Planificación Automática para el Comportamiento de Personajes de Videojuegos como Extensión de Unreal Engine

Autores

Diego Romero-Hombrebueno Santos
Mario Sánchez Blanco
José Manuel Sierra Ramos

Director

Federico Peinado Gil

Trabajo de Fin de Grado en Ingeniería del Software
Facultad de Informática
Universidad Complutense de Madrid

Curso 2019–2020

Automatic Planning for Video Game Characters Behavior as Unreal Engine Plugin

Authors

Diego Romero-Hombrebueno Santos
Mario Sánchez Blanco
José Manuel Sierra Ramos

Director

Federico Peinado Gil

End-Of-Degree Project in Software Engineering
Faculty of Informatics
Complutense University of Madrid

Year 2019–2020

Agradecimientos

Gracias a Federico Peinado por guiarnos y ayudarnos durante todo el desarrollo del trabajo.

Gracias a Jeff Orkin por diseñar el modelo de planificación de acciones orientada a objetivos.

Gracias a Víctor Blanco por revisar todo el código y plantearnos una serie de mejoras sobre la herramienta.

Gracias a Jorge Osorio por su ayuda en el marketing relacionado con la publicación del *code plugin* en la tienda de recursos de *Unreal Engine*.

Gracias a *Epic Games* por ofrecer una plataforma en la que publicar nuestra herramienta, dándole visibilidad y difundiéndola.

Gracias a *Narratech Laboratories* por promocionar nuestra aplicación a través de medios y redes sociales.

Gracias a todas las personas que han respondido a nuestro cuestionario y han realizado el tutorial.

Y gracias a vosotros, desarrolladores, por utilizar nuestra herramienta.

Resumen

La inteligencia artificial es uno de los pilares fundamentales sobre los que se ha asentado la industria de los videojuegos. Por este motivo, gran parte de los recursos empleados en el desarrollo de un videojuego son destinados al ámbito relacionado con la simulación de inteligencia y, especialmente, a mejorar el comportamiento de los personajes no jugadores.

Sin embargo, no todos los estudios disponen de los mismos medios para hacer frente a los costes asociados al desarrollo de inteligencia artificial. Esto provoca que las pequeñas compañías o desarrolladores independientes tengan que recurrir en muchas ocasiones a obtener recursos a través de tiendas de contenidos. El problema surge con la falta de recursos de calidad relacionados con inteligencia artificial en dichas tiendas.

Para solucionar este problema, hemos creado una herramienta para *Unreal Engine* que permite llevar a cabo la planificación automática, bajo una arquitectura GOAP, del comportamiento en personajes no jugadores de una manera sencilla. Esta herramienta ha sido desarrollada como *code plugin*, lo que permite que sea incluida fácilmente en cualquier proyecto, y se ha publicado en la tienda de recursos de *Unreal Engine*, para hacerla más accesible a cualquier desarrollador.

Palabras clave

Algoritmo de Búsqueda A*, *Blueprint*, *C++*, Controlador, Estado del Mundo, Inteligencia Artificial, Personaje No Jugador, Planificación de Acciones Orientada a Objetivos.

Índice

Índice de figuras

Capítulo 1

Introducción

"Sigue hambriento, sigue alocado."
— Steve Jobs

Como estudiantes del Grado en Ingeniería de Software, siempre hemos tenido un gran interés por el ámbito de la inteligencia artificial (IA) y las posibles aplicaciones que se pueden hacer de la misma en diversos entornos como es el caso de los videojuegos.

Así pues, empezaremos realizando un repaso histórico a la industria de los videojuegos y, especialmente, nos centraremos en la evolución de la IA, con la intención de fundamentar los objetivos de nuestro trabajo de fin de grado. Además, ponemos de manifiesto la relación de este proyecto con las asignaturas cursadas y también planteamos la estructura del trabajo que vamos a seguir.

1.1. La industria de los videojuegos

Desde los comienzos de la industria de los videojuegos en la década de los 70, el negocio del desarrollo, distribución, promoción y venta de videojuegos se encuentra en continuo crecimiento. El impacto social que generó la aparición de los videojuegos marcó un cambio revolucionario en el mundo del entretenimiento. Desde sus orígenes hasta hoy en día, este fenómeno no ha dejado de expandirse y evolucionar, limitado únicamente por el progreso de la evolución tecnológica (Informa Tech Holdings, 2019).

Con el paso de los años, los videojuegos se han convertido en una de las

principales opciones de ocio para muchos consumidores. La popularidad que han llegado a alcanzar les han permitido obtener el estatus de medio artístico, como puede ser el cine o la televisión, en tan sólo unas cuantas décadas de existencia.

Este gran auge se debe sobre todo a su alta capacidad de adaptación dentro del mercado, sabiendo adecuarse a las necesidades y los gustos de muchos sectores de la sociedad. La evolución constante es la mejor forma de definir a esta industria que ha llegado a convertirse en un gran motor económico que genera miles de millones de dólares anualmente (véase Figura 1.1).

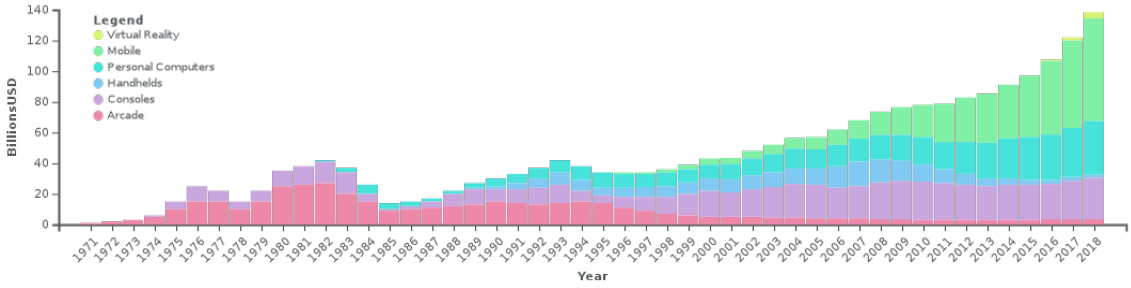

Figura 1.1: Ingresos de la industria de los videojuegos de 1971 a 2018 (Naramura, 2019).

Inicialmente, el mercado de los videojuegos se encontraba dominado por las grandes editoriales de videojuegos que ofrecen apoyo financiero a las desarrolladoras en el proceso de crear videojuegos. Al tratarse de un mercado con grandes expectativas de crecimiento, la competencia por hacerse un hueco en él siempre ha sido muy fuerte y de difícil supervivencia en un entorno en el que la capacidad económica es crucial para mantenerse estable (B., 2018).

Durante muchos años, las grandes compañías absorbían a las pequeñas empresas de desarrollo y, así, se hacían con el control del mercado. Este hecho limitaba el desarrollo de videojuegos independientes, que se veían casi obligados a ceder ante las grandes empresas o a conformarse con la distribución *shareware* o entre amigos.

Sin embargo, el panorama cambió radicalmente en la última década con la aparición de nuevos medios que han facilitado la posibilidad de entrada al mercado y mantenimiento en el mismo a todo tipo de desarrolladores. Entre estos nuevos medios cabe destacar el surgimiento de las plataformas digitales de venta como *Epic Store* (Epic Games, 2018) o *Steam* (Valve Corporation, 2003), donde ahora es posible la distribución y venta de videojuegos en un

mercado mundial con muy poca inversión por parte del desarrollador independiente (Desarrollo Español de Videojuegos, 2019). Este tipo de tiendas permiten evitar el obstáculo que se encuentran muchos estudios de desarrollo a la hora de llevar a cabo grandes campañas de marketing para dar a conocer su producto.

Además, la liberación de herramientas para la creación de videojuegos como *Unity* (Unity Technologies, 2005) o *Unreal Engine* (Epic Games, 1998) han servido para facilitar el proceso de desarrollo. Estas herramientas permiten a los desarrolladores la creación de su producto en un entorno unificado que ahorra costes de recursos y tiempo, debido a que muchas funcionalidades ya se encuentran implementadas de serie en el propio motor.

Con el uso de estos nuevos recursos, los estudios de desarrollo independientes se han abierto un hueco en el mercado y ahora sí pueden llegar a competir por los clientes junto a las grandes compañías.

1.2. Inteligencia artificial para videojuegos

Uno de los pilares del progreso de la industria de los videojuegos es la evolución que ha tenido lugar en el campo de la IA (Pascual, 2019). Este ámbito se centra en la labor de investigación y desarrollo de técnicas que permiten simular inteligencia en el juego, particularmente en el comportamiento de los personajes no jugadores (PNJ). Estos agentes son capaces de evaluar situaciones que son interpretadas como variables, y ejecutar acciones basándose en principios de optimización y coherencia para cumplir con una serie de determinados propósitos.

La gran diferencia de este tipo de IA para videojuegos frente a la tradicional es su enfoque fuertemente práctico. Esto se ve reflejado en que la IA para PNJs se centra en la consecución de comportamientos creíbles, es decir, que imite el comportamiento humano pero no necesariamente que sea infalible resolviendo problemas o tomando decisiones racionalmente perfectas.

La necesidad de este tipo concreto de IA para videojuegos es lo que ha llevado a que las empresas de desarrollo de videojuegos dediquen una cantidad considerable de sus recursos en mejorar sus herramientas para la creación de PNJs. Las grandes compañías han ido compitiendo por conseguir ser las creadoras de los personajes dotados con IA más compleja y realista, lo que ha provocado que a lo largo de la historia reciente hayan surgido diversas alternativas a la hora de gestionar la creación de IA para estos agentes.

Por otro lado, las pequeñas empresas o los desarrolladores independientes se han visto relegados a un segundo plano en lo que a IA para videojuegos se refiere. Al no poder afrontar los altos costes que conllevaría el desarrollo de técnicas de IA propias, sus proyectos se ven obligados a recurrir a fuentes externas que les faciliten herramientas y recursos apropiados. Como se trata de uno de los pilares más importantes del desarrollo de videojuegos, suelen tratarse de recursos escasos en el mercado o de calidad insuficiente en muchas ocasiones. Esto conlleva a que muchas de estas pequeñas empresas opten por simplificar al máximo el comportamiento de sus PNJs para reducir el impacto de la IA en sus videojuegos, provocando también una reducción del realismo y la credibilidad de los mismos.

1.3. Especificación de objetivos

Nuestra meta para este proyecto es construir una herramienta que ayude a definir el comportamiento de PNJs de una forma más flexible, modular y que se pueda utilizar con facilidad. Por este motivo, nos marcamos una serie de objetivos con la intención de especificar claramente el propósito del trabajo.

Nuestro primer objetivo será identificar una carencia importante hoy en día en las herramientas de IA para PNJs que tienen a disposición los desarrolladores *indie*. Para ello, nos valdremos de investigar los modelos de planificación de IA más utilizados, y de analizar la disponibilidad actual de recursos en el mercado de los videojuegos.

Después desarrollaremos una herramienta que cubra las necesidades identificadas anteriormente. Así pues, llevaremos a cabo un proceso adecuado de ingeniería que incluya especificación, diseño, implementación y pruebas de la herramienta.

Finalmente, nos marcamos como objetivo corregir dicha carencia lanzando al mercado la herramienta. Además, con la realimentación que recibamos de la comunidad, seguiremos manteniéndola en futuras iteraciones.

1.4. Asignaturas relacionadas

Dado que uno de nuestros objetivos es la implementación de una herramienta de software, haremos gran uso de los conocimientos adquiridos de

programación en asignaturas como Fundamentos de la Programación y Tecnología de la Programación.

En Fundamentos de la Programación adquirimos las bases de lógica y razonamiento a la hora de afrontar un desarrollo de cualquier tipo de software. Además, en dicha asignatura aprendimos a programar en código *C++*, que nos será extremadamente útil debido a que es el lenguaje utilizado en la programación en *Unreal Engine*.

Por parte de Tecnología de la Programación, pondremos en práctica los conocimientos adquiridos sobre la programación orientada a objetos. Este modelo de programación nos ayudará a plantear un diseño modular de los distintos componentes de nuestra herramienta utilizando técnicas de abstracción, encapsulamiento, herencia y polimorfismo.

Otras asignaturas estrechamente relacionadas con nuestro proyecto son Estructura de Datos y Algoritmos, y Técnicas Algorítmicas en Ingeniería del Software. Estas dos asignaturas nos ayudarán a ser capaces de analizar la eficiencia de nuestros algoritmos, así como a ser capaces de desarrollar soluciones óptimas a los problemas que se nos planteen.

Por otro lado, el proceso de planificación y gestión del desarrollo de la herramienta nos requerirá de la aplicación de los conocimientos adquiridos en las asignaturas de Ingeniería del Software, Modelado de Software, y Gestión de Proyectos Software y Metodologías de Desarrollo. Dichas asignaturas forman la parte fundamental de nuestro aprendizaje relacionado con el análisis y diseño del software, así como la gestión del equipo de trabajo.

En relación a la aplicación de algoritmos específicos para la creación de IA, nos valdremos de lo aprendido en Ingeniería del Conocimiento. Cabe mencionar que esta asignatura fue la que despertó realmente nuestro interés por llevar a cabo un trabajo de fin de grado relacionado con la IA.

Finalmente, sería importante hacer mención al resto de asignaturas que hemos cursado y no aplicaremos de manera explícita (como Bases de Datos, Gestión Empresarial, Matemática Discreta y Lógica, o Redes), pero que también nos han aportado competencias generales que indirectamente contribuyen en este proyecto.

1.5. Estructura del trabajo

Para llevar a cabo la consecución de los objetivos previamente descritos, hemos optado por dividir el trabajo en dos partes claramente diferenciadas.

Por un lado, realizaremos una labor de investigación para identificar los modelos de planificación que se podrían implementar y para comparar las fortalezas y debilidades de cada uno de ellos. Como parte de esta labor de investigación, también se realizará un análisis de la herramientas disponibles actualmente en el mercado.

Finalmente, en base a los resultados obtenidos en la parte de investigación, procederemos a realizar la segunda parte del trabajo que consistirá en el desarrollo a nivel práctico de la implementación de nuestra herramienta y el lanzamiento en el mercado de la misma.

Capítulo 2

Metodología y herramientas

"La mayoría del software actual es muy parecido a una pirámide egipcia, con millones de ladrillos puestos unos encima de otros sin una estructura integral, simplemente realizada a base de fuerza bruta y miles de esclavos."
— Alan Kay

En este capítulo abordaremos la metodología que hemos aplicado para la realización de este trabajo y discutiremos las distintas herramientas que hemos utilizado durante el transcurso del mismo.

2.1. Metodología

En relación a la metodología empleada, hemos tenido en consideración analizar qué modelo de proceso se adecuaba más a nuestro equipo y a la labor que teníamos que desempeñar. Para ser más específicos, podemos distinguir entre la metodología empleada durante el proceso de investigación y durante el proceso de desarrollo.

2.1.1. Proceso de investigación

Enfocándonos en la labor de investigación, hemos optado por utilizar un modelo de investigación híbrido entre la investigación histórica y el estudio de casos. Esto se debe a que nuestro análisis requiere del uso de estos dos

modelos de forma conjunta para obtener las conclusiones necesarias para afrontar la labor de desarrollo.

La primera parte del análisis constituye un estudio transversal de la IA para videojuegos. Aquí hemos realizado una revisión histórica de los avances en IA; haciendo especial hincapié en las distintas técnicas de toma de decisiones que existen.

El objetivo de esta parte de la labor de investigación es identificar qué modelos de planificadores existen, así como cuál de ellos es el que más eficazmente puede modelar el comportamiento de los PNJs.

A continuación hemos realizado la segunda parte de la investigación, que consiste en analizar qué necesidades existen en el mercado en relación a la disponibilidad de IA en las tiendas de recursos de *Unity* y *Unreal Engine*. Para ello, hemos tomado como base las conclusiones obtenidas en el análisis descrito en el párrafo anterior.

El objetivo de esta segunda parte es descubrir qué necesidades de recursos de IA hay en el mercado y cuáles de ellas aún no han sido cubiertas.

2.1.2. Proceso de desarrollo

En nuestro caso, para el proceso de desarrollo decidimos que lo más recomendable sería utilizar una metodología ágil inspirada en JIT, de la que destacamos el uso de *Kanban* (Kanban Tool, 2015).

Kanban es una palabra japonesa que significa *"tarjetas visuales"* y fue formulado por Taiichi Ohno para *Toyota Motor Corporation* en 1947. Así pues, esta metodología se basa en el uso de tarjetas (que representan tareas) puestas en un tablero o muro (que representa visualmente el estado de las tareas). Nosotros estimamos que cada tarjeta supondría 3 horas de trabajo continuo desde su inicio hasta su finalización pasando todas las pruebas.

Establecimos tres hitos a lo largo del desarrollo. El primero es crear una herramienta que realizara la planificación con arquitectura GOAP en *Unreal Engine* íntegramente en *C++*. El segundo hito es desarrollar otra implementación de GOAP que permita usarse como *code plugin* de *Unreal Engine*, permitiendo la herencia mediante *Blueprints* de ciertas funcionalidades. El tercer hito es la publicación del *code plugin* en la tienda de recursos de *Unreal Engine*.

Los distintos riesgos que contemplamos al inicio del proyecto fueron: el abandono de uno de los integrantes del equipo, y problemas con el uso de

ciertas herramientas. Para el caso de abandono, establecemos como plan de contingencia que se reduzca un hito (eliminando las tareas asociadas del tablero) por cada miembro que abandonase. Para problemas que surjan con el uso de ciertas herramientas, se trataría de buscar medios alternativos de uso o solventar las dudas a través de consulta a expertos en el tema.

Las tareas constituyen la unidad básica de elementos de trabajo, y en ellas se indica específicamente qué trabajo hay que llevar a cabo. Además, se pueden crear subtareas si fuese necesario fraccionar el trabajo a realizar para una tarea concreta. Este conjunto de tareas pueden tener cuatro estados posibles: *Pendiente*, *En desarrollo*, *En prueba* y *Terminado*.

El tablero se divide en cuatro columnas o espacios que corresponden con cada uno de los posibles estados de las tareas, mencionados en el párrafo anterior. De esta manera cada columna del tablero está representando un paso en el flujo de trabajo de las tareas.

Cuando se crea una nueva tarea se incorpora a la columna *Pendiente* del tablero, hasta que se comience a trabajar en ella y entonces se pase la misma a *En desarrollo*. Después de terminar el desarrollo, la tarea pasa a *En prueba*. Una vez la tarea ha pasado las pruebas, se mueve a la columna *Terminado*. De esta forma, se pueden realizar fácilmente un seguimiento del progreso del trabajo e identificar los cuellos de botella que puedan generarse.

Dada la importancia que reside en el tablero de tareas, es crucial que, a parte de que las tareas estén lo mejor definidas posibles, haya una comunicación fluida entre los miembros del equipo. Por este motivo, llevamos a cabo reuniones semanales, entre los tres integrantes siempre que podemos presencialmente o, si no fuese posible por cualquier causa, por videoconferencia. En dichas reuniones cada miembro comparte con los demás lo que ha hecho desde la última reunión y lo que hará a continuación. Además, también llevamos a cabo reuniones (cada dos semanas aproximadamente de media) con nuestro tutor para controlar el progreso del proyecto y resolver dudas que nos surjan.

Sin embargo, no podría considerarse que hemos seguido *Kanban* si no fuese porque hemos marcado límites de trabajo en el proceso (conocidos como límites *WIP*). Estos límites establecen el número máximo de tareas que se pueden tener en cada etapa, para asegurarnos que una tarea pasa al siguiente estado solamente cuando haya capacidad disponible en el destino.

Así pues, establecemos que cada miembro del proyecto solo pueda tener un máximo de dos tareas asignadas en la fase de desarrollo, aunque se permite tener hasta cuatro tareas en la fase de prueba. Estos límites se marcan para

evitar que se sature la carga de trabajo y detectar más rápidamente las áreas problemáticas en el flujo.

2.2. Herramientas

Para llevar a cabo el trabajo hemos necesitado utilizar una serie de herramientas de diversa índole. Con motivo de explicar de la manera más ordenada posible cuáles hemos utilizado, haremos una clasificación de las herramientas que hemos usado en base a la funcionalidad que necesitábamos de las mismas.

En base a este criterio, podemos distinguir entre herramientas enfocadas a facilitarnos la comunicación entre los miembros del grupo y el tutor; herramientas ligadas directamente con el desarrollo de la aplicación; y herramientas destinadas a la gestión de datos y documentación.

2.2.1. Comunicación

Para las herramientas destinadas a la comunicación, hemos escogido utilizar aquellas que más libertad nos daban a la hora de tener acceso desde distintos dispositivos y que nos permitiesen mantener un contacto más fluido entre los miembros y tutor del proyecto.

2.2.1.1. *Slack*

Slack (Slack Technologies, 2013) es una plataforma para la gestión de grupo de trabajo y proyectos de forma colaborativa. Se pueden crear diferentes canales para tratar diferentes temas, así como dotar de diferentes permisos a los usuarios para centrar el contenido de cada uno en los canales que le sean relevantes.

Aunque antes de iniciar el proyecto no teníamos constancia de la existencia de esta herramienta, hemos podido comprobar que nos ha sido de mucha utilidad al tratarse no solo de una forma de comunicación sino como un medio de contacto con otros miembros de la comunidad.

Esta herramienta la hemos utilizado como medio de comunicación principal con el tutor, así como para coordinar la colaboración con otros estudiantes y personal de *Narratech Laboratories* (Narratech Laboratories, 2020) para la asistencia a seminarios y la participación en reuniones de trabajo.

2.2.1.2. *Discord*

Discord (Discord Inc., 2015) es una aplicación que permite la comunicación directa entre usuarios a través de diferentes dispositivos, destacando especialmente por su funcionalidad de videollamadas entre múltiples usuarios. Además, otro aspecto muy útil de esta herramienta es la posibilidad de hacer retransmisión en directo desde el dispositivo de un miembro del equipo para permitir que los otros vean lo que está haciendo en pantalla.

Como no siempre hemos podido quedar presencialmente para llevar a cabo las reuniones entre los miembros del grupo, hemos usado esta herramienta para llevar a cabo dichas reuniones mediante videoconferencia.

Además, en ciertas ocasiones hemos necesitado que otro miembro del grupo viese directamente lo que estábamos realizando en pantalla desde nuestro ordenador de sobremesa. Para estos casos nos ha sido extremadamente útil utilizar la función de compartir pantalla y poder ver directamente lo que esa persona estaba viendo en ese momento.

2.2.1.3. *WhatsApp*

Al igual que *Discord*, *WhatsApp* (WhatsApp Inc., 2009) es otra aplicación que permite enviar y recibir mensajes instantáneos a través de un dispositivo móvil o un ordenador. También permite el intercambio de archivos de audio, imagen, texto y vídeo.

Pese a que en el aspecto funcional es muy parecido a *Discord*, hemos optado por utilizar *WhatsApp* para la comunicación más directa e informal entre miembros del grupo.

Hemos utilizado esta herramienta para mantener conversaciones que, por cuestiones de distancia, no podíamos tener cara a cara. Podría decirse que el uso de esta herramienta nos ha permitido trabajar como si realmente estuviésemos formando parte de un grupo de trabajo en una misma oficina, donde la comunicación es rápida y fluida.

2.2.1.4. Otros

A parte de las herramientas de comunicación mencionadas anteriormente, también hemos utilizado otros medios en menor medida como han sido el correo electrónico, llamadas telefónicas y la plataforma del Campus Virtual de la Universidad Complutense de Madrid (UCM).

2.2.2. Desarrollo

La elección de gran parte de las herramientas de desarrollo se ha visto directamente condicionada a la plataforma que hemos elegido para implementar nuestra aplicación.

2.2.2.1. *Lucidchart*

Lucidchart (Lucid Software Inc., 2020) es herramienta online para la creación de diagramas que permite a los usuarios colaborar y trabajar juntos en tiempo real. La utilizamos para crear todos los diagramas del trabajo y exportarlos en distintos formatos para poder utilizarlos.

2.2.2.2. *Office*

Como parte del paquete de ofimática *Office* (Microsoft, 2013), hemos utilizado principalmente *Word* y *Powerpoint*. *Word* es la herramienta que hemos utilizado para documentar todo aquella información no expuesta directamente en la memoria (borradores de actas, resúmenes de reuniones de grupo, etc.). *Powerpoint* nos ha servido para la realización de las diapositivas de presentaciones y la creación de gráficos.

2.2.2.3. *Overleaf*

Overleaf (WriteLaTeX Limited, 2012) es un programa de edición de texto con colaboración en línea que permite interpretar la sintaxis de LaTeX. Con esta aplicación hemos podido realizar y publicar la documentación de la memoria de nuestro trabajo de fin de grado.

Elegimos utilizar esta aplicación por recomendación de nuestro tutor del proyecto, dado que existían recursos que nos podrían facilitar el desarrollo de la memoria, así como la posibilidad de compilar y visualizar el resultado en tiempo real.

2.2.2.4. *Unity*

Unity es un motor multiplataforma para el diseño y desarrollo de videojuegos. Esta herramienta fue una de las primeras en ofrecerse al público

de forma libre, en un intento por democratizar el desarrollo de videojuegos (Axon, 2016).

En esta herramienta se combina la edición gráfica con la programación en código *C#*. Además, cuenta con una tienda oficial con una gran cantidad recursos a disposición de los desarrolladores.

2.2.2.5. *Unreal Engine*

Unreal Engine es uno de los entornos de desarrollo integrados (IDEs) más populares y usados por los desarrolladores del momento. No es solamente una aplicación para desarrollar videojuegos, sino que es un completo sistema que integra funcionalidades de todo tipo (incluyendo diseño, modelado, implementación, pruebas, etc.).

El desarrollo de aplicaciones en esta herramienta se basa en el uso conjunto de código *C++* y un sistema de visual de *scripts* llamando *Blueprints*. Además, *Unreal Engine* tiene a su disposición una gran cantidad de recursos gráficos que nos facilitan el desarrollo del apartado visual de nuestra aplicación.

2.2.2.6. *Visual Studio*

Como *Unreal Engine* se basa en código *C++*, hemos elegido utilizar *Visual Studio* (Microsoft, 1997) como plataforma de desarrollo, ya que lo habíamos usado anteriormente en otros proyectos y tenemos gran experiencia con él.

2.2.3. Gestión

A la hora de escoger las herramientas para la gestión de los recursos disponibles y creados, nos decidimos por aquellas herramientas con las que ya tuviésemos experiencia de otros proyectos realizados durante el grado.

2.2.3.1. *Drive*

Drive (Google, 2012) es un servicio gratuito de alojamiento de archivos administrado por *Google*. Este servicio ofrece alojamiento en la nube para poder almacenar y compartir archivos y carpetas. Lo hemos usado para compartir de manera rápida apuntes y anotaciones tomadas durante las

reuniones, así como información y otros recursos relevantes para la realización del proyecto.

2.2.3.2. *GitHub*

GitHub (Microsoft, 2008) es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones *Git*. En esta herramienta hemos alojado todo lo relacionado con el código del proyecto para llevar un control de la evolución del código.

2.2.3.3. *GitLFS*

Dado que el contenido del proyecto engloba la necesidad de alojamiento de una cantidad considerable de recursos gráficos, hemos utilizado *Git Large File Storage* (abreviado *GitLFS*) para gestionar archivos de gran tamaño. .

Esto ha sido necesario porque *GitHub* limita el almacenamiento de archivos que ocupen más de 100 MB. Para solucionarlo, *GitLFS* se encarga de alojar los archivos más grandes en sus servidores y enlazar mediante punteros el contenido de los mismos en el proyecto de *GitHub*.

En un principio se pensó en usar *Perforce* no obstante se descartó ya que no teníamos experiencia utilizando esta herramienta además se necesitaba un ordenador que actuara como servidor condicionando a que si se quería modificar el código este tenia que estar encendido

2.2.3.4. *Jira*

Para la administración de las tareas del proyecto, el seguimiento de errores e incidencias, y la gestión del equipo hemos utilizado *Jira* (Atlassian, 2002). El uso de esta herramienta nos ha permitido llevar a cabo la gestión de las tareas y el tablero para poder seguir la metodología *Kanban*.

Capítulo 3

Estado de la cuestión

"Preguntarse cuándo los ordenadores podrán pensar es como preguntarse cuándo los submarinos podrán nadar."
— Edsger W. Dijkstra

La IA ha sido uno de los temas más relevantes de estudio en el ámbito de los videojuegos en los últimos tiempos. Otros trabajos en los que participó nuestro tutor, que profundizan sobre la capacidad de la IA para emular el comportamiento humano en relación a su temperamento (Héctor Gómez-Gauchía, 2006) o bajo un modelo *Belief-Desire-Intention* (Federico Peinado, 2008), son un claro ejemplo de ello.

Siguiendo la línea de dichos trabajos, en nuestro proyecto nos centraremos en el estudio del realismo que se puede llegar a obtener con el uso de diferentes modelos de planificación de IA.

Así pues, trataremos de obtener conclusiones que nos permitan identificar qué problemas existen actualmente relacionados con dichos modelos, y poder establecer una especificación de objetivos a cumplir.

3.1. Inteligencia artificial para videojuegos

La aplicación principal de la IA para videojuegos es dotar de capacidad de decisión creíble a los PNJs. Para ello, se asocia un controlador a cada elemento que queramos dotar de vida en el juego. Este controlador se encarga de llevar a cabo la gestión de las capacidades del personaje, dada la situación actual del entorno donde se encuentra dicho individuo. Para realizar este

proceso de gestión de capacidades, con el objetivo de que el personaje cumpla uno o varios propósitos, se aplica el uso de IA.

Este uso de la IA para videojuegos tiene como objetivo conseguir que el comportamiento del PNJ sea plausible y expresivo para lo que se busca en el juego. El hecho de conseguir que los PNJs se comporten como si fuesen personajes controlados por jugadores humanos permite dotar de mayor inmersión al usuario que se encuentra jugando al videojuego (Statt, 2019). Por este motivo, hoy en día es muy importante para las empresas desarrolladoras asegurarse de que la IA que aplican a los PNJs se acerque a ese ideal.

Los planificadores de IA se presentan como un método eficaz para realizar este proceso de gestión de capacidades. Así pues, el controlador se vale del uso de planificadores como mecanismo para solucionar los problemas que se le plantean al PNJ en un determinado entorno (y Heliodoro Tejedor Navarro, 2012). Estos planificadores se encargan de evaluar el entorno y las capacidades del personaje, con el objetivo de determinar cómo debe comportarse el PNJ en cada situación concreta.

3.2. Toma de decisiones

A lo largo de la historia de los videojuegos han ido surgiendo diversas modelos de planificación con grandes diferencias entre sí. El hecho de que existan diferentes alternativas se debe a que aplican diferentes técnicas de toma de decisiones, con sus respectivas debilidades y fortalezas. Por este motivo, dependiendo de las necesidades de cada videojuego concreto, puede ser más recomendable utilizar un modelo de planificación u otro.

3.2.1. Planificación reactiva

La planificación reactiva es ampliamente conocida en la industria y se centra en *"cómo hacer"* algo (Brom, 2005). Este tipo de planificación se lleva utilizando desde los orígenes del uso de IA para videojuegos. Esto ha llevado a que sea un modelo de planificación muy asentado en la industria, lo que ha provocado que sea comúnmente utilizado en la mayoría de los videojuegos.

Este tipo de planificación se basa en el principio de acción-reacción. Este modelo se centra en la implementación de un sistema de decisión basado en explicar el comportamiento que debe realizar el PNJ en función del estímulo que reciba del entorno, es decir, la reacción que debe tener el individuo frente

a un evento del mundo (Vassos, 2012). Esto implica que el desarrollador se encarga tanto de implementar los comportamientos como la transición entre ellos.

Del conjunto de las técnicas de toma de decisiones empleadas en la planificación reactiva, podemos destacar dos tipos concretos debido a su repercusión e importancia a lo largo de la historia de los videojuegos: las máquinas de estados finitas y los árboles de comportamiento.

3.2.1.1. Máquina de estados finita

La máquina de estados finita es una de las técnicas de toma de decisiones más antiguas (Buttice, 2019). Videojuegos tan famosos como *Pac-Man* (Iwatani, 1980) o *Half Life* (Valve Corporation, 2009) utilizan este tipo de planificación para regular el comportamiento de sus PNJs.

Su desarrollo es muy sencillo e intuitivo, dado que se puede modelar fácilmente mediante diagramas de estados (véase Figura 3.1). Cada nodo del diagrama de estados es una acción que puede realizar el individuo, y cada flecha indica la condición que se debe cumplir para pasar de una acción a otra.

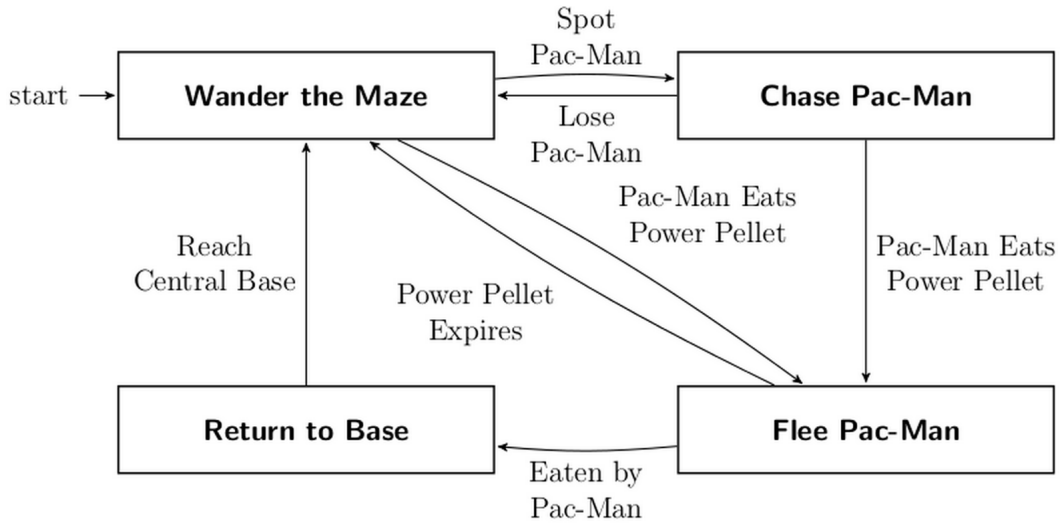

Figura 3.1: Máquina de estados finita de un PNJ en *Pac-Man*.

El problema de este modelo es que el comportamiento que se puede obtener está muy limitado. Desarrollar un diagrama de estados pequeño es muy fácil y rápido, pero la complejidad aumenta muchísimo al incrementar el nú-

mero de nodos (acciones) y flechas (transiciones) que se quieren incluir en el comportamiento del PNJ.

3.2.1.2. Árboles de comportamiento

Los árboles de comportamiento surgieron como un avance evolutivo para paliar las debilidades de las máquinas de estado finitas. Uno de los primeros videojuegos en utilizar este tipo de planificación para los PNJs enemigos fue *Halo* (2001). Este modelo se basa en la implementación de una jerarquía de nodos que parte de un nodo raíz y se ramifica hasta llegar a los nodos hoja (véase Figura 3.2).

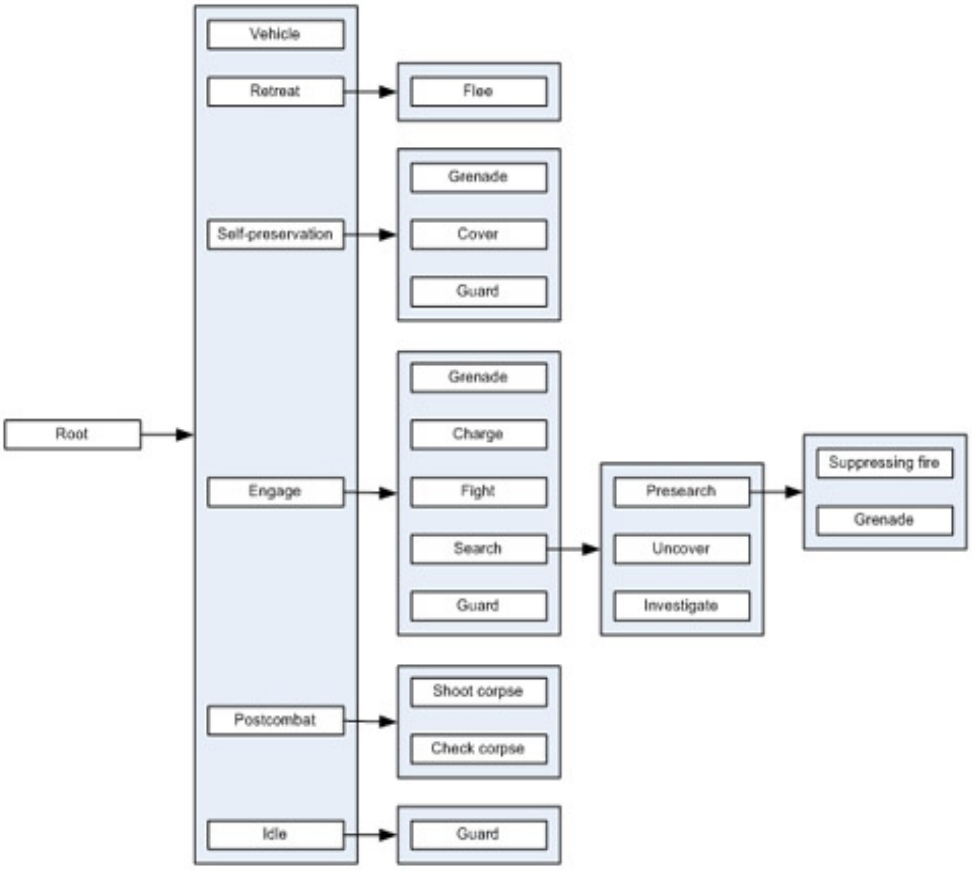

Figura 3.2: Árbol de comportamiento de un PNJ en *Halo*.

El comportamiento del individuo se establece mediante la ejecución secuencial de las tareas incluidas en los árboles. Dicha ejecución se inicia en la raíz del árbol y se va navegando por las distintas ramas hasta finalizar

la ejecución. Cada hoja del árbol es una tarea que puede realizar el PNJ, mientras que los nodos intermedios establecen las condiciones de ejecución de sus hijos.

Los hijos de un nodo *Secuencia* se ejecutan uno a uno hasta completar la lista de hijos, independientemente si se han realizado con éxito o no. Los hijos de un nodo *Selector* se ejecutan uno a uno hasta que uno de ellos se realice con éxito, lo que termina la ejecución del *Selector*.

Hoy en día es la técnica de toma de decisiones más utilizada, ya que es muy sencilla como la máquina de estados finita, pero más potente al permitir realizar comportamientos más complejos (Rasmussen, 2016). Sin embargo, sigue teniendo el mismo problema de falta de realismo al seguir utilizando una estrategia de comportamiento predefinida por el desarrollador (Bogost, 2017). Además a la hora de añadir nuevas tareas al personaje, si estas tienen dependencias con algunas de las ya existentes, es necesario replantear la estructura del árbol.

3.2.2. Planificación automática

La planificación automática es un tipo de planificación más moderno y se centra en el *"qué puede hacer"* el individuo (Malik Ghallab, 2016). Al ser más reciente, este tipo de planificación ha sido menos desarrollada y escasos videojuegos del mercado la utilizan. Esto ha provocado que los videojuegos más reconocibles de utilizar este tipo de planificación sean aquellos de alto presupuesto (triple A) que han sido desarrollados por grandes empresas.

En este modelo, el planificador recibe los posibles comportamientos que pueda llevar a cabo un determinado PNJ y se encarga de generar dinámicamente las transiciones entre ellos. La principal diferencia con la planificación reactiva es que, en el caso de la automática, la solución de un problema no se conoce hasta que se ejecute el planificador, es decir, hasta que se establezcan en vivo las transiciones entre acciones.

La planificación de acciones orientada a objetivos (GOAP, por sus siglas en inglés) es el modelo más representativo de planificación automática del comportamiento de PNJs. Esta técnica de planificación surgió como una evolución de *Stanford Research Institute Problem Solver* (STRIPS) (y Nils John Nilsson, 1971).

3.3. Planificación de acciones orientada a objetivos

GOAP (*Goal-Oriented Action Planning* en inglés) es una arquitectura de planificación especialmente diseñada para la gestión en tiempo real del comportamiento de PNJs autónomos en videojuegos. Este modelo de planificación automática fue ideado por Jeff Orkin, dándose a conocer inicialmente por sus trabajos en IA para videojuegos en *Monolith Productions* (Orkin, 2006).

El primer videojuego donde Jeff Orkin realizó una aplicación de la planificación GOAP fue *F.E.A.R.* (Monolith Productions, 2005). Desde entonces han ido surgiendo múltiples videojuegos de diferentes compañías que han utilizado arquitecturas GOAP para la gestión de la IA de los PNJs. Como ejemplos destacados de haber aplicado GOAP, cabe señalar a los videojuegos *Fallout 3* (Bethesda Game Studios, 2008), *Deux Ex: Human Revolution* (Eidos Montréal, 2011), *Tomb Raider* (Crystal Dynamics, 2013) o *La Tierra Media: Sombras de Mordor* (Monolith Productions, 2014).

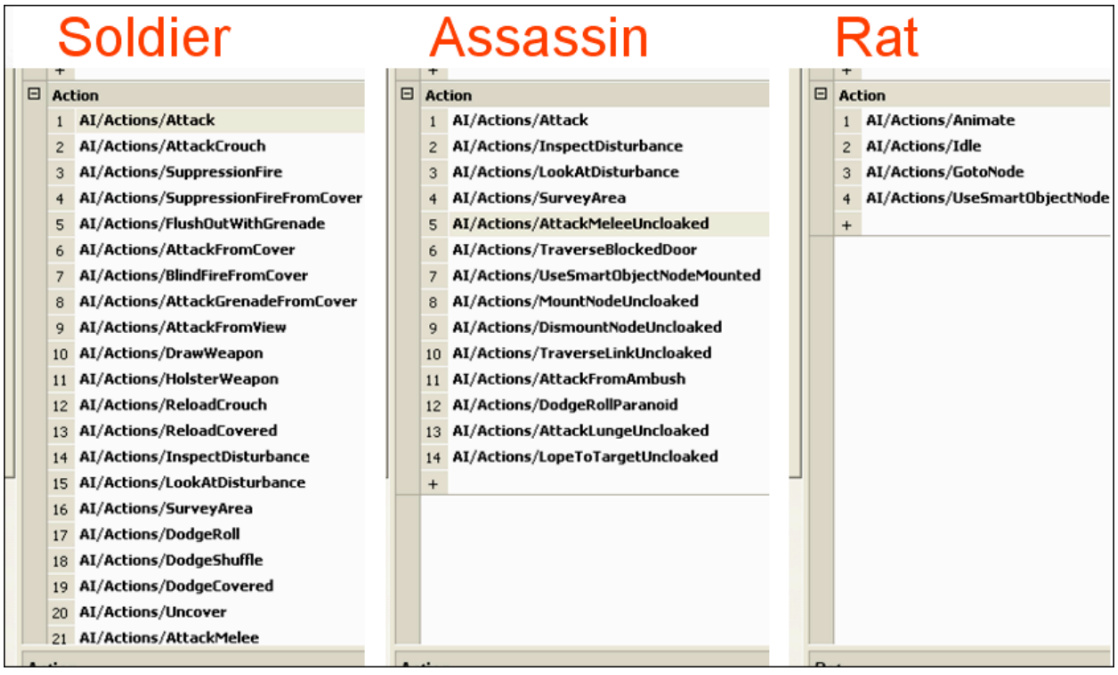

Figura 3.3: Lista de acciones GOAP de varios PNJs en *F.E.A.R.*.

El modelo GOAP se basa en la gestión dinámica del conjunto de acciones disponibles de un agente con el fin de cumplir los objetivos que hayan sido

establecidos (véase Figura 3.3). La secuencia particular de acciones a realizar depende no solo del objetivo, sino también del estado actual del mundo y del agente. Esto significa que si se proporciona el mismo objetivo para diferentes agentes o estados del mundo, se pueden obtener unas secuencias de acciones completamente diferentes, lo que hace que la IA sea más dinámica y realista (Horti, 2017).

Además, GOAP facilita muchísimo los cambios que puedan hacerse al comportamiento de los agentes, ya que no existe la necesidad de implementar las transiciones entre acciones como sucede en las planificaciones reactivas. Esto implica que con el mero hecho de agregar o eliminar acciones de un agente, se pueden producir diferentes resultados en el comportamiento del mismo. Este tipo de diseño modular de las acciones nos permite conseguir una IA altamente flexible y muy dinámica.

3.3.1. Componentes

GOAP estructura su planificación en base a dos elementos claramente diferenciados: estados y acciones. Cada agente dispone de una lista de acciones que puede realizar si se cumplen ciertos estados. La realización de una acción implica un cambio de estados, es decir, las acciones representan las transiciones entre estados. Este cambio de paradigma es otra de las grandes características de este modelo de planificación táctica, ya que los nodos representan estados y las ramas corresponden a las acciones.

3.3.1.1. Estados

Un estado representa un instante del mundo expresado en una lista de pares clave-valor formados por una referencia y un parámetro binario. Cada par en si mismo representa una condición atómica y única a cada agente. Dicho de otra manera, los estados simbolizan la evolución del conocimiento del agente sobre la situación del mundo en función de las acciones que se van realizando.

El objetivo del agente es conseguir que el mundo inicial del que parte incluya el mundo meta que tiene asignado, es decir, que la lista de pares del mundo inicial contenga la lista de pares del mundo meta. Para ello, el agente dispone de una serie de acciones que le permitirán ir añadiendo o modificando pares en la lista del mundo inicial.

3.3.1.2. Acciones

Las acciones son el conjunto de comportamientos que puede realizar el agente. Sin embargo, no todas las acciones de un agente pueden realizarse en cualquier momento, sino que dependerá si se cumplen las precondiciones de cada acción específica. Esto quiere decir que la acción solamente estará disponible o no en función del estado actual del mundo.

Las precondiciones de una acción son un estado del mundo de por si, es decir, son un conjunto de pares clave-valor formados por una referencia y un parámetro binario. Para que la acción esté disponible se requiere que el estado del mundo actual incluya el estado de precondiciones de la propia acción.

Además, una vez ejecutada la acción, se aplicarán los efectos que lleve asociados. Dichos efectos, al igual que las precondiciones, también son un estado del mundo. Esto implica que al completarse la acción, se modificará el estado actual del mundo en función de los efectos que lleve asociados. De esta manera, las acciones pueden modificar un estado del mundo para pasar a otro distinto.

3.3.2. Heurística

GOAP utiliza el algoritmo de búsqueda A* para realizar la planificación de las acciones. De esta manera, el planificador se encarga de ir recreando la estructura de nodos (estados de mundo) unidos por ramas (acciones) que permiten llegar del mundo inicial al mundo meta.

El algoritmo A* se basa en aplicar una función de evaluación de costes para poder recrear el camino más óptimo que permita alcanzar el mundo meta (Millington, 2009). Por ello, cada acción lleva asignado un peso que se corrige dinámicamente en tiempo de ejecución, ya que el coste de cada acción varía según el momento en el que se vaya a realizar.

En el caso de *GOAP* la distancia heurística es equivalente a la suma de los predicados sin satisfacer del estado del mundo del nodo final (Orkin, 2003).

3.3.3. Ejemplo de uso

Para comprender mejor cómo funciona GOAP, vamos a describir su funcionamiento mediante un sencillo ejemplo. Se trata de un escenario en el que

el objetivo del agente será conseguir abrir una puerta. Para ello, dispondrá de tres acciones: *Abrir Puerta*, *Romper Puerta* y *Coger Llave*. Utilizando dichas acciones tendrá que conseguir alcanzar el mundo meta descrito, teniendo en cuenta que partirá de un mundo inicial en el que la puerta esta cerrada y no tiene la llave.

En los siguientes gráficos se puede apreciar una descripción visual del proceso que se realiza a la hora de generar el camino óptimo de forma dinámica por GOAP. Los bloques azules representan los estados inicial y meta, mientras que los bloques morados y grises representan las acciones disponibles y no disponibles, respectivamente.

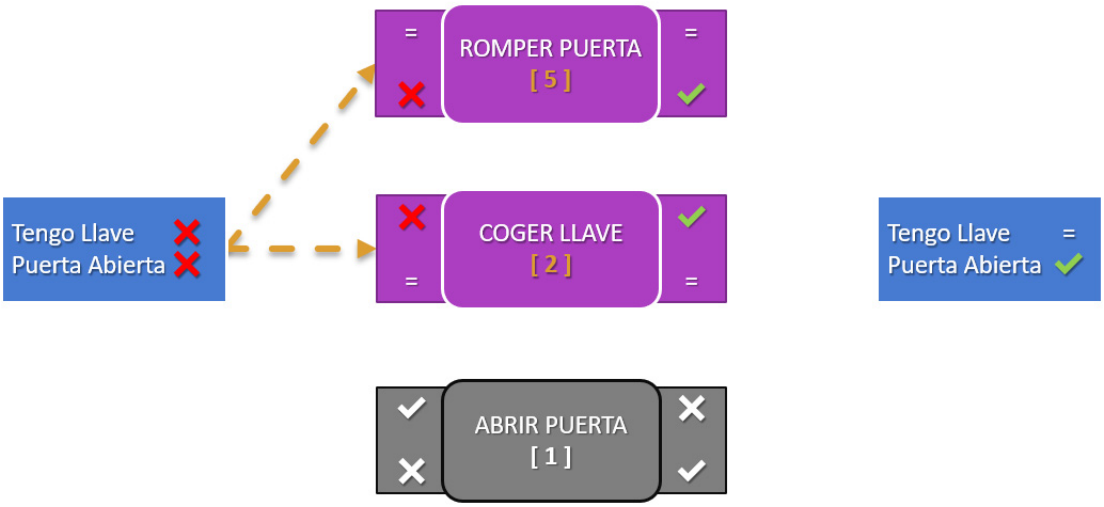

Figura 3.4: Resolución de problema de IA usando GOAP (1 de 3).

Inicialmente, el agente solo podrá realizar las acciones *Romper Puerta* y *Coger Llave*, ya que la acción *Abrir Puerta* requiere como precondición que se tenga la llave (véase Figura 3.4).

Si tomase el camino de realizar la acción *Romper Puerta*, se generaría el efecto de que la puerta se abriese y habría cumplido con el objetivo (véase Figura 3.5). Sin embargo, este camino no seria el que habría elegido el algoritmo A*, ya que el coste no es el más óptimo como veremos a continuación.

Por otro lado, si el agente toma la opción de realizar la acción *Coger Llave*, se generaría el efecto de que tendría la llave. Como consecuencia, este cambio del mundo actual implica que se habilitaría el uso de la acción *Abrir Puerta*.

Finalmente podemos apreciar que al ejecutar la acción *Coger Llave* y, después, *Abrir Puerta*; obtenemos el camino más óptimo para cumplir con

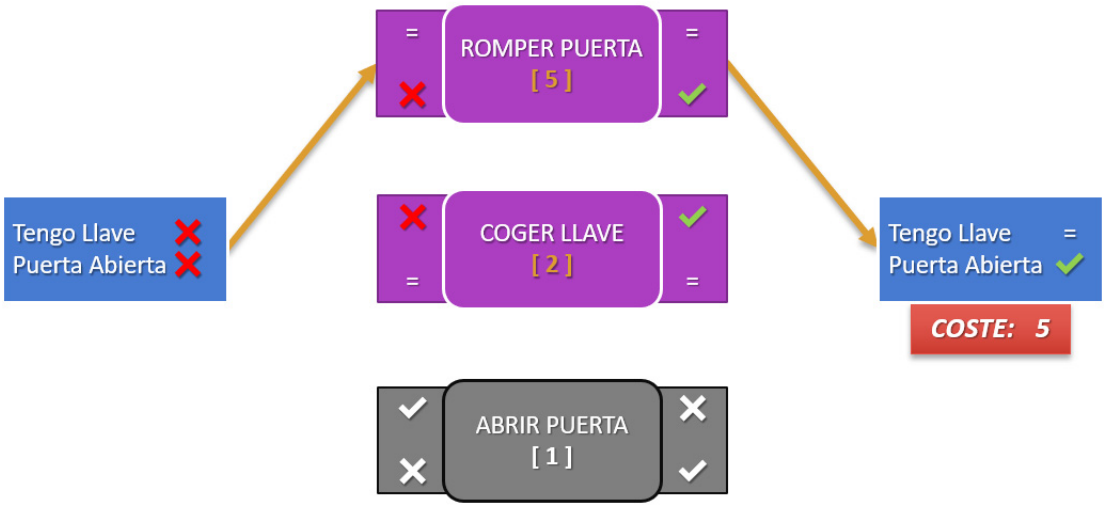

Figura 3.5: Resolución de problema de IA usando GOAP (2 de 3).

el objetivo de abrir la puerta (véase Figura 3.6).

3.4. Identificación del problema

El desarrollo de IA para PNJs es uno de los pilares más importantes para conseguir un realismo adecuado que permita al usuario final sumergirse en la experiencia que se le ofrece. Cuanto mejor desarrollada esté la IA del videojuego, mayor será la inmersión del usuario.

El problema se encuentra a la hora de decidir qué tipo de modelo seguir en la planificación de IA. Esto se debe a que dicho desarrollo supone un gran coste e inversión de recursos por parte de los desarrolladores de videojuegos. Por este motivo, la flexibilidad de elegir un sistema u otro se ve limitada por la capacidad del estudio de desarrollo para afrontar mayor o menor coste en este ámbito.

El panorama se agrava especialmente en el caso de las compañías independientes de menor tamaño, ya que la disponibilidad de recursos de este tipo de desarrolladoras es muy limitada. En este caso, muchas veces recurren a utilizar recursos de software libre que les permitan abaratar costes.

En el caso de los entornos de desarrollo de videojuegos, las compañías *indie* suelen utilizar *Unity* y *Unreal Engine*, debido a que se tratan de software de código abierto con tienda de contenidos accesible por cualquier usuario, Esto les permite reducir costes, dado que se nutren en gran medida del con-

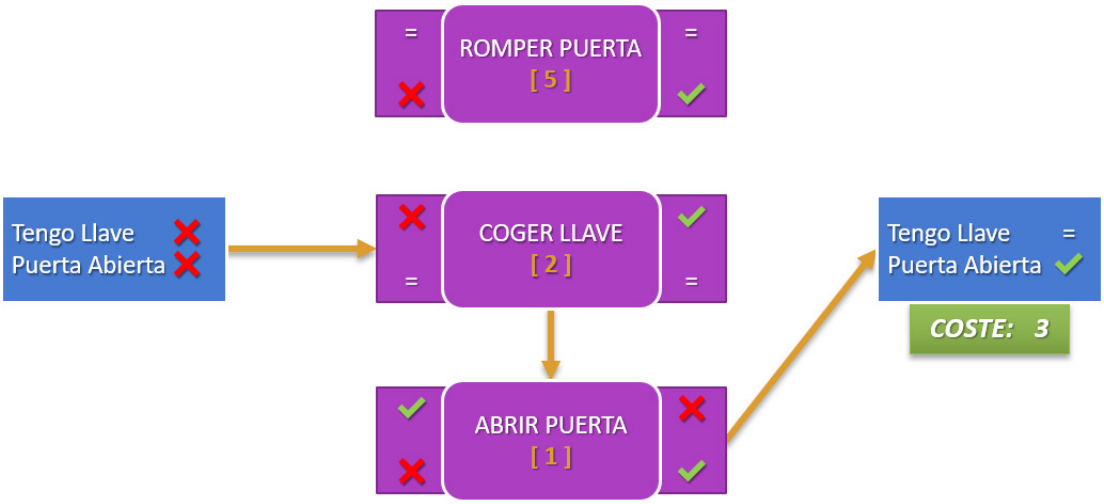

Figura 3.6: Resolución de problema de IA usando GOAP (3 de 3).

tenido creado por terceros en dichas tiendas.

Sin embargo, existe una gran limitación de recursos relacionados con el ámbito de la planificación automática de IA. Esto ha supuesto que los desarrollos de videojuegos *indie* se vean obligados a utilizar planificadores reactivos, mientras que las grandes compañías sí se pueden permitir el desarrollo de modelos de planificación mas avanzados.

Actualmente en la tienda de contenidos de *Unity* se encuentran disponibles tan solo 2 herramientas de planificación automática con arquitectura GOAP: *SGOAP* (Tinny Studios, 2020) y *ReGOAP* (Ferraro, 2018). Y aún más grave es el caso de la tienda de *Unreal Engine*, donde solo encontramos la herramienta VisAI (Visualistic Studios, 2019) que menciona GOAP en su descripción, pero ni siquiera la utiliza.

En conclusión, vemos que existe una clara necesidad en los desarrolladores que utilizan *Unreal Engine* de una herramienta que les facilite la generación de IA en base a un modelo GOAP. La disponibilidad de dicha herramienta ayudaría a reducir la brecha que existe entre las grandes compañías y las desarrolladoras *indie* en el ámbito de la IA.

Capítulo 4

Investigación y especificación de requisitos

"Primero resuelve el problema. Entonces, escribe el código."
— John Johnson

Habiendo visto en detalle el estado actual de la IA para videojuegos, así como la disponibilidad de recursos por parte de los desarrolladores para crear personajes, llegamos a la conclusión de que sería realmente muy útil para la industria el desarrollo de una herramienta que facilite resolver este problema. En concreto, el desarrollo de una herramienta para *Unreal Engine*, ya que según lo que hemos visto es el entorno de desarrollo con mayor necesidad de recursos de planificación automática.

En relación al modelo de planificación a implementar, realizaremos una comparativa de los modelos expuestos en el capítulo anterior con el objetivo de desarrollar aquel que se adecúe mejor a las necesidades del mercado y ofrezca un resultado mejor en aspectos relevantes como complejidad, escalabilidad y realismo.

Dicha comparativa, junto al análisis del mercado actual de la industria de los videojuegos, nos permitirá especificar más concretamente los requisitos que deberá cumplir la herramienta que vamos a desarrollar.

4.1. Comparativa de planificadores

En primer lugar, cabe reseñar que no existe un modelo de planificación perfecto. Esto se debe principalmente a que no sólo es importante analizar el coste de planificar de un modelo u otro en relación a tiempo o recursos. También influye enormemente el resultado que se quiera alcanzar en el desarrollo de cada IA específica. Esto significa que la elección de un planificador u otro dependerá en gran medida de las necesidades propias de cada videojuego en cuanto al resultado esperado en el comportamiento de cada individuo.

Sin embargo, sí existen ciertos factores que claramente nos permiten diferenciar una forma de planificar de otra. Estos factores nos permiten identificar debilidades y fortalezas de cada modelo de planificación. Con estas diferencias entre unos planificadores y otros, podemos establecer una comparación que nos ayude a discernir que modelo de planificación se adecúa más a las necesidades de la industria del videojuego.

4.1.1. Complejidad

Uno de los aspectos más fácilmente identificable es la complejidad de un modelo u otro. La complejidad hace referencia a la dificultad por parte del desarrollador de entender, utilizar e implementar un sistema de planificación específico.

En relación a este factor, la planificación reactiva se presenta como la opción con menor complejidad en comparación con la planificación automática que es bastante más sofisticada (Carryer, 2019). Este hecho se debe principalmente a que los modelos de planificación reactiva son aquellos que llevan utilizándose más tiempo en el desarrollo de IA para videojuegos. Al tratarse de modelos con más años de existencia, son aquellos que más tiempo llevan probándose y estudiándose en la industria. La mayoría de desarrolladores ya conocen y han trabajado con este tipo de planificadores, mientras que planificadores como *GOAP* no son tan conocidos.

Además, como en los planificadores reactivos se preestablecen las transiciones entre acciones, es más fácil representarlos gráficamente. Esto conlleva que el ser humano sea capaz de comprender mejor el comportamiento esperado de un PNJ concreto en base a dichos gráficos. En el caso de la planificación automática, al no haber transiciones predefinidas entre acciones, es más complicado entender el comportamiento de un PNJ sin llegar a ver la ejecución dinámica del mismo (Hayes, 2019).

4.1.2. Escalabilidad

Otro de los factores a tener en cuenta a la hora de elegir un modelo de planificación es la capacidad de adaptarse a cambios. Este aspecto es muy importante ya que el desarrollo de cualquier videojuego implica que se pueda dar la necesidad de tener que modificar el comportamiento de la IA de los PNJs para adaptarlos a nuevas situaciones.

Uno de los problemas de las planificaciones reactivas es la dificultad que conlleva tener que realizar modificaciones sobre planes ya creados. Este problema surge por el mero hecho de que las acciones de estos modelos de planificación están enlazadas mediante transiciones, lo que implica que añadir o quitar una acción requiera rehacer prácticamente todo el plan (Chotrani, 2018).

Sin embargo, este problema desaparece casi por completo en GOAP debido a que no existen transiciones predefinidas entre acciones. Esto implica que añadir o quitar una acción no afecta al planificador, ya que este se encarga de crear el plan de manera dinámica en tiempo de ejecución con las acciones que disponga en ese momento. Esta libertad que nos ofrece GOAP facilita en gran medida la posibilidad de ampliar o modificar el comportamiento de los PNJs.

4.1.3. Realismo

En el desarrollo de cualquier videojuego se debe tener muy en cuenta que el usuario final sea capaz de sentirse inmerso en la experiencia que se le ofrece. Para conseguir una mejor experiencia de usuario se suele tener mucho cuidado en aspectos como la narrativa, el apartado gráfico y la IA.

De los aspectos que ayudan a generar mayor realismo en los videojuegos, aquel que afecta directamente a los modelos de planificación es el comportamiento de los PNJs generado mediante técnicas de IA. El usuario espera que los agentes que interactúan en el videojuego se comporten de una manera creíble.

En este factor reside la principal diferencia entre los modelos de planificación reactiva y automática. Esto se debe a que las técnicas de toma de decisiones como los árboles de comportamiento o las máquinas de estados finitas predefinen el comportamiento de los PNJs, mientras que GOAP genera dinámicamente en tiempo real el comportamiento de los agentes, en base a sus objetivos y las acciones disponibles (Long, 2007).

En la planificación reactiva, el desarrollador se encarga de definir las secuencias de acciones (mediante transiciones) que guiarán el comportamiento del individuo. Esto implica que la IA no es realmente *"inteligente"* como para establecer dicha secuencia, ya que solo se limita a seguir un guión establecido por el diseñador en forma de árbol de comportamiento o de máquina de estados finita.

Por otro lado, la planificación automática permite obtener mayor realismo en el comportamiento de los agentes al no tener que predefinir las transiciones entre acciones, y dejar libertad al planificador de definir la secuencia de acciones en tiempo real. Esto consigue que la IA sea más independiente en la toma de decisiones y enriquezca la experiencia del usuario al adaptar su comportamiento de manera distinta en cada situación.

4.2. Análisis de mercado

A la hora de desarrollar un videojuego, hay que tener en cuenta que existe una doble limitación de recursos que influyen directa o indirectamente en el proceso de desarrollo.

Por un lado, nos encontramos con una limitación directa en el propio entorno de trabajo, debido a que la industria de los videojuegos requiere cada vez optimizar más el tiempo de desarrollo para ser capaces de obtener mejores resultados en el menor tiempo posible. Esto implica que cada vez esta siendo más importante ser capaces de reutilizar contenido y recursos, de tal manera que se agilicen aquellos procedimientos que en menor o mayor medida se repiten una y otra vez en distintos proyectos (y Marcus Toftedahl, 2019).

Esta necesidad de mercado es especialmente notoria en aquellos equipos de desarrollo en los que no existe gran margen de maniobra, como son las desarrolladoras *indie*. Este tipo de compañías deben ser capaces de optimizar muchísimo sus recursos para mantener un margen de beneficio que sea rentable, teniendo en cuenta que compiten en un mercado con grandes empresas de desarrollo que tienen altos presupuestos.

Por este motivo, tiendas de recursos como las de *Unity* y *Unreal Engine* son la principal fuente de herramientas que les permiten abaratar costes y agilizar procesos de desarrollo. Este hecho esta favoreciendo que cada vez más desarrolladoras utilicen dichos entornos de desarrollo, en vez de crear uno propio como hacen las grandes empresas (véase Figura 4.1). En este

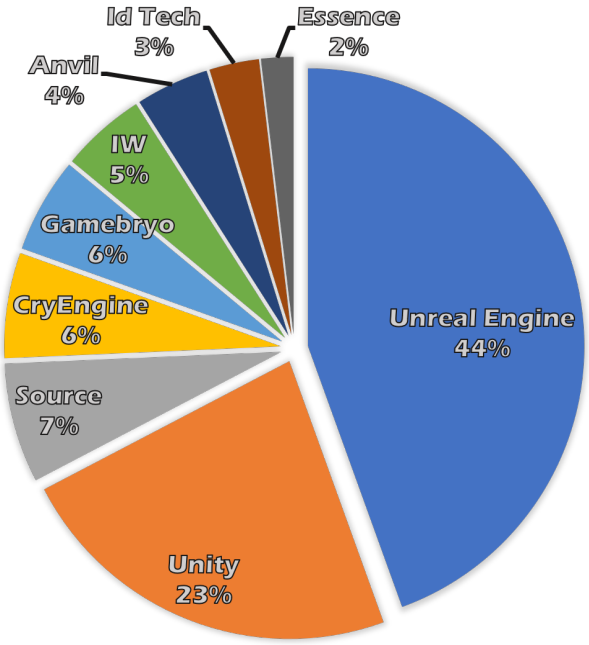

Figura 4.1: Cuota de mercado de los 10 entornos de desarrollo más utilizados en *Steam* (Toftedahl, 2019).

aspecto, la disponibilidad de herramientas en las tiendas de recursos limita de forma indirecta al proceso de desarrollo, especialmente, de las compañías más pequeñas.

En relación a herramientas relacionadas con IA, nos encontramos con una cantidad aceptable de material en ambas tiendas de recursos mencionadas anteriormente. Sin embargo, si nos centramos específicamente en herramientas de planificación automática (concretamente GOAP), podemos detectar una carencia clarísima de las mismas.

4.2.1. *Unity Asset Store*

En la tienda de recursos de *Unity* nos encontramos con tan solo 2 herramientas de planificación con arquitectura GOAP (*SGOAP* y *ReGOAP*). De dichas herramientas, hemos podido probar en profundidad la herramienta *ReGOAP* debido a que es de código abierto.

Esta investigación sobre *ReGOAP* nos ha permitido conocer mejor qué necesidades podría tener un desarrollador a la hora de utilizar un planificador de este tipo. Además, nos ha servido para mejorar nuestros conocimientos

sobre el modelo GOAP y adquirir ideas sobre cómo afrontar el diseño de nuestra herramienta.

4.2.2. *Unreal Marketplace*

En un principio, en nuestra búsqueda de recursos sobre GOAP en la tienda de *Unreal Engine*, encontramos una herramienta llamada *VisAI* que en su descripción reflejaba que utilizaba GOAP para planificar las acciones de los individuos. Este hecho supuso una motivación extra a nuestro trabajo, ya que se nos presentaba la posibilidad de tener un espejo en el que poder comparar los resultados de nuestro trabajo.

Lamentablemente, tras probar *VisAI*, descubrimos que no utilizaba una arquitectura GOAP para la planificación de las acciones, sino que simulaba la aplicación de la lógica de GOAP para la elección de árboles de comportamiento. Dicho de otra manera, *VisAI* ofrecía árboles de comportamiento predefinidos que el desarrollador podía "combinar" de distintas maneras para crear su propia IA.

4.3. Resultado de la investigación

La realización de la comparativa entre los distintos modelos de planificación nos ha permitido poner de relieve las fortalezas y debilidades de cada planificador. Esta comparativa nos ha servido para decidir que el modelo de planificación que vamos a utilizar en nuestra herramienta será la arquitectura GOAP.

La decisión de elegir este modelo viene fundamentada principalmente en la gran escalabilidad que nos ofrece un sistema como *GOAP*, así como el realismo que se puede llegar a obtener utilizando dicho planificador. Aunque sabemos que *GOAP* se trata de un sistema de mayor complejidad que los planificadores reactivos, estamos convencidos de que puede ser positivo enfrentarnos al desafío de conseguir que una herramienta así sea fácil de utilizar por cualquier desarrollador.

Por otro lado, el análisis de mercado nos ha permitido descubrir la clara falta de una herramienta de planificación con arquitectura *GOAP* en la tienda de recursos de *Unreal Engine*. Por este motivo, consideramos que ayudaríamos a cubrir esa necesidad del mercado si realizamos el desarrollo de una herramienta para dicha plataforma.

Además, llevaremos a cabo la implementación de la herramienta de tal forma que se pueda integrar dentro del entorno de desarrollo de *Unreal Engine* de manera nativa. Este tipo de herramientas conocidas como *code plugins* (complementos de código, en inglés) tienen la peculiaridad de que permiten ser incluidas en proyectos ya creados, y no solamente en proyectos que empiecen desde cero.

Finalmente, con el objetivo de hacer la herramienta lo más accesible posible, vamos a preparar el *code plugin* para que el desarrollador pueda utilizar todas sus funcionalidades a través de *Blueprints*. Esto implica que nosotros prepararemos toda la implementación en código *C++*, para después facilitar una conexión mediante *Blueprints* con los que el desarrollador pueda acceder y trabajar con el planificador.

Capítulo 5

Análisis y diseño

"La función de un buen software es hacer que lo complejo aparente ser simple."
— Grady Booch

En este capítulo se pone de manifiesto cómo ha sido el análisis y el diseño software de nuestra herramienta, así como una explicación del proceso de desarrollo y de su seguimiento que hemos llevado a cabo durante este proyecto. Además, completamos el capítulo con el detalle del modelo de diseño aplicado en nuestro sistema.

5.1. Seguimiento del proyecto

Al igual que sucede con cualquier tipo de proyecto de software, hemos necesitado utilizar ciertos mecanismos de seguimiento que nos ayudasen a gestionar de la manera más óptima posible el desarrollo de nuestra herramienta. De dicho seguimiento podemos distinguir la parte relativa al uso de una metodología de trabajo concreta, y la parte referente a la coordinación y comunicación con nuestro tutor del Trabajo de Fin de Grado.

5.1.1. *Kanban*

Como se ha mencionado anteriormente, hemos utilizado *Kanban* como sistema para gestionar el flujo de trabajo y las tareas. En concreto se utilizó una herramienta online llamada *Jira* que ofrece un tablero *Kanban* con

distintos estados (véase Figura 5.1). Durante el proceso de desarrollo, cada tarea pasaba obligatoriamente por cuatro estados bien definidos.

En el primer estado, el estado *Pendiente*, se encontraban todas las tareas que quedaban por realizar y que estaban a la espera para ser desarrolladas. En el segundo estado, *En proceso*, estaban aquellas en desarrollo y que, una vez finalizadas, pasaban al siguiente estado, *En prueba*, donde se realizaban distintas pruebas: unitarias, de componentes, y otras. Si estas tareas conseguían superar todas las pruebas, pasaban al último estado, *Terminado*, donde se encuentran todas las funcionalidades completamente desarrolladas y probadas. Si alguna tarea fallaba en las pruebas, esta retrocedía a la anterior fase y se aumentaba la prioridad de la misma para que no retrasara el proyecto.

Las tareas tienen un título que funciona como resumen, una descripción más detallada sobre lo que se pretende conseguir al realizar dicha tarea, y como hemos dicho, una prioridad que varía entre *lowest*, *low*, *medium*, *high* y *highest* (de menor a mayor). También se incluye como responsable de la tarea a uno de los integrantes del grupo, y se pueden incluir dependencias si la acción debe realizarse una vez otra ha sido desarrollada, y otros datos como fechas o etiquetas.

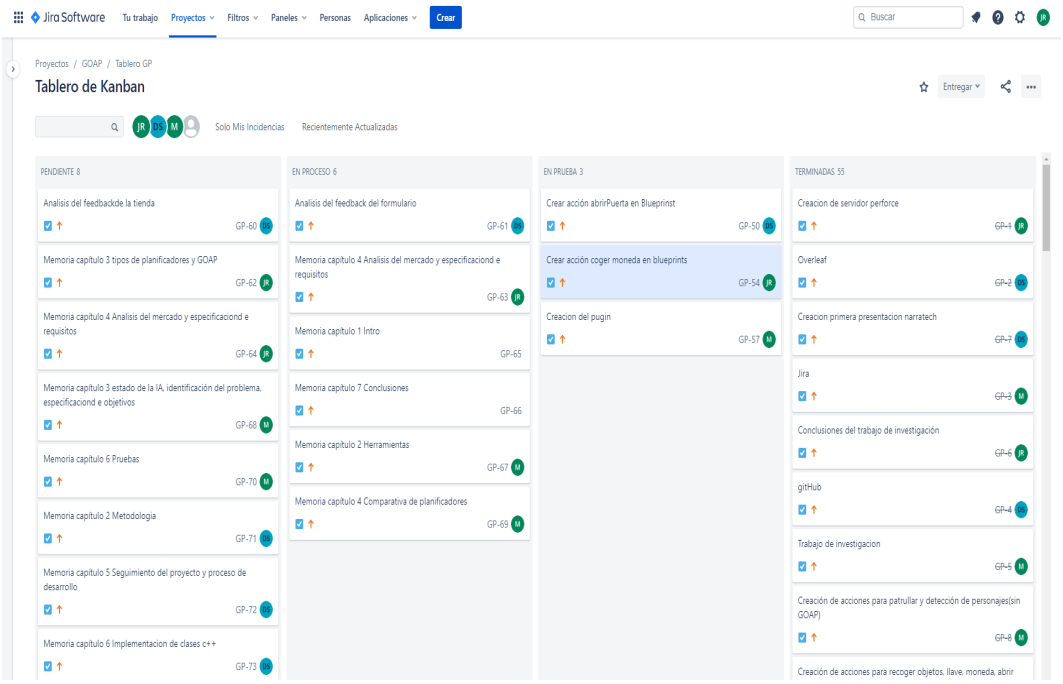

Figura 5.1: Captura del tablero *Kanban* utilizado en *Jira*.

5.1.2. Reuniones

De media una vez cada dos semanas realizábamos una reunión conjunta con el director del trabajo y todos los integrantes del equipo. En algunos periodos de tiempo las reuniones se realizaban con más frecuencia debido a la importancia del tema a tratar, como puede ser un bloqueo en el proceso de investigación o desarrollo, así como para las pruebas internas y corrección de errores.

En todas las reuniones se tomaban actas sobre los temas tratados. Estas actas servían de resúmenes sobre los problemas encontrados y el trabajo que se iba a desarrollar en las semanas siguientes. En el Apéndice E se encuentran los mencionados resúmenes que extraíamos de las reuniones.

También se realizaron presentaciones grupales organizadas por *Narratech Laboratories* donde distintos desarrolladores exponían sus avances sobre Trabajos Fin de Grado, Trabajos Fin de Máster y Tesis Doctorales en marcha. Este tipo de reuniones nos permitían dar y recibir realimentación periódica entre distintos grupos de trabajo, siendo muy enriquecedoras ya que los temas tratados estaban directamente relacionados con el ámbito de la IA. En total tuvimos un total de tres presentaciones a lo largo del curso, detallándose también en el Apéndice E un resumen del contenido de las mismas.

5.2. Proceso de desarrollo

Para realizar el desarrollo de la herramienta se eligió utilizar el entorno de desarrollo *Unreal Engine*. Debido a que uno de los objetivos del proyecto era publicar el *code plugin* en la tienda de contenidos de *Unreal Markerplace*, era requisito indispensable utilizar la última versión disponible de dicho motor, es decir, la versión 4.25.

5.2.1. Prototipo

Antes de comenzar con la implementación propiamente dicha de la herramienta, realizamos un prototipo que nos sirviese para aprender correctamente el funcionamiento interno de *Unreal Engine 4*.

En este prototipo, llevamos a cabo la implementación de una IA sin planificación dinámica que requiriese gestionar unas pocas acciones como recoger elementos del entorno e interactuar con ellos sobre otros.

El objetivo era adquirir cierto nivel de conocimiento sobre *Unreal Engine* y la planificación de IA para encontrarnos mejor preparados a la hora de desarrollar algoritmos más complejos.

5.2.2. Implementación funcional

Tras realizar el prototipo, pasamos a desarrollar la implementación del código principal de la herramienta en lenguaje *C++*. En esta primera implementación se llevo a cabo la creación de todos los componentes del modelo GOAP, es decir, se desarrollo la aplicación del algoritmo de búsqueda A* junto con todas las demás clases que simulaban acciones, estados del mundo, planificador y controlador.

5.2.3. Implementación modular

Una vez tenemos realizada la implementación de la herramienta íntegramente en código *C++*, pasamos a realizar una segunda implementación de la herramienta en la que preparamos en código *C++* la edición de algunas clases sobre *Blueprints*. Este segundo desarrollo se lleva a cabo porque queremos que la herramienta sea lo más modular y fácil de utilizar posible, ya que este es también otro de los objetivos de nuestro proyecto.

En esta segunda implementación, se ha tratado de hacer más accesible la herramienta a los desarrolladores. Esto ha sido posible gracias a que los *Blueprints* ofrecen una capa de interfaz mas fácilmente comprensible por el usuario de la herramienta. Con esta intención, le otorgamos flexibilidad al desarrollador para integrar con total facilidad nuestra herramienta desarrollada como *code plugin* en cualquier proyecto de *Unreal Engine*.

Cabe mencionar que tanto en la primera implementación en código *C++*, así como en la implementación preparada para la edición de algunas clases sobre *Blueprints*, se llevan a cabo las correspondientes pruebas para asegurarnos del correcto funcionamiento de ambas alternativas.

La segunda implementación es la que preparamos como *code plugin* para poder subirla a la tienda de contenidos de *Unreal Engine*. Además, esta implementación de la herramienta es la que utilizamos para la realización de pruebas con usuarios, para lo cual llevamos a cabo una encuesta formada por un fase de pruebas y un cuestionario posterior, con el objetivo de obtener valoraciones directas sobre el manejo de nuestra herramienta que nos permitiese mejorar el proyecto.

5.3. Modelo de diseño

Ahora pasaremos a detallar una serie de diagramas de diversa índole que, por su relevancia dentro del proyecto, consideramos importante destacar. Este tipo de diagramas nos permiten obtener una visión global del contenido del proyecto desde el punto de vista del diseño software.

5.3.1. Diagramas de clases

Aquí se muestra el diagrama que incluye todas las clases que componen la herramienta, así como las dependencias que hay entre ellas y las dependencias con aquellas clases propias de *Unreal Engine* (véase Figura 5.2).

Para facilitar la lectura del diagrama, a continuación se detallan los diagramas de clases de *Action* (véase Figura 5.3) y *Controller* (véase Figura 5.4) específicamente por separado, al tratarse de las clases más relevantes del proyecto.

Action es la clase padre de la cual todas las acciones que pueda realizar el PNJ heredan. Los objetos de tipo *Action* deben llevar asociados unas precondiciones y unos efectos, expresados como clases *WorldState*.

La clase *Controller* es la que se utiliza para crear el controlador de cada PNJ. Esta clase hereda de la clase *AIController* de *Unreal Engine*, y añade funcionalidades extra para gestionar el planificador de la clase *Planner* junto a la lista de acciones de tipo *Action* disponibles para su uso.

5.3.2. Diagramas de flujo

De los diagramas de flujo realizados, consideramos oportuno mencionar especialmente el método de *generatePlan* que pertenece a la clase *Controller* (véase Figura 5.5).

Dicho método se encarga de la creación de un plan para conseguir que el estado del mundo inicial incluya el mundo meta a partir del uso de las acciones asociadas a este controlador. La lógica aplicada en este método es la misma que se utiliza en el algoritmo de búsqueda A*, con el objetivo de encontrar el camino de acciones de tipo *Action* más óptimo para llegar de un *WorldState* a otro.

5.3.3. Diagramas de secuencia

Uno de los métodos fundamentales de la clase *Controller* es *beginPlay*, que se ejecuta una vez se han inicializado todos los componentes del escenario (véase Figura 5.6).

En el primer bucle de dicho método se recogen todas las acciones que se le asignaron al controlador. Después, en los dos siguientes bucles se transforman los estados del mundo que están en formato *Atom* (pares clave-valor con una cadena de texto y un parámetro binario) a estados del mundo en formato *WorldState*. En el último bucle se crean las precondiciones y los efectos de cada acción. Finalmente, se inicializa el planificador con la lista de acciones, el mundo inicial y el mundo meta.

Para crear las precondiciones y los efectos de cada acción se ha implementado el método *createPE* en la clase *Action* (véase Figura 5.7). Este método se encarga de crear un *WorldState* para cada lista de precondiciones y efectos.

Otro de los métodos mas importantes de la herramienta es *executeGOAP*, que pertenece a la clase *Controller* (véase Figura 5.8). Este método es el encargado de crear y ejecutar el planificador siguiendo la arquitectura GOAP.

En primer lugar, se crea un nuevo plan a través del planificador de tipo *Planner*. Después, se ejecuta la primera acción devuelta por el planificador, lo que implica que se modifique también el estado del mundo actual con los efectos de dicha acción. Finalmente, el método *executeGOAP* devuelve un valor booleano al controlador indicando si la ejecución del planificador ha encontrado una solución al problema o no.

El método empleado en *executeGOAP* para crear el plan es *generatePlan*, que está implementado en la clase *Planner* (véase Figura 5.9). El funcionamiento de dicho método ha sido previamente descrito en el diagrama de flujo asociado al mismo.

A continuación vamos a tratar en detalle el funcionamiento del método *getAdjacent* que se utiliza en *generatePlan* (véase Figura 5.10). El motivo de centrar nuestra atención en este método es porque aquí se encuentra una de los aspectos más evidentes de la aplicación del modelo GOAP.

El método *getAdjacent* se encarga de devolver la lista de acciones que el agente puede realizar en el estado del mundo inmediatamente después de la realización de la acción actual. Este método tiene el papel fundamental de iniciar el proceso de transición entre dos acciones, es decir, es el principio de la construcción dinámica del plan de comportamiento de la IA.

Por último, creemos que sería conveniente detallar el funcionamiento del método *isIncluded* de la clase *WorldState* (véase Figura 5.11). Este método es el que nos permite comprobar si un estado del mundo esta teniendo lugar en otro, es decir, si la lista de pares clave-valor de un *WorldState* esta incluida en otro *WorldState*.

El método *isIncluded* es el que utilizamos para comprobar si se puede realizar una acción en *getAdjacent*, o también para saber si hemos alcanzado el mundo meta en *generatePlan*.

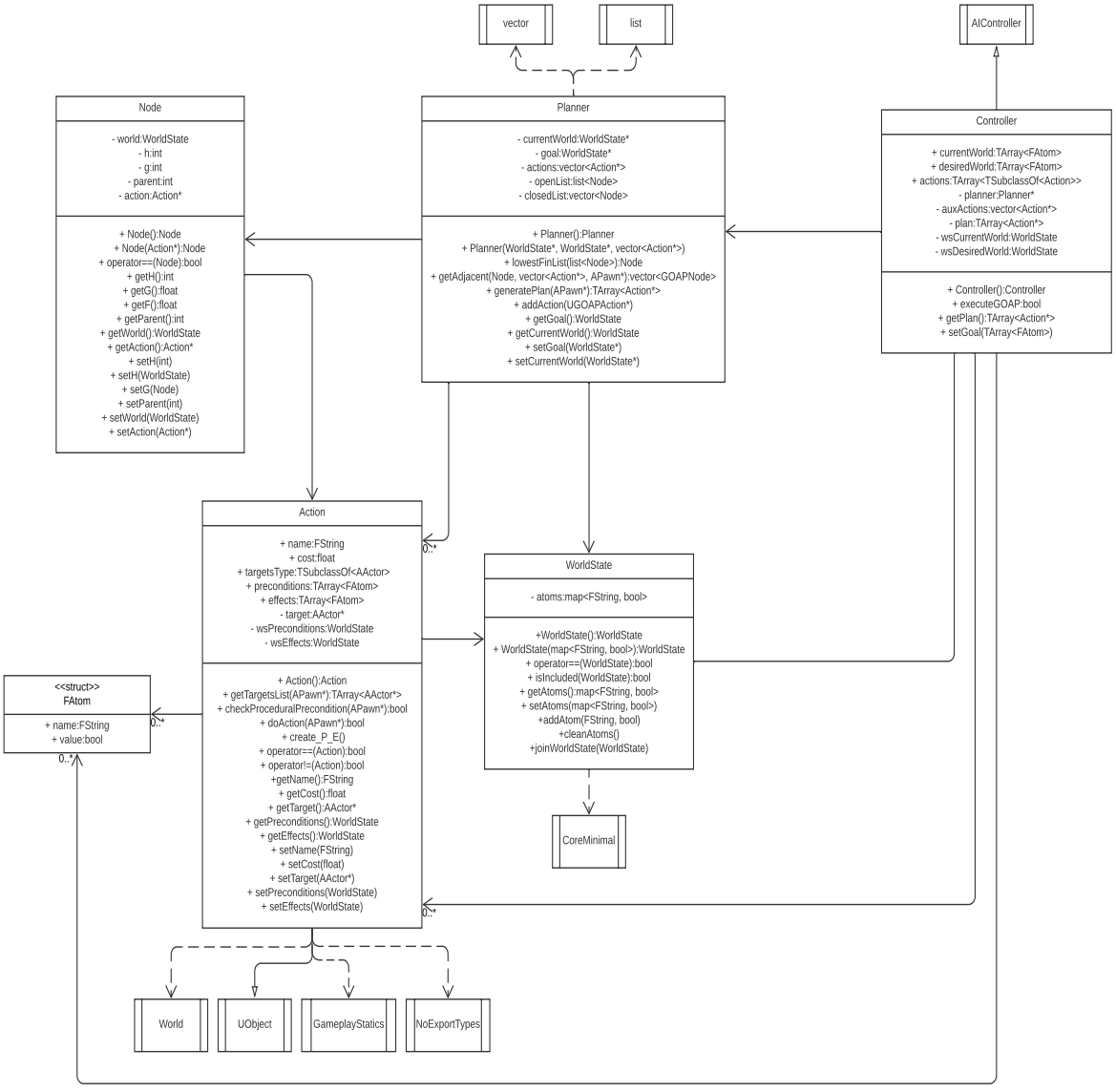

Figura 5.2: Diagrama de clases de la herramienta.

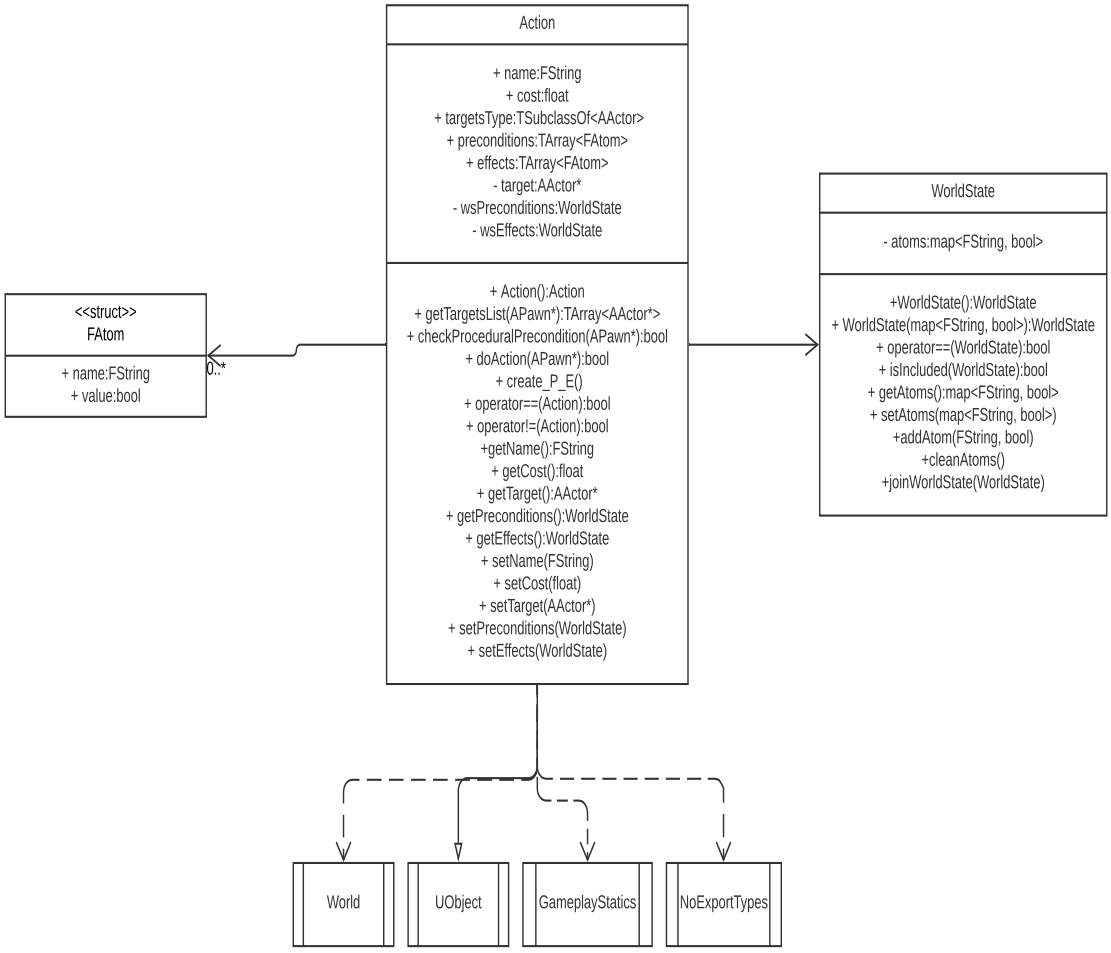

Figura 5.3: Diagrama de clase de *Action*.

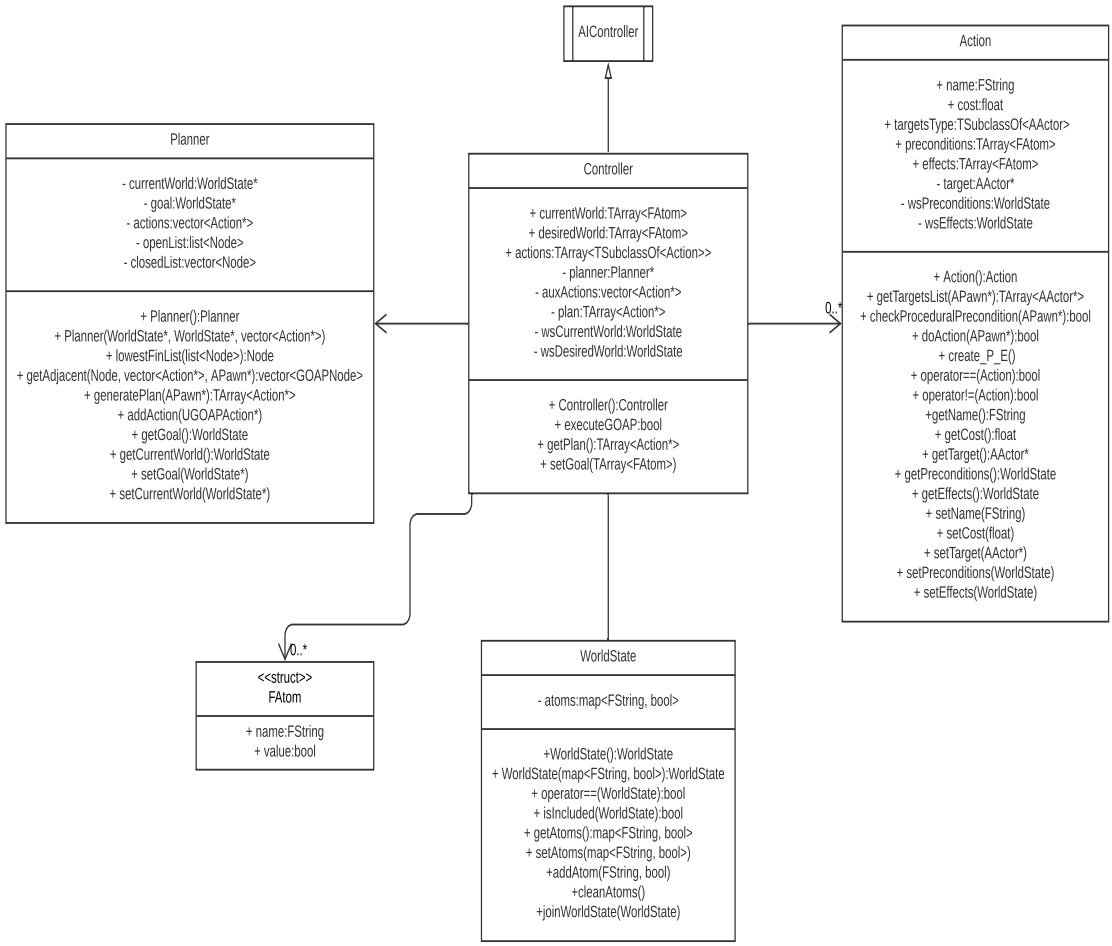

Figura 5.4: Diagrama de clase de *Controller*.

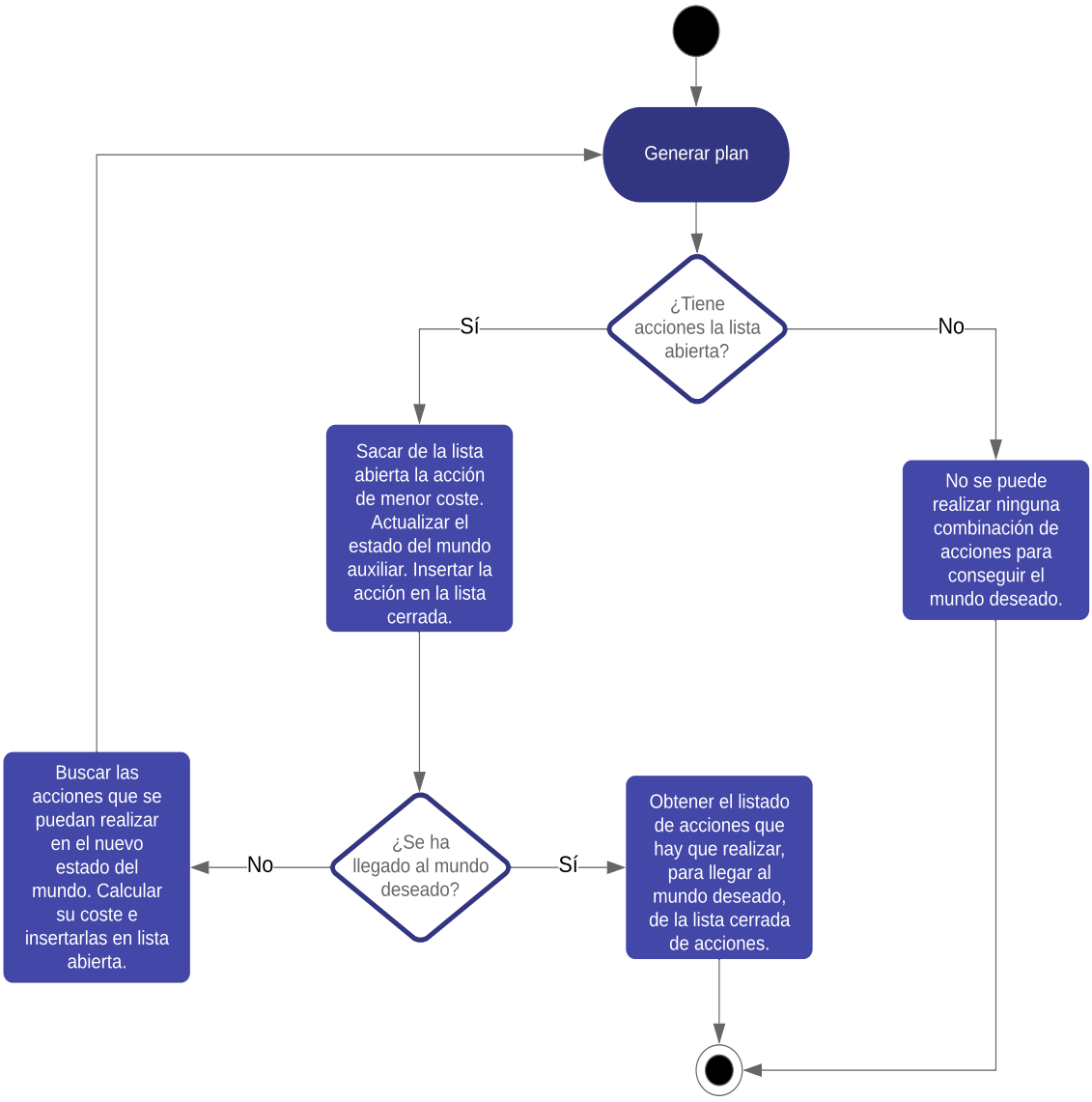

Figura 5.5: Diagrama de flujo de *generatePlan*.

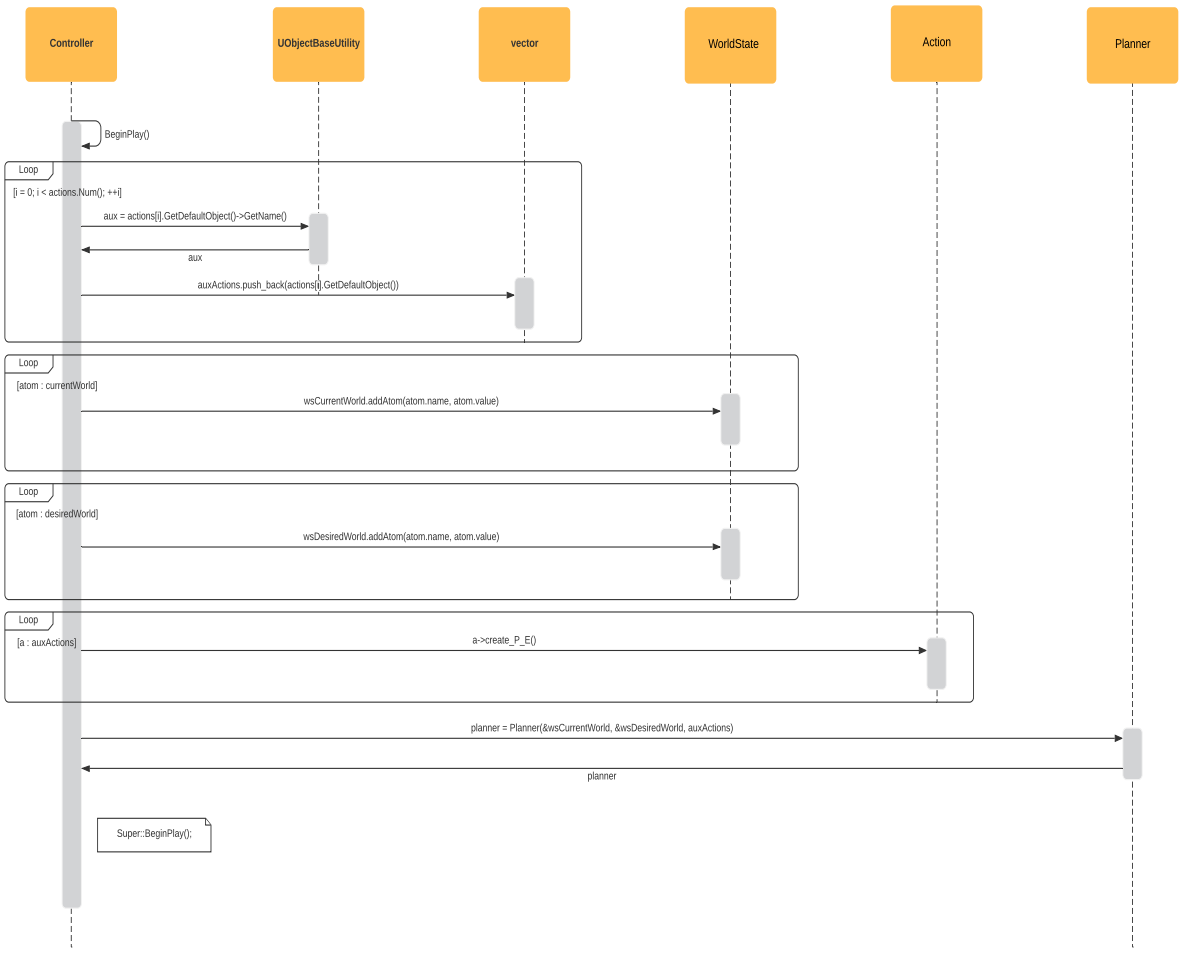

Figura 5.6: Diagrama de secuencia de *beginPlay*.

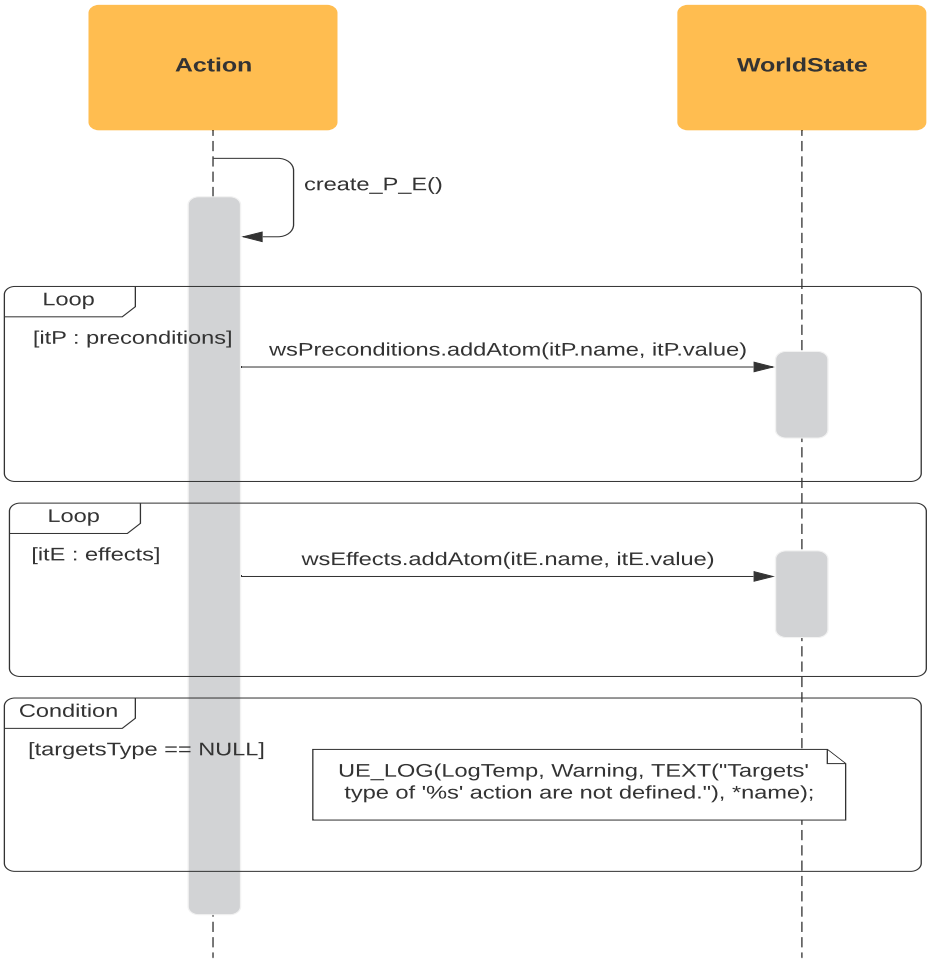

Figura 5.7: Diagrama de secuencia de *createPE*.

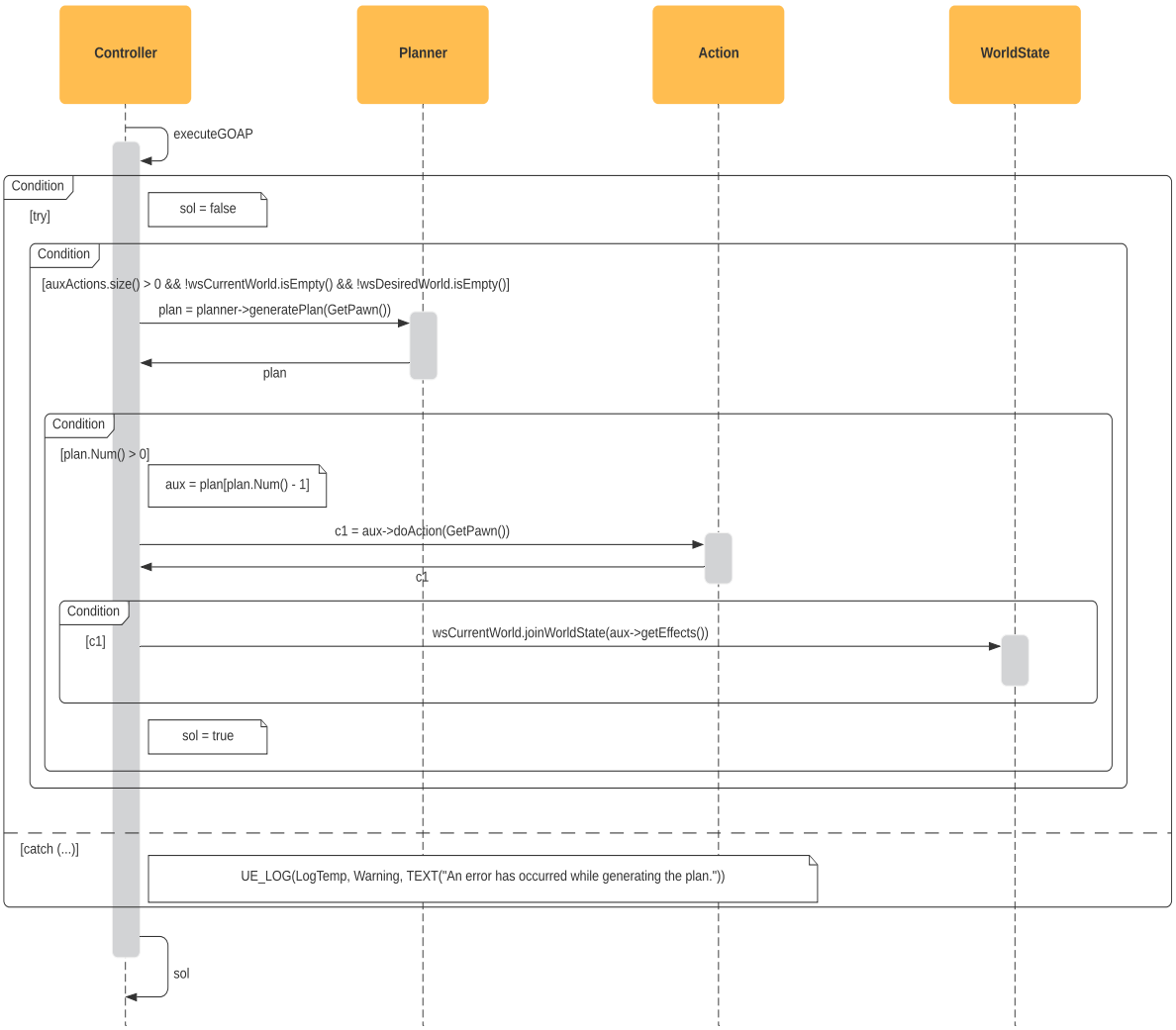

Figura 5.8: Diagrama de secuencia de *executeGOAP*.

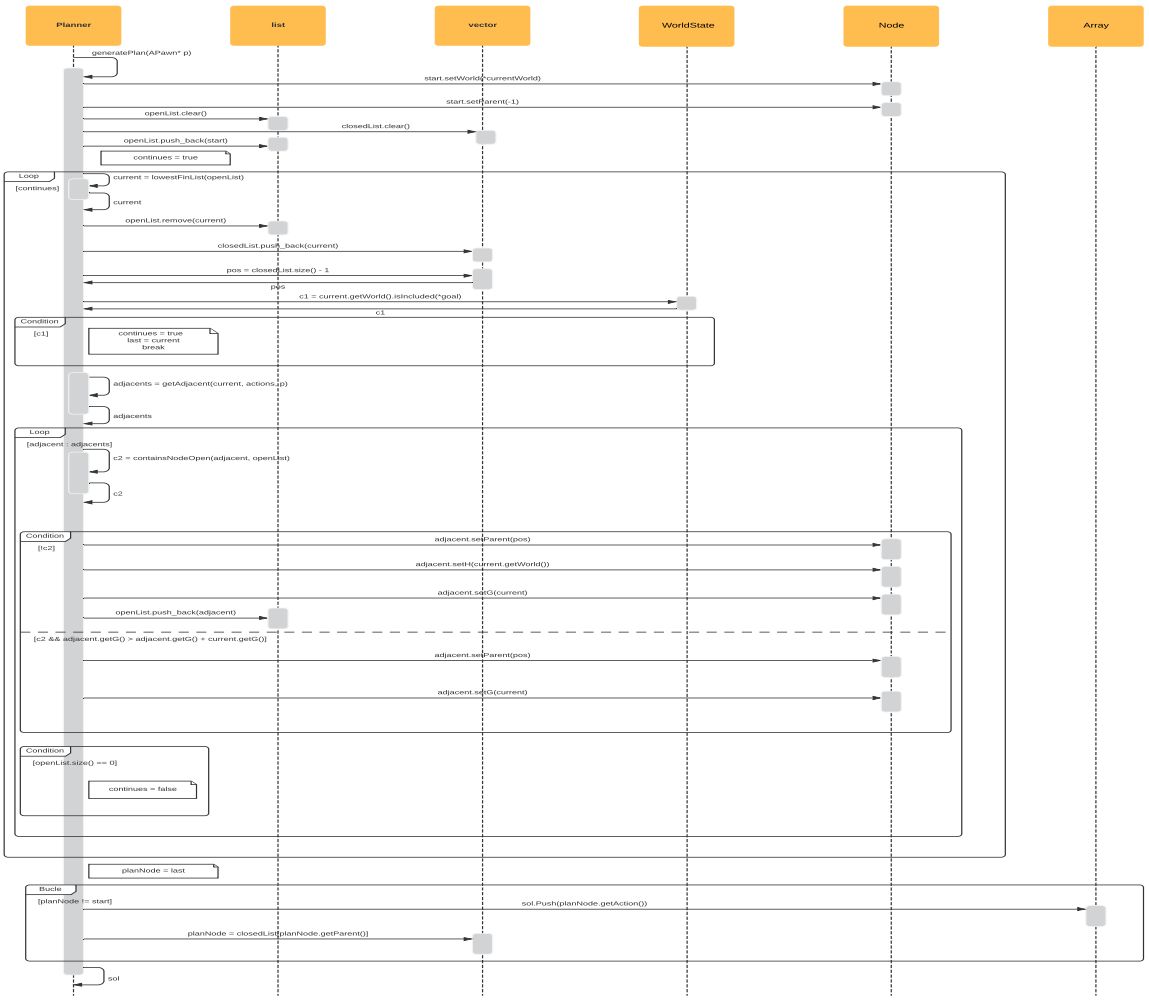

Figura 5.9: Diagrama de secuencia de *generatePlan*.

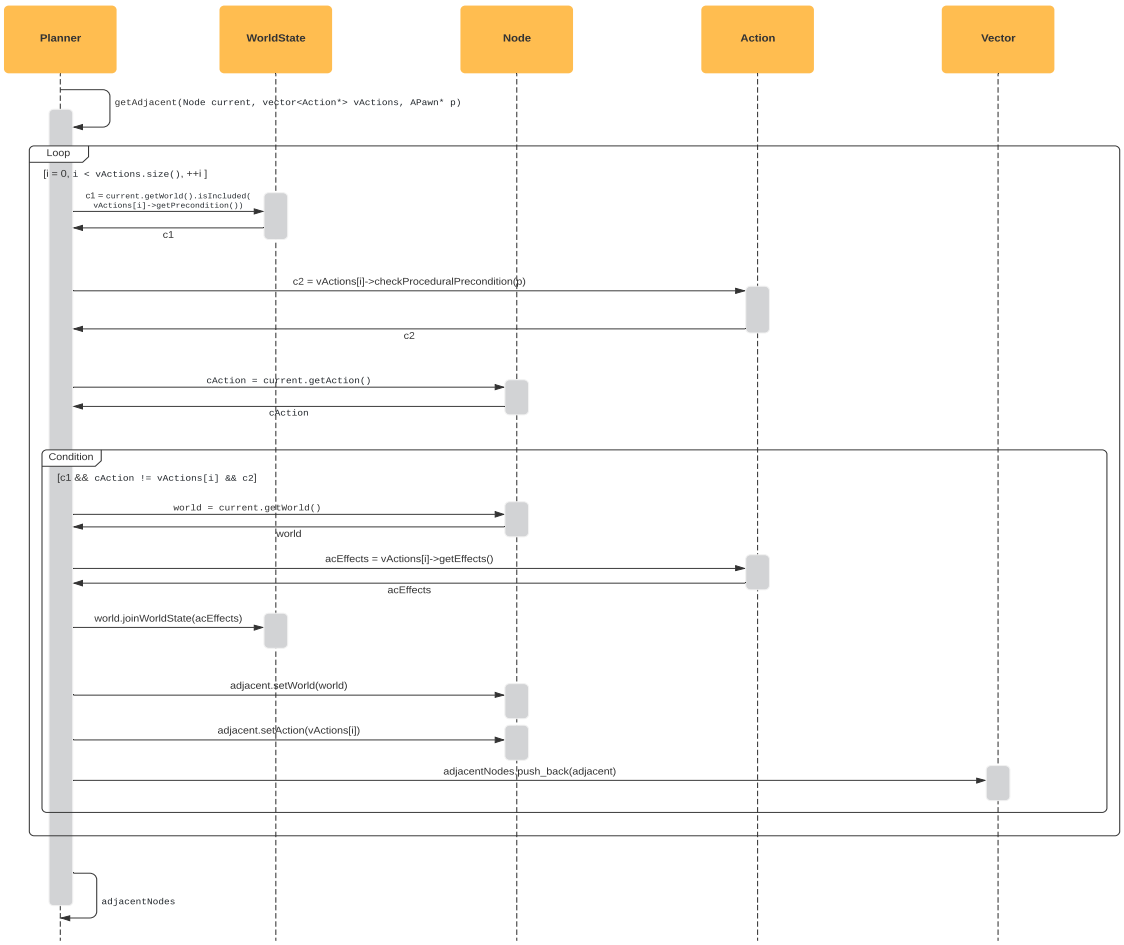

Figura 5.10: Diagrama de secuencia de *getAdjacent*.

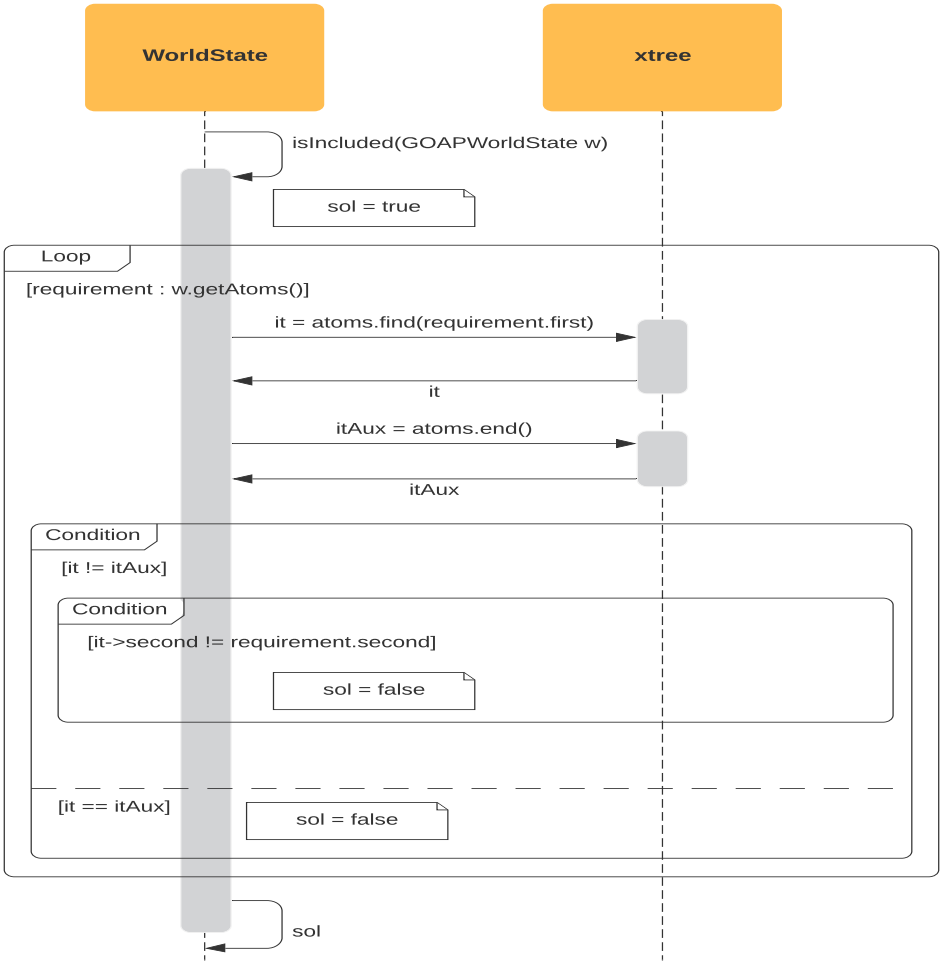

Figura 5.11: Diagrama de secuencia de *isIncluded*.

Capítulo 6

Implementación y pruebas

"Medir el progreso del desarrollo de software por líneas de código es como medir el progreso de la construcción de un avión por su peso."
— Bill Gates

Dado que la herramienta ha sido desarrollada como un *code plugin* para *Unreal Engine*, consideramos necesario distinguir entre la parte implementada en código *C++* que incluye toda la arquitectura GOAP, de la parte que está preparada para ser heredada mediante *Blueprints*. Además, en este capítulo se encuentra disponible una descripción de las pruebas que se han realizado sobre la herramienta, utilizando un escenario de demostración, así como los resultados obtenidos en las mismas.

6.1. Implementación de clases *C++*

En la implementación de nuestra herramienta hemos desarrollado cinco clases *C++* y un *struct* auxiliar (*FAtom*). Para ello, hemos declarado cada clase y sus funciones en archivos **.H**, mientras que hemos utilizado los archivos **.CPP** para incluir las implementaciones de todos los métodos de dichas clases. A continuación entraremos en detalle de cada una de estas clases, incidiendo especialmente en aquellas funcionalidades más relevantes.

6.1.1. *Action*

Action es la clase que contiene los atributos y la funcionalidad de una acción. Cada elemento de tipo *Action* tiene un atributo *name* que permite identificarlo y distinguirlo del resto.

Como ya hemos mencionado anteriormente, esta clase incluye como atributos la lista de precondiciones que se deben cumplir en el mundo actual para poder realizar la propia acción. De la misma manera, la clase también incluye la lista de efectos que provoca sobre el mundo actual al completarse la acción. Tanto las precondiciones como los efectos se expresan mediante atributos de tipo *WorldState*.

Por otro lado, la clase lleva un atributo que indica el tipo de objetivo de la acción. Este atributo se utiliza para definir el tipo que tiene el actor sobre el que se realizará la acción. El objetivo de este atributo es evitar la duplicidad de acciones que realicen el mismo comportamiento sobre distintos actores.

Esta clase es *Blueprintable*, lo que quiere decir que desde el editor de *Unreal Engine* se pueden generar *Blueprints* que hereden de ella. Además, tiene dos funciones (*doAction* y *checkProceduralPrecondition*) que se implementan directamente a través de *Blueprints* por el desarrollador.

6.1.2. *WorldState*

WorldState representa el estado del mundo y está compuesto por *atoms*, que son predicados los cuales representan las características que definen el mundo. Los predicados son pares clave-valor de tipo *String* y *Boolean*, que se guardan en un lista de tipo *map*.

Esta clase incluye métodos que facilitan las comprobaciones entre distintos elementos de tipo *WorldState*, así como un método para añadir y modificar el estado del mundo aplicando la lógica del modelo GOAP.

6.1.3. *Planner*

La clase *Planner* es el núcleo de GOAP, ya que contiene y gestiona la lógica del planificador de acciones. Esta clase recibe una lista de acciones de tipo *Action*, así como los estados del mundo inicial y meta de tipo *WorldState*.

En *Planner* se han implementado los métodos necesarios para poder ge-

nerar el plan de acciones menos costoso utilizando el algoritmo A* en un marco de arquitectura GOAP. Utilizando esta clase se generan los árboles de nodos con las soluciones a problemas de planificación, en los que cada nodo se representa mediante la clase *Node*.

El nodo inicial es el estado actual del mundo, mientras que el nodo final es el estado del mundo deseado. Los nodos intermedios son estados de mundos posibles, mientras que las aristas son las acciones disponibles.

Como elementos de apoyo en el desarrollo del algoritmo A*, esta clase contiene dos listas de nodos: *openList* y *closedList* (lista abierta y lista cerrada, en inglés, respectivamente). En la lista abierta se encuentran los nodos a los que se puede acceder, pero todavía no han sido explorados; mientras que la lista cerrada contiene los nodos que ya han sido visitados.

6.1.4. *Node*

Node es una clase auxiliar para representar los nodos dentro del algoritmo A* del planificador. Como representante de un nodo, contiene el estado del mundo actual de tipo *WorldState*, así como la acción de tipo *Action* que se ha realizado para llegar a él.

Esta clase también contiene la información del coste G, que es el acumulado desde el nodo inicial hasta llegar a él, y el coste H, que es el coste heurístico, equivalente al número de predicados distintos entre el estado del mundo del nodo actual y el del nodo final. Ambos costes sumados dan el valor de F, que es la función de evaluación, por la que se verán seleccionados cuando se encuentren en la lista abierta, en orden creciente.

6.1.5. *Controller*

La clase *Controller* hereda de *AIController*, y se encarga de gestionar las acciones disponibles, el estado del mundo actual y el estado del mundo meta, así como las llamadas al planificador. *Controller* representa la IA del agente, ya que sobre esta clase recae la responsabilidad de la toma de decisiones en base a la información de la que dispone.

Además, esta es una clase *Blueprintable* y, al igual que sucedía con *Action*, está preparada para que el desarrollador genere *Blueprints* que hereden de ella. Esto permite al desarrollador que pueda adaptar la IA a su gusto según las necesidades de cada proyecto.

6.2. Herencia mediante *Blueprints*

Para que un desarrollador pueda utilizar el complemento de código que hemos desarrollado (véase Figura 6.1), se le ofrece la posibilidad de crear directamente *Blueprints* que hereden de las dos clases que hemos habilitado como *Blueprintable*: *Action* y *Controller* (véase Figura 6.2).

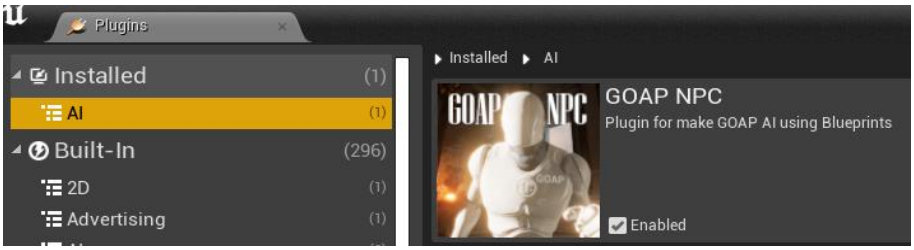

Figura 6.1: Instalación del *code plugin* en *Unreal Engine*.

Por cada acción que queramos que pueda realizar nuestra IA, habrá que crear un *Blueprint* que herede de *Action*. De la misma manera, por cada IA distinta que queramos tener, habrá que crear un *Blueprint* que herede de *Controller*.

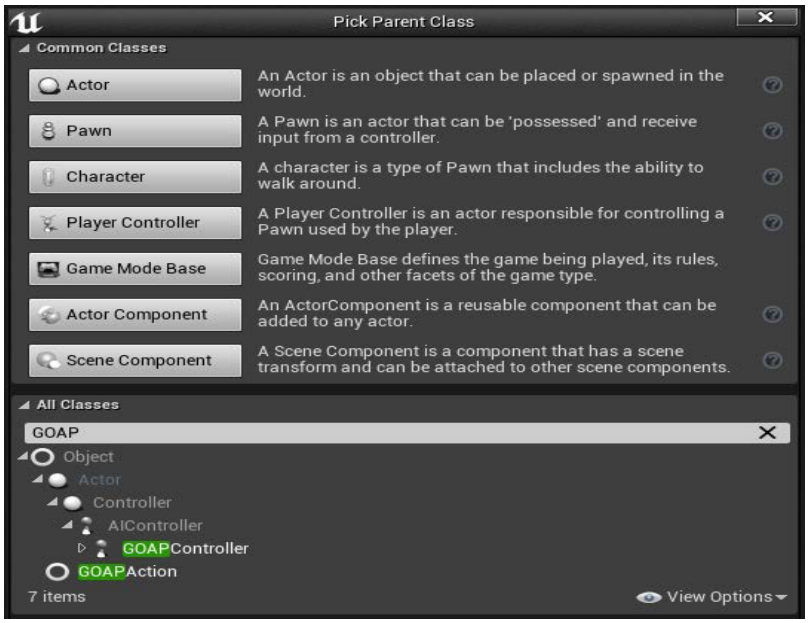

Figura 6.2: Creación de *Blueprints* a partir de las clases del *code plugin*.

Sin embargo, esto no quiere decir que tengamos que crear una IA por cada PNJ, ya que cada instancia creada tanto de *Action* como *Controller*

son reutilizables para distintos individuos al no estar ligadas exclusivamente a un único agente.

6.2.1. *Action*

Los *Blueprints* que heredan de *Action* contienen las características y la lógica que conforman una acción dentro de una arquitectura GOAP. Sobre estos *Blueprints* el desarrollador podrá editar el comportamiento que quiera que realice el PNJ al ejecutar la acción, así como añadir, modificar o quitar precondiciones y efectos de la propia acción (véase Figura 6.3).

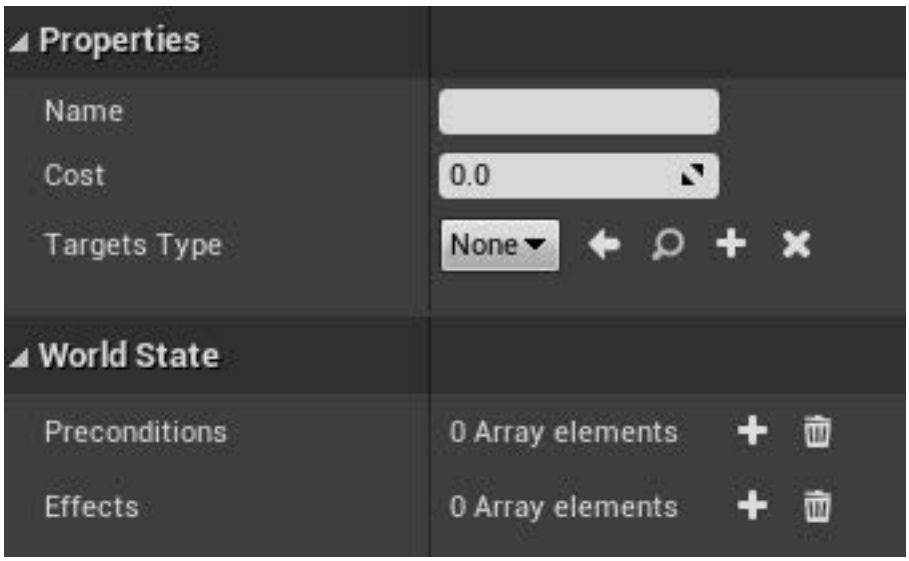

Figura 6.3: Interfaz para la edición de los atributos de *Action*.

También se han preparado varios métodos para que el desarrollador pueda implementarlas a su gusto en base a las necesidad de cada proyecto. Una de estas funciones es *doAction*, en la que se desarrolla la definición del comportamiento de la acción, es decir, lo que se espera que realice el PNJ al ejecutar la acción.

Por otro lado, el método *checkProceduralPrecondition* sirve para establecer los requisitos necesarios para que la acción pueda ser realizada, más allá de que se cumplan las precondiciones para poder realizar la acción. Este método es muy útil en el caso de que el desarrollador necesite implementar condiciones adicionales que resulte mas cómodo expresarlas en una función, en vez de pares clave-valor en la lista de precondiciones (véase Figura 6.4).

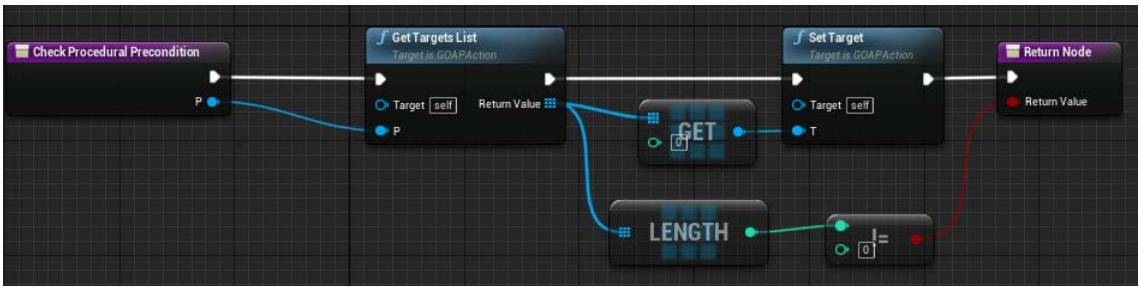

Figura 6.4: Ejemplo de uso del método *checkProceduralPrecondition*.

6.2.2. *Controller*

Controller es una clase que hereda de *AIController*, lo que le dota de las funcionalidades básicas de un controlador de IA. Además, esta enriquecido con los atributos y funciones necesarios para ejecutar GOAP (véase Figura 6.5).

Figura 6.5: Interfaz para la edición de los atributos de *Controller*.

El estado del mundo actual y el estado del mundo meta se pueden añadir fácilmente a través de la interfaz del *Controller* como si de un formulario se tratase (véase Figura 6.6). Al igual que sucede con los atributos de *Action*, hemos tratado de implementar la herramienta lo más accesible posible a cualquier tipo de desarrollador, ya sea con mayor o menor experiencia en la programación en *Unreal Engine*.

De la misma forma, añadir o quitar una acción sobre la lista de acciones disponibles de la IA es tan sencillo como hacer la modificación deseada en el atributo *Actions* en *Controller* (véase Figura 6.7). Una vez más, se puede apreciar cómo la herramienta se ha preparado para que el desarrollador pueda fácilmente implementar GOAP de la manera menos compleja posible.

A parte de los métodos que posee gracias a la herencia de *AIController*, la clase *Controller* dispone de varias funciones adicionales que están directamente relacionadas con el funcionamiento de GOAP. Entre estos nuevos

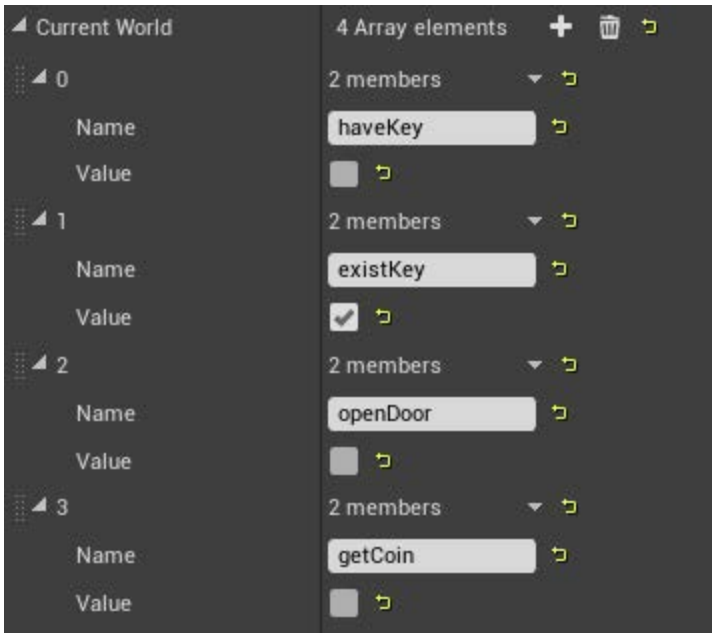

Figura 6.6: Ejemplo de edición del estado del mundo actual en *Controller*.

métodos cabe destacar *executeGOAP*, que es la función que se encarga de generar el plan e iniciar la ejecución de la primera acción del mismo. Este método puede ser llamado periódicamente en tiempo de ejecución para realizar la replanificación dinámica, permitiendo así la posibilidad de que el PNJ reaccione frente a cambios ajenos al mismo que se produzcan en el estado del mundo.

6.3. Demo

Para mostrar el funcionamiento de nuestra herramienta, hemos llevado a cabo la realización de un proyecto de demostración (véase Figura 6.8). Se trata de un escenario sencillo que podemos navegar donde hay una plataforma en el centro, encima de la cual se encuentra una moneda y una puerta que bloquea el único acceso a esa parte superior de la plataforma. Además, en la escenario también encontramos dos llaves y un PNJ.

El PNJ en cuestión es controlado por una IA que utiliza nuestro *code plugin* GOAP NPC para la planificación de su comportamiento. Su objetivo inicial es seguir al jugador (véase Figura 6.9), pero si se presiona en la tecla **Z**, su objetivo pasará a ser conseguir la moneda (véase Figura 6.10).

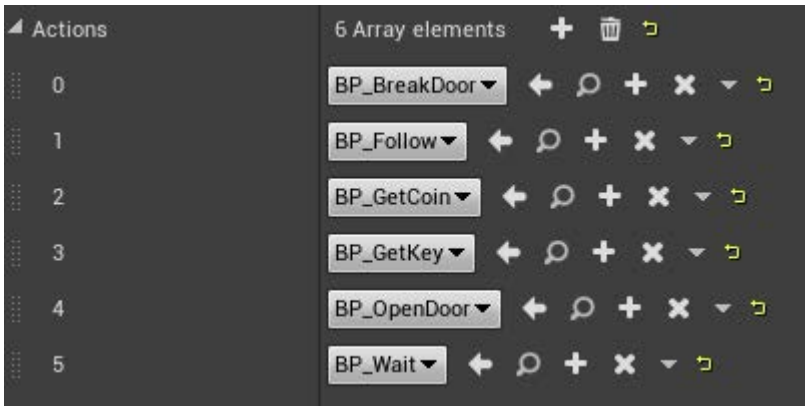

Figura 6.7: Ejemplo de edición de las acciones en *Controller*.

Así pues, para conseguir la moneda, la IA del PNJ tendrá que planificar qué acciones realizar para conseguir completar su objetivo. Para ello, hace uso de la planificación automática para generar en tiempo de ejecución el comportamiento a seguir por el agente.

Cabe mencionar que la IA planificará las acciones a desarrollar si se produce un cambio en el estado del mundo, como podría ser que desaparecieran las llaves, se abriese la puerta automáticamente o le cambiásemos de objetivo. Cada objetivo puede resolverse de varias maneras, por lo que sera la IA quién decida, en base a los costes de las acciones, cuál es el plan óptimo a seguir (véase Figura 6.11).

Para dar realimentación al usuario de la demo, el PNJ muestra sobre su cabeza un icono y un texto de color blanco que identifican la acción que esté realizando en ese momento, así como un texto de color verde que indica el objetivo actual que está tratando de alcanzar (véase Figura 6.12).

Por otro lado, utilizando las teclas **C** y **V** del teclado es posible visualizar en la parte superior izquierda de la pantalla los predicados de los estados del mundo actual y deseado. También es posible cambiar entre los objetivos entre seguir al jugador o conseguir la moneda, presionando las teclas **X** y **Z**, respectivamente. Además, con el personaje que controla el jugador podemos pasar por encima de las llaves para recogerlas, lo que provoca que el PNJ no pueda conseguir esa llave.

A continuación mostramos un enlace a un vídeo sobre se muestra un ejemplo de ejecución de la demostración: `https://www.youtube.com/watch?v=aInzVukXzAg`.

Figura 6.8: Vista aérea del escenario de la demo.

6.4. Pruebas

Las pruebas realizadas sobre nuestra herramienta se han llevado a cabo de manera continuada durante todo el desarrollo. Además, para que las pruebas fueran lo más completas posibles, se realizaron en ambos proyectos desarrollados, tanto en la herramienta como en el escenario de demostración.

En el primer proyecto, al estar todas las funcionalidades desarrolladas íntegramente en código C++, se llevaron a cabo todo tipo de pruebas a excepción de las pruebas con usuarios. Para realizar las pruebas sobre este proyecto específico, se ha empleado el uso de las herramientas disponibles en los entornos de *Visual Studio* y *Unreal Engine*.

En relación al segundo proyecto, que ha sido el que se ha preparado como *code plugin* para poder ser integrado como un modulo en *Unreal Engine*, se han llevado a cabo todo tipo de pruebas, siendo especialmente importantes las pruebas con usuarios reales que comentaremos al final de esta sección.

6.4.1. Unitarias

La realización de las pruebas unitarias se llevaron a cabo mediante el uso del *Automation System* (Sistema de Automatización, en inglés) de *Unreal Engine*, que permite la generación de pruebas automáticas a distintos niveles

Figura 6.9: PNJ realizando la acción de seguir al jugador.

de detalle.

Esta herramienta la hemos utilizado para verificar el correcto funcionamiento de las métodos de cada módulo. Gracias a estas pruebas hemos podido identificar y aislar errores específicos que nos han ido surgiendo durante la implementación del código.

6.4.2. De componentes

Las pruebas de componentes se han realizado sobre cada módulo o clase de la herramienta. Estas pruebas se realizaron también mediante el uso del *Automation System* de *Unreal Engine* (véase Figura 6.13).

Uno de los aspectos mas útiles de esta herramienta es la posibilidad de evaluar no sólo el correcto funcionamiento de cada componente, sino también obtener información relacionada con el rendimiento. Gracias a dicha información, hemos podido realizar refactorizaciones de código y mejoras de optimización.

6.4.3. Con usuarios reales

Uno de los procesos de pruebas más importante que hemos llevado a cabo ha sido las pruebas con usuarios reales, ya que tenemos el objetivo de

Figura 6.10: PNJ realizando la acción de conseguir una llave.

nuestra herramienta es que sea utilizada por desarrolladores independientes para facilitar el uso de técnicas de IA en los personajes de sus videojuegos.

Por un lado, preparamos un cuestionario que consistía en la realización de una prueba guiada de nuestro *code plugin*. En esta prueba se pedía al encuestado que desarrollase un entorno en el que fuese necesaria la creación de un planificador de IA para resolver un determinado problema.

Debido a la crisis ocasionada por la pandemia de la *COVID-19* (Organización Mundial de la Salud, 2020), las pruebas con usuarios no han podido realizarse de manera presencial como estaban previstas inicialmente. Debido a ello, se optó por distribuir el material y el enlace a un cuestionario en línea para nuestros probadores. Además, hemos publicitado e incentivado con el sorteo de un videojuego este evento de realización de pruebas a través del sitio web y las redes sociales de *Narratech Laboratories* (véase Figura 6.14).

Después de finalizar la prueba, se le pedía al encuestado que respondiese una serie de preguntas relacionadas con dicha prueba. Aprovechando las respuestas recibidas, hemos podido realizar un análisis sobre aspectos positivos y negativos de nuestra herramienta que nos ayudase a mejorar la calidad de nuestro producto, de cara a la publicación oficial en la tienda de recursos de *Unreal Engine*.

Una de las conclusiones más destacadas que hemos obtenido del cuestionario es que, pese a que gran parte de los encuestados desconocía la existencia de GOAP (véase Figura 6.15), han considerado que nuestra herramienta es

Figura 6.11: PNJ realizando la acción de romper la puerta.

muy fácil de utilizar (véase Figura 6.16). Esto nos da a entender que hemos sido capaces de cumplir el objetivo de realizar una aplicación adaptada a cualquier tipo de desarrollador, no sólo a los más experimentados y conocedores de técnicas de planificación automática. Además, 7 de cada 10 encuestados estaba seguro en que utilizaría el *code plugin* en sus proyectos de *Unreal Engine* (véase Figura 6.17), lo que apunta a una aceptación del mercado muy positiva.

En base a los resultados del cuestionario, se han llevado a cabo ciertas mejoras sobre nuestro *code plugin* y, finalmente, tras superar el proceso de certificación de Epic Games, ha sido aprobada su publicación en la tienda de recursos para el desarrollador de *Unreal Marketplace* (véase Figura 6.18). Esto nos ha permitido recibir aún más realimentación mediante valoraciones y comentarios que han ayudado a mejorar la calidad de la herramienta.

Figura 6.12: PNJ realizando la acción de recoger la moneda.

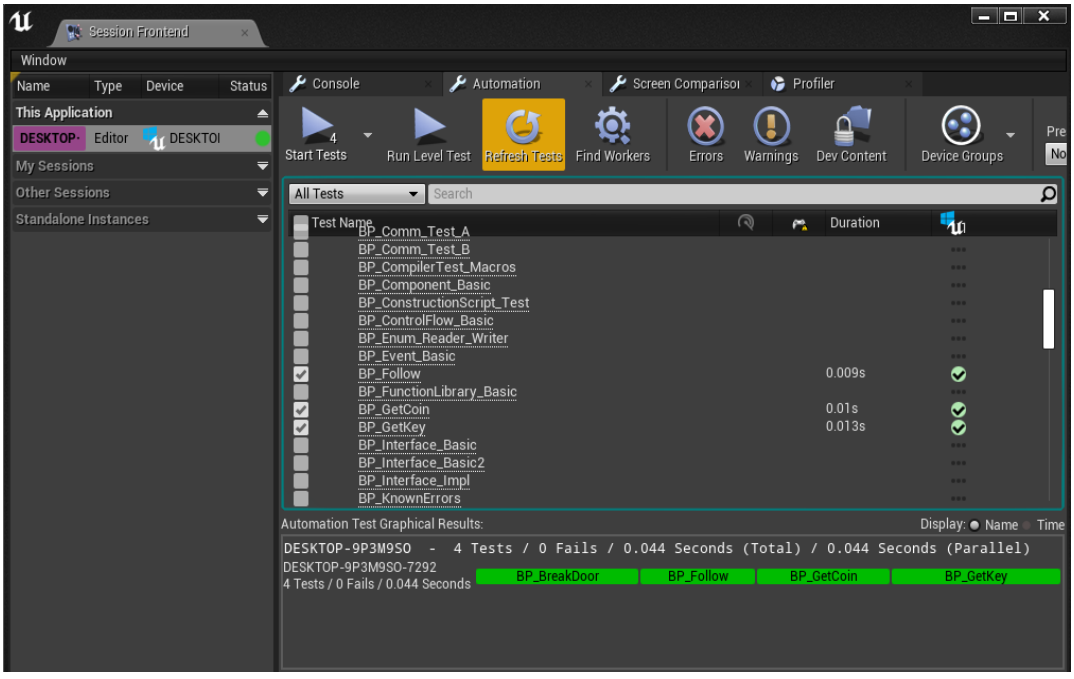

Figura 6.13: Resultado de pruebas realizadas con *Automation System*.

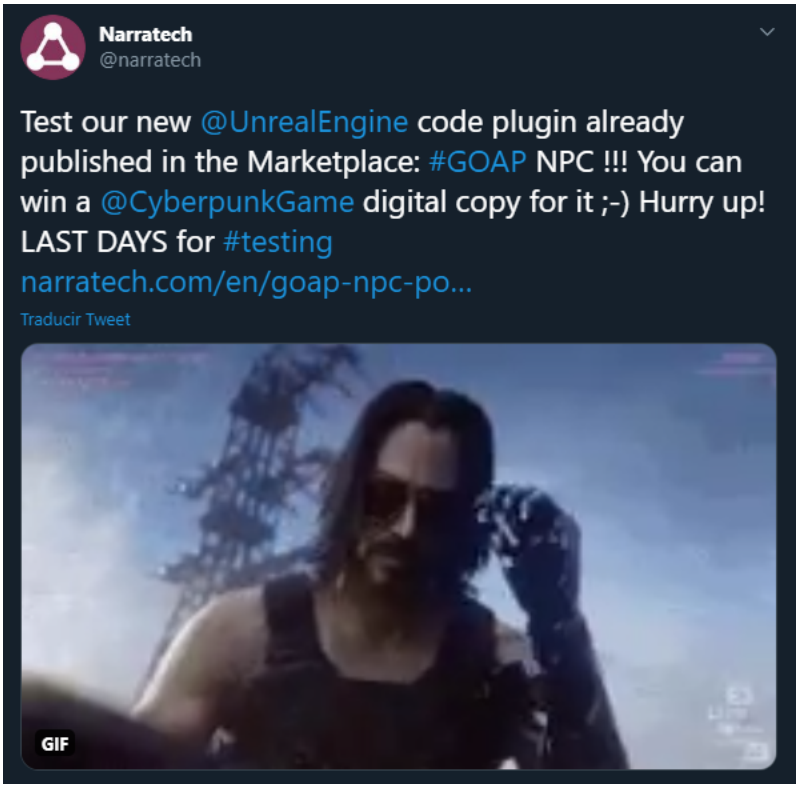

Figura 6.14: Tweet para promocionar las pruebas con usuarios.

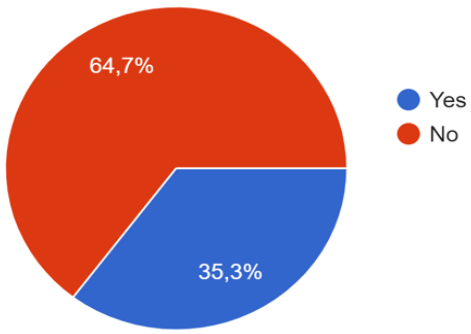

Figura 6.15: Proporción de los encuestados con conocimiento previo sobre GOAP.

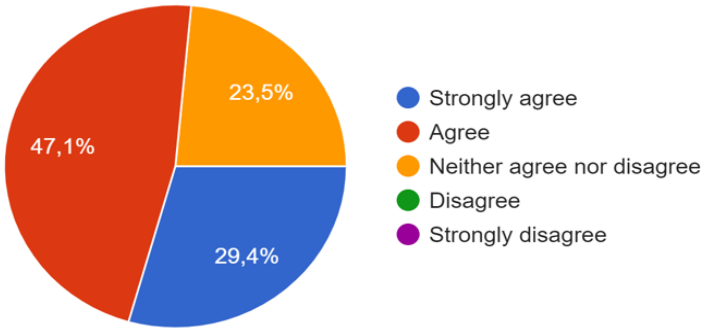

Figura 6.16: Valoración de los encuestados sobre la facilidad de uso del *code plugin*.

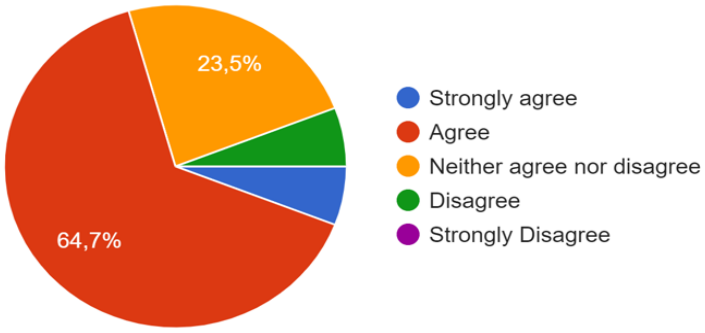

Figura 6.17: Valoración de los encuestados sobre la posibilidad de utilizar el *code plugin* en sus proyectos.

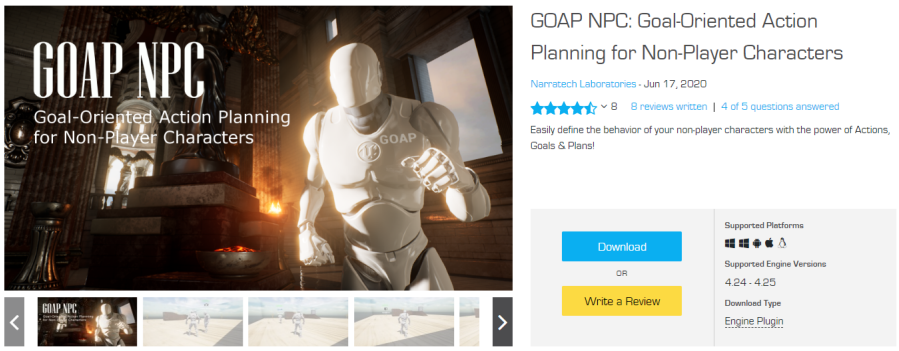

Figura 6.18: Página de nuestra herramienta en la tienda de recursos de *Unreal Engine*.

Capítulo 7

Conclusiones y trabajo futuro

"La mayoría de los buenos programadores no se dedican a la programación porque esperen que la gente les pague o les alabe, sino porque les divierte programar."
— Linus Torvalds

Tras haber concluido la realización del proyecto, aquí exponemos algunas reflexiones sobre el trabajo llevado a cabo y sus consecuencias. Además, enumeramos posibles mejoras o ampliaciones sobre nuestra herramienta de cara a futuros trabajos que puedan dar continuidad a este proyecto.

7.1. Conclusiones

Antes de comenzar con este proyecto, teníamos en mente que queríamos realizar algún tipo de contribución al mundo del desarrollo de videojuegos mediante la realización del Trabajo de Fin de Grado. En este sentido, y con la ayuda de nuestro director, indagamos sobre la posibilidad de desarrollar una herramienta de toma de decisiones con alguna técnica de IA aplicada a videojuegos.

En la investigación que llevamos a cabo sobre la evolución de la IA para videojuegos y el estado actual de la misma, llegamos a la conclusión de que existían grandes diferencias entre unos modelos de planificación y otros. En base a estas diferencias, elaboramos una comparativa de los planificadores en la que nos centramos en evaluar las fortalezas y debilidades de cada modelo.

Por otro lado, también realizamos una análisis del estado actual del mer-

cado de los desarrolladores de videojuegos, poniendo especial énfasis en la disponibilidad de herramientas de IA para que pequeños estudios o desarrolladores independientes puedan crear juegos con PNJs interesantes.

Esta doble labor de investigación nos permitió llegar a la conclusión de que había una evidente falta de recursos de IA para el desarrollo *indie* de videojuegos. Esta falta de recursos era especialmente notoria en la tienda de contenidos de *Unreal Engine*, *Unreal Marketplace*, donde no existía ninguna herramienta de IA que utilizase una arquitectura GOAP para la planificación automática del comportamiento de PNJs.

Así pues, nos marcamos como objetivo el desarrollo de una herramienta que permitiese cubrir esta necesidad del mercado. Para ello, primero consideramos que sería útil realizar un primer prototipo en el que probásemos por nosotros mismos las funcionalidades de *Unreal Engine* relacionadas con la IA.

Una vez realizada esta toma de contacto con *Unreal Engine*, nos vimos capacitados para llevar a cabo el diseño de la que iba a ser nuestra herramienta de planificación automática bajo una arquitectura GOAP.

A continuación procedimos a realizar la implementación de la herramienta íntegramente en código *C++*. En esta primera aproximación, el objetivo era cumplir con el aspecto funcional de la aplicación, es decir, conseguir que se planificase realmente de forma automática y en tiempo de ejecución.

Después de terminar de desarrollar dicha herramienta, nos centramos en alcanzar otro de los objetivos del proyecto. En este caso, quedaba pendiente por cubrir el aspecto práctico de la aplicación. Esto significaba que teníamos que conseguir que la herramienta fuese fácilmente accesible por cualquier desarrollador y se pudiese añadir a cualquier proyecto de *Unreal Engine*.

Por este motivo, realizamos una segunda implementación de la herramienta como *code plugin*. Este tipo de implementación nos permitía desarrollar las funcionalidades de la aplicación en código *C++*, y preparar ciertas clases para ser heredadas a través de *Blueprints*. De esta manera, cualquier usuario podría adaptar la arquitectura GOAP que hemos creado a las necesidades especificas de su proyecto.

Finalmente, tras haber realizado una fase de pruebas con usuarios, publicamos la herramienta en la tienda de recursos de *Unreal Engine*. Este hecho marcó el hito final de nuestro proyecto, ya que habíamos conseguido cumplir con todos los objetivos que establecimos en un principio y habíamos hecho una contribución real y tangible a la industria del videojuego, como fue siempre nuestro deseo.

Como último detalle, mencionar que este proyecto ha supuesto una experiencia muy gratificante para todos los miembros del grupo. Hemos aprendido muchísimo sobre el proceso que supone llevar a cabo el desarrollo de una aplicación desde cero hasta publicarla en un medio oficial para su uso y distribución por parte de la comunidad de desarrolladores.

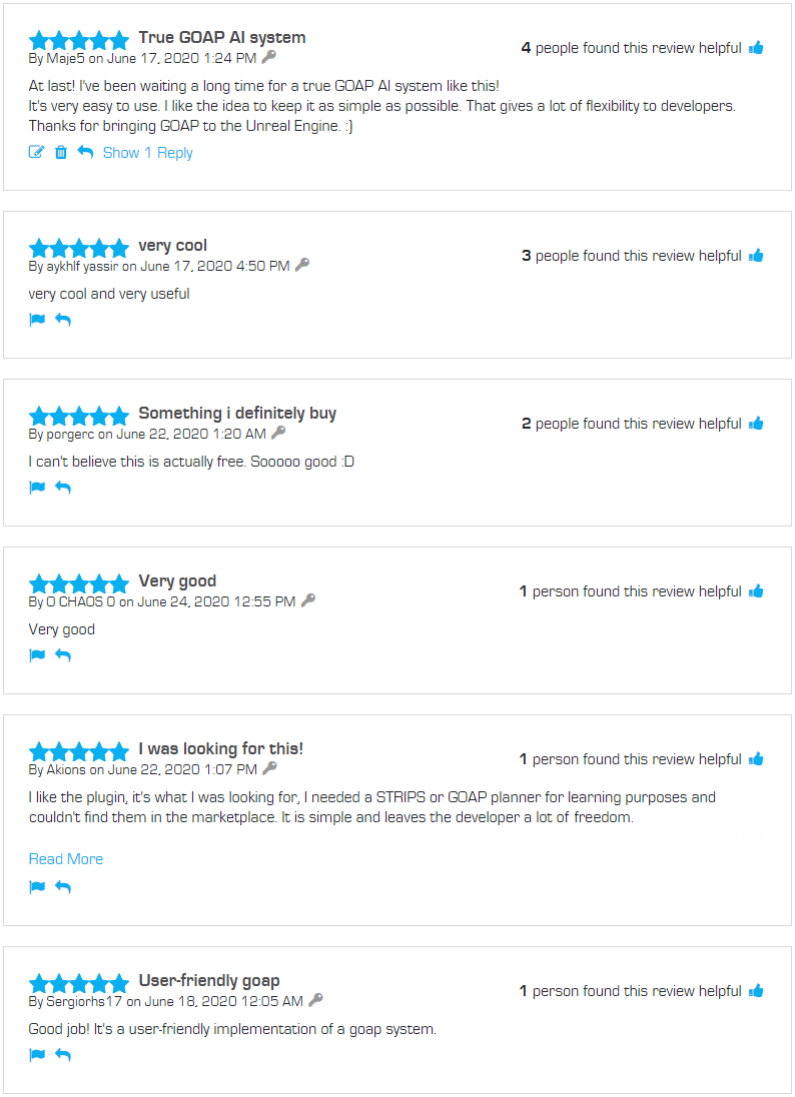

Figura 7.1: Algunas valoraciones recibidas en la página de nuestra herramienta en la tienda de *Unreal Engine*.

Además, gracias a las buenas valoraciones recibidas por los desarrolladores que han utilizado nuestra herramienta, podemos constatar que la aplicación está cumpliendo con su propósito (véase Figura 7.1). Esto hace que

nos sintamos realmente orgullosos por la aportación que hemos realizado al mundo de la Ingeniería del Software orientada al desarrollo de videojuegos, viendo cómo hemos conseguido contribuir a solucionar el problema de la falta de recursos de planificación automática para el comportamiento de PNJs en *Unreal Engine*.

7.2. Trabajo futuro

La evolución constante de la industria de los videojuegos requiere una continua adaptación a las nuevas tecnologías y necesidades que surjan. Este hecho provoca que cualquier herramienta disponible en una tienda de recursos tenga que ser mantenida en el tiempo para que siga siendo útil y eficiente para los desarrolladores.

Esta es una realidad que también se aplica a la herramienta que hemos desarrollado, ya que será necesario continuar manteniéndola en el tiempo de cara a adecuarla a futuras versiones de *Unreal Engine*. Además, el reciente anuncio de *Unreal Engine 5* (Epic Games, 2020) probablemente conllevará la necesidad de realizar grandes cambios para adaptar la herramienta a este nuevo entorno de desarrollo.

Por otro lado, gracias a las valoraciones y sugerencias realizadas por los usuarios sobre nuestra aplicación, hemos obtenido información sobre posibles mejoras que podrían llevarse a cabo.

Entre estas mejoras cabe destacar el desarrollo de otros modelos de algoritmia en la heurística de la planificación. Pese a que la arquitectura GOAP se basa en el uso del algoritmo A*, sí es cierto que podría ampliarse la herramienta para permitir la utilización de otros algoritmos de búsqueda informada, y también explorar también el uso de distintas heurísticas.

Otra ampliación que podría realizarse en un futuro es la integración del *Environment Query System* (EQS) en nuestra herramienta. Este sistema es un complemento de *Unreal Engine* que se encarga de recolectar información del entorno con el objetivo de facilitar la toma de decisiones en situaciones concretas que le plantease el desarrollador.

Actualmente, nuestra herramienta puede utilizarse de manera conjunta con EQS, pero requeriría que el usuario se encargase de implementar la comunicación entre un sistema y otro. Así pues, la ampliación que se podría llevar a cabo consistiría en integrar directamente el uso de EQS en la lógica del planificador.

Bibliografía

Atlassian. Jira. Disponible en https://www.atlassian.com/es/software/jira.

Axon, S. Unity at 10: For better —or worse— game development has never been easier. Disponible en https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/.

B., F. M. El 99 % de los juegos independientes vende menos de 3.000 copias. Disponible en https://www.vidaextra.com/industria/el-99-de-los-juegos-independientes-vende-menos-de-3-000-copias.

Bethesda Game Studios. Fallout 3. Disponible en https://fallout.bethesda.net/games/fallout-3.

Bogost, I. Artificial Intelligence Has Become Meaningless. Disponible en https://www.theatlantic.com/technology/archive/2017/03/what-is-artificial-intelligence/518547/.

Brom, C. Hierarchical reactive planning: Where is its limit? Disponible en https://artemis.ms.mff.cuni.cz/main/papers/Ents-gardener-2005.pdf.

Buttice, C. Finite State Machine: How It Has Affected Your Gaming For Over 40 Years. Disponible en https://www.techopedia.com/finite-state-machine-how-it-has-affected-your-gaming-for-over-40-years/2/33996.

Carryer, S. The Brains in Games: Video Game AI. Disponible en https://towardsdatascience.com/the-brains-in-games-video-game-ai-d0f601ccdf46.

Chotrani, A. What Is GOAP, and Why Is It Not Already Mainstream? Disponible en https://es.slideshare.net/AakashChotrani/what-is-goap-and-why-is-it-not-already-mainstream.

Crystal Dynamics. Tomb Raider. Disponible en https://tombraider.square-enix-games.com/en-us.

Desarrollo Español de Videojuegos. Libro Blanco del Desarrollo Español de Videojuegos. Disponible en http://www.dev.org.es/images/stories/docs/libro%20blanco%20dev%202019.pdf.

Discord Inc. Discord. Disponible en https://discord.com/.

Eidos Montréal. Deux Ex: Human Revolution. Disponible en https://square-enix-games.com/games/deus-ex-human-revolution.

Epic Games. Unreal Engine. Disponible en https://www.unrealengine.com/en-US/.

Epic Games. Epic Store. Disponible en https://www.epicgames.com/store/es-ES/.

Epic Games. A first look at Unreal Engine 5. Disponible en https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5?sessionInvalidated=true.

Federico Peinado, M. C. y. D. P. Revisiting Character-Based Affective Storytelling under a Narrative BDI Framework. Disponible en https://www.fdi.ucm.es/profesor/fpeinado/publications/2008-peinado-revisiting.pdf.

Ferraro, L. ReGoap. Disponible en https://assetstore.unity.com/packages/tools/ai/regoap-77376.

Google. Drive. Disponible en https://www.google.com/intl/es_ALL/drive/.

Hayes, D. R. Gemini Rising AI - Goal-Oriented Action Planning. Disponible en https://drhayes.io/2019/10/25/gemini-rising-ai-goal-oriented-action-planning/.

y Heliodoro Tejedor Navarro, J. D. *Inteligencia artificial*. 2012.

Horti, S. Why F.E.A.R.'s AI is still the best in first-person shooters. Disponible en https://www.rockpapershotgun.com/2017/04/03/why-fears-ai-is-still-the-best-in-first-person-shooters/.

Héctor Gómez-Gauchía, F. P. Automatic Customization of Non-Player Characters using Players Temperament. Disponible en https://www.fdi.ucm.es/profesor/fpeinado/publications/2006-gomez-automatic.pdf.

Informa Tech Holdings. 2019 GDC State of the Game Industry. Disponible en https://reg.gdconf.com/GDC-State-of-Game-Industry-2019.

Iwatani, T. Pac-Man. Disponible en https://www.pacman.com/en/.

Kanban Tool. Kanban Guide. Disponible en https://kanbantool.com/kanban-guide/introduction.

Long, E. Enhanced NPC Behaviour using Goal Oriented Action Planning. Disponible en http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.8964&rep=rep1&type=pdf.

Lucid Software Inc. Lucidchart. Disponible en https://lucidchart.zendesk.com/hc/es-419/articles/207300186-C%C3%B3mo-comenzar-con-Lucidchart.

Malik Ghallab, D. N. y. P. T. *Automated Planning and Acting*. 2016.

y Marcus Toftedahl, H. E. A Taxonomy of Game Engines and the Tools that Drive the Industry. Disponible en https://www.diva-portal.org/smash/get/diva2:1352554/FULLTEXT01.pdf.

Microsoft. Visual Studio. Disponible en https://visualstudio.microsoft.com/es/.

Microsoft. GitHub. Disponible en https://github.com/.

Microsoft. Microsoft Office. Disponible en https://www.office.com/.

Millington, I. *Artificial Intelligence for Games*. tercera edición, 2009.

Monolith Productions. F.E.A.R. Disponible en https://www.lith.com/.

Monolith Productions. Tomb Raider. Disponible en https://www.lith.com/.

Naramura, Y. Peak Video Game? Top Analyst Sees Industry Slumping in 2019. Disponible en https://www.bloomberg.com/news/articles/2019-01-23/peak-video-game-top-analyst-sees-industry-slumping-in-2019.

Narratech Laboratories. Narratech Laboratories'Website. Disponible en http://narratech.com/.

y Nils John Nilsson, R. E. F. STRIPS: A New Approach to the Application of .Theorem Proving to Problem Solving. Disponible en https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf.

Organización Mundial de la Salud. Brote de enfermedad por coronavirus (COVID-19). Disponible en https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019.

Orkin, J. Applying Goal-Oriented Action Planning to Games. Disponible en http://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf.

Orkin, J. Three States and a Plan: The A.I. of F.E.A.R. Disponible en http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.

Pascual, J. A. Así está cambiando los videojuegos la inteligencia artificial. Disponible en https://www.hobbyconsolas.com/listas/cambiando-videojuegos-inteligencia-artificial-375749.

Rasmussen, J. Are Behavior Trees a Thing of the Past? Disponible en https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php.

Slack Technologies. Slack. Disponible en https://slack.com/intl/es-es/.

Statt, N. How Artificial Intelligence will revolutionize the way videogames are developed and played. Disponible en https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning.

Tinny Studios. SGOAP: AI Solution. Disponible en https://assetstore.unity.com/packages/tools/ai/sgoap-ai-solution-167167#reviews.

Toftedahl, M. Which are the most commonly used Game Engines? Disponible en https://www.gamasutra.com/blogs/MarcusToftedahl/20190930/350830/Which_are_the_most_commonly_used_Game_Engines.php.

Unity Technologies. Unity. Disponible en https://unity.com/es.

Valve Corporation. Steam. Disponible en https://store.steampowered.com/?l=spanish.

Valve Corporation. Half Life. Disponible en https://www.half-life.com/es/halflife.

Vassos, S. Introduction to STRIPS Planning and Applications in Video-games. Disponible en http://www.dis.uniroma1.it/~degiacom/didattica/dottorato-stavros-vassos/.

Visualistic Studios. VisAI - FPS - An AI Framework. Disponible en https://www.unrealengine.com/marketplace/en-US/product/visai-an-advanced-modular-ai-system.

WhatsApp Inc. WhatsApp. Disponible en https://www.whatsapp.com/.

WriteLaTeX Limited. Overleaf. Disponible en https://www.overleaf.com/.

Appendix A

Abstract and keywords

A.1. Abstract

Artificial intelligence is one of the fundamental pillars on which the video game industry has settled. For this reason, much of the resources used in the development of a video game are destined to the field related to intelligence simulation and, especially, to improve the behavior of non-player characters.

However, not all studies have the same means of dealing with the costs associated with the development of artificial intelligence. This causes that small companies or independent developers often have to rely on obtaining resources through content stores. The problem arises with the lack of quality resources related to artificial intelligence in these stores.

To solve this problem, we have created a tool for *Unreal Engine* that allows automatic planning, under a GOAP architecture, of the behavior of non-player characters in a simple way. This tool has been developed as a *code plugin*, allowing it to be easily included in any project, and it has been published in the *Unreal Engine*'s marketplace, to make it more accessible to any developer.

A.2. Keywords

Artificial Intelligence, A* Search Algorithm, *Blueprint*, *C++*, Controller, Goal-Oriented Action Planning, Non-Player Character, World State.

Appendix B

Introduction

"Stay hungry, stay foolish."
— Steve Jobs

As students of the Software Engineering Degree, we have always had a great interest in the field of artificial intelligence (AI) and the possible applications that can be made of it in various environments such as video games.

So, we will start by doing a historical review of the video game industry and, especially, focused on the evolution of AI, with the intention to establish the objectives for our end-of-degree project. In addition, we highlight the relation of this project with the subjects taken and we also outline the project structure that we are going to follow.

B.1. Video game industry

Since the beginning of the video game industry in the 1970s, the business of video game development, distribution, promotion and sale has been in continuous growth. The social impact generated by the appearance of video games marked a revolutionary change in the world of entertainment. From its origins to today, this phenomenon has continued to expand and evolve, limited only by the progress of technological evolution (Informa Tech Holdings, 2019).

Over the years, video games have become one of the main leisure options for many consumers. The popularity they have reached have allowed them

to obtain the status of an artistic medium, such as cinema or television, in just a few decades of existence.

This great boom is mainly due to its high adaptability within the market, knowing how to adapt to the needs and tastes of many sectors of society. Constant evolution is the best way to define this industry that has become a great economic engine that generates billions of dollars annually (see Figure B.1).

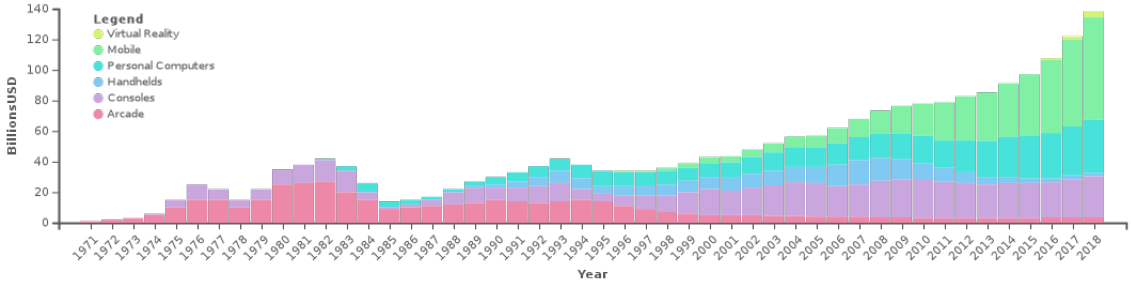

Figure B.1: Video game industry revenue from 1971 to 2018 (Naramura, 2019).

Initially, the video game market was dominated by large video game publishers that offer financial support to developers in the process of creating video games. As it is a market with high growth expectations, the competition to gain a foothold in it has always been very strong and difficult to survive in an environment in which economic capacity is crucial to remain stable (B., 2018).

For many years, large companies absorbed small development companies and thus took control of the market. This fact limited the development of independent video games, who were almost forced to give in to large companies or settle for *shareware* distribution or among friends.

However, the panorama changed radically in the last decade with the appearance of new media that have facilitated the possibility of entering the market and stay in it for all kinds of developers. These new media include the emergence of digital sales platforms such as the *Epic Store* (Epic Games, 2018) or *Steam* (Valve Corporation, 2003), where it is now possible to distribute and sell video games in a global market with very little investment by the independent developer (Desarrollo Español de Videojuegos, 2019). This type of store allows to avoid the obstacle that many development studies face when carrying out large marketing campaigns to publicize their product.

In addition, the release of tools for creating video games such as *Unity*

(Unity Technologies, 2005) or *Unreal Engine* (Epic Games, 1998) have served to facilitate the development process. These tools allow developers to create their product in a unified environment that saves time and resource costs, since many functionalities are already implemented as standard in the engine itself.

With the use of these new resources, independent development studios have carved out a niche for themselves in the market and now they can compete for customers with large companies.

B.2. Artificial intelligence in video games

One of the pillars of the progress of the video game industry is the evolution that has taken place in the field of AI (Pascual, 2019). This area focuses on the research and development of techniques that allow simulating intelligence in the game, particularly on the behavior of non-player characters (NPC). These agents are capable of evaluating situations that are interpreted as variables, and executing actions based on optimization and coherence principles to fulfill a series of determined purposes.

The biggest difference between this type of AI for games versus the traditional one is its strongly practical approach. This is reflected in the fact that AI for NPCs focuses on achieving credible behaviors, that is, that it imitates human behavior but not necessarily that it is infallible solving problems or making rationally perfect decisions.

The need for this specific type of video game AI is what has led video game development companies to dedicate a considerable amount of their resources to improving their tools for creating NPCs.

Large companies have been competing to become the creators of characters with more complex and realistic AI, which has led to the emergence of various alternatives throughout recent history when managing the creation of AI for these agents.

On the other hand, small companies or independent developers have been relegated to the background when it comes to AI for video games. Unable to cope with the high costs of developing their own AI techniques, their projects are forced to turn to outside sources to provide them with appropriate tools and resources. As this is one of the most important pillars of video game development, they are usually scarce resources on the market or of insufficient quality on many occasions. This leads to many of these small companies

choose to simplify the behavior of their NPCs as much as possible to reduce the impact of AI in their video games, also causing a reduction in their realism and credibility.

B.3. Specification of objectives

Our goal for this project is to build a tool that helps define the behavior of NPCs in a more flexible, modular, and user-friendly way. For this reason, we set ourselves a series of objectives with the intention of clearly specifying the purpose of the work.

Our first objective will be to identify a current major shortage in the AI tools for NPCs available to *indie* developers. To do this, we will investigate the most widely used AI planning models, and analyze the current availability of resources in the video game market.

Then we will develop a tool that meets the needs identified above. Thus, we will carry out an appropriate engineering process that includes specification, design, implementation and testing of the tool.

Finally, we set ourselves the objective of correcting this deficiency by launching the tool on the market. In addition, with the feedback we receive from the community, we will continue to maintain it in future iterations.

B.4. Related subjects

Given that one of our objectives is the implementation of a software tool, we will make great use of the acquired knowledge of programming in subjects such as Programming Foundations and Programming Technology.

In Programming Foundations we acquire the bases of logic and reasoning when facing a development of any kind of software. In addition, in this subject we learned to program in *C++* code, which will be extremely useful to us because it is the language used in *Unreal Engine* programming.

On the Programming Technology side, we will put into practice the knowledge acquired about object-oriented programming. This programming model will help us to propose a modular design of the different components of our tool using abstraction, encapsulation, inheritance and polymorphism techniques.

Other subjects closely related to our project are Data Structure and Algorithms, and Algorithmic Techniques in Software Engineering. These two subjects will help us to be able to analyze the efficiency of our algorithms, as well as be able to develop optimal solutions to the problems that arise.

On the other hand, the process of planning and managing the development of the tool will require us to apply the knowledge acquired in the subjects of Software Engineering, Software Modeling, and Software Project Management and Development Methodologies. These subjects represent the fundamental part of our learning related to the analysis and design of the software, as well as the management of the work team.

In relation to the application of specific algorithms for the creation of AI, we will use what we have learned in Knowledge Engineering. It is worth mentioning that this subject was the one that really sparked our interest in carrying out an end-of-degree work related to AI.

Finally, it would be important to mention the rest of the subjects we have taken and we will not apply explicitly (such as Databases, Business Management, Discrete Mathematics and Logic, or Networks), but which have also provided us with general competences that indirectly contribute to this project.

B.5. Project structure

To carry out the objectives previously described, we have chosen to divide the work into two clearly differentiated parts.

On the one hand, we will carry out an investigation to identify the planning models that could be implemented and to compare the strengths and weaknesses of each of them. As part of this research, an analysis of the tools currently available on the market will also be conducted.

Finally, based on the results obtained in the research, we will proceed to carry out the second part of the work, which will consist of the practical development of the implementation of our tool and its launch on the market.

Appendix C

Conclusions and future work

"Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program."
— Linus Torvalds

After completing the project, here we present our reflection on the work carried out. In addition, we will point out possible improvements or extensions to our tool taking into account the possibility of carrying out future work as a continuation of ours.

C.1. Conclusions

Before starting this project, we had in mind that we wanted to make some kind of contribution to the world of video game development by performing the end-of-degree project. In this direction, and with the help of our tutor, we inquired about the possibility of developing an AI tool applied to video games.

In the research we carried out on the evolution of video game AI and the current state of it, we concluded that there were great differences between some planning models and others. Based on these differences, we prepared a comparison of the planners in which we focused on evaluating the strengths and weaknesses of each model.

On the other hand, we also carry out an analysis of the current state of the video game market, placing special emphasis on the availability of AI tools

for video game development by small studios or independent developers.

This double investigation allowed us to conclude that there was an evident lack of AI resources for *indie* video game development. This lack of resources was especially noticeable in the *Unreal Engine*'s marketplace, where there was no AI tool that used a GOAP architecture for automatic planning of NPC behavior.

Thus, we set ourselves the objective of developing a tool that would cover this market need. For this, we first considered that it would be useful to make a prototype in which we tested the *Unreal Engine* functionalities related to AI.

Once this approach to *Unreal Engine* was made, we were able to carry out the design of what was to be our automatic planning tool under a GOAP architecture.

Then we proceeded to implement the tool entirely in *C++* code. In this first approach, the objective was to meet the functional aspect of the application, that is, to ensure that it was actually planning automatically and at runtime.

After finishing developing this tool, we focus on achieving another of the project's objectives. In this case, it was pending to cover the practical aspect of the application. This meant that we had to make the tool easily accessible by any developer and addable to any *Unreal Engine* project.

For this reason, we carry out a second implementation of the tool as a *code plugin*. This type of implementation allowed us to develop the functionalities of the application in *C++* code, and prepare certain classes to be inherited through *Blueprints*. In this way, any user could adapt the GOAP architecture that we have created to the specific needs of their project.

Finally, after having carried out a testing phase with users, we published the tool in the *Unreal Engine*'s marketplace. This fact marked the final milestone of our project, since we had managed to meet all the objectives that we established initially.

As a last detail, we want to mention that this project has been a very rewarding experience for all group members. We have learned a lot about the process involved in developing an application from scratch to publishing it in an official store for use and distribution.

In addition, thanks to the positive evaluations received by the developers who have used our tool, we can verify that the application is fulfilling its purpose (see Figure C.1). This makes us really proud of the contribution

we have made to the world of Software Engineering oriented to video game development, seeing how we have managed to contribute to solving the problem of the lack of automatic planning resources for the behavior of NPCs in Unreal Engine.

C.2. Future work

The constant evolution of the video game industry requires continuous adaptation to new technologies and emerging needs. This fact causes that any tool available in a marketplace has to be maintained over time to remain useful and efficient for developers.

This is a reality that also applies to the tool that we have developed, since it will be necessary to continue maintaining it over time in order to adapt it to future versions of *Unreal Engine*. Furthermore, the recent announcement of *Unreal Engine 5* (Epic Games, 2020) will probably carry the need for major changes to adapt the tool to this new development environment.

On the other hand, thanks to the evaluations and suggestions made by users about our application, we have obtained information about possible improvements that could be carried out.

These improvements include the development of other algorithmic models in planning heuristics. Although the GOAP architecture is based on the use of the A * algorithm, it is true that the tool could be expanded to allow the use of other algorithms.

Another extension that could be made in the future is the integration of the *Environment Query System* (EQS) in our tool. This system is a complement of *Unreal Engine* that is responsible for collecting data from the environment in order to facilitate decision-making in specific situations that arise for the developer.

Currently, our tool can be used in conjunction with EQS, but would require the user to be in charge of implementing communication between one system and another. Thus, the extension that could be carried out would consist of directly integrating the use of EQS in the planner logic.

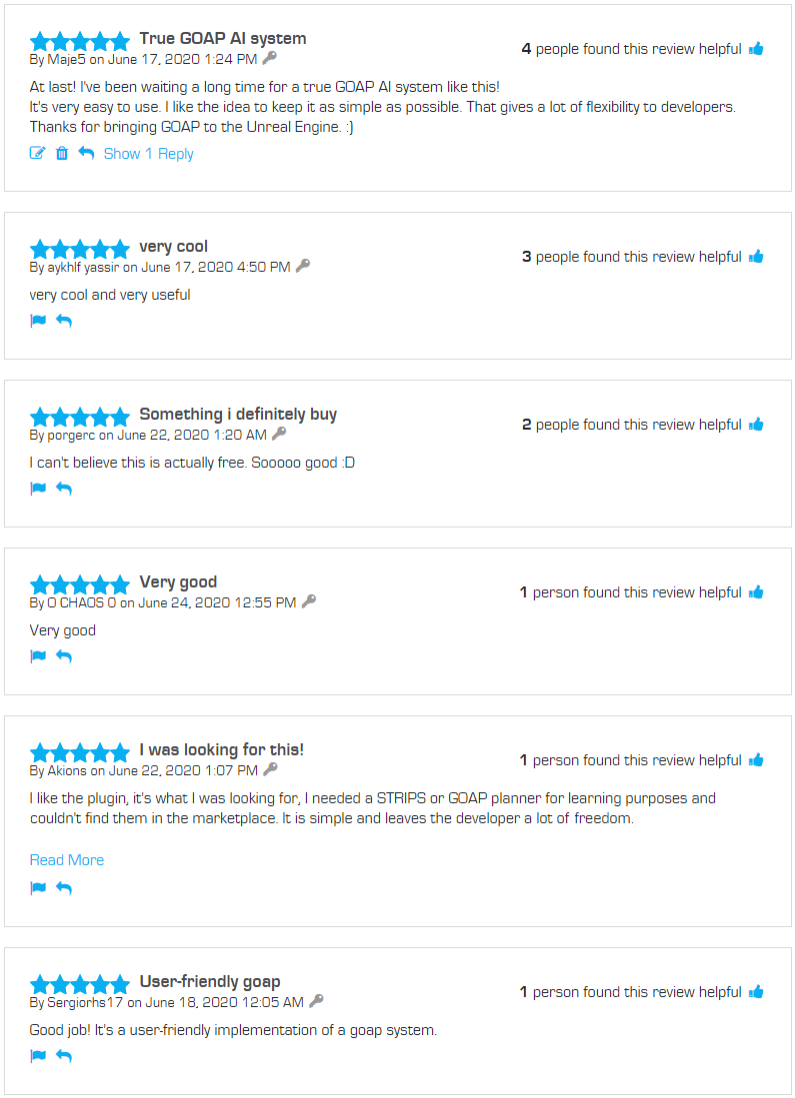

Figure C.1: Some feedback received on the page of our tool in the *Unreal Engine*'s marketplace.

Apéndice D

Aportaciones individuales de cada autor

D.1. Diego Romero

Antes de empezar con el desarrollo propiamente dicho del trabajo de fin de grado, tuvimos que realizar una búsqueda de herramientas que íbamos a necesitar en el proyecto. En la primera reunión, se me asignó el trabajo de investigar sobre el uso de *Overleaf*, ya que era la herramienta que íbamos a utilizar para desarrollar la memoria.

Así pues, me encargué de documentarme sobre la sintaxis en *Overleaf* para, después, explicarles al resto de miembros del grupo como se utilizaba. Además, me encargué de preparar la estructura de las cáscaras de la memoria, así como toda la configuración relacionada con el entorno de *Overleaf*.

Por otro lado, después de que descartásemos el uso de *Perforce*, me encargué de buscar una alternativa para alojar el proyecto. Dicha alternativa la encontré en el uso conjunto de *GitHub* con *GitLFS*. Después de probar su funcionamiento, preparé el repositorio del proyecto para que *GitLFS* se encargase automáticamente de la gestión de los archivos de gran tamaño, y expliqué su funcionamiento al resto del equipo.

En el trabajo de investigación, me encargué de recopilar información sobre los distintos modelos de planificación que eran utilizados en las herramientas de IA disponibles en las tiendas de recursos de *Unity* y *Unreal Engine*. Además, me documenté sobre la evolución historica de la IA en videojuegos, poniendo especial énfasis en el aspecto de la planificación.

Después de terminar de recabar información, puse en común lo que había investigado con el resto de miembros del grupo. De la misma manera, recibí información sobre las investigaciones del resto.

Con toda esta información recabada, debatimos sobre los modelos de planificación empleados en videojuegos y la disponibilidad de los mismos en las tiendas de recursos. Finalmente, llegamos a la conclusión de realizar el desarrollo de la herramienta de planificación automática con una arquitectura GOAP e implementarlo como *code plugin* para *Unreal Engine*.

Después me encargué de organizar los resultados obtenidos en la investigación y realicé la presentación de los mismos en la primera reunión de *Narratech Laboratories*.

A continuación me encargué de realizar la implementación del controlador para el prototipo que desarrollamos en *Unreal Engine*. Este controlador se encargaba de gestionar las acciones de los PNJs y contenía la lógica de ejecución por la cual se realizaba una acción u otra.

Tras finalizar el prototipo, nos centramos en el diseño de los diagramas de clases, flujo y secuencia. De estos me encargué de realizar varios diagramas de secuencia como *generatePlan*.

Una vez terminado el diseño, llevamos a cabo el desarrollo del algoritmo principal sobre el que se basaría la planificación automática. En este caso me encargué de realizar las pruebas sobre dicho algoritmo, así como la corrección de los errores que encontré.

Posteriormente seguí realizando varias tareas de desarrollo como la creación de la clase *Action* y la implementación de varios métodos de la clase *Planner*.

Como había realizado varias tareas seguidas de implementación, las siguientes tareas que me asigné fueron relacionadas con hacer pruebas en las clases *Node* y *WorldState* para que no se acumulasen tareas *En prueba*.

Después fui alternando tareas de desarrollo (como *executeGOAP* del *Controller*) y tareas de pruebas (tratando de coger métodos que no hubiese implementado yo en la medida de lo posible).

Como esta implementación la íbamos a hacer íntegramente en código *C++*, me encargué de desarrollar las acciones de coger la moneda y derribar la puerta. En ambos casos dejé desarrollado las precondiciones, los efectos y el comportamiento que tenia que desarrollar el PNJ al ejecutar las acciones.

Una vez finalizada la implementación funcional de la herramienta, procedí

a realizar la adaptación de la clase *Action* para permitir que se pudiesen crear *Blueprints* heredados de la misma directamente desde *Unreal Engine*. Además, me encargué de desarrollar las acciones de abrir la puerta y seguir al jugador en *Blueprints* para la demo.

Tras terminar la implementación del *code plugin*, me dediqué a preparar la guía de uso de la herramienta y el cuestionario para las pruebas con usuarios reales. Así pues, también me encargué de realizar el análisis de los resultados obtenidos en dicho cuestionario.

Finalmente mencionar que durante todo el transcurso del proyecto, fui documentando en la memoria el trabajo realizado, así como revisé el contenido escrito por el resto de miembros del grupo.

D.2. Mario Sánchez

Desde el inicio hasta el final del proyecto, el reparto de trabajo ha sido equitativo entre los tres integrantes del grupo. En concreto, mi trabajo individual inició tras la primera reunión, mi cometido consistía en investigar herramientas para la gestión de tareas, ya que el proyecto se desarrollaría utilizando metodologías ágiles debido a que trabajaríamos con tecnologías con las que no habíamos tratado antes. Además cada uno de nosotros tenía que buscar información, de forma individual, sobre inteligencia artificial en videojuegos: herramientas existentes, plataformas de desarrollo, metodologías, algoritmos, etc.

Exploré distintas herramientas y plataformas para la gestión de tareas en metodologías ágiles, algunas de estas son *Trello*, *Projects* de *GitHub* o *Microsoft Planner*. Tras probarlas y comprobar las ventajas y desventajas de cada una, compartí la experiencia con mis compañeros y terminamos decantándonos por Jira, ya que nos presentaba ventajas como notificaciones, acceso gratuito, experiencia previa y popularidad

En cuanto al otro tema de investigación, mi búsqueda se centró en torno a GOAP, del que habíamos hablado con el profesor, encontré un método de planificación similar, llamado STRIPS, que a diferencia de GOAP, no tenía la capacidad de tener funciones como precondición de las acciones que lo componían. También investigué algunos juegos que utilizaban GOAP y alternativas a este, como pueden ser los *Behavior Tree* de *Unreal Engine* o las máquinas de estado. En la puesta en común con mis compañeros, presentamos los resultados de nuestra investigación y se decidió el rumbo a seguir, desarrollar

una herramienta para la creación de inteligencia artificial utilizando GOAP en *Unreal Engine*, ya que no existía ninguna.

Puesto que trabajaríamos con *Unreal Engine* y dado que no lo había utilizado nunca, realicé unos tutoriales de *C++* para *Unreal Engine* que se ofrecen en el propio motor. Tras esto desarrollé, por mi cuenta una pequeña prueba en la que un personaje patrullaba por el escenario y si el jugador se ponía en su campo de visión, el personaje comenzaba a seguirle. Esto serviría para comprender cómo estructurar el controlador del personaje en c++ y cómo vincularlo a los *Blueprints*.

Una vez que todos tuvimos claras las características principales de la programación en *Unreal Engine*, decidimos la estructura de clases que seguiríamos, nos repartimos la realización de diagramas, generamos las tareas en Jira.

Antes de empezar a desarrollar el proyecto decidimos generar el algoritmo para comprenderlo y experimentar con él. A partir de un pequeño prototipo en un entorno experimental, creado por José, que consistía en un grafo sobre el que se aplicaba un algoritmo de búsqueda, desarrollé otro algoritmo, en este caso un A*, que es el genérico de GOAP, además lo desvinculé del grafo, de forma que generase el grafo él mismo en base a los nodos y aristas que recibía, de esta forma no construía el grafo completo, únicamente los nodos que exploraba hasta llegar al nodo final. Más tarde, tras las pruebas y algunas mejoras realizadas por mis compañeros, esto se convertiría en el núcleo del proyecto, con varios cambios tras adaptarlo a *Unreal Engine*.

Durante el periodo de desarrollo también me hice cargo de realizar y exponer la presentación en la segunda reunión de *Narratech Laboratories*, que se realizó mediante videoconferencia debido a la crisis del *COVID-19*. En ella presenté la motivación del proyecto, el funcionamiento de GOAP y una pequeña demostración de un posible escenario en *Unreal Engine*, realizado por mis compañeros.

De las tareas del tablero de *Kanban* completé varias de desarrollo y documentación, tanto en *C++* como en *Blueprints*. Por otro lado algunas de las tareas consistieron en realizar test unitarios de algunos de los métodos que habían desarrollado mis compañeros y más tarde la refactorización y prueba.

Para la demo implementé algunas de las acciones, con sus funciones de precondición. Después me hice cargo del sistema para mostrar la acción en curso por el personaje mediante un icono flotante. Más adelante también hice la funcionalidad de mostrar los predicados de los estados del mundo actual y deseado, cuando se presionase una tecla.

Una vez completado el código y la demo, puesto que queríamos publicar la herramienta en la tienda de *Unreal Engine*, era necesario abstraerla del proyecto, cosa que me resultó sencilla ya que la desarrollamos de forma que no tuviese dependencia con el proyecto, pero aun así fue necesario recrearlo en forma de *code plugin*. También me encargué de parte de la documentación del *code plugin* para la tienda.

Una vez completadas las pruebas del código y subirse a la tienda, realicé una optimización del código, quitando algunas librerías *STD* que habíamos utilizado por otras propias de *Unreal Engine*. Y una vez obtenido el *feedback* tanto de la tienda como del formulario que realizaron mis compañeros, traté los datos para analizar.

En cuanto a mi trabajo en la memoria, fui completando los capítulos que me tocaron a medida que avanzábamos con el proyecto, también he repasado el resto de la memoria, ayudando con la revisión y corrección.

D.3. José Manuel Sierra

Tras la primera reunión se decidió investigar sobre las tecnologías que se iban a usar durante el desarrollo del proyecto. Aparte de investigar sobre todas ellas y conocer su funcionamiento; me encargué de crear un servidor *Perforce* para alojar el código. Este programa más adelante lo reemplazamos por *GitHub*.

Tras esto iniciamos un trabajo de investigación donde, de manera individual, cada integrante del equipo recopilaba información sobre GOAP para, pasado un tiempo, realizar una puesta en común y sacar conclusiones. En mi caso investigué las tiendas de *Unreal* y *Unity* buscando contenido que tuviera alguna relación con los planificadores GOAP. Aparte de esto leí varias páginas web y lecturas recomendadas por el tutor relacionadas con el tema que explicaban el funcionamiento de GOAP sacando conclusiones de los pilares fundamentales de este.

En la puesta en común concretamos donde y como se iba a codificar decidimos crear un ejemplo en *Unreal Engine* para familiarizarnos con esta plataforma. En este ejemplo, que no usaba GOAP, me encargué de crear algunas acciones del PNJ como recoger objetos, llaves, monedas y abrir puertas si se había recogido una llave previamente.

Tras crear este ejemplo empezamos a desarrollar los distintos diagramas que nos iban a servir como guías a la hora de implementar el código. Realicé

algunos diagramas de secuencia y los diagramas de clases; aparte de esto intentábamos, que una persona revisara los diagramas creados por otro. Revisé algunos diagramas de secuencia hechos por mis compañeros.

Seguido a esto empezamos a crear el algoritmo principal de GOAP. Este se encargaba de buscar la mejor secuencia de acciones en un estado del mundo; realizaos varias versiones. Cada integrante del equipo creo una versión de este algoritmo en mi caso creé un algoritmo que usaba grafos con caminos valorados y una búsqueda en altura para encontrar la mejor solución.

Una vez creado el algoritmo principal decidimos empezar a crear las demás clases necesarias para el correcto funcionamiento del proyecto. Nos las repartimos por métodos intentando que todos los miembros del grupo realizáramos funciones de todas las clases. En mi caso realicé algunos métodos de la clase *Controller*, creé la clase *Planner* con sus atributos, *getters*, *setters* y constructores aparte de crear algún método de esta y distintos métodos de la clase *WorldState*.

Mientras codificábamos íbamos probando los métodos ya acabados, siempre intentado no probar métodos creados por nosotros mismos. Realice distintas pruebas a algunos métodos de la clase *Planner* y distintos métodos de la clase *Worldstate* además de comprobar la correcta creación de las clases *Node* y *Controller*. Cuando todos los métodos de una clase estaban codificados empezábamos con las pruebas de sistema donde las repartimos equitativamente entre los miembros del grupo.

Con esto teníamos creado el código principal pero nos faltaba crear las acciones que podía realizar el PNJ. Empezamos creando primero estas acciones en código c++ en este caso me toco realizar la acción de *abrirPuerta* entre otras. Seguido a esto creamos las mismas acciones pero en *Blueprints*; me toco realizar acciones como coger llave y coger moneda.

Una vez creadas todas las acciones decidimos empezar a crear el *code plugin* para publicarlo en la tienda. En esta parte me encargue de crear los distintos vídeos que mostraban cómo funcionaba el *code plugin* y un ejemplo de este. Esto lo realice con varios editores de vídeo así como programas de grabación de pantalla. Estos vídeos fueron publicados en *YouTube* en el canal de *Narratech laboratories* y referenciados en la documentación.

Una vez publicado el *code plugin*, creados los formularios y las distintas guías, esperamos unos días para obtener respuestas de los usuarios que estaban probando el *code plugin*; tras estos días me encargué de analizar las respuestas que habíamos recibido en los comentarios de la tienda así como los comentarios del formulario. Con estos comentarios se mejoro el *code plugin*,

me encargué de refactorizar la clase *Planner* y añadir las mejoras necesarias a esta clase mencionadas por los usuarios.

Durante el desarrollo del proyecto se ha ido rellenando la memoria; la cual en las primeras reuniones nos distribuimos. De esta realicé algunas partes de distintos capítulos a parte de crear de manera común entre todos los miembros el capítulo 1 y 7. Una vez finalizada la memoria decidimos revisarla y nos dividimos está en capítulos para no dejar ninguno sin revisar.

Apéndice E

Actas de las reuniones

E.1. Con el tutor

A continuación se detalla el contenido de los temas tratados en las reuniones que hemos ido teniendo a lo largo del curso con nuestro tutor, y nos permite poner de manifiesto cómo se ha ido produciendo la evolución de nuestro trabajo de fin de grado.

E.1.1. 10 de octubre de 2019

En la primera reunión que tuvimos con el tutor, establecimos cómo iba a ser el proceso a realizar para el trabajo de fin de grado. Se decidió que haríamos primero un trabajo de investigación sobre inteligencia artificial para videojuegos, para después desarrollar una herramienta que fuese útil dentro de dicho marco.

Además, hablamos sobre las distintas tecnologías que íbamos a utilizar durante el desarrollo del proyecto. Decidimos utilizar *Perforce* para alojar el proyecto, *Overleaf* para crear y dar formato a la documentación, y *Jira* para el control y gestión del trabajo. Para las siguientes semanas, nos marcamos como objetivo familiarizarnos con todas estas tecnologías y hacer pruebas en ellas.

E.1.2. 31 de octubre de 2019

En esta reunión hablamos sobre los problemas que habíamos tenido con *Perforce* y, tras debatir otras opciones, nos decantamos por usar *GitHub* en combinación con *GitLFS* como servicios de control de versiones y gestión de almacenamiento.

Por otro lado, marcamos las pautas sobre el proceso de investigación que íbamos a llevar a cabo sobre el estado actual de la inteligencia artificial en videojuegos, poniendo especial interés en los diferentes modelos de planificación disponibles y la disponibilidad de recursos relacionados con ellos.

E.1.3. 28 de noviembre de 2019

Este día dejamos establecida cuál sería la estructura del contenido de la memoria y empezamos documentar las conclusiones obtenidas de los trabajos de investigación. En base a los resultados de la investigación, prepararíamos una presentación para exponerla en la reunión general con otros miembros de *Narratech Laboratories*.

Otro de los asuntos que tratamos en esta reunión fue la realización de un prototipo en el que desarrollásemos un ejemplo sencillo de planificación de inteligencia artificial en *Unreal Engine*. El objetivo de este prototipo seria ayudarnos a comprender mejor el uso de dicho motor gráfico de cara a la puesta en marcha de la implementación de nuestra herramienta.

E.1.4. 13 de enero de 2020

Tras presentar el prototipo al tutor, empezamos a valorar cómo afrontar el desarrollo de nuestra herramienta. Así llegamos a la conclusión de que sería razonable realizar primero una aproximación totalmente implementada en código *C++*.

En esta primera implementación nos centraríamos en la parte funcional de la herramienta, es decir, en conseguir desarrollar un planificador que realmente aplicase una arquitectura GOAP en un entorno dinámico.

E.1.5. 5 de marzo de 2020

Este día presentamos al tutor el primer modelo de nuestra herramienta desarrollado íntegramente en *C++*. La herramienta ya permitía la resolución de problemas de inteligencia artificial utilizando una arquitectura GOAP. Sin embargo, aún faltaban por añadir algunas mejoras como la replanificación en tiempo de ejecución, y preparar una demo que permitiese probar fácilmente la herramienta.

E.1.6. 26 de marzo de 2020

Una vez decretada la cuarentena por la pandemia de la *Covid-19*, nos vimos obligados a llevar a cabo las reuniones a través de videollamada, y no presencialmente como se habían llevado a cabo hasta este momento.

En esta reunión, expusimos las correcciones que habíamos llevado a cabo sobre nuestra herramienta y presentamos una demo en la que se realizaba un ejemplo de uso. Esto significó un gran hito en nuestro de desarrollo, ya que finalmente habíamos conseguido desarrollar una herramienta totalmente funcional para la creación de inteligencia artificial con GOAP en *Unreal Engine*.

Sin embargo, sabíamos que el trabajo aún no estaba del todo terminado porque faltaba por conseguir que la herramienta fuese lo más accesible posible para que cualquier desarrollador pudiese utilizarla en sus proyectos. Por este motivo, decidimos llevar a cabo una segunda implementación que permitiese integrar la herramienta como *code plugin* en *Unreal Engine*.

E.1.7. 23 de abril de 2020

Para este día ya teníamos desarrollada la segunda implementación de la herramienta como *code plugin* para *Unreal Engine*. Además, se reprodujo la demo creada en la primera implementación, pero esta vez utilizando las clases preparadas para ser heredadas mediante *Blueprints*.

En este momento, nos marcamos como objetivo conseguir publicar el *code plugin* en la tienda de contenidos de *Unreal Engine*. Para ello, empezamos a estudiar qué requisitos teníamos que cumplir para nos permitiesen la publicación de la herramienta y, por otro lado, propusimos la posibilidad de llevar a cabo un cuestionario que nos permitiese obtener feedback sobre posibles mejoras de nuestra herramienta previo a la publicación de manera oficial.

E.1.8. 26 de mayo de 2020

En esta reunión terminamos de definir el cuestionario que íbamos a realizar para obtener feedback sobre nuestra herramienta y lo pusimos en marcha. Además, tras haber evaluado los requisitos de la tienda de *Unreal Engine* y haber realizado los ajustes necesarios, llegamos a la conclusión que era conveniente inicial el proceso de publicación lo antes posible porque el tiempo de evaluación de nuevos recursos solía ser largo.

E.1.9. 3 de junio de 2020

El motivo de realizar esta reunión fue para poner de manifiesto las causas por las que *Epic Games* había rechazado nuestra herramienta. Afortunadamente, se trataban principalmente de defectos de forma que pudimos solventar rápidamente. Una vez realizados los cambios, volvimos a solicitar la publicación del *code plugin*.

E.1.10. 15 de junio de 2020

Finalmente, *Epic Games* aceptó nuestro *code plugin* y lo publicamos en la tienda de contenidos. Además, aprovechamos los resultados que ya estábamos obteniendo del cuestionario para añadir unas últimas mejoras a la herramienta.

En esta reunión presentamos el borrador de la memoria al tutor, siendo esta valorada positivamente para su presentación en la convocatoria de Junio. De cara a la entrega definitiva, el tutor nos propuso una serie de mejoras que debíamos llevar a cabo sobre la memoria.

E.2. *Narratech Laboratories*

Como miembros del grupo *Narratech Laboratories*, hemos participado en el desarrollo de varias reuniones con otros miembros que están actualmente desarrollando trabajos de fin de grado, trabajos de fin de máster o tesis doctorales. En estas reuniones se ponen en común los avances que ha llevado a cabo cada grupo, con el objetivo de compartir conocimientos y colaborar unos con otros.

E.2.1. 19 de diciembre de 2019

Esta fue la primera de las reuniones del grupo *Narratech Laboratories*, en la cual realizamos una presentación sobre el avance del trabajo de fin de grado. Expusimos la idea de nuestro proyecto, así como los resultados de la labor de investigación que habíamos llevado a cabo sobre inteligencia artificial aplicada en videojuegos. Además, presenciamos cómo otros grupos exponían sus respectivos trabajos, y recibimos feedback sobre la herramienta que queríamos desarrollar.

E.2.2. 2 de abril de 2020

Debido a las circunstancias excepcionales provocadas por la pandemia del *Covid-19*, esta reunión se realizo de manera telemática a través de videoconferencia. Aquí realizamos una presentación en la que mostramos nuestra primera demo de la implementación en código *C++* de una arquitectura GOAP en *Unreal Engine*. Al igual que en la anterior reunión, otros grupos también expusieron los avances de sus respectivos proyectos, y recibimos feedback de cara a la implementación de la herramienta como *code plugin*.

E.2.3. 1 de julio de 2020

En esta reunión se llevará a cabo una presentación final de cuáles han sido las conclusiones obtenidas en el proyecto, así como se mostrará el *code plugin* en funcionamiento. El objetivo de esta reunión es que sirva como prueba para saber qué mejorar de cara a la defensa final de trabajo.