punto anterior, este apartado se ha tocado brevemente. En orden de ambientar un poco el juego, se ha empleado GIMP 2.0, un programa de edición de imágenes y fotografías gratuito, para crear unas pocas texturas que colocar en la arena y con las que generar el skybox de la escena para simular el estilo y ambiente de la película.

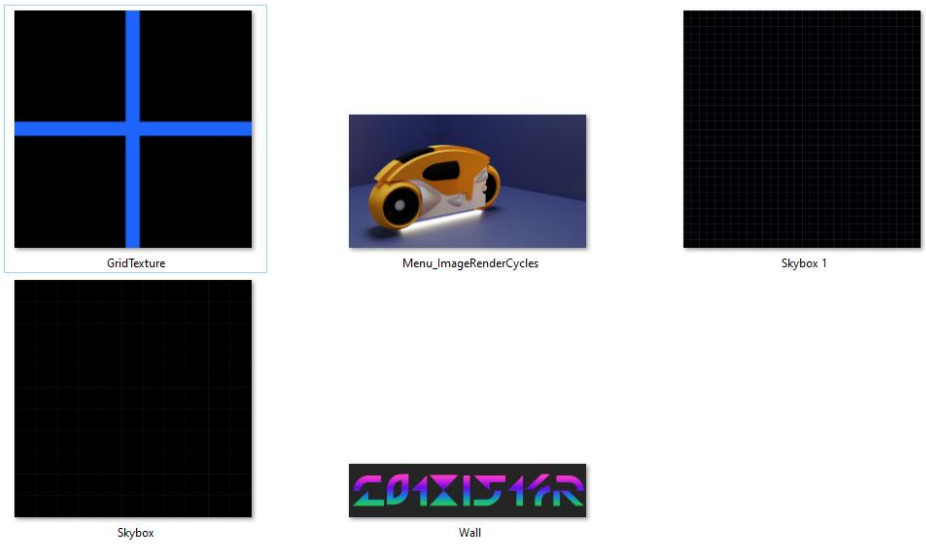


Figura 4.25 Texturas del videojuego

4.4.3 Pruebas

Las pruebas que se han realizado han sido ejecuciones prácticas donde la parte implementada ha sido forzada por el tester intentando encontrar casos límite donde la funcionalidad falle.

4.4.3.1 Pruebas unitarias

- **Test Movimiento 1:** Al lanzarse el programa se espera que el agente avance sin necesidad de ningún input.
- **Test Movimiento 2:** Se espera que el agente siga avanzando tras un giro.
- **Test Giro 1:** Al presionar la flecha izquierda el agente realiza correctamente un giro de 90º hacia la izquierda.
- **Test Giro 2:** Al presionar repetidamente la flecha izquierda el agente realiza correctamente sucesivos giros de 90º hacia la izquierda

- **Test Giro 3:** Al presionar la flecha derecha el agente realiza correctamente un giro de 90º hacia la derecha.
- **Test Giro 4:** Al presionar repetidamente la flecha derecha el agente realiza correctamente sucesivos giros de 90º hacia la derecha.
- **Test Giro 5:** Al ir alternando las teclas presionadas se espera que el agente se mueva en un patrón de escalera.
- **Test Muro 1:** Se espera que el muro se genere siguiendo el recorrido en línea recta del agente.
- **Test Muro 2:** Se espera que el muro realice una esquina perfecta siguiendo el rastro del agente cuando gira.
- **Test Muro 3:** Se espera que el muro parezca un elemento continuo.
- **Test Destrucción 1:** Se espera que el agente se destruya cuando choca contra las paredes de la arena.
- **Test Destrucción 2:** Se espera que el agente se destruya cuando choca contra su propio muro.
- **Test Destrucción 3:** Se espera que el agente se destruya cuando choca contra el muro de otro agente.
- **Test Destrucción 4:** Se espera que el muro generado por el agente se destruya a la vez que el agente.
- **Test UI 1:** Se espera que el botón “Jugar” inicie el juego.
- **Test UI 2:** Se espera que el botón “Opciones” oculte el menú principal y muestre el menú de opciones.
- **Test UI 3:** Se espera que el botón “Controles” oculte el menú principal y muestre el menú de opciones.
- **Test UI 4:** Se espera que el botón “Salir” cierre la aplicación.
- **Test UI 5:** Se espera que el slider “Volumen” modifique acorde los valores del controlador de audio.

- **Test UI 6:** Se espera que la selección de “Pantalla completa” cambie el programa a formato de pantalla completa.
- **Test UI 7:** Se espera que la no selección de “Pantalla completa” cambie el programa a formato ventana.
- **Test UI 8:** Se espera que la lista desplegable “Calidad de gráficos” muestre correctamente las opciones disponibles.
- **Test UI 9:** Se espera que la lista desplegable “Calidad de gráficos” muestre por defecto la opción activa.
- **Test UI 10:** Se espera que al seleccionar un elemento de la lista desplegable “Calidad de gráficos” esa configuración pase a estar activa.
- **Test UI 11:** Se espera que la lista desplegable “Resolución” muestre correctamente las opciones disponibles.
- **Test UI 12:** Se espera que la lista desplegable “Resolución” muestre por defecto la opción activa.
- **Test UI 13:** Se espera que al seleccionar un elemento de la lista desplegable “Resolución” esa configuración pase a estar activa.
- **Test UI 14:** Se espera que el botón “Atrás” oculte el menú actual y muestre el menú principal.
- **Test UI 15:** Se espera que la UI mantenga su posición al cambiar la resolución del juego.
- **Test Pausa 1:** Se espera que el menú de pausa se active pulsando la tecla “escape”.
- **Test Pausa 2:** Se espera que el menú de pausa se cierre pulsando la tecla “escape”.
- **Test Pausa 3:** Se espera que el juego se detenga cuando el menú de pausa esté activo.
- **Test Pausa 4:** Se espera que el juego se reanude cuando el menú de pausa se cierre.
- **Test Pausa 5:** Se espera que el botón “Reanudar” cierre el menú de pausa.

- **Test Pausa 6:** Se espera que el botón “Salir” cierre el juego y cambie la escena al menú principal.
- **Test Victoria 1:** Se espera que al destruirse todos los agentes enemigos el juego se pause y muestre un cartel de victoria.
- **Test Victoria 2:** Se espera que el cartel de victoria devuelva al usuario al menú principal al hacer click.
- **Test Derrota 1:** Se espera que al destruirse el jugador el juego se pause y muestre un cartel de derrota.
- **Test Derrota 2:** Se espera que el cartel de derrota devuelva al usuario al menú principal al hacer click.

4.4.3.2 Pruebas de Sistema:

- **Prueba 1:** Seguir los pasos esperados de una ejecución normal para comprobar un correcto funcionamiento del sistema por defecto: Abrir la aplicación > navegar el menú principal > lanzar el juego > jugar > volver al menú principal > salir.
- **Prueba 2:** Se añade el paso intermedio de modificar las opciones a valores diferentes a los predeterminados comprobar un correcto funcionamiento del sistema una vez modificado: Abrir la aplicación > navegar el menú principal > abrir opciones > modificar la configuración > volver al menú principal> lanzar el juego > jugar > volver al menú principal > salir.
- **Prueba 3:** Se comprueba la persistencia de los cambios entre ejecuciones: Abrir el juego > navegar el menú principal > abrir opciones > modificar la configuración > volver al menú > salir > volver a abrir el juego > navegar el menú principal > lanzar el juego > jugar.

4.5 EL PRODUCTO DEL DESARROLLO

A continuación, se mostrarán diferentes capturas del producto final en funcionamiento:

Figura 4.26 Ejemplo de inicio del juego (Fuente: Propia)

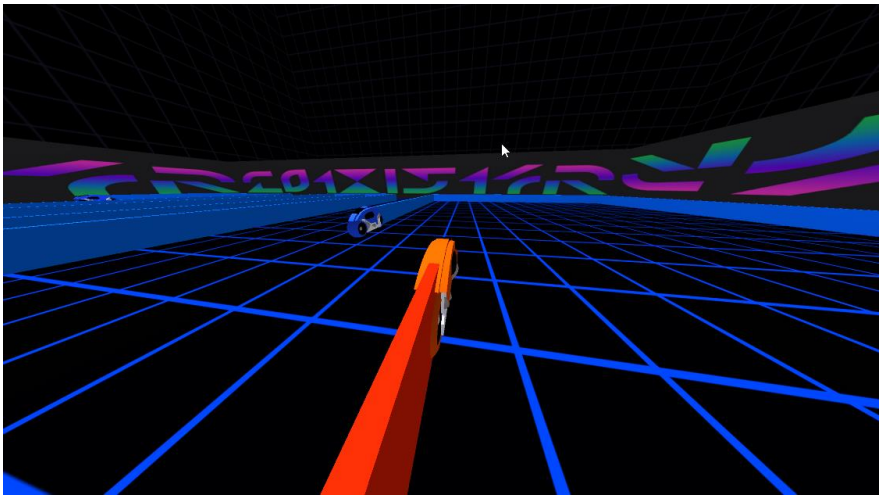


Figura 4.27 Ejemplo de ejecución del juego (Fuente: Propia)

Figura 4.28 Ejemplo victoria en el juego (Fuente: Propia)

5 Evaluación

En este apartado se demuestra la validez de la solución, teniendo encuentra si se han completado los suficientes requisitos y objetivos establecidos. En primer lugar, se valorará desde la funcionalidad de cada componente del sistema y a continuación del resultado final valorando el sistema entero.

5.1 PROCESO DE EVALUACIÓN

Como punto de evaluación se ha tenido en cuenta si el programa funciona correctamente y las tecnologías se han aplicado correctamente al desarrollo del proyecto.

A mayores, se ha compartido el proyecto con un número reducido de conocidos (10) cuyo nivel y conocimiento del medio varían en gran medida entre ellos, para obtener una opinión variada de los aspectos del juego.

A los que han participado se les ha pedido una opinión abierta del juego y que respondiesen a las siguientes preguntas:

- ¿Has tenido algún problema de ejecución con el juego como pueda ser que no se abra o crashee?
- ¿Algún problema de rendimiento durante la partida?
- ¿Has ganado alguna partida?
- ¿Qué opinas respecto a la dificultad?
- ¿Algún problema con el control?

5.2 ANÁLISIS DE RESULTADOS

Al final del desarrollo se puede observar que el juego funciona correctamente y que la aplicación de machine learning a los enemigos ha sido satisfactoria, al funcionar tan bien o mejor que su contraparte tradicional.

De las encuestas se han obtenido las siguientes respuestas:

- Ninguno de ellos ha tenido problemas ejecutando el juego
- Ninguno de ellos ha tenido problemas entendiendo su funcionamiento.

- Todos han conseguido ganar alguna partida sin mucha dificultad.
- La opinión general es que la sensación de velocidad y control es decente.
- Gran parte echa en falta algún atajo de teclado para navegar los menús.

6 Conclusión

6.1 APORTACIONES REALIZADAS

Una vez finalizado el trabajo podemos sacar una serie de conclusiones con respecto al mismo.

El uso de machine learning para controlar algún aspecto de un videojuego es una idea con mucho potencial para crear experiencias diferentes a las que producen las formas usuales, pero a cambio es más complicado conseguir llegar a una funcionalidad básica desde la cual explorar, por lo que los métodos clásicos como las “máquinas de estados finitos” son todavía bastante más confiables y fáciles de configurar.

En cuanto al proyecto, hemos alcanzado el objetivo principal, creando un videojuego simple y funcional, en el que tanto las mecánicas del juego como el estilo visual son un claro reflejo de la película TRON en la que se inspira. El juego ofrece un pequeño momento de entretenimiento y su fácil acceso lo convierte en algo casual a lo que se podría jugar durante esperas cortas. Un buen reflejo del juego que lo inspiró.

6.2 TRABAJOS FUTUROS

Este proyecto se encuentra en una fase temprana y tiene un gran margen de mejora:

- Mejorando la sensación de kinestesia del juego.
- Añadir mecánicas menores al movimiento: control de velocidad o añadir poderes, por ejemplo.
- Añadir mecánicas mayores a la jugabilidad manteniendo las bases de movimiento.
- Mejorar la IA para que aguanten más, aprendan comportamientos más complejos o incluso que los agentes desarrollen estrategias de grupo.
- Añadir diferentes entornos en los que pelear y desafíos.
- Añadir personalización a los vehículos.

- Cambiar el modelo de jugabilidad a un juego arcade de puntuación por supervivencia.

6.3 PROBLEMAS ENCONTRADOS

Los principales problemas que a los que me he enfrentado han sido:

- Problemas de comunicación entre diferentes tecnologías con diferentes versiones.
  La versión de ML-Agents actual no se corresponde con las versiones más actuales de Python y Pytorch, así que he tenido que reinstalar varias veces los diferentes componentes especificando las versiones que necesitaba.
- La potencia de mi ordenador. A la hora de entrenar la IA la limitación en potencia de computación se ha notado, los entrenamientos con un número relativamente bajo de pasos tardaban alrededor de 3h.
- Entrenamiento por currículum. Este tipo de entrenamiento permite ir aumentando la complejidad del problema al que la IA se enfrenta de forma progresiva llegando así a comportamientos más elaborados. Pero no he conseguido que funcione correctamente ni siquiera en los proyectos de ejemplo donde viene tanto el código como los documentos de configuración completamente hechos.
- Fallo crítico del ordenador que impedía ejecutar cualquier programa a excepción del explorador de Windows y que resultó en la necesidad de formatear el ordenador y reinstalar Windows desde 0.

6.4 OPINIONES PERSONALES

Este proyecto me ha permitido aprender múltiples herramientas que apenas había empleado antes y aprender a diseñar y programar un software que utiliza tecnologías y ciertas formas que desconocía.

Además, he ampliado mis conocimientos en el campo de machine learning, al realizar aprendizaje por refuerzo el cual conocía conceptualmente pero nunca había empleado.

Bibliografía

[1] Tron, «Wikipedia,» 24 Julio 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Tron. [Último acceso: 03 Septiembre 2021].

[2] OpenAI Five, «OpenAI © 2015–2021,» [En línea]. Available: https://openai.com/five/. [Último acceso: 08 Septiembre 2021].

[3] Hello Neighbor 2, «Youtube,» 26 Marzo 2021. [En línea]. Available: https://youtu.be/Hlj2SyGnlvc. [Último acceso: 08 Septiembre 2021].

[4] E. J. Hastings, G. K. Ratan y S. O. Kenneth, «Evolving content in the Galactic Arms Race video game,» 2009. [En línea]. Available: https://ieeexplore.ieee.org/document/5286468. [Último acceso: 10 Noviembre 2021].

[5] d. M. Xav, «Meet the computer that's learning to kill and the man who programmed the chaos,» 7 Junio 2014. [En línea]. Available: https://www.engadget.com/2014-06-06-meet-the-computer-thats-learning-to-kill-and-the-man-who-progra.html?guce_referrer=aHR0cHM6Ly9lcy53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAACTr96KAJscqj9JrZfAK8Oh4Xd2j_DLbW4vxtEFIo7VpvuxcvVmE6PuIT3ynPWUJ8Xu-6s9jPTi5TkH. [Último acceso: 10 Noviembre 2021].

[6] Maya, «Autodesk.es,» [En línea]. Available: https://www.autodesk.es/products/maya/overview. [Último acceso: 29 Enero 2021].

[7] Blender, «Blender.org,» [En línea]. Available: https://www.blender.org/. [Último acceso: 25 Enero 2021].

[8] A. Price, «Youtube,» Blender Guru, 09 Octubre 2020. [En línea]. Available: https://youtube.com/playlist?list=PLjEaoINr3zgEq0u2MzVgAaHEBt--xLB6U. [Último acceso: 10 Marzo 2021].

[9] U. Engine, «Epic Games, Inc. 2004-2021,» [En línea]. Available: https://www.unrealengine.com/en-US/. [Último acceso: 01 Febrero 2021].

[10] P. Santamaría, «el output,» 16 Diciembre 2019. [En línea]. Available: https://eloutput.com/noticias/cultura-geek/the-mandalorian-vfx-unreal-engine/. [Último acceso: 03 Septiembre 2021].

[11] Unity, «Unity Technologies,» [En línea]. Available: https://unity.com/es. [Último acceso: 01 Febrero 2021].

[12] Unity Technologies, «unity Learn,» 09 Diciembre 2020. [En línea]. Available: https://learn.unity.com/. [Último acceso: 20 Febrero 2021].

[13] G. Ognjanovski, «Everything you need to know about Neural Networks and Backpropagation,» 14 Enero 2019. [En línea]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a. [Último acceso: 9 Noviembre 2021].

[14] OpenAI Gym, «OpenAI © 2015–2021,» [En línea]. Available: https://gym.openai.com/. [Último acceso: 01 Julio 2021].

[15] ML-Agents, «© 2021 Unity Technologies,» [En línea]. Available: https://unity.com/es/products/machine-learning-agents. [Último acceso: Junio 2021].

[16] C. Payne, «OpenAI,» 25 Abril 2019. [En línea]. Available: https://openai.com/blog/musenet/. [Último acceso: 20 Agosto 2021].

[17] Brackeys, «Youtube,» 24 Mayo 2020. [En línea]. Available: https://www.youtube.com/watch?v=4HpC--2iowE&ab_channel=Brackeys. [Último acceso: 15 Mayo 2021].

[18] P. d. Byl, «unity Learn,» 06 Abril 2021. [En línea]. Available: https://learn.unity.com/course/artificial-intelligence-for-beginners?language=en. [Último acceso: 01 Mayo 2021].

[19] Immersive Limit LLC, «unity Learn,» 20 Octubre 2020. [En línea]. Available: https://learn.unity.com/course/ml-agents-hummingbirds?language=en. [Último acceso: 19 Julio 2021].

[20] Brackeys, «Youtube,» 29 Noviembre 2017. [En línea]. Available: https://youtu.be/zc8ac_qUXQY. [Último acceso: 02 Agosto 2021].

[21] Brackeys, «Youtube,» 20 Diciembre 2017. [En línea]. Available: https://youtu.be/JivuXdrIHK0. [Último acceso: 05 Agosto 2021].

[22] Brackeys, «Youtube,» 6 Diciembre 2017. [En línea]. Available: https://youtu.be/YOaYQrN1oYQ. [Último acceso: 04 Agosto 2021].

[23] Brackeys, «Youtube,» 31 Mayo 2017. [En línea]. Available: https://youtu.be/6OT43pvUyfY. [Último acceso: 18 Agosto 2021].

[24] Tron, «Youtube,» 18 Mayo 2012. [En línea]. Available: https://www.youtube.com/watch?v=Syv1w2L8i68&ab_channel=YouTubeMovies. [Último acceso: 10 Septiembre 2021].

ANEXO A: Control de Versiones

Durante el desarrollo de este proyecto se ha empleado Git como controlador de versiones. Se ha creado un repositorio local empleado las herramientas para Git que ofrece VisualStudio 2019 donde se ha ido actualizando el proyecto a medida que avanzaba.

VisualStudio permite controlar el repositorio a través de interfaz gráfica pudiendo seleccionar que archivos añadir, cuadro de texto para detallar la información del commit, y ver que archivos están actualizados en el repositorio y cuales no están añadidos o han sido modificados. También contiene herramientas para cambiar de versión de archivo a otro en el repositorio, así como solucionar problemas de fusión de código si se diesen.

Se ha establecido como norma general realizar un *commit* tras la creación de una nueva clase y sus atributos, y tras la implementación de un nuevo método y/o modificación de una función existente.

El repositorio solo consta de una rama principal donde se ha ido actualizando el proyecto.

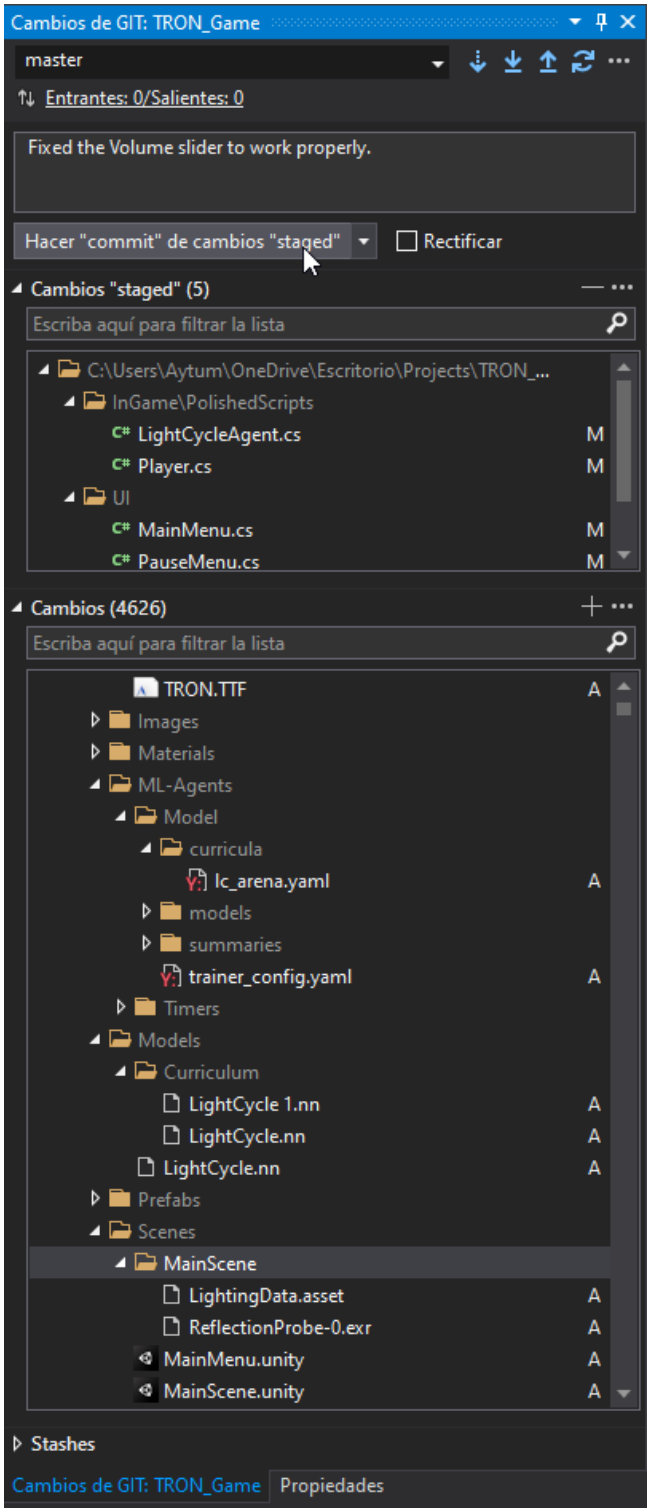


Figura A.1 Repositorio Git local del proyecto visto en VisualStudio y ejemplo de commit (Fuente: Propia)

ANEXO B: Seguimiento de proyecto fin de carrera

El seguimiento del proyecto se ha realizado a través de reuniones semanales con la tutora. Las primeras reuniones estuvieron centradas en explicar que se había realizado ya: la idea detrás del proyecto, tecnologías que se emplean, diseño y explicación del código y soluciones desarrolladas. A partir de las explicaciones iniciales se pasó a comentar y enseñar lo que se había hecho durante la semana, así como que se pretendía hacer a continuación. En estas reuniones también se han consultado problemas que no se habían resuelto entre reuniones y ayudado a resolver los mismos, además de dudas y anotaciones a la hora de redactar esta memoria.

ANEXO C: Manual de usuario

En este anexo se explica como ejecutar la aplicación desarrollada y como desenvolverse en la misma.

- **Ejecutar la aplicación:**

Para ejecutar el juego no es necesario realizar ninguna instalación, simplemente lanzar la aplicación debería iniciar el juego correctamente y dirigir al usuario al menú principal.

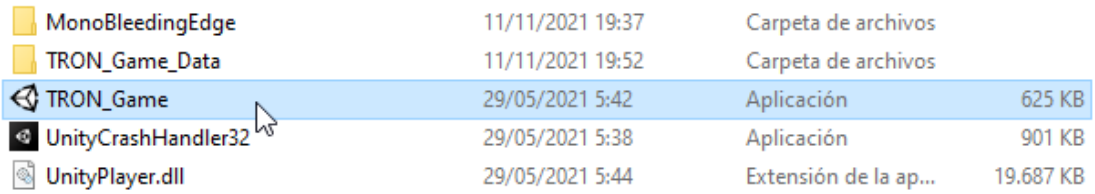


Figura C.1 Carpeta del juego y sus archivos (Fuente: Propia)

- **Usar la aplicación:**

Al abrir la aplicación nos encontraremos con el menú principal (Figura C.2) el cual será el principal nexo de navegación del juego. Todos los menús del juego se navegan usando el ratón y haciendo click en la opción deseada.

Desde el menú principal podemos elegir entre las diferentes opciones:

Jugar: Al seleccionar este botón cambiaremos de escena y pasaremos a la parte jugable. Nuestro objetivo será sobrevivir hasta que todos los enemigos hayan muerto. Para ello tendremos que evitar chocar con cualquier obstáculo que nos encontremos delante girando a izquierda o derecha, a la vez que intentamos atrapar a los enemigos para que ellos choquen. La “Moto de Luz” no se detendrá en ningún momento por lo que tendremos que emplear reflejos y algo de táctica para lograr ganar.

Figura C.2 Menú principal al abrir la aplicación (Fuente: Propia)

En la parte jugable dejaremos de controlar la aplicación con el ratón, y pasaremos a manejarla empleando el teclado. Mientras se juega usaremos las flechas de dirección izquierda y derecha para hacer girar nuestra "Moto de Luz" hacia la dirección deseada. Al seleccionar la dirección giraremos en un ángulo de 90º instantáneamente (Figura C.3).

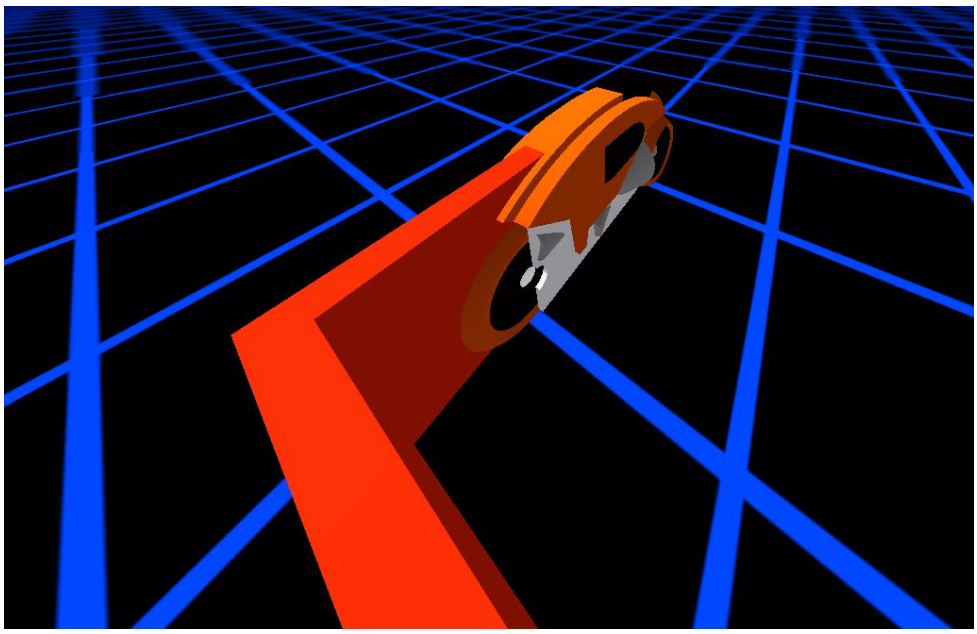


Figura C.3 Movimiento de la "Moto de Luz" en el juego (Fuente: Propia)

Aparte el jugador podrá detener el juego en cualquier momento presionando la tecla "escape" y entrar al menú de pausa (Figura x). Este menú permite realizar

un descanso del juego momentáneo y resumir la partida, o abandonarla definitivamente y regresar al menú principal.

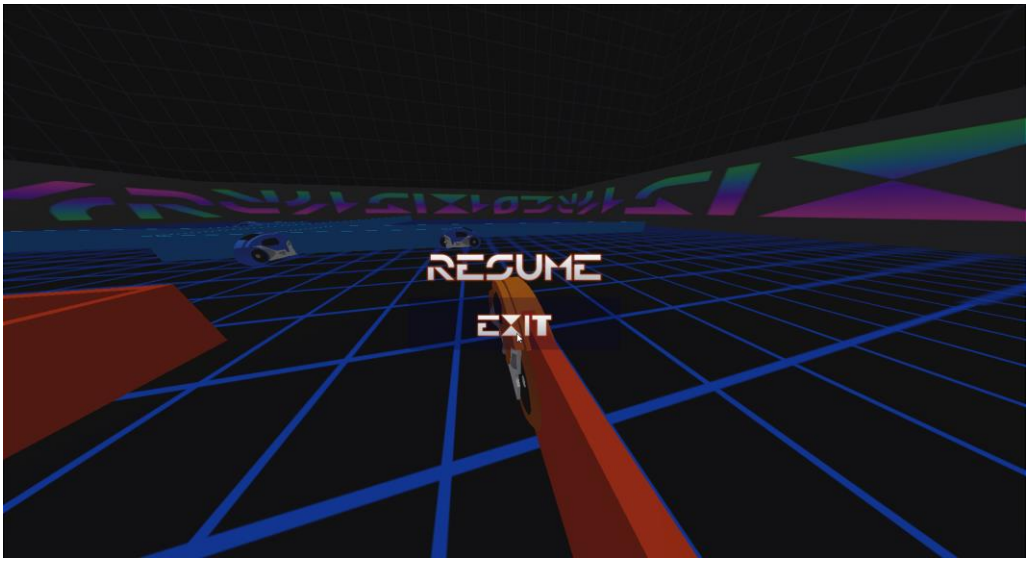


Figura C.4 Menú de pausa en el juego (Fuente: Propia)

Opciones: Este menú nos permite personalizar la apariencia gráfica del juego. A continuación, se detallará que hace cada una de las opciones:

- *Volumen:* Modificando el valor de la barra podremos reducir o incrementar el volumen con el que se reproduce el audio del juego.

Figura C.5 Slider de Volumen (Fuente: Propia)

- *Resolución:* Al hacer click en esta opción se desplegará una lista de las diferentes resoluciones disponibles. Cuando seleccionemos uno de los elementos, la resolución a la que se ve la aplicación se modificará acorde al valor elegido. Esta opción puede modificar sustancialmente el rendimiento una aplicación.

Figura C.6 Cuadro desplegable con las diferentes resoluciones disponibles (Fuente: Propia)

- *Calidad:* Al igual que el punto anterior, hacer click en esta opción desplegará otra lista donde se podrá elegir la calidad a la que el motor gráfico renderizará el juego. Esta opción puede modificar sustancialmente el rendimiento una aplicación.

Figura C.7 Cuadro desplegable con las diferentes opciones de calidad gráfica disponibles (Fuente: Propia)

- *Pantalla completa:* Marcar o desmarcar esta opción hará que la aplicación se ejecute en pantalla completa o modo ventana respectivamente.

Figura C.8 Cuadro seleccionable de pantalla completa (Fuente: Propia)

Controles (Figura C.9): En este menú se presentan al jugador que teclas se emplean para controlar la aplicación una vez se pasa al juego:

- Flecha izquierda (<): para girar a la izquierda.
- Flecha derecha (>): para girar a la derecha.
- Escape (*esc*): para pausar el juego.

Figura C.9 Información de los controles en la aplicación (Fuente: Propia)

Salir: Este botón cierra la aplicación y nos devuelve al escritorio.