



“Combate Singular”, uso de IA en los videojuegos. Profesionalizador (TFP)

Autor: David Pastor Rubio

Tutor: Xavier Pallicera Malivern

Profesor: José Arnedo Moreno

Grado en ingeniería informática
Área de Sistemas de Información.



Esta obra está sujeta a una licencia de Reconocimiento- NoComercial-SinObraDerivada [3.0 España de Creative Commons.](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Copyright © 2022 David Pastor Rubio.

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>"Combate Singular", uso de IA en videojuegos.</i>
Nombre del autor:	<i>David Pastor Rubio</i>
Nombre del colaborador/a docente :	<i>Xavier Pallicera Malivern</i>
Nombre del PRA:	<i>José Arnedo Moreno</i>
Fecha de entrega (mm/aaaa):	<i>06/2022</i>
Titulación o programa:	<i>Ingeniería informática</i>
Área del Trabajo Final:	<i>Videojuegos</i>
Idioma del trabajo:	<i>Español</i>
Palabras clave	<i>IA, Videojuego, Machine Learning</i>
Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo</i>	
<p>Con la intención de investigar diferentes metodologías de generación de IA para juegos de combate, se ha generado un entorno sobre el que, en un primer término, dos jugadores puedan realizar combates utilizando el mismo ordenador. El siguiente paso, ha sido crear una IA, utilizando la técnica de árboles de comportamiento. Para finalizar, y utilizando el módulo de machine learning, el cual posee la plataforma de desarrollo, se ha entrenado una nueva IA, utilizando la programada.</p> <p>Como último paso de nuestro trabajo, se ha realizado una comparativa entre las IA, la que ha sido programada y la entrenada, por medio de estadísticas de combate con jugadores reales. El juego se ha desarrollado en Unity, usando assets tanto comerciales como gratuitos. Para el desarrollo de la IA basada en arboles de decisión, se ha generado toda, programáticamente. En relación al entrenamiento de la IA, se ha usado el módulo de Machine Learning de Unity ML-Agents.</p> <p>La dinámica del juego, ambientada en el medievo, es, la de ir paulatinamente ganando a los adversarios de diferentes territorios, para al final, cuando todos ellos queden derrotados, hacernos con el control del Reino. En cada territorio, nos encontraremos con diferentes tipos de adversarios, con diferentes habilidades y aspectos.</p>	
Abstract (in English, 250 words or less):	
<p>With the aim of research different methodologies for generating an AI for fighting games, it has been created an environment for two players. The next step, it has been creating an AI using the Behaviour Tree technique. For finalizing, and using the machine learning library of the framework, we have trained a new AI, using the previous programmed AI.</p>	

Our last step of the TFG, it has been to do a comparative between the two AI developed through statistics with real players. The game has been developed with Unity using free and commercial assets. The AI Behaviour tree has been created programmatically and regarding the training of the other AI, it has been used the machine learning library of Unity called ML Agents.

The mechanics of the game, set in the middle age, is to win different combats with the best knight of each territory. If you achieve to defeat every opponent you win the game. On each match it will find different opponents and scenarios with different skills and environments.

Agradecimientos

Agradecer a toda mi familia y en especial a mi mujer y a mis hijos, todo el apoyo que me han dado durante estos años de carrera.

Resumen

Con la intención de investigar diferentes metodologías de generación de IA para juegos de combate, se ha generado un entorno sobre el que, en un primer término, dos jugadores puedan realizar combates utilizando el mismo ordenador. El siguiente paso, ha sido crear una IA, utilizando la técnica de árboles de comportamiento. Para finalizar, y utilizando el módulo de machine learning, el cual posee la plataforma de desarrollo, se ha entrenado una nueva IA, utilizando la programada.

Como último paso de nuestro trabajo, se ha realizado una comparativa entre las IA, la que ha sido programada y la entrenada, por medio de estadísticas de combate con jugadores reales. El juego se ha desarrollado en Unity, usando assets tanto comerciales como gratuitos. Para el desarrollo de la IA basada en arboles de decisión, se ha generado toda, programáticamente. En relación al entrenamiento de la IA, se ha usado el módulo de Machine Learning de Unity ML-Agents.

La dinámica del juego, ambientada en el medievo, es, la de ir paulatinamente ganando a los adversarios de diferentes territorios, para al final, cuando todos ellos queden derrotados, hacernos con el control del Reino. En cada territorio, nos encontraremos con diferentes tipos de adversarios, con diferentes habilidades y aspectos.

Abstract

With the aim of research different methodologies for generating an AI for fighting games, it has been created an environment for two players. The next step, it has been creating an AI using the Behaviour Tree technique. For finalizing, and using the machine learning library of the framework, we have trained a new AI, using the previous programmed AI.

Our last step of the TFG, it has been to do a comparative between the two AI developed through statistics with real players. The game has been developed with Unity using free and commercial assets. The AI Behaviour tree has been created programmatically and regarding the training of the other AI, it has been used the machine learning library of Unity called ML Agents.

The mechanics of the game, set in the middle age, is to win different combats with the best knight

of each territory. If you achieve to defeat every opponent you win the game. On each match it will find different opponents and scenarios with different skills and environments.

Palabras clave

Proyecto, Machine Learning, Behavior Tree, Combate, Comparativa.

Índice

1. Introducción.....	12
1.1. Introducción/Prefacio.....	12
1.2. Descripción/Definición	13
1.3. Objetivos generales	15
1.3.1. Objetivos secundarios	16
1.4. Metodología y proceso de trabajo.....	16
1.5. Presupuesto	16
1.6. Planificación.....	17
2. Estado del arte	18
2.1. Antecedentes y evolución de los juegos de lucha.....	18
2.2. Métodos de creación de IA.....	20
3. Propuesta.....	25
3.1. Idea del juego.....	25
3.1.1. Descripción.....	25
3.1.2. Interacción juego-jugador.....	26
3.1.3. Plataforma destino	26
3.2. Conceptualización.....	26
3.2.1. Ambientación del juego	26
3.3. Personajes.....	27
3.3.1. Jugador principal	27
3.3.2. Contrincantes	27
3.3.3. Acciones de los jugadores	27
3.3.4. Efectos especiales	28
3.3.5. Objetos mágicos	28
3.4. Interfaz de usuario.....	29
3.4.1. Introducción	29
3.4.2. Menús de usuario	29
3.4.3. Mapa de los combates.....	31
3.5. Comparativa entre las IA	31
4. Diseño.....	33

4.1. Unity, la plataforma elegida.....	33
4.2. Requerimientos técnicos del desarrollo.....	34
4.3. Herramientas usadas	34
4.4. Recursos utilizados.....	35
4.5. Arquitectura del juego	36
5. Implementación.....	38
5.1. La Clase Movement.....	38
5.2. Las Clases Potions y PotionsActivations	38
5.3. Las Clases Data y DataCombat	39
5.4. Otras clases relacionadas con el juego.....	40
5.5. Clases relacionadas con la GUI.....	40
5.6. Desarrollo IA con árboles de comportamiento.....	40
5.7. Desarrollo de IA con aprendizaje automático.....	45
5.7.1. Sesión de entrenamiento	47
5.8. Comparativa entre las dos IA.....	51
5.8.1. Datos y graficas	51
5.8.2. Conclusiones.....	53
6. Demostración	54
6.1. Requisitos del juego.	54
6.2. Instrucciones del juego	54
7. Conclusiones.....	58
Bibliografía.....	59
Anexos.....	61

Figuras y tablas

Lista de imágenes, tablas, gráficos, diagramas, etc., numeradas, con títulos y las páginas en las cuales aparecen. Para actualizar cada uno de los índices, hay que hacer botón derecho con el ratón y escoger la opción “Actualizar campos”.

Índice de ilustraciones

Ilustración 1: Juego PONG © Atari.....	12
Ilustración 2: Street Fighter © Capcom.....	13
Ilustración 3. Tokens del juego.....	14
Ilustración 4. Personajes y animaciones.....	14
Ilustración 5. Escenario de muestra.....	15
Ilustración 6. Planificación del proyecto.....	17
Ilustración 7. Karate Champ. ©Technos Japan.....	18
Ilustración 8. Mortal Kombat © Midway Games.....	19
Ilustración 9.Shadow Fight 3 © Nekki Limited.....	20
Ilustración 10. Finite States Machine.....	21
Ilustración 11. Árbol de comportamiento.....	22
Ilustración 12. Red Neuronal.....	24
Ilustración 13. Ciclo aprendizaje reforzado.....	24
Ilustración 14. Escenario Hielo.....	26
Ilustración 15. Escenario Fuego.....	26
Ilustración 16. Personaje caminando.....	27
Ilustración 17. Salta de defensa.....	27
Ilustración 18. Personaje herido.....	27
Ilustración 19. Ataques del personaje.....	28
Ilustración 20. Objetos mágicos.....	28
Ilustración 21. Escena de introducción.....	29
Ilustración 22. Menú principal.....	29
Ilustración 23. Opciones del Juego.....	30
Ilustración 24. Modo de combate.....	30
Ilustración 25. Mapa de combates.....	31
Ilustración 26. Gráficas en Unity.....	31
Ilustración 27 KPIs en el juego ML.....	32
Ilustración 28 KPIs en el Juego BT.....	32
Ilustración 29. Arquitectura del juego.....	37
Ilustración 30. Código de animación de objetos.....	39
Ilustración 31. Graduación de dificultad.....	41
Ilustración 32 Árbol de comportamiento.....	43
Ilustración 33. Código del árbol de comportamiento.....	44
Ilustración 34. Relación Código-Árbol.....	44
Ilustración 35. Configuración del comportamiento.....	45

Ilustración 36 Inicio del episodio de entrenamiento.....	46
Ilustración 37 Final del episodio de entrenamiento.	46
Ilustración 38. Castigo impacto en el borde	47
Ilustración 39 Recompensa o castigo según distancia.....	47
Ilustración 40 Entrenamiento en 4 escenarios.	48
Ilustración 41. configuración de la red neuronal.....	48
Ilustración 42. Creación de la red neuronal.....	49
Ilustración 43. Diagrama de la sesión de entrenamiento.	49
Ilustración 44 Uso de la red neuronal.....	50
Ilustración 45. Datos totales estadística.....	51
Ilustración 46 Comparativa puntos.....	51
Ilustración 47 Comparativa jugadas.	52
Ilustración 48 Tendencia tiempos.....	52
Ilustración 49 Tabla de requisitos © Unity.....	54
Ilustración 50 Contenido del directorio.	54
Ilustración 51 Opciones de juego.	55
Ilustración 52 Slider de volumen.	55
Ilustración 53 Nuevo juego.....	56
Ilustración 54 Modo Arena.	56
Ilustración 55 Advertencia de reinicio.....	56
Ilustración 56. Objeto mágico: Vida.....	57
Ilustración 57 Objeto mágico: Poción.....	57
Ilustración 58 Objeto mágico: Hechizo.....	57

Índice de tablas

Tabla 1 Tabla de Assets.	35
Tabla 2 Movimientos personajes.	55

1.Introducción

1.1. Introducción/Prefacio

Ya desde los primeros videojuegos, se han programado diferentes tipos de Inteligencias artificiales. Tenemos por ejemplo los primeros juegos de ajedrez por ordenador que usaba el algoritmo MiniMax, sirviendo como base para el popular Pong, que era capaz de vencernos(solo al principio), jugando al tenis.¹

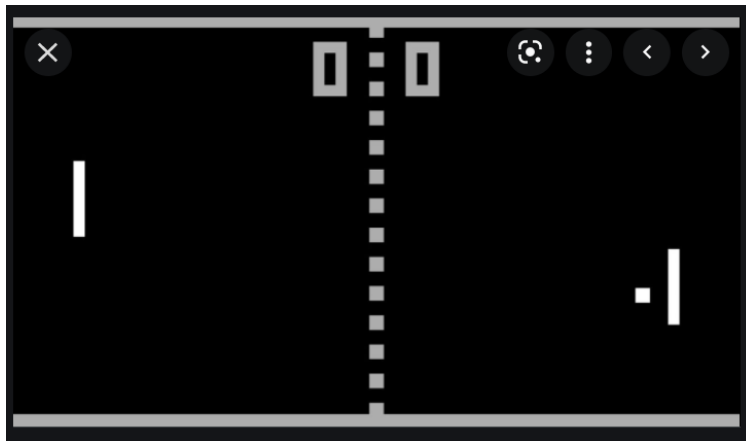


Ilustración 1: Juego PONG © Atari

A lo largo de las décadas diferentes técnicas se han aplicado y han hecho avanzar enormemente el comportamiento de los NPC(Non playable character), como las máquina de estados o los árboles de decisión. Pero recientemente, con el auge de las nuevas técnicas de aprendizaje automático, han hecho que estas nuevas IA aprendan por si solas y lleguen a ser un verdadero desafío.

El denominado Machine Learning, está cada vez más en boga y su aplicación en video juegos también². Mi principal motivación para realizar este juego, es combinar la parte lúdica, con la investigación por mi parte de todo lo que son las nuevas técnicas de aprendizaje automático.

¹ **Midokura.com** (2019)Artificial Intelligence in video games [Artículo en línea] midokura.com [20 de Mayo de 2022] <https://www.midokura.com/artificial-intelligence-in-video-games/>

² **Golinuxcloud.com** (2020) Machine Learning in Video Games [Artículo en línea] midokura.com [20 de Mayo de 2022] <https://www.golinuxcloud.com/machine-learning-in-video-games/#one>

Me gustaría que en este trabajo, tuviera igual o más importancia, la parte del desarrollo de IA. La elección de un juego de combate no ha sido trivial y esta decisión está tomada para poder probar este tipo de técnicas.

1.2. Descripción/Definición

Lo que se pretende con este proyecto, aparte de poner en práctica los conocimientos adquiridos en gestión de proyectos, es la de poder aplicar conceptos teóricos adquiridos en asignaturas como grafos o Inteligencia artificial.

Por medio de la puesta en marcha de un juego de combate en 2D, se pretende, aparte de crear un objeto lúdico, probar diferentes tecnologías de aprendizaje automático y experimentar con ellas. El proyecto tendrá una parte más de juego completo y una parte más dedicada a comprobar la efectividad de las nuevas herramientas de *machine learning* que están siendo usadas en el desarrollo de video juegos.

Queremos comparar como es el comportamiento de una IA de manera programada, frente a una que aprende cada vez que realiza una partida, en el contexto del desarrollo de un videojuego completo.

El juego, ambientado en la Edad Media, consistirá en hacerse con el reino por medio de combates hombre a hombre con el guerrero más importante de cada territorio. En cada una de estas zonas tendremos diferentes contrincantes, con habilidades y aspectos propios. La referencia más directa puede ser el clásico Street Fighter.



Ilustración 2: Street Fighter © Capcom

Al principio del juego, después de la presentación, nos aparecerá un mapa sobre el que podemos pinchar para enfrentarnos al caballero más poderoso de este territorio y de esta manera hacernos con él. Una vez logrados todos los territorios, habremos ganado el juego.

Para darle un poco de animación al juego podrán aparecer pócimas de recuperación, inmortalidad momentánea..etc.



Ilustración 3. Tokens del juego

Tendremos un modo de juego entre dos jugadores humanos, en el que no entrará en juego la conquista del reino y solo será el combate en sí. Una de las opciones que tendremos en la opción de juego para conquistar el reino, será seleccionar el tipo de IA, que podrá ser programada o de aprendizaje automático.

Para la programada utilizaremos la técnica de árboles de comportamiento. El mayor reto en esta parte será crear una IA que sea entretenida, que tenga su dificultad pero que no sea invencible y que en los diferentes territorios en los que luchar, se observen diferencias.

Para la que vamos a entrenar queda la incertidumbre de a que cotas puede llegar, todo depende de la cantidad de entrenamiento que se le dé.

Los *Assets* serán todos adquiridos de la *Unity Asset Store*, bien de forma gratuita, bien de forma pagada. Para los personajes utilizaremos los siguientes que se pueden ver en la ilustración.

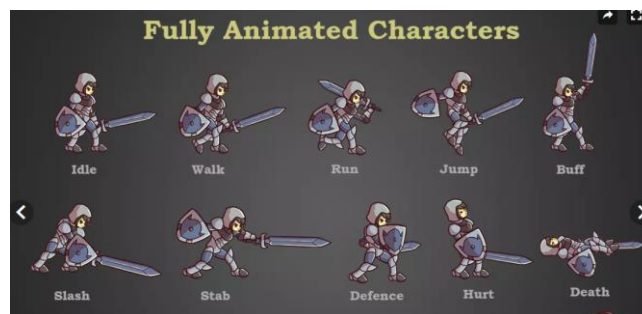


Ilustración 4. Personajes y animaciones

Dado que el objetivo de la asignatura no es ni el diseño ni la animación, utilizaremos la aplicación para la generación de personajes, que viene con el pack. Esta está basada en *Spine*, un paquete de Unity para crear animaciones de personajes.

<https://assetstore.unity.com/packages/2d/characters/character-editor-fantazia-181572>

<http://es.esotericsoftware.com/spine-in-depth>

Utilizaremos diferentes fondos para cada territorio o condado, dándole una característica especial a cada uno. Uno con climatología más lluviosa, otro helado, uno desértico..etc.

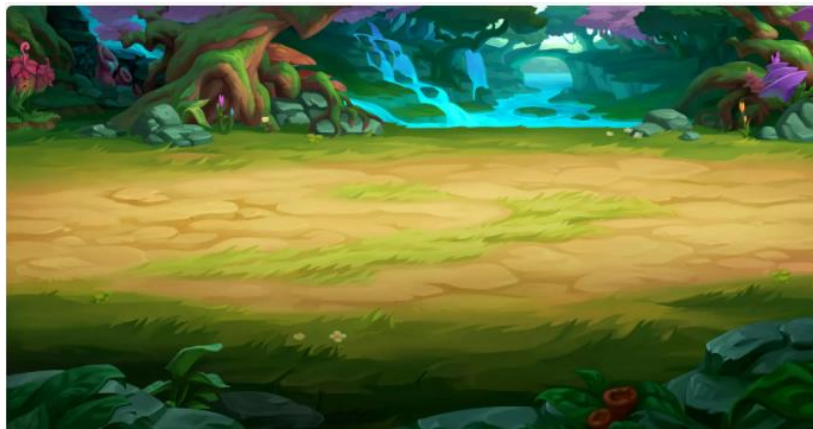


Ilustración 5. Escenario de muestra

1.3. Objetivos generales

Uno de los objetivos principales del proyecto es el del estudio, prueba y comparativa de dos de los métodos que se pueden usar en la generación de personajes controlados por inteligencia artificial en video juegos. Se han elegido uno para desarrollar programáticamente y otro por arboles de decisión. Queremos comprobar que diferencias tenemos en el desarrollo y que resultados nos da cada una de estas soluciones. Debido al tiempo disponible para el desarrollo no se han seleccionado más métodos.

También es **un objetivo general, aparte del anterior, realizar un juego de principio a fin, que sea estéticamente atractivo y que no solo sea el soporte para programar la IA.** Necesitamos que las IA presenten cierto atractivo al jugar con ellas, que sean más fáciles al principio y más difíciles al final. Todo esto supondrá también un reto dado que, sin tener experiencia previa en el desarrollo de videojuegos, hará falta aprender las mecánicas para lograr este objetivo.

1.3.1. Objetivos secundarios

- Aprendizaje de uso de la plataforma Unity.
- Gestión de un proyecto desde su inicio hasta su fin.
- Aprender como es el desarrollo de un videojuego.
- Uso de herramientas de versionado.

1.4. Metodología y proceso de trabajo

Tenemos tres grandes bloques de desarrollo en este proyecto:

1. La parte del juego en sí, con el control de los personajes, el mapa de territorios..etc
2. El desarrollo de la IA con árboles de decisión, la cual tenemos que afinar muy bien para que sea jugable.
3. EL entrenamiento de IA de *Machine Learning*, la cual no podemos avanzar cuál será su comportamiento y será la parte experimental del proyecto.

Con todo esto, lo que queremos es hacer avanzar el proyecto todo a la vez y tener versiones funcionando al acabar cada hito, que identificamos con las entregas de las PEC(menos la primera que es de planificación). Por todo ello, utilizaremos el Modelo de Prototipos, donde cada uno de ellos será presentado al finalizar la PEC.

En cuanto tiempo en cada PEC, dedicaremos al menos la mitad del tiempo a desarrollar y dejar pulido el aspecto y controles del juego y la otra mitad a desarrollar las IA, mejorándolas poco a poco. La idea final es tener el juego acabado lo antes posible y poder pulir el comportamiento de las IA's tanto como se pueda. Debemos tener en cuenta que el desarrollo de estas se puede prolongar indefinidamente ya que siempre cabe posibilidad de mejora y debemos parar, para poder realizar la presentación del juego con los resultados obtenidos.

1.5. Presupuesto

Este trabajo se enmarca dentro del trabajo académico con la finalidad de experimentar con dos tipos de IA y realizar una comparativa. Su fin último no es a comercialización del juego como un producto vendible. El juego actúa como marco de trabajo para realizar el estudio de las IA. Por tanto un estudio de mercado no aplica para este trabajo.

1.6. Planificación

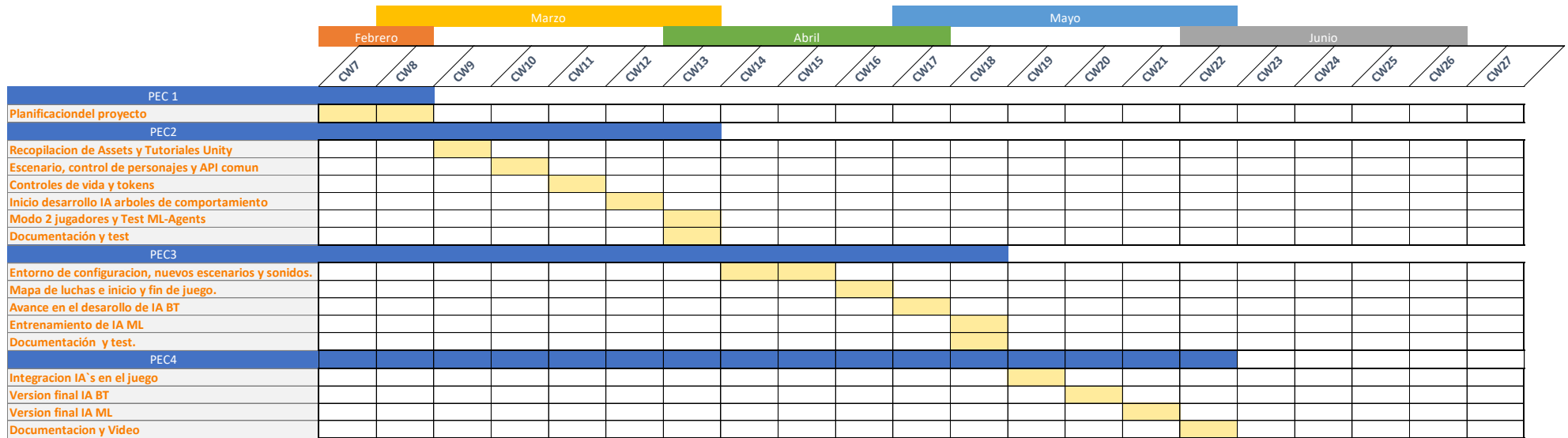


Ilustración 6. Planificación del proyecto.

2.Estado del arte

2.1. Antecedentes y evolución de los juegos de lucha

Como hemos mencionado anteriormente, uno de los primeros juegos que se comercializó fue el Pong. Este juego pretendía simular la realidad de los deportes en la pantalla de tu televisión. Con esta primera aproximación, no era raro pensar que lo siguientes juegos también seguirían la estela de este, de ahí que aparecieran los juegos de lucha.³

Los primeros de la lista que podemos mencionar fueron Karate Champ y Yie Ar Kung-Fu. Estos juegos sentaron las bases de los juegos de lucha y propusieron soluciones que todavía hoy se siguen teniendo en cuenta, como son la barra de vida o poder elegir entre diferentes personajes.



Ilustración 7. Karate Champ. ©Technos Japan

Pero el verdadero “bombazo” en este tipo de juego lo podemos ver en Street Fighter en su segunda versión.

“Street Fighter fue el verdadero padrino de los juegos de lucha”.⁴

Esta introdujo el concepto de combos, en el cual el jugador podía encadenar una serie de golpes si el primero de ellos conectaba con el enemigo.

³ **Barriga, N.,Gajardo I., Besoain, F.** (Noviembre,2019) Introduction to Behavior Algorithms for Fighting Games[Artículo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/339176190_Introduction_to_Behavior_Algorithms_for_Fighting_Games

⁴ **Stuart, K.** (Junio, 2019) Kapow! The history of fighting games [Artículo en línea] 80.lv [1 de Mayo de 2022] <https://www.theguardian.com/games/2019/jun/01/kapow-the-history-of-fighting-games>

Los juegos más modernos han requerido innovaciones con el fin de evitar la pérdida de cuota de mercado. Por ejemplo, generando una intrahistoria dentro del juego, o como la introducción de algunas de dudosa ética como la de “fatilities” dentro del famoso *Mortal Kombat*, donde el personaje ganador infringe una cruel destrucción a su oponente.



Ilustración 8. Mortal Kombat © Midway Games

Conforme los juegos de lucha avanzan se hace más evidente tener IA más avanzadas y más creíbles lo supone un reto que puede ser mayúsculo dependiendo de los retos que se quieran conseguir. ⁵De entre los nuevos juegos de lucha, podemos destacar el *Shadow Fight 3*, el cual introduce ciertas peculiaridades que lo hacen interesante como:

- El uso de *autoblocks*, en el que si el personaje no se mueve, el oponente no ataca.
- El ritmo lento y las animaciones suaves como componente táctico. El jugador puede tomar decisiones dependiendo del ataque con más tiempo y el ritmo más pausado de las animaciones añade una estética atractiva al juego.
- Choques realistas gracias al ritmo pausado, dejando contratacar al oponente dando un aspecto realista de la batalla.

⁵ **Dragovalovskiy, M.** (Diciembre, 2019) AI in Fighting Games: Dissecting Shadow Fight 3[Artículo en línea] 80.lv [30 de Marzo de 2022] <https://80.lv/articles/ai-in-fighting-games-dissecting-shadow-fight-3>



Ilustración 9.Shadow Fight 3 © Nekki Limited.

2.2. Métodos de creación de IA

El segundo aspecto interesante y quizás el más motivador de este trabajo es el referente a la creación de las AI que controlan al oponente. Muchos de estos juegos son usados en modo multijugador, y los jugadores que pasan muchas horas en el juego prefieren este modo, pero todavía tenemos una buena cuota de mercado en el jugador esporádico en el que el modo de juego va a ser contra la máquina.⁶

Para implementar las diferentes tipos de IA disponemos de diferentes métodos que seguidamente se enumerarán y explicaran.

Finite States Machine:

Podemos decir que esta modalidad es una de las más comunes y una de las primeras que se utilizó para programar IA de NPC⁷. Este método hace referencia a distintos estados que puede tener una máquina imaginaria con sus transiciones establecidas entre estas.

Este modelo de comportamiento tiene entradas y salidas, donde estas no solo tienen una dependencia con las entradas actuales, sino además con las anteriores. La representación

⁶ Barriga, N.,Gajardo I., Besoain, F. (Noviembre,2019) Introduction to Behavior Algorithms for Fighting Games[Artículo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/339176190_Introduction_to_Behavior_Algorithms_for_Fighting_Games

⁷ Devang Jagdale(Octubre,2021) Finite State Machine in Game Development[Artículo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/355518086_Finite_State_Machine_in_Game_Development

gráfica de la máquina se realiza mediante un diagrama de estados, aunque también es posible realizarlo con un diagrama de flujo.

Una de las decisiones que se toman para utilizar este método, es la sencillez de su lógica, pudiendo ser utilizado en equipos donde no todo el mundo es programador. Usando un estado como inicio y desde este, el juego puede elegir el camino dependiendo de los inputs recibidos. Si ninguno de estas señales de entrada disparan ninguna transición, en ese caso no se realiza ninguna acción.

Uno de los grandes problemas de este método es en proyectos de mucha dimensión, si hacemos una máquina de estados demasiado grande, esto puede agregar complejidad en el código si no se tiene un buen diseño inicial, lo que impide escalarlo si empezamos con un diseño más reducido.⁸

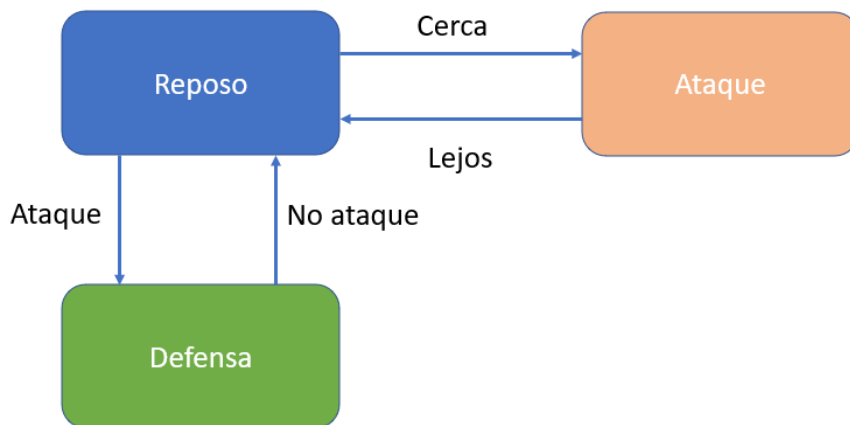


Ilustración 10. Finite States Machine

Behaviour Trees:

Otro de los patrones que más se está usando en la actualidad es el de árboles de comportamiento. En vez de tener unos estados definidos con unas transiciones entre estos, lo que se definen es un árbol de nodos, creando las ramas que determinan varios comportamientos.

Empezando desde la raíz del árbol, el próximo paso es evaluar el primer nodo, en el caso de que este no sea una hoja, que nos daría un comportamiento final. Este puede ser de dos tipos:

⁸ **Yoonas A. Sekhvat** (Enero,2017) Behavior Trees for Computer Games [Artículo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/312869797_Behavior_Trees_for_Computer_Games

Secuenciador: Donde se visitan todos los hijos del nodo. Este actúa como una puerta AND, donde todos ellos deben retornar un valor satisfactorio para que el total lo sea.

Selectores: Donde se evalúan los nodos y se va pasando en orden de uno a otro en caso de que el anterior falle. Si encontramos uno con comportamiento satisfactorio este retornará comportamiento positivo. Si todos fallan retornará fallo. Su comportamiento podríamos asemejarlo a una puerta OR.

La idea es evaluar las condiciones para poder llegar a un nodo hoja donde tenemos el comportamiento que queremos aplicar. Este se reproducirá a cada tic, hasta que la evaluación del árbol le obligue a cambiar a otro nodo hoja que hará cambiar el comportamiento del personaje.

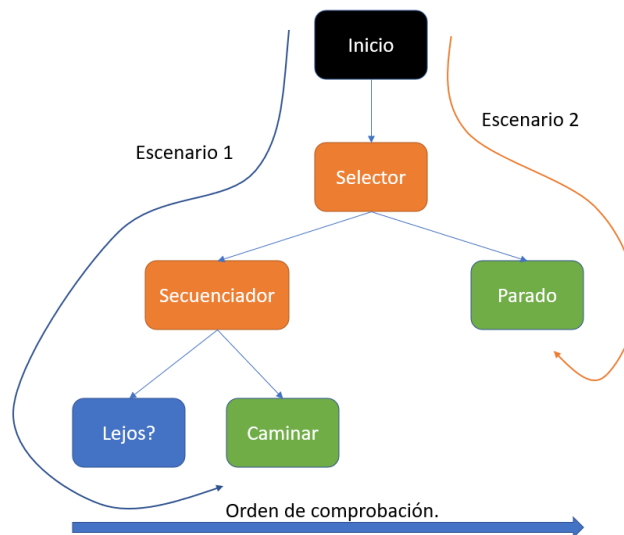


Ilustración 11. Árbol de comportamiento.

Como podemos ver en el ejemplo, en el escenario 1, evaluamos el primer selector, como el nodo lejos nos retorna positivamente, seguimos con el selector y llegamos al nodo “caminar”, que es donde residirá la instrucción de caminar. Este se seguirá ejecutando, en tanto en cuando la evaluación de “lejos” sea positiva. En el escenario 2, la evaluación de lejos retorna negativamente y el selector cambia al siguiente nodo ejecutando la instrucción de parado.

Monte Carlo Search tree:

Este árbol de búsqueda realiza un análisis de los movimientos más prometedores. Este sistema amplía el árbol de búsqueda, basándose en un muestreo aleatorio. Se realizan

diferentes juegos de manera aleatoria, de principio a fin, de manera que podamos ponderar los diferentes nodos con los datos recogidos, para en los siguientes juegos utilizar las mejores jugadas.

Una de las maneras de implementar este árbol sería por ejemplo, determinar una serie de intentos con diferentes movimientos para golpear al contrario de manera aleatoria y evaluar posteriormente cuales nos han resultado exitosos, para de esta manera repetirlos a posteriori. Podemos dividir este proceso en rondas, donde cada una de estas tiene cuatro pasos.

- **Selección:** Empezando desde la raíz, se selecciona diferentes nodos hijos hasta alcanzar la hoja objetivo.
- **Expansión:** Creamos un nodo hijos desde cualquier movimiento válido.
- **Simulación:** Realizar una reproducción aleatoria desde el nodo creado.
- **Retro propagación:** Utilizar el resultado del paso anterior para ponderar el camino desde la raíz hasta el nuevo nodo.

Uso de Redes neuronales:

La última opción que vamos a presentar no necesita ser programada, pero si que hace falta preparar el entorno para que el sistema sepa lo que debe o no debe de hacer en cada caso. Uno de estos ejemplos son las redes neuronales.

Una de las maneras de usar las redes neuronales es el aprendizaje reforzado. Recibiendo recompensas o castigos, podemos modelar la red neuronal para que pondere los caminos de esta y a base de repetición elija la mejor opción.

Los denominados Agentes(abstracción del personaje que aprende), en principio no tiene un objetivo definido, el aprendizaje es a base de prueba y error. La interacción con el entorno les hace recibir refuerzos positivos o negativos.

Las redes neuronales están formadas por nodos en diferentes capas, donde podemos distinguir:

- **La capa de entrada:** Desde este input la red neuronal recibe la información del entorno.
- **La capa o capas interiores:** En esta se realizan diferentes operaciones con los datos aportados de la capa de entrada. Con estas operaciones se genera la salida de la capa final.

- **La capa de salida:** Determina cual será la acción a realizar y conecta con las acciones que debemos dar a nuestro personaje.

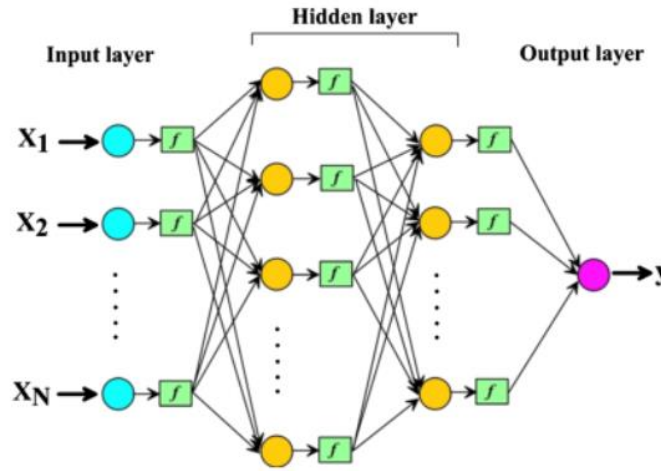


Ilustración 12. Red Neuronal⁹

El ciclo del aprendizaje reforzado, en su aplicación práctica en un juego sería la siguiente:

- **Observación:** Recoge información de su entorno para alimentar el siguiente paso. Una posibilidad sería tener al oponente cerca.
- **Decisión:** En base al paso anterior se toma una decisión.
- **Acción:** La decisión desencadena una acción. Esta podría ser, por ejemplo, atacar o defender.
- **Recompensa:** Si hemos fallado en nuestro cometido se nos penaliza, en caso contrario se nos recompensa.



Ilustración 13. Ciclo aprendizaje reforzado

⁹ Barzegar, R., Adamowski, J., & Moghaddam, A. A. (2016). Application of wavelet-artificial intelligence hybrid models for water quality prediction: a case study in Aji-Chay River, Iran. Stochastic environmental research and risk assessment, 30(7), 1797-1819.

3.Propuesta

3.1. Idea del juego

3.1.1. Descripción

Combate Singular pretende, aparte de ser un juego de combate, experimentar con dos tipos de IA. Como ya se ha comentado anteriormente, la primera, una versión programada basada en arboles de comportamiento, y la segunda, capaz de aprender y evolucionar con los combates que va realizando. Este entrenamiento será realizado previamente, dado que en la versión actual del paquete para Unity, *ML-agents*, solo permite el entrenamiento en la fase de desarrollo.

Aunque para programas IA en juegos de combate la manera más habitual ha sido desde un principio las máquinas de estado, **se ha podido comprobar que *Behavior Tree* tiende a ser una inteligencia más adaptable y cercana al comportamiento humano**¹⁰. Además, personalmente suponía un mayor reto hacer una IA programada en este sentido. Un paso más adelante hubiera sido usar MCTS, pero su procesamiento es más lento y es una opción no tan usada como los árboles de comportamiento y dada la falta de tiempo no se hubiera podido poner en marcha una opción más de IA¹¹. Para la opción de aprendizaje automático solo teníamos una opción en Unity.

El juego, en esencia, no se aparta mucho de la tradición de los juegos clásicos de lucha. En él tendremos las características clásicas como puedan ser el contador de vida o la selección de adversarios. La misión de nuestro personaje será la de acabar con todos sus enemigos para poder ganar el juego. Este será un modo de juego contra la máquina en el que podremos elegir entre la IA programada y la que aprende. Para finalizar tendremos un modo dos jugadores en el cual se tendrá en cuenta solo en combate en curso.

¹⁰ **Barriga, N.,Gajardo I., Besoain, F.** (Noviembre,2019) Introduction to Behavior Algorithms for Fighting Games[Artículo en línea]Researchgate.net[30 de Abril de 2022] https://www.researchgate.net/publication/339176190_Introduction_to_Behavior_Algorithms_for_Fighting_Games

¹¹ **Ada-Rhodes Short, Bryony L. DuPont, and Matthew I. Campbell**(2018) A Comparison of Tree Search Methods for Graph Topology Design Problems[Artículo en línea] <https://par.nsf.gov/> [30 de Marzo de 2022] <https://par.nsf.gov/servlets/purl/10077591>

3.1.2. Interacción juego-jugador

Debido a que el juego será para ordenadores personales, la interacción del personaje será por medio de del teclado, pudiendo desplazarse de izquierda a derecha, saltar hacia atrás para huir y dos tipos de ataques y una defensa. Para selección de los menús de selección de los combates y las opciones, se utilizará el ratón.

3.1.3. Plataforma destino

El juego estará desarrollado para PC. La libertad que nos otorga, dada la falta de restricciones en el diseño, nos ayudará a poner todo nuestro esfuerzo en otras características del juego. No se descarta en un futuro desarrollar para plataformas móviles, dado que su potencial es mayor y el cliente ocasional se encuentra más en este tipo de plataformas.

3.2. Conceptualización

3.2.1. Ambientación del juego

El juego se sitúa en el medievo/lugar fantástico, donde nuestro personaje debe enfrentarse a distintos contrincantes de diferentes partes del reino para poder llegar a ser el rey de este. Cada contrincante cederá su territorio si pierde en un combate singular con nuestro personaje.

Se han usado unos personajes animados con el programa *Spine*,(los cuales han sido adquiridos a terceros), dado que este tipo de animación nos permite un mayor grado de control de los movimientos y características del personaje. Para los escenarios también han sido adquiridos unos fondos de alta resolución que sitúan al personaje de manera inequívoca en distintos territorios del reino con su climatología característica.



Ilustración 14. Escenario Hielo



Ilustración 15. Escenario Fuego

3.3. Personajes

3.3.1. Jugador principal

Para el jugador principal, su estética será de lo más común, intentando crear contraste con los personajes que vayan apareciendo. Si bien es verdad que durante la consecución del juego, podrá aparecer algún cambio estético otorgado por alguna poción, la estética permanecerá igual.

3.3.2. Contrincantes

Para los contrincantes se les creará un estética característica que vaya acorde con el lugar de donde proceden. Además las características de lucha, dificultad y dimensiones del personaje cambiarán de uno a otro. Se pretende dar una cierta diversidad que mantenga al jugador enganchado al juego.

3.3.3. Acciones de los jugadores

El movimiento será lateral en el eje Y, cambiando de dirección según vayamos cambiando de derecha a izquierda. Para escapar rápidamente podremos realizar un salto hacia atrás.



Ilustración 16. Personaje caminando



Ilustración 17. Salta de defensa

Cuando el personaje sea atacado y herido, este será desplazado hacia atrás para evitar el ensañamiento del oponente



Ilustración 18. Personaje herido.

En cuanto a los ataques disponemos de dos tipos, los cuales producen mermas en la vida del oponente de maneras distintas. Si le impactamos en la cabeza, restamos más puntos que en el pecho.



Ilustración 19. Ataques del personaje.

3.3.4. Efectos especiales

Para dar más acción al juego, se desarrollarán efectos asociados a los ataques, movimientos de cámara, ralentizaciones de las acciones, sonido y bandas sonoras que aporten acción.

También se creará una presentación y explicación de la historia a principio del juego que sitúe al jugador en la dinámica del este.

3.3.5. Objetos mágicos

Se ha creado tres objetos mágicos que ayudan en el combate. El primero de ellos con forma de jamón nos recupera la vida hasta el máximo, el siguiente con forma de reloj, ralentiza los ataques del contrincante y el tercero con forma de poción nos hace crecer al doble de nuestro tamaño adquiriendo una ventaja competitiva con el oponente.



Ilustración 20. Objetos mágicos

3.4. Interfaz de usuario

3.4.1. Introducción

Para la introducción, primero mostraremos el título del juego para seguidamente, dar una breve explicación del juego, de forma que el jugador sepa cuál es la finalidad del juego y hacia donde le lleva este.



Ilustración 21. Escena de introducción.

3.4.2. Menús de usuario

Para la configuración del juego, así como para la elección de los combates se ha diseñado una interfaz de usuario que funciona por medio del ratón. En ellos primero, tenemos un menú principal para poder iniciar el juego, salir de este y elegir diferentes configuraciones del juego.



Ilustración 22. Menú principal.

Dentro de las opciones del juego podemos elegir o bien el modo partida en el que podremos elegir uno de los tipos de IA, o bien el modo multijugador o Arena Mode. También podremos

consultar los controles de los personajes, así como ajustar el nivel de volumen. Desde este mismo menú de opciones tendremos la opción de acceder a las estadísticas globales del juego.



Ilustración 23. Opciones del Juego.

Dentro ya del escenario, el jugador podrá recordar en qué modo está jugando gracias a una leyenda colocada en la parte de arriba del espacio de combate.



Ilustración 24. Modo de combate

3.4.3. Mapa de los combates

Una vez entrado dentro del mapa de los combates, dependiendo del modo elegido, tendremos o todos los escenarios bloqueados menos uno, en modo partida, o todos abiertos en modo área. Esto es así, para que en el modo partida te vaya guiando por los escenarios cada vez más dificultosos. En modo Arena no es necesario porque juegas contra otro jugador y lo que quieres elegir es el escenario.



Ilustración 25. Mapa de combates.

3.5. Comparativa entre las IA

Para poder comparar las IA, vamos a realizar una serie de partidas en los diferentes escenarios. De esta manera, podremos ver cuál es la más competitiva con respecto a un jugador humano. Los datos los iremos recopilando y después alimentarán unas gráficas que podremos acceder desde la parte de opciones del juego. Esta será la última parte del desarrollo cuando las dos IA estén implementadas satisfactoriamente.

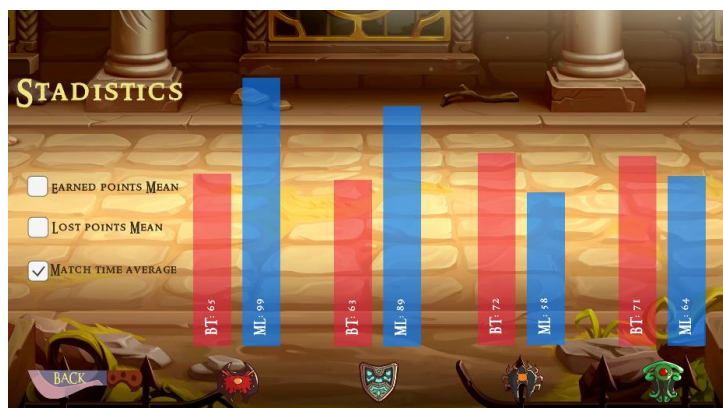


Ilustración 26. Gráficas en Unity.

Los datos que vamos a recoger serán la duración de la partida, media de puntos ganados y media de puntos perdidos. Esta será la información resumen de todas las partidas que podremos ver accediendo desde opciones del juego.

Adicionalmente a esto y para poder realizar un mayor análisis, también vamos a guardar un fichero en formato CSV con la información más en crudo y sin procesar. Con ella generaremos una hoja de Excel con distintas métricas. Toda esta información más detallada se ampliará en la sección 4.13.

Una información más, añadida a las dos que hemos mencionado es la visualización de KPI dentro de la partida. Para que sea solo una visión momentánea y no estorbe demasiado la visión del juego y la armonía del GUI, esta información solo será accesible cuando la barra espaciadora sea pulsada y se podrá quitar de la misma manera. La información que podremos ver será: los golpes que hemos recibido en el cuerpo, los que nos han dado en la cabeza, aquellos que hemos bloqueado, aparte del comportamiento de la IA en referencia a que estado está.



Ilustración 27 KPIs en el juego ML



Ilustración 28 KPIs en el Juego BT.

4. Diseño

4.1. Unity, la plataforma elegida

Anteriormente, ya hemos mencionado que se ha elegido Unity (2020.3.26f1) como herramienta de desarrollo para este juego. Enumeraremos y explicaremos los motivos por los que decantarse por esta plataforma:

- **Popularidad:** Unity es una de las plataformas de desarrollo de juegos más populares. Esto es una gran ventaja que nos pone a nuestra disposición multitud de tutoriales y feedback de la comunidad, que nos puede ayudar en el desarrollo de nuestro trabajo. Si no tenemos experiencia en el desarrollo de juegos es esencial disponer de información para aclarar dudas. Por ejemplo, en la tienda de juegos Steam, una de las más populares para PC, la plataforma más usada por los desarrolladores independientes es Unity¹².
- **Licencias disponibles de manera gratuita para proyectos personales.** Tener la posibilidad de desarrollar sin tener que desembolsar ninguna cantidad de dinero es una gran baza para poder elegir esta plataforma. Aunque no sea GNU, la posibilidad que nos brinda el software de poder usar toda su funcionalidad sin realizar ningún desembolso es altamente atractiva. Enlace a la página de Unity personal <https://store.unity.com/es/products/unity-personal> .
- **Uso de C# como lenguaje de scripting.** A nivel personal es una gran ventaja que sea este el lenguaje de facto en la plataforma Unity, dado que es un lenguaje que él que realiza este proyecto domina. Aparte de esto tenemos en conjunción con Unity, el uso de Visual Studio para todo lo relacionado con el desarrollo. Esta herramienta potentísima de Microsoft¹³, nos puede ayudar de manera muy evidente en el desarrollo del juego, aportando herramientas de *refactoring*, guiado visual, autocompletado de código..etc
- **Disponibilidad del paquete ML-agents para el desarrollo de programas con inteligencia artificial.** Como una de las cosas que queremos desarrollar en este juego son dos tipos de IA y una de ellas queremos que sea de aprendizaje automático,

¹² **Allen, Joseph** (Agosto,2021) Unity Is The Most Popular Steam Game Engine By Far [Artículo en línea] techraptor.net [30 de Abril de 2022] <https://techraptor.net/gaming/news/unity-is-most-popular-steam-game-engine-by-far>

¹³ **McKinnon,Jenni** (Febrero,2022) 10 Best IDE Software[Artículo en línea] <https://websitesetup.org/> [30 de Abril de 2022] <https://websitesetup.org/best-ide-software/>

disponer de este paquete en el entorno de desarrollo, con una integración más o menos aceptable, nos permitirá el desarrollo de esta.

4.2. Requerimientos técnicos del desarrollo

Para la realización de este proyecto se ha utilizado un ordenador portátil de la marca Xiaomi con las siguientes especificaciones:

- **Sistema operativo:** Windows 10.
- **CPU:** INTEL i5@1.8Ghz.
- **Tarjeta gráfica:** Nvidia GeForce MX150.
- **RAM:** 8GB.

4.3. Herramientas usadas

Para el desarrollo del juego, aparte de Unity, se ha usado otro tipo de herramientas tanto de desarrollo de software como de edición de archivos multimedia. Enumeraremos y describiremos estas:

- **Visual Studio 2019:** IDE necesario para editar los scripts. Es una potente herramienta que se instala casi por defecto en Unity.
- **Bitbucket, SourceTree & Git:** *SourceTree* es la herramienta visual para el mantenimiento de versiones de código de *Atlassian*. Se ha usado en conjunción con Git, la que podríamos llamar la herramienta madre, la cual en algunos casos nos ayuda con ciertas funcionalidades. De la misma marca hemos usado la plataforma BitBucket, para publicar el código y guardarlo en remoto.
- **Python & PyTorch:** Este es el lenguaje que se ha usado para el entrenamiento de la IA con Machine Learning. Has sido necesario para poder correr *PyTorch*, herramienta de aprendizaje automático.
- **CUDA:** Nos permite usar la GPU para codificar los algoritmos de la herramienta de machine learning.
- **Garage Band:** Herramienta para la edición musical en la cual se han creado algunos de los sonidos del juego.
- **OBS:** Programa de grabación de lo que sucede en el ordenador para grabar los videos de las entregas de las PEC.

También cabe resaltar algunas librerías usadas dentro de Unity como:

- **Spine runtime:** Librería de animación de personajes que usa los Assets de personajes que hemos adquirido.

- **ML-agents:** Librería de Machine Learning para Unity.
- **Barracuda:** librería redes neuronales para correr estas tanto en CPU como GPU.
- **TextMeashPro:** Librería de manejo de fuentes con muchas más opciones de las que vienen con Unity

4.4. Recursos utilizados

Para la realización de este trabajo se han adquirido, algunos de pago y otros de manera gratuita, recursos de terceros disponibles en la plataforma de *Unity Asset Store*. Esto es debido a que el objetivo de este trabajo es la programación de un juego y no su desarrollo gráfico. Tomada esta decisión, se ha intentado buscar los mejores recursos posibles para que el aspecto final fuera muy atractivo. Seguidamente enumeraremos los que se han usado:

Tabla 1 Tabla de Assets.

Nombre	Autor	Licencia	Enlace	Comprado/Gratis
Character Editor(Fantazia)	Wurrad	Unity Asset Store EULA	Fantazia	Comprado
2D Battle Backgrounds Pack	EggRollGame	Unity Asset Store EULA	2DBattle Backgrounds Pack	Comprado
Free games Items	AhNinniah	Unity Asset Store EULA	FreeGames Items	Gratis
2D items set	Homeless	Unity Asset Store EULA	2D item set	Gratis
Sonidos de MixKit.com	Mixkit.com	Mixkit License	https://mixkit.co	Gratis.

4.5. Arquitectura del juego

Presentaremos en este apartado como es la arquitectura global del juego, para de este modo apreciar las interacciones con las diferentes escenas y cuál es la línea temporal y hacia donde te lleva el juego para su finalización.

En una primera instancia tenemos la introducción que podemos saltar en el caso que pulsemos la tecla escapar, esto nos llevará a un menú donde podremos elegir las tres opciones principales del juego:

- **Nueva partida o Modo Arena:** En este ítem podremos empezar el juego en sus dos modalidades. Contra un segundo jugador, que denominamos multijugador y es el modo arena, o jugar una partida contra el ordenador, en el cual tienes que derrotar a los caballeros de los cuatro territorios para ser el nuevo rey.
- **Exit:** Esta opción nos hará salir del juego.
- **Options:** Dentro de este menú podemos distinguir entre cuatro bloques:
 - Información sobre donde están los controles de los jugadores.
 - Ajuste del volumen de la partida.
 - Selección del tipo de juego, dando opción de dos tipos de IA.
 - Paso a la escena donde se presentan las estadísticas de las partidas.

Una vez dentro del mapa de combates podemos elegir el escenario de combate si estamos en modo Arena/Multijugador o seguir el flujo de juego si hemos seleccionado alguna de las dos IA. En este último modo la dificultad irá aumentando conforme hallamos superado los distintos retos e iremos habilitando los distintos escenarios paulatinamente. Además si pulsamos sobre "volver" nos advertirá que iniciaremos los combates desde cero y perderemos la partida en curso, cosa que no pasa en modo multijugador.

Dentro de las escena estadísticas podremos ver tres tipos de ellas. Para visualizarlas solo tendremos que elegir una de las opciones disponibles para que el programa cargue los datos en la pantalla. La información que nos muestra será global de todas las partidas realizadas.

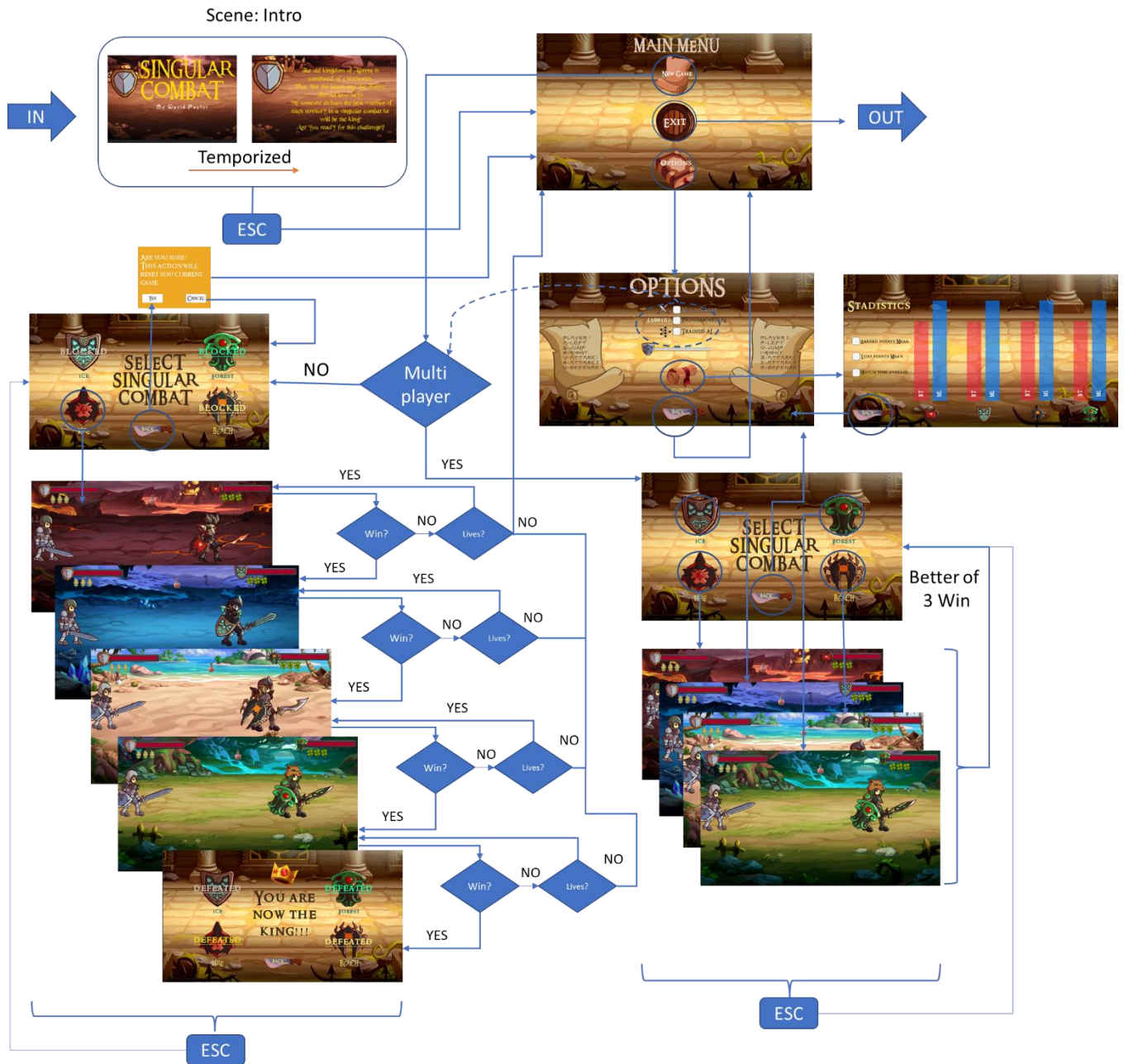


Ilustración 29. Arquitectura del juego

Dentro de cada escenario podemos observar, la barra de vida de cada uno de los contrincantes, así como el máximo de vidas que tiene cada uno en multijugador y solo el protagonista en modo un jugador.

En modo partida, si superamos todos los retos, nos anuncia que hemos ganado y nos lleva de nuevo al menú principal. Las características especiales de cada elemento serán explicadas en los apartados siguientes.

5. Implementación

5.1. La Clase Movement

Esta es la función encargada de todo el movimiento del personaje. Para animarlo no hemos usado ningún tipo de herramienta de Unity para controlar los estados y lo hemos hecho programáticamente.

Los personajes adquiridos en la plataforma de Assets de Unity no son *Sprites* con diferentes dibujos de cada *frame*, son animaciones hechas con el programa *Spine* que posteriormente con una librería de *runtime* en Unity se pueden reproducir en el juego.

Aparte de poder cambiar fácilmente el aspecto, se puede asignar a cada elemento de la animación de *Spine*, un *collider* para controlar cuando impactamos con el enemigo de una forma más precisa. Para asegurarnos que la animación se completa antes de pasar a la siguiente tenemos la función *setAnimation()* que comprueba si ya ha sido completada para dar paso a la siguiente. Esta nos ayuda con el control de estados.

Para controlar el movimiento del personaje, capturaremos las pulsaciones del teclado y asignaremos a esta una animación, dotándola de movimiento si es preciso. También podemos apreciar que tenemos la opción remote. Esto es así porque cuando queremos controlar el personaje desde una IA, debemos establecer un canal para que sin tener que pulsar, demos la opción de controlar el personaje.

Esta es la clase común para el control total del personaje y gracias a los parámetros que podemos establecer como variables accesibles al script podemos adaptarla a diferentes tipos de personajes.

5.2. Las Clases Potions y PotionsActivations

Con estas dos clases podemos controlar los premios que aparecen a mitad de combate y que ayudan a los personajes a darles cierta ventaja sobre el otro. Estos aparecen de manera aleatoria en el centro del combate en la parte de arriba y tiene un efecto de zoom in/zoom out que hemos puesto en marcha usando las funciones de redimensionado y aprovechando el *Fixedupdate()* del programa y sentencias de control.

```

void FixedUpdate()
{
    Vector3 scale = transform.localScale;
    scale.y = dim; // your new value
    scale.x = dim;

    if (up)
    {
        dim = dim + 0.2f;
        if (dim > 50)
        {
            up=false;
        }
    }

    if (!up)
    {
        dim = dim - 0.2f;
        if (dim < 30)
        {
            up=true;
        }
    }
}

```

Ilustración 30. Código de animación de objetos

Los objetos aparecen por unos segundos y se recogen saltando o impactando con la espada. Para ello hemos usado las funciones *collider*. Para la poción que aumenta la vida, accedemos a la clase **Movement** que es la que tiene el control de esta y por medio de la variable pública *life*, la restauramos al máximo. Para la que nos aumenta de tamaño, usamos el mismo método pero accediendo a su *scale* por medio otra vez de la clase **Movement** y finalmente la que ralentiza al enemigo accedemos a su *setAnimation*, que controla la velocidad de las animaciones.

En la clase **Potions** controlamos su animación y recompensa y en la **PotionsActivation** el tiempo que se muestran y la restauración del estado de los personajes después de acabar el efecto de los objetos mágicos.

5.3. Las Clases Data y DataCombat

Con estas clases, alimentaremos la escena de las estadísticas. Para recoger los datos hemos optado por dos opciones. Una de ellas es almacenar las medias en las *PlayerPreferences* de Unity. Estos son los datos que usará Unity para mostrar los resultados de las medias. Los datos serán calculados previamente antes de ser guardados.

La otra manera de almacenar los datos es en un fichero .CSV. Con él, ya externamente podremos realizar más gráficas pero esta vez con una hoja de Excel a fin de poder mostrarlo

en esta memoria. Los datos por esta vía son almacenados en crudo para poder generar más tipo de gráficas y poder comparar de una manera más precisa las dos IA.

5.4. Otras clases relacionadas con el juego

Sword. Esta función controla el *collider* de las espada y detecta donde ha impactado para restar más si lo ha hecho en la cabeza o en el pecho. De un total de 30 puntos de vida, para el impacto en el pecho se le resta 1 y para un impacto en la cabeza 2.

CameraController. Controla el movimiento de la cámara que va siguiendo al personaje principal.

Result. Controla el flujo del juego en el modo partida. Desbloquea y da por concluido los combates y te lleva a la pantalla final en caso de victoria total.

5.5. Clases relacionadas con la GUI

MapState. Controla el estado del Mapa.

Menu_Entry. Controla la carga de escenas desde el menú, así como los modos Arena y multijugador.

Intro. Controla la temporización de la introducción para cambiar los textos de esta y dar paso a la siguiente escena.

Score. Actualiza la vida restante de los personajes en la barra de vida.

IntroMusicControl. Nos ayuda a que no se corte la música durante los cambios de escena.

SiderController. Controla el estado del *slide* volumen de la escena *Options*.

ToggleState. Controla los estados del *toggle* en la escena *options*.

OnScreenStatistics. Es la clase encargada de mostrar la información de los KPIs en pantalla.

TextMode. Nos muestra el tipo IA y el modo de juego en pantalla.

5.6. Desarrollo IA con árboles de comportamiento

Seguidamente podremos ver como hemos diseñado la IA programada basada en arboles de comportamiento. Como ya se explicó en apartado anteriores, este árbol dispone de secuenciadores y selectores, los cuales se encargan del control del flujo del árbol. En las hojas podemos encontrar las validaciones y las acciones.

Los validadores devuelven el estado de FAILURE o SUCCESS. En el caso de que sea este último, pasa al nodo que implementa una acción y devuelve la acción RUNNING, en caso

contrario pasa a la siguiente rama. Esta comprobación se realiza a cada *tick* del *update()*. Empezando por la primera rama, desde la raíz va comprobando cada una de ellas hasta alcanzar un nodo hoja que tiene la acción. Si durante la subsecuentes comprobaciones se dan las condiciones, esta acción se repetirá hasta que el comprobador devuelva FAILURE.

Se ha creado una clase padre *Node*, desde la que derivan todos los nodos que implementan tanto validación como acciones. La idea es que de una forma genérica puedan heredar las mismas funciones que cada uno de ellos deriva e implementa según lo que queramos realizar con ellos. Así por ejemplo, cada uno de ellos sobrescribe la función *Evaluate()*, dotándola de las características que queremos comprobar.

Es bien conocido que podemos hacer tan potente la IA que sea invencible, por ello en las acciones y las evaluaciones, vamos a dejar que ciertos parámetros tengan un grado de aleatoriedad. La manera de hacerlas más dificultosas o menos será estrechando este parámetro. Poniendo como ejemplo la acción de defender.

```

2 referencias
public DefenceNode(Movement player, Movement ai, Action action, int difficult)
{
    this.player = player;
    this.ai = ai;
    this.action = action;
    this.difficult = difficult;
}

4 referencias
public override NodeState Evaluate()
{
    randDefence = Random.Range(1, difficult);

    if (randDefence == 1)
    {
        ai.setAction(action);
    }

    return NodeState.RUNNING;
}
    
```

Ilustración 31. Graduación de dificultad.

Podemos ver que si es igual a 1 se pondrá en marcha la acción de defender, si la variable “*difficult*” cada vez es más grande haremos que la IA sea más débil debido a que aumentamos el rango de que aleatoriamente está defendiendo. Por el contrario, cuando más cerca de 1 más difícil será derrotarla.

Todo el código que pone en marcha la IA con árboles de comportamiento está dentro del directorio *BTreeAI*.

Seguidamente comentaremos brevemente la funcionalidad de los script contenidos:

- Genéricos:
 - **Node.cs**: Clase padre sobre la que derivan cada uno de los nodos, tanto los de control de flujo como los evaluadores y las acciones. Contiene la funcionalidad genérica de paso de estados hacia los nodos superiores y de forma genérica la función *evaluate()*, que es la que los nodos que evalúan o realizan una acción sobrescriben.
 - **Selector.cs**: Comprueba cada uno de los hijo en busca de uno que devuelva un estado SUCCESS o RUNNING para pasar al siguiente nivel del árbol o poner en marcha una acción.
 - **Sequence.cs**: Realiza la secuencia de los nodos que están en su nivel pasando al siguiente si el comprobador no devuelve FAILURE.
 - **Nodestate.cs**: Enumeración de los estados.

- Específicos:
 - **IdleNode.cs**: Nodo que pone al personaje en idle
 - **AttackingNode.cs**: Comprueba la posición del arma enemiga.
 - **DefenceNode.cs**: Pone al personaje en acción de defensa.
 - **EnemyPositionNode.cs**: Comprueba la posición del enemigo.
 - **EnemyPositionWalkNode.cs**: Comprueba la posición del enemigo para caminar hacia él.
 - **NotDefendingNode.cs**: Comprueba si el enemigo está atacando y no está defendiendo.
 - **StabNode.cs**: Acción de atacar al enemigo.
 - **WalkNode.cs**: Acción de caminar.

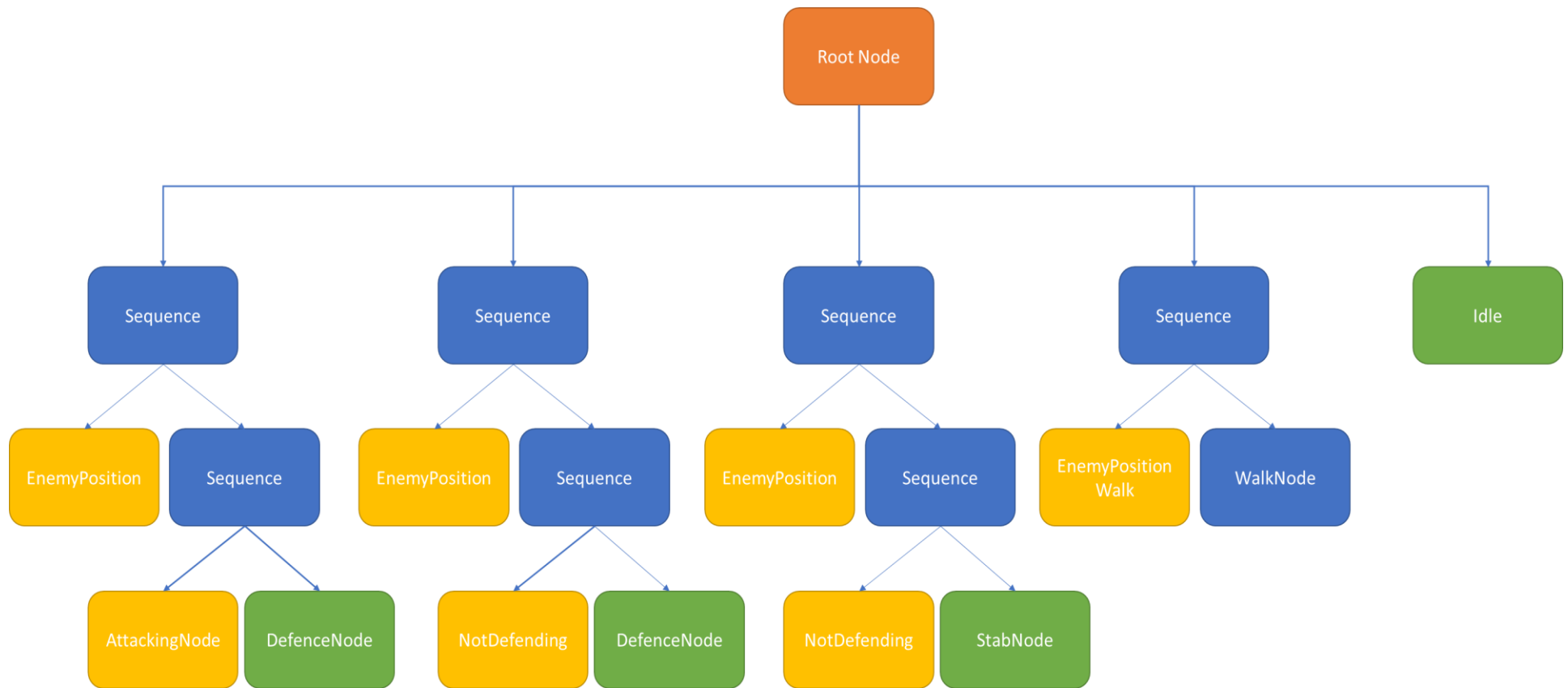


Ilustración 32 Árbol de comportamiento.

```
private Node SetupTree()
{
    Node root = new Selector(new List<Node>
    {
        new Sequence(new List<Node>
        {
            new EnemyPositionNode(playerMov,iAMov,distance,""),
            new Sequence(new List<Node>
            {
                new AttackingNode(playerMov,iAMov),
                new DefenceNode(playerMov,iAMov,Action.defence,defenceShield) //hardest 1
            },
            ),
        },
        ),
        new Sequence(new List<Node>
        {
            new EnemyPositionNode(playerMov,iAMov,distance,""),
            new Sequence(new List<Node>
            {
                new NotDefendingNode(playerMov,iAMov,heightSwordDefence), //Height sword
                new DefenceNode(playerMov,iAMov,Action.jump,defenceJump) //hardest 1
            },
            ),
        },
        ),
        new Sequence(new List<Node>
        {
            new EnemyPositionNode(playerMov,iAMov,distance,""),
            new Sequence(new List<Node>
            {
                new NotDefendingNode(playerMov,iAMov,heightSwordStab),
                new StabNode(playerMov,iAMov,attacksRange) //range of attacks minimun 4
            },
            ),
        },
        ),
        new Sequence(new List<Node>
        {
            new EnemyPositionWalkNode(playerMov,iAMov,"attack"),
            new WalkNode(playerMov,iAMov)
        },
        ),
        new IdleNode(playerMov,iAMov)
    });
};
```

Ilustración 33. Código del árbol de comportamiento.

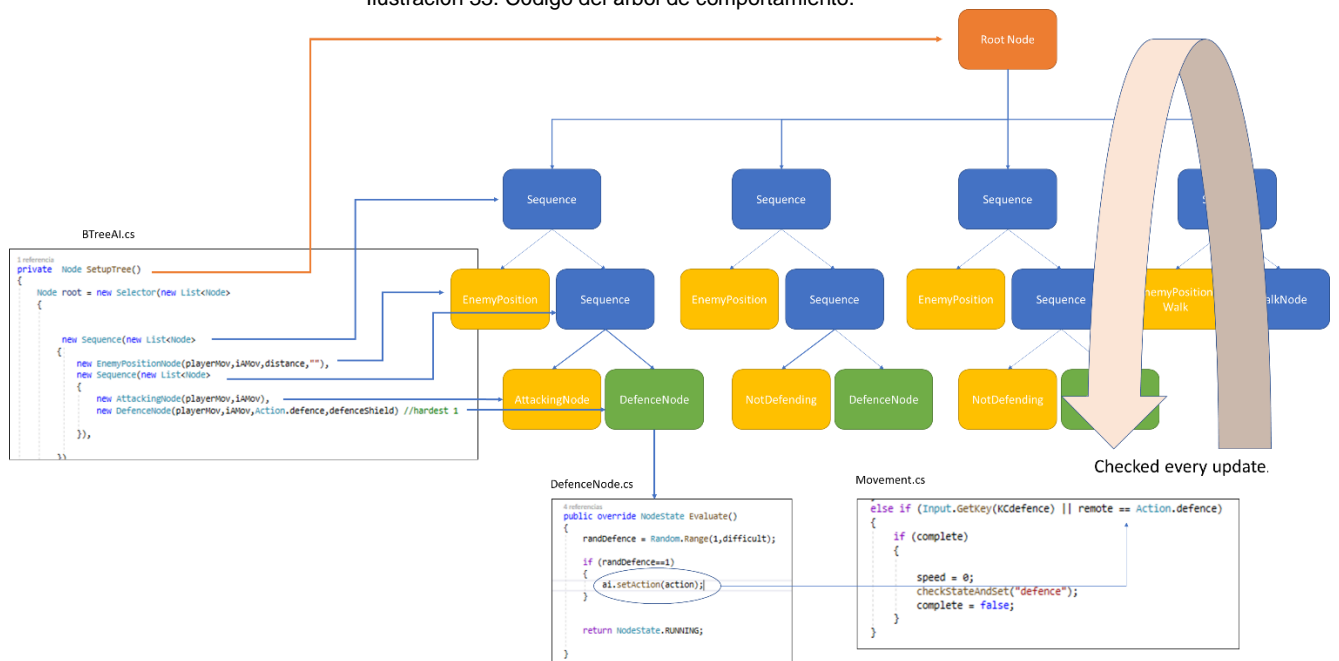


Ilustración 34. Relación Código-Árbol

Si comparamos el código con el árbol que hemos expuesto anteriormente, podemos ver la similitudes de los dos. Podemos observar los 4 niveles, las ramas y hojas de este. La función *SetupTree()*, devuelve un objeto tipo *Node*, que es evaluado en cada *Fixedupdate()* de la clase *BTreeAI.cs*

5.7. Desarrollo de IA con aprendizaje automático

La otra de las IA que hemos implementado , no la vamos a programar, vamos a hacer que aprenda de manera automática lo que tiene que hacer. Para ello será necesario usar el paquete *ML-agents* de Unity. Aparte de esto necesitaremos el intérprete de Python y las librerías de aprendizaje automático *PyTorch*. Una vez instalada y creado el entorno para que se realice el entrenamiento debemos crear una clase que derive de la clase *Agent*. En ella tendremos elementos claves como son:

- El control de los episodios. Que podemos llamar cuando empieza o acaba cada iteración del entrenamiento y dar una recompensa por finalizar.
- La observaciones con las que proveeremos a la IA, en nuestro caso vamos a comunicarle, nuestra posición, la del contrario y el estado del oponente.
- También tendremos las acciones que puede hacer, que en nuestro caso serán los movimientos del personaje, sus ataques y sus defensas.

Todas estas aparte de implementarlas en el código, también se requerirá que se lo advirtamos al paquete de *ML-agents* añadido al *GameObject* que queremos controlar.

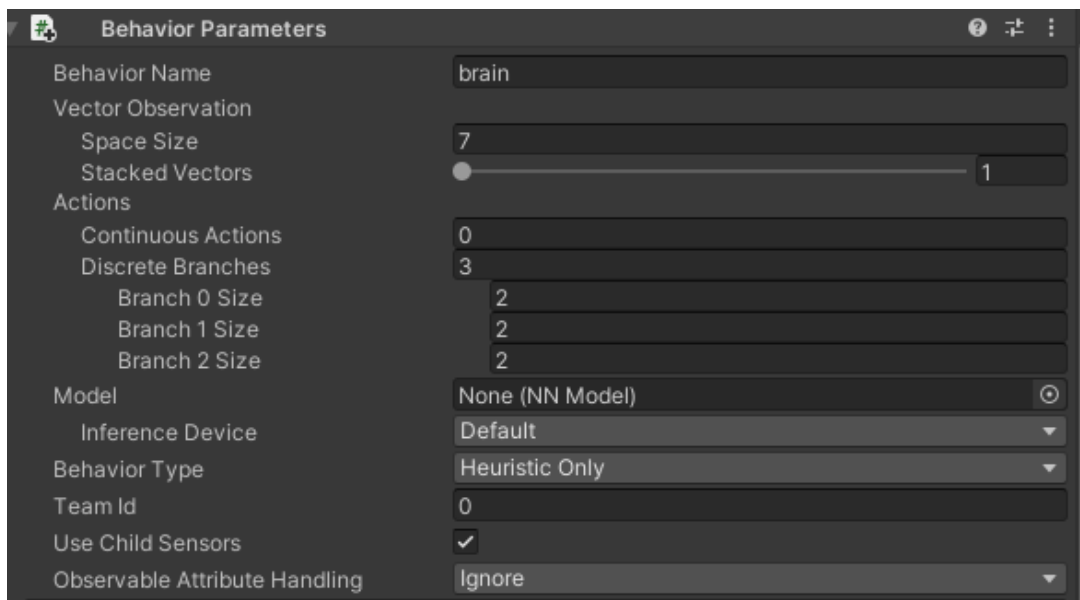


Ilustración 35. Configuración del comportamiento.

Además tenemos un comportamiento que es el Heurístico, que nos sirve para comprobar si las acciones que le pasamos a la red neuronal son correctas. Comprobando con teclas establecidas que el personaje se mueve.

Como podemos ver, tenemos 7 observaciones, 3 de cada vector de movimiento y una de la acción del enemigo. En las acciones tenemos 6, divididas en 3 ramas. En la primera rama controlamos el movimiento izquierda derecha, la segunda los dos ataques y en la tercera, las dos defensas, con el escudo y salta hacia atrás. El método para controlar el personaje es el mismo que en la anterior, tenemos la opción remote para controlarlo.

Un aspecto importante de las sesiones de entrenamiento son los episodios. Un episodio es una iteración del entrenamiento y debemos definir las características de este al principio y advertir cuando se ha acabado. Para que en el entrenamiento la Red Neuronal no se quede atrapada por siempre en algún bucle infinito durante su entrenamiento, podemos establecer un máximo de pasos para que empiece de nuevo el episodio.

```
public override void OnEpisodeBegin()
{
    if (onTraining)
    {
        transform.localPosition = new Vector2(0f, 0f);
        oponent.transform.localPosition = new Vector2(0f, 0f);
        oponent.life = 30;
        neuralEngine.life = 30;
    }
}
```

Ilustración 36 Inicio del episodio de entrenamiento.

```
if (oponent.getLife() <= 15)
{
    SetReward(1f);
    if (onTraining) EndEpisode();
}
if (neuralEngine.getLife() <= 15)
{
    SetReward(-1f);
    if (onTraining) EndEpisode();
}
```

Ilustración 37 Final del episodio de entrenamiento.

Otro punto clave será establece las recompensas y castigos con la función, *AddReward()*. En la imagen de abajo podemos ver que aplicamos un castigo si se impacta en los bordes del escenario de lucha.

```

Mensaje de Unity | 0 referencias
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.name.Equals("Layer0"))
    {
        AddReward(-0.1f);
        Debug.Log("SetReward Tope!!");
    }
}
    
```

Ilustración 38. Castigo impacto en el borde

En trozo de código que hemos adjuntado podemos ver que gratificamos si ataca cuando el enemigo está cerca y penalizamos si lo hace cuando está lejos.

```

}
else if (attack == 2)
{
    if(diff > 500) AddReward(-0.01f);
    else AddReward(0.015f);

    neuralEngine.setAction(Action.attack2);
}
    
```

Ilustración 39 Recompensa o castigo según distancia.

5.7.1. Sesión de entrenamiento

Para que el entrenamiento usamos un escenario especial que replicamos 4 veces con la intención de hacer el tiempo de entrenamiento más corto. Esta una opción que tiene ML-agents muy interesante para poder reducir la carga de trabajo.

Específicamente para los entrenamientos hemos creado una rama en GIT. Esto es debido a que es necesario cambiar demasiadas cosa en el código cada vez para pasar de entrenamiento a juego. Como se ha comentado antes, usaremos la IA programada para entrenar a la red neuronal.



Ilustración 40 Entrenamiento en 4 escenarios.

Para poner en marcha una sesión de entrenamiento debemos establecer el máximo de iteraciones que se repetirá. Este es establecido en un fichero .yaml en el que entre otras cosas podemos ajustar también la cantidad de capas.

```

behaviors:
  brain:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambda: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 200000
    time_horizon: 64
    summary_freq: 10000
    
```

Ilustración 41. configuración de la red neuronal.

Utilizaremos la IA programada para entrenar a esta, donde le aplicaremos un nivel de dificultad igual al mayor que tiene en el juego, luego dependiendo de su nivel de entrenamiento las iremos colocando para que cada vez sea más difícil derrotar al personaje.

Una vez configurado este fichero, el siguiente paso será poner el comportamiento en modo normal, en Behavior Parameters y lanzar exteriormente por consola la sesión.

En el momento que nos aparece el logo de Unity, podeos pulsar *play* y la sesión de entrenamiento comienza. Al acabar todas las iteraciones obtendremos un fichero .onnx que será la red neuronal que después podremos cargar en el módulo de comportamiento.

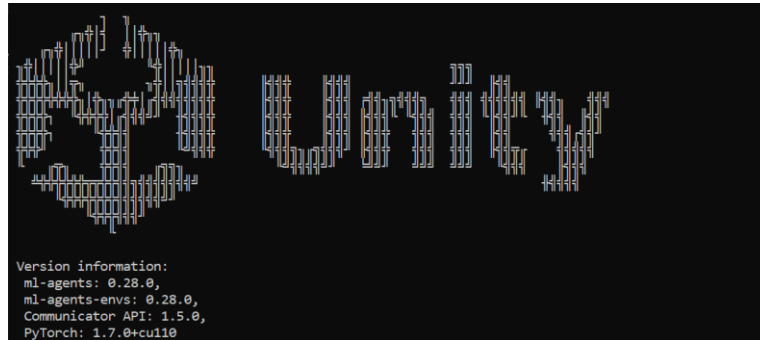


Ilustración 42. Creación de la red neuronal.

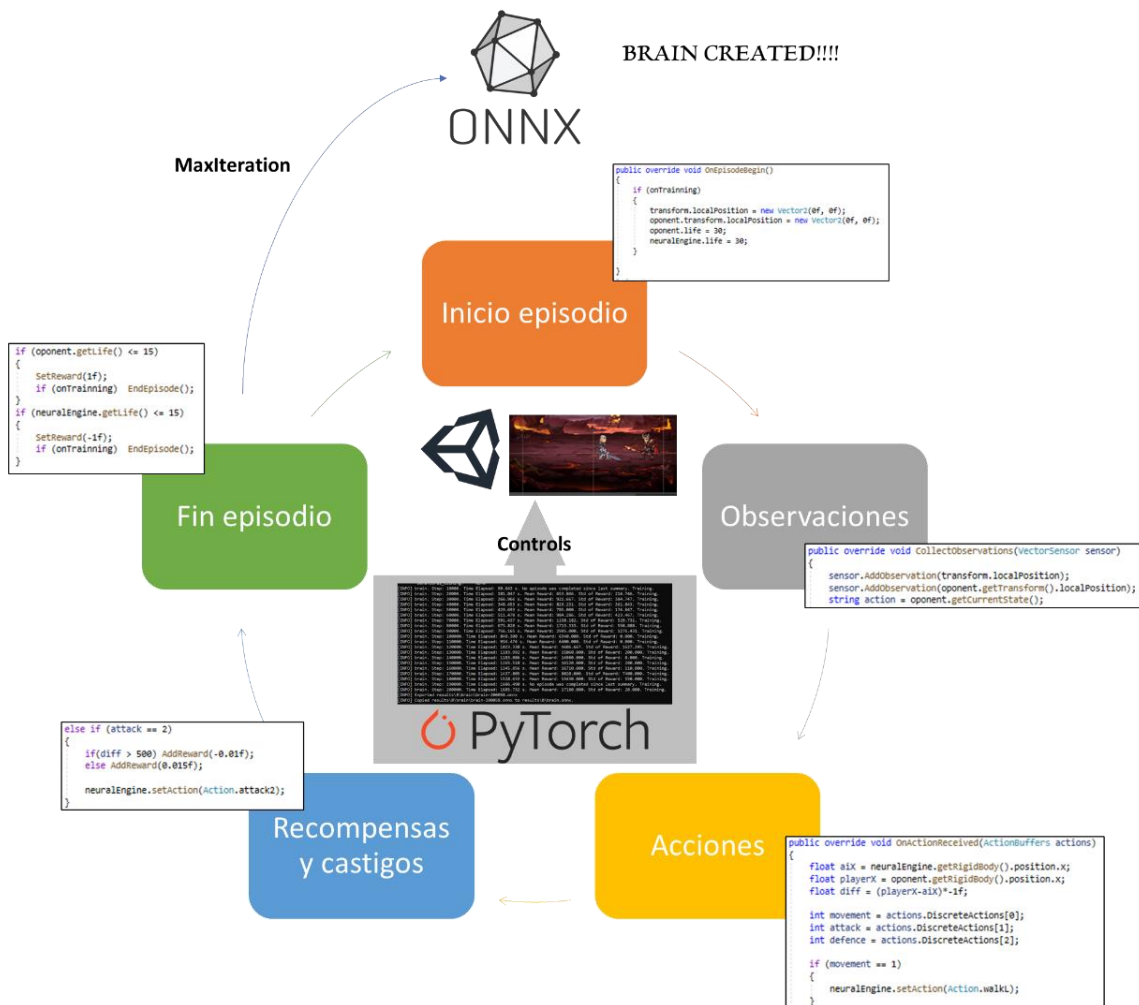


Ilustración 43. Diagrama de la sesión de entrenamiento.

Como podemos ver en el diagrama, una iteración de la sesión de entrenamiento, empieza con el inicio del episodio. Seguidamente por medio del programa externo pyTorch, se van realizando observaciones, que la red transforma en acciones. Si la acción realizada tras una observación es correcta, se gratifica, en caso contrario es penalizada. Todo esto se va repitiendo hasta llegar al máximo de iteraciones, en el que se crea la red neuronal.

Los mejores resultados han sido obtenidos desde 200.000 iteraciones en adelante. Dependiendo de las recompensas y castigos con las que hayamos configurado, los escenarios, de la dificultad del oponente, las IA se comportarán de una manera u otra y llegarán a su máximo de recompensas por episodio en un momento determinado. Finalmente hemos obtenido 4 IA con diferentes niveles de entrenamiento que van desde 200.000 y pasan por 350.000. 550000 y 700.000.

Una vez obtenidas estas IA, necesitamos integrarlas en el juego real, por ello en el *GameObject* pasaremos a modo *inference* y deshabilitaremos las opciones que controlan el flujo del juego cediendo el control a la aplicación general.

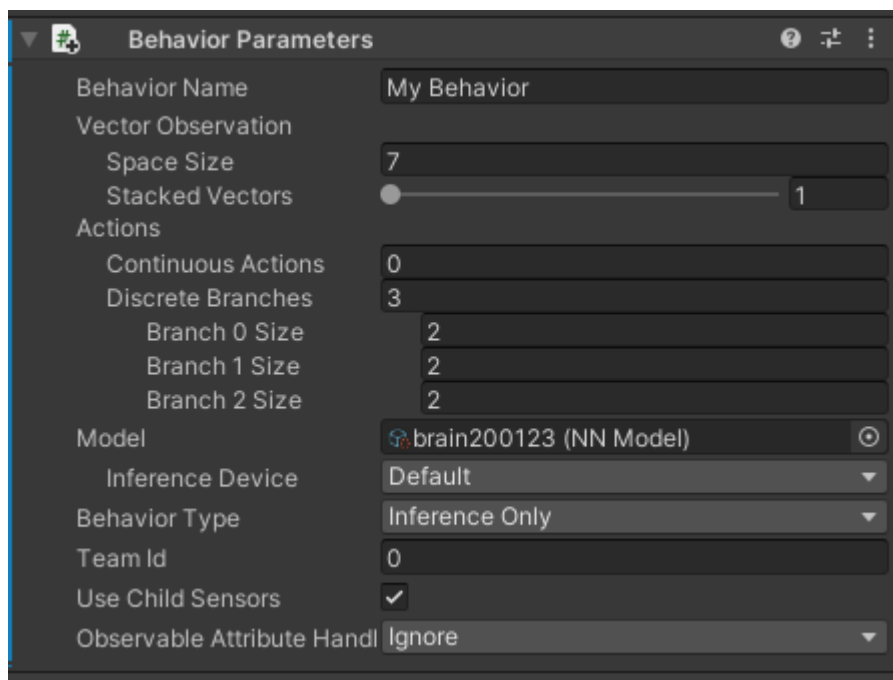


Ilustración 44 Uso de la red neuronal.

5.8. Comparativa entre las dos IA

5.8.1. Datos y graficas

Como hemos mencionado anteriormente, hemos guardado en un archivo csv los datos en crudo de los combates. Este archivo contiene la fecha del combate, los puntos ganados, los perdidos y el tiempo del combate. De aquí también podremos extrapolar las partidas ganadas. Hemos utilizado Excel para la generación de las gráficas y los resultados comparativos han sido estos. Se han realizado un total de 10 partidas por escenario y tipo de IA, lo cual arroja un total de 80 partidas para poder realizar el estudio. En la gráfica, cuando nos referimos a BT es *Behavior Tree* y ML es *Machine Learning*.

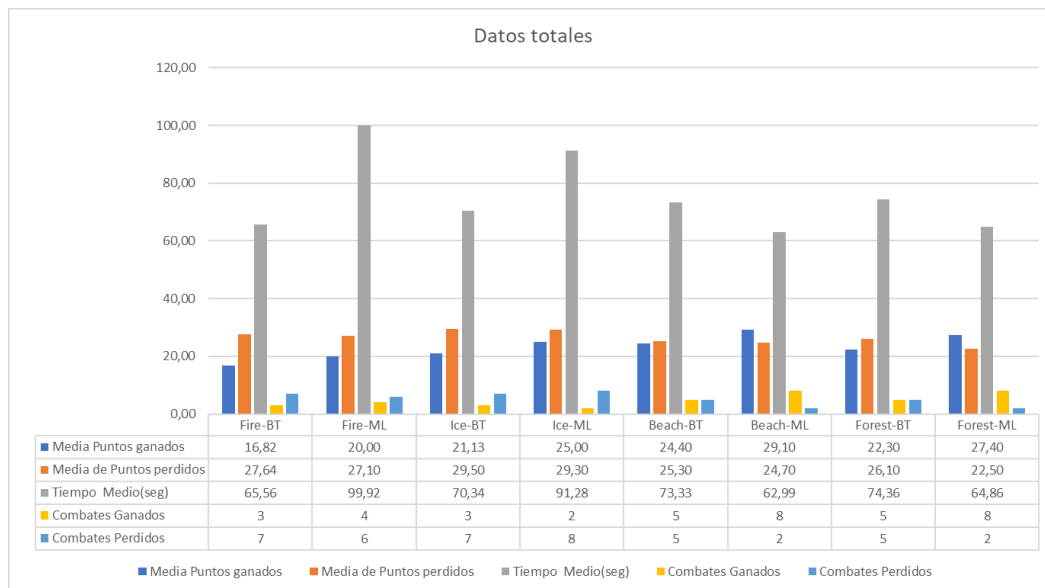


Ilustración 45. Datos totales estadística.

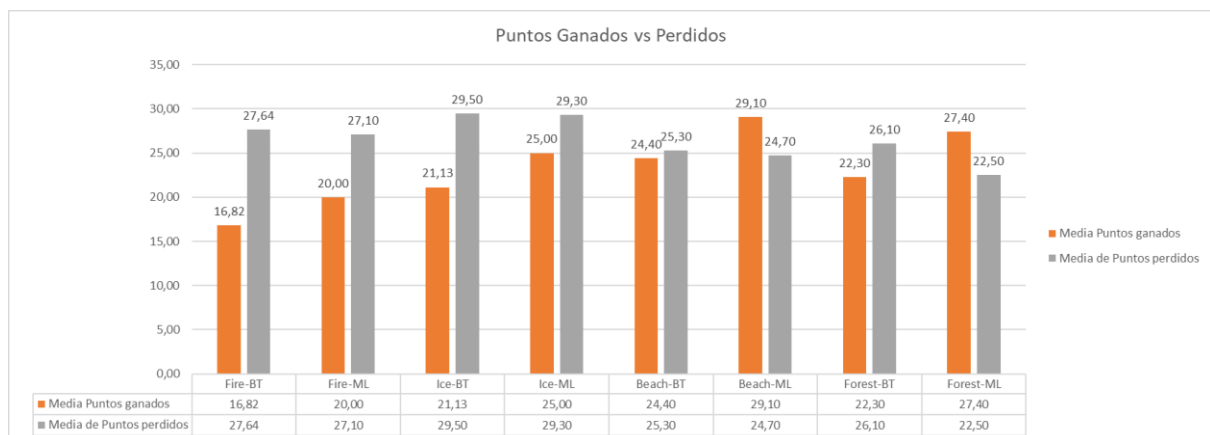


Ilustración 46 Comparativa puntos.

Podemos ver que la media de los puntos(o vida quitada al contrario), es siempre mayor el ML que en BT. Con respecto a los puntos perdidos(vida que te quitan a tí), los resultados son un poco más igualados, con una ligera ventaja para ML.

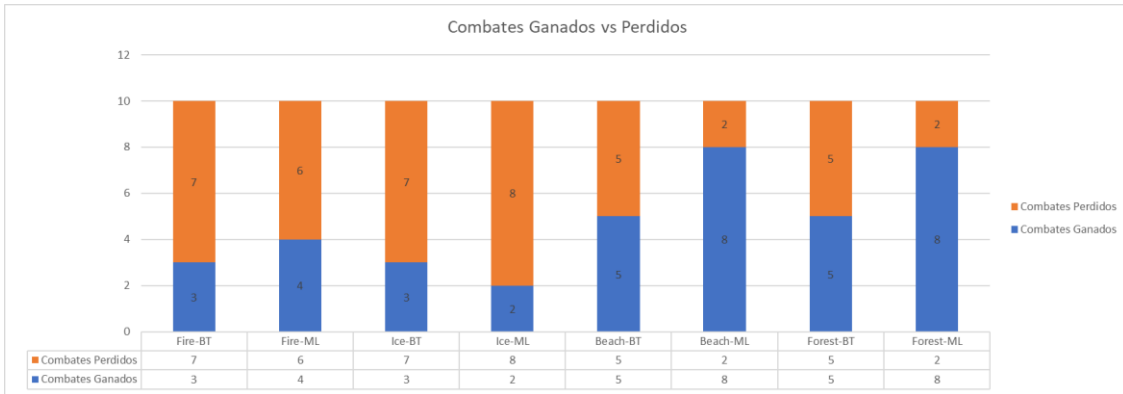


Ilustración 47 Comparativa jugadas.

La siguiente grafica nos muestra, de las 10 partidas, cuáles han sido ganadas por cada una de la IA. Podemos ver que la IA de *Machine Learning* ha sido la única que ha podido ganar más de 5.

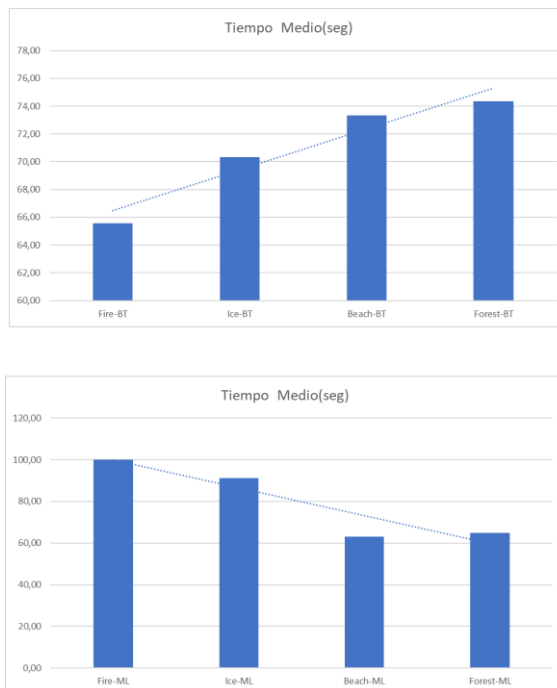


Ilustración 48 Tendencia tiempos.

En cuanto a los tiempos de duración del combate tenemos un hecho curioso. En la IA de BT, el tiempo va aumentando con la dificultad, cosa que es la manera opuesta a lo que pasa en la IA de ML. Una de las posibles explicaciones es que en la IA programada vamos añadiendo dificultad haciendo que se defienda más y necesitando más tiempo para poder derrotarla. Por el contrario la IA de ML, tiende a tomar más la iniciativa y a atacar más, lo que supone que dependiendo del entrenamiento encaja mejor los golpes en el contrario haciendo que el combate dure menos.

5.8.2. Conclusiones

De lo que se desprende de los datos obtenidos en las gráficas parece ser que la IA que ha sido entrenada, tiene mayor superioridad. Esto es más palpable sobre todo cuando más entrenada está, donde incluso después de una ronda de 10 partidas es capaz de ganar más de 5 al oponente humano.

De manera subjetiva y desde la jugabilidad, la IA que ha sido programada, el comportamiento de esta se asemeja más a lo que esperas de un oponente. Combinando los ataques y las defensas como se espera. Bastante lógico si las hemos creado nosotros.

En referencia a la que ha sido entrenada, podemos decir que su comportamiento es a veces sorprendente, porque no esperas ciertos ataques y te puede derrotar. Como también otras, donde no entiendes porque hace ciertas acciones que no llevan a ningún sitio.

Respecto a la parte de desarrollo de las IA, es necesario remarcar, que la que ha sido desarrollada por árboles de comportamiento, la escritura de código específico para que funcione es donde reside todo el trabajo, cosa que no ocurre con la de aprendizaje automático, donde la experimentación y la puesta en marcha de los escenarios de entrenamiento centran todo el trabajo. En referencia a estas últimas, para sacarles todo el rendimiento, es necesario cierta experiencia trabajando con este tipo de tecnologías, porque hasta que no controlas sus dinámicas, los resultados pueden llegar a ser un poco frustrantes.

6. Demostración

6.1. Requisitos del juego.

Combate Singular es un juego que está dirigido a la plataforma de ordenadores de escritorio. Dado que el juego no requiere de potencia gráfica, cualquier ordenador actual con un mínimo de prestaciones ha de ser capaz de ejecutarlo.

El juego se ejecutó en varios ordenadores, pero todos poseían Windows 10 y tenían una tarjeta gráfica dedicada. Para poder poner unos requisitos mínimos, una de las opciones que tenemos es visitar la página de documentación Unity en la versión en la que se ha desarrollado el juego que ha sido la 2020.3.26f1. Dentro de esta página podemos ver las recomendaciones para entorno de escritorio.

<https://docs.unity3d.com/2020.3/Documentation/Manual/system-requirements.html>

Operating system	Windows	Universal Windows Platform	macOS	Linux
Operating system version	Windows 7 (SP1+), Windows 10 and Windows 11	Windows 10, Xbox One, HoloLens	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	x86, x64 architecture with SSE2 instruction set support.	x86, x64 architecture with SSE2 instruction set support, ARM, ARM64.	x64 architecture with SSE2.	x64 architecture with SSE2 instruction set support.
Graphics API	DX10, DX11, DX12 capable.	DX10, DX11, DX12 capable GPUs.	Metal capable Intel and AMD GPUs	OpenGL 3.2+, Vulkan capable.
Additional requirements	Hardware vendor officially supported drivers. For development: IL2CPP scripting backend requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Hardware vendor officially supported drivers. For development: Windows 10 (64-bit), Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Apple officially supported drivers. For development: IL2CPP scripting backend requires Xcode. Targeting Apple Silicon with IL2CPP scripting backend requires macOS Catalina 10.15.4 and Xcode 12.2 or newer.	Gnome desktop environment running on top of X11 windowing system Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor) Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver.
For all operating systems, the Unity Player is supported on workstations, laptop or tablet form factors, running without emulation, container or compatibility layer.				

Ilustración 49 Tabla de requisitos © Unity.

6.2. Instrucciones del juego

Para poder ejecutar el juego, dentro del directorio podremos encontrar el ejecutable que lo lanza. Este será fácilmente distinguible dado que su nombre es Singular Combat y podemos apreciar que el icono es un escudo.







 MonoBleedingEdge	7/5/2022 0:08
 Singular Combat_BurstDebugInformation_DoN...	15/5/2022 16:50
 Singular Combat Data	15/5/2022 16:59
 Singular Combat	15/5/2022 16:50
 UnityCrashHandler64	6/1/2022 1:05
 UnityPlayer.dll	6/1/2022 1:05

Ilustración 50 Contenido del directorio.

Ya dentro del juego primero tendremos una introducción que nos sumerge en la historia del juego. Para no tenerla que ver todas las veces que entramos en el juego, podemos salir de ella pulsando cualquier tecla.

Desde este punto, y como hemos visto en apartados anteriores accedemos al menú principal, donde tendremos tres opciones:

- Jugar.
- Salir del juego
- Opciones.

Dentro del menú opciones es desde donde podemos configurar como queremos jugar. Primero de todo tenemos dos modos de juego: el Multijugador y el Modo partida. En el menú, la primera es la de multijugador y la dos siguientes son modo partida, pudiendo elegir el tipo de IA.



Ilustración 51 Opciones de juego.

Los controles del personaje no son seleccionables para esta versión del juego y los podemos visualizar también dentro de la escena de las opciones. Los controles son:

Tabla 2 Movimientos personajes.

Jugador 1 A-Izquierda S-Salto hacia atrás D-Derecha V-Ataque 1 B-Ataque 2 N-Defensa	Jugador 2 P-Izquierda O-Salto hacia atrás I-Derecha Z-Ataque 1 X-Ataque 2 C-Defensa
--	--

También podemos regular el volumen de la música con el deslizador con el botón en forma de escudo.



Ilustración 52 Slider de volumen.

La última opción que tenemos disponible es la de visualizar las estadísticas, pulsando sobre el pergamino accederemos a estas. En ella podemos ver las medias de los diferentes combates, de puntos ganados, perdidos o el tiempo medio de las partidas.

Cuando vayamos a jugar, dependiendo del modo que hayamos elegido, en opciones podremos ver, o Arena Mode (Multijugador) o bien New Game (Partida).



Ilustración 53 Nuevo juego.



Ilustración 54 Modo Arena.

En el modo arena podremos acceder libremente a cualquier escenario, en el modo Juego, nos irá dirigiendo por los escenarios donde cada vez será más complicado. Se nos mostrarán los que tenemos abiertos para jugar, los que hemos ganado y los que están bloqueados a la espera que se superen los anteriores.

Una vez iniciado el modo juego, si salimos del mapa, nos advertirán que tu partida será iniciada otra vez. Esto no ocurre en el modo multijugador, donde tenemos libertad de ir a cualquier escenario.

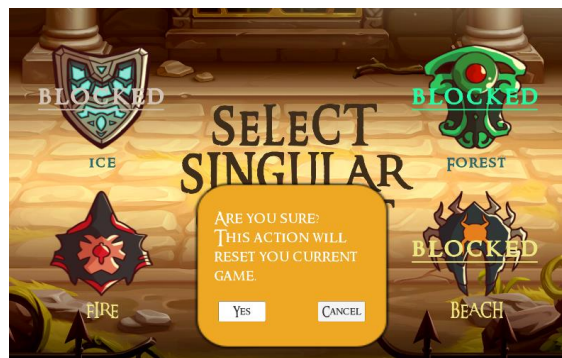


Ilustración 55 Advertencia de reinicio.

Dentro del escenario de lucha tendremos dos tipos de golpes, el ataque 1, el cual propina una estocada en el cuerpo y el ataque 2, que es un golpe con la espada desde arriba. Cada jugador tiene un total de 30 puntos de vida y cuando recibe un golpe en el cuerpo se les resta

1 punto, y cuando lo recibe en la cabeza 2. Es posible que con el ataques desde arriba impacte en la cabeza y el cuerpo restándole 3 en total. Para el personaje principal y el modo multijugador, cada uno de los personajes tiene 3 vidas para utilizar. Por cada vez que pierda un combate se le restará una. En modo partida, una vez se hayan derrotado a todo los caballeros de los distintos territorios el jugador habrá ganado el juego.

Durante la partida podremos recoger objetos mágicos que nos ayudarán a vencer al enemigo, estos solo estarán disponible en el modo partida. Los objetos y sus efectos son:

- **Alimento:** Este nos permitirá restaurar nuestra vida al máximo.



Ilustración 56. Objeto mágico: Vida.

- **Poción:** Nos permite aumentar nuestro tamaño obteniendo ventaja sobre el contrincante.



Ilustración 57 Objeto mágico: Poción.

- **Hechizo:** Este hechizo ralentiza los ataques del contrincante.



Ilustración 58 Objeto mágico: Hechizo.

Mas relacionado con la visión de lo que está sucediendo a nivel de indicadores, si pulsamos sobre la barra espaciadora podremos observar los KPI de las partidas, que nos muestran los ataques que hemos conseguido colocar, los que hemos podido bloquear y cual es comportamiento que están teniendo las IAs.

Aparte de las cosas más específicas, el juego ha sido diseñado para que la navegación sea correcta y no te lleve a ningún callejón sin salida. Teniendo como punto central el menú, se puede volver a él fácilmente con los botones de regresar.

7. Conclusiones

El desarrollo de este juego ha sido un verdadero reto para mí. En él, he tenido que poner en práctica muchas más cosas de las que hubiera pensado. Habiendo participado en otro tipo de desarrollos donde cuestiones como la estética, la jugabilidad..etc no son tomadas en cuenta, me ha supuesto un cambio de mentalidad respecto al desarrollo.

Después de haber experimentado con el desarrollo de videojuegos, me doy cuenta de la extrema dificultad que tienen y de cómo intervienen de manera decisiva en el desarrollo otras disciplinas. En una empresa dedicada a estas labores, las tareas están distribuidas por área de experiencia, pero en este trabajo todo recae sobre ti y muchas veces te encuentras dedicando mucho tiempo a temas que realmente no dominas, pero en definitiva un proyecto de fin de grado también es un reto a superar en muchos aspectos y en este, lo que indico ha sido uno de ellos.

Mención aparte merece el desarrollo de las IA. Un trabajo apasionante y que es lo que más me ha interesado. A nivel personal he logrado que interactúen con el jugador, le presenten retos y que hagan el juego, al menos a mi nivel de conocimiento, interesante.

En cuanto al uso de *ML-agents*, la herramienta para *machine learning*, he podido ver, experimentar y aplicar este tipo de tecnología de la cual tenía muchas ganas de probarla e intentar aplicarla a un juego. He de reconocer que para llegar a cotas mucho más altas es necesario dedicarle más tiempo y conocer mucho más en profundidad los entresijos y trucos para que los personajes sean mucho más increíbles y efectivos. Me lo dejo para el futuro.

En referencia a la IA con árboles de conocimiento, he podido modelarla mejor, y crearle un comportamiento más como lo haría un humano. Estoy bastante satisfecho con el resultado pero he visto que para mejorarla mucho más, se debe trabajar las heurísticas que permiten a un jugador humano adivinar los puntos débiles de la IA.

Como broche final he recogido datos para comparar los comportamientos de las 2 IA. He creído conveniente aportar este tipo de análisis para mostrar que rendimiento se puede sacar de los dos tipos de IA.

Bibliografía

Barzegar, R., Adamowski, J., & Moghaddam, A. A. (2016). Application of wavelet-artificial intelligence hybrid models for water quality prediction: a case study in Aji-Chay River, Iran. *Stochastic environmental research and risk assessment*, 30(7), 1797-1819

Barriga, N., Gajardo I., Besoain, F. (Noviembre, 2019) Introduction to Behavior Algorithms for Fighting Games [Artículo en línea] Researchgate.net [30 de Marzo de 2022] https://www.researchgate.net/publication/339176190_Introduction_to_Behavior_Algorithms_for_Fighting_Games

Pinilla, A. (Julio, 2018) . Design and development of a Roguelike game with procedurally generated dungeons and Neural Network based AI. [Artículo en línea] Repositori.uji.es [30 de Marzo de 2022] http://repositori.uji.es/xmlui/bitstream/handle/10234/179991/MEMORIA_PinillaBermejoAndoniTFG.pdf?sequence=1&isAllowed=y

Pêcheux, M. (Noviembre, 2021) How to create a simple behaviour tree in Unity/C# [Artículo en línea] Medium.com [30 de Marzo de 2022] <https://medium.com/geekculture/how-to-create-a-simple-behaviour-tree-in-unity-c-3964c84c060e>

Haytam, Z. (Enero, 2007) BEHAVIOR TREES – Introduction [Artículo en línea] blog.zhaytam.com [30 de Marzo de 2022] <https://blog.zhaytam.com/2020/01/07/behavior-trees-introduction/>

Dragovalovskiy, M. (Diciembre, 2019) AI in Fighting Games: Dissecting Shadow Fight 3 [Artículo en línea] 80.lv [30 de Marzo de 2022] <https://80.lv/articles/ai-in-fighting-games-dissecting-shadow-fight-3>

Stuart, K. (Junio, 2019) Kapow! The history of fighting games [Artículo en línea] 80.lv [1 de Mayo de 2022] <https://www.theguardian.com/games/2019/jun/01/kapow-the-history-of-fighting-games>

Midokura.com (2019)Artificial Intelligence in video games [Articulo en línea] midokura.com [20 de Mayo de 2022] <https://www.midokura.com/artificial-intelligence-in-video-games/>

Golinuxcloud.com (2020) Machine Learning in Video Games [Articulo en línea] midokura.com [20 de Mayo de 2022] <https://www.golinuxcloud.com/machine-learning-in-video-games/#one>

Devang Jagdale(Octubre,2021) Finite State Machine in Game Development[Articulo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/355518086_Finite_State_Machine_in_Game_Development

Yoones A. Sekhavat (Enero,2017) Behavior Trees for Computer Games [Articulo en línea]Researchgate.net[30 de Marzo de 2022] https://www.researchgate.net/publication/312869797_Behavior_Trees_for_Computer_Games

Barzegar, R., Adamowski, J., & Moghaddam, A. A. (2016). Application of wavelet-artificial intelligence hybrid models for water quality prediction: a case study in Aji-Chay River, Iran. Stochastic environmental research and risk assessment, 30(7), 1797-1819.

Allen, Joseph (Agosto,2021) Unity Is The Most Popular Steam Game Engine By Far [Articulo en línea] techraptor.net [30 de Abril de 2022] <https://techraptor.net/gaming/news/unity-is-most-popular-steam-game-engine-by-far>

McKinnon,Jenni (Febrero,2022) 10 Best IDE Software[Articulo en línea] <https://websitesetup.org/> [30 de Abril de 2022] <https://websitesetup.org/best-ide-software/>

Anexos

Anexo A: Repositorio BitBucket

<https://bitbucket.org/filassa/singularcombat/src/main/>

Anexo B: Proyecto compilado

<https://bitbucket.org/filassa/singularcombat/downloads/>