Universitat Politècnica de Catalunya - BarcelonaTech

Facultat d'Informàtica de Barcelona

Trabajo de Final de Grado

Desarrollo de una Plataforma de IA Evolutiva para la Supervivencia

Autor: Guillermo Serrahima
Grado en Ingeniería Informática, Especialidad de Computación

Fecha de defensa: 2 de Julio de 2018

Director:
Javier Béjar,
Departamento de Ciencias de la Computación

Junio 2018

Resum

Aquest treball de fi de grau consisteix en la creació d'una plataforma de desenvolupament d'intel · ligència artificial evolutiva i adaptable a un entorn variable. Aquesta plataforma ha sigut desenvolupada com una modificació del joc Minetest, un joc de codi obert programat en C++ amb generació de terrenys procedurals en 3 dimensions, el qual les intel · ligències artificials hauràn d'aprendre a navegar.

La finalitat d'aquest projecte té 2 vessants: per un costat es busca consolidar diferents coneixements vists i apresos al llarg del grau d'informàtica, i per l'altre es busca profunditzar en matèria i fer toma de contacte, estudiar i posar en pràctica algunes de les tècniques d'intel · ligència artificial populars actualment.

En aquesta memòria s'explica el context actual d'aquestes tecnologies, així com les decisions de disseny preses pel desenvolupament de la plataforma de IA, i el seu funcionament i ús per a la creació de futures IAs sobre la plataforma.

Resumen

Este trabajo de final de grado consiste en la creación de una plataforma de desarrollo de inteligencia artificial evolutiva y adaptable a un entorno variable. Esta plataforma ha sido desarrollada como una modificación del juego de Minetest, un juego de código abierto programado en C++ con generación de terrenos procedural en 3 dimensiones, el cual las inteligencias artificiales deberán aprender a navegar.

La finalidad de este proyecto tiene dos vertientes: por un lado se busca consolidar varios conocimientos vistos y aprendidos en el grado de ingeniería informática, y por el otro se busca profundizar en materia y tomar contacto, estudiar y poner en práctica algunas técnicas de inteligencia artificial que son populares actualmente.

En esta memoria se explica el contexto actual de dichas tecnologías, así como las decisiones de diseño tomadas para el desarollo de la plataforma de IA, y su funcionamiento y uso para el desarrollo de futuras IAs sobre ésta.

Abstract

This final degree project consists in the creation of a development platform for artificial intelligence capable of evolution and adaptability to a variable environment. This platform has been developed as a modification of the Minetest game, an open source C++ game with procedural generation of terrain in 3 dimensions, which the artificial intelligence must learn to navigate.

The goal of this project is two-fold: on one side, it serves as a way to strengthen a variety of knowledge acquired and studied during the degree; on the other, it serves as a way to deepen certain knowledge and have contact with, study, and put to practice some of the current popular artificial intelligence techniques.

In this project report the current context of said techniques is explained, as well as the design decisions taken along the way for the development of the AI platform, in addition to its features and usage for the development of future AIs on top of it.

Índice

1 Introducción

1.1 Introducción y contextualización

1.1.1 Formulación del problema

El uso de inteligencias artificiales (o IAs) es prevalente desde casi el inicio del desarrollo de videojuegos, tanto como oponentes del jugador, en sustitución a posibles oponentes reales (como en los primeros videojuegos de ajedrez y otros juegos de mesa clásicos clásicos competitivos) como elementos del entorno (personajes no jugables con los que poder hacer interacciones limitadas) hasta aliados o compañeros de aventura.

Por otro lado, género de los juegos de supervivencia, popularizado en los útimos años, ofrece un gran abanico de posibilidades al jugador en un entorno de mundo abierto, sirviendo de "caja de arena" para dar rienda suelta a la creatividad de uno.

El problema, por eso, es que la inteligencia artificial que uno puede encontrar en dichos juegos es casi inexistente, contando como mucho con animales con deambulación aleatoria o criaturas hostiles con tomas de decisiones muy sencillas únicamente para contrarrestar una sensación de mundo despoblado; o bien con personajes no jugables (por el jugador, también llamados NPCs) de interacciones guionizadas fijas, lo que significa para el jugador una experiencia artificial, mecánica, con una interactividad muy limitada, que rompe la suspensión de incredulidad y la inmersión.

Por lo tanto, el propósito de este proyecto surge de la posibilidad de ofrecer una experiencia de inteligencia artificial menos fija y más orgánica, no tan necesariamente diseñada en torno a la existencia del jugador y teniendo en cuenta el entorno que la rodea.

Al mismo tiempo, en el contexto de proyecto de un estudiante de universidad, se busca consolidar los conocimientos adquiridos aplicándolos en un contexto visible y palpable, así como profundizar en los campos de la informática que me atraen y hacia los que me quiero encaminar.

1.1.2 Objetivo del proyecto

En concreto, el propósito general definido se puede dividir en varias secciones, de la siguiente manera:

1. En el contexto de proyecto de fin de grado, se busca consolidar los diversos conocimientos aprendidos a lo largo de la carrera en una sola aplicación palpable, así como ahondar en el campo de la informática que más me atrae y conocer el estado actual de dicho campo.

2. Se busca un primer contacto los diversos sistemas populares y extendidos actualmente para la adaptación de inteligencias artificiales a las condiciones que

las rodean: es decir, aprendizaje automático, así como el estudio y comparación de las utilidades de cada uno para los diversos tipos de proyectos, y ver cuál puede ser más revelante al proyecto actual.

3. Finalmente, se busca desarrollar un sistema expandible de desarrollo de IA, que permita desarrollar en un futuro inteligencia artifical adaptable siguiendo los diseños relevantes visto en el punto anterior, asi como un prototipo sencillo de inteligencia que utilice dicho sistema o plataforma.

1.1.3 Actores implicados

Este proyecto es relevante a varios actores determinados, entre los cuáles se incluyen:

1. **Diseñador y desarrollador:** Las tareas de diseño y desarrollo inicial recaen sobre mí ya que el proyecto lo realizo solo.

2. **Comunidad de desarrolladores:** Los desarrolladores de complementos para el juego sobre el que se va a realizar este proyecto pueden estar interesados ya que pueden tener ideas o proyectos que se puedan construir sobre éste.

3. **Comunidad de jugadores:** Los usuarios del videojuego sobre el que se va a programar esta IA pueden beneficiarse de los resultados de este proyecto (o de otros elaborados a partir de éste) ya que puede enriquecer la experiencia de dicho juego.

4. **Desarrolladores de IA:** Otros desarrolladores de inteligencias artificiales también pueden beneficiarse de este proyecto, ya que puede servir como otro ejemplo práctico de aplicaciones de técnicas de IA en videojuegos, así como una plataforma que facilite la entrada al desarrollo, sobre todo a nuevas generaciones que puedan adquirir interés en la programación con Minetest como su primer contacto.

1.2 Contexto

1.2.1 Inteligencia artificial

Estructura

Una inteligencia artificial es traicionalmente un programa de toma de decisiones; en un videojuego, es costumbre utilizarlas para controlar elementos que ajenos al jugador 11. En el desarrollo de inteligencias artificiales hay varias técnicas que utilizar para definir la toma de decisiones por un lado, y la optimización de dicha toma de decisiones por el otro. Específicamente, las técnicas de toma de decisiones pueden llevarse a cabo con:

- Toma de decisiones por **máquina de estados**: cada estado tiene un comportamiento (o conjunto de comportamientos) predefinido, y una serie de comprovaciones y eventos es la que se encarga de saltar de un estado a otro.

- Toma de decisiones por **árbol de decisiones**: se construye y recorre el árbol buscando, para cada caso, la rama con la acción más beneficiosa para la IA.
- Toma de decisiones por **función de utilidad**: cada acción posible a realizar se valora mediante una *función de utilidad* que permite asignar una puntuación al resultado, y luego se elige el conjunto de acciones que dan el mejor resultado.

La toma de decisiones puede ser estática, es decir, puede realizarse siempre dentro de las mismas condiciones; o puede llevarse a cabo a través de mejoras u optimizaciones de forma dinámica, cambiando los valores de tomas de decisiones según lo que aparenta ser más beneficioso conforme se van probando las acciones. Esto va bien en sistemas donde la información que se tiene es incompleta, o se trata de un escenario variable que no siempre exactamente se encuentra con las mismas condiciones, o no se puede valorar directamente el beneficio de una acción en un paso en concreto.

Aprendizaje automático

De IAs dinámicas existen varios tipos:

- **Algoritmos de evolución:** Estos algoritmos incorporan una serie de valores numéricos que controlan la toma de decisiones (por ejemplo pesos en la función de utilidad o en la valoración de la mejor rama en un árbol) que pueden alterarse, y se genera una población de IAs con el mismo algoritmo pero distintas características, que luego se poda seleccionando un grupo de las características con mejores resultados, y luego se genera una nueva generación de IAs para la población copiando una de las IAs "supervivientes" y modificando algunas de las características (mutaciones) o recombinando características a partir de otras 2 IAs supervivientes (reproducción).
- **Aprendizaje supervisado:** la filosofía general es utilizar un algoritmo de decisiones aleatorias, comparar el resultado obtenido con el resultado deseado, e introducir dinámicamente modificaciones al algoritmo para reducir el error producido.
- **Aprendizaje por refuerzo:** en este tipo de aprendizaje, al algoritmo se le proporciona una valoración o puntuación según lo apropiado que es el resultado, y conforme se van realizando ejecuciones, el algoritmo tiende a aquellas combinaciones de acciones que han producido las mejores valoraciones. Esta valoración, cuando busca maximizarse o resulta positiva para el algoritmo, también se la conoce como "premio"; y cuando resulta negativa, como "castigo".
- **Aprendizaje no supervisado:** Al contrario que en el supervisado, no se conoce el resultado deseado desde un punto inicial, por lo que el objetivo del algoritmo es encontrar patrones que aproximen un mejor resultado.

Representación de la información

Hay varias formas también de representar los datos y las acciones a realizar; por ejemplo, tablas con las puntuaciones de los resultados (evolución y aprendizaje reforzado); **redes neuronales**, que son una combinación de elementos de computación pequeños (neuronas) conectados entre sí (sinapsis) que analizan una parte de la entrada, y dan una salida que, combinada con las de otras neuronas, permite aproximar una función de toma de decisiones (aprendizaje supervisado y no supervisado); ...

Estas técnicas se pueden combinar todas entre sí (evolución de árboles de decisiones; neuroevolución, utilizando mecanismos de evolución en vez de corrección de errores para modificar la red neuronal, ...). Sin embargo, no conocemos desde un punto inicial qué es o no un resultado óptimo, así que no se pueden aplicar técnicas de aprendizaje supervisado; el único criterio fijo para valorar la eficacia es "sobrevivir lo máximo posible". En concreto, las técnicas que más relevantes resultan al proyecto son:

- **Técnicas de aprendizaje reforzado:** específicamente, la técnica conocida como *Q-Learning*, donde se define una función Q (del inglés *"quality"*), que calcula una puntuación para cada combinación entre estado posible de la IA y acción posible a tomar en dicho estado.
- **Técnicas de evolución de algoritmos:** tanto en una simulación naturalística de la población (reproducción entre los NPCs controlados por la IA) como de forma artificial (comprovación periódica de las IAs de la población, y sustitución de la población por una generación más eficiente - sobre todo si la reproducción "natural" no permite avance suficientemente rápido de las IAs).
- **Estructuras de redes neuronales:** ya que el entorno de este proyecto es uno complejo, con un conjunto de acciones posibles muy amplio (del que probablemente la IA sólo trate una parte), y permite una aproximación más abstracta de la toma de decisiones.

1.2.2 IAs en videojuegos y otras inspiraciones

Las aplicaciones de inteligencias artificiales en los videojuegos son muy variadas 3: desde control de personajes no jugables por el jugador (conocidos como NPCs por sus siglas en inglés: *"Non-Playable Character"*), al control de oponentes artificiales para dar una experiencia de juego desafiante pero equilibrada; pasando por inteligencias que en vez de controlar un elemento concreto de juego controlan el tipo y la asiduidad de eventos que puedan ocurrir alrededor del jugador, para influenciar la experiencia de juego (también llamado *gameplay*) en una dirección u otra.

Para este proyecto son relevantes las IAs desarrolladas con el objetivo de incorporar aprendizaje del entorno, y las técnicas utilizadas, para poder permitir la incorporación de técnicas similares en la plataforma. En cuanto a esto, destacan los siguientes juegos:

Black & White

Black & White (2001, Lionhead Studios) es un juego de estrategia en el que el jugador toma el papel de un Dios patrón de un poblado ancestral, y debe guiarlo a través de varios escenarios. Bajo el control indirecto del jugador se encuentra una criatura mitológica controlada por una IA que puede ser educada para preferenciar unas acciones sobre otras, mediante refuerzo positivo (llevado a cabo con el jugador acariciando a la criatura después de una acción que se quiere premiar) o negativo (azotando la criatura después de una acción que se quiere evitar repetir).

Aquí el jugador de forma palpable es el que proporciona la función de premio del aprendizaje por refuerzo. En el proyecto dicho refuerzo vendrá a través del propio entorno, pero es un modelo que se puede transportar fácilmente al escenario de este trabajo.

Figure 1: *Criatura* tigre con los miembros del pueblo antiguo de fondo

Figure 2: *Criatura* tortuga entreteniendo a la población

Creatures

En *Creatures* (1996, Millenium Interactive) 2, el jugador es encargado de cuidar de una criatura llamada "Norn", para superar una serie de puzzles a través de varios niveles.

Los Norns tienen una serie de necesidades definidas, como comer; y tienen varias fases de vida en las que cambia su metabolismo. De la misma forma, pueden morir, y son reemplazados por un descendiente del Norn anterior.

En el "cerebro" de los Norn hay varios elementos que influyen en la toma de decisiones:

1. La **percepción** de la criatura de lo que le rodea.
2. Las **características heredadas** del antecesor.
3. Las habilidades que **aprende** conforme el juego avanza.

Este cerebro se compone de varios ”lóbulos” formados por redes neuronales interconectadas para la evolución de la inteligencia.

El desarrollo del juego resaltó el hecho que las redes neuronales eran demasiado complejas para ser entrenadas de 0, así que se realizaba un entrenamiento parcial previo al lanzamiento del juego y luego se permite que las criaturas evolucionen durante el juego a partir de ese entrenamiento parcial base.

Así mismo, el cerebro está conectado a un ”cuerpo” complejo, que también puede verse afectado por el entorno, como pueden ser **toxinas** o **bacterias** introducidas en la ingesta de alimentos, así como una lógica dependiente de la edad del Norn (dividida en 3 etapas: **juventud**, **adultez** y, si sobrevive el tiempo suficiente, **senescencia**, para incrementar la sensación de paso del tiempo sobre el organismo).

La combinación de ambos elementos (cuerpo y mente) da lugar a comportamientos emergentes considerablemente complejos. Es un dato interesante a tener en cuenta.

En conjunto, la IA de un Norn incorpora redes neuronales con técnicas de aprendizaje por evolución y mutabilidad, así como aprendizaje reforzado por el jugador. De todas formas, el juego identifica genes clave para el comportamiento de un Norn y los bloquea para evitar que las mutaciones puedan provocar un retroceso considerable en la inteligencia de la criatura.

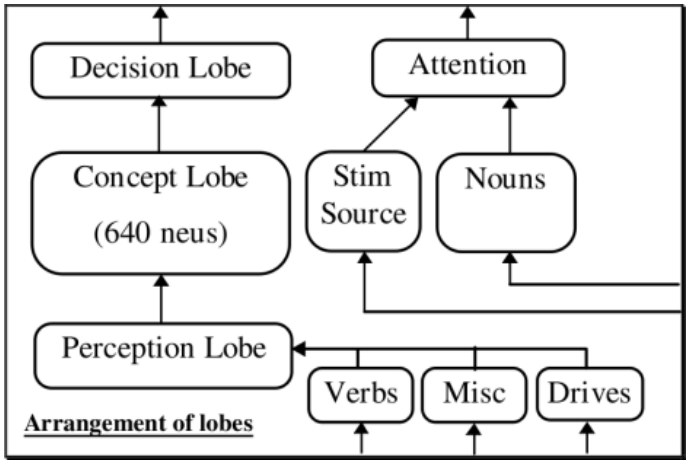


Figure 3: Diagrama del funcionamiento del cerebro de un Norn de *Creatures 2*

Los Sims

La saga de *Los Sims* (primera entrega en 2000, Maxis) es una saga popular en la que el jugador controla una unidad familiar de "Sims" (personas humanoides), con una serie de necesidades básicas que el jugador debe cubrir que, dependiendo de la independencia configurada para los Sims, puede influir en tomas de decisiones por parte del propio Sim; así como (en posteriores entregas) deseos y miedos que el jugador debe cumplir o evitar para una vida feliz del Sim; así como una simulación de relaciones interpersonales entre Sims que les permiten crear amistades y enemistades con otros Sims.

En este caso, la independencia de acción de la IA está enormemente limitada y sólo es apreciable a través de cambiar la configuración del juego, pero puede servir de inspiracón sobre elementos que influyan en el algoritmo de adaptación más allá del entorno físico (también como *Creatures*).

Figure 4: Captura de la primera entrega, con el panel de necesidades a la vista

1.2.3 Videojuegos de supervivencia

En los últimos años, el mercado de los videojuegos ha sido testigo del surgimiento y popularización de los géneros de:

- ***Sandbox***, caracterizado por no tener un objetivo claro de "victoria", y cuya atracción principal es el permitir la libertar creativa del jugador;
- y **supervivencia**, en la que el jugador se pone en contra de los elementos y cuyo objetivo principal es evitar la muerte, ya sea cumpliendo una serie de necesidades periódicamente (controlar el hambre o la temperatura corporal), recolectando materiales para construcción de un refugio o de herramientas, y cazando o recolectando alimentos de lo que proporciona el entorno.

El juego pionero en la popularización de dichos géneros es **Minecraft** 7, un juego en un mundo 3D con un terreno representado con vóxeles que empezó puramente

como un juego *sandbox* y que fue introduciendo paulatinamente diversos elementos de supervivencia. Otros juegos populares del género son **Terraria** 9, con un mundo 2D de vista frontal pero que incorpora elementos de RPGs; **Don't Starve** 5, con un mundo 2D de vista cenital que incorpora un objetivo final de juego; y otros varios, menos relevantes para el proyecto.

1.3 Sobre este documento

1.3.1 Estructura

Este documento se divide en 4 grandes secciones, que se resumen aquí para una presentación rápida del contenido de cada parte, e identificar las partes que más pueden interesar al lector:

1. **Introducción:** la sección a la que pertenece esto, con información sobre el estado actual de las tecnologías, y productos que utilizan técnicas similares o relevantes a este proyecto; así como la formulación del problema y objetivo del proyecto, y el contexto en el mercado actual de éste.

2. **Entorno y planificación:** repaso general del entorno del proyecto, el alcance, y la metodología a seguir, así como un repaso de los entornos de juego y de desarrollo, y la presentación de unas primeras ideas a tener en cuenta de cara al desarrollo de la plataforma de inteligencia artificial.

3. **Desarrollo:** explicación de las decisiones de diseño tomadas a lo largo del desarrollo, así como el funcionamiento de la plataforma y del prototipo de IA creado como ejemplo de uso de la plataforma.

4. **Gestión del proyecto:** análisis técnico del calendario de planificación, diagrama de Gantt, los recursos del proyecto disponibles, los costes económicos y de sostenibilidad que implica el proyecto, así como una sección del seguimiento final y correcciones hechas a los planes originales.

1.3.2 Vocabulario

Hay una serie de vocabulario específico que se utilizará a menudo a lo largo de este documento, por lo que se presenta aquí su significado para aclarar posibles confusiones, así como palabras que no se utilizarán tanto pero cuyo significado es relevante para las definiciones de las primeras; así como la forma en que estas palabras se utilizan en concreto en relación a Minetest, el juego y el entorno del desarrollo alrededor del cual este proyecto se ha desarrollado:

Desarrollo de software general

- **IA:** Inteligencia artificial; una pieza de software que controla un elemento independiente y ajeno al usuario (o jugador) de forma autónoma, con la posibilidad de tener en cuenta cambios en el entorno o acciones del usuario.

- **Procedural:** se refiere al sistema pseudoaleatorio de generar contenido dinámicamente de forma que permita variabilidad entre ejecuciones. En los videojuegos (como en Minetest) se suele utilizar para la generación de mapas, de forma que sean navegables pero novedosos a cada partida nueva, y así mantener fresca la experiencia de exploración del mundo de juego.
- **Pathfinding:** cualquier mecanismo utilizado para encontrar un camino entre 2 puntos. Se utiliza ampliamente en el mundo de los videojuegos para la navegación de las IAs a través del espacio.
- **Algoritmo evolutivo:** algoritmo con parámetros modulares y modificables dinámicamente durante la propia ejecución, con el objetivo de buscar una combinación de parámetros que dé buen resultado para la resolución de un problema.
- **Plug-in:** complemento de un programa que se instala sobre el original, con el objetivo de ampliar las funcionalidades que el programa original ofrece.

Desarrollo de videojuegos

- **API:** conjunto de funciones predefinidas y herramientas ofrecidas para el desarrollo sobre una plataforma ya existente. La API de Minetest ofrece una serie de funciones para modificar el entorno de juego y para abstraer y facilitar la ejecución de las tareas sencillas asociadas al control de dichas modificaciones.
- **Gameloop:** bucle de código que se ejecuta contínuamente para la correcta ejecución del juego.
- **Callback:** función diseñada y programada después de la creación del *gameloop*, que luego el propio *gameloop* recibe como parámetro, y se encarga de llamarla a cada iteración en el momento adecuado.
- **(Game) step:** Una iteración única del *gameloop*; también se utiliza para referirse a cada una de las *callbacks* que se debe ejecutar una vez por *step* (en la API de Minetest, las *callbacks* generales para cada step se llaman `on_step(...)`). A veces también son referidas aquí como turnos.
- **Tile:** del inglés "baldosa", se refiere a la división del espacio 2D de un juego en cuadrados, llamados *tiles*, con el objetivo de facilitar tanto la parte del diseño de navegación a través del espacio de juego, así como la representación visual y gráfica.
- **Vóxel:** unidad espacial de tamaño arbitrario que se utiliza para dividir el espacio tridimensional contínuo de un videojuego en un espacio discreto; se suele utilizar como el equivalente 3D tanto a los píxeles en 2D como a las *tiles* en 2D.

Experiencia de juego

- **Mod:** complemento o *plug-in* de juego que expande la funcionalidad del juego original; un juego que permite la creación e instalación de *mods* suele describirse como *"modeable"*. Los *mods* de Minetest, desarrollados por una

comunidad de fans, añaden funcionalidades no existentes en el juego como mecánicas de hambre o seres vivos.

- **Modpack:** un complemento de juego que expande la funcionalidad considerablemente, y que se implementa como un conjunto de *mods*, de forma que se puedan activar o desactivar las características de forma personalizada en base a los *mods* que lo componen.
- **"Vanilla":** expresión inglesa para referirse al estado en el que se encuentra un juego *modeable* al que no se le ha instalado ningún *mod*.
- **Mob:** Forma alternativa de referirse a criaturas del entorno del juego, cuya función suele ser la de recurso, obstáculo o enemigo para el jugador.
- **Spawnpoint:** Punto del espacio en un juego configurado como las coordenadas de aparición de nuevos jugadores, o de jugadores que han muerto en el juego. En Minetest, el **spawnpoint** por defecto es el punto aleatorio (cerca de las coordenadas (0,0,0)) en el que se aparece al crear y entrar en un mundo por primera vez, y hasta que el jugador duerme en una cama, momento en el cual las coordenadas de la cama se establecen como nuevo *spawnpoint*.

2 Entorno y planificación

2.1 Descripción del proyecto

Como hemos visto arriba, el propósito de este proyecto surge del deseo por una experiencia de inteligencia artificial más orgánica, que muestre una capacidad de adaptación al entorno, y la idea de crear un entorno donde ofrecerla sea posible. Esto se va a llevar a cabo desarrollando una plataforma de desarrollo de inteligencia artificial, y un prototipo de IA creado sobre ese sistema, capaz de adaptarse al entorno, diseñado tras comparar diversas técnicas y seleccionando lo que se considere más adecuado al trabajo.

Concretamente, se ha elegido crear dicha plataforma sobre un entorno ya existente: el videojuego **Minetest** 10. Se trata de un juego modelado con los objetivos iniciales y las características de diseño de **Minecraft**, pero con una filosofía de código abierto, y un sistema de desarrollo de componentes (o *"plug-ins"* en inglés) programables en Lua fácilmente accesible a desarrolladores experimentados y aficionados por igual.

2.1.1 Entorno de juego

Minetest es un juego *sandbox* tridimensional escrito en C++, con el espacio dividido en vóxeles, hy generación de mundos procedural, donde el jugador tiene libertad de explorar, construir y excavar el entorno a su antojo.

El proyecto original se plantea como un entorno de experimentación más que un juego en sí, cuyo objetivo era emular el estilo de diseño de Minecraft, un juego con características similares escrito en Java que alcanzó una enorme popularidad en los últimos años. Por ello, Minetest carece de varias características que Minecraft tiene:

más notablemente, la existencia de criaturas que pueblen el mundo, variedad de herramientas que el jugador puede construir a partir de los elementos encontrados en el entorno, así como la posibilidad de viajar a otros mundos con unos entornos específicos distintos al mundo inicial. Sin embargo, al tratarse de un proyecto de código abierto ejecutable de forma gratuita atrajo una considerable popularidad, por lo que se implementó un sistema de modificaciones o *mods* que se podían instalar sobre Minetest, programables por la comunidad de fans de forma ajena al creador del proyecto original. De esta forma, los jugadores que así lo deseen pueden recrear las características de Minecraft que puedan faltar en el Minetest *vanilla*. Este sistema, además, permite escoger al jugador de forma individual la complejidad que desea de su experiencia de juego personal.

Como una parte del entorno más, se ha decidido referirse a la población de IAs con nombres de criaturas de folklore, ya que no generan las expectativas de inteligencia que podrían venir asociadas al nombre de alguna especie animal, o a la presentación de las IAs como NPC humanos.

2.1.2 Alcance del proyecto

Hay 3 objetivos claros del proyecto a alcanzar en orden:

1. Diseñar un modelo de representación del entorno, para poder abstraer la interacción de la IA con el entorno.

2. Elaborar un mod de Minetest con una estructura que permita la abstracción y facilite del desarrollo de una IA.

3. Integrar un prototipo de ejemplo sobre la plataforma, como ejemplo y prueba de uso.

Otro elemento clave del desarrollo será probar y comparar las distintas técnicas de aprendizaje, watudiar si es más eficaz o potente Q-Learning enfrente a algoritmos genéticos 8, o si por el contrario, la genética es una alternativa mejor; así como comparar la representación de los modelos de acciones, si la complejidad escala demasiado para utilizar una simple tabla, o si una red neuronal es demasiado costoso para el modelo elegido en la representación del entorno 1.

2.1.3 Metodología y rigor

Se ha seguido una metodología IID (*Iterative and Incremental Development*, "desarrollo iterativo e incremental") con 4 etapas principales del proyecto, en el que, para cada etapa, se sigue un proceso de planificación, implementación, pruebas durante desarrollo, y una actividad de evaluación, en la que se mejora o se corrige el código.

2.2 Descripción de los entornos

El entorno de desarrollo de este proyecto es Minetest, que utiliza un sistema de *scripts* en Lua para la creación de sus *mods*; así como las herramientas relacionadas con el desarrollo para Minetest. El entorno físico es el espacio tridimensional en el que tiene lugar el juego.

2.2.1 Entorno físico

La forma del terreno se genera proceduralmente a partir de funciones de ruido Perlin (que veremos con detalle más adelante). Es una aplicación muy habitual de esta implementación concreta de forma de ruido, ya que permite generar paisajes interesantes, con valles y montañas, garantizando una cierta facilidad de navegación.

El terreno en sí está formado por vóxeles o cubos de tierra, roca u otros materiales sólidos. Estos cubos son referidos internamente como nodos. El mundo se divide en biomas, representando la variedad de climas de la Tierra, y dependiendo de dichos biomas puede haber también cubos de arena (imitando el desierto), paisajes nevados, etc. En una gran mayoría de biomas suelen crecer árboles, que obstaculizan el paso pero se pueden talar por parte del jugador, para la creación de herramientas o como material de construcción.

El jugador tiene un valor numérico que representa la vida. Cuando ésta se reduce a 0, el avatar del jugador muere, y reaparece en el *spawnpoint*.

Todo aquél elemento de juego que no es un nodo, y generalmente puede moverse por el espacio, es definido internamente como una *Lua entity*, o "entidad", a excepción del jugador, que es el único elemento de la clase `Player`. Las entidades de Lua tienen 2 variables asociadas: `name`, que es el nombre con el que se registra el tipo de entidad en el motor de Minetest; y `object`, que es una referencia al objeto que representa a la entidad en el espacio de juego. Este objeto es del tipo `LuaEntitySAO`, que hereda todas las propiedades de la clase `ObjectRef`, de la que también hereda `Player`. Las *Lua entity*, los elementos de tipo `ObjectRef`, y a los elementos de tipo `Player` serán referidos como entidades, objetos y jugadores respectivamente.

Existen otras características del terreno y de las herramientas existentes en el Minetest *vanilla*, pero no son relevantes al desarrollo de este proyecto así que no las trataremos aquí.

Como hemos mencionado antes, el espacio de juego se divide en cubos o vóxeles. Esto permite una representación de la posición de un elemento en el espacio con coordenadas numéricas *x, y, z*; en concreto, $+x$ indica un sentido hacia el norte, $+y$ representa un sentido ascendente en vertical, y $+z$ representa un sentido hacia el este. Además, para todos los eobjetos que no son nodos, existe una propiedad llamada *"yaw"*, que indica, mediante un valor numérico en radianes, la dirección a la que está encarado. Un ángulo de 0 grados significa estar mirando al norte, o $+x$; un ángulo de 270º ($3\ \pi/2$ en radianes) significa estar mirando al este, o $+z$.

2.2.2 Entorno de desarrollo

Programar para Minetest es, en primer lugar, programar para Lua. Lua es un lenguaje de programación imperativo, es decir, que se basa en la mutación de variables siguiendo el orden de las instrucciones escritas.

El propio Minetest tiene una API disponible para la elaboración de *plug-ins* del juego en Lua, que será ampliamente utilizada; y para programar en Lua es suficiente con cualquier editor de texto. Sin embargo, Lua **no** dispone de un debugger, así que no habrá ninguna herramienta utilizable con esa función.

El proyecto se mantendrá en un repositorio público de GitHub, siguiendo la idea de ofrecer una plataforma sobre la cual poder elaborar otros proyectos de IA en Minetest sobre una base similar a la de este proyecto.

3 Desarrollo

3.1 Decisiones de diseño y contratiempos

3.1.1 Implementación de *mobs*

Siendo una comunidad de fans pequeña pero prolífera, existen varios *mods* de *mobs*, o criaturas, para poblar el mundo de Minetest con vida similar a la de Minecraft: animales de granja que dan comida y pieles, zombies que presentan amenaza por la noche, etc.

Para no contaminar el mercado de *mods* con un nuevo sistema de implementación de *mobs* sobre entidades genéricas de Minetest (aparte que realizar una implementación propia podría suponer incompatibilidades con otros *mods* de la misma naturaleza) se ha decidido basar el trabajo sobre otro *mod* de la comunidad que provea este aspecto de la implementación.

Tras barajar varios *mods*, al final se ha optado por *Mobs Redo*, que es un conjunto de *mods* de este estilo desarrollados por una misma persona, que provee una plataforma de desarrollo de *mobs* sobre la cual el resto de criaturas del conjunto están implementadas.

3.1.2 Pathfinding y navegación del espacio

Hay 2 formas principales de tratar la navegación del espacio para el prototipo de IA:

- Se define un estado ”movimiento” y se utiliza una técnica de *pathfinding* popular (A*, Dijkstra, ...) para calcular una ruta de la posición de la IA al objetivo, haciendo que la IA siga la ruta siempre que se encuentre en este estado; o bien
- se definen distintas acciones de movimiento como parte de la IA, y se deja que se desarrolle una estrategia de navegación como parte de la evolución del algoritmo.

Al principio se consideró la primera opción; la propia API de Minetest ofrece una función que permite seleccionar uno de los métodos de *pathfinding* definidos internamente (el algoritmo de Diijkstra, y dos variaciones del A* según optimización), y devuelve una ruta calculada compuesta de nodos navegables (aire, principalmente). Sin embargo, esta función aún se encuentra en modo experimental, es sensible a cualquier error en las coordenadas, y no garantiza el correcto funcionamiento.

Otros *mods*, como *Mobs Redo*, implementan un mecanismo de *pathfinding* interno basado en la función de Minetest pero con el objetivo de suplir los fallos de la función original, pero estas implementaciones son todas internas y codependientes de otras partes del *mod*, por lo que no están disponibles para ser llamadas desde un *mod* externo ni tampoco pueden ser copiadas con facilidad sin tener también que copiar una gran cantidad de otras partes del *mod*. Estos *mods* han sido aparentemente desarrollados con el único caso de uso del desarrollo de criaturas imitando el comportamiento simple de los originales de Minecraft, y no para un caso de uso como el de este proyecto (a pesar de que *Mobs Redo* en concreto ofrece la posibilidad de crear *callbacks* personalizados para el desarrollo basado en la plataforma).

Se ha intentado aprovechar la existencia de estas funcionalidades en *Mobs Redo*, pero la forma en la que se provocaba su llamada era cuestionable en el mejor de los casos, y no había forma factible de reimplementar la función interna de *pathfinding*, por lo que finalmente se ha optado por la segunda opción.

3.1.3 Persistencia y almacenamiento

Para mantener la persistencia entre cargas de partidas, la información debe guardarse en un archivo y ser cargada la próxima vez que se entra al mundo, ya que el motor de Minetest sólo conserva la información de una entidad que tiene en Minetest *vanilla*, y es en la ejecución de los códigos de *mods* que incorpora el resto de propiedades personalizadas. Para esto, la API de Minetest ofrece 2 funciones, `minetest.serialize(...)`, que recibe una tabla de Lua como entrada y devuelve un *string* que representa la tabla, y `minetest.deserialize(...)` que interpreta un *string* con el mismo formato que los de salida de `serialize` y devuelve la tabla o el elemento serializado.

Sin embargo, estas funciones utilizan las funciones nativas de Lua para interpretar *strings*, que son conocidas por tener un buffer muy limitado y poder interpretar *strings* de tamaño pequeño, o si no, son truncados por la función. Esto ha limitado el tamaño del algoritmo evolutivo del prototipo. Ensayo y error ha demostrado que el límite es alrededor de unos 2000 elementos, en el caso de la deserialización de Minetest.

Así y todo, para poder guardar tanto las características de la criatura como los genes del algoritmo genético, para cada criatura se han guardado 2 archivos, separando los genes del resto de características.

3.1.4 Genética vs. aprendizaje reforzado

Se ha barajado y estudiado las ventajas y desventajas de cada uno de los métodos de evolución de algoritmos para el desarrollo del prototipo de ejemplo, así como el diseño de la propia plataforma.

Por un lado, el espacio en memoria de un algoritmo genético se dispara con facilidad. La implementación de algoritmo genético que se ha tomado es la creación de una lista de genes para cada posible combinación de características de entrada, donde se guarda el valor de salida; un exceso de espacio en memoria puede retrasar el funcionamiento correcto del *gameloop* de forma excesiva, o resultar intratable para el sistema de persistencia de Minetest por ser demasiado grande.

Por otro lado, al encontrarse la criatura en un terreno 3D tan variable es muy difícil diseñar un sistema de estados agnóstico al terreno, que garantice que una acción en unas condiciones concretas sea siempre igual de beneficioso en distintas posiciones del terreno, e intentar incluir las diferentes posibles posiciones en los estados dispara el número de éstos, así como garantizar que una acción A para un estado S_i siempre mueva a la criatura a un estado S_{i+1}; mientras que el algoritmo genético puede prescindir de considerar estados. La implementación de Q-Learning que se ha considerado es la de una tabla de estados × acciones disponibles, con cada casilla un valor que representa el siguiente estado que se alcanza realizando esa acción.

Se ha realizado una estimación del espacio necesario para cada modelo diseñado para un algoritmo genético y uno de *Q-Learning*, y son los siguientes:

Condición	**Algoritmo genético**	**Q-Learning**
Niveles de hambre y vida	3 valores (bueno, malo, crítico)	2 valores (bueno, malo)
Dirección de la comida más cercana	8 valores (8 direcciones cardinales)	8 valores (8 direcciones cardinales)
Vecinos navegables	3 valores de altura (escarpado, caída, nivelado); 4 direcciones	2 valores (navegable, no-navegable); 4 direcciones
Comida guardada	—	2 valores (sí, no)*
Posibles acciones	12 valores	12 valores
Valores totales	**3*8*3^4 = 1944**	**2*8*2^4*12 = 3072**

(Más detalles sobre el significado de estos valores en la sección de *Implementación de la IA prototipo*)

Como hemos mencionado antes, el límite de deserialización de Minetest se encuentra alrededor de los 2000 valores en un mismo *string*. Teniendo en cuenta que el algoritmo genético ofrece una mayor precisión en un espacio ocupado menor y que el número de acciones disponibles no repercute en el tamaño del espacio final ocupado, es por esto que se ha decantado por priorizar el mecanismo de algoritmo genético para el prototipo de IA.

Eliminar los valores de comida guardada podría reducir el espacio ocupado por la tabla del algoritmo de Q-Learning, pero se ha considerado necesario porque en el caso de Q-Learning, intentar comer sin tener comida en el bolsillo peude repercutir negativamente en la estimación del beneficio de la acción "comer" en general, mientras que el algoritmo genérico puede volver a intentarlo unos turnos más tarde con la misma probabilidad.

Otra solución sería reducir las direcciones de la comida a 4 (de forma que si se encuentra, por ejemplo, al norte y al este a la vez, sólo se vea uno de los dos), pero en ese caso la precisión del algoritmo de Q-Learning se ha visto reducida casi a la mitad que el algoritmo genético, por lo que sigue saliendo a cuenta directamente centrar los esfuerzos en el algoritmo genético.

3.1.5 Redes neuronales

Una de las primeras opciones barajadas para el desarrollo de IAs y, en concreto, del prototipo de IA de prueba de la plataforma, fue la incorporación del uso de redes neuronales en los algoritmos evolutivos. Existe incluso una librería para ello en Lua, llamada *torch*, que se había considerado utilizar para este fin, ya que reinventar la rueda e implementar de cero un sistema de red neuronal escapaba al alcance de este proyecto, ya que requeriría de más esfuerzo y tiempo del disponible y tampoco era el objetivo principal.

Sin embargo, la plataforma se ha planteado desde un inicio como un *mod* accesible y fácil de usar, destinado a la mejora de la experiencia de juego, por lo que se han tomado una serie de decisiones de diseño siguiendo las convenciones del desarrollo de *mods* de Minetest. Esto incluye varias convenciones de seguridad de mods; sin embargo, el juego bloquea por defecto la ejecución de *mods* que requieren librerías externas al juego, y para poder hacerlo se debe requerir en el código un "entorno no seguro" a la API de Minetest. Se ha estimado que un entorno no seguro no garantiza una experiencia agradable de juego, aparte del hecho que varias decisiones de diseño tomadas para seguir las convenciones de seguridad no tendrían sentido, y la mejor ruta de acción sería hacer un segundo mod con un entorno no seguro que expanda la primera plataforma. Pero, de nuevo, escapa al alcance y al tiempo disponible del proyecto, por lo que se ha descartado.

3.1.6 Tiempo de entrenamiento

Otro imprevisto del desarrollo de este proyecto, aunque no afecta directamente al tiempo de desarrollo si no a la ruta de decisiones tomadas, es el hecho que Minetest, en tiempo real de juego, ha resultado ser un entorno de entrenamiento lento para las IAs. Para solventarlo, se han considerado 2 opciones:

1. Elaborar un programa sencillo de entrenamiento previo al juego, y luego utilizar dentro del propio juego el algoritmo resultante de entrenar la IA.

2. Añadir al proceso de Minetest de creación de un mundo nuevo donde jugar, una etapa de generación y entrenamiento de una IA, donde utilizar luego en dicho mundo.

Primero se ha elaborado el programa externo, y con ello se han medido los tiempos, para observar si era factible introducir el propio entrenamiento en el proceso de creación de un mundo nuevo. Pero incluso sin outputs que puedan retrasar el programa, un entrenamiento super básico requiere media hora, pero apenas da para ninguna evolución del algoritmo; y entrenar una población de 100 criaturas durante 5000 generaciones (con un máximo de 300 steps pero apenas superando los 150), que empiezan a ser números razonables, ya requiere alrededor de 27 horas. Por lo tanto, se ha descartado la segunda posibilidad.

3.2 Estructura de la plataforma

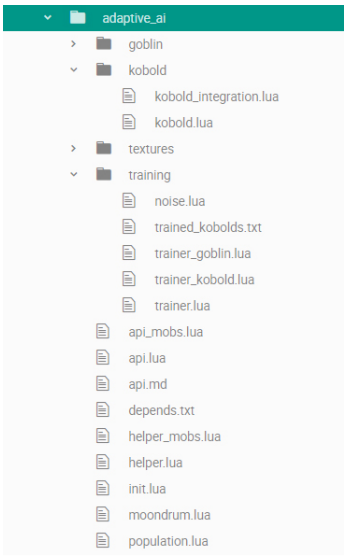


Figure 5: Árbol de jerarquía de carpetas del *mod*.

El *mod* de Minetest de este proyecto se ha desarrollado bajo el nombre de *Adaptive AI*. En la figura 5 se puede apreciar la jerarquía de carpetas y archivos de dicho *mod*. La distribución de los archivos es la siguiente:

Carpeta raíz:

- **`depends.txt`** - Archivo requerido por la API de Minetest. Aquí figuran los nombres de los *mods* de Minetest de los que el mod actual depende, y que deben leerse antes que el mod actual, incluyendo partes del juego *vanilla* que se definen internamente también como *mods*. Éste es dependiente de los *mods* de **`default`** (el núcleo del juego de Minetest) y **`mobs`** (el identificador de *Mobs Redo*).

- **`init.lua`** - Archivo requerido por la API de Minetest. Este es el archivo principal leído y ejecutado por el juego al cargar un *mod*. Cualquier otro archivo debe ser ejecutado desde aquí (con las funciones `minetest.get_modpath(mod_name)` para obtener la ruta a la carpeta del *mod*

sin romper las medidas de seguridad de Minetest, y `dofile(...)` para ejecutar el archivo).

- **`helper.lua`** - Aquí se definen una serie de funciones de ayuda, como suma de una lista, normalización de vectores, una implementación de "pop" de array sobre tablas de Lua, etc.
- **`helper_mobs.lua`** - Aquí se definen una serie de funciones de ayuda extraídas de *Mobs Redo* y adaptadas para su posible uso en este *mod*.
- **`api_mobs.lua`** - Aquí se definen varios callbacks estándar con funcionalidades extraídas de *Mobs Redo*.
- **`api.lua`** - Aquí se definen las funciones de la API de esta plataforma; en concreto, el registro de entidades como IAs adaptables, así como un esqueleto de *step callback* estándar para IAs.
- **`api.md`** - Explicación de la API en un archivo de Markdown para todos aquellos que quieran usar esta plataforma.
- **`moondrum.lua`** - Implementación de un nuevo nodo que representa un tipo de seta único de este *mod*, para servir de comida a la IA prototipo y así no interferir con otros posibles usos de nodos de setas (u otros elementos comestibles) de otros *mods*.
- **`population.lua`** - Controlador genérico de poblaciones de criaturas, que el *script* de
- **`[criatura]_integration.lua`** debe utilizar en la integración al juego.

[**Carpeta de criatura**]**:** En este caso, se trata de la carpeta **kobold**, donde se encuentra la IA prototipo.

- **`[criatura].lua`** - Implementación base de la IA, sin utilizar ninguna funcionalidad de la API de Minetest (para poder entrenarla de forma externa usando este archivo). En caso del prototipo, **`kobold.lua`**.
- **`[criatura]_integration.lua`** - Integración de la IA de la criatura con la API de Minetest. En caso del prototipo, **`kobold_integration.lua`**.

Este sistema de nomenclatura no es obligatorio mientras las distintas partes se referencien entre sí correctamente.

Training:

- `trainer.lua` - Este *script* presenta una estructura de entrenamiento de IAs según su implementación base.
- **trainer_[criatura].lua** - Este *script* implementa las funcionalidades de `trainer.lua` de forma específica para la criatura que debe entrenar; en caso del prototipo, `trainer_kobold.lua`.
- `trained_[criatura]s.txt` - Archivo generado con la información esencial para crear instancias de IA dentro del juego que utilicen los resultados del entrenamiento.

Como con la [**carpeta de criatura**], este sistema de nomenclatura no es obligatorio mientras las distintas partes se referencien entre sí correctamente.

Textures: Se trata de una carpeta requerida por la API de Minetest, que contenga los archivos de texturas necesarios para el *mod* (aunque no haya ninguna).

3.3 Modelo de la IA

Independientemente del algoritmo elegido para implementar la inteligencia artificial, se han definido los siguientes comoponentes como necesario para el desarrollo de una IA sobre esta plataforma:

- **Tabla de acciones:** se trata de un conjunto etiquetas legibles por humanos de las acciones posibles por parte de la IA; una tabla de *strings* asociadas a índices de número entero, como un array.
- **Función de decisión:** esta función recibe como parámetro de entrada una entidad de la IA, y opcionalmente características de su entorno (si no vienen incluidas ya internamente con la entidad de la IA); y devuelve el índice de la acción a realizar correspondiente de la tabla de acciones.
- **Tabla de *action managers*:** se trata de una tabla de funciones anónimas que se encargan de ejecutar las acciones a realizar; a estas funciones se les refiere como *action managers*. Cada *action manager* debe estar asociado a un índice de número entero igual a la acción representada por la etiqueta con mismo índice de la tabla de acciones.
- **Función *manager*:** esta función se encarga de ejecutar las instrucciones necesarias para la interacción de la IA con el entorno, el control del movimiento, el control de las características internas, y el llamar al *action manager* adecuado para cada acción tomada por la función de decisión.

La implementación del *manager* se realiza utilizando el *step callback* de *Mobs Redo*, que a su vez está implementado sobre el *step callback* estándar de la API de Minetest, y debe estar implementado para la IA en el momento del registro en la ejecución de la API.

3.3.1 Implementación de la IA prototipo: Kobold

Se han definido las siguientes características base para la IA:

- `view_range`: Característica introducida por *Mobs Redo*. Rango de visión del kobold, cuyo valor es 20. En *Adaptive AI*, la comida que se encuentre fuera de un radio de 20 unidades (un vóxel es $1 \times 1 \times 1$ unidades) es indetectable por parte del kobold.
- `jump`: Característica introducida por *Mobs Redo*. Altura de salto, cuyo valor es 2. Esto le permite saltar 1 nivel hacia arriba, pero no más de uno a la vez.
- `hunger_max`: Valor inicial y máximo de "hambre". Se reduce cada vez que tiene lugar un *step*, y se aumenta cuando se realiza la acción "comer" con al menos una unidad de comida en el bolsillo.
- `hp_max`: Característica introducida por *Mobs Redo*. Valor inicial y máximo de vida. Cuando llega a 0, el kobold muere. Cuando el hambre llega a 0, la vida empieza a disminuir.
- `fear_height`: Característica introducida por *Mobs Redo*. Altura máxima que una criatura debe estar dispuesta a bajar, para evitar daño innecesario por caída.
- `climb`: Altura máxima que la criatura puede estar dispuesta a subir. Debería representar la altura máxima alcanzable gracias a `jump`, pero en unidades de vóxeles.

En todo momento, si se detecta comida dentro del rango de visión y a menor distancia que la posición guardada de comida más cecana, se guarda la nueva posición en la propiedad `closest_food`.

Luego tiene otras variables de control internas como `total_genes`, con el número total de genes (1944); o `mutation_rate`, que es 0.2; de esta forma son fáciles de ubicar en el archivo Lua y de modificar en caso que así se desee; pero no tienen nignuna relevancia más allá de la creación de nuevos kobolds.

Una variable de control que vale la pena mencionar es el `focus`, o concentración, que es un valor entre 0 y 1 que representa la probabilidad de que un kobold mantenga la decisión tomada en el turno anterior.

Como hemos visto en la sección de *Decisiones de diseño*, hemos considerado los siguientes valores a tener en cuenta en la función de decisión:

- **Nivel de hambre/vida:** según los rangos en los que se encuentre el hambre, se determina una de 3 posibles categorías: 0 (*bueno*), 1 (*malo*) o 2 (*crítico*).
- **Dirección de la comida:** según la dirección en la que se encuentre la comida más cercana (si hay), se devuelve uno de 8 valores asignados a las flechas cardinales: 0 (NE), 1 (NO), 2 (SE), 3 (SO), 4 (N), 5 (S), 6 (E), 7 (O). Si no se ha detectado comida, se devuelve 5 (S). Si se ha detectado comida

pero ha habido algún error en los cálculos, se devuelve 4 (N). En la versión original se devolvía una dirección random en estos 2 últimos casos de error, pero dificultaban el análisis del correcto funcionamiento del algoritmo.

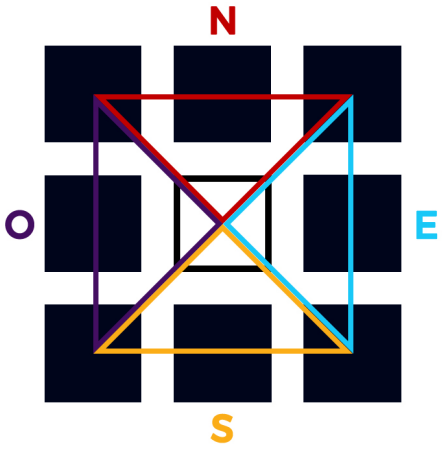


Figure 6: Direcciones de la navegabilidad de vecinos, vista cenital.

- **Navegabilidad de los nodos vecinos:** Considerando las 8 posibles combinaciones de *x, z* adyacentes a la posición del kobold, se observa la altura del suelo sólido. En cada una de las 4 direcciones cardinales, se hace una media de los 3 valores de alturas (en la figura 6, representado con 4 triángulos de colores distintos, 1 por dirección); por ejemplo, para la dirección norte se hace media de las casillas NO, N, y NE; y así sucesivamente. Luego, se analiza la diferencia entre la altura obtenida y la altura en la que se encuentra el kobold, y entonces se asigna una de 3 categorías distintas para cada una de las direcciones: 0 (*nivelado*), 1 (*caída*, para descensos mayores a `fear_height`), y 2 (*escalada*, para ascensos mayores a `climb`).

La no-navegabilidad en ascenso se trata distinto que en descenso, porque en descenso se permite descender (a coste de perder vida), mientras que en ascenso no.

La función de puntuación se explica con detalle en la sección de entrenamiento, que es donde más relevancia tiene del programa entero, pero a grandes rasgos se basa en el hambre y la vida restantes únicamente, para dejar lugar a la posibilidad de ver emerger distintas estrategias de supervivencia.

3.4 Estructura del entrenamiento

El entrenamiento se realiza de forma externa al juego, mediante un *script* en lua ejecutable desde consola, que utilice las funcionalidades y la estructura provista en `training/trainer.lua`.

En el *script*, se realizan los siguientes pasos:

1. Se define un tamaño de población máximo P y un número de generaciones G durante las que entrenar la población (dados por el *script* secundario de entrenamiento personalizado a la criatura); y se generan P criaturas para llenar la población.

2. Para cada generación, se genera un escenario de dimensiones $N \times M$ (dados también por el *script* secundario) y una altura máxima L. Para cada casilla, se asigna un valor de 0 a L, representando la altura.

   - Este valor se calcula mediante una función de ruido Simplex, una función similar, pero de cálculo más sencillo, a la función de ruido Perlin, que es la función utilizada en Minetest para generar los mundos. El objetivo de esto es el de tener un entorno similar al de Minetest, pero simplificado para facilitar y agilizar la ejecución del entrenamiento.
   - El valor se calcula de la siguiente manera: para cada casilla (x, z) (siguiendo el estándar impuesto por Minetest), se calcula el ruido Simplex 2D para los puntos $a = Simplex(\frac{x}{N/4}, \frac{z}{M/4}$, $b = Simplex(x/2, z/2)$, y $c = Simplex(x, y)$; y luego se suman para obtener el valor $a + b + 1.5 * c$. Con esto, se obtiene un paisaje relativamente suave, garantizando la navegabilidad a casi todas partes del mapa, pero con los valores de mayor variación introduciendo pequeños desniveles, permitiendo el entrenamiento de circumnavegar diferencias de altura más grandes que `climb` y `fear_height`.
   - El entrenamiento permite la declaración de una función que introduzca información extra. Por ejemplo, el entrenador de kobolds introduce la aparición de setas, así como la posibilidad de definir casillas como impasables, para aprender a sortear obstáculos no saltables.

3. Para cada criatura, se la sitúa en una casilla aleatoria del mapa, y durante 300 *steps*, o hasta que el kobold vea su vida reducida a 0, se llama a la función de decisión, que luego se interpreta para ejecutar la acción que toma la criatura en sus condiciones actuales. A cada paso, además, se llama a la función de puntuación definida en el archivo de `[criatura].lua`, hasta que se pasa a la siguiente criatura (aunque a todas aquellas criaturas que no sobreviven hasta el último turno se les reduce la puntuación final a un porcentaje equivalente al porcentaje de turnos sobrevividos respecto al número de turnos totales).

- El entrenamiento permite la declaración de una función que añada comprobaciones del mapa a cada turno. En el caso de los kobolds, se recorre el mapa mirando la distancia de cada seta al kobold, y se actualiza la propiedad de `closest_food` si es necesario; y se comprueba, en cada casilla sin seta, si la posibilidad de aparición de una seta por casilla y turno da lugar a dicha aparición.

4. Al final de cada generación, se ordena la población de criaturas por puntuación, se eliminan $(1-s)*P$ criaturas, donde s es el porcentaje de selección establecido desde el *script* de entrenamiento secundario (a no ser que la mejor criatura no supere un umbral mínimo de puntuación, en cuyo caso se elimina la población entera y se vuelve a empezar). Luego, las criaturas supervivientes se utilizan para generar nuevas criaturas, hasta volver a tener un número P de criaturas.

    - La criatura con más puntuación de cada generación tiene al menos 1 descendencia asegurada.
    - En el caso de los kobolds, se cruzan los genes del algoritmo genético, con un 20% de posibilidades de generar un nuevo gen aleatoriamente, o con un 50% de posibilidades de coger el gen del padre o de la madre dentro del 80% restante.

5. Se repite el proceso desde el paso 2 hasta haber realizado G generaciones. Tras lo cual, se coge a la mejor criatura (o k mejores criaturas) de la última generación, y se guardan en archivo mediante una función definida en el *script* de `[criatura].lua`.

3.4.1 Ejemplo de entrenamiento: Kobolds

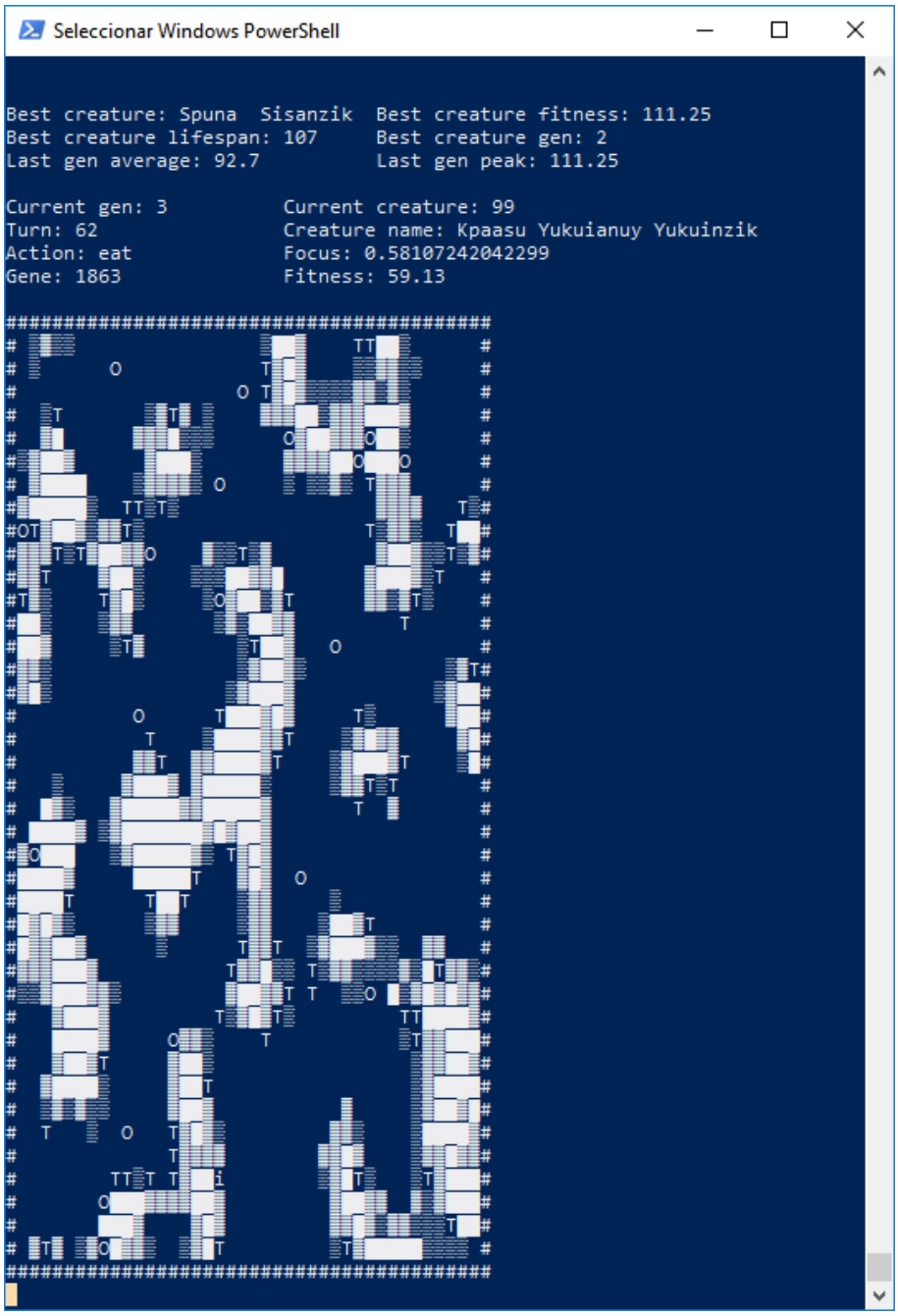


Figure 7: Pantalla de entrenamiento de kobolds.

En la figura 7 podemos observar una pantalla habitual durante el entrenamiento de los kobolds.

En lo alto de la pantalla podemos observar cuál ha sido la criatura con mejor puntuación de todas las generaciones anteriores (en este caso “Spuna Sisanzik”), su puntuación, el número de turnos que ha sobrevivido, en qué generación obtuvo la mejor puntuación, así como la media de puntuación y la máxima puntuación de la generación inmediatamente anterior, para comparar y observar si se acerca a superar el récord actual.

Luego podemos ver información sobre la criatura siendo entrenada en este momento. Se indica la generación actual, el número de la criatura dentro de la población, el turno o *step* en el que se encuentra, y su nombre (compuesto por un nombre de pila, un segundo nombre, que indica la criatura madre que lo generó, y un apellido que indica la criatura de generación 1 de la que desciende; el propósito de los 2 últimos es únicamente el de poder controlar a qué linaje pertenece cada criatura, para llevar control de la descendencia). finalmente, podemos ver la acción que ha decidido tomar en este turno, así como su `focus`, el número identificador del gen utilizado para tomar la última decisión, y la puntuación acumulada hasta ese momento.

Por último, podemos observar el mapa de entrenamiento, generado aleatoriamente en cada generación. En concreto, podemos observar en el mapa:

- Las "`T`" representan setas *moondrum*.
- Las "`O`" representan obstáculos impasables.
- La "`i`" representa la posición actual del kobold.
- Los 4 grados de blanco y la asuencia de blanco indican distintas alturas. Blanco completo indica que la casilla se encuentra a 2 niveles o más por debajo de la criatura, la ausencia completa de blanco representa que la casilla se encuentra a 2 niveles o más por encima de la criatura; y el resto de blancos son valores intermedios, con el blanco intenso pero incompleto representando el mismo nivel que el kobold.

Función de puntuación

La función de puntuación de un kobold se calcula de la siguiente manera:

1. Se calcula el porcentaje de hambre decaída respecto al máximo:
   $h = (hunger\_max - self.hunger)/hunger\_max$
2. Se calcula el porcentaje de vida restante respecto al máximo:
   $v = (hp\_max - self.hp)/hp\_max$
3. Se calcula $h + 2 * v$, y se redondea el valor final a un número entero.

El objetivo es premiar el estar saciado a menudo, pero con un margen laxo si no se está saciado al máximo el máximo tiempo, haciendo foco en sobrevivir y no perder vida.

4 Gestión del proyecto

4.1 Planificación del proyecto

Esta sección fue redactada al inicio del proyecto. Se ha dejado la planificación original, y se ha añadido una sección anexa que repasa los cambios necesarios para adaptarse a los contratiempos y cambios en las decisiones de diseño mencionados en el punto **3.1**.

4.1.1 Estimación de tiempo

Este proyecto tiene una duración aproximada de 4 meses, partiendo de un trabajo previo de reunión de información, al desarrollo técnico del proyecto en sí, hasta la entrega de la memoria y defensa oral final del proyecto, alrededor del del 25-29 de junio aproximadamente.

4.1.2 Fases del proyecto

El proyecto es mayoritariamente lineal, con unas dependencias bastante marcadas de una actividad a la siguiente; lo cual hace difícil sopesar el efecto de un imprevisto en una actividad considerablemente avanzada en el proyecto desde un inicio, o bien el posible efecto acumulado de varios imprevistos. Así mismo, es difícil utilizar un tiempo reducido en las actividades más centrales del proyecto sin que la calidad de los resultados se vea afectada gravemente.

Para poder combatirlo, se han designado una serie de actividades como *"prescindibles"* (codificadas en amarillo en el diagrama de Gantt), siendo éstas las primeras actividades cuyo tiempo se recortará o se suprimirá en caso de que imprevistos o retrasos puedan afectar al cumplimiento de las metas del proyecto en las fechas determinadas; y de esta forma conseguir que el tiempo dedicado a las actividades más intensivas se vea afectado lo menos posible.

El tiempo estimado para cada actividad es aproximado y puede variar durante el transcurso del proyecto. En el caso de que una actividad termine antes de lo previsto, la actividad siguiente se iniciará de inmediato. En el caso de que una actividad se alargue más de lo necesario, se sustraerá el tiempo extra de otras actividades menos relevantes (actividades *prescindibles* primero).

En caso de que al final del proyecto el balance de tiempo sea positivo, el tiempo sobrante extra será invertido en las actividades de *expansión de IA*, *"Clean-up"* y *corrección de errores*.

Hito inicial

Esta frase es un arco general de actividades destinadas a la documentación y planificación del proyecto en sí, y corresponde con el contenido de la asignatura de GEP (Gestión de Proyectos).

En concreto, las actividades que componen este arco son las siguientes:

Actividad	**Horas**
Alcance del proyecto y contextualización	24
Planificación temporal	8
Gestión económica y sostenibilidad	9
Presentación preliminar	6.5
Pliego de condiciones	6.5
Documento final y presentación oral	26
Total	**80**

Diseño de la IA

Este arco está compuesto por la reunión de información y la investigación de técnicas necesarias para futuras etapas del proyecto, cuyo comienzo precede al trabajo en sí y luego se desarrolla a lo largo de las primeras etapas.

Las actividades de este arco consisten principalmente en el estudio de técnicas de IA generales y populares, el análisis a grandes rasgos las características principales del entorno de la IA y planificación de su representación abstracta, así como la definición de un esquema de las entradas que la IA debe considerar, así como del posible conjunto de salidas que deben determinar la acción de la IA a realizar; así como un tiempo especial de familiarización con técnicas de NN (redes neuronales, por las siglas en inglés *"neural networks"*), ya que son las técnicas de IA menos familiares para el autor del proyecto, y 2 periodos tardíos de revisión y refinamiento del análisis del entorno y del diseño del comportamiento, según necesidades no previstas surgidas durante las pruebas del *desarrollo de IA*.

Las actividades que lo componen han sido definidas de la siguiente manera:

Actividad	**Horas**
Familiarización con las técnicas de IA	40
Análisis básico del entorno	28
Revisión del análisis	0 - 80
Diseño del comportamiento	24
Revisión del comportamiento	0 - 80
Familiarización con técnicas de NN	20
Total	**112 - 272**

Se dispone de un amplio margen de tiempo para lidiar con imprevistos o necesidades de corrección de los modelos planteados incialmente, que en caso de no ser necesarios pueden ser reaprovechados para futuras actividades críticas.

Creación de Plug-in

Este arco de actividades tiene un objetivo claro y bien definido, y es es el de obtener un plug-in mínimo funcional, que sea detectado por el menú de selección de plug-ins de Minetest, que pueda ser seleccionado, y que pueda utilizarse sin provocar errores de ejecución del programa; como un equivalente de "Hola Mundo" en el entorno de plug-ins de Minetest, enfocado de cara a los objetivos del proyecto.

Específicamente, el plug-in mínimo que se busca conseguir es definido de la siguiente manera: se desea un *"Mob Spawner"*, esto es, una herramienta que permita crear instancias (*"spawn"*) de una criatura (*"mob"*) no inteligente dentro del juego. Esto generalmente se realiza mediante la utilización de un objeto especial arrojadizo que marca el punto de coordenadas de la nueva instanciación.

Las actividades se definen de la siguiente manera:

Actividad	**Horas**
Instalación y configuración de las herramientas	12
Familiarización con el entorno de Minetest	12
Creación del *Mob Spawner*	60
Pruebas y correcciones	6 - 60
Obtención de Plug-in funcional	*Hito*
Total	**90 - 144**

4.1.3 Esqueleto de IA

El objetivo de esta etapa es el de construir el esqueleto de una IA en el plug-in funcional obtenido en la etapa anterior, haciendo que el *mob* que instancia el *Mob Spawner* sea capaz de moverse (en una dirección aleatoria) por el mapa. Esto es a la vez una actividad y un hito objetivo.

Actividad	**Horas**
Creación de una IA de movimiento aleatorio	30

Desarrollo de IA

El propósito de esta etapa es la meta central del proyecto: el uso y la comparación de diversas técnicas de aprendizaje de IA para desenvolverse en el entorno determinado. El prototipo inicial de la IA se desarrollará con las funcionalidades básicas de moverse por el mapa y buscar alimento con el que mantenerse vivo. Las correcciones y revisiones de estas IAs se realizarán simultáneamente al desarrollo, así que no se definen como actividades separadas.

Actividad	**Horas**
Creación y pruebas: Algoritmo genético (GA)	40
Creación y pruebas: *Reinforcement learning* (RL)	40
Creación y pruebas: *Genetic NN* (GNN)	40
Creación y pruebas: *RLNN*	40
Total	**160**

Expansión, "clean-up" y correcciones

Estas 3 actividades recogen el tiempo sobrante del proyecto, si hay, o proporcionan tiempo prescindible a redistribuir a otras actividades que requieran más tiempo del planeado, y respectivamente tienen los objetivos de: aumentar las habilidades de la IA (posibles mejoras podrían ser un sistema de hostilidades y simpatías, o percepción de temperatura, etc.); modularizar y limpiar el código, y hacerlo más accesible a futuros desarrolladores que trabajen sobre la plataforma desarrollada en este proyecto; y corregir los últimos errores de código no solventados en etapas anteriores.

Actividad	**Horas**
Expansión de IA	0 - 40
"Code clean-up"	0 - 20
Corrección de errores	0 - 40
Total	**0 - 100**

Hito final

Esta es la etapa designada a la preparación de la defensa oral del proyecto de fin de Grado ante un tribunal, y toma por lo menos el tiempo desde la entrega de la memoria del proyecto hasta la fecha de la defensa oral, que por norma es mínimo 1 semana de antelación.

Diagrama de Gantt

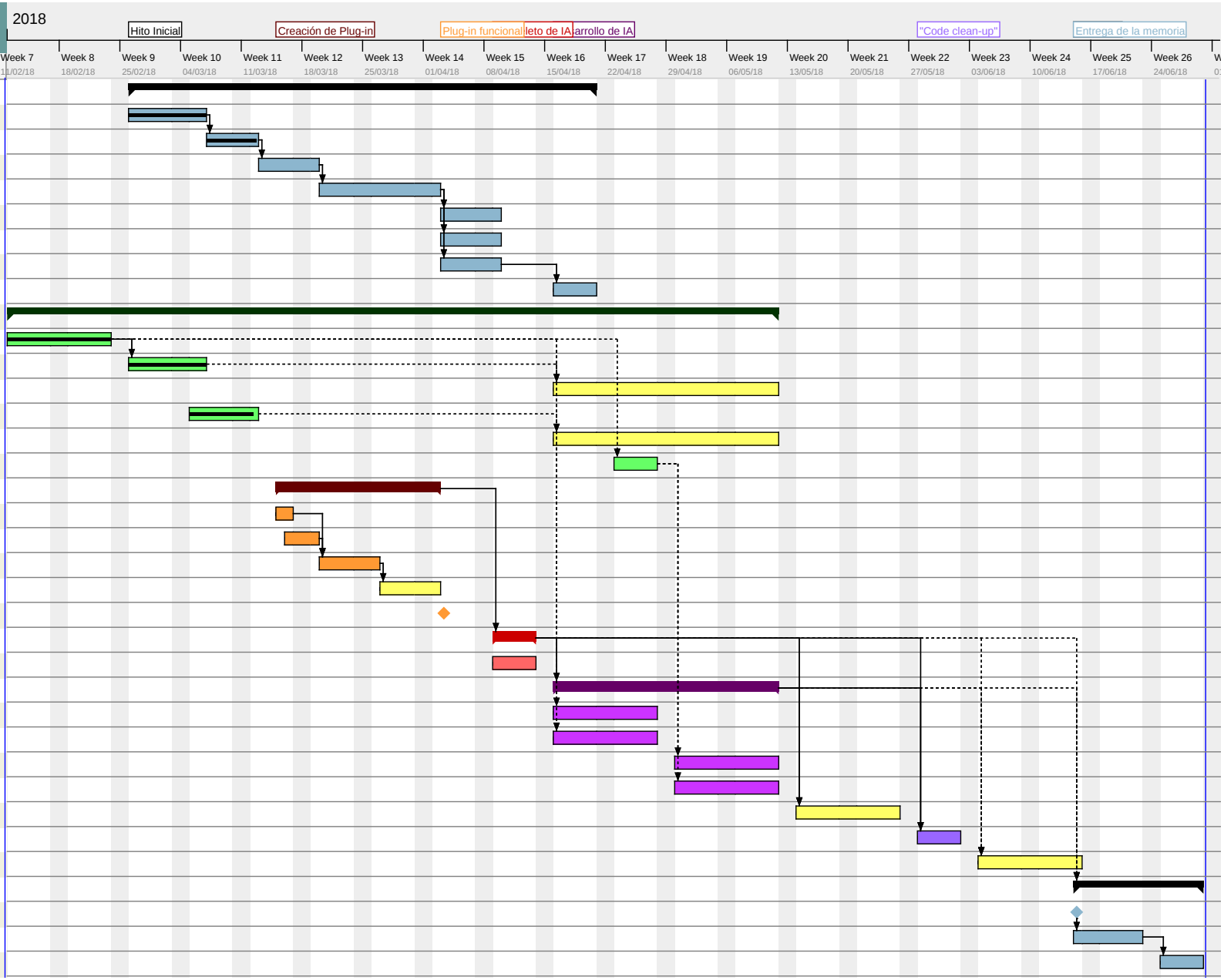
Se encuentra en la siguiente página para una visibilidad más clara.

Las actividades han sido codificadas mediante colores según los bloques a los que pertenecen, para una lectura visual más rápida de cada arco:

- Azul: hito inicial e hito final.
- Verde: etapas de diseño y planificación.
- Rojo: elaboración de plug-in básico.
- Púrpura: desarrollo de los distintos algoritmos de IA.
- Lila: limpieza de código.
- Amarillo: actividades definidas como *prescindibles* o altamente modificables.

Gantt Chart

Name	Begin date	End date
Hito Inicial	26/02/18	20/04/18
Alcance del proyecto y co...	26/02/18	06/03/18
Planificación temporal	07/03/18	12/03/18
Gestión económica y soste...	13/03/18	19/03/18
Presentación preliminar	20/03/18	02/04/18
Pliego de condiciones	03/04/18	09/04/18
Documento final	03/04/18	09/04/18
PowerPoint de presentaci...	03/04/18	09/04/18
Defensa oral	16/04/18	20/04/18
Diseño de la IA	12/02/18	11/05/18
Familiarización con las téc...	12/02/18	23/02/18
Análisis básico del entorno	26/02/18	06/03/18
Revisión del análisis	16/04/18	11/05/18
Diseño esquemático del c...	05/03/18	12/03/18
Revisión del diseño comp...	16/04/18	11/05/18
Familiarización con técnic...	23/04/18	27/04/18
Creación de Plug-in	15/03/18	02/04/18
Instal. y config. de las herr...	15/03/18	16/03/18
Familiarización con Minet...	16/03/18	19/03/18
Creación de "Mob Spawner"	20/03/18	26/03/18
Pruebas y correcciones	27/03/18	02/04/18
Plug-in funcional	03/04/18	03/04/18
Esqueleto de IA	09/04/18	13/04/18
Creación de IA de mov. al...	09/04/18	13/04/18
Desarrollo de IA	16/04/18	11/05/18
Creación y prueba: GA	16/04/18	27/04/18
Creación y prueba: RL	16/04/18	27/04/18
Creación y prueba: GNN	30/04/18	11/05/18
Creación y prueba: RLNN	30/04/18	11/05/18
Expansión de la IA	14/05/18	25/05/18
"Code clean-up"	28/05/18	01/06/18
Corrección de errores	04/06/18	15/06/18
Hito Final	15/06/18	29/06/18
Entrega de la memoria	15/06/18	15/06/18
Preparar la lectura	15/06/18	22/06/18
Lectura del TFG	25/06/18	29/06/18



4.1.4 Recursos necesarios

Para la definición de los recursos humanos necesarios del proyecto, se considera una plantilla mínima necesaria para el desarrollo de un proyecto similar, llevado a cabo por un equipo de desarrollo:

Recursos humanos

- Director de proyecto
- Diseñador
- Programador

En el caso de este proyecto, los roles de diseñador y programador se llevan a cabo por la misma persona.

Por otro lado, para el desarrollo de este proyecto hace falta una serie de recursos tanto de *hardware* como de *software*, pero sobre todo de *software*. En una primera consideración inicial, los recursos necesarios son los siguientes:

Recursos de hardware

- Ordenador personal estándar
- Tarjeta gráfica relativamente actual
  - Disponemos de una Nvidia GT950M

Recursos de software

- Windows 10, MacOS y/o GNU/Linux
  - Disponemos de Windows 10 y Arch Linux
- Entorno de desarrollo de Lua
- Editor de texto
- Control de versiones: Git
- Minetest
- *Framework* Torch para Lua
- Herramientas de documentación: LaTeX, Google Docs, ...

4.2 Gestión económica y sostenibilidad

4.2.1 Autoevaluación del dominio de sostenibilidad

El nivel de dominio de sostenibilidad es mejorable. Aunque se tienen los conocimientos suficientes como para saber distinguir y establecer qué elementos tienen mayor o menor impacto ambiental, y qué evitar utilizar o realizar para mantener un impacto menor, faltan conocimientos que permitan una estimación de la cantidad de impacto que aporta cada uno de los elementos del desarrollo de un proyecto. Así mismo, los conocimientos de impacto social y económico están bastante poco desarrollados, sobre todo por falta de experiencia en el campo.

Sí que es cierto que ha habido un aprendizaje previo al proyecto, pero se concentró la atención del impacto informático especialmente en el campo ambiental, dejando de lado tanto la parte económica como los posibles efectos en la sociedad. Sin embargo, tratándose la informática de un campo de trabajo donde el incremento de los costes económicos es más o menos correlativo con un incremento en costes ambientales, tomar una actitud frugal (reutilización de materiales, agrupación de recursos compartidos por parte de usos distintos para utilización simultánea, etc.) es una posición rápida para una primera minimización de ambos dos tipos de coste.

Queda, por tanto, profundizar en la estimación de costes ambientales y económicos, así como adquirir conocimiento sobre los casos en que un coste económico menor supone un impacto ambiental mayor, así como poder evaluar dichos costes en cantidades precisas y poder sopesar la valoración entre costes en casos como ése; y queda también estudiar los diversos posibles impactos sociales que un proyecto de desarrollo de software informático puede tener.

4.2.2 Gestión económica: presupuesto

Aunque este proyecto es dirigido por una sola persona que asume todos los roles del proyecto, más un director del proyecto, se toman en cuenta los distintos roles por separado y su coste estimado en circunstancias en las que el proyecto es llevado por un equipo, así como el coste de los materiales y las herramientas utilizadas.

La aproximación de la duración del trabajo en los cálculos de amortización de recursos no humanos es de 4 meses. La aproximación en los cálculos de recursos humanos, utilizando el diagrama de Gantt realizado en la planificación del proyecto e incluyendo el máximo de horas de las actividades de margen de error, es de 786 horas.

Costes: Recursos humanos

Los roles definidos son los siguientes: director del proyecto, diseñador, y desarrollador. Asumiendo los bloques de tareas del Diseño para el diseñador, y los de programación de la Plataforma para el desarrollador, las estimaciones de las horas son las siguientes:

Rol	**Coste/hora**	**Horas**	**Total**
Director de proyecto	50'00€/h	80 h	4.000'00€
Diseñador	35'00€/h	272 h	9.520'00€
Programador	30'00€/h	434 h	13.020'00€
Total		**786 h**	**26.540€**

Costes: Recursos de hardware

En el caso de hardware no hay mucho a considerar, ya que en el coste del propio portátil se incluyen las partes de hardware necesarias para el correcto funcionamiento de Minetest y de desarrollo de la IA (tarjeta gráfica y CPU).

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización**
Portátil ASUS R510V	807'99€	1	3 años	89'78€
Total	**807'99€**			**89'78€**

La vida útil estimada del portátil es la duración de la garantía para reparaciones. Como el modelo de portátil no incluye sistema operativo, el coste respectivo tendrá en cuenta por separado.

Costes: Recursos de software

En este caso consideramos todos los costes de uso y de compra de licencias necesarias para la utilización de las herramientas de software necesarias.

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización**
Windows 10 Pro N	259'00€	1	3 años	28'78€
Arch Linux	0'00€	1	3 años	0'00€
Minetest	0'00€	-	-	0'00€
Git Bash	0'00€	-	-	0'00€
Torch	0'00€	-	-	0'00€
Total	**259'00€**			**28'78€**

La vida útil de los sistemas operativos se ha estimado según la vida útil estimada del ordenador donde se instalan.

Costes indirectos

Los recursos materiales utilizados requieren un consumo constante de energía, que aportan unos gastos externos al proyecto pero ligados a éste, que deben tenerse en cuenta.

Producto	**Precio**	**Consumo**	**Coste aproximado**
Consumo eléctrico	0'12€/kWh	377'28 kWh	45'27€
Internet	26'90€/mes	4 meses	107'60€
Material de oficina	24'70€	-	24'70€
Total			**177'57€**

Coste total

Sumando todos los costes estimados obtenemos el presupuesto aproximado del proyecto.

Concepto	**Coste aproximado**
Recursos humanos	26.540'00€
Recursos de hardware	807'99€
Recursos de software	259'00€
Costes indirectos	177'57€
Total	**27.784'56€**

4.2.3 Control de gestión

Posibles imprevistos en la duración de las actividades que impliquen un retraso podrían significar el incremento de horas de trabajo de los miembros del equipo de desarrollo, suponiendo un incremento proporcional del coste. Sin embargo, la existencia de actividades suprimibles que permiten el realojamiento del tiempo invertido a otras actividades permite un cierto margen de actuación en caso de aparición de imprevistos sin que el tiempo total se vea afectado.

4.2.4 Sostenibilidad

Dimensión económica

Como se trata de un proyecto corto con unos recursos materiales de vida útil larga, las amortizaciones salen muy reducidas, por lo que debe tenerse en cuenta un uso futuro posterior al proyecto para dichos recursos para que el dinero invertido no quede desaprovechado.

En mi caso personal, los recursos materiales con coste ya se habían adquirido con anterioridad al proyecto, y el consumo eléctrico y de internet ya se realizan se desarrolle el proyecto o no, por lo que no hay ningún coste considerable a raíz de este trabajo.

Dimensión ambiental

En cuanto al impacto del desarrollo del proyecto, el coste es relativamente alto, ya que apenas se pueden amortizar los recursos materiales en el caso de que se adquieran con el único objetivo del desarrollo de este proyecto. Una solución sencilla es reutilizar recursos materiales de los que ya se dispongan (casi cualquier interacción con software requiere de usuarios de tener un ordenador personal, siendo Windows el sistema operativo más popular, y de pagar un consumo eléctrico y de internet asociado), o bien reutilizar los servicios adquiridos para este proyecto en futuros usos o incluso usos simultáneos relacionados con la informática.

Dimensión social

El mayor impacto social que puede tener este proyecto es en la comunidad de jugadores de videojuegos. No existe una necesidad palpable de inteligencias artificiales, porque en el mercado actual los desarrolladores aún dependen de técnicas clásicas de IA que, aunque inconsistentes, de comportamiento visiblemente mecánico, o difíciles de balancear, resultan familiares para los usuarios que, aunque los perciben como oponentes poco naturales, aceptan ésto como el coste a pagar por una mayor capacidad de predicción del oponente y la comodidad que proporciona la familiaridad. Además, con la proliferación de juegos multijugador y *on-line*, que precisamente enmascaran esta falta de IAs rivales decentes ofreciendo en su lugar rivales humanos, los problemas de IAs rivales se hacen incluso menos perceptibles.

Sin embargo, un enfoque desde la perspectiva del aprendizaje puede ofrecer una IA balanceada, adecuadamente imperfecta, para ofrecer un desafío interesante sin ser insuperable, al mismo tiempo que ofrece una experiencia fresca, al aportar nuevos comportamientos menos predecibles, elaborados de forma pseudoaleatoria, que pueden crear la ilusión de una personalidad de cara al jugador.

4.2.5 Matriz de sostenibilidad

	PPP	**Vida útil**	**Riesgos ambientales**
Ambiental	10	10	0
Económico	10	10	-5
Social	8	15	0
Rango de sostenibilidad	**58**		

4.3 Seguimiento del proyecto

4.3.1 Cambios de planificación

Han habido dos elementos importantes que dan lugar a los cambios más grandes del diagrama de Gantt planificado al seguido finalmente: se ha subestimado el tiempo requerido para la familiarización con la API de Minetest y el desarrollo robusto de la plataforma, y se ha sobrevalorado la posibilidad de actividades "extras" como la creación de más de un prototipo de IA. Se ha podido probar un algoritmo de *Q-Learning*, pero se ha determinado no factible, por lo que no se ha conservado para la versión final, y no se ha salido de la fase de investigación y diseño para las redes neuronales, siendo que el obstáculo del sistema de seguridad de *mods* de Minetest ha impedido el uso planeado de la librería *torch*, y el objetivo principal siendo la plataforma, implementar una red neuronal de 0 era insostenible ya que quitaba demasiado tiempo de desarrollo a la plataforma.

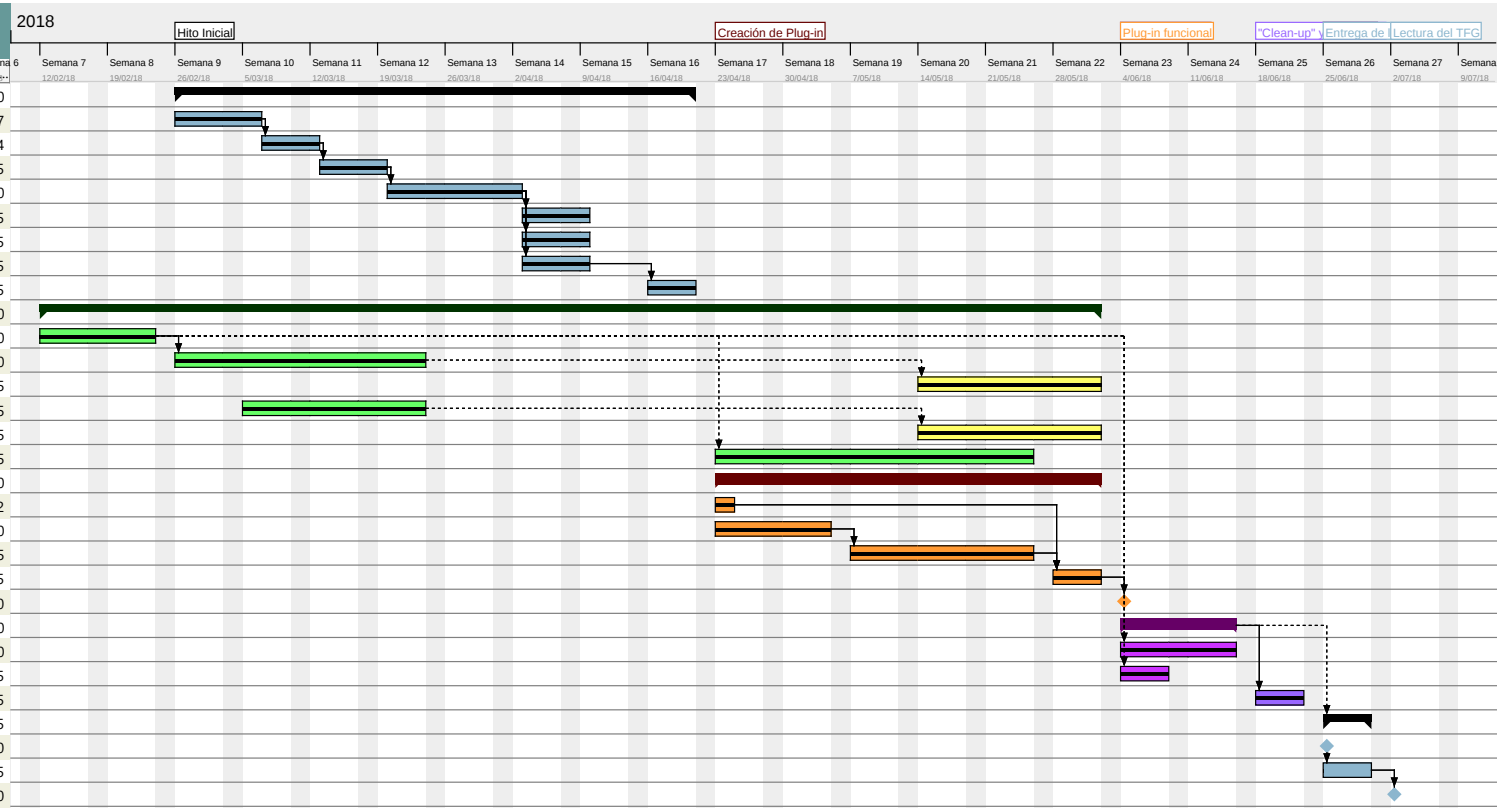
Sin embargo, la investigación de los algoritmos no dependientes de redes ha requerido menos tiempo del estimado, así como el desarrollo de la IA de algoritmo genético sin red neuronal, y el desarrollo del esqueleto de la IA se ha podido realizar simultáneamente con éste último.

Para poder adaptar el calendario a los imprevistos mencionados, se ha utilizado el tiempo extra de las actividades realizadas en menor tiempo del estimado, así como se han reducido la actividad final de corrección de errores, ya que la metodología IID lleva implícita prueba y correcciones al final de cada etapa de desarrollo, y la actividad de expansión de las IAs, ya que el objetivo del proyecto es el desarrollo de la plataforma, donde una expansión y mejora de las IAs se pueda hacer de forma indefinida más allá de la finalización del proyecto.

A continuación se incorpora un diagrama de Gannt actualizado.

Diagrama de Gantt

Nombre	Fecha de i...	Fecha de fin	Du...
Hito Inicial	26/02/18	20/04/18	40
Alcance del proyecto y contextualización	26/02/18	6/03/18	7
Planificación temporal	7/03/18	12/03/18	4
Gestión económica y sostenibilidad	13/03/18	19/03/18	5
Presentación preliminar	20/03/18	2/04/18	10
Pliego de condiciones	3/04/18	9/04/18	5
Documento final	3/04/18	9/04/18	5
PowerPoint de presentación oral	3/04/18	9/04/18	5
Defensa oral	16/04/18	20/04/18	5
Diseño de la IA	12/02/18	1/06/18	80
Familiarización con las técnicas de IA	12/02/18	23/02/18	10
Análisis básico del entorno	26/02/18	23/03/18	20
Revisión del análisis	14/05/18	1/06/18	15
Diseño del comportamiento	5/03/18	23/03/18	15
Revisión del diseño comportamiento	14/05/18	1/06/18	15
Familiarización con técnicas de NN	23/04/18	25/05/18	25
Creación de Plug-in	23/04/18	1/06/18	30
Instal. y config. de las herramientas	23/04/18	24/04/18	2
Familiarización con Minetest	23/04/18	4/05/18	10
Esqueleto modular de IA	7/05/18	25/05/18	15
Control de población	28/05/18	1/06/18	5
Plug-in funcional	4/06/18	4/06/18	0
Desarrollo de IA	4/06/18	15/06/18	10
Creación y prueba: GA	4/06/18	15/06/18	10
Creación y prueba: RL	4/06/18	8/06/18	5
"Clean-up" y corrección	18/06/18	22/06/18	5
Hito Final	25/06/18	29/06/18	5
Entrega de la memoria	25/06/18	25/06/18	0
Preparar la lectura	25/06/18	29/06/18	5
Lectura del TFG	2/07/18	2/07/18	0



El nuevo calendario modifica las estimaciones de tiempo y costes. Aquí se encuentran las estimaciones actualizadas:

Corrección: Estimación del tiempo

Fase del proyecto	Horas
Hito inicial	80
Diseño de la IA	200
Creación del Plug-in	180
Esqueleto de IA (incluido en la creación de plug-in)	*90*
Desarrollo de IA	120
"Clean-up" y corrección de errores	40
Total	**540**

No se cuenta el tiempo invertido en desarrollar el esqueleto de IA de la plataforma, ya que se reanaliza como parte del tiempo invertido en la creación del plug-in.

Corrección: Costes de recursos humanos

Rol	Coste/hora	Horas	Total
Director de proyecto	50'00€/h	80 h	4.000'00€
Diseñador	35'00€/h	200 h	7.000'00€
Programador	30'00€/h	340 h	10.200'00€
Total		**540 h**	**21.200'00€**

Costes totales

Concepto	Coste aproximado
Recursos humanos	21.200'00€
Recursos de hardware	807'99€
Recursos de software	259'00€
Costes indirectos	177'57€
Total	**22.444'56€**

4.3.2 Metodología y rigor

La metodología IID ha demostrado ser la adecuada para este tipo de desarrollo, ya que cada actividad se presta de forma natural a la subdivisión en 4 fases que la caracteriza, y lo que ha permitido amortiguar un posible impacto de readaptar el calendario.

4.3.3 Incorporación de conocimientos

Aunque no se ha hecho un análisis exhaustivo del coste computacional del código desarrollado, el conocimiento adquirido a lo largo de la carrera, así como un análisis rápido de las limitaciones de Lua 6, ha ayudado a crear directamente soluciones de coste bajo, y a desarrollar código evitando de forma preventiva una estructura que presenta errores comunes de redundancia o de consumo excesivo de recursos.

Los conocimientos previos de Ia han facilitado la investigación y diseño de los algoritmos a desarrollar.

4.3.4 Leyes y regulaciones

Como videojuego, Minetest puede ser sujeto a la regulación PEGI, y por tanto la plataforma desarrollada, como mod de Minetest puede ser sujeta a la misma regulación.

Como proyecto de código abierto, el juego es gratis y de descarga y uso libre; se encuentra distribuido bajo la licencia CC BY-SA 3.0 (Creative Commons Attribution-ShareAlike 3.0 Unported) 4. No se requiere que los mods sean también proyectos de código abierto, pero es considerado una buena práctica, y además permite y facilita futuras modificaciones y ampliaciones por parte de la comunidad, por lo que la plataforma desarrollada en este proyecto también será de código abierto.

5 Conclusiones y posibles mejoras

Se ha mencionado que para solucionar el problema de límite de memoria en la lectura de los archivos de persistencia, se ha separado la información de cada criatura en 2 archivos. Esto se puede repetir, para elaborar una IA compuesta por varias, cada una con un rol conreto (movimiento, búsqueda de comida, etc.), y una IA general que decida en cada turno el módulo a cargo del control de la criatura; es una primera idea de mejora fácilmente aplicable a lo que ya tenemos.

Utilizando esta idea, se puede despiezar la tabla del algoritmo de *Q-Learning* en varios archivos también, y así solventar el problema del exceso de espacio de memoria requerido.

Otra posible mejora del *mod* desarrollado es la de crear un segundo *mod* que incorpore un entorno no seguro (y medidas de seguridad extra para paliarlo), que incorpore las funcionalidades de redes neuronales utilizando librerías como *torch*.

Así y todo, el objetivo principal, que era la creación de una plataforma donde poder desarrollar IA evolutiva, junto con una IA de algoritmo genético construida sobre dicha plataforma que preuba su funcionalidad, ha sido alcanzado, y se encuentra en un estado que puede ser liberado al público de Minetest para uso por parte de la comunidad.

6 Bibliografía

References

[1] Jayesh Bapu Ahire. Artificial Neural Network: Some Misconceptions. *https://medium.com/swlh/artificial-neural-network-some-misconceptions-cb93e80b34bb*, 2018.

[2] Alex J. Champandard. Evolving with Creatures' AI: 15 Tricks to Mutate into Your Own Game. *http://aigamedev.com/open/review/creatures-ai/*, 2007.

[3] Alex J. Champandard. Top 10 Most Influential AI Games. *http://aigamedev.com/open/review/top-ai-games/*, 2007.

[4] Creative Commons. Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0). *https://creativecommons.org/licenses/by-sa/3.0/*, 2007.

[5] Klei Entertainment. Don't Starve. *https://www.klei.com/games/dont-starve*, 2013.

[6] Roberto Ierusalimschy. Lua Performance Tips. *https://www.lua.org/gems/sample.pdf*, 2008.

[7] Mojang. Minecraft. *https://minecraft.net/*, 2009.

[8] OpenAI. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *https://blog.openai.com/evolution-strategies/*, 2017.

[9] Re-Logic. Terraria. *https://terraria.org/*, 2011.

[10] The Minetest Team. Minetest. *https://www.minetest.net/*, 2011.

[11] Georgios N. Yannakakis and Julian Togelius. Game AI Book. *http://gameaibook.org/*, 2018.