Desarrollo de Comportamientos para Personajes No Jugadores basado en Planificación mediante Redes Jerárquicas de Tareas

Development of Behaviours for Non-Player Characters based on Planning through Hierarchical Task Networks

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Ismael Fernández Pereira

Director
Federico Peinado Gil

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de Comportamientos para Personajes No Jugadores basado en Planificación mediante Redes Jerárquicas de Tareas

Development of Behaviours for Non-Player Characters based on Planning through Hierarchical Task Networks

Trabajo de Fin de Grado en Desarrollo de Videojuegos
Departamento de Ingeniería de Software e Inteligencia Artificial

Autor
Ismael Fernández Pereira

Director
Federico Peinado Gil

Convocatoria: *Junio 2024*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

27 de mayo de 2024

Agradecimientos

Me gustaría mostrar mi más sincero agradecimiento a mi familia por su apoyo y al profesor Federico Peinado Gil por su desempeño como tutor de este trabajo de fin de grado.

Resumen

La Inteligencia Artificial para Videojuegos ofrece varios paradigmas para modelar la toma de decisiones de los personajes no jugadores.

Las máquinas de estados y los árboles de comportamiento son seguramente las soluciones más populares en la industria, aunque en determinadas situaciones pueden ser necesarios otros enfoques más proactivos, como la planificación automática.

En este trabajo proponemos un sistema para el desarrollo de comportamientos para personajes no jugadores que está basado en la planificación mediante redes jerárquicas de tareas. El objetivo es ofrecer un modelo flexible que permita a los desarrolladores de videojuegos utilizar el enfoque que quieran para sus personajes, ya sea más reactivo -en respuesta de los cambios que se producen en el mundo virtual- o más proactivo -en función de los propios objetivos e intenciones del personaje-.

Se ha desarrollado una implementación de este sistema en C# para Unity, integrándolo como una alternativa de inteligencia artificial para personajes en un escenario de ejemplo llamado Liquid Snake.

El funcionamiento se ha validado mediante la realización de pruebas en el laboratorio y también contrastando opiniones de personas que usan otras herramientas más clásicas.

Palabras clave

Árboles de Comportamiento, Planificación de Acciones Orientada a Objetivos, Inteligencia Artificial, Desarrollo de Videojuegos, Unity.

Abstract

Artificial Intelligence for Video Games offers several paradigms for modeling the decision making of non-player characters.

State machines and behavior trees are certainly the most popular solutions in the industry, although in certain situations other more proactive approaches, such as automatic planning, may be necessary.

In this paper we propose a system for developing behaviors for non-player characters that is based on planning using hierarchical task networks. The goal is to provide a flexible model that allows game developers to use the approach they want for their characters, either more reactive -in response to changes in the virtual world- or more proactive -depending on the character's own goals and intentions-.

An implementation of this system has been developed in C# for Unity, integrating it as an alternative artificial intelligence for characters in an example scenario called Liquid Snake.

The performance has been validated by testing in the lab and also by contrasting opinions of people using other more classical tools.

Keywords

Behavior Trees, Goal Oriented Action Planning, Artificial Intelligence, Video Game Development, Unity.

Índice

Índice de figuras

Capítulo 1

Introducción

La inteligencia artificial (IA) en los videojuegos ofrece muchas posibilidades para modelar el comportamiento y el proceso de toma de decisiones de los personajes no jugadores (NPCs, del inglés Non-Player Characters). Un objetivo habitual es desarrollar enemigos que ofrezcan un desafío a los jugadores, manteniendo el equilibrio entre diversión y dificultad.

Algunos de los paradigmas que más se han popularizado en la implementación de NPCs hasta el punto de convertirse en estándares de la industria son las máquinas de estado (FSMs, del inglés Finite State Machines), que definen el comportamiento en función del estado interno del personaje, y los árboles de comportamiento (BTs, del inglés Behaviour Trees), que lo definen a través de una serie de tareas interconectadas en forma de jerarquía.

Este trabajo explora el paradigma de planificación automática, en particular, el modelo denominado redes jerárquicas de tareas (HTN, del inglés Hierarchical Task Network).

El objetivo es conseguir desarrollar una herramienta HTN de uso libre para Unity y comparar su diseño, rendimiento y experiencia de usuario con los de otras técnicas convencionales más utilizadas.

1.1. Motivación

Durante el desarrollo de los videojuegos, la IA es uno de los aspectos que requiere más tiempo y dedicación. Para facilitar este proceso se han utilizado una variedad de paradigmas, entre los cuales han destacado los que ofrecían una mayor facilidad a la hora de implementarlos y usarlos. Aun así, con el paso del tiempo los comportamientos de los agentes han necesitado adquirir un nivel de complejidad y credibilidad cada vez mayor.

Para solucionar este problema se han usado paradigmas más avanzados,

como la planificación automática. El primer modelo en hacerse famoso fue GOAP (Goal Oriented Action Plannig), a partir del cual se han aplicado otros, algo menos conocidos, como HTN (Hierarchical Task Network). Este último es una alternativa a GOAP que ofrece mayor control a la hora de diseñar los planes que los agentes van a ejecutar, ayudando a definir mejor su comportamiento. A pesar de ser un modelo que proporciona características muy atractivas de cara a desarrollar videojuegos, apenas se encuentran herramientas, tanto gratuitas como de pago, integradas en los principales entornos de desarrollo como Unity o Unreal Engine.

1.1.1. Propósito

El objetivo de este trabajo es construir una herramienta que permita definir el comportamiento de múltiples NPCs usando el modelo HTN. La herramienta será desarrollada como una extensión del entorno de desarrollo de videojuegos más popular hoy en día, Unity, y se publicará como un recurso gratuito, ofreciendo así una alternativa gratuita de un modelo de IA no tan explotado en los videojuegos.

1.1.2. Alcance

El alcance esperado durante el desarrollo de la herramienta es el de construir una base sólida y funcional del modelo de redes jerárquicas de tareas.

Esto quiere decir que se podrá definir de forma correcta el comportamiento de uno o varios NPCs, que actuarán simultáneamente y de forma ininterrumpida.

También se dispondrá de un entorno de demostración de la herramienta que permita ilustrar la implementación y funcionamiento de la misma.

Los aspectos que no abarca este proyecto son el disponer de un sistema que permita depurar en profundidad el funcionamiento interno de la herramienta o el de los NPCs durante la ejecución, ni tampoco una interfaz de usuario visual que facilite el uso de la herramienta a la hora de implementar los distintos comportamientos HTN.

1.1.3. Asignaturas relacionadas

A continuación se van a listar una serie de asignaturas del grado de Desarrollo de Videojuegos que están relacionadas con el trabajo realizado durante el proyecto:

- **Metodologías ágiles de producción:** relacionada con la organización del proyecto y la gestión del repositorio.

- **Diseño de videojuegos:** relacionada con el diseño del entorno de demostración y el problema que el personaje no jugador debe resolver.
- **Motores de videojuegos:** relacionada con el uso del motor de videojuegos Unity.
- **Inteligencia artificial para videojuegos:** relacionada de forma directa con el proyecto, ya que consiste en el desarrollo de una IA mediante redes jerárquicas de tareas.
- **Proyectos I, II y III:** relacionado con todo lo necesario para realizar un proyecto de gran alcance en un espacio prolongado de tiempo.

1.2. Estructura del trabajo

La estructura de este trabajo se compone de los siguientes apartados:

- **Capítulo 1: Introducción.** Se presentan la motivación, el propósito y el alcance que tiene este trabajo y las asignaturas del grado relacionadas con este proyecto.
- **Capítulo 2: Estado de la cuestión.** Se revisa el estado de la cuestión en materia de la IA usada dentro de los videojuegos o como técnica para ayudar en su desarrollo. Además, se estudian los distintos modelos que existen y algunos trabajos similares al proyecto.
- **Capítulo 3: Objetivos y requisitos.** Se presentan los objetivos principales del trabajo y se hace una especificación de todos los requisitos necesarios para poder desarrollar la herramienta propuesta.
- **Capítulo 4: Metodología.** Se explica la metodología usada para desarrollar el proyecto y qué herramientas concretas se han utilizado en el proceso.
- **Capítulo 5: Desarrollo.** Se realiza una explicación detallada del diseño e implementación del proyecto.
- **Capítulo 6: Resultados.** Se comienza con una explicación de las pruebas. A continuación, se comentan los resultados obtenidos de las pruebas con usuarios reales, realizadas para validar el funcionamiento de la herramienta y evaluar su utilidad para desarrolladores reales.
- **Capítulo 7: Conclusiones.** Se muestran las conclusiones generadas como producto de la comparación de la versión final de la herramienta con los objetivos definidos al principio, para validar si estos se han cumplido.

Capítulo 2

Estado de la cuestión

La Inteligencia Artificial en los videojuegos se suele utilizar para crear enemigos que generan un reto satisfactorio que el jugador pueda disfrutar superando. Con el tiempo, se han explorado otros usos como personajes acompañantes, que asisten al jugador a la hora de superar puzzles o combates, e incluso algunos relacionados con el ámbito narrativo, que permiten variaciones significativas en un entorno controlado y enriquecen la experiencia del jugador (Riedl et al. (2011)).

2.1. Conceptos

En esta sección se van a explorar algunos de los paradigmas más usados a la hora de afrontar la implementación de la Inteligencia Artificial en videojuegos, repasando los modelos más populares y mencionando algunos ejemplos de uso.

2.1.1. Máquinas de estados

Las máquinas de estados finitos (FSMs, del inglés Finite State Machines), pertenecen a un modelo computacional que define el comportamiento de un agente dividiéndolo en los diferentes estados en los que puede encontrarse en cualquier momento (Graham (2017)).

Además del conjunto de estados que componen al agente, las máquinas de estados tienen un conjunto de transiciones que sirven para definir las condiciones que hacen que se pase de un estado a otro. Las transiciones son dependientes del estado en que se encuentran y solo se procesan las que pertenecen al estado actual del agente.

Una de las ventajas de las máquinas de estado es que son a la vez una

herramienta de diseño e implementación, ya que hay que definir los estados y las transiciones antes de plasmarlos en el código. Por ello, es uno de los modelos más utilizados, sobre todo en la resolución de problemas sencillos.

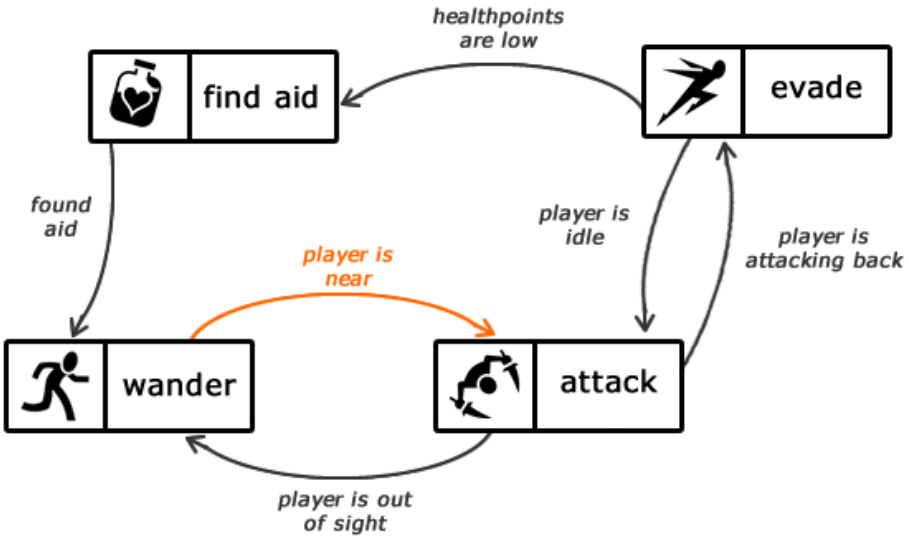

Figura 2.1: Ejemplo de máquina de estados.

2.1.2. Árboles de comportamiento

Los árboles de comportamiento (BTs, del inglés Behaviour Trees), definen el comportamiento de un agente a través de una serie de nodos interconectados en forma de jerarquía (Champandard y Dunstan (2019)), permitiendo resolver problemas más complejos que podrían suponer un problema para las máquinas de estado.

Todos los nodos están anidados bajo el nodo raíz, que se encarga de controlar en árbol durante la ejecución. Además de la raíz existen otros dos tipos de nodos:

- **Nodos ejecutores:** Los nodos ejecutores representan las acciones que el agente puede realizar. Durante la ejecución, se encargan de enviar una señal que indica el estado en el que se encuentran: Ejecutando, Éxito o Fallo.
- **Nodos controladores:** Los nodos controladores siempre tienen nodos hijos, ya que se encargan de controlar el orden de ejecución dentro del árbol. Dentro de los nodos controladores existen dos tipos: el selector, que ejecuta los nodos hijos de izquierda a derecha y termina cuando

uno de ellos devuelve éxito, y el secuencia, que ejecuta todos sus nodos hijos de izquierda a derecha hasta que uno de ellos devuelve fallo.

Algunas de las ventajas de los árboles de comportamiento son la capacidad de representar comportamientos muy complejos de forma global, reduciendo los problemas de diseño, y la capacidad de reutilizar ciertas partes de los árboles gracias a que son modulares.

Los casos de uso más conocidos son los videojuegos Halo 2, Halo 3, Bioshock y Spore (Agis et al. (2020)).

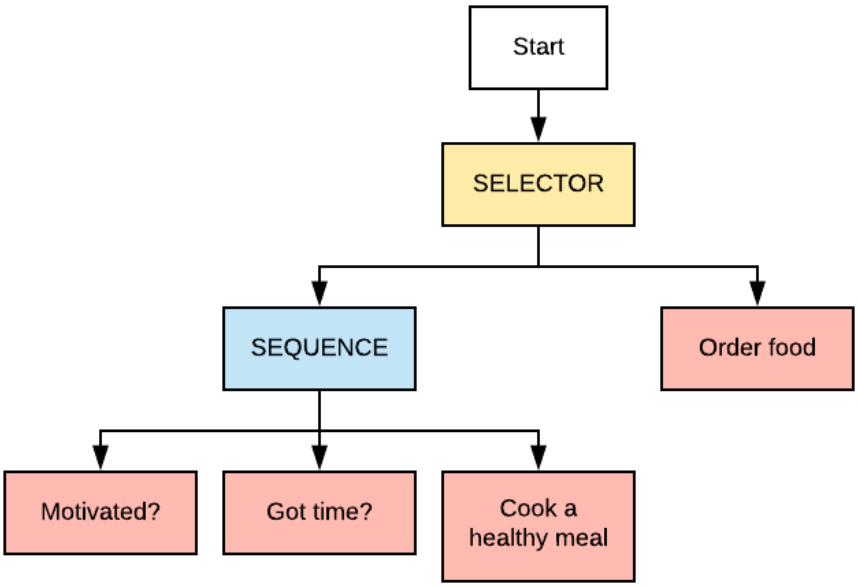

Figura 2.2: Ejemplo de árbol de comportamiento.

2.1.3. GOAP

La planificación de acción orientada a objetivos (GOAP, del inglés Goal Oriented Action Plannig) es uno de los primeros modelos basados en la planificación automática que se popularizó en el ámbito del desarrollo de videojuegos. Fue concebido durante la creación de F.E.A.R, un videojuego que destacó por el comportamiento de los enemigos, ya que reaccionaban a las acciones del jugador de forma muy realista (Orkin (2006)).

GOAP está compuesto por una serie de elementos que le dan forma y permiten definir el comportamiento de los agentes:

- **Estado del mundo:** El estado del mundo es un conjunto de variables que sirve para almacenar información sobre el estado en el que se encuentra el entorno por el que navega el agente.

- **Acciones:** Las acciones del agente se representan de manera individual y son independientes entre ellas. Todas poseen una serie de condiciones para ser ejecutadas y una serie de efectos que aplican al terminarse.

- **Objetivos:** Los objetivos son condiciones que el agente debe satisfacer para completar su comportamiento. Ya que los agentes pueden tener más de un objetivo, estos suelen estar ordenados por prioridad.

- **Planificador:** El planificador se encarga de elegir que acciones se tienen que ejecutar para completar los objetivos del agente y se encargan de forma una lista de acciones conectadas y ordenadas. Siempre que sea necesario, el planificador hará nuevos planes hasta que el agente alcance sus objetivos.

Las ventajas que ofrece la planificación automática se concentran en la capacidad que los agentes tienen de actuar de forma que predicen ciertos acontecimientos a través del plan, consiguiendo comportamientos mucho más realistas que con otros modelos.

Desde F.E.A.R, GOAP ha sido implementado en otros grandes proyectos como Tomb Raider o La Tierra Media: Sombras de Mordor (Chris Conway y Jacopin (2015)).

Figura 2.3: Captura del videojuego F.E.A.R.

2.1.4. HTN

Las redes jerárquicas de tareas (HTN, del inglés Hierarchical Task Network) son otro modelo de planificación automática que se diferencia de GOAP por permitir la introducción de jerarquías dentro de los planes.

Estas jerarquías se crean a través de agrupaciones de tareas simples o primitivas en lo que se conoce como tareas compuestas. Estas agrupaciones están construidas por los diseñadores del juego y ayudan a dar un control mayor a la hora de definir el comportamiento de los agentes, ya que obligan al plan a procesar conjuntos de tareas mediante una descomposición (Humphreys (2019)).

Al igual que GOAP, existen los elementos mencionados en la sección anterior: estado del mundo, acciones con sus condiciones y efectos, objetivos definidos a través de las jerarquías y el planificador.

Algunos ejemplos de videojuegos que utilizan HTN son Killzone 3 y Transformers 3: Fall of Cybertron (Soemers y Winands (2016)).

Figura 2.4: Captura del videojuego Transformers 3: Fall of Cybertron.

2.2. Herramientas similares

Actualmente, los entornos integrados más populares, como Unreal Engine o Unity, ofrecen múltiples herramientas de planificación automática gratuitas e incluso de pago a los desarrolladores. El modelo más común es GOAP, aunque existen algunas implementaciones de HTN. Los siguientes apartados describen algunas de estas herramientas, que comparten similitudes con el

objetivo que se pretende alcanzar con este proyecto.

2.2.1. S GOAP

Este recurso, disponible en la tienda de Unity (Studios (2020)), ofrece una herramienta de Inteligencia Artificial basada en GOAP que implementa una estructura fácil de usar sobre la que construir proyectos, además de funciones que facilitan el proceso de depuración.

Como indica el paradigma GOAP, se dispone de un estado del mundo con toda la información necesaria para los agentes, sus objetivos y sus acciones, un planificador, los agentes que actúan sobre el estado del mundo y lo modifican, y las acciones y sensores que utilizan en sus comportamientos.

El usuario tiene a su disposición un componente con el que crear a los agentes. Este componente permite determinar los objetivos que pueden tener a lo largo de la ejecución y la prioridad en la que están ordenados. El siguiente paso a seguir consiste en determinar las acciones que puede realizar cada agente. Las acciones son otro componente que se puede configurar especificando el coste de la acción y añadiendo sus prerrequisitos y consecuencias. Estos últimos sirven para conectar unas acciones con otras o con alguno de los objetivos del agente. De esta forma se construye el esquema que el planificador utiliza para darle su comportamiento al agente durante la ejecución.

Todas las características mencionadas anteriormente se pueden modificar desde el propio editor de Unity en cada uno de sus respectivos componentes y el estado del mundo se configura automáticamente al hacerlo. Aun así, la lógica que define a las acciones recae en la implementación por parte del usuario, que sólo tiene que modificar lo que la herramienta ofrece como plantilla.

La otra gran parte de esta herramienta es la funcionalidad proporcionada para facilitar el proceso de depuración. Existe una ventana específica para este trabajo y está dividida en tres apartados.

El primer apartado es una descripción general que muestra los objetivos y las acciones del agente seleccionado en ese momento. El segundo apartado permite simular una planificación sin ejecutar el proyecto, mostrando por orden de prioridad los objetivos que se intentarían cumplir y, en cada uno de ellos, las distintas secuencias de acciones que cumplirían el objetivo, ordenadas de menor a mayor coste. El tercer y último apartado se utiliza durante la ejecución y muestra el objetivo actual del agente seleccionado en ese momento, el plan que está siguiendo para cumplirlo y la lista de acciones que es capaz de realizar, con sus datos (el coste, si es una acción disponible, si está en cooldown, etc.) actualizándose en tiempo real.

Está parte de la herramienta es lo que la hace destacar sobre otras implementaciones, ya que ofrece una mayor accesibilidad a la hora de su uso y entendimiento por parte del usuario. También es una función muy útil cuando es necesario optimizar la estructura creada para el proyecto.

2.2.2. Badger HTN

Este recurso, disponible en la tienda de Unity (Tools (2023)), ofrece una herramienta de Inteligencia Artificial basada en HTN que implementa una estructura simple sobre la que construir el proyecto, además de una ventana de depuración.

Siguiendo el modelo HTN, se dispone de un estado del mundo con toda la información necesaria para los agentes, una red jerárquica con sus objetivos y acciones, un planificador y los agentes que actúan sobre el estado del mundo y lo modifican.

El usuario tiene a su disposición un *scriptable object* llamado pizarra en el que debe configurar el estado del mundo, añadiendo todas las variables que luego se usarán como prerrequisitos y consecuencias en las acciones. La pizarra se añadirá después a otro *scriptable object* llamado árbol, que contiene la red jerárquica donde se define el comportamiento del agente.

Para crear un agente existe un componente al que se asigna un árbol. La red jerárquica se puede configurar en una ventana que permite crear y vincular nodos, conectando acciones y objetivos. La prioridad de los objetivos se establece siguiendo una búsqueda en anchura. Las acciones se pueden configurar especificando si son independientes o parte de una secuencia, además de poder añadir sus prerrequisitos y consecuencias, usando la información disponible en la pizarra. De esta forma se construye el esquema que el planificador utiliza para darle su comportamiento al agente durante la ejecución.

Cada nodo de la red jerárquica es una acción y debe crearse un componente nuevo con toda la lógica que se necesita para definir su comportamiento. La implementación recae en el usuario, que tiene que heredar de los componentes que la herramienta ofrece como plantilla.

La ventana de depuración que se incluye con la herramienta es la misma en la que se construyen las redes jerárquicas. Durante la ejecución, permite observar como las variables del agente cambian en tiempo real y se delinean de color verde los nodos que se están ejecutando, mostrando el plan con sus acciones y el objetivo actual.

Capítulo 3

Objetivos y requisitos

El propósito de este proyecto es desarrollar un sistema que permita implementar el comportamiento para personajes no jugadores basado en planificación mediante redes jerárquicas de tareas. Para ello se va a desarrollar el planificador correspondiente, usando el algoritmo de búsqueda en anchura o BFS (del inglés Breadth First Search) mediante código C# para el entorno de desarrollo Unity.

3.1. Objetivos

En esta sección se especifican los objetivos principales que se desean cumplir a lo largo del proyecto:

1. Diseñar un algoritmo de planificación basado en el algoritmo de búsqueda en anchura.
2. Diseñar un conjunto de tareas primitivas y compuestas, definiendo un proceso de descomposición de tareas que use los operadores de selección y secuencia.
3. Diseñar un almacén de variables de cualquier tipo donde guardar y modificar la representación del estado del mundo, incluyendo las condiciones y los efectos de las tareas.
4. Desarrollar un escenario de ejemplo en Unity que sirva como demostración del sistema creado, donde se defina el comportamiento de un personaje controlado por la máquina resolviendo un problema complejo.
5. Comparar la eficacia de este sistema frente a otros paradigmas de Inteligencia Artificial para Videojuegos usando soluciones al mismo problema pero creadas con árboles de comportamiento.

3.2. Requisitos

A continuación se listarán los requisitos que la herramienta que se desea desarrollar deberá cumplir al terminar el proyecto:

- La base de la herramienta será implementada en C# sin dependencias con la aplicación en la que se desarrolla.
- La implementación de la herramienta utilizará funcionalidad de Unity.
- Se podrán definir distintos estados del mundo.
- Las variables del estado del mundo serán de tipo *object* para poder utilizar cualquier tipo de dato.
- Las variables se almacenarán en un diccionario clave valor para realizar las comprobaciones más rápidamente.
- Se podrán definir tareas simples y tareas compuestas.
- Se podrán definir tareas compuestas de tipo selector y tipo secuencia.
- Las condiciones y los efectos de las tareas serán de tipo *bool* para realizar las comprobaciones más rápidamente.
- Se podrán gestionar varios agentes al mismo tiempo.
- El planificador generará planes válidos constantemente para todos los agentes.
- Cuando un plan falle o se acabe, el planificador realizará un nuevo plan inmediatamente.
- La depuración del planificador se realizará a través de la consola de Unity.

Capítulo 4

Metodología

En este capítulo se describe la metodología utilizada durante el desarrollo. Los temas que se abordan son la planificación general del proyecto, sus distintas iteraciones a lo largo del tiempo y las herramientas utilizadas para alcanzar los objetivos planteados.

4.1. Planificación general

Para organizar el proyecto se ha utilizado la metodología ágil de desarrollo llamada Scrum, aprendida durante la asignatura *Metodologías ágiles de producción* en el primer año del Grado en Desarrollo de Videojuegos. La decisión de usar Scrum se ha tomado teniendo en cuenta su familiaridad, al haberse usado anteriormente en las asignaturas Proyecto 1, Proyecto 2 y Proyecto 3.

Scrum es un proceso que consiste en priorizar el desarrollo incremental sobre la planificación total del proyecto, realizando iteraciones periódicas en las que se consulta con el cliente el avance del producto. Esto permite tener una mayor flexibilidad frente a cambios y mejora la productividad, ya que se priorizan las tareas más importantes y se reciben comentarios directos de parte del cliente.

El curso se ha dividido en tres etapas de desarrollo, llamadas hitos, cuya duración se ha comprendido entre: el comienzo del curso y las vacaciones de Navidad, la vuelta de Navidad y las vacaciones de Semana Santa y la vuelta de Semana Santa y el final del curso. Durante cada hito, se han realizado reuniones con el tutor cada dos semanas. Estos periodos de tiempo más cortos son conocidos como *sprints* y sirven para mostrar el progreso del proyecto y comentar los pasos a seguir.

A cada uno de los hitos se ha asignado un objetivo principal a cumplir,

para facilitar la organización y el progreso del trabajo:

- **Hito de Navidad**: El objetivo de este hito es el estudio e investigación de la planificación automática y, en concreto, su aplicación en los videojuegos. La idea inicial era trabajar sobre el paradigma conocido como GOAP (Goal Oriented Action Plannig) aunque se termina orientando el trabajo al paradigma HTN (Hierarchical Task Network).
- **Hito de Semana Santa**: El objetivo de este hito es el diseño y la implementación de una herramienta HTN para definir comportamientos de NPCs en el entorno de desarrollo Unity. Durante su desarrollo, se decide realizar una demostración de la herramienta a través de un prototipo de videojuego.
- **Hito Final**: El objetivo de este hito es la realización de experimentos y la redacción de la memoria del proyecto. Se decide que los experimentos se hagan a través de una comparación entre la solución a un mismo problema de varios alumnos utilizando técnicas convencionales, como los árboles de comportamiento, frente al autor de este trabajo utilizando su herramienta HTN.

Todas las tareas realizadas durante el proyecto han sido documentadas a través un proyecto de Pivotal Tracker. Las tareas han sido puntuadas en base al número de horas de trabajo realizadas, sumando un total aproximado de 90 horas a lo largo de todo el curso.

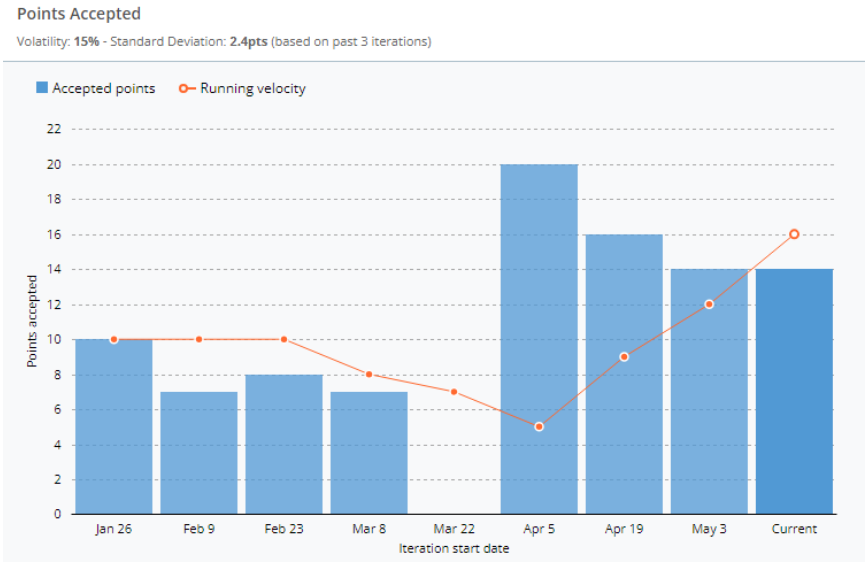

Figura 4.1: Puntos e iteraciones.

4.2. Iteraciones

Durante el desarrollo de la herramienta HTN no se realizaron grandes cambios, ya que se trata del grueso del proyecto y la decisión de usar ese paradigma fue tomada en el primer hito. Por otro lado, el prototipo usado para demostrar el funcionamiento de la herramienta ha sufrido variaciones a lo largo de distintas etapas.

4.2.1. Primera etapa

La primera concepción del prototipo se basada en el videojuego *Payday 2*, cuyo objetivo es la infiltración y posterior robo de un banco. La idea era construir una escena simplificada en Unity que recrease un robo. En ella, el NPC tendría que alcanzar la sala en la que se encuentra el dinero sin ser detectado por los guardias de seguridad patrullando y las cámaras. Tanto su comportamiento como el de los guardias se definiría usando HTN.

4.2.2. Segunda etapa

La segunda concepción del prototipo surge a raíz de la necesidad de mejorar el carácter visual para obtener un aspecto de mayor acabado. Tras investigar los distintos proyectos de Unity Learn se da con uno llamado John Lemon's Haunted Jaunt, que recrea una situación muy similar a la descrita en la anterior etapa. En este caso, el NPC debe avanzar a través de una mansión encantada evitando a los fantasmas que patrullan y unas estatuas encantadas hasta llegar a la salida. Se pasa entonces a utilizar este entorno y definir el comportamiento tanto del jugador como de los fantasmas usando HTN.

4.2.3. Última etapa

La última concepción del prototipo tuvo lugar poco después de la segunda, tras la confirmación de la posibilidad de usar el entorno Liquid Snake. Se realizó el cambio ya que, a parte de ofrecer un acabado visual a la altura, se iba a utilizar también en la práctica final de la asignatura de *Inteligencia Artificial para Videojuegos* de tercero del Grado en Desarrollo de Videojuegos. Esto abría la posibilidad de afrontar un problema más complejo que en los prototipos anteriores, además de realizar la misma solución que los alumnos y así, poder facilitar la realización de experimentos comparando las mejores implementaciones de los alumnos usando árboles de comportamiento frente la implementación de este proyecto usando HTN.

4.3. Herramientas utilizadas

A continuación se nombran todas las herramientas utilizadas durante el proyecto, tanto para su desarrollo como para su organización.

4.3.1. Github

Es una plataforma de desarrollo que permite almacenar proyectos utilizando un sistema de control de versiones Git. Se ha utilizado para subir el proyecto y realizar cambios a lo largo del desarrollo.

El enlace del repositorio es el siguiente: `https://github.com/Narratech/TFG-Fernandez`.

4.3.2. Slack

Es un servicio de comunicación que dispone de múltiples chats para realizar consultas al director del trabajo y poder organizar reuniones.

4.3.3. Pivotal Tracker

Es un servicio de organización de proyectos que permite gestionar los *sprints* y asignar tareas aplicando la metodología ágil deseada.

El enlace del proyecto es el siguiente: `https://www.pivotaltracker.com/n/projects/2689599`.

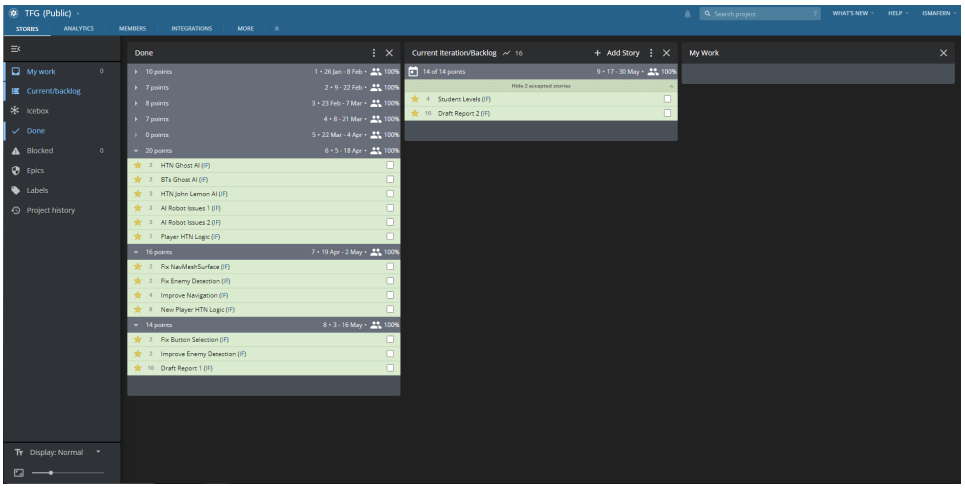

Figura 4.2: Proyecto de Pivotal Tracker.

4.3.4. Drive

Es un servicio de alojamiento de archivos que permite compartir distintos tipos de documentos con varias personas. Se ha utilizado para almacenar documentos y compartir archivos con el director del trabajo.

4.3.5. Overleaf

Es un editor de LaTex online que permite crear y editar documentos. Se ha utilizado para redactar la memoria del proyecto.

Capítulo 5

Desarrollo

En este capítulo se van a describir todos los aspectos relacionados con el desarrollo de la herramienta HTN y el entorno de demostración llamado Liquid Snake. Los contenidos que se cubrirán son el diseño de la herramienta y la implementación de la misma en el entorno de demostración.

5.1. Diseño

En está sección se describen en detalle las distintas clases que componen la herramienta. Todas ellas representan uno de los elementos que forma parte de las redes jerárquicas de tareas.

5.1.1. Estado del mundo

Clase responsable de almacenar variables que contienen información importante sobre el estado actual del juego. Estas variables se utilizan para definir las condiciones de las tareas y, si es necesario, pasar datos para construir la lógica de los operadores.

Está formado por un diccionario clave valor en el que las claves se almacenan en variables de tipo *string* y los valores en variables de tipo *object*, de forma que se pueda utilizar cualquier tipo de dato para pasar información.

Contiene métodos que permiten añadir propiedades al estado del mundo, cambiar el valor a una propiedad ya existente y obtener el valor de una propiedad ya existente.

5.1.2. Condición

Clase que define el requisito que una tarea necesita cumplir antes de poder realizarse. Se comprueba a través de las variables del estado del mundo.

Está formada por un identificador del mismo tipo que las claves del diccionario que contiene el estado del mundo, *string*, y un valor de tipo *bool*.

Su único método realiza la comprobación de que el valor de la condición es el mismo que el valor de la variable del estado del mundo que recibe a través de los parámetros.

5.1.3. Efecto

Clase que define la modificación del valor de una de las variables del estado del mundo, que se aplica cuando la tarea a la que pertenece se completa con éxito.

Está formada por un identificador del mismo tipo que las claves del diccionario que contiene el estado del mundo, *string*, y un valor de tipo *bool*.

Su único método realiza la modificación del valor de la variable del estado del mundo que recibe a través de los parámetros.

5.1.4. Tarea

Clase de la que heredan los dos tipos de tarea, primitiva y compuesta, y define los elementos que ambas comparten.

Está formada por un enumerado que indica el tipo de la tarea, una variable de tipo tarea compuesta que almacena, en caso de ser cierto, la tarea compuesta que la contiene y una lista de condiciones que definen los requisitos que debe cumplir para poder realizarse.

Contiene métodos que permiten añadir condiciones a la lista de condiciones y comprobar que la tarea es válida, validando una por una las condiciones de la lista con el estado del mundo que reciben a través de los parámetros.

5.1.5. Tarea primitiva

Clase que define una acción simple. Hereda de la clase Tarea y, por lo tanto, contiene los elementos definidos anteriormente.

Está formada por una lista de efectos que modifican el estado del mundo después de su ejecución y un operador que define la lógica de la acción representada por la tarea.

Contiene métodos que permiten añadir efectos a la lista de efectos, asignar el operador que la tarea va a ejecutar y aplicar las modificaciones de los efectos, uno por uno, en las variables del estado del mundo que reciben a través de los parámetros.

5.1.6. Tarea compuesta

Clase que define una acción compleja. Hereda de la clase Tarea y, por lo tanto, contiene los elementos definidos anteriormente.

A diferencia de las tareas primitivas, no posee una lista de efectos o un operador porque está formada por una lista de tareas, tanto primitivas como compuestas, que, en el caso de las primitivas, ya los contienen. Además, posee un enumerado que indica el tipo de descomposición en el que se procesan las tareas de la lista, que puede ser mediante selección o como una secuencia.

Su único método permite añadir tareas de cualquiera de los dos tipos a la lista de tareas.

5.1.7. Operador

Clase de la que heredan todos los operadores que el usuario programe al usar la herramienta. Define la lógica de la acción ejecutada por la tarea que lo contiene cuando cumple todas sus condiciones.

Está formada por un enumerado que indica el estado en que se encuentra la acción (Ejecutándose, Éxito o Fracaso) y el estado del mundo actual, que utiliza, si es necesario, para construir la lógica de la acción.

Contiene métodos vacíos que se heredan y definen en los nuevos operadores creados por el usuario. Estos son: un método que se llama en cada tick de la ejecución y contiene toda la lógica de la acción, un método que para la acción y un método que reinicia la acción.

5.1.8. Planificador

Clase responsable de construir el plan que el agente va a realizar. Este plan está formado a partir de una lista de tareas primitivas que cumplen sus condiciones y pueden ejecutarse en secuencia.

Está formada por una tarea raíz a partir de la cual empieza el proceso de descomposición, el estado del mundo actual, una cola de tareas que se necesitan procesar y una lista enlazada que almacena el plan que se va a ejecutar.

Contiene dos métodos principales, el método que crea el plan y el método que ejecuta el plan:

- **Crear el plan:** Lo primero que se hace es crear una copia del estado del mundo actual que se utilizará durante la creación del plan para simular que las tareas que se van procesando cumplen las condiciones necesarias para ejecutarse y después aplicar sus efectos en dicha copia. Acto seguido, se añade la raíz a la cola de tareas que se necesitan procesar y comienza la descomposición. Cada tarea en la cola pasa por el mismo proceso del bucle hasta que la cola se vacía y se termina obteniendo un plan. La primera comprobación es el tipo de la tarea, primitiva o compuesta. En el caso de ser una tarea compuesta, se validan sus condiciones y se comprueba el tipo de descomposición. Si es de tipo selección se itera sobre la lista de tareas y se añade a la cola la primera que valide sus condiciones. Si es de tipo secuencia se añaden todas directamente a la cola. En el caso de ser una tarea primitiva, se validan sus condiciones y, en caso afirmativo, se aplican sus efectos a la copia del estado del mundo y se añade la tarea a la lista enlazada que almacena el plan. En caso negativo, se comprueba si dentro de la lista enlazada hay una o varias tareas con el mismo padre que la que acaba de fallar las condiciones para quitarlas. Esto se hace porque significa que se estaba procesando una tarea perteneciente a una tarea compuesta de tipo secuencia y ha fallado, haciendo toda la secuencia invalida. Al acabar el plan, se devuelve la lista enlazada con todas las tareas primitivas en el orden en que se tienen que ejecutar.

- **Ejecutar el plan:** Este método se llama en cada tick de la ejecución y realiza varias comprobaciones. Si la lista enlazada con el plan está vacía significa que el plan ha acabado o nunca llego a crearse, entonces llama a la creación de uno nuevo. En caso contrario, selecciona la primera tarea de la lista enlazada y valida sus condiciones. Si falla, llama a la creación de un nuevo plan. Si no, llama al método que ejecuta la lógica del operador de dicha tarea. Si durante la ejecución de la tarea el operador falla se llama a la creación de un nuevo plan. Si la tarea termina su ejecución con éxito, se aplican sus efectos al estado del mundo y se quita de la lista enlazada para realizar las comprobaciones con la siguiente tarea del plan.

5.2. Implementación

En esta sección se especifica todo lo relacionado con la implementación de la herramienta HTN en el entorno de demostración elegido y los operadores desarrollados para construir el comportamiento final del agente.

5.2.1. Entorno de demostración

Además del desarrollo de la herramienta HTN, se consideró necesario implementar un agente que utilizase las redes jerárquicas de tareas para definir su comportamiento. La idea inicial consistía en la creación de un prototipo de videojuego que se usase como demostración.

Tras varias iteraciones se decidió utilizar un entorno llamado Liquid Snake, ya que proporciona un escenario junto a un personaje no jugador y enemigos. Este entorno se usaría también en la última práctica de la asignatura de *Inteligencia artificial para videojuegos* del tercer año del grado de Desarrollo de Videojuegos, proporcionando un problema específico a los alumnos que debía ser resuelto mediante árboles de comportamiento. Al abordar el mismo problema utilizando HTN se obtiene, además de un ejemplo de funcionamiento de la herramienta, una comparación directa de las soluciones conseguidas con distintos paradigmas.

El problema consiste en definir el comportamiento del personaje no jugador de forma que sea capaz de avanzar por una serie de habitaciones conectadas hasta alcanzar la meta. Durante el camino, el personaje se encontrará una serie de puertas que deberá abrir pulsando los botones correspondientes mientras evita ser detectado por los enemigos patrullando el escenario. Estos enemigos perseguirán y atacarán al personaje una vez lo detectan, reduciendo su vida. Por lo tanto, el personaje deberá evitar la muerte escondiéndose en zonas que lo ocultan de los enemigos y recogiendo objetos que recuperan su vida.

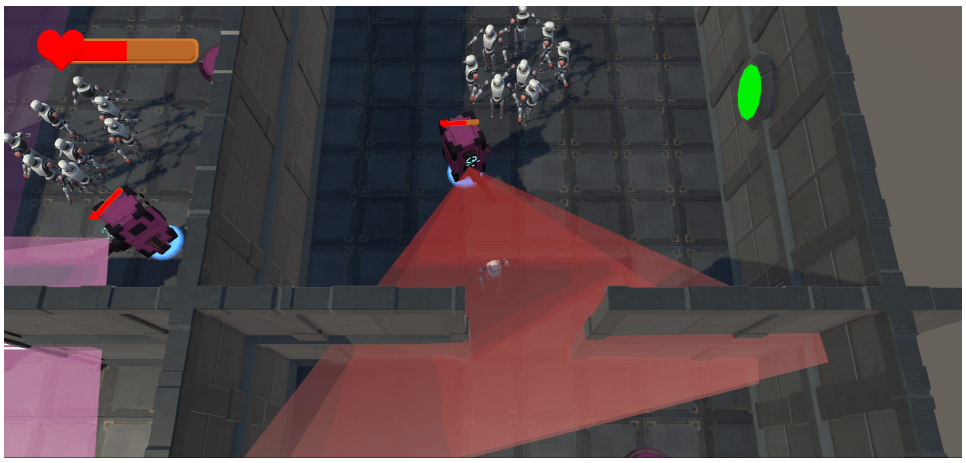

Figura 5.1: Captura del entorno Liquid Snake.

5.2.2. Comportamiento del agente HTN

Para poder resolver el problema descrito anteriormente se implementaron una serie de operadores que contienen la lógica de las acciones representadas por las tareas primitivas que formarán parte de la red jerárquica que define el comportamiento del agente.

- **MoveTo:** Acción que mueve al personaje a una posición establecida en los parámetros del operador. Utiliza el sistema de navegación de Unity para mover a un *NavMeshAgent* por una *NavMeshSurface* siempre que sea posible alcanzar el destino seleccionado.
- **SelectAccess:** Acción que proporciona la posición obtenida tras cruzar una de las salidas de la habitación actual. La salida se elige de forma aleatoria priorizando las que den acceso a una habitación no visitada. Si una de las salidas es el destino final del nivel, se elige por encima de todas las demás.
- **SelectButton:** Acción que proporciona la posición del botón asignado a la puerta que se está intentando cruzar en ese momento. La puerta pertenece a la salida seleccionada previamente.
- **ClosestObject:** Acción que proporciona la posición del objeto más cercano al jugador. El objeto seleccionado pertenece a una lista que se rellena a medida que son descubiertos durante la exploración del nivel. Se utiliza para obtener la posición de los escondites y de los objetos de curación.
- **Wait:** Acción simple que simula una pausa en el comportamiento del jugador. Dura el tiempo establecido en los parámetros del operador.

Para actualizar la información almacenada en el estado del mundo, utilizada por los operadores, se ha creado un controlador que gestiona las habitaciones del nivel. Cada habitación dispone de su controlador, que se encarga de guardar en el estado del mundo la información de las salidas de la habitación y las posiciones de los objetos de curación y los escondites, siempre que haya alguno. La información de una habitación solo se almacena cuando el agente entra en ella por primera vez. Esta detección se realiza mediante un *trigger* que cubre todo el espacio de la habitación y solo detecta al personaje no jugador.

Además del controlador de las habitaciones, que se utiliza como sensor para actualizar el estado del mundo, también se ha creado un controlador para definir al agente. En este componente es donde se crea el estado del mundo, añadiendo todas las variables que van a utilizarse, y donde se construye la red jerárquica de tareas que termina usando el planificador.

A continuación se mostrará la red jerárquica de tareas mediante dos diagramas. En el primero aparece de forma completa, con dos de sus nodos representados con el color rojo. Estos nodos contienen el mismo fragmento de la jerarquía, que se ha reutilizado. En el segundo se muestra en su totalidad el contenido de los nodos de color rojo, que representa la acción de ir al destino previamente seleccionado. La red jerárquica de tareas se muestra separada de esta manera para facilitar su visualización.

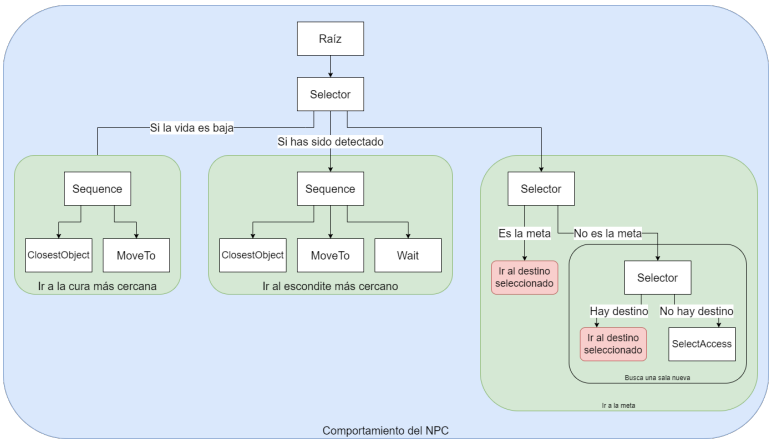

Figura 5.2: Diagrama completo de la red jerárquica de tareas.

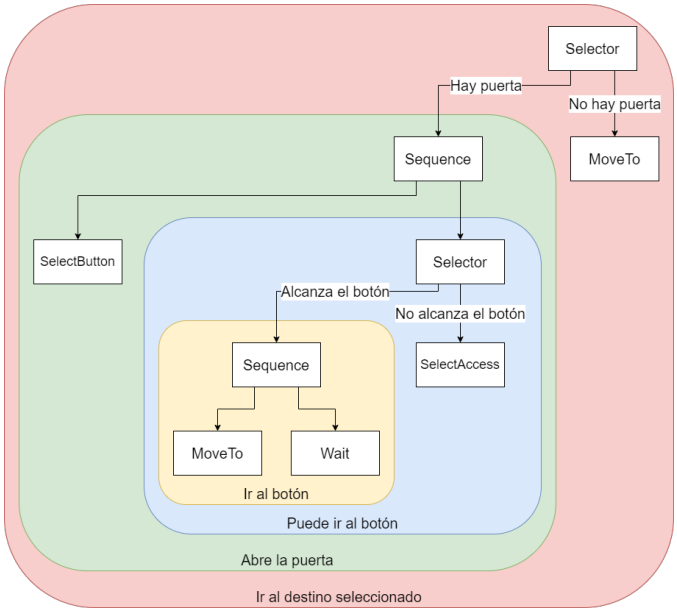

Figura 5.3: Diagrama parcial de la red jerárquica de tareas.

Capítulo 6

Resultados

En este capítulo se discutirán las pruebas realizadas sobre el trabajo para estudiar su funcionalidad y analizar los resultados obtenidos de las mismas.

6.1. Pruebas

Para probar el funcionamiento y la utilidad de la herramienta se ha decidido comparar la implementación realizada en el proyecto con la hecha por los alumnos de la asignatura de *Inteligencia artificial para videojuegos* para una de sus prácticas, ambas sobre el mismo entorno de demostración y con el mismo problema. En el caso de los alumnos se ha utilizado una herramienta distinta para resolver el problema, empleando árboles de comportamiento en vez de redes jerárquicas de tareas. Esto permitirá comparar los dos modelos y sus diferencias a la hora de utilizarse.

El grupo de alumnos que han dado acceso a su práctica para realizar las pruebas está compuesto por un total de 40 personas, con una edad comprendida entre los 20 y 25 años. Todos ellos disponían de experiencia previa con el motor de videojuegos Unity, pero ninguno había usado con anterioridad ni el entorno *Liquid Snake* ni la herramienta de árboles de comportamiento *Behavior Bricks*. Tampoco tenían experiencia usando árboles de comportamiento hasta la realización de la práctica y solo habían utilizado máquinas de estado previamente.

6.2. Resultados

El objetivo del problema a resolver es que el personaje no jugador alcance la meta mientras avanza por una serie de salas conectadas por accesos que pueden o no tener una puerta. Estas puertas están conectadas a un botón que

es necesario pulsar para abrirlas. Además, el personaje deberá mantenerse con vida mientras evita a un grupo de enemigos que patrullan el escenario y lo atacan cuando lo ven.

Tanto la implementación realizada para el proyecto como las hechas por los alumnos consiguen que el personaje no jugador cumplan con su objetivo de alcanzar la meta. Aun así, se pueden realizar múltiples observaciones al respecto.

6.2.1. Primera observación

La primera diferencia observada es la dependencia entre el problema y la solución. Las soluciones de los alumnos utilizan los árboles de comportamiento para guiar al personaje por el escenario estableciendo de manera previa a la ejecución todos los pasos que debe seguir para tener éxito. El problema de este tipo de implementación es que no funciona al aplicarse a otros escenarios con una distribución distinta de salas, puertas y botones sin cambiar los árboles, de forma que cada caso necesita una solución específica.

La implementación realizada con HTN plantea una solución más abstracta que permite al personaje adaptarse a la situación a medida que avanza. En vez de guiarlo paso por paso, se van eligiendo caminos de forma aleatoria, priorizando los que llevan a las salas no visitadas, hasta encontrar la meta. De esta manera, no existe una solución predefinida y se puede utilizar la misma implementación con problemas distintos.

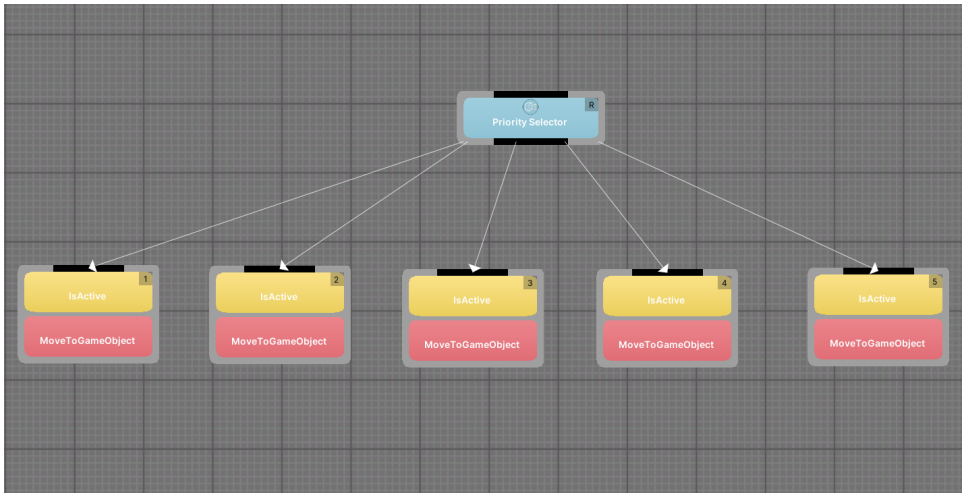

Figura 6.1: Árbol de comportamiento con el orden de los botones.

6.2.2. Segunda observación

La segunda diferencia observada es la complejidad de los árboles de comportamiento frente a las redes jerárquicas de tareas. En las soluciones realizadas por los alumnos, los árboles contienen una gran cantidad de nodos y, en ocasiones, hasta otros árboles anidados que no se están reutilizando. Esto reduce la legibilidad de la solución y hace más difícil su depuración.

La implementación realizada con HTN prioriza la reutilización de las tareas primitivas y algunos fragmentos de la jerarquía, reduciendo el tamaño de la red como consecuencia. Además de facilitar la compresión de la solución y su depuración, también se facilita la escalabilidad, abriendo la posibilidad de añadir nuevos comportamientos en un futuro.

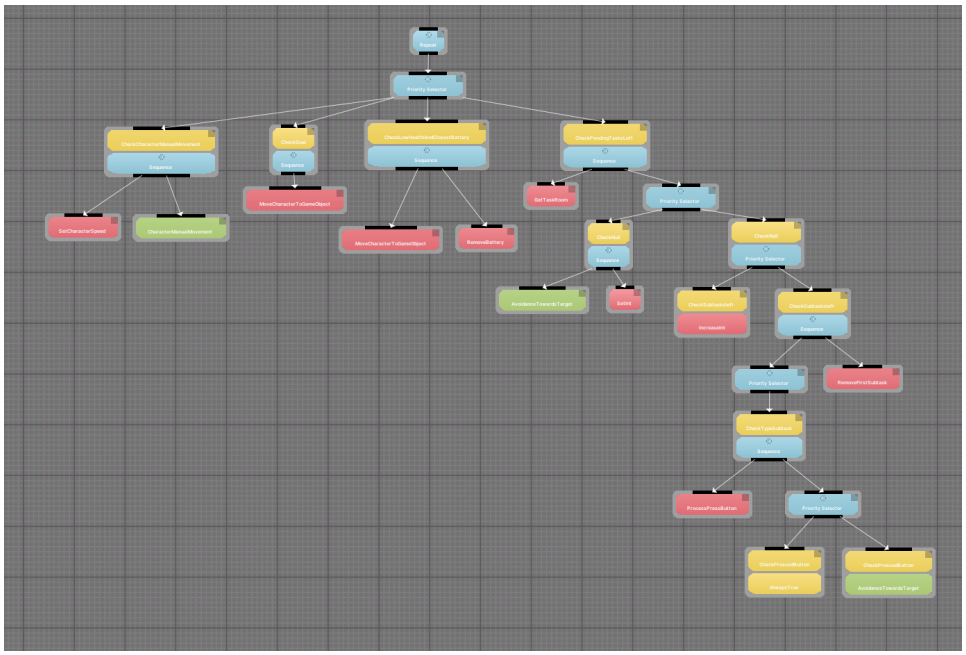

Figura 6.2: Árbol de comportamiento completo.

6.2.3. Tercera observación

La tercera diferencia observada es el tiempo que se tarda en resolver el problema. Las soluciones obtenidas mediante árboles de comportamiento son más lentas que la solución hecha mediante redes jerárquicas de tareas. El principal motivo de que esto ocurra es la propia naturaleza de los modelos.

Los árboles de comportamiento tienen una tendencia a ser reactivos haciendo que, en ocasiones, el personaje se quede en un bucle repitiendo las mismas acciones a pesar de que van a continuar fallando. Estos bucles siem-

pre terminan cuando las acciones acaban teniendo éxito pero se pierde mucho tiempo en el proceso.

La redes jerárquicas tienen una tendencia a ser proactivas, ya que se planifica con antelación una lista de tareas a realizar. De esta forma, se evita perder el tiempo repitiendo acciones que van a fallar y se maximiza el tiempo en el que se realizan las que van a tener éxito.

6.3. Discusión

Una vez realizadas todas las observaciones y analizados los resultados se puede concluir que para resolver problemas de mayor complejidad, que cuentan con una gran cantidad de toma de decisiones en tiempo real, la planificación automática ofrece varias ventajas sobre modelos como los árboles de comportamiento o las máquinas de estado.

A pesar de ser más difíciles de entender y utilizar, si se realiza el planteamiento correcto, las redes jerárquicas de tareas permiten:

- Obtener soluciones más abstractas que no se restringen empleando un único método de resolución del problema y se adaptan a varios escenarios.
- Reutilizar múltiples tareas primitivas o fragmentos de la jerarquía de forma que se recude el tamaño de las redes, aumentando su legibilidad y depuración, y permitiendo que sean más escalables.
- Reducir el tiempo en que se completa la solución del problema en ejecución gracias la planificación previa de las acciones que se van a realizar, evitando bucles de acciones que terminan fallando.

Aunque la herramienta desarrollada durante el proyecto proporciona todas las ventajas y beneficios mencionados anteriormente, si la comparamos con herramientas similares a disposición del usuario en entornos de desarrollo como Unity, podemos observar que aun queda mucho trabajo para ofrecer las mismas funciones y características.

Los puntos débiles de la herramienta recaen en la ausencia total de una interfaz de usuario que facilite su comprensión y utilización, y las limitaciones a la hora de depurar tanto fuera como dentro de la ejecución. Estos aspectos son de vital importancia a la hora de conseguir que más gente utilice la herramienta, especialmente aquellos que no tengan tantos conocimientos sobre planificación automática y, en concreto, redes jerárquicas de tareas.

Capítulo 7

Conclusiones

En este capítulo se repasarán los objetivos propuestos durante el desarrollo del proyecto, comentando si han sido alcanzados o no. Además, se añadirán algunas líneas de trabajo futuro para mejorar o ampliar la herramienta desarrollada.

Un artículo con resultados preliminares sobre este trabajo ha sido aceptado en el III Congreso Español de Videojuegos. Lleva por título *Towards a Non-Player Character Framework based on Automatic Planning and Hierarchical Task Networks* (Fernández y Peinado (2024)).

A continuación se enumeran los objetivos y se analiza si han sido cumplidos:

1. **Diseñar un algoritmo de planificación basado en el algoritmo de búsqueda en anchura**. Este objetivo se ha alcanzado, tal y como se describe en el Capítulo 5. El desarrollo del algoritmo de planificación ha sido posible usando como referencia uno de los capítulos del libro *Game AI Pro*, adaptando el ejemplo al resto de elementos de las redes jerárquicas de tareas que ya se habían desarrollado.

2. **Diseñar un conjunto de tareas primitivas y compuestas, definiendo un proceso de descomposición de tareas que use los operadores de selección y secuencia**. Este objetivo se ha alcanzado, tal y como se describe en el Capítulo 5. Inicialmente se desarrollaron las tares primitivas y, más adelante, las compuestas, ya que son una agrupación de las primitivas. Con las tareas compuestas se establecieron los dos tipos de descomposición: la selección, que elige solo la primera tarea que se valida siguiendo el orden de prioridad, y la secuencia, que elige el conjunto de tareas al completo siempre y cuando todas sean válidas.

3. **Diseñar un almacén de variables de cualquier tipo donde guar-**

dar y modificar la representación del estado del mundo, incluyendo las condiciones y los efectos de las tareas. Este objetivo se ha alcanzado, tal y como se describe en el Capítulo 5. Para lograrlo se utiliza el tipo *object* de C# que permite convertir las variables a cualquier otro tipo. De esta forma no se limita la información almacenada por el estado del mundo. En el caso de las condiciones y los efectos solo se utiliza el tipo *bool* para facilitar la validación de las tareas.

4. **Desarrollar un escenario de ejemplo en Unity que sirva como demostración del sistema creado, donde se defina el comportamiento de un personaje controlado por la máquina resolviendo un problema complejo**. Este objetivo se ha alcanzado, tal y como se describe en el Capítulo 5. En vez de crear el entorno de demostración desde cero, se optó por utilizar uno ya existente para acelerar el desarrollo del problema que el personaje no jugador debía resolver y también la implementación de la solución a dicho problema, además de ofrecer un acabado visual más atractivo.

5. **Comparar la eficacia de este sistema frente a otros paradigmas de Inteligencia Artificial para Videojuegos usando soluciones al mismo problema pero creadas con árboles de comportamiento**. Este objetivo se ha alcanzado, tal y como se describe en el Capítulo 6. Para realizar las comparaciones se han utilizado las prácticas de varios alumnos de la asignatura *Inteligencia artificial para Videojuegos* que resolvían el mismo problema que este proyecto pero usando el paradigma de los árboles de comportamiento.

7.1. Trabajo futuro

A pesar de haber cumplido con los objetivos y las expectativas del proyecto, algunos apartados de la herramienta se pueden mejorar o, incluso, expandir. A continuación se exponen una serie de líneas de trabajo futuro a priorizar.

- Creación de una guía de uso de la herramienta y tutoriales con ejemplos.
- Mejora del código, en concreto la documentación de los elementos de la herramienta HTN.
- Desarrollo de una interfaz de usuario visual que facilite y agilice el uso de la herramienta.
- Desarrollo de funciones de depuración tanto fuera, en tiempo de desarrollo, como dentro de la ejecución.

Los resultados obtenidos durante este proyecto en forma de herramienta, que recibirá el nombre de HTN NPC, ofrecen una base sólida que permite utilizar un modelo de IA para Videojuegos menos usado y conocido, como son las redes jerárquicas de tareas, con la finalidad de popularizarlo en el ámbito del desarrollo de videojuegos independientes.

Todo el desarrollo ha sido publicado en el repositorio de GitHub del siguiente enlace: `https://github.com/Narratech/TFG-Fernandez`. Se ha utilizando una licencia LGPL 2.1 para publicarlo como software de uso libre.

Appendix A

Introduction

Artificial intelligence (AI) in video games offers many possibilities for modeling the behavior and decision-making process of non-player characters (NPCs). A common goal is to develop enemies that offer a challenge to players while maintaining a balance between fun and difficulty.

Some of the paradigms that have become popular in the implementation of NPCs to the point of becoming industry standards are Finite State Machines (FSMs), which define behavior based on the internal state of the character, and Behavior Trees (BTs), which define behavior through a series of interconnected tasks in the form of a hierarchy.

This work explores the automatic planning paradigm, in particular, the Hierarchical Task Network (HTN) model.

The objective is to develop a free-to-use HTN tool for Unity and to compare its design, performance and user experience with those of other more commonly used conventional techniques.

A.1. Motivation

During the development of video games, AI is one of the aspects that requires the most time and dedication. To facilitate this process, a variety of paradigms have been used, among which those that offered greater ease of implementation and use have stood out. Even so, with the passage of time, agent behaviors have needed to acquire an increasing level of complexity and credibility.

More advanced paradigms, such as automatic planning, have been used to solve this problem. The first model to become famous was GOAP (Goal Oriented Action Planning), from which others, somewhat less well known, have been applied, such as HTN (Hierarchical Task Network). The latter is

an alternative to GOAP that offers greater control when designing the plans that the agents are going to execute, helping to better define their behavior. Despite being a model that provides very attractive features for developing video games, there are hardly any tools, both free and paid, integrated in the main development environments such as Unity or Unreal Engine.

A.1.1. Purpose

The goal of this work is to build a tool that allows defining the behavior of multiple NPCs using the HTN model. The tool will be developed as an extension of today's most popular video game development environment, Unity, and will be released as a free resource, thus offering a free alternative of a not so exploited AI model in video games.

A.1.2. Scope

The expected scope during the development of the tool is to build a solid and functional basis of the hierarchical task network model.

This means that it will be possible to correctly define the behavior of one or several NPCs, which will act simultaneously and uninterruptedly.

A demonstration environment will also be available to illustrate the implementation and operation of the tool.

The aspects not covered by this project are the availability of a system that allows to debug in depth the internal operation of the tool or the NPCs during the execution, nor a visual user interface that facilitates the use of the tool when implementing the different HTN behaviors.

A.1.3. Related subjects

The following is a list of subjects of the Video Game Development degree that are related to the work done during the project:

- **Agile production methodologies:** related to project organization and repository management.
- **Video game design:** related to the design of the demonstration environment and the problem that the non-player character must solve.
- **Game engines:** related to the use of the Unity video game engine.
- **Artificial intelligence for videogames:** directly related to the project, since it consists of the development of an AI through hierarchical task networks.

- **Project I, II and III:** related to everything necessary to carry out a large-scale project over a long period of time.

A.2. Structure of the work

The structure of this work consists of the following sections:

- **Chapter 1: Introduction.** The motivation, purpose and scope of this work and the degree subjects related to this project are presented.
- **Chapter 2: State of the art.** The state of the art in AI used within video games or as a technique to aid in their development is reviewed. In addition, the different models that exist and some works similar to the project are studied.
- **Chapter 3: Objectives.** The main objectives of the work are presented and a specification of all the requirements necessary to develop the proposed tool is made.
- **Chapter 4: Methodology.** The methodology used to develop the project and the specific tools used in the process are explained.
- **Chapter 5: Development.** A detailed explanation of the design and implementation of the project is provided.
- **Chapter 6: Results.** It begins with an explanation of the tests. Then, the results obtained from the tests with real users, carried out to validate the performance of the tool and to evaluate its usefulness for real developers, are discussed.
- **Chapter 7: Conclusions.** The conclusions generated as a result of the comparison of the final version of the tool with the objectives defined at the beginning are shown in order to validate whether these have been met.

Appendix B

Conclusions

This chapter will review the objectives proposed during the development of the project, commenting on whether they have been achieved or not. In addition, some lines of future work will be added to improve or extend the developed tool.

A paper with preliminary results on this work has been accepted in the III Spanish Congress of Video Games. It is entitled *Towards a Non-Player Character Framework based on Automatic Planning and Hierarchical Task Networks* (Fernández y Peinado (2024)).

The following is a list of the objectives and an analysis of whether they have been met:

- **Design a planning algorithm based on the breadth-first search algorithm**. This objective has been achieved, as described in chapter 5. The development of the planning algorithm has been possible using as reference one of the chapters of the book *Game AI Pro*, adapting the example to the rest of the elements of the hierarchical task network that had already been developed.
- **Design a set of primitive and compound tasks, defining a task decomposition process using the selection and sequence operators**. This objective has been achieved, as described in chapter 5. Initially, primitive tasks were developed and, later, compound tasks were developed since they are a grouping of primitive tasks. With the compound tasks, the two types of decomposition were established: the selection, which chooses only the first task to be validated following the priority order, and the sequence, which chooses the whole set of tasks as long as all of them are valid.
- **Design a variable storage of any type where to store and modify the representation of the state of the world, including the**

conditions and effects of tasks. This goal has been achieved, as described in chapter 5. To achieve this, C# *object* type is used, which allows variables to be converted to any other type. In this way, the information stored by the state of the world is not limited. In the case of conditions and effects only the type *bool* is used to facilitate the validation of the tasks.

- **Develop an example scenario in Unity that serves as a demonstration of the created system, where the behavior of a machine-controlled character solving a complex problem is defined**. This goal has been achieved, as described in chapter 5. Instead of creating the demo environment from scratch, we chose to use an existing one to speed up the development of the problem that the non-player character had to solve and also the implementation of the solution to that problem, in addition to offering a more attractive visual aspect.
- **Compare the effectiveness of this system against other paradigms of Artificial Intelligence for Videogames using solutions to the same problem but created with behavior trees**. This objective has been achieved, as described in chapter 6. To make the comparisons we have used the practices of several students of the subject *Artificial Intelligence for Videogames* that solved the same problem as this project but using the paradigm of behavior trees.

B.1. Future work

Despite having met the objectives and expectations of the project, some sections of the tool can be improved or even expanded. The following are a series of lines of future work to be prioritized.

- Creation of a guide for the use of the tool and tutorials with examples.
- Improvement of the code, in particular the documentation of the elements of the HTN tool.
- Development of a visual user interface to facilitate and speed up the use of the tool.
- Development of debugging functions both outside, at development time, as well as within the execution.

The results obtained during this project in the form of a tool, which will be called HTN NPC, offer a solid foundation that allows the use of a less used and less known AI model for Video Games, such as hierarchical task

networks, with the aim of popularizing it in the field of independent video game development.

All development has been published in the GitHub repository at the following link: `https://github.com/Narratech/TFG-Fernandez`. An LGPL 2.1 license has been used to release it as free of use software.

Bibliografía

Agis, R., Gottifredi, S. y Garcia, A. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 2020.

Champandard, A. J. y Dunstan, P. The behavior tree starter kit. *Game AI Pro 360*, 2019.

Chris Conway, P. H. y Jacopin, E. Goal-oriented action planning: Ten years old and no fear! 2015. `https://www.youtube.com/watch?v=gm7K68663rA`.

Fernández, I. y Peinado, F. Towards a non-player character framework based on automatic planning and hierarchical task networks. iii congreso español de videojuegos (cev), en el marco del vii congreso español de informática. june 17–21, 2024, a coruña, spain. *Libro CEUR Workshop Proceedings*, 2024.

Graham, D. R. A reusable, light-weight finite-state machine. *Game AI Pro 360*, 2017.

Humphreys, T. Exploring htn planners through example. *Game AI Pro 360*, 2019.

Orkin, J. Three states and a plan: The a.i. of f.e.a.r. 2006.

Riedl, M. O., Thue, D. y Bulitko, V. Game ai as storytelling. 2011.

Soemers, D. J. N. J. y Winands, M. H. M. Hierarchical task network plan reuse for video games. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, páginas 1–8, 2016.

Studios, T. S goap: Ai solution. 2020. `https://assetstore.unity.com/packages/tools/behavior-ai/s-goap-ai-solution-167167`.

Tools, B. Badger htn. 2023. `https://assetstore.unity.com/packages/tools/behavior-ai/badger-htn-244599`.