UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

APRENDIZAJE POR REFUERZO Y GENERACIÓN PROCEDURAL EVOLUTIVA PARA UN VIDEOJUEGO DE ESTRATEGIA

Carlos Amat Bujaldon

Trabajo de Final de grado

Director: Javier Béjar Alonso
Tutor de GEP: Joan Sardà

Facultad de Informática de Barcelona
Grado en Ingeniería Informática
Especialidad de Computación

Resumen

La Inteligencia Artificial y los videojuegos van de la mano, mientras que la IA puede utilizar los videojuegos como un terreno de entrenamiento, los videojuegos pueden aprovechar múltiples técnicas de IA para crear comportamientos e incluso contenido. En este proyecto se intenta desarrollar un Agente Inteligente, utilizando las técnicas del aprendizaje por Refuerzo para que sea capaz de jugar diversos escenarios en el videojuego The Battle for Wesnoth, además, en este proyecto también se intenta desarrollar escenarios jugables para el mismo videojuego mediante generación procedural, utilizando técnicas evolutivas.

Resum

La Intel·ligència Artificial i els videojocs van de la mà, mentre que la IA pot utilitzar els videojocs com un terreny d'entrenament, els videojocs poden aprofitar múltiples tècniques de IA per a crear comportaments i fins i tot contingut. En aquest projecte s'intenta desenvolupar un Agent Intel·ligent, fent servir les tècniques de l'aprenentatge per Reforç per tal que sigui capaç de jugar diversos escenaris en el videojoc The Battle for Wesnoth, a més, en aquest projecte també s'intenta desenvolupar escenaris jugables per al mateix videojoc mitjançant generació procedural, utilitzant tècniques evolutives.

Abstract

Artificial intelligence and video games go hand in hand, while AI can use video games as a training ground, video games can use multiple IA techniques to create behaviors and even content. This project attempts to develop an Intelligent Agent, using the techniques of Reinforcement Learning to be able to play several scenarios in the game The Battle for Wesnoth, and this project also attempts to develop playable scenarios for the same video game by procedural generation, using evolutionary techniques.

1. Contexto

Introducción

Los videojuegos han tenido una estrecha vinculación con la Inteligencia Artificial (IA) desde sus inicios, el algoritmo MiniMax es un ejemplo de las primeras iteraciones en esta relación, con sus usos en las primeras versiones de Ajedrez para ordenadores sentaría las bases de la inclusión de la IA en los videojuegos, desde el lanzamiento de Pong, pasando por Space Invaders y llegando a Pac-Man primer juego en implementar un algoritmo de Path-Finding [1].

La IA no siempre tiene en el desarrollo de un videojuego la misma relevancia, dependiendo del género y su aportación durante el desarrollo puede hacer variar su impacto, es por eso por lo que para este Proyecto se ha elegido el género de la estrategia que es sin duda donde tiene mayor relevancia. Es importante añadir que la IA no solo aparece en forma de Agentes Inteligentes que interactúan con el entorno dentro de los videojuegos (conocidos como NPCs), sino que también pueden intervenir durante el desarrollo del videojuego [2], con algoritmos capaces de crear múltiples tipos de contenido (Generación Procedural) como Escenarios, Objetos, música, diálogos, etc.

Por otra parte, no solo los videojuegos se benefician de algoritmos de IA, si no que para la investigación y desarrollo de algoritmos de IA orientados a otros campos, los videojuegos sirven como excelentes terrenos de prueba donde entrenar Agentes Inteligentes, como por ejemplo, un entrenamiento de un algoritmo con técnicas de Aprendizaje Automático como es el caso de la primera parte de este trabajo, donde se definirá y desarrollará un Agente Inteligente capaz de aprender a jugar a un videojuego de Estrategia por turnos, utilizando Aprendizaje por Refuerzo.

1.1. Estado del arte

Inteligencia artificial aplicada en videojuegos

La inteligencia Artificial, como se ha comentado en la introducción a este proyecto, está presente en múltiples niveles en el desarrollo de los videojuegos, tanto al utilizar algoritmos de IA en la creación de contenido de manera dinámica, como a la hora de crear comportamientos que interactúen con los jugadores, a día de hoy, el Aprendizaje Automático no es tan prevalente en el desarrollo de videojuegos, principalmente, debido a su complejidad y coste, esto hace que las desarrolladoras de videojuegos concluyan que por ahora suponen un desafío y un coste que puede no ser acorde a sus beneficios, al menos, para la mayoría de los géneros

jugables. Es por eso, que para conseguir comportamientos inteligentes se tiende a trabajar con otros algoritmos de IA que no necesitan ser entrenados, estos algoritmos más simples, pueden llegar a cumplir su cometido ya que se les suele permitir hacer “trampas”, otorgando a la IA información privilegiada del entorno que un jugador no podría tener en la misma situación. De esta forma, controlando esa información, se puede alterar la dificultad a la que se enfrenta el jugador sin necesidad de entrar a algoritmos de Aprendizaje Automático (ahorrándote desarrollo y entrenamiento).

Esto no quiere decir que no se hayan lanzado videojuegos donde el Aprendizaje Automático sea una funcionalidad vital del videojuego, uno de los principales ejemplos es Black & White donde se implementa Aprendizaje por Refuerzo sobre una criatura, en este caso, la criatura tomará acciones y el jugador le dará el feedback de que tan bueno o malo ha sido su decisión, de esta forma la criatura se adaptara tratando de optimizar su recompensa acumulada.

El aprendizaje automático es posible que aún no tenga la relevancia que podría tener como una funcionalidad dentro de los videojuegos, pero sí que está teniendo mucha relevancia lo contrario, los videojuegos usados para entrenar Agentes Inteligentes, el último y más potente hasta la fecha DeepMind de Google que ha demostrado que en pocas horas de entrenamiento es capaz de superar a los mejores jugadores humanos [3], en juegos clásicos como Go o el Ajedrez, o incluso llegando hasta videojuegos más modernos, con reglas mucho más complejas como Starcraft II o Dota 2. De hecho, en la actualidad se celebran múltiples torneos donde diferentes IAs se enfrentan por ver quien consigue los mejores resultados.

Generación Procedural

La generación procedural no es algo reciente, juegos como Rogue lanzado en 1980 ya la implementaba para la creación de mazmorras. En la figura 1 se puede observar un mapa generado siguiendo una de las primeras reglas de la primera versión procedural, con una distribución de 3x3 salas.

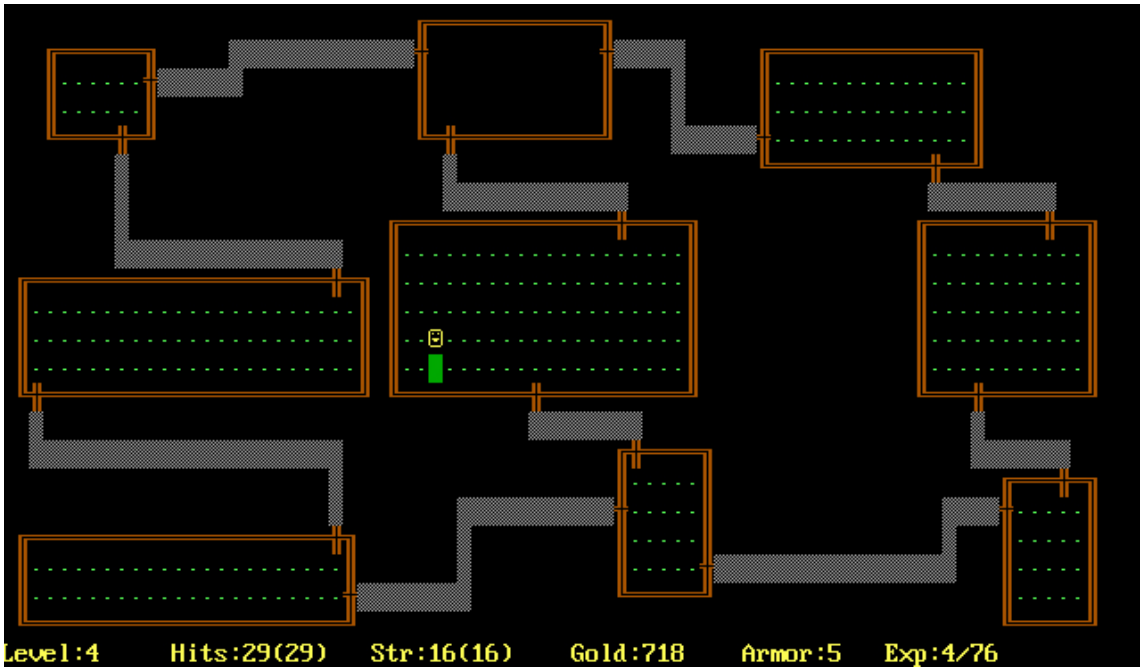


Figura 1. Mapa generado del videojuego Rogue primera versión ASCII siguiendo la generación de salas 3x3

Fuente: https://gamedevelopment.tutsplus.com/

La generación procedural en sus inicios fue popular entre los desarrolladores por varios motivos, durante las décadas 80 y 90, sin lugar a dudas la principal ventaja era la eficiencia en espacio, tener mapas prediseñados con todos sus sprites era costoso en almacenamiento y en esa época, el almacenamiento era uno de los principales limitadores a la hora de desarrollar videojuegos, era necesario que los videojuegos pudieran guardarse en disquetes para los juegos de ordenador o en cartuchos si el juego era para videoconsola, ya que con el método procedural el contenido se genera bajo demanda, no es necesario tener los mapas de antemano almacenados en disco, únicamente se debía tener los diferentes sprites que necesitara el algoritmo para generar los mapas, esto permitía poder tener mapas más grandes y tener mayor variedad de sprites [4].

Otra de las ventajas, es que también se ahorra durante el desarrollo, por ejemplo, parte del trabajo del Diseñador de niveles (aún debe desarrollarse el algoritmo, diseñar los sprites y generar unas reglas claras para que el mapa sea jugable). Con el paso de los años, el almacenamiento ha dejado de ser un problema y la generación de escenarios se ha simplificado, en gran parte, gracias a los nuevos motores gráficos como Unreal Engine 4 o Unity3D, esto, más la pérdida de popularidad del género Roguelike, hizo que se estancara la progresión de los métodos de generación procedural.

La aparición del videojuego Minecraft en 2011 cambio radicalmente esto, este videojuego y su popularidad volvió a poner en el escaparate la generación procedural y su otra gran virtud, la variedad de contenido que aporta a un juego, en el caso de Minecraft, cada partida genera un nuevo mundo de cero, completamente distinto al anterior, haciendo que los jugadores puedan volver a jugar el juego disfrutando nuevas experiencias respecto a su anterior partida, Minecraft supuso un punto de inflexión para la generación procedural y hoy está más viva que nunca y así lo demuestran juegos como No Mans Sky.

El caso de No Mans Sky es especial, actualmente ostenta el récord de tener el más grande de toda la historia de los videojuegos, y esto es gracias ni más ni menos que a la generación procedural que implementa. En este videojuego, el jugador puede explorar libremente el espacio y cuenta con un mapa generado completamente de manera procedural, el número de planetas a explorar supera los 18 trillones (18.446.744.073.709.551.616 para ser exactos) planetas [5].

1.2. Definición de conceptos

Aprendizaje Automático

El Aprendizaje Automático es el área de la Inteligencia Artificial que estudia los algoritmos que permiten a los Agentes Inteligentes adaptarse a un entorno, este estudio se divide en tres

áreas básicas: Aprendizaje Supervisado, Aprendizaje no Supervisado y Aprendizaje por Refuerzo [6].

Aprendizaje por Refuerzo

Es una de las tres áreas básicas del Aprendizaje Automático y será la solución sobre la que se trabajará en entre trabajo para conseguir una IA capaz de superar niveles en el videojuego Battle for Wesnoth.

Esta técnica consiste en lo siguiente: dado un agente y un conjunto de estados, el agente elige una acción dentro de un conjunto de posibles acciones, estas acciones, le permiten cambiar del estado actual a un nuevo estado, con el objetivo de llegar al estado que sea considerado una solución del problema, una vez se alcanza el nuevo estado, el agente recibe información (feedback) de como de buena ha sido su elección, a partir de ese feedback el agente es capaz de interpretar el resultado para adaptar su comportamiento para elecciones futuras [7].

Hay varios conceptos relevantes que definir para entender el funcionamiento del aprendizaje por refuerzo.

ENTORNO

Es el escenario del problema y sobre el que el agente tiene que operar, cada momento en el tiempo y su situación es lo que se considerará en adelante como un estado y cada acción del agente alterará la disposición de este entorno (nuevo estado). En nuestro caso, al ser un juego que enfrenta directamente al menos a dos adversarios, las acciones del oponente también provocan cambios en el entorno, incluso factores externos a los jugadores iniciados por el sistema podrían alterar el entorno.

CONJUNTO DE ESTADOS

Un estado es una situación del entorno en un momento concreto donde todos aquellos datos relevantes para el problema a resolver se representan en variables, en el caso de un videojuego como Battle for Wesnoth tenemos variables como el número de unidades aliadas, el número de unidades enemigas, sus posiciones, sus estadísticas (Vida, Ataque, Velocidad, …), el tipo de terreno, etc…

CONJUNTO DE ACCIONES

Son todas las posibles decisiones que tiene a su disposición el Agente, un ejemplo de acción seria mover la unidad N a la posición (X, Y), esto provocaría alcanzar un nuevo estado.

FUNCIÓN DE RECOMPENSA

Cada acción del agente recibe un feedback en forma de recompensa o de penalización, de esta forma, el Agente es capaz de identificar qué decisión ha sido positiva para su posición en la partida o no, el Agente no optimizará directamente sus resultados para obtener un beneficio inmediato, sino que lo que tratará es optimizar la recompensa acumulada. En la figura 2 se puede observar el proceso de interacción entre el Agente y el Entorno.

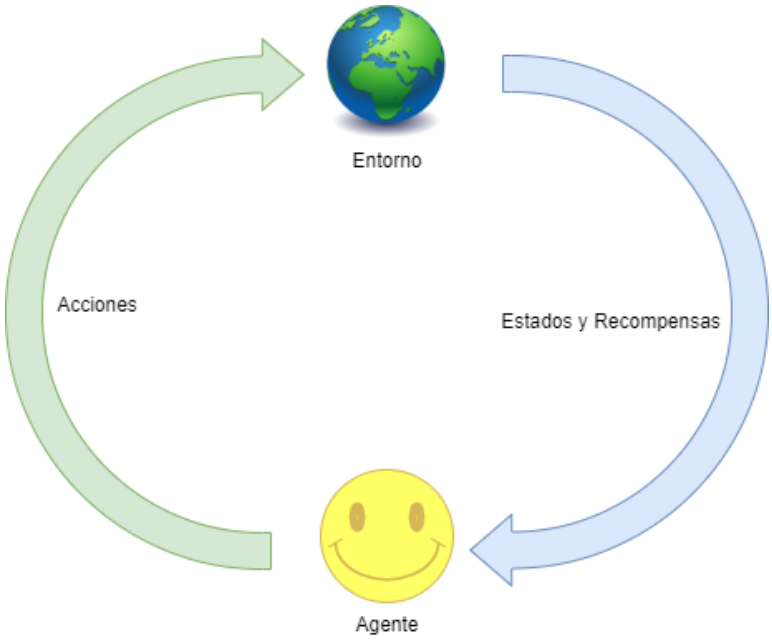


Figura 2: Procesos del aprendizaje por refuerzo.

Fuente: Elaboración Propia [8]

Generación procedural

La generación procedural consiste en la creación de contenido a través de algoritmos, en el caso de los videojuegos esto incluye escenarios, objetos 3d, personajes, animaciones, etc...

Algoritmo Evolutivo

Son algoritmos que utilizan mecanismos inspirados en la evolución biológica: Reproducción, Mutación, Recombinación y Selección.

El proceso de evolución del algoritmo comienza con una primera generación simple/aleatoria, a partir de aquí, se evalúan los individuos y se seleccionan los más aptos para reproducirlos,

descartando aquellos que no cumplen con la evaluación mínima, este último paso se ira repitiendo con cada nueva generación [9].

The Battle for Wesnoth

Es un videojuego Open Source (GPLv2) de estrategia por turnos escrito en el lenguaje C++ y complementado por el lenguaje de scripting Lua [10]. En este videojuego, el jugador controla múltiples unidades (como se observa en la figura 3) con el objetivo de abrirse paso a través de los enemigos y derrotar al comandante enemigo [11].

Figura 3. Interfaz del videojuego The Battle for Wesnoth

Fuente: https://www.wesnoth.org/

1.3. Identificación del problema

En este Trabajo de Final de Grado (TFG), se estudiará la manera de implementar el modelo del Aprendizaje por Refuerzo sobre el videojuego de estrategia por turnos The Battle for Wesnoth, con tal de crear una IA capaz de aprender a jugar y superar diferentes escenarios/niveles. En este proyecto también se implementará un algoritmo evolutivo para generar de manera procedural escenarios donde entrenar a la IA.

Por esto, el problema se divide en dos partes.

Agente inteligente usando técnicas de Aprendizaje por refuerzo

Actualmente, las desarrolladoras de videojuegos están optando por mantener sistemas más sencillos de IA al aprendizaje automático, ahorran tiempo y dinero en comparación a si usaran técnicas de aprendizaje automático, además, en muchos videojuegos consideran que no es necesario llegar a este punto, cierto es, que en algunos géneros de videojuegos utilizar aprendizaje automático puede no otorgarte ventajas respecto a otras IAs, pero en otros géneros, como los juegos de estrategia (como el tratado en este TFG) es interesante ver el rendimiento que podría llegar a alcanzar, tanto a nivel de optimización como de simulación de comportamiento humano. En The Battle for Wesnoth no existe ninguna IA implementada oficialmente usando ninguno de los métodos de Aprendizaje Automático, así que en este proyecto se intenta introducir este tipo de IA y comparar sus resultados con los otros modelos de IA ya implementados.

A otro nivel más abstracto, también como objetivo del proyecto, se analizará si la complejidad de implementar una solución de este tipo y se verán sus resultados, para así, poder comprobar si tienen razón las desarrolladoras de videojuegos sobre uso de estas técnicas.

Generación procedural de escenarios usando algoritmos evolutivos

The Battle for wesnoth cuenta con una gran cantidad de escenarios, pero todos ellos diseñados manualmente, también cuenta con un editor de mapas que facilita este trabajo. En este TFG se tratará de automatizar la generación de escenarios de manera procedural usando algoritmos evolutivos, el objetivo es que los mapas generados por este algoritmo sirvan de campo de entrenamiento para el Agente, mientras que el Agente también servirá para evaluar el grado de viabilidad y dificultad del mapa según su rendimiento en estos. La condición de victoria de un escenario suele ser derrotar el comandante enemigo.

Con esa condición definida, es importante que los mapas generados, sean justos con ambos jugadores y sus unidades, para esto, habrá que tener en cuenta muchos factores como las ventajas de algunas casillas del terreno, posibilidades similares de navegación por el escenario, etc.

1.4. Actores implicados

El sistema que se trata en este proyecto tiene como se ha comentado en el anterior apartado dos partes bien diferenciadas, en ambas partes los **desarrolladores y los jugadores** son partes implicadas comunes, aunque el impacto sobre ellos sea diferente en cada parte.

En la primera parte de Aprendizaje por Refuerzo, sirve como punto de partida para dar un comportamiento inteligente a la CPU, además, para los jugadores, supone una nueva experiencia jugable ya que se podría ofrecer una IA adaptable con una mayor naturalidad de esta IA enfrente a otras ya implementadas. Además, hay otros actores implicados, más allá del ámbito de los videojuegos, estos son los investigadores/desarrolladores de IA los cuales encuentran en los videojuegos un fantástico terreno de pruebas para sus agentes.

Respecto a la generación procedural de contenido, se trata de indagar las posibilidades que puede ofrecer a los desarrolladores de videojuegos durante el desarrollo de este a la hora de generar parte del contenido de esta forma, además, esta solución puede ofrecer contenido generado de manera prácticamente ilimitada otorgando a los jugadores experiencias diferentes en distintas iteraciones.

2. Justificación

2.1. Elección del videojuego

Se ha elegido el videojuego The Battle for Wesnoth por varias razones.

1. Es un videojuego de código abierto lo que permite su libre modificación.
2. Es un videojuego de estrategia, sin lugar a duda uno de los mejores géneros para evaluar Agentes Inteligentes.

3. Dentro de los juegos de estrategia de código abierto es del que existe una documentación más amplia debido a la longevidad del proyecto (primera versión de 2005).

4. Es un proyecto desarrollado en C++, lenguaje sobre el que ya se tiene experiencia ya que ha sido estudiado en el ámbito de la FIB.

5. No utiliza ningún Game Engine para gestionar scripts o Game Objects, el tiempo para llevar a cabo el proyecto es ajustado con lo que no necesitar un periodo de aprendizaje y adaptación a un Game Engine era una ventaja sobre otros de los videojuegos que si usaban alguno.

2.2. Elección del aprendizaje por refuerzo como solución para la IA

Se ha elegido el aprendizaje por refuerzo sobre otros tipos de aprendizaje automático como el aprendizaje supervisado por múltiples motivos [12], los más importantes son que no depende de tener un dataset con partidas para que el algoritmo estudie, cosa con la que no cuento para este proyecto, tampoco es realista generar un dataset lo suficientemente grande como para poder entrenar de manera precisa al agente en el tiempo que hay de proyecto, el otro motivo, es que para un videojuego como The Battle for Wesnoth el modelo del aprendizaje por refuerzo es más similar a como aprendería un jugador humano [13].

Actualmente no existe ninguna IA utilizando Aprendizaje Automático ni proyecto en marcha para The Battle for Wesnoth, aunque sí que ha surgido el tema varias veces en los foros comentando la viabilidad de poder utilizar Deep Learning en una nueva IA, pero nunca llegó a resultar [14], o al menos, no hay resultados publicados al respecto. Por el contrario, sí que existen proyectos similares trabajando con Q-learning [15] o SARSA [16] [17] en diferentes juegos de otros géneros.

2.3. Uso de algoritmos evolutivos en la generación procedural

La decisión de usar algoritmos evolutivos para la generación procedural llego con el último cambio de enfoque que recibió el proyecto, después de evaluar la situación en la que estaba la concepción del proyecto, pareció que podría aprovecharse el hecho de tener un Agente inteligente, ya que este podría entrenar en los mapas generados otorgando así un feedback que recibiría el algoritmo de generación procedural para mejorar la siguiente generación de escenarios. Una de las formas de conseguirlo es naturalmente con algoritmos evolutivos. Los algoritmos evolutivos siguen la estructura de la figura 4.

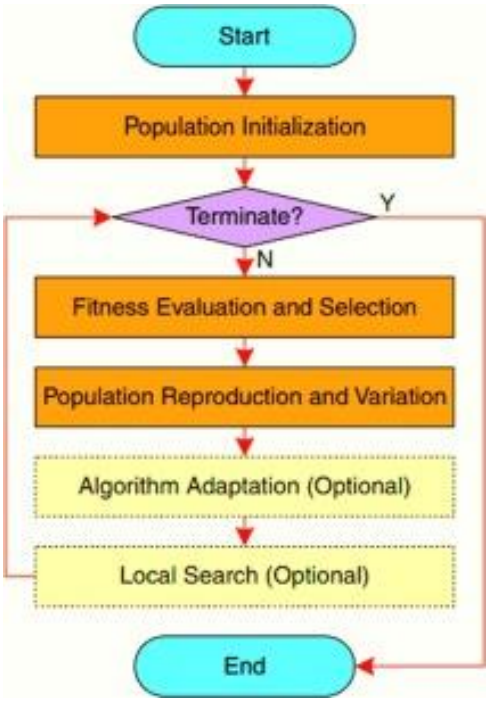


Figura 4. Visión general de la estructura de los algoritmos evolutivos

Fuente: https://www.sciencedirect.com/

3. Alcance

En cualquier proyecto, siempre es importante definir los objetivos, sus requerimientos y los posibles obstáculos para poder llevar a cabo una planificación correcta, sobre todo con un tiempo tan ajustado.

3.1. Objetivos

El objetivo general del proyecto es diseñar, implementar y entrenar un agente inteligente para el videojuego The Battle for Wesnoth utilizando técnicas del aprendizaje por refuerzo, además, diseñar, implementar un algoritmo evolutivo que se encargue de la generación procedural de contenido. Estos objetivos se dividen en los siguientes subobjetivos:

1. Estudiar y analizar los diferentes algoritmos para controlar el aprendizaje del Agente y establecer cuál es el que se adapta mejor a nuestra situación (SARSA, Q-Learning).

2. Analizar el código fuente existente del juego para poder llegar a modificarlo en este proyecto.

3. Desarrollar una inteligencia artificial para el videojuego The Battle for Wesnoth usando la metodología del aprendizaje por refuerzo

4. Analizar múltiples estrategias del juego para encontrar una función de recompensas optima.

5. Conseguir la interacción del Agente con el entorno del videojuego.

6. Entrenar el Agente en diferentes entornos de dificultad progresiva.

7. Desarrollar un algoritmo evolutivo para la generación procedural de escenarios en The Battle for Wesnoth.

8. Evaluar el rendimiento del Agente en los mapas generados y obtener feedback según como se desenvuelva el Agente en el mapa para evaluar la calidad de los mapas generados.

3.2. Requerimientos

3.2.1. REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales del sistema ya han sido definidos en los objetivos y subobjetivos.

3.2.2. REQUERIMIENTOS NO FUNCIONALES

1. Tanto el Agente Inteligente, como el algoritmo de generación procedural, tienen que ser flexibles y fáciles de adaptar a posibles cambios. En ambos casos, debería ser fácil adaptarlos a nuevos escenarios y en caso de que nuevas unidades, tipos

de terreno u otros factores in-game fueran añadidos, debería ser también relativamente rápida la adaptación del código a estos nuevos elementos.

2. Los escenarios deben ofrecer una navegabilidad mínima y una distribución lógica de los elementos, en el caso del agente, el comportamiento debe ser acorde a las reglas del juego.

3.3. Obstáculos

1. Desconocimiento de la estructura de The Battle for Wesnoth. El autor desconoce esta estructura en el momento de empezar a realizar este proyecto, es un código extenso que ha sufrido muchas ampliaciones y modificaciones desde 2005, partes del código están en desuso y la documentación de ciertas partes no está actualizada, esto y lo ajustado de los tiempos hace que sea complicado (si no imposible) analizar exhaustivamente el código.

2. Problemas con la implementación del Agente Inteligente, ya que hay que interactuar con el código ya escrito por los desarrolladores de The Battle for Wesnoth pueden surgir problemas para conseguir comunicar las acciones del agente al entorno o a la hora de recoger los datos de feedback, por ello y como se verá más en detalle en el apartado de planificación temporal se han sobreestimado la duración de las tareas de implementación.

4. Metodología

Como ya se ha comentado previamente, los tiempos para terminar el proyecto son ajustados, es por eso por lo que en este proyecto se seguirá la metodología Agile. La idea principal de esta metodología es dividir el desarrollo en pequeños sprints [18] siguiendo la estructura que se muestra en la figura 5.

El software con el que se gestionará la planificación será Microsoft Project.

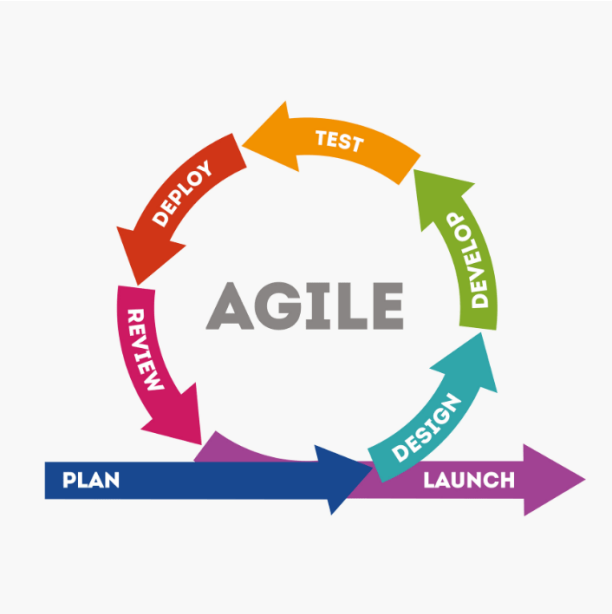


Figura 5. Metodología Agile

Fuente: https://www.cognodata.com/

La elección de esta metodología permitirá dividir el proyecto en iteraciones con un subconjunto de los subobjetivos y de las funcionalidades esperadas, además es una metodología muy flexible y que permite detectar rápidamente obstáculos y riesgos [19].

Se planea 4 iteraciones para las tareas de implementación (dos para el agente y otras dos para la generación procedural), una primera donde se desarrollan las funcionalidades más básicas y las siguiente para ampliar las funcionalidades.

4.1. Validación

Para verificar que el desarrollo sigue los tiempos marcados para cada tarea y sus respectivos objetivos/subobjetivos marcados, se planificarán reuniones con el director del proyecto después de cada sprint, para evaluar que se esté cumpliendo la planificación, en caso de que estén habiendo desvíos de la planificación se buscarán soluciones para minimizarlo.

5. Planificación Temporal

Este proyecto se ha iniciado la semana del 22 de febrero y está planteado para que esté terminado la semana del 14 de junio de 2021 como fecha máxima, se sobreestiman los tiempos de las tareas más críticas con tal de poder hacer frente a los diferentes obstáculos que

podamos encontrar durante el desarrollo. La semana siguiente seria la preparación de la defensa ante el tribunal y la otra finalmente la propia defensa.

5.1. Descripción de las tareas

Las tareas se dividen en 3 bloques principales: Las Tareas de documentación, las tareas de investigación, tareas de diseño e implementación y tareas de verificación.

5.1.1. TAREAS RELATIVAS A LA DOCUMENTACIÓN

TG1. Alcance del Proyecto

Primera fase donde se establece la base del proyecto. Se contextualiza el proyecto y se define el alcance.

TG2. Planificación temporal

Con tal de alcanzar los objetivos que se han marcado en el alcance del proyecto en el tiempo estipulado se realiza una planificación temporal, en esta planificación, se organiza todo el proyecto en tareas con sus dependencias y tiempos estimados.

TG3. Presupuesto

Se hace una estimación de los costes económicos que llevara realizar el proyecto.

TG4. Informe de sostenibilidad

Se analiza el impacto medioambiental del desarrollo del proyecto, así como el impacto durante su vida útil.

TG5. Memoria

A lo hecho en las tareas de documentación vinculadas a GEP se le suma toda la parte técnica del desarrollo del proyecto.

TG6. Preparación de la defensa

Una vez terminado el desarrollo del proyecto y completada la memoria es necesario defenderlo ante el tribunal, para esto, se requiere una preparación de cara al día de la presentación.

RN1. Reuniones

Reuniones con el director del TFG, abarcan desde ordinarias para ver como avanzan las diferentes tareas de la fase de implementación hasta la reunión de hito intermedio y la reunión final.

5.1.2. TAREAS DE INVESTIGACIÓN

TR1. Preparación del entorno de desarrollo

El videojuego esta desarrollado en C++ y Lua, es necesario preparar todo el entorno para poder compilar y ejecutar este código, además, requiere la instalación de múltiples dependencias (19 librearías de base + múltiples librerías opcionales para habilitar ciertas funcionalidades).

TR2. Recopilación de datos relevantes del videojuego

Con tal de poder crear un Agente Inteligente capaz de jugar The Battle for Wesnoth es necesario conocer la jugabilidad del juego, así como los objetivos para ganar una partida y diferentes estrategias, de esta forma, se le podrá dar el conjunto de acciones y el feedback necesario para alcanzar el objetivo del proyecto.

TR3. Análisis del código fuente del videojuego

En este proyecto se trabaja sobre un videojuego Open Source de una envergadura considerable. Esta fase se puede dividir en dos partes:

En la primera, hay que analizar principalmente como conseguir implementar nuestro agente para que interactúe con el entorno, como extraer la información relevante para el entrenamiento del agente y como conseguir los resultados de una partida.

En la segunda, hay que recuperar los diferentes modelos del juego (sprites) y organizarlos por su tipo: unidades, estructuras, terreno, etc… Una vez hecho esto, hay reunir las características que hacen que un mapa sea jugable, como diseñarlos y como introducirlos en el código del juego para poder ser ejecutados en partida.

5.1.3. TAREAS DE DISEÑO E IMPLEMENTACIÓN

TI1. Diseño del Agente Inteligente

Con la información extraída del juego y las necesidades de la IA descritas se analizan diferentes algoritmos a implementar bajo la técnica del Aprendizaje por refuerzo. Se describen sus funcionalidades y como se recibe y envía la información.

TI2. Diseño del algoritmo de generación procedural

Con la información extraída del juego, se describen las funcionalidades del generador de escenarios, se definen unas prioridades durante el diseño para mejorar la calidad de los mapas según su navegabilidad, dificultad, tamaño, etc…

TI3. Implementación del Agente Inteligente

Se desarrolla el Agente Inteligente según se ha descrito en la fase de diseño.

TI4. Implementación del algoritmo de generación procedural

Se desarrolla el Algoritmo de generación procedural según se ha descrito en la fase de diseño.

TI5. Entrenamiento del Agente Inteligente

Esta fase es consecuencia directa de la fase de implementación del Agente, ya que si se hacen cambios en la implementación que afecten al comportamiento se tiene que volver a entrenar al agente.

TI6. Evaluación de los mapas generados (Feedback para las nuevas generaciones)

En esta fase el mapa se refinará la fase de selección dependiendo el desempeño del Agente y de otras variables como: Navegabilidad, Dificultad, Estructuras, etc…

Al igual que durante el entrenamiento del Agente, en esta fase el peso de las variables puede depender de algunos cambios en la implementación.

5.1.4. TAREAS DE VERIFICACIÓN

TP1. Pruebas del Agente

En esta fase el Agente es puesto a prueba tras el entrenamiento y se analizan los resultados del agente en este escenario.

TP2. Pruebas del algoritmo de generación procedural

Durante esta fase se comprueba tanto personalmente como a través del agente las características del mapa para analizar su viabilidad.

5.2. Dependencias

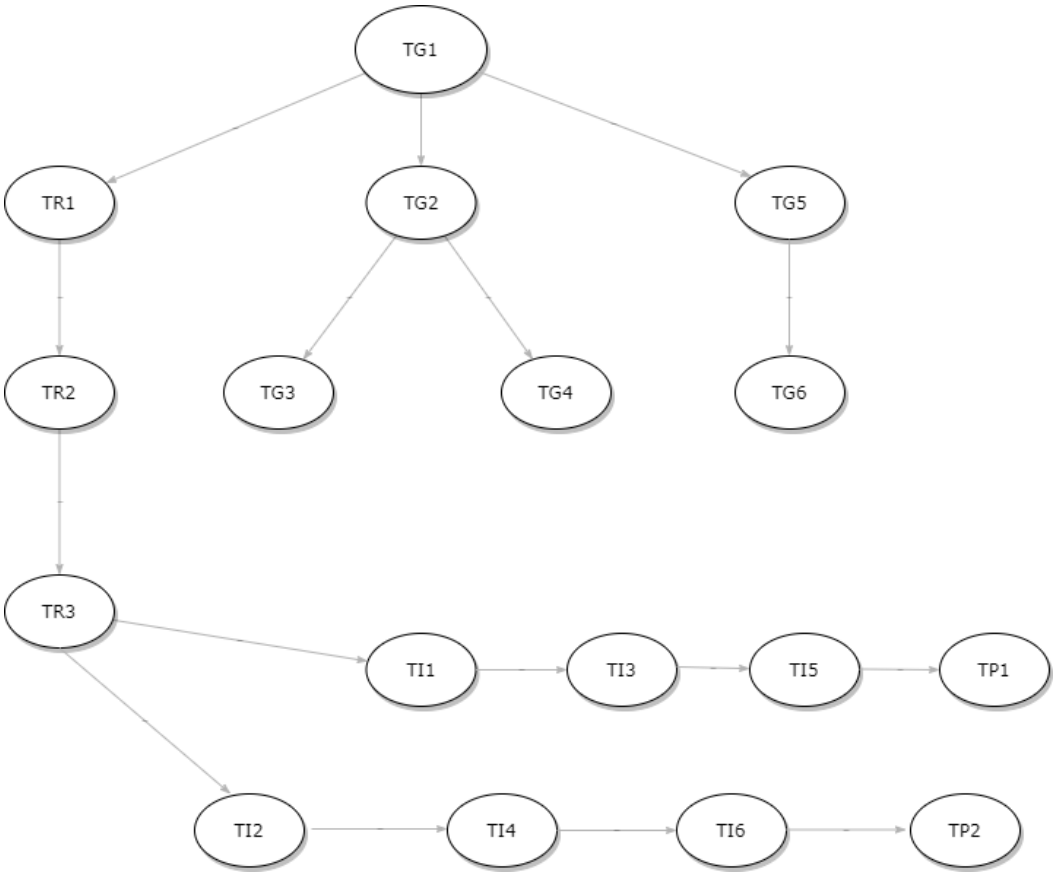


Figura X. Dependencias entre tareas (Elaboración propia)

5.3. Recursos

5.3.1. RECURSOS HUMANOS

Para este proyecto se van a necesitar cuatro perfiles distintos: Un director de proyecto (DP), un analista de sistemas (AS), un programador (P) y un tester (T). Sus roles serán explicados en más detalle en el apartado de costes de personal, las tareas asignadas a cada uno son las indicadas en la tabla 1.

5.3.2. RECURSOS MATERIALES

En este caso, hay dos tipos de recursos materiales, el Hardware y el Software. A nivel de Hardware únicamente tenemos el pc y sus periféricos, a nivel de Software tenemos los programas que se utilizaran a lo largo del proyecto, más adelante se detallaran cuáles son estos.

5.4. Estimación temporal de tareas y dependencias

Código	Tarea	Dependencias	Perfil	Tiempo (h)
TG1	Alcance del proyecto		DP	20
TG2	Planificación temporal	TG1	DP	15
TG3	Presupuesto	TG2	DP	6
TG4	Informe de sostenibilidad	TG2	DP	6
TG5	Memoria		DP	50
TG6	Preparación de la defensa	TG5	DP	10
RN1	Reuniones		DP, AS, P, T	10
TR1	Preparación del entorno de desarrollo	TG1	AS	5
TR2	Recopilación de datos relevantes del videojuego	TR1	AS	20
TR3	Análisis del código fuente del videojuego	TITR2	AS	20
TI1	Diseño del Agente Inteligente	TR3	AS	20
TI2	Diseño del algoritmo de generación procedural	TR3	AS	18
TI3	Implementación del Agente Inteligente	TI1	P	140
TI4	Implementación del algoritmo de generación procedural	TI2	P	110
TI5	Entrenamiento del Agente Inteligente	TI3	P	30 (90)
TI6	Evaluación de los mapas generados (Feedback para las nuevas generaciones)	TI4, (TI3 para ultima iteración)	P	25
TP1	Pruebas del Agente	TI5	T	20
TP2	Pruebas del algoritmo de generación procedural	TI6	T	20
			Total	545

Tabla1. Tareas con sus dependencias y tiempos estimados (El entrenamiento del Agente ocurre de manera simultánea a otras tareas) (Elaboración propia)

5.5. Gestión del riesgo

En cualquier proyecto siempre hay riesgo de toparse obstáculos durante el desarrollo. Este proyecto no es ninguna excepción y por ello hay que tratar de prever estas situaciones y preparar una contingencia con el fin de adaptarse y reestructurar aquellas tareas que se vean afectadas.

Los problemas que pueden ocurrir se clasifican en tres tipos.

Problemas de desarrollo

En esta categoría caen problemas de implementación comentados en el apartado de obstáculos y podrían provocar que se alarguen las tareas más de lo previsto. Estos contratiempos pueden ser solventados gracias a sobreestimar la duración ligeramente de las tareas que se espera sean más conflictivas, (Las fases de diseño y las de implementación de los algoritmos) de manera que se pueda reubicar ese tiempo en solventar el problema. Además, la metodología Agile hace más sencillo la detección precoz de errores y la reestructuración de tareas.

Errores en la planificación

Debido al desconocimiento en el momento de hacer la planificación inicial de la estructura ya existente del videojuego The Battle for Wesnoth, podría darse el caso de que una tarea supere su tiempo estimado, este problema, en caso de darse en tareas independientes a la implementación de los algoritmos, se puede solventar de manera natural ampliando su duración, haciendo que se solape con otras tareas que no dependan de esta. Las tareas más críticas podrían superar este problema gracias a la sobreestimación. Sobre el papel, se espera que la fase de la implementación del Agente Inteligente sea la más propensa a poder sufrir algún contratiempo, es por eso por lo que su asignación de tiempo (y su sobreestimación) es superior.

Otros problemas ajenos al proyecto

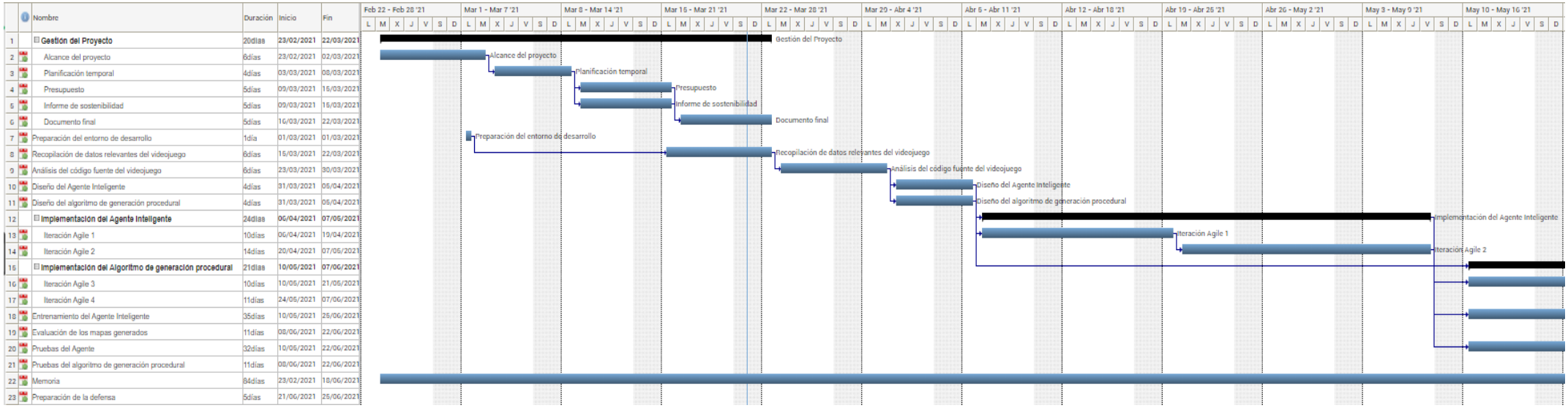
Podrían ocurrir circunstancias que debido a causas externas no se pudiera cumplir con los tiempos de planificación en un momento concreto, en este caso, habría que o bien aumentar la dedicación diaria de horas para compensar el desbarajuste, o bien, aprovechar las diferentes sobreestimaciones de algunas tareas para reajustar los tiempos. Se podrían dedicar hasta dos semanas extras (solapándose con el entrenamiento del Agente en paralelo) en caso de una desviación extrema.

5.6. Representación de la metodología Agile en el Gantt

En la Figura 5 se representa el diagrama de Gantt del proyecto, es importante añadir que durante la fase de diseño e implementación hay un total de cuatro sprints (cada sprint tiene una duración aproximada de 2/3 semanas):

1. Implementación de la base funcional del Agente Inteligente.
2. Optimización de la implementación, la función de recompensas y el comportamiento del agente.
3. Implementación de la base funcional del Algoritmo de generación procedural.
4. Optimización de la implementación y el comportamiento de la generación procedural.

FIGURA 5. DIAGRAMA DE GANTT REPRESENTANDO LA PLANIFICACIÓN TEMPORAL DEL PROYECTO (ELABORACIÓN PROPIA)



	Nombre	Duración	Inicio	Fin
1	**Gestión del Proyecto**	20días	23/02/2021	22/03/2021
2	Alcance del proyecto	6días	23/02/2021	02/03/2021
3	Planificación temporal	4días	03/03/2021	08/03/2021
4	Presupuesto	5días	09/03/2021	15/03/2021
5	Informe de sostenibilidad	5días	09/03/2021	15/03/2021
6	Documento final	5días	16/03/2021	22/03/2021
7	Preparación del entorno de desarrollo	1día	01/03/2021	01/03/2021
8	Recopilación de datos relevantes del videojuego	6días	15/03/2021	22/03/2021
9	Análisis del código fuente del videojuego	6días	23/03/2021	30/03/2021
10	Diseño del Agente Inteligente	4días	31/03/2021	05/04/2021
11	Diseño del algoritmo de generación procedural	4días	31/03/2021	05/04/2021
12	**Implementación del Agente Inteligente**	24días	06/04/2021	07/05/2021
13	Iteración Agile 1	10días	06/04/2021	19/04/2021
14	Iteración Agile 2	14días	20/04/2021	07/05/2021
15	**Implementación del Algoritmo de generación procedural**	21días	10/05/2021	07/06/2021
16	Iteración Agile 3	10días	10/05/2021	21/05/2021
17	Iteración Agile 4	11días	24/05/2021	07/06/2021
18	Entrenamiento del Agente Inteligente	35días	10/05/2021	25/06/2021
19	Evaluación de los mapas generados	11días	08/06/2021	22/06/2021
20	Pruebas del Agente	32días	10/05/2021	22/06/2021
21	Pruebas del algoritmo de generación procedural	11días	08/06/2021	22/06/2021
22	Memoria	84días	23/02/2021	18/06/2021
23	Preparación de la defensa	5días	21/06/2021	25/06/2021

6. Gestión Económica

En este apartado se van a identificar los diferentes factores económicos que están implicados en el desarrollo del proyecto.

6.1. Identificación de costes

Se dividen en los costes de personal, los costes genéricos, la partida de imprevistos y la partida de contingencia.

6.1.1. COSTES DE PERSONAL

Para comenzar es necesario identificar los perfiles que se necesitan para desarrollar el proyecto. Para este proyecto se necesitan 4 perfiles de personal:

Director de proyecto

Se encarga de realizar las tareas de gestión, documentación y presentación del proyecto.

Analista de sistemas

Realizara la fase de análisis del videojuego, así como el posterior diseño de los algoritmos a partir de la información recopilada.

Programador

Realizara las tareas de implementación, así como el entrenamiento del agente.

Tester

Validara el correcto funcionamiento de ambos algoritmos mediante pruebas en el propio videojuego.

En la tabla 1 se representan los costes de personal del proyecto.

Perfil	Coste/año	Coste/hora	Horas	Coste
Director de Proyecto	45.923 €	19,57 €	117	2.289,69 €
Analista de Sistemas	33.832 €	15,16 €	93	1.409,88 €
Programador	27.897 €	12,50 €	315	3.937,50 €
Tester	20.347 €	10 €	50	500 €
			Total	**8.137,07 €**

Tabla1. Costes de Personal (Se incluyen las 10h de reunión a todo el personal) (Elaboración propia)

6.1.2 Asignación de tareas y costes

Código	Tarea	Director de Proyecto	Analista de Sistemas	Programador	Tester	Coste
TG1	Alcance del proyecto	20 h				391,40 €
TG2	Planificación temporal	15 h				293,55 €
TG3	Presupuesto	6 h				117,42 €
TG4	Informe de sostenibilidad	6 h				117,42 €
TG5	Memoria	50 h				978,50 €
TG6	Preparación de la defensa	10 h				195,70 €
RN1	Reuniones	10 h	10 h	10 h	10 h	572,30 €
TR1	Preparación del entorno de desarrollo		5 h			75,80 €
TR2	Recopilación de datos relevantes del videojuego		20 h			303,20 €
TR3	Análisis del código fuente del videojuego		20 h			303,20 €
TI1	Diseño del Agente Inteligente		20 h			303,20 €
TI2	Diseño del algoritmo de generación procedural		18 h			272,88 €
TI3	Implementación del Agente Inteligente			140 h		1.750 €
TI4	Implementación del algoritmo de generación procedural			110 h		1.375 €
TI5	Entrenamiento del Agente Inteligente			30 h		375 €
TI6	Evaluación de los mapas generados (Feedback para las nuevas generaciones)			25 h		312,50 €
TP1	Pruebas del Agente				20 h	200 €
TP2	Pruebas del algoritmo de generación procedural				20 h	200 €
	Sueldos					**8.137,07 €**
	Seguridad Social del personal (30%)					**2.441,10 €**
	Coste total de Personal (Sueldos + SS)	117 h	93 h	315 h	50 h	**10.578,17 €**

Tabla2. Asignación de tareas y costes (Elaboración propia)

6.1.3 Costes genéricos

Hardware

Para este proyecto solo se va a requerir a nivel de hardware con un ordenador y sus correspondientes periféricos, el videojuego sobre el que se trabaja en este proyecto no es muy demandante a nivel gráfico, pero sí que se requiere algo más a nivel de procesador y memoria.

Para nuestro caso, nos servirá un ordenador de 1500 € + 200€ de pantalla y periféricos, haciendo un coste total de 1700 €.

Software

A nivel documental se trabaja con las herramientas de Microsoft de Office 365.

Para la planificación se trabaja con Microsoft Project.

Como entorno de desarrollo se va a utilizar Visual Studio 2019 Community Version, ya que es el recomendado por el equipo de desarrollo de The Battle for Wesnoth y ya cuenta con el proyecto implementado en este.

Algunos de estos programas requieren de suscripciones a sus servicios, estas suscripciones se alargarán durante los 4 meses de proyecto que duran las tareas a nivel documental, siguiendo los costes reflejados en la tabla3.

Amortizaciones

A nivel de software, el coste total es el coste amortizado de los programas, por un lado, tenemos el software gratuito cuya amortización es igual al coste (0 €). Por otro lado, tenemos software con suscripción mensual que solo se mantendrá durante el tiempo de proyecto que se les dará uso, por lo que se amortizan al 100% de su coste de suscripción.

A nivel de hardware, asumiendo que el ordenador tiene una vida útil de 5 años y que la duración del proyecto es de 4 meses, llegamos a que su coste amortizado es de (4/60) x 1700 = **113,34 €**.

Al cabo de los 4 meses de proyecto y asumiendo una depreciación del valor del hardware de un 20% del valor inicial, el valor residual = 1700 - (340 + 113,34) = 1.246,66 € que podrían ser deducidos del presupuesto asumiendo su posterior venta/traspaso.

Producto	**Tipo de licencia**	**Coste mensual**	**Coste Total**
Amortiza. Hardware	-	28,33 €	113,32 €
Office 365	Microsoft 365 Empresa Estándar	10,50 €	42 €
Microsoft Project	Project Plan 3	25,30 €	101,20 €
Visual Studio 2019	Community	0 €	0 €
C++	-	0 €	0 €
Lua	-	0 €	0 €
Suma total		**64,13 €**	**256,52 €**

Tabla3. Costes genéricos (Elaboración propia)

Los costes de personal más los costes genéricos ascienden a <u>10.834,69 €</u>.

6.2 Contingencias

En todo proyecto es importante tener en cuenta que pueden ocurrir contratiempos durante su desarrollo, estos contratiempos, muy probablemente acaben teniendo un impacto sobre el presupuesto inicial. Para estar preparados para estos posibles contratiempos se prepara una partida de contingencia del 20% del presupuesto. En la tabla 4 aparece el importe destinado a esta partida.

Presupuesto	% de Cont.	Contingencia
8.393,59 €	20	1678,72 €

Tabla4. Partida de Contingencia (Elaboración propia)

6.3 Imprevistos

Es necesario evaluar los costes de los posibles imprevistos que puedan ocurrir, estos podrían ser: un fallo de hardware, errores de software, retraso de tareas con el consiguiente incremento de los tiempos en alguna de las diferentes fases del proyecto.

1. Fallo de hardware: Un fallo de hardware en el ordenador o sus periféricos, el peor de los casos, se necesitaría adquirir hardware de las mismas características con un coste aproximado de 1700 €. Teniendo en cuenta la duración del proyecto y el tiempo de vida útil del ordenador, se estima que las posibilidades de que esto ocurra son de alrededor de un 5%. En la tabla 5 se representa el impacto sobre el presupuesto.

Recurso	Coste	Riesgo	Partida de imprevistos
Ordenador	1700 €	5%	85 €

Tabla5. Partida de imprevistos (Elaboración propia)

2. Errores de software: En el caso de que hubiera algún fallo de alguno de los programas no tendría un impacto en el presupuesto ya que para todos existen alternativas gratuitas o al menos de menor coste que el original.

3. Retraso de tareas: Como se especifica durante el alcance y la contextualización del proyecto, las tareas criticas están sobreestimadas en tiempo, así que ya se contempla ese tiempo extra en el presupuesto inicial.

6.4. Coste total del proyecto

Concepto	**Importe**
Costes de Personal (SSS incluida)	10.578,17 €
Amortización Hardware	113,32 €
Amortización Software	143,20 €
Partida de Contingencia	1678,72 €
Partida de Imprevistos	85 €
Coste Total	**12.598,41 €**

Tabla6. Coste Total (Elaboración propia)

6.5. Control de gestión

A lo largo del desarrollo del proyecto, se ira revisando si los costes de cada tarea se ajustan a lo presupuestado, en caso de que se aprecie una desviación evidente se tratara de minimizar en la medida de lo posible.

Para controlar esta desviación se utilizará el siguiente calculo:

Desviación del coste = (Ce – Cr) * Hr

Donde:

Ce: Coste estimado

Cr: Coste real

Hr: Horas reales

Aplicando este cálculo, podremos saber que desviación hay sobre una tarea en concreto y por lo tanto reaccionar con las soluciones mencionadas previamente en la sección de contingencia o imprevistos.

En el caso de que una desviación sea favorable, los recursos podrían ser redirigidos en caso de ser necesario a otras tareas más críticas o simplemente reajustar la planificación y los presupuestos.

7. Sostenibilidad

7.1. Autoevaluación

Después de responder la encuesta de sostenibilidad, he podido ver cuál es mi conocimiento actual relativo al tema. A nivel teórico considero que tengo bastante asumido el impacto y la relevancia que pueden tener las TIC en el ámbito de la sostenibilidad, en cambio, a nivel práctico disto más de tener el nivel óptimo ya que no lo he trabajado en ningún proyecto que

tuviera un impacto considerable en este campo y que, por lo tanto, hubiera que trabajar en un estudio de la sostenibilidad del proyecto. En cuanto a la concienciación, sí que considero que entiendo el impacto que tienen nuestras acciones, incluidas las de nivel personal sobre el medio ambiente.

7.2. Dimensión económica

En este apartado la estimación en general es realista, aunque es cierto que se han sobreestimado las tareas de implementación en base a la previsión de que puedan surgir imprevistos durante estas fases, y al ser las de mayor duración junto al ser las más críticas es importante que se cumpla con los tiempos.

Aunque hoy en día existen muchos trabajos sobre videojuegos de aprendizaje por refuerzo no existe ninguno sobre The Battle for Wesnoth, si existen algunas tentativas, pero sin resultados, al menos que se hayan hecho públicos.

El trabajo está centrado en la investigación y el aprendizaje así que no busca competir con ninguna propuesta actual a nivel económico.

7.3. Dimensión social

A nivel personal, siempre he sentido un gran interés por la interacción que existe entre la Inteligencia Artificial y los videojuegos, es por eso, por lo que en este proyecto se intenta profundizar más en una de las áreas del aprendizaje automático como es el Aprendizaje por refuerzo, adquiriendo así un mayor conocimiento de esta área.

En lo que respecta al beneficio de la sociedad, este proyecto podría tener efectos positivos en la comunidad de The Battle for Wesnoth, tanto ayudando a los creadores de contenido generando escenarios de manera procedural, como a los jugadores, aportando más contenido jugable. Fuera de esta comunidad, el proyecto en si no aporta novedades, ya que grandes multinacionales como Google tienen, como es lógico, un avance mucho mayor en la materia estudiada.

7.4. Dimensión ambiental

En este proyecto se ha conseguido una planificación que permite que todo el desarrollo se haga con un único ordenador, y este, es el único impacto ambiental (junto a su consumo eléctrico).

Las características del ordenador, como ya se ha descrito previamente, son estándar, al igual que sus periféricos (pantalla, ratón, teclado). Esto quiere decir que el impacto ambiental de este proyecto no es superior al impacto que podría tener una persona que hace sus quehaceres diarios desde un ordenador con una dedicación de horas similar.

8. Battle for Wesnoth

8.1 Componentes relevantes

Hay múltiples componentes que forman el juego, en este caso los relevantes para el desarrollo de este proyecto son los siguientes:

Unidades: Son los personajes controlables tanto por el jugador como por la IA que interactúan entre ellos y con el entorno, estos tienen varios parámetros que son necesarios tener en cuenta a la hora de valorar la recompensa de una acción. Al mismo tiempo, las unidades tienen ciertos atributos que afectaran a su capacidad de movimiento y a su capacidad de combate:

- Nivel
- Vida
- Defensa
- Armas
    - Daño
- Experiencia
- Posición
- Tipo de unidad
- Puntos de acción

Facción: Las unidades pertenecen a un bando el cual recae sobre el control de un jugador o IA

Escenarios: Es el mapa que representa el terreno de juego, en este mapa las unidades se desplazan por el para llevar a cabo sus acciones de movimiento o combate. El escenario está formado por múltiples casillas, las cuales según su tipo tienen ciertos efectos como el modificar los atributos de las unidades que se encuentra en la casilla, y en algunos casos, dependiendo del tipo de unidad, también cuentan con un coste de puntos de movimiento que son los puntos necesarios para atravesar esa casilla, dependiendo de la unidad que intente atravesar la casilla el coste variara, ya que hay facciones que son rápidas en según que entornos.

Condición de victoria: Dependiendo el escenario o la campaña la condición de victoria puede variar, pero generalmente es una de las siguientes:

- Acabar con todas las unidades enemigas

- Acabar con el comandante enemigo

- Sobrevivir X turnos

- Alcanzar una posición objetivo

8.2 Tipos de acciones

Las unidades tienen varios tipos de acciones dependiendo del escenario, las de ataque que inician una fase de combate entre dos unidades y consumen todos los puntos de acción de la unidad que inicia el ataque. Después están las de desplazamiento, que a diferencia de las acciones de ataque consumen puntos de acción equivalentes a la suma del coste de atravesar todas las casillas que se han atravesado en el desplazamiento.

También hay otros dos tipos de acciones en algunos escenarios:

El reclutamiento de unidades, bajo ciertas condiciones y bajo un coste de oro el comandante puede reclutar nuevas unidades al campo de batalla dependiendo de su facción, para esto es necesario tener oro suficiente para reclutar la unidad y que el comandante este posicionado en el castillo.

Sistema de aldeas: Las aldeas se puede conquistar cada vez que se completa un turno, las aldeas conquistadas recompensan con oro al jugador, este oro tiene un único uso y es el de permitir al comandante reclutar nuevas unidades.

Después de analizar el impacto que tiene en las partidas el reclutamiento se ha decidido que esta funcionalidad no estará presente en los escenarios usados para entrenar al Agente Inteligente, los tres principales motivos de esta decisión son que se busca simular un comportamiento similar a otros juegos de estrategia por turnos como Fire Emblem o Shining Force donde no existe este tipo de funcionalidad, el segundo motivo, es que después de analizar partidas donde se usa esta funcionalidad, muchas veces actúa en detrimento de la partida provocando que la duración de la partida se multiplique por varias veces y, el tercer y último motivo, es que el objetivo es codificar las acciones de combate, con lo que las acciones de reclutamiento no se tratarían eficazmente (ni siquiera la IA por defecto del juego recluta eficazmente, está programada para reclutar siempre la unidad más barata). Las aldeas sí que estarán presentes en el mapa, pero serán otro tipo de terreno más con sus bonificaciones, ya que el oro no tiene más uso que el de reclutar unidades. Por el contrario, sí que se tendrán en cuenta durante la segunda fase de generación de escenarios, para que así los mapas generados tengan la posibilidad de jugarse en los diferentes modos con el resto de las funcionalidades si así lo desea el jugador.

8.3 Comportamiento de la IA por defecto

La IA principal del juego se conoce como RCA AI y funciona a partir de una lista de acciones candidatas, cada turno evalúa el resultado de una acción y se ejecuta según la puntuación de evaluación de la lista.

Está formada por cuatro componentes principales:

- El main_loop stage, que es el bucle que evalúa las decisiones.
- La lista de acciones candidatas, que son todas las acciones disponibles en un turno
- Los Aspectos que sirven para modificar ciertos parámetros que afectan al comportamiento de la IA.
- Las metas, son posiciones en el mapa que priorizara el algoritmo, solo se aplica en acciones de movimiento.

En la versión actual los scores se distribuyen de la siguiente forma:

Goto (score 200.000): Son las metas descritas anteriormente.

Retreat Injured (192.000-95.000): Retira unidades en peligro.

Spread Poision (190.000): Intenta envenenar unidades, evitando unidades a las que no se puede envenenar o que ya hayan sido envenenadas.

Move leader to goals (140.000): Mueve el líder hacia el objetivo de la meta.

Move leader to Keep (120.000): Mueve el líder hacia el castillo.

Combat (100.000): Ataca a un enemigo a rango.

Healing (80.000): Retira aliados dañados a sanarse en aldeas (si es que está habilitada esta opción).

Villages (60.000): Mueve la unidad a una aldea desocupada u ocupado por el enemigo.

Move to any enemy (1.000): Se aproxima a un enemigo, esta acción ocurre en los modos que se van a usar como Escenarios ya que, el desarrollo del Agente se va a centrar en el combate y en escenarios donde no hay Reclutamiento.

8.4 Complejidad computacional

La complejidad computacional del juego es muy alta, el mapa tiene más de 220 tipos de terreno básicos, pero además existen combinaciones de terrenos que amplían por mucho este valor, el tamaño de un mapa es muy variable, pero por lo general suelen rondar entre los 15x15 y los 30x30 (No necesariamente deben ser igual de altos que de anchos). Esto hace que sin aun haber añadido unidades ya estemos hablando de un escenario de por ejemplo 30x30 donde cada una de las 900 casillas que lo forman tienen muy por encima de 220 alternativas por casilla. A la que añadimos una unidad, se añaden varios parámetros que llevan aún más allá esta complejidad, por ejemplo, su posición en una de las 900 casillas, el tipo de unidad, su estado (Envenenado, petrificado, etc…), su ataque, su vida, el número de armas que posee, movilidad, alcance, etc.

Esto hace que para modelar este entorno y poder hacer un Agente y un Algoritmo de generación procedural en el tiempo que hay destinado al proyecto y más teniendo en cuenta que el Agente y el Algoritmo genético son partes separadas y bien diferencias del proyecto, se ha optado por simplificar ciertos parámetros en cada una de las partes:

Simplificaciones para el Agente: En el caso del Agente y para simplificar el conjunto de estados sin afectar al rendimiento final contra la IA por defecto, se ha optado por representar únicamente pero de forma bastante detallada los estados y acciones relacionadas con el combate, esto quiere decir, que el agente tomara la decisión de o bien hacer una acción de aleatoria o, la mejor acción de combate posible según el conocimiento actual dependiendo de la variable Epsilon-greedy que se detallara en apartados posteriores, mientras que las acciones de movimiento se limitaran a acciones de aproximación hacia los enemigos.

Simplificaciones para el Algoritmo genético de Generación Procedural: En este caso, las limitaciones vienen en el número total disponible de tiles, tanto las básicas como sus combinaciones. Además, estará limitado a la geografía del terreno, la construcción del camino entre los castillos de ambos jugadores está garantizado (siempre debe existir) y se añadirá a cada uno de los individuos de la última generación, para crear el camino sin ser demasiado disruptor en la geografía del mapa se utilizará una implementación del algoritmo de búsqueda A*.

9. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un modelo de aprendizaje automático donde mediante una secuencia de decisiones, el Agente Inteligente intenta alcanzar una meta en un entorno potencialmente complejo e incierto. A través de un método de prueba y error, el agente interactúa con el entorno obteniendo recompensas o penalizaciones dependiendo de la calidad de la decisión tomada, para finalmente alcanzar una solución al problema. En este caso, y, a diferencia de otros modelos como el de aprendizaje supervisado el agente no obtiene ninguna información externa más allá de la recompensa por sus acciones directas sobre el entorno.

9.1 Definiciones

Para entender el aprendizaje por refuerzo es necesario definir varios conceptos:

- **Agente Inteligente:** El agente es el algoritmo que realiza las acciones interactuando con el entorno.

- **Entorno:** Es el escenario en el que interactúa el agente. En este entorno el agente mediante un estado actual y una acción es capaz de avanzar a un nuevo estado. En

este proyecto, el entorno es el mapa con las unidades alias/enemigas y las diferentes casillas con sus características que forman el mapa.

- **Estado:** Un estado es la definición de un momento concreto del entorno en el tiempo. En nuestro caso un estado es definido a partir de las posiciones y estado de los atributos de las unidades que están en juego.

- **Acción:** Es la actividad o movimiento que el agente puede desencadenar con tal de moverse entre los diferentes estados, en el caso de Battle for Wesnoth, una acción puede ser inicializar un combate entre dos unidades que como resultado afectará a las estadísticas de ambas unidades que participan en la lucha o incluso la desaparición de estas del terreno de juego, también hay acciones que no son de combate, si no de desplazamiento, donde una unidad avanza hasta otra casilla libre en el terreno de juego.

- **Recompensa:** La recompensa es la forma en la que el Agente recibe feedback de que tan buena o mala ha sido su decisión tras una acción. Un ejemplo de decisión que tendrá una recompensa positiva es acabar con una unidad enemiga, mientras que perder una unidad en el turno del agente será tomada como una decisión negativa.

9.2 Proceso de Decisión de Markov

Ya que nuestro entorno es discreto se modela como un problema de Decisión de Markov, ya que estos sirven para modelar sistemas dinámicos estocásticos controlados, se definen de la siguiente forma:

- Un conjunto de estados S (que contiene un estado inicial S_0)
- Un conjunto de acciones A aplicables desde el estado actual
- La probabilidad de transición para cada par de estados y una acción aplicable sobre el estado actual
- La función de recompensas R para las acciones de transición entre estados

Con la probabilidad de transición entre estados y sus recompensas se define el modelo del mundo. Si el modelo es conocido, es posible calcular de manera directa la estrategia óptima para la resolución del problema mediante el uso de la programación dinámica, en caso contrario, si el modelo es desconocido habrá que aproximas mediante estimaciones las recompensas futuras al decidir ejecutar una acción desde un estado. En el caso de este proyecto el modelo es desconocido, por lo tanto, el algoritmo será off-policy.

9.3 Off-policy vs On-policy

Los algoritmos de aprendizaje por refuerzo Off-policy, evalúan una política que sea diferente de la política observacional utilizada para generar los datos. Esto contrasta con los algoritmos

On-policy, donde una política *p* se actualiza a través de los datos recopilados por la propia política.

Un ejemplo de un algoritmo On-policy es SARSA.

Para el agente de este proyecto se va a aplicar la estrategia Off policy, más concretamente el algoritmo Q-Learning que estima un valor a cada par Estado-Acción bajo una política voraz (Greedy).

9.4 Exploración vs Explotación

Como se ha visto en la sección 9.2, en el aprendizaje por refuerzo el algoritmo toma decisiones sobre qué acción ejecutar desde un estado en un momento concreto, y para tomar esta decisión, es el propio agente el que mediante prueba y error debe almacenar la información sobre las recompensas obtenidas al tomar esa decisión en otras situaciones previas. Es por esto por lo que las decisiones a tomar se dividen en dos tipos:

- Acciones de Exploración: Se toma una decisión con el objetivo de recolectar nueva información.

- Acciones de Explotación Se toma una decisión con el objetivo de que con la información actual que se dispone sea la mejor posible.

Durante una acción de exploración aleatoria lo que se trata es de intentar alcanzar nuevos estados y acciones en el espacio de búsqueda, por el contrario, las de explotación tratan de aprovechar la información adquirida durante las exploraciones para alcanzar la solución.

Es importante que el equilibrio entre ambos tipos de acciones sea el oportuno para el espacio de búsqueda, de lo contrario podemos encontrarnos con algoritmos que exploran demasiado, perdiendo oportunidades al no escoger las mejores opciones disponibles, o con algoritmos que solo tratan de escoger las mejores acciones posibles según la información actual, cosa que si la exploración previa no ha sido suficiente, también se perderán oportunidades al no haberlas descubierto.

El mecanismo más popular para evitar este tipo de situaciones es el epsilon greedy (ε-greedy). El parámetro ε se sitúa entre 0 y 1. El agente a la hora de escoger una acción, escogerá una acción de exploración opciones con una probabilidad de 1- ε y en el resto de los casos una acción de explotación. Esto evidentemente tampoco es una buena solución por si sola ya que aun con la mayor parte del espacio de búsqueda explorado y, por la tanto, bajo el conocimiento del Agente, en caso de que por ejemplo ε fuera 0.5 aun en la mitad de las ocasiones seguiría seleccionando acciones de exploración. Es por eso por lo que para evitar este tipo de situaciones el valor de ε debe comenzar en un numero cercano a 1 y a partir de ahí debe ir decayendo hasta alcanzar un mínimo cercano a 0, o incluso 0 en algunos casos, de esta manera, empezará el algoritmo con una gran parte de los movimientos siendo de exploración y acabara prácticamente haciendo todos de explotación una vez ya se tiene una información considerable sobre el entorno.

9.5 Métodos de Aprendizaje

- **Métodos de Monte Carlo:** En este método el Agente aprende directamente a través de la experiencia de una interacción con el entorno. No es necesario el conocimiento del mundo.

- **Programación dinámica:** En este tipo de algoritmo el conocimiento del mundo es necesario y se usan para procesar un modelo completo del entorno mediante un Proceso de Decisión de Markov. En general al requerir un modelo perfecto del entorno a efectos prácticos suele ser costosos y más complejos de modelar y aplicar.

- **Time Difference Learning (TD Learning):** Es el resultado de la combinación de los métodos de Monte Carlo y la programación dinámica, en este método el modelo del mundo no es necesario, ya que al igual que con los métodos de Monte Carlo se puede aprender directamente de la experiencia en la interacción con el entorno, su principal característica es que en TD Learning es posible estimar un valor de retorno sin completar una secuencia de estados-acciones. Para este proyecto se va a utilizar un método de TD Learning.

9.6 Q-Learning

Q-Learning es un algoritmo Off-Policy del tipo TD-Learning donde el conocimiento se presenta en valores Q(s,a). Donde Q(s,a) representa el descuento esperado al ejecutar la acción (a) desde el estado actual (s).

El objetivo de este algoritmo es maximizar la recompensa esperada eligiendo la mejor acción desde un estado. Esto se representa media la Q-function también conocida como la ecuación de Bellman:

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

Q(s,a) se interpreta como la recompensa al aplicar la acción a sobre el estado s, indicando la calidad de esta transición. La ecuación de Bellman se describe la mayor recompensa para un par (s,a) es la recompensa inmediata más el máximo de la recompensa futura del siguiente estado.

```
-- PSEUDOCODIGO Q-LEARNING --

Se escogen los parametros para el algoritmo (α (0,1] y ε > 0)
Se inicializa la Qtable Q(s,a)

Loop por cada episodio:
    Inicializar estado s
    Loop por cada paso del episodio:
        Elegir una accion a del estado s usando la politica derivada de Q (en nuestro caso ε-greedy)
        Ejecutar la acción a, recibir el feedback en forma de recompensa r y alcanzar un nuevo estado s'
        Se aplica la ecuación de Bellman para calcular Q(s,a)
    repetir desde el nuevo estado s'
```

Pseudocódigo del algoritmo Q-learning (Elaboración propia)

Se inicializa la tabla y se va actualizando a medida que el agente selecciona una acción que permite la transición de un estado s a un nuevo estado s', al completarla recibe una recompensa inmediata r que permitirá actualizar el valor de la función Q.

La variable ε indica el factor greedy y es la tasa que indica la probabilidad de que se seleccione una acción de exploración o una de exploración. Mientras que la variable γ indica el factor de descuento que sirve como ponderación de las recompensas futuras contra las anteriores.

9.6.1 Agente Q-Learning: Representación de Estados, Acciones y Recompensas

Para el diseño e implementación de este agente, se han probado varias representaciones de estados, así como de funciones de recompensas en varios escenarios para tratar de encontrar un modelo lo más general posible. Durante el desarrollo se ha probado una representación con la vida actual de las unidades aliadas y enemigas, con una función de recompensa que asignaba el valor de recompensa según la diferencia de vida de las unidades enemigas antes del turno – la vida de las unidades enemigas tras el turno.

La segunda representación calculaba la suma total de vidas, ataque y defensa de ambas facciones, y calculaba la diferencia.

Finalmente, en la tercera y última representación de los estados, se ha optado por una representación más fiel a cada unidad por separado ya que las unidades se mueven de manera individual se ha considerado importante tener la información más individualizada, por lo que los estados se representan con Vida de la unidad, Ataque de la primera arma, Ataque de la segunda arma (En ambos ataques ya se incluye el número de ataques totales por arma).

Esta última representación tiene una característica extremadamente buena, y es que el aprendizaje es útil para cualquier escenario del juego, puedes utilizar el conocimiento adquirido de un mapa diferente y que sea útil en un nuevo mapa, esto es debido a que codifica las estadísticas de vida y daño de las armas, algo que no se ve alterado al cambiar de mapa y que incluso una misma entrada, puede dar un buen resultado a diferentes tipos de unidades que en un momento del tiempo coincidan sus estadísticas.

En el caso de la función de reward definitiva, esta otorga a una acción una recompensa de la siguiente forma:

Sí la unidad que ejecuta la acción elimina a una unidad enemiga la función de rewards otorga 50 puntos, si el combate termina sin bajas en ese caso el reward se calcula a partir de cuanto daño a aplicado al enemigo – cuanto daño a recibido por parte del enemigo.

9.6.2 Escenarios implementados

Para probar el comportamiento del Agente Inteligente, se van a probar 4 escenarios. De dificultad ascendente.

9.6.2.1 Escenario 1

El escenario 1 es el más básico de todos, el tamaño del escenario es de 15x15, solo hay un tipo de terreno (Pasto Verde) con lo que las bonificaciones de terreno y sortear obstáculos del terreno quedan fuera de la ecuación. Además, todas las unidades serán las mismas para las dos facciones, aun así, no será exactamente igual para ambos bandos ya que aun siendo la misma unidad hay cierta diversidad en las estadísticas de las unidades (Vida, Ataque, Numero de ataques, etc.) aunque las estadísticas finales cuentan con un cierto balance ya que la suma ponderada de sus estadísticas debe ser similar.

Elementos relevantes

En este caso los elementos para tener en cuenta son las 8 unidades aliadas y sus estadísticas, así como las 8 unidades enemigas y sus correspondientes estadísticas.

Recompensa

La recompensa es equivalente a la resta del daño provocado – daño recibido, con una bonificación de +50 puntos en caso de que una unidad enemiga sea derrotada por esa acción.

Expectativa

Se espera que el Agente pierda frente a la IA por defecto durante las primeras partidas mientras siga tomando opciones de Exploración. Pero, poco a poco, la diferencia debería irse reduciendo a medida que las acciones de explotación empiezan a tomar protagonismo, se espera al final que el Agente inteligente supere con solvencia a la IA por defecto, ya que en este escenario, la elección de cuándo y a quien atacar es lo más importante, y esto es en lo que se especializa el Agente.

9.6.2.2 Escenario 2

El escenario 2 se desarrolla en mismo mapa que el primer escenario, donde: solo hay un tipo de terreno (Pasto Verde) con lo que las bonificaciones de terreno y sortear obstáculos del terreno quedan fuera de la ecuación. Pero, esta vez habrá 8 tipos de unidades distintas por lo que el uso individualizado de ellas será vital para conseguir un buen rendimiento, en caso de la IA por defecto a largo plazo no debería comportarse tan bien como el Agente que gracias a representación de los estados y las acciones que, aunque no se recoja el tipo de unidad de manera directa sí se recoge la información relevante de sus estadísticas que al final es lo importante a nivel de jugabilidad y rendimiento.

Elementos relevantes

En este caso los elementos para tener en cuenta son las 8 unidades aliadas y sus stats, así como las 8 unidades enemigas y sus stats.

Recompensa

La recompensa es equivalente a la resta del daño provocado – daño recibido, con una bonificación de +50 puntos en caso de que una unidad enemiga sea derrotada por esa acción.

Expectativa

Se espera que el Agente pierda frente a la IA por defecto durante las primeras partidas mientras siga tomando opciones de Exploración. Pero poco a poco la diferencia debería irse reduciendo a medida que las acciones de explotación empiezan a tomar protagonismo, se espera que se pueda llegar a superar el comportamiento de la IA por defecto en este escenario.

9.6.2.2 Escenario 3

El escenario 3 se desarrolla en mismo mapa que el primer y el segundo escenario, donde: solo hay un tipo de terreno (Pasto Verde) con lo que las bonificaciones de terreno y sortear obstáculos del terreno quedan fuera de la ecuación. Pero, esta vez, además de 8 tipos de unidades distintas, se suma la introducción del comandante, también en este caso la IA por defecto a largo plazo no debería comportarse tan bien como el Agente en la gestión de las múltiples unidades de diferente tipo (Escenario 2) pero sí que gestiona mejor el uso del comandante. Se compararán los resultados con los del escenario 2.

Elementos relevantes

En este caso los elementos para tener en cuenta son las 8 unidades aliadas y sus estadísticas, así como las 8 unidades enemigas y sus estadísticas. Además de las estadísticas de los comandantes.

Recompensa

La recompensa es equivalente a la resta del daño provocado – daño recibido, con una bonificación de +50 puntos en caso de que una unidad enemiga sea derrotada por esa acción.

Expectativa

Se espera que el Agente pierda frente a la IA por defecto durante las primeras partidas mientras siga tomando opciones de Exploración. Pero poco a poco la diferencia debería irse reduciendo a medida que las acciones de explotación empiezan a tomar protagonismo, es el escenario mas complicado se espera que el resultado mejore y se aproxime a la IA por defecto e incluso la supere. Habrá que ver si el factor comandante es crítico en el resultado.

9.6.2.2 Escenario 4

El escenario 4 se desarrolla en un nuevo escenario existente en el juego sin ningún tipo de limitación a diferencia de los anteriores.

Elementos relevantes

En este caso los elementos para tener en cuenta son las 10 unidades aliadas y sus estadísticas, así como las 10 unidades enemigas y sus estadísticas. Diferentes tiles de terreno con sus diferentes bonificaciones, caminos que permitan circular a las unidades entre la geografía del terreno.

Recompensa

La recompensa es equivalente a la resta del daño provocado – daño recibido, con una bonificación de +50 puntos en caso de que una unidad enemiga sea derrotada por esa acción.

Expectativa

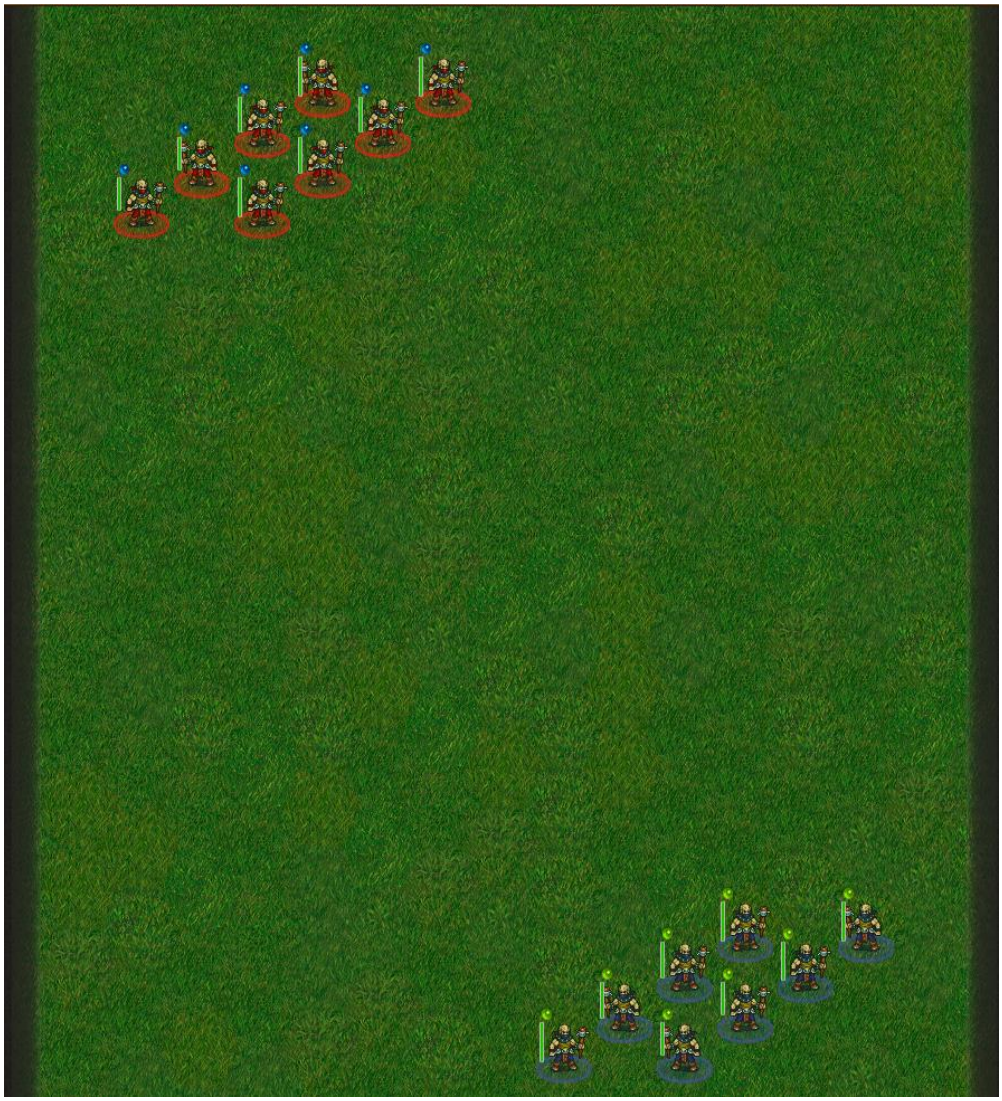
Se espera que el Agente pierda frente a la IA por defecto durante las primeras partidas mientras siga tomando opciones de Exploración. Pero poco a poco la diferencia debería irse reduciendo a medida que las acciones de explotación empiezan a tomar protagonismo, se espera que se pueda acercar e incluso superar a la IA por defecto, aunque preocupa la posible superioridad en el despliegue de tropas por parte de IA por defecto ya que aquí el terreno es variable y puede haber bloqueos naturales que provoquen dispersión de las tropas dejándolas a merced de emboscadas. Es el escenario mas complicado.

9.7 Resultados

En este apartado se van a estudiar los resultados obtenidos tras ejecutar las pruebas en los escenarios descritos en el apartado 9.6.2 y se va a comparar el resultado esperado frente al resultado final.

9.7.1 Resultado escenario 1

Previsualización del escenario



Durante el entrenamiento

Sorpresivamente incluso durante las primeras partidas no se observa una ventaja para la IA por defecto, y partir de la partida 8 empieza a encadenar una tasa de victorias superior al 50%.

Se observa también un comportamiento interesante por parte de la IA por defecto, esta no inicia el ataque en turno 1 si no que espera a que el enemigo se aproxime.

Al alcanzar la partida número 20 se empiezan a contabilizar los resultados ya que al no haber variedad de unidades el conjunto de estados posible es pequeño en comparación a situaciones con variedad de unidades.

Se han jugado otras 100 partidas y el resultado es el siguiente:

El resultado es 67 victorias para el Agente por 33 de la IA por defecto.

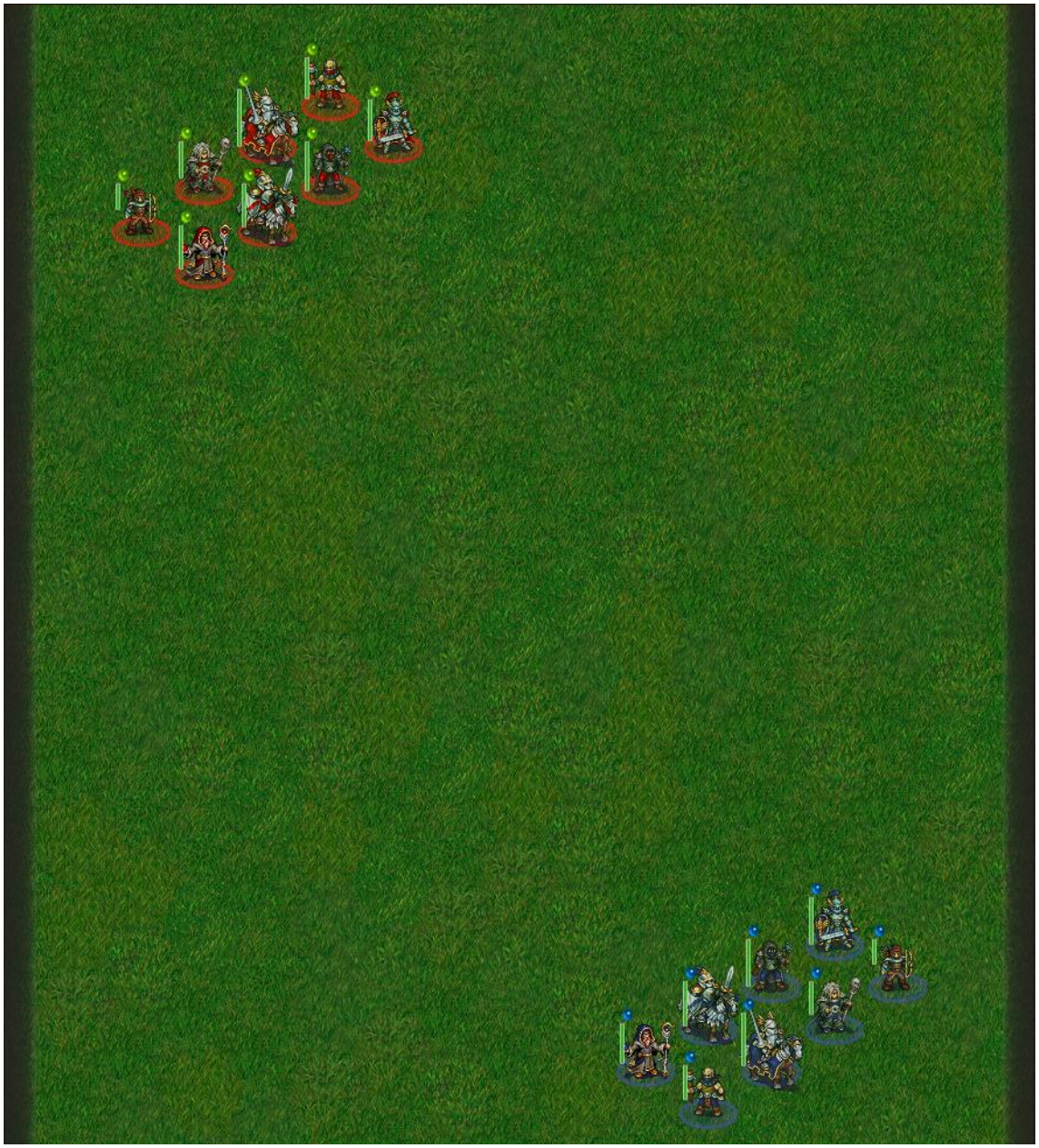
Elaboración propia

Se ha observado que el factor aleatorio de fallar ataques tiene un mayor impacto de lo esperado, lo que provoca que el Agente tenga falsos positivos y falsos negativos aprendidos en el conocimiento.

Aunque el resultado ha sido positivo, para confirmar estos porcentajes sería una buena práctica ejecutar al menos varios cientos de veces más este escenario, pero debido a que no es posible acelerar las partidas y que se requiere la ejecución de un script entre medias de cada partida (por motivos que se explicaran en el apartado 11.2), se ha hecho imposible poder ejecutar ese número de pruebas.

9.7.2 Resultado escenario 2

Previsualización del escenario



El entrenamiento se lleva a cabo durante 100 partidas. Durante estas partidas la IA por defecto se impone utilizando la misma estrategia del primer escenario esperando a que se aproximen los enemigos, a diferencia de la anterior partida donde el daño de las unidades era bajo en comparación con la vida, esto, sumado a que las acciones de exploración del agente sufren una mayor penalización le da la ventaja durante esta fase a la RC AI.

Se han jugado otras 100 partidas y el resultado es el siguiente:

El resultado es 54 victorias para el Agente por 46 de la IA por defecto.

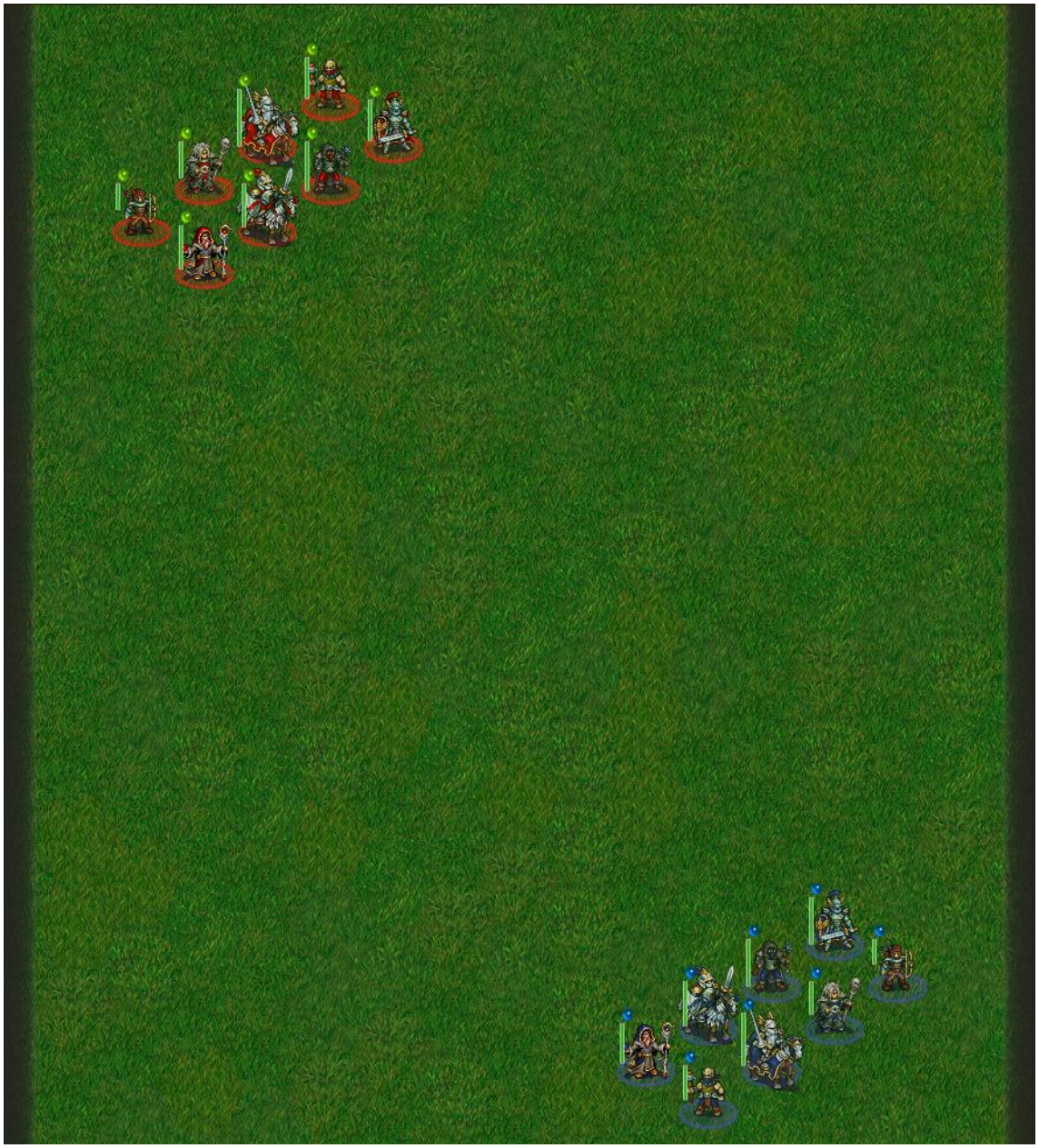
Se ha observado que el factor aleatorio de fallar ataques tiene un mayor impacto de lo esperado, lo que provoca que el Agente tenga falsos positivos y falsos negativos aprendidos en el conocimiento.

Aunque el resultado ha sido positivo, tras observar detenidamente no solo el resultado de las ultimas partida si no todo el desarrollo de estas se confirma aun tras 200 partidas el Agente se ve obligado por la falta de conocimiento a seguir tomando algunas acciones de exploración, por lo que sigue haciendo algunos movimientos de exploración debido a que en ocasiones no hay acciones con una recompensa conocida disponibles.

Tras otras 100 partidas más se obtiene el siguiente resultado (total 200):

9.7.3 Resultado escenario 3

Previsualización del escenario



El entrenamiento se lleva a cabo durante 100 partidas. Durante estas partidas la IA por defecto cambia de estrategia y mantiene en la retaguardia al comandante mientras avanza con el resto de las tropas.

Se han jugado otras 200 partidas y el resultado es el siguiente:

El resultado es 78 victorias para el Agente por 122 de la IA por defecto.

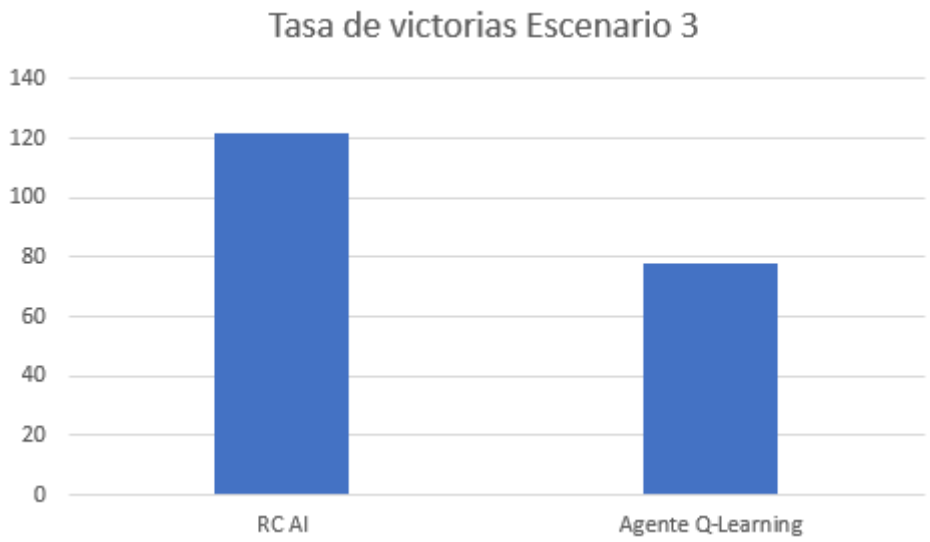


Grafico tras las 200 partidas

Al igual que en el resto de los escenarios la aleatoriedad de fallar ataques afecta a los resultados a lo poder hacer pruebas sobre un gran conjunto de partidas.

El resultado no ha sido bueno en comparación a la IA por defecto, aunque se observa una mejora sustancial en las ultimas 50 partidas donde la tasa de victorias ha sido cercana al 50%:

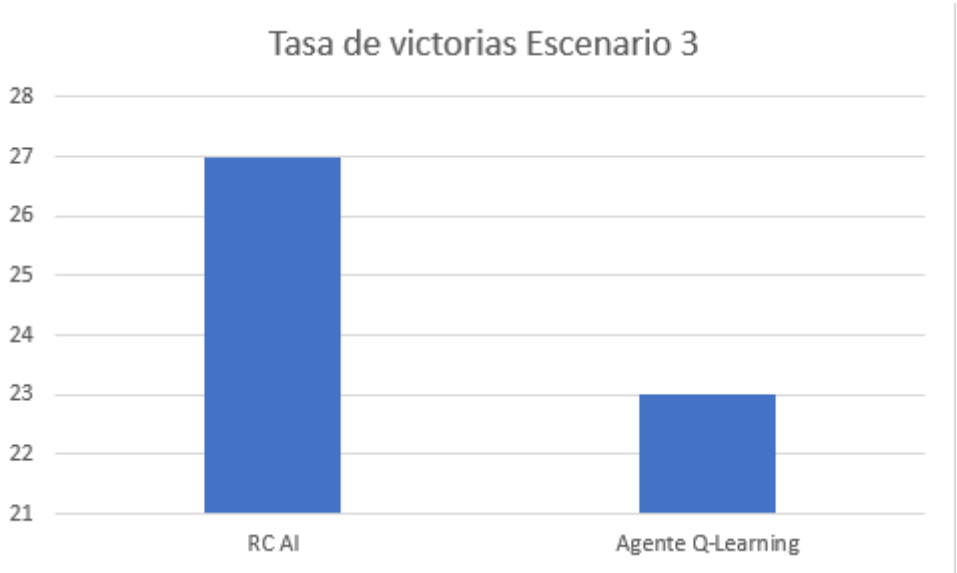
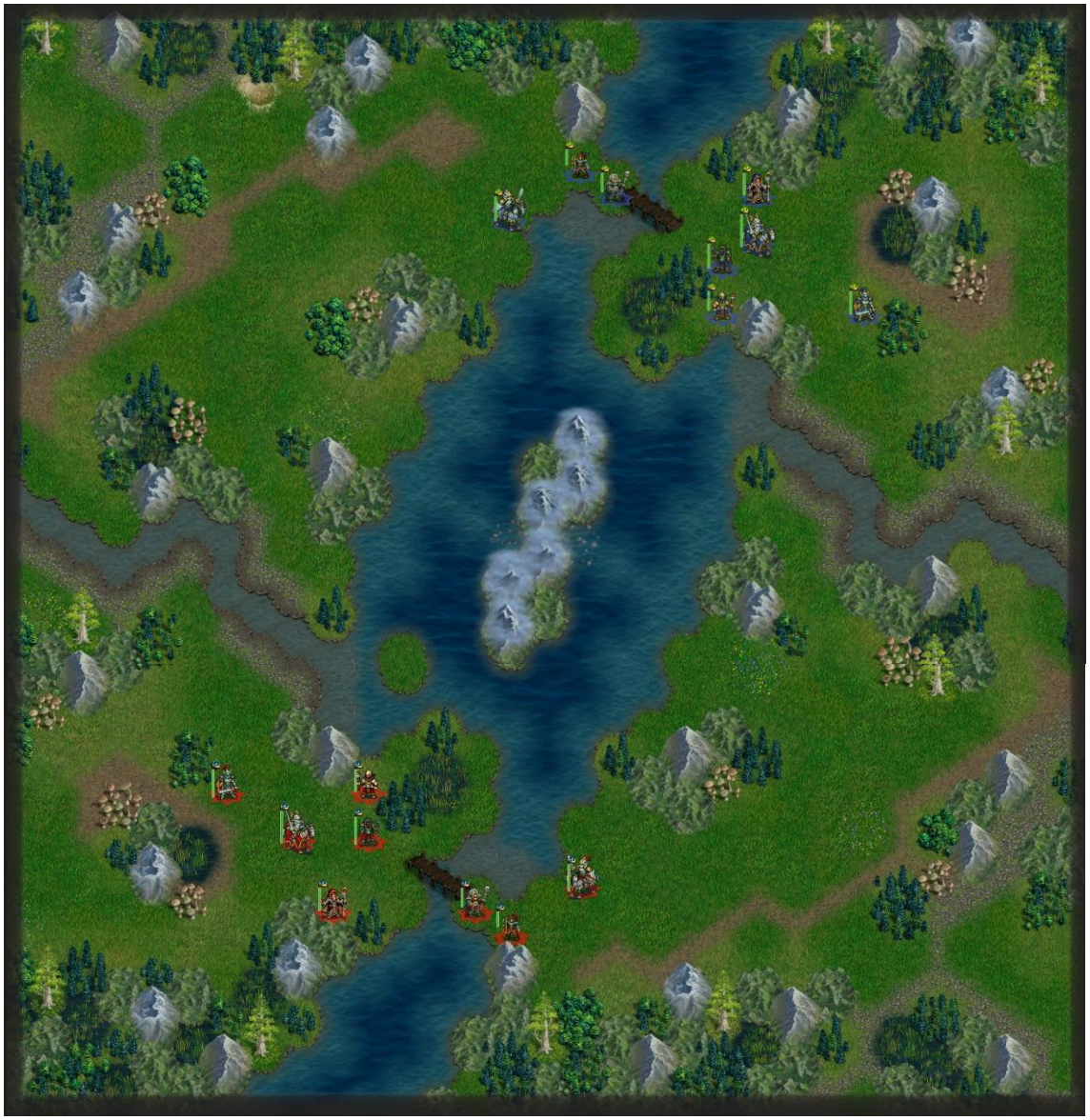


Grafico de las ultimas 50

Es probable que con un mayor conocimiento pudiera superar a la IA por defecto, aunque ha quedado claro que la estrategia alrededor del comandante es superior en la IA por defecto, algo que tampoco sorprende teniendo en mente que no se han modelado el Agente con los comandantes en mente.

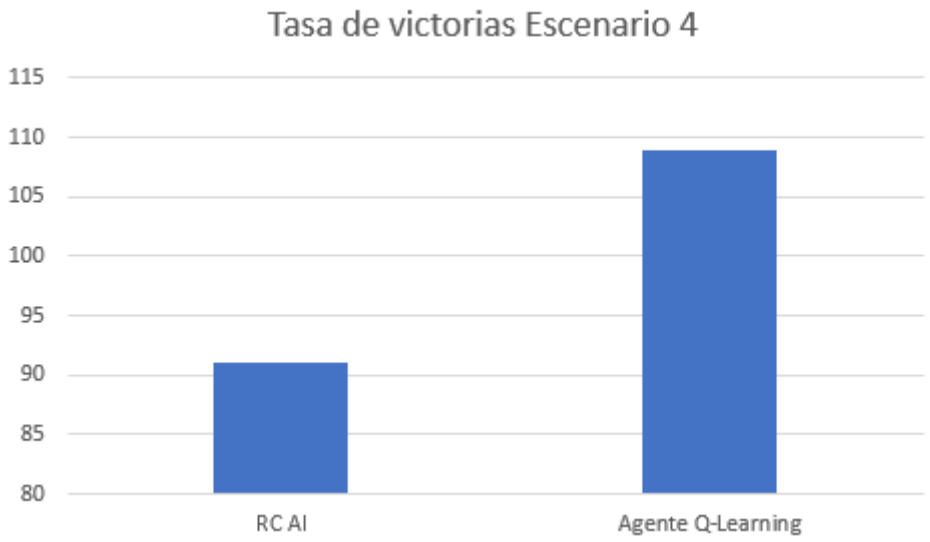
9.7.1 Resultado escenario 4

Previsualización del escenario:



El entrenamiento se lleva a cabo durante 200 partidas. Durante estas partidas la IA por defecto vuelve a la estrategia de mantener su posición.

Se han jugado otras 200 partidas y el resultado es el siguiente:



Victorias en las 200 partidas tras 200 de entrenamiento

Como se puede observar el resultado es ligeramente peor que en el escenario 4, es algo normal debido a que al final a pesar de que el escenario es mucho más complejo, la IA por defecto opta por una estrategia pasiva, lo que estrecha el cerco donde se va a combatir, la perdida de ratio de victorias pese a haber tenido más entrenamiento puede deberse a dos factores:

- La aleatoriedad de fallar ataques durante el combate
- Que al ser un mapa más largo las unidades se dispersan más mientras el enemigo mantiene el núcleo junto, esto provoca que la caballería que llega antes al enemigo no sea tan eficaz al estar en inferioridad, esto no ocurría en el escenario 2 debido a que ambos equipos estaban mucho más cerca unos de otros.

10.0 Generación Procedural Evolutiva de Escenarios

La generación procedural de contenidos consiste en generar mediante algoritmos, reglas o programación evolutiva contenido jugable. Este contenido jugable puede ir desde las estadísticas de un objeto o personaje, pasando por el propio diseño de un personaje hasta llegar a poder construir escenarios enteros con estas estrategias. Estos mapas están formados por múltiples.

10.1 Generación de escenarios en Battle for Wesnoth

En este proyecto se trata de crear la geografía de un escenario 2D formado por casillas, cada casilla contiene un terreno con sus características particulares. En este tipo juegos los escenarios suelen estar ofrecer diferentes variedades de terreno en cada mapa para ofrecer múltiples opciones estratégicas para las diferentes facciones, esto hace que ciertos pequeños biomas sean necesarios para mantener la coherencia del escenario, pero sin ocupar grandes extensiones de terreno sin variedad.

Aunque originalmente no existe como característica de la celda, se ha creído importante añadir que cada casilla tenga su propia representación de altura, ofreciendo así posteriormente la capacidad de crear un heurístico alrededor de esta información, que será importante para evitar grandes desniveles de altura en celdas adyacentes. El juego tiene mas de 220 Tiles debido a las múltiples campañas con las que consta el juego, para el algoritmo hemos seleccionado las relevantes para el juego principal ya que muchas de esas 220 son simplemente modificaciones visuales de las de la campaña principal.

10.2 Tiles utilizados por el algoritmo

Para la selección de tiles se han escogido los originales de la primera campaña, y entre estos se han escogido los más relevantes divididos por alturas para abarcar todas las alturas disponibles

Terreno	Nombre	Codificación	Altura
	Agua	Ww	0
	Playa	Ds	1
	Desierto	Dd	2

	Pasto seco	Gd	2
	Pasto semi-seco	Gs	2
	Pasto verde	Gg	2
	Pasto verde + Bosque combinado	Gg^Fms	2
	Pantano	Ss	2
	Dunas	Hd	3
	Colinas secas	Hhd	3
	Colinas + Bosque combinado	Hh^Fms	3
	Montañas secas	Md	4
	Montañas	Mm	4
	Montañas nevadas	Ms	5

10.3 Algoritmos evolutivos

Muchos problemas de optimización tienen una alta complejidad para solucionar por medio de técnicas tradicionales, es aquí donde entran en juego los algoritmos evolutivos, estos algoritmos utilizan estrategias que imitan la evolución natural de las especies. Sobre este esquema los algoritmos evolutivos se clasifican en cuatro métodos:

- **Estrategias Evolutivas:** Fueron diseñados para afrontar la resolución de problemas de optimación tanto discretos como continuos, trabajan con codificaciones de las posibles soluciones para problemas aritméticos y utilizando los operadores de Selección, Cruce y mutación. Alcanza la optimización de soluciones tras eliminar las peores soluciones que están por debajo del promedio de aptitud.

- **Programación Evolutiva:** La representación de estados se hace mediante números reales para cualquier estructura de datos y se emplean también los operadores de mutación y selección. En este caso la principal diferencia con el resto de los métodos es que no se emplea operador de cruce. Es la principal característica que los diferencia de los algoritmos Genéticos.

- **Algoritmos Genéticos:** Modelan el proceso evolutivo natural con representaciones de genes y cromosomas. Trabajan con estas representaciones del problema y aplican transformaciones a estas soluciones mediante los operadores de: Selección, Cruce y Mutación. Durante la fase de selección se escogen los individuos con una probabilidad p según su calidad determinada por la función de fitness. Durante esta fase de selección es posible aplicar una variante del operador de selección: **elitismo** que hace que un porcentaje de los individuos más aptos se traspase directamente a la siguiente generación saltándose así los pasos de Cruce y Mutación, aunque una copia suya aún puede ser seleccionada para ser cruzada y mutada en otra selección, esta solución es por la que se ha optado en este proyecto y se ampliara en los siguientes apartados.

- **Programación Genética:** Los individuos son programas o autómatas, representados como árboles y usando operadores de Cruce y Mutación además de los mecanismos de selección de los Algoritmos Genéticos.

10.4 Representación Algoritmo Genético

En los algoritmos genéticos es necesario definir ciertos conceptos, por un lado, el concepto de **población** hace referencia a un conjunto de soluciones generada en un paso del algoritmo, estas soluciones que forman parte de la población son llamadas **individuos**, estos individuos al mismo tiempo están formados por un conjunto de variables que forman parte del proceso de optimización llamados **cromosomas** y al mismo tiempo estos cromosomas están formados por cadenas donde cada elemento de esta cadena es un **gen**, cada gen se representa como un valor entre 0 y 255 que posteriormente se traduce en un tile.

Cada uno de los individuos es una solución del espacio de búsqueda, y estos, evolucionan a través de las generaciones, a través de los operadores de Selección, Cruce y Mutación.

En lo que hace a Battle for Wesnoth, se ha optado por la siguiente codificación:

La población es el conjunto de escenarios obtenidos en cada generación, en la generación inicial los mapas son de tamaño N*M (es variable según definición inicial y el tamaño no varía durante los saltos generacionales), y las celdas se completan aleatoriamente con las opciones definidas en el apartado 10.2.

Los individuos son cada uno de los escenarios de una población.

Los cromosomas son la representación del escenario codificada por celdas.

Los genes son cada una de las celdas.

10.5 Operadores

Como en todo algoritmo genético el algoritmo de este proyecto cuenta también con los tres principales operadores: Selección, Cruce y Mutación.

10.5.1 Operador de Selección

Se han desarrollado dos versiones de este operador, ya que la primera versión el algoritmo nunca llegaba a converger hacia una buena solución. Esta primera versión seleccionaba 2 individuos entre los de mejor puntuación con una probabilidad p y los enviaba para aplicar los siguientes operadores (cruce y mutación) con una cierta probabilidad. Esta versión tenia el problema de que debido a que incluso con factores de mutación bajos y con operadores de cruce poco agresivos los resultados tocaban techo lejos del objetivo, alrededor de un 0,16-0.18 de score en las mejores situaciones para acabar generando nuevas generaciones con un score inferior a la anterior ya que los individuos más aptos se diluían entre los cruces y las mutaciones. Es por esto por lo que se ha optado por aplicar una solución con **Elitismo** donde automáticamente el 25% de los individuos más aptos se transmiten a la siguiente generación.

De esta forma se puede permitir tener operadores de cruce más agresivos, así como un mayor factor de mutación ya que los mejores son salvaguardados. El 75% restante de la población se genera de la misma forma que en la versión 1 aplicando cruce y mutación sobre los individuos seleccionados.

10.5.2 Operador de Cruce

En este operador se crean recombinaciones de dos individuos, el punto de cruce se escoge de forma aleatoria dentro del cromosoma, al acabar el cruce se pasan al siguiente operador de mutación.

10.5.3 Operador de Mutación

Este operador es extremadamente importante sobre todo cuando se está aplicando la estrategia de Elitismo, si este operador no se aplicara o su factor de mutación fuera muy bajo las generaciones tenderían a una homogenización estacando la evolución de los individuos de futuras generaciones debido a que habría muy poca variabilidad genética.

Durante este operador se recorre cada uno de los genes de un individuo y se modifican con la probabilidad del factor de mutación (para este proyecto se ha usado entre 0.05 y 0.25).

10.6 Función de Aptitud (Fitness function)

La función de Aptitud evalúa que tan buena es una solución del espacio de búsqueda.

En este caso, para evaluar la calidad de un escenario hay múltiples detalles para tener en cuenta, por un lado, la parte más relevante es que el mapa sea navegable, para esto el mapa tiene que garantizar ciertos tiles que son navegables por todas las unidades del juego. Es por esto por lo que previamente se ha creado una distribución de los tiles por altura como se ha podido comprobar en la tabla del apartado 10.2. Estas alturas, permiten mantener una distribución ordenada que mediante un Heurístico podrá ser evaluado.

10.6.1 Heurísticos

Estos heurísticos formaran la función de aptitud, cada heurístico tendrá un peso que le dará mayor o menor impacto en el cálculo de la Aptitud de un escenario.

Heurístico de Distribución de alturas

Este heurístico calcula la proporción a partir de la comparación de alturas entre una celda y sus adyacentes, sí la diferencia de altura entre la celda a comparar y una adyacente es igual o inferior 1 la variable height_ratio se incrementa en 1 y se incrementa el contador en 1, si no únicamente el contador se incrementa en 1. Esto se aplica sobre todas las celdas de un escenario y al final la proporción se calcula como **score = height_ratio / contador**.

Heurístico de Distribución de biomas

Este heurístico calcula la proporción a partir de la comparación del tipo de terreno que se encuentra en la celda y el que se encuentra en las celdas adyacentes, sí el tipo de terreno es el mismo, entonces, la variable biome_ratio se incrementa en 1 y se incrementa el contador en 1, si no únicamente el contador se incrementa en 1. Esto se aplica sobre todas las celdas de un escenario y al final la proporción se calcula como **score = biome_ratio / contador**.

El objetivo de este heurístico es que se formen pequeños biomas con el mismo tipo de terreno hay que tener cuidado con el peso en la puntuación final de este heurístico a diferencia del anterior en este no se trata de converger hacía ya que esto se traduce en un mapa con un único bioma formado por un único tipo de terreno.

Heurístico de Distribución simétrica

Este heurístico calcula la proporción a partir de la comparación del tipo de terreno que se encuentra en la celda y el que se encuentra en la celda en la posición inversa (simétrica), sí el tipo de terreno es el mismo, entonces, la variable simetria_act se incrementa en 2. Esto se aplica sobre la mitad superior de las celdas ya que la mitad inferior es la región donde se debe proyectar la simetría del terreno, al final la proporción se calcula como **score = simetria_act / totalGenes**.

El objetivo de este heurístico es que se formen pequeños biomas con el mismo tipo de terreno a diferencia del anterior en este no se trata de converger hacia 1 si no hacia (0.6-0.8) ya que si el score de un mapa en este heurístico es 1 quiere decir que solo hay un tipo de terreno.

10.6.2 Representación de la función aptitud

La función de aptitud calcula el score del escenario a partir de los tres heurísticos descritos anteriormente. Y esto lo hace de la siguiente forma:

Score = sim_score * 0.4 + height_score * 0.3 + biome_score * 0.3

10.7 Resultados Algoritmo genético de generación procedural

En este apartado se va a poner a prueba el algoritmo de generación procedural de mapas a lo largo de 4 escenarios con diferentes aproximaciones de la función de aptitud. Al final se compararán resultados entre ellos para ver como rinde mejor y que heurísticos parecen más útiles.

Escenario 1, heurístico de distribución de alturas

En este escenario se va a optimizar con la función de aptitud:

Score = Height_score

O lo que es lo mismo, el score final es directamente el resultado del heurístico de distribución de alturas. El punto de convergencia que se considera para parar el algoritmo es de > 0.95 en un intervalo entre [0,1].

El algoritmo a convergido en ese valor tras 513 generaciones de 100 individuos.

El mapa de mayor Score dentro de la generación resultante es el siguiente:

Como se observa en el mapa generado, el escenario es 100% jugable, dominan los 3 tipos de pasto en el escenario junto a bosque debido a que su rango es superior al resto, por lo que por probabilidad es lo que debe ocurrir, se observan pequeñas distribuciones de altura que en las celdas adyacentes salta por una diferencia superior a 1. El mapa es bastante coherente, pero se echa en falta biomas mejor formados.

Escenario 2, heurístico de distribución de biomas

En este escenario se va a optimizar con la función de aptitud:

Score = biome_score

O lo que es lo mismo, el score final es directamente el resultado del heurístico de distribución de biomas. El punto de convergencia que se considera para parar el algoritmo es de > 0.6 en un intervalo entre [0,1], ya que como se ha indicado previamente su optimización no tiene a 1, si no al rango [0.6-0.8] ya que biome_score == 1 quiere decir que solo hay un bioma con un tipo de terreno ocupando todo el mapa y esto no es lo que se busca.

El algoritmo a convergido en ese valor tras 10107 generaciones de 100 individuos.

El mapa de mayor Score dentro de la generación resultante es el siguiente:

En este nuevo mapa se observa como predominan 4 biomas, el bioma de montaña predominante en la parte este del mapa y en la parte noroeste, el bioma de bosque al suroeste, el bioma de pantano en el centro y el bioma de pantano – pasto el noroeste.

El mapa es jugable pero poco navegable para unidades terrestres.

Escenario 3, heurístico de distribución de simetría

En este escenario se va a optimizar con la función de aptitud:

Score = sim_score

O lo que es lo mismo, el score final es directamente el resultado del heurístico de distribución de simetría. El punto de convergencia que se considera para parar el algoritmo es de > 0.7 en un intervalo entre [0,1].

El algoritmo a convergido en ese valor tras 9026 generaciones de 100 individuos.

El mapa de mayor Score dentro de la generación resultante es el siguiente:

Como se puede apreciar, el mapa contiene variedad de terrenos, pero le faltan algunos biomas y sobre todo balance en las alturas, de los 3 es visualmente el menos coherente, aunque a nivel de gameplay ofrece en un buen balance por tener el factor 0.7 de simetría.

Escenario 4, combinación final con los 3 heurísticos

En este escenario se va a optimizar con la función de aptitud:

Score = sim_score * 0.3 + biome_score * 0.4 + height_score * 0.3

O lo que es lo mismo, el score final es la suma ponderada de los diferentes heurísticos. El punto de convergencia que se considera para parar el algoritmo es de > 0.6 en un intervalo entre [0,1].

El algoritmo a convergido en ese valor tras 3145 generaciones de 100 individuos.

El mapa de mayor Score dentro de la generación resultante es el siguiente:

Es el mejor de los tres mapas obtenidos, gracias a la simetría está bastante bien balanceado a nivel de jugabilidad, se observan varios biomas formados, dando una sensación de naturaleza y la altura está bastante bien distribuida.

11. Desviación de la planificación

Finalmente, el proyecto se ha acabado desviando debido a diversos obstáculos encontrados, en la planificación original ya se estimaba que en caso de que se desviara demasiado la planificación se estudiaría la ampliación de matrícula y, finalmente, así ha sido.

11.1 Tabla de tareas actualizada

Código	Tarea	Dependencias	Perfil	Tiempo estimado (h)	Tiempo real aproximado
TG1	Alcance del proyecto		DP	20	20
TG2	Planificación temporal	TG1	DP	15	15
TG3	Presupuesto	TG2	DP	6	6
TG4	Informe de sostenibilidad	TG2	DP	6	6
TG5	Memoria		DP	50	90
TG6	Preparación de la defensa	TG5	DP	10	10
RN1	Reuniones		DP, AS, P, T	10	10
TR1	Preparación del entorno de desarrollo	TG1	AS	5	5
TR2	Recopilación de datos relevantes del videojuego	TR1	AS	20	40
TR3	Análisis del código fuente del videojuego	TITR2	AS	20	40
TI1	Diseño del Agente Inteligente	TR3	AS	20	50
TI2	Diseño del algoritmo de generación procedural	TR3	AS	18	25
TI3	Implementación del Agente Inteligente	TI1	P	140	215
TI4	Implementación del algoritmo de generación procedural	TI2	P	110	135
TI5	Entrenamiento del Agente Inteligente	TI3	P	30 (90)	60(150)
TI6	Evaluación de los mapas generados (Feedback para las nuevas generaciones)	TI4, (TI3 para ultima iteración)	P	25	25
TP1	Pruebas del Agente	TI5	T	20	50
TP2	Pruebas del algoritmo de generación procedural	TI6	T	20	20
			Total	545	827

Elaboración propia

11.2 Obstáculos y motivos de la desviación por tareas

El principal y mayor obstáculo para la finalización de este proyecto ha sido con diferencia como lograr integrar el Agente Inteligente en el entorno de Battle for Wesnoth, ya que el Agente debe interactuar con el entorno y recibir feedback por sus acciones es necesario que el Agente este directamente integrado al código del juego de esta forma es posible interactuar, recibir feedback y que el Agente sea seleccionable como una IA in-game.

En un principio se intentó implementar en C++ junto al mainloop del juego, resulto extremadamente complicado hasta el punto de que incluso los actuales desarrolladores de IA del juego no hacen modificaciones sobre el main_loop en C++ si no que directamente crean complementos en LUA aprovechando la modularidad en stages del diseño. Esta dificultad sumada a que la documentación de la parte de IA en C++ esta desactualizada y parte del código en desuso, hizo imposible encontrar la forma de conseguir interactuar ni recibir el feedback necesario en esta primera implementación en C++.

Tras estos inconvenientes se decidió alargar el proyecto para replantear el desarrollo en LUA como finalmente ha sido, no sin tampoco varios inconvenientes que han alargado aún más el proyecto.

El más importante de los problemas encontrados en el desarrollo en LUA ha sido que en Wesnoth, solo una parte de la librería estándar de LUA está disponible para su uso debido a medidas de seguridad. Por ejemplo, no se puede usar el módulo IO de LUA con lo que no es posible cargar (solo se puede cargar y ejecutar otros scripts LUA) ni escribir en ficheros y esto ha sido un problema…

El Agente necesita almacenar el conocimiento entre partidas y la única forma es guardándolo en algún archivo, ya que tampoco era posible imprimir en consola nada tampoco se podía redirigir de manera externa.

Finalmente se ha encontrado una forma para conseguir guardar el conocimiento, esta forma es imprimir con mensajes de debug el conocimiento en el log principal del juego, crear un script externo en Python que recopile únicamente la información de conocimiento del log y que este mismo script construya un script en LUA cuya función sea retornar el conocimiento restableciendo su estructura.

Este ha sido el principal y más duro obstáculo que superar, pero ha habido otros como: limitación en que información del game_state se puede obtener desde LUA, la imposibilidad de iniciar partidas si en el equipo1 no hay comandante y otros pequeños obstáculos más sencillos de superar.

En lo que respecta al algoritmo de generación procedural ha sido mucho más sencillo gracias a que se ha podido implementar de manera totalmente externa al código del juego por lo que no había nada de lo que depender de terceros. El único “problema” que ha surgido durante la implementación, fue cuando la función de fitness no convergía hacia los valores esperados, pero esto como se ha comentado en un apartado anterior se ha solventado aplicando la técnica del elitismo en la selección de individuos.

11.3 Coste Final del Proyecto Tabla

Código	Tarea	Director de Proyecto	Analista de Sistemas	Programador	Tester	Coste
TG1	Alcance del proyecto	20 h				391,40 €
TG2	Planificación temporal	15 h				293,55 €
TG3	Presupuesto	6 h				117,42 €
TG4	Informe de sostenibilidad	6 h				117,42 €
TG5	Memoria	90 h				978,50 €
TG6	Preparación de la defensa	10 h				195,70 €
RN1	Reuniones	10 h	10 h	10 h	10 h	572,30 €
TR1	Preparación del entorno de desarrollo		5 h			75,80 €
TR2	Recopilación de datos relevantes del videojuego		40 h			606,40 €
TR3	Análisis del código fuente del videojuego		40 h			606,40 €
TI1	Diseño del Agente Inteligente		50 h			758 €
TI2	Diseño del algoritmo de generación procedural		25 h			379 €
TI3	Implementación del Agente Inteligente			215 h		2.687,50 €
TI4	Implementación del algoritmo de generación procedural			135 h		1.687,50 €
TI5	Entrenamiento del Agente Inteligente			60 h		750 €
TI6	Evaluación de los mapas generados (Feedback para las nuevas generaciones)			25 h		312,50 €
TP1	Pruebas del Agente				50 h	500 €
TP2	Pruebas del algoritmo de generación procedural				20 h	200 €
	Sueldos					**11.229,34 €**
	Seguridad Social del personal (30%)					**3.368,80 €**
	Coste total de Personal (Sueldos + SS)	117 h	170 h	445 h	80 h	**14.598,15 €**

Asignación de tareas y costes actualizada (Elaboración propia)

12. Integración de conocimientos

Este es un proyecto dentro de la especialidad de Computación más concretamente se ha utilizado el conocimiento adquirido de las siguientes asignaturas del grado de ingeniería informática:

PRO1: Esta asignatura ha aportado las bases en programación usadas para este proyecto que posteriormente se han visto ampliadas en siguientes asignaturas.

PRO2: Introducción a estructuras de datos lineales y programación orientada a objetos con sus clases de datos. Diseño modular.

M1: Conceptos de grafos: recorridos, conexión y distancia en grafos. Conocer y saber aplicar algoritmos para determinar la conectividad y para calcular distancias en grafos.

EDA: Se ha usado el conocimiento adquirido en esta asignatura para resolver un problema con restricciones de tiempo y espacio, diseñar, analizar, comparar e implementar los algoritmos disponibles que ofrecían una solución al problema. También diseñar, analizar, comparar e implementar diferentes estructuras de datos para el manejo de la información del agente y del algoritmo de generación procedural, completar y modificar implementaciones en C++ y ejecutar estrategias de búsqueda de información.

PE: La existencia de aleatoriedad para los resultados de una misma acción hace que haya que estudiar el impacto que tiene el RNG sobre el comportamiento del Agente debido a la función de rewards.

PROP: Al ser también un proyecto de programación tiene múltiples similitudes al TFG, no solo en su estructuración sino en algunas fases del desarrollo, por ejemplo, en el program testing para la prueba y depuración de programas, diseño de una planificación y la introducción al uso de herramientas para el desarrollo de programas. También han sido útiles los conocimientos de la capa de persistencia de datos.

A: Conceptos algorítmicos básicos e introducción a los algoritmos greedys.

IA (máxima relevancia en este proyecto): Resolución de problemas mediante búsqueda: Representación como espacio de estados, algoritmos genéticos, Representación de conocimiento y razonamiento, Aprendizaje Automático.

LP: Introducción a los lenguajes funcionales (HASKELL) y de scripting (PYTHON), en este caso se han utilizado conceptos aprendidos sobre los lenguajes funcionales para aprender el Wesnoth Formula Language y LUA (aunque este último es imperativo), también se ha utilizado Python en la implementación del algoritmo genético de generación procedural y.

VJ: Se ha utilizado el conocimiento del gameloop, que ha servido para extraer información relevante y para llevar a cabo la interacción del agente con el entorno sin disrumpir la ejecución normal del mismo.

APA (máxima relevancia en este proyecto): Aunque no se ha cursado esta asignatura, los conocimientos adquiridos de su material han sido muy relevantes, ampliando lo visto en IA de aprendizaje automático hacia el aprendizaje por refuerzo y más concretamente al Q-learning, también ha servido para comparar los diferentes tipos de aprendizaje automático.

SID: Conocer el concepto de agente inteligente.

13. Identificación de leyes y regulaciones

En este proyecto se ha trabajado sobre un videojuego Open Source bajo licencia GPL v2, esto implica que cualquiera puede usar, compartir, estudiar o modificar este proyecto, y cuyo objetivo es protegerlo de intentos de apropiación que restrinjan esas libertades a nuevos usuarios cada vez que la obra sea distribuida, ampliada o modificada.

14. Conclusiones

En lo que respecta el desarrollo del Agente Inteligente, parece que en situaciones concretas donde el modelo elegido es muy relevante, tiene la ventaja sobre la IA por defecto que es una buena IA tradicional con 18 años de mejoras y ampliaciones, el hecho de poder competir frente a esta IA e igualar o ganarle con mayor o menor solvencia en algunos escenarios en concreto, donde no se tiene en cuenta el control de aldeas y el reclutamiento es bastante satisfactorio, y más, teniendo en cuenta que por los motivos contados en el apartado 11.2 no se ha podido desarrollar al nivel que me hubiese gustado ni tampoco entrenar hasta alcanzar un punto óptimo donde poder explotar eficientemente el conocimiento. Por poner un ejemplo, para lanzar 100 partidas del escenario 4 y recoger el conocimiento de los logs con el script externo se tardaban más de 8 horas. Aunque, ha sido algo decepcionante no poder ver todo el potencial del Agente tras miles de partidas de entrenamiento, quizás así, el resultado en el Escenario 3 podría haber sido distinto.

En lo que respecta al algoritmo de generación procedural evolutiva, ha sido un camino más placentero ya que no se dependía del código del juego, en este caso el desarrollo ha sido fructífero y los mapas generados por los heurísticos escogidos son coherentes y balanceados hasta cierto punto.

Tras este proyecto se ve el potencial que tiene el aprendizaje automático y más concretamente el aprendizaje por refuerzo al programar comportamientos en los videojuegos, superando incluso comportamientos más tradicionales, aunque como se ha visto el "problema" es que deben ser entrenados lo suficiente como para empezar a ver resultados, además encontrar un buen modelo de codificación de los estados puede ser complicado.

También hemos visto el potencial de los algoritmos genéticos, permitiendo automatizar la creación de contenido con bastante simplicidad, por ejemplo, este algoritmo es capaz de crear cientos de mapas por segundo de una calidad y equilibrio buena (una vez optimizado el espacio de soluciones), hacer un trabajo con algoritmos tradicionales no ofrecería tal variedad de mapas distintos y hacerlos a mano es ineficiente.

14.1 Opinión personal

Trabajar en este proyecto ha sido una experiencia satisfactoria, pese a los obstáculos encontrados. El videojuego The Battle for Wesnoth es un gran juego dentro del género de la estrategia por turnos y poder crear un algoritmo que aprenda a jugarlo, así como otro algoritmo que sea capaz de crear escenarios jugables es una gran experiencia para cualquier tanto de los videojuegos como de la IA.

Ya que no había llegado a matricular la asignatura de APA y me parecía un campo dentro de la IA muy interesante, decidí juntar ese interés con la afición a los videojuegos.

Antes de este proyecto, no había llegado a trabajar con técnicas de aprendizaje automático y muy poco con algoritmos genéticos.

Me ha servido para ver cómo se aplican estas técnicas, para en un futuro poder aplicarlas ya sea de nuevo en el campo de los videojuegos o fuera de este.

Sin lugar a dudas contento con la elección del tema, aunque quizás de haber sabido las limitaciones que me iba a encontrar para implementarlo en The Battle for Wesnoth, hubiera escogido otro juego del mismo género o similar.

15. Trabajo Futuro

- Encontrar alguna alternativa para poder entrenar al Agente de forma más eficiente.
- Una ampliación que además serviría como solución a lo anterior, seria experimentar con Deep Reinforcement Learning, para ello se necesitaría una base de datos de partidas guardadas, pero si se consigue no solo sería una mejora, sino que además facilitaría el aprendizaje.
- Tratar de conseguir una solución del Agente Standalone, esto acabaría con todas las restricciones y problemas.
- Ampliar la codificación de estados para incluir no solo acciones de combate sino también de desplazamiento.
- Añadir también reclutamiento a las acciones del Agente.
- Nuevos heurísticos para la generación procedural en colaboración con algún jugador profesional que pueda detectar desbalanceo en los escenarios generados.

15. Bibliografía

[1] Analytics Insight, The evolution of artificial intelligence in video games. URL: https://www.analyticsinsight.net/the-evolution-of-artificial-intelligence-in-video-games

[2] Theverge, How artificial intelligence will revolutionize the way video games are developed and played URL: https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning

[3] Sitn.hms.Harvard, AI in video games: Toward a More Intelligent Game. URL: https://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/

[4] Davidepesce, Procedural generation in Game Development. URL: https://www.davidepesce.com/2020/02/24/procedural-generation-in-game-development/

[5] Hipertextual, No Man's Sky. URL: https://hipertextual.com/2016/08/no-mans-sky

[6] APA, Aprendizaje Automático – FIB, Introduction to Machine Learning. URL: https://www.cs.upc.edu/~bejar/apa/teoria/APA1a-IntroML.pdf

[7] APA, Aprendizaje Automático – FIB, Reinforcement Learning. URL: https://www.cs.upc.edu/~bejar/apa/teoria/APA10a-Reinforcement.pdf

[8] APP-DIAGRAM, Flowchart Maker & Online Diagram Software. URL: https://app.diagrams.net/

[9] ScienceDirect, Evolutionary Algorithms. URL: https://www.sciencedirect.com/topics/computer-science/evolutionary-algorithms

[10] Lua documentación oficial, URL: https://www.lua.org/

[11] The Battle for Wesnoth, Game page. URL: https://www.wesnoth.org/

[12] TheSharperDev, Supervised v. Unsupervised v. Reinforcement Learning. URL: https://thesharperdev.com/supervised-v-unsupervised-v-reinforcement-learning-an-introduction/

[13] Techvidvan, Reinforcement Learning Algorithms and Applications. URL: https://techvidvan.com/tutorials/reinforcement-learning/

[14] The Battle for Wesnoth, AI based on Deep Learning. URL: https://forums.wesnoth.org/viewtopic.php?t=44006&p=621135

[15] Universidad de Sevilla, Aprendizaje por refuerzo: algoritmo Q-Learning. URL: http://www.cs.us.es/~fsancho/?e=109

[16] Towardsdatascience, Reinforcement learning: Temporal-Difference, SARSA, Q-Learning & Expected SARSA. URL: https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e

[17] Stackexchange, When to chose SARSA vs Q-Learning. URL: https://stats.stackexchange.com/questions/326788/when-to-choose-sarsa-vs-q-learning

[18] Red hat, ¿Qué es la metodología ágil?. URL: https://www.redhat.com/es/devops/what-is-agile-methodology

[19] Cognodata, 12 principios de la metodología Agile en el desarrollo de proyectos. URL: https://www.cognodata.com/blog/principios-metodologia-agile-desarrollo-proyectos/