Desarrollo de un Sistema de Control de Bots para Unreal Engine basado en la Teoría de la Utilidad

Alfredo Hernández Burgos

FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE
E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Grado en Ingeniería Informática

Madrid, 14 de septiembre de 2018

Director: Prof. Dr. Federico Peinado Gil
Codirector: Maximiliano Miranda Esteban

Agradecimientos

En primer lugar agradecer a Federico Peinado Gil y a Maximiliano Miranda Esteban, director y co-director de este proyecto por haber confiado en mi y por aunar esfuerzos cuando parecía que la cosa se ponía difícil, por esos fines de semana revisando e indicando por que camino seguir.

A mis padres por aguantarme en esos momentos de estrés y por darme fuerzas cuando parecía que no había manera de seguir adelante, tanto en el proyecto como en la carrera en general. Por esas noches en las que se asomaban por el pasillo para ver si ya me había acostado.

A mi hermana por ser una dura meta por alcanzar, por decirme las cosas claras cuando era necesario y por aliviar ese peso que yo mismo me ponía sobre mis hombros.

A mis amigos que aunque parecía que los tenia olvidados y que no los veía yo sabia que estaban ahí y que me apoyaban como podían.

A Alex con quien empece este proyecto pero que por circunstancias de la vida no pudimos acabarlo juntos.

A mis compañeros de trabajo a los que aburría contándoles lo que estaba haciendo y con los que me ponía pesado buscando un hueco para que probasen el proyecto y me rellenasen los formularios.

GRACIAS

Índice general

Índice general 1

Resumen 7

1. Introducción 8

2. Estado de la técnica 13

2.1. Entornos de desarrollo . . . 15

2.1.1. Unreal Engine . . . 16

2.1.2. Unity . . . 20

2.1.3. Herramientas basadas en Utilidad . . . 21

2.2. Teoría de Utilidad . . . 25

2.3. Accesibilidad . . . 29

3. Objetivos y especificación 31

3.1. Objetivos . . . 32

3.2. Especificación de requisitos software . . . 33

3.2.1. Nodos de Utilidad . . . 33

3.2.2. Controlador . . . 33

3.2.3. Librerías . . . 34

3.2.4. Mapas de Pruebas . . . 34

3.2.5. Tutoriales . . . 35

4. Metodología y herramientas 37

4.1. Metodología . . . 37

4.2. Herramientas . . . 41

4.2.1. Desarrollo 41
4.2.2. Comunicación 43
4.2.3. Alojamiento compartido y control de versiones 44
4.2.4. Redacción de la memoria 45

5. Análisis, diseño e implementación **47**
5.1. Clases y Estructuras 47
5.1.1. *UtilityNode* 48
5.1.2. *UtilityNodeComplex* 50
5.1.3. *MasterUtilityController* 51
5.1.4. *UtilityActions* 54
5.1.5. Curvas de Utilidad 55
5.2. Creación de Bots 56
5.2.1. Métodos Alternativos 64

6. Pruebas y resultados **66**
6.1. Pruebas internas 66
6.1.1. MapUtilityComplex 67
6.1.2. MapUtilitySimple 70
6.1.3. Utility VS Tree 71
6.2. Pruebas con usuarios 72
6.2.1. Análisis 73

7. Conclusiones **79**
7.1. Trabajo Futuro 82

Referencias **84**

A. Introduction **85**

B. Conclusions **90**

B.1. Future Work 93

C. Instructions for use **95**

Resumen

Este proyecto está orientado al campo de la Inteligencia Artificial aplicada en el ámbito de los videojuegos. De forma mas específica, ha consistido en el desarrollo de un sistema que nos permite crear inteligencias artificiales capaces de hacer uso de la Teoría de la Utilidad en un entorno de desarrollo de videojuegos como Unreal Engine, motor elegido para el desarrollo. Otro aspecto importante a la hora de crear este sistema era hacerlo accesible a personas que pudiesen contar con pocos conocimientos de programación, por lo que se han intentado sistematizar mucho los pasos requeridos para su utilización.

Para poder llevar a cabo este proyecto se investigaron herramientas de varios motores de videojuegos que hicieran lo mismo que tratábamos de hacer nosotros, de esta forma nos dimos cuenta de la escasez de dichas herramientas y de la necesidad de su creación.

Se crearon varios prototipos en los que usábamos la utilidad de formas algo artificiales y forzadas hasta ser capaces de conseguir que funcionase, y cuando lo conseguimos lo fuimos refinando hasta llegar al resultado final. También nos vimos en la necesidad de crear varios mapas de juego para mostrar su funcionamiento y poder comparar entre varias inteligencias artificiales creadas. Finalmente incluimos un tutorial, que veíamos una necesidad, dado que queríamos poder ofrecérselo a desarrolladores con poca experiencia.

Palabras clave

Lógica Probabilista,Inteligencia Artificial, Bots, Videojuegos, Utilidad, NPC, Árboles de Comportamiento, Controladores, Curvas de Utilidad

Title

Development of a Control System for Unreal Engine Bots based on Utility Theory

Abstract

This project is oriented to the field of Artificial Intelligence applied to the industry of video games. More specifically, it consist in the development of a system that allows us to create bots that can use the Utility Theory in a video game engine such as Unreal Engine, the chosen environment for development. Another important aspect when creating this system is that it is easier for non-experienced programmers so it is a much simpler system with less necessary steps for its use.

In order to carry out this project, tools of various game engines were investigated that did or tried to do the same thing that we did, in this way we realized the lack of tools and the need for their creation.

Several prototypes were created in which Utility Theory was used in different ways until it was possible to achieve it and when we succeeded in refining the tool, we reach the final result. We also have the need to create several game maps to show how they work and compare several artificial intelligences created. and finally we did a tutorial that could help developers with less experience.

Keywords

Probabilistic Logic, Artificial Intelligence, Bots, Video Games, Utility, NPC, Behavior Tree, Controller, Utility Curves

Capítulo 1

Introducción

Los videojuegos se han convertido hoy día en algo que va más allá de ser simples aplicaciones informáticas destinadas al entretenimiento. Aunque darles una definición precisa es complejo, entre el software que se cita tradicionalmente al hablar de *primeros videojuegos de la historia* (Goldberg, 2011; Kent, 2001) encontramos el simulador de misiles que Goldsmith y Mann programaron en 1947, el clásico juego de NIM que creó Ferranti en 1951, y las tres en raya que aparecían en la tesis de Douglas en 1952.

A pesar de la incertidumbre de saber con exactitud cual es el primer videojuego, si sabemos que la aparición de la primera idea de videojuego corresponde al año 1951 y fue introducida por Ralph H.Baer cuando le sugirió la idea de hacer un juego interactivo para una televisión tras un encargo (Kent, 2001). Después, en los años 70, podemos hablar de la primera revolución de los videojuegos en la sociedad americana con la aparición de Magnavox Odyssey, cuyo creador principal fue Ralph H.Baer, y del juego Pong de la mano de Atari, considerados por muchos como las primeras producciones de videojuegos comerciales pues ya no eran proyectos experimentales. La idea de videojuego ha ido cambiando con el paso de los años, adaptándose a las necesidades de cada tiempo y llegando a lo que es hoy en día: el mayor producto de entretenimiento consumido en todo el mundo (Desjardins, 2017).

La creciente demanda de videojuegos, y sobre todo, la exigencia de los usuarios, ha llevado a la industria de los videojuegos a tener que mejorar constantemente los aspectos principales de estos: jugabilidad, las características visuales, la historia, la inteligencia artificial, entre otros muchos aspectos; lo que ha llevado a que sea una industria que hoy en día es capaz de facturar cifras que rondan los 100000 millones de dólares (en 2017) (*Día Internacional del Gamer, una industria con 15,8 millones de jugadores habituales en España*, 2018).

Podemos distinguir múltiples aspectos que son necesarios establecer a la hora de plantearse crear un videojuego desde cero, como por ejemplo el diseño, las mecánicas, contexto, gráficos, publico objetivo, etc., algunos de estos aspectos han visto variado su nivel de importancia a lo largo del tiempo.

En la época actual, por ejemplo, se ha convertido en un aspecto muy importante que el juego este orientado a ser multijugador, (Tassi, 2014) llegando al punto de que en muchas producciones de videojuegos se ha tendido a reducir el tiempo dedicado al desarrollo de los modos conocidos como de campaña individual, o *singleplayer* (respecto a juegos anteriores de las mismas sagas) para darle más importancia a los modos multijugador (o *multiplayer*). Un claro ejemplo de este caso los encontramos en el género de los *shooter* (o juegos de acción con disparos), o los juegos de deportes; géneros que hoy en día no concebimos sin un modo multijugador.

Este hecho a provocado que la creación de inteligencias artificiales en los juegos sea un aspecto que cada vez ha ido ganando más importancia. Esto es debido a que los jugadores habituales, una vez acostumbrados a jugar con y contra otros jugadores humanos, sienten más complicada su inmersión en el universo del juego si pasan a enfrentarse de nuevo a jugadores automáticos, con reacciones no humanas o ilógicas.

También ocurre que se utilizan agentes artificiales para controlar a personajes no humanos, por lo que una reacción humana en un personaje de este tipo también seria ilógica. El objetivo, por lo tanto, es poder crear agentes con inteligencias artificiales que guarden una relación satisfactoria con el contexto del juego y sean capaces de reaccionar al entorno de la forma que el desarrollador desee.

Por estas razones la inteligencia artificial en los videojuegos es un aspecto cada vez más importante que tiene que seguir evolucionando e implementando las distintas teorías que surgen en este ámbito de la informática como por ejemplo la teoría de decisión, la teoría de la utilidad, los arboles de comportamiento, las redes bayesianas ...

Para ser capaces de trabajar en la inclusión de estas teorías y de otros aspectos en los videojuegos necesitamos tener herramientas de desarrollo que se integren y trabajen adecuadamente con motores de juego. Estos motores modernos como Unreal Engine o Unity, han venido casi a monopolizar el desarrollo de videojuegos en los últimos 10 años, pero mucho antes de su aparición, los motores de juego estaban escritos en una sola entidad con unas metodologías determinadas. Por ejemplo un juego de Atari 2600 debía estar diseñado con una metodología *bottom up* para sacar el máximo rendimiento del sistema.

El rápido avance de las recreativas hizo replantearse este método de desarrollo pasando así a utilizar una base de reglas *hard-coded* con un pequeño numero de niveles y datos gráficos. Surgieron así los primeros motores de juego en 2D desarrollados y usados por las compañías.

El auge del 3D en 1990 con juegos como Doom y Quake ya planteaban la división entre "game content" (reglas especificas del juego y datos) y "game assets" (detección de colisiones y entidad del juego). En 1998 y con la idea de separación entre motor y contenido surgió Unreal Engine [1] , el uso de estos motores ha ayudado de manera significativa tanto a pequeños desarrolladores como a empresas globalizadas en el sector ayudando así al desarrollo de videojuegos de una manera eficiente y rápida.

[1]importante decir que Unreal Engine se liberó hace unos años, haciendo de esta forma que sea un motor gráfico que facilita su utilización tanto en educación como en otros campos debido a su fácil acceso

Como explicábamos anteriormente el reciente crecimiento de juegos multijugador en la industria ha hecho que la importancia que tenia la inteligencia artificial en los juegos se vea incrementada, pues cuando jugamos contra el propio entorno del juego necesitamos reacciones más humanas de tal forma que no nos parezcan raros los comportamientos de lo *NPC's*, haciendo que no se pierda la experiencia inmersiva de los juegos. Esta creciente demanda de una experiencia de juego inmersiva e interacciones inteligentes con *NPC's* centra el foco en las técnicas usadas por los desarrolladores para definir la Inteligencia artificial. Un método muy extendido dentro del desarrollo de videojuego son los árboles de comportamiento (*Behavior trees*),un modelo matemático (jerarquizado) de ejecución que define los diferentes caminos finitos que pueden ejecutarse si se cumple cierto criterio o de manera secuencial dependiendo del caso.La demanda de agentes inteligentes más complejos hace que esta técnica no sea tan efectiva, en este contexto surge la teoría de la utilidad como una extrapolación de la teoría usada en el campo de la economía. La idea principal de este nuevo paradigma es que toda posible acción o estado dado por un modelo se puede describir con un parámetro denominado "utilidad" que describe lo importante que es esa acción en un determinado contexto y para un determinado personaje. Con esta idea se pretenden abordar alguno de problemas y limitaciones existentes en los árboles de comportamiento y que pueden ser solucionados de esta forma.

Existen diversas herramientas para el desarrollo de inteligencias artificiales en motores de juegos para dotar a agentes de diversos comportamientos, sin embargo, la mayoría de estas herramientas son de pago, y la curva de dificultad para aprender a utilizarlas satisfactoriamente suele ser muy pronunciada. Resultando así muy compleja la valoración de su utilización por parte de desarrolladores en producciones de un tamaño discreto o incluso mediano, ya que se debe evaluar si realmente merece la pena acometer un desembolso económico en herramientas costosas de dominar.

Esto es un problema, tanto para los desarrolladores pequeños como para las grandes empresas que han de dedicar tiempo y esfuerzo en la creación de herramientas que, por otra parte, debido al tiempo y esfuerzo empleados en su desarrollo, no suelen liberar más tarde.

Unreal Engine [2] es el motor al que pensamos destinar nuestra herramienta. Cuenta de base con una herramienta de creación de árboles de comportamiento para implementar inteligencias artificiales, pero carece de otras herramientas para el desarrollo de agentes más complejas o para mejorar dichas inteligencias artificiales, haciendo que estas puedan reaccionar a estímulos o aprendan. Conseguir que puedan tener estas capacidades más avanzadas no es posible utilizando únicamente arboles de comportamiento, este es un problema que a la larga mermará la capacidad de este motor para resolver situaciones complejas. Por ello es importante empezar a desarrollar software que unifique estas ideas respecto a las inteligencias artificiales en *UtilityNetwork* motores de juegos como Unreal Engine como nuestro sistema de control de bots, *UtilityNetwork*.

[2] Unreal Engine no cuenta de base con ninguna herramienta que utilice la Teoría de la Utilidad de forma independiente para crear inteligencias artificiales o mejorarlas, por otro lado desarrolladores independientes si han creado plugins que mejoran o añaden ciertos aspectos a la creación de inteligencias artificiales

Capítulo 2

Estado de la técnica

¿Qué es la inteligencia artificial? Para plantearnos esta pregunta primero tenemos que preguntarnos qué es la inteligencia; la inteligencia es definida como la facultad de la mente que nos permite aprender, entender, razonar, tomar decisiones y formarnos una idea de realidad, (OxfordDictionaries, s.f.)de tal forma que con esta definición como base podríamos definir la inteligencia artificial como un programa de computación capaz de simular total o parcialmente los aspectos anteriormente mencionados. Teniendo esto en cuenta podremos ver si algunas de las apariciones mecánicas y programas a los largo de la historia pueden ser realmente catalogadas como inteligencias artificiales o no.

El termino inteligencia artificial fue acuñado por John McCarthy en 1956 durante la conferencia de Darthmounth y fue definido como: "...la ciencia e ingenio de hacer maquinas inteligentes, especialmente programas de cómputo inteligentes"(McCarthy, 1956), aunque el termino fue acuñado en ese momento la idea llevaba formándose desde años antes, desde que Alan Turing en 1950 es el primero que habla de la idea, proponiendo una respuesta para considerar a una maquina inteligente en uno de sus artículos (Turing, 1950), dando lugar al famoso *Test de Turing.* Aun así las bases sobre la inteligencia artificial se pueden considerar aun más antiguas, datando incluso de la época de la antigua Grecia, por supuesto en estos tiempos se aplicaba a accesorios mecánicos y no llegaba a las implicaciones que tiene actualmente la inteligencia artificial.

Según Turing si una máquina es capaz de emular un comportamiento humano podremos decir que esta dotada de inteligencia, por lo tanto lo que propone en el *Test de Turing*, es una interacción entre un juez y un ente que puede ser una máquina o un humano ya que el juez no lo sabe, al final de la interacción que consiste en una serie de preguntas por parte del juez, es este mismo el que determina la naturaleza del ente con el que ha estado interactuando, si una máquina es capaz de hacer creer al juez que es un humano se considera que ha pasado el test.

Aunque la definición de inteligencia artificial ha ido evolucionando desde su primera aparición, el objetivo siempre será intentar imitar el comportamiento humano tal y como se define en el *Test de Turing* y con esto en mente la inteligencia artificial a sido capaz de expandirse para poder ser utilizada en distintos campos.

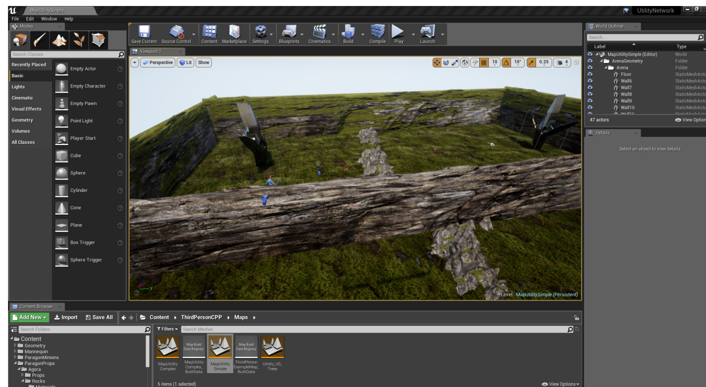
En la actualidad la inteligencia artificial se utiliza en múltiples campos como la medicina, la robótica industrial, la atención al cliente ... y por supuesto en el mundo de los videojuegos que es el campo que nos interesa tratar en este proyecto. Es, precisamente, en este campo donde más se busca que los comportamientos de las inteligencias artificiales simulen un comportamiento humano, pues de los NPCs suele esperarse un comportamiento humano en muchos escenarios distintos, aunque no exclusivamente, dado que también nos puede interesar simular comportamientos animales.

A la hora de intentar crear inteligencias artificiales y viendo lo limitadas que están las herramientas que tienen esto como objetivo en motores como Unreal Engine, hemos decido crear un sistema que nos facilite la creación de estas inteligencias artificiales y que ademas nos aporta una opción mas allá de los árboles de comportamiento que facilita Unreal Engine. En nuestro caso lo que hemos decidido utilizar como base para las inteligencias artificiales es la Teoría de la Utilidad (que explicaremos mas adelante), usando esta teoría esperamos poder crear inteligencias artificiales que parezcan tomar realmente las elecciones más útiles para el personaje en cada momento, de forma que se acerque aun más al comportamiento humano que tendremos como objetivo final y también intentaremos que el proceso de crear dichas inteligencias artificiales sea lo mas cómodo y sencillo posible evitando un periodo de aprendizaje demasiado largo, tanto para personas con conocimiento previo del motor como para aquellos que se están iniciando en el.

2.1. Entornos de desarrollo

En cuanto al entorno de desarrollo, o motor de videojuegos, aunque decidimos usar Unreal Engine también valoramos Unity como opción, dado que son dos de los motores gratuitos mas conocidos. De forma que investigamos con qué herramientas contaban cada uno a la hora de crear inteligencias artificiales. A continuación explicaremos brevemente sus características.

2.1.1. Unreal Engine



En lo que respecta a Unreal Engine 4 las herramientas básicas de las que dispone para poder crear y trabajar con las inteligencias artificiales son:

- Behavior Tree: Son los llamados árboles de comportamiento, en ellos se especifica lo que queremos que un personaje haga de forma que cada hoja del árbol es una acción especifica a ser realizada. Los nodos intermedios pueden ser *sequence* y *selector*, ambos se ejecutan de izquierda a derecha, pero el selector ejecuta sus hijos hasta que uno de ellos consigue ejecutarse con éxito dando el *selector* como exitoso y fallando en caso de que fallen todos sus hijos, por otro lado en *sequence* se ejecutan sin parar hasta que uno de los hijos falla. También podemos usar los decoradores en los distintos nodos, hay distintos decoradores y su función es modificar el orden de las *task* y añadir funcionalidad a la hora de de ejecutarlas. Las *task* son las hojas o nodos finales del árbol, las acciones a realizar, a pesar de haber *task* definidas por el propio motor podemos crear funciones que sirvan como hojas personalizados con funciones mas complejas que las que vienen ya implementadas y que realicen las acciones que nos sean necesarias, de hecho sera necesario crearlas para poder definir el comportamiento de nuestras inteligencias artificiales dado el escaso catálogo de *task* con el que cuenta Unreal Engine.

- Blackboard: En este *blueprint* de Unreal Engine se almacenan los datos y variables necesarios de una instancia específica de un componente de inteligencia artificial para que un árbol de comportamiento pueda hacer referencia de ellas. Al estar directamente vinculados con los árboles de comportamiento, en la interfaz podremos pasar directamente de un árbol de comportamiento a su *blackboard* asociada y viceversa.

- AI Controller: Es un controlador en el que indicamos que *blackboard* y que *behavior tree* vamos a usar, este controlador se encargara de controlar de forma autónoma y ajena al jugador, a un personaje del juego. La forma en que lo hace es mediante la observación y toma de decisiones reaccionando al mundo que le rodea. En el controlador aparte de ejecutar el árbol de comportamiento, tendremos que gestionar los posibles cambios que se produzcan en las variables, tanto las contenidas en el propio controlador como las que fuesen añadidas a la *blackboard*. Aunque la gestión se hace de forma autónoma, ha tenido que ser programada previamente indicándole que funciones usar o como modificará las variables en función de que acciones recibidas.

- AI Perception: Aunque pueda parecer que no esta tan vinculado a las clases anteriores es un componente realmente necesario si queremos que la inteligencia artificial sea capaz de interactuar con objetos del mundo mediante la simulación de sentidos como la vista o el oído. Es un componente que agregaremos a los blueprint de los personajes, con el podemos añadir sensores al personaje de forma que sea capaz de escuchar, ver, tocar ... de forma que pueda reaccionar a estímulos externos y de esta forma poder ser capaz de cambiar las variables en consecuencia y poder realizar acciones en función de estos cambios.

Estas son las clases que podemos utilizar en Unreal Engine si solo tenemos en cuenta las herramientas básicas proporcionadas por el propio motor de Unreal Engine 4, por otro lado la comunidad de Unreal ha creado una serie de plugins que o bien nos facilitan el trabajar con las herramientas mencionadas anteriormente o bien porque utilizando esas herramientas ellos ya han creado inteligencias artificiales funcionales que nos puedan servir o bien porque han creado alguna herramienta que llega mas lejos que las básicas proporcionadas por el propio motor como puedan ser las redes neuronales.

El Dr. Ismael Sagredo Olivenza en su tesis *Aplicación de técnicas de aprendizaje automático supervisables por el diseñador al desarrollo de agentes inteligentes en videojuegos* (Sagredo, s.f.) hace un análisis sobre Unreal Engine y mas concretamente sobre la utilización de arboles de comportamiento para la creación de inteligencias artificiales señalando algunas dificultades que presenta e indicando algunos cambios en su implementan en Unreal Engine frente a la implementan tradicional tradicional de los mismos, como por ejemplo la desaparición en Unreal Engine del nodo *parallel*, que si es utilizado comúnmente en arboles de comportamiento.

Uno de los problemas que nos podemos encontrar con la creación de inteligencias artificiales en Unreal Engine es que si trabajamos solo con el *AIController* se puede crear una IA muy simple fácilmente, pero si queremos hacerla mas completa la curva de complejidad se hace muy pronunciada, pero por otro lado si trabajamos con *Behavior trees* tenemos que trabajar obligatoriamente con las *blackboard* y con el *AIController* incrementando mucho la complejidad si no se ha trabajado nunca con ellos o no se tienen muy claros algunos conceptos sobre su utilización, pues a pesar de que los arboles de comportamiento suelen parecer fáciles de usar, tienen una curva de aprendizaje muy grande tanto para personas que están empezando como para personas que si tienen experiencia en Unreal Engine pero que no han profundizado en la utilización de los arboles de comportamiento como comentábamos anteriormente. En el *blackboard* definimos las variables que se usaran el arbol de comportamiento, pero estas variables serán modificadas en el *AIController*, por otro lado las distintas *task*, *decorator* ... son definidos como funciones externas que también necesitaran de variables para pasar la información necesaria, siendo fácil perderse y no saber donde hay que hacer cada cosa, pues tenemos tres clases muy relacionadas entre si y en las que podemos modificar las mismas variables y funciones, siendo muy complicado el distinguir en cual tenemos que modificarlas.

2.1.2. Unity

Por otro lado Unity no ofrece tantas facilidades como Unreal Engine, tiene una clase *AI* que tendremos que usar al crear nuestra propia inteligencia artificial. De tal forma que al crear la clase de nuestra inteligencia artificial escribiremos en la cabecera *Using UnityEngine.AI* indicando que podremos usar todas funciones implementadas en la clase AI y después procederemos a implementar nuestra clase con las funciones que queramos crear en ella. Comparándolo con Unreal Engine, al contar únicamente con la clase que creemos para la inteligencia artificial y la clase creada por Unity que nos proporcionará las funciones base que necesitemos, sabemos que únicamente tendremos que trabajar en la clase que creemos evitándonos confusiones sobre donde modificar las variables, esto nos da la ventaja de poder trabajar desde cero en la inteligencia artificial sin necesidad de seguir un camino ya establecido anteriormente como podemos observar en Unreal Engine que prácticamente obliga a usar arboles de comportamiento, también al contar únicamente con la clase de la inteligencia artificial si encontramos algún problema o no se comporta como deseamos el error solo puede estar en esa clase. Por otro lado obliga al usuario a tener una base bien establecida de conocimientos sobre inteligencia artificial y programación si queremos crear una inteligencia artificial, siendo complicado que una persona intente crear una viendo algún tutorial que es la imagen que da en ocasiones Unreal Engine.

Podríamos decir que es un motor puramente orientado para programadores experimentados, dando la oportunidad de implementar todo en código pero que no da tantas facilidades como Unreal Engine a personas que puedan experimentar con el desarrollo de videojuegos con unos conocimientos pobres de programación o que aunque si tengan conocimientos amplios de programación no tienen experiencia en las diversas áreas necesarias en la creación de un videojuego.

2.1.3. Herramientas basadas en Utilidad

Conociendo ya las herramientas con las que cuenta cada motor de base, nosotros queríamos algo diferente de lo que ya aportaban, lo que queríamos era crear un sistema que hiciese uso de la Teoría de la Utilidad para crear sus inteligencias artificiales, aunque la idea era crear la herramienta para Unreal Engine busqué herramientas tanto de Unreal Engine como de Unity lo que encontramos fueron dos opciones una para Unity y otra para Unreal Engine, que serán explicadas a continuación.

Apex Utility AI

Esta es la opción de Unity, es una herramienta de pago que hace uso de la Teoría de la Utilidad, la interfaz es muy similar a la de Unreal Engine, haciendo uso de cajas como si fuesen nodos de una red, en dichas cajas parecen estar las posibles acciones a realizar. No deja muy claro como se trata la utilidad o como modificar los valores internos del personaje para que sea susceptible o no a los distintos aspectos del juego que le hacen cambiar de una acción a otra. Promete ser una herramienta que hace uso de la teoría de la utilidad para crear inteligencias artificiales diciendo que facilita la creación, utilización y verificación de errores, pudiendo ser ampliada por desarrolladores.

Cuentan con videos donde se pueden ver ejemplos de como se comporta una inteligencia creada mediante este sistema, así como una breve explicación de como crearla que aunque puede no resolver del todo nuestras dudas nos da una idea general, pero si queremos ver mas detalladamente como funciona cuenta con videos y documentación mas detallada donde explica en detalle diferentes aspectos de la herramienta.

Parecía una herramienta muy similar a lo que pretendíamos crear, una candidata a intentar emular en Unreal Engine, al menos parcialmente pues algunos aspectos parecían un poco confusos como la utilización de las curvas de utilidad, que aunque mencionadas no parecían ser vistas en ningún momento y parecían utilizar números enteros y racionales para calcular las utilidades pero en ningún momento parecían ser aplicados en una curva o función.

En la siguiente imagen por ejemplo muestra valores en función de la proximidad al enemigo, pero el tratamiento de esos valores a la hora de calcular la utilidad de una acción parece ser directo sin depender de una curva.

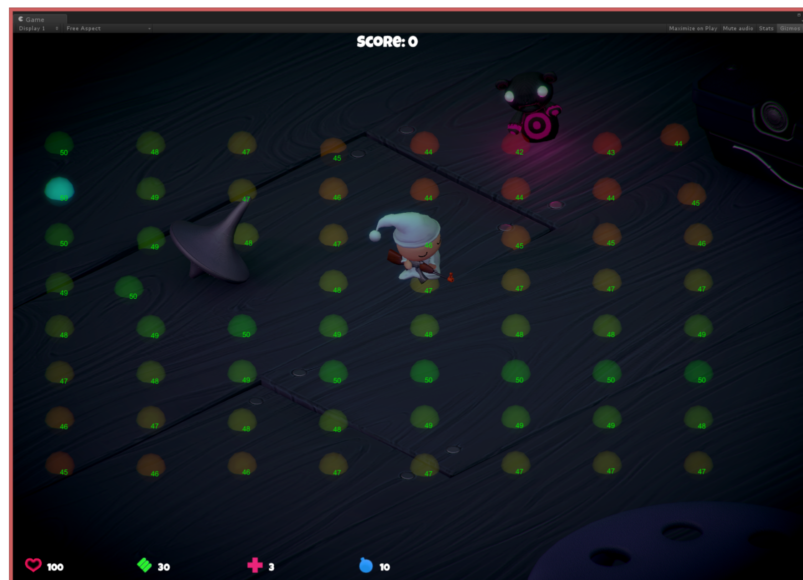


Figura 2.1: Ejemplo Apex Utility

Por lo tanto a falta de acceder directamente a la herramienta e intentar utilizarla, podemos ver que si que hace uso de la teoría de la utilidad, pero hay cosas tanto del tratamiento de los valores, como de la utilización de la herramienta que no quedan para nada claras, por lo tanto cada uno tendrá que juzgar si vale la pena en función de las necesidades de cada uno y de lo que consigamos aprender antes de acceder a la herramienta.

BT Utility Plugin

Por otro lado la propuesta de Unreal Engine que encontramos era un plugin creado por un desarrollador independiente orientado a los árboles de comportamiento. Este desarrollador lo que a conseguido es añadir el uso de la utilidad en los árboles de comportamiento que ya nos proporciona Unreal Engine, para ello han añadido un *composite* que ha llamado *Utility* en este nodo lo que hace es calcular la utilidad de las distintas ramas reordenando las ramas del nodo de tal forma que las ejecuta por orden de utilidad, es una buena forma de añadir la utilidad a los árboles de comportamiento pero no tenemos que olvidar que siguen siendo árboles de comportamiento y por lo tanto siguen teniendo los mismas limitaciones y problemas aunque le añadamos el uso de utilidad.

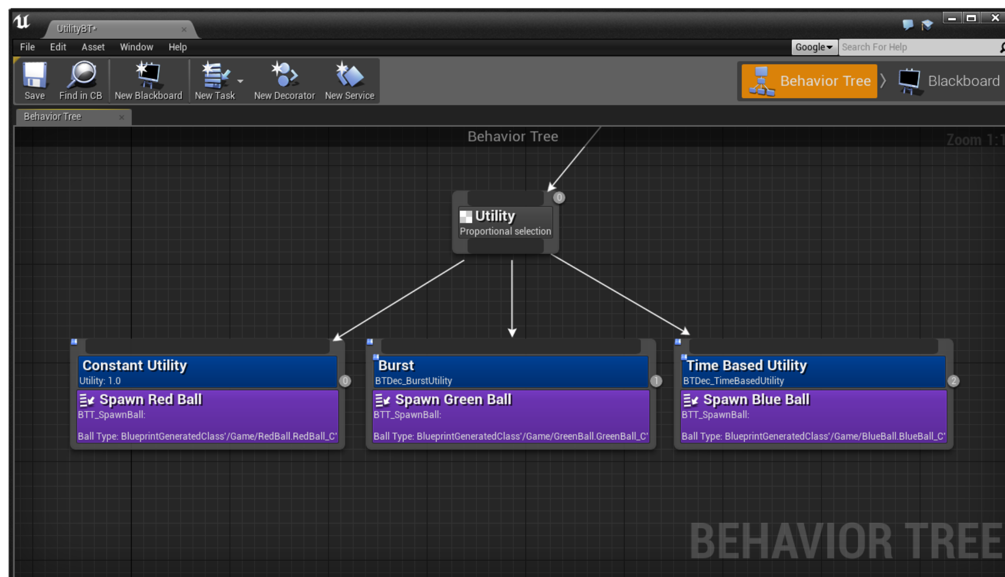


Figura 2.2: Behavior Tree Utility

Los problemas a los que me refiero son:

Cuando un árbol ejecuta una hoja no deja de ejecutarla hasta que falle, de tal forma que si no falla pero no es capaz de alcanzar su objetivo se puede quedar bloqueado intentando ejecutar una hoja, la única forma de evitar esto es poniendo limitaciones de tiempo que corten la ejecución, otro de los problemas es que al ejecutar las hojas la única forma de que la IA reaccione a estímulos externos para cancelar una acción seria evaluando esos estimulaos externos en las hojas por lo tanto tendríamos que modificar todas las hojas creadas para que sean capaces de reaccionar a todos los distintos estímulos que queramos, teniendo que crear no solo las hojas para cada acción que queramos que se tenga que realizar, si no que también tendríamos que modificar todas las hojas ya creadas, después de esto tendríamos que añadir un *decorator* al nodo para que cuando se cancele la ejecución de la acción el nodo sea capaz de saber cual es la hoja a la que tiene que priorizar . Por otro lado como tendremos que seguir trabajando con árboles de comportamiento la complejidad de crear inteligencias artificiales no solo no se reduce, sino que se incrementa pues tendremos que incluir la dificultad de la inclusión de la utilidad en los arboles, aunque no encajaba el todo con lo que pretendíamos se empezaba a acercar bastante a nuestros objetivos, pues realmente si que hacia uso de la teoría e la utilidad.

Aunque encontramos muchas herramientas orientadas a mejorar la inteligencia artificial para los motores en los que nos enfocamos, realmente aparecieron muy pocas que hiciesen uso de la utilidad o la mencionases de alguna forma, estas fueron las únicas herramientas que encontramos que se podían adaptar a lo que queríamos conseguir con este proyecto.

A continuación daremos una explicación de que es la teoría de la utilidad para que se pueda ver que es exactamente y cual es la funcionalidad que queremos añadir a las inteligencias artificiales que se puedan crear en Unreal Engine.

2.2. Teoría de Utilidad

Para poder dar una idea más clara de lo que queremos conseguir en este proyecto tenemos que introducir el concepto de utilidad que es lo que queremos que se pueda utilizar en las inteligencias artificiales de Unreal Engine.

La teoría de la utilidad se usó inicialmente en economía, siendo León Walras el que formaliza matemáticamente el tratamiento de la utilidad que seria mejorado más tarde por von Neumann y Morgenstern en su libro *The Theory of Games and Economic Behavior* (von Neumann J. Morgenstern O., 1944). Esta teoría explica que siendo el consumidor consciente de toda la información sobre los servicios o bienes que se venden en el mercado y que no varían en función de las acciones del consumidor, la única forma de medir la utilidad de estos bienes o servicios para el consumidor es una escala subjetiva de gustos única para cada consumidor, de tal forma que el valor (utilidad) que cada consumidor le da a dichos bienes, es completamente subjetivo y depende de múltiples variables únicas para cada consumidor.

En el campo de la informática lo que nos motiva a hacer uso de la utilidad para crear inteligencias artificiales es que asumimos que el mundo esta sujeto a cambios que son parcialmente observables y nuestra inteligencia artificial tendrá que elegir en función a esos cambios pensando en el resultado inmediato que mas útil sea para él, pero al igual que en la explicación anterior sobre la utilidad la elección dependerá de nuestro agente pudiendo hacer que agentes distintos elijan de forma diferente en función de ciertas variables internas que serán las que nos ayuden a calcular la utilidad de tomar una acción para cada agente, por ejemplo un agente podría tener miedo de un individuo al verlo pero otro simplemente lo ignoraría o elegiría atacarlo, este cambio de un agente a otro podría estar relacionado con una variable que nos indique la agresividad, la forma de calcular la utilidad de una acción en función de estas variables serán las curvas de utilidad, que nos indicaran los correspondientes valores de utilidad en función de los valores de las variables.

Estas curvas serán algo completamente necesario para nosotros pues sin ellas tendríamos que dividir en rangos todas las variables o asignar a cada valor su respectiva utilidad por cada punto de la curva, esto resultaría en una cantidad de trabajo que haría inviable el plantearnos hacer varias inteligencias artificiales teniendo en cuenta que cada una contaría con varias variables que tendríamos que utilizar, por otro lado con las curvas solo tendremos que generar una curva por cada variable, pudiendo reutilizar estas curvas en otras variables o inteligencias artificiales. Por ejemplo tengamos en cuenta la siguiente gráfica:

Figura 2.3: Curva de Utilidad

Tomando como ejemplo esta gráfica sea cual sea la variable con la que la usemos podremos ver que si el valor de la variable que será la indicada en el eje x es 0, la utilidad resultante que será la indicada en el eje y sera 1, pero a partir del valor 20 de la variable la utilidad pasara a tener un valor de 0. Estas curvas son las que nos asignan la utilidad de tal forma que los valores de utilidad se calcularan en función de estas curvas, que pueden ser directa, inversamente proporcionales o tener distintos vértices.

Normalmente la teoría de la utilidad se usa conjuntamente con la teoría de la probabilidad, dando lugar a la teoría de la decisión. En la teoría de la utilidad asumimos un mundo parcialmente observable en el que solo podemos actuar en función del estado del mundo que observamos y del que somos conscientes, y haciéndolo en función de las variables únicas de cada personaje o sujeto. Por otro lado en la teoría de la decisión, reaccionamos a un estimulo externo, pero la decisión de que acción tomar sobre este estimulo no se debe directamente a este estimulo si no a la utilidad de las posibles consecuencias derivadas de las acciones que tomemos, dando lugar de esta forma a que una acción con consecuencias de una gran utilidad pero con una probabilidad casi nula de que ocurra podría no ser elegida nunca frente a otra con consecuencias menos útiles pero con mas posibilidades se ocurrir , siendo mas rigurosos lo que hace es elegir la acción mas útil en promedio de los estados esperados utilizando la probabilidad para ponderar dichos estados. Siendo así la teoría de decisión no actúa directamente en función del estimulo si no que tiene en cuenta las todas las posibles consecuencias de las acciones derivadas de actuar a ese estimulo, es decir piensa o actúa en función de lo que pueda pasar en el futuro, aunque este podría considerarse un comportamiento muy humano al tomar este tipo de decisiones empezamos a tomar el mundo de una forma cada vez mas observable dando lugar a pensar si en realidad se están tomando en cuenta cosas que en realidad no tendríamos forma de saber, en ocasiones cuando nos enfrentamos a inteligencias artificiales que usan la decisión en videojuegos nos puede dar la sensación de que hacen trampa pues parecen por ejemplo tomar decisiones en función de la vida que nos queda cosa en en principio no tendrían por que saber, por otro lado también podríamos pensar que tienen un comportamiento muy humano pues lo que están haciendo en cierta forma es adivinar, donde vamos a estar o la vida que nos queda pero en realidad lo que están calculando son probabilidades.

2.3. Accesibilidad

Con la teoría de la utilidad explicada, hemos cubierto uno de los puntos que queremos conseguir con nuestro sistema de control de bots que es la inclusión de la teoría de la utilidad en la creación de inteligencias artificiales en Unreal Engine, pero otro punto muy importante es la accesibilidad. Queremos que, contrariamente a la dificultad que hemos encontrado en el uso de arboles de comportamiento en Unreal Engine, sobre todo para personas que no sean programadores o los denominados *no-técnicos*, nuestro sistema tenga una serie de pasos muy definidos de forma que se puedan seguir como una guía a la hora de crear las inteligencias artificiales. Algunas de estas dificultades que encontramos en los arboles de comportamiento son mencionadas por el Dr. Ismael Sagredo Olivenza en su tesis *Aplicación de técnicas de aprendizaje automático supervisables por el diseñador al desarrollo de agentes inteligentes en videojuegos*(Sagredo, s.f.), cuando analiza la utilización de los arboles de comportamiento en Unreal Engine. De forma que para conseguir que nuestro sistema pueda ser utilizado por personas con conocimientos muy básicos de programación o no sean programadores tendremos que hablar de la facilidad, simplicidad o lo intuitivo de nuestro sistema pero este punto es difícil de abordar pues es algo muy subjetivo que puede ser diferente para cada individuo que use nuestro sistema.

Es obvio que siempre que intentamos hacer una herramienta intentamos que sea lo mas fácil de usar posible, por ello cuando al usar Unreal Engine que parece dar muchas facilidades en este sentido con una interfaz gráfica relativamente fácil de usar, nos encontramos con los arboles de comportamiento empezamos a plantearnos si realmente es tan sencillo de utilizar como plantean, otras aspectos también nos hacen plantearnos este hecho pero analizaremos el caso de los árboles de comportamiento que es realmente la parte de Unreal Engine que esta mas relacionada con nuestro sistema.

Cuando trabajamos con los árboles de comportamiento en Unreal Engine, inicialmente parecen muy sencillos pero que cuando profundizas trabajando con ellos la curva de aprendizaje en algunos casos se hace difícil de superar, puesto que la interacción entre las varias clases que hay que utilizar conjuntamente con ellos hace difícil saber donde hay que incluir cada cosa, o en caso de fallar donde hay que arreglar los fallos entre las varias clases que estamos usando.

Para intentar suplir estas dificultades hay tutoriales proporcionados por el propio Unreal Engine, por supuesto la comunidad de programadores de Epic Game siempre esta dispuesta a ayudar y claro que también podemos encontrar ayuda externa en videotutoriales o foros, pero lo ideal seria que solo con la herramienta y las instrucciones dadas fuésemos capaces de usarla sin necesitar semanas o meses en conseguir algo básico que necesitemos por ello parece que esta accesibilidad si que la consigue cuando hablamos de la interfaz, pero cuando hablamos de como trabajar con el programa en si, encontramos problemas y puede ser complicado.

Para intentar solucionar estos problemas que encontramos en Unreal Engine nosotros lo que hemos intentado que nuestro sistema de control de bots es que tenga una serie de instrucciones muy claras y sistematizadas que simplemente siguiéndolas seamos capaces de utilizar sin necesidad de ayuda externa o al menos minimizándola.

Capítulo 3

Objetivos y especificación

El objetivo principal de este proyecto es implementar un sistema de control de bots para Unreal Engine, que nos de la posibilidad de poder usar la teoría de la utilidad en los controladores de IA de dicho motor, de esta forma aparte de tener con los árboles de comportamiento con los que cuenta Unreal Engine tendríamos otra forma para definir el comportamiento de los bots. Para ello tendremos que crear una serie de clases que nos permitan tener los datos necesarios para trabajar con la utilidad tal y como lo hemos pensado y que nos permitan calcular los valores que necesitamos, en estas clases tendremos que poder declarar las acciones que se podrán realizar y se tendrá que poder trabajar con las gráficas calculando valores en estas. También crearemos un tutorial para poder mostrar como crear y usar dichas clases de forma que ayuden a cualquier persona que quiera utilizarlo a familiarizarse con los procedimientos que hay que seguir de cara a que siguiendo dichos procedimientos de forma sistemática sea capaz de crear una inteligencia artificial que haga uso de la utilidad acorde las necesidades del usuario.

Para poder mostrar de que forma funciona este sistema de control de bots que hemos llamado UtilityNetwork también se creará una pequeña demostración o simulación en la que se implementarán varias inteligencias artificiales que hagan uso de la utilidad pero que cuenten tanto con curvas diferentes para las mismas acciones, como con formas distintas de tratar la utilidad, simple (en el caso de usar una única variable para calcular el nivel de utilidad) como compleja (se podrán usar múltiples variables para calcular el nivel de utilidad). De esta forma podremos observar a varias inteligencias artificiales con comportamientos aparentemente iguales que reaccionaran de formas ligeramente distintas a estímulos externos o a los cambios que estos producen en su entorno .

Para poder crear esta demostración crearemos algunas acciones que estarán a disposición del usuario, pero el usuario tendrá que crear sus propias acciones acordes a lo que quiera que su personaje haga en el juego o demostración, todas estas acciones se crearan en una librería de funciones llamada *UtilityActions*.

3.1. Objetivos

Los objetivos generales del proyecto que debemos cumplir para poder desarrollar el sistema de UtilityNetwork en Unreal Engine son los siguientes:

- Conseguir crear una clase que nos permita definir el nodo que contara con los campos necesarios para que pueda ser calculada la utilidad.
- Conseguir que la clase anterior pueda evaluar la utilidad tanto con una única variable como con varias.
- Crear otra clase que nos permita trabajar con los nodos creados, de cara a que sean evaluados en su totalidad y nos devuelva la opción a tomar.
- Permitir la creación de gráficas que nos permita poder calcular la puntuación que se consigue en un nodo en función de ciertas variables.

- Proporcionar una serie de acciones básicas que podrán ser aumentadas por el usuario.
- Crear mapas donde poder visualizar los distintos comportamientos.

3.2. Especificación de requisitos software

A continuación explicaremos brevemente cuales son los requisitos software así como la interacción de los usuarios con dichas partes del software,tanto los necesarios para llevar a a cabo la finalización del sistema, como aquellos necesarios por el sistema para realizar ciertas tareas

3.2.1. Nodos de Utilidad

Estos nodos son *UtilityNode y UtilityNodeComplex*, son las estructuras que el usuario tiene que crear e inicializar para poder ser capaz de calcular la utilidad de realizar ciertas acciones, es la base del sistema y sin estos nodos no habría nada que calcular en el controlador.

3.2.2. Controlador

Esta clase es *MasterUtilityController*, es donde trabajará el usuario, junto con los nodos de utilidad es la la parte mas importante pues esta clase ya cuenta con las funciones necesarias para trabajar con los nodos así como las variables que interconectan el controlador con el bot, si no existiese el usuario tendría que utilizar *AIController*, implementando todas las funciones y variables cada vez que crease una clase de este tipo, pero con *MasterUtility-Controller*, bastará con crear un hijo y trabajar en él o también podemos hacer una copia suya en la que poder trabajar.

3.2.3. Librerías

Como queremos ver si una persona es capaz de crear una inteligencia artificial desde cero con UtilityNetwork, hacer que sigan los pasos no es necesariamente un problema, pero las acciones que queremos que se realicen si pueden serlo , pues una persona sin conocimientos o con conocimientos muy básicos de Unreal Engine puede tener problemas en este aspecto debido a la curva de aprendizaje de este motor. Por esta razón decidimos que era necesario crear una librería de funciones en las que se implementarán todas las funciones que cumplan las funciones de acciones de los NPC, así como de las funciones auxiliares necesarias para poder crear estas funciones. Por otro lado si un usuario quisiese añadir nuevas acciones a las inteligencias artificiales seria en esta librería donde se recomienda que sean añadidas.

3.2.4. Mapas de Pruebas

Para este proyecto hemos decidido crear una serie de mapas que nos permitan mostrar que las inteligencias artificiales creadas usando el método implementado en este sistema pueden reaccionar o actuar en función de como cambia el mundo observable por dicha inteligencia artificial y que dichas reacciones puedan ser completamente diferentes si usamos acciones diferentes o similares si usamos las mismas acciones pero modificando sus curvas.

3.2.4.1. Comparación entre Métodos

El objetivo de este mapa es intentar comparar como se comportan dos inteligencias artificiales que intentan tener el mismo comportamiento pero que están implementadas con dos métodos distintos, en nuestro caso una de ellas esta utilizando los arboles de comportamiento (behavior tree) que proporciona Unreal Engine y la otra esta utilizando nuestro sistema que hace uso de la utilidad para definir el comportamiento.

Para esta comparación, la inteligencia artificial que hemos creado lo que intenta es recorrer el mapa tratando de comer o coger una serie de botiquines y en caso de que se cruce con nuestro personaje (el jugador) debería huir de nosotros o detener su movimiento esperando a que nos alejemos.

En este mapa sera el observador el que determine las fallas y carencias o beneficios y aciertos de cada uno de los métodos.

3.2.4.2. Comparaciones con mismo método

Para estos mapas de prueba, las dos inteligencias a comparar estarán utilizando el mismo método, nuestro sistema de control de bots. Pero las diferencias entre los NPC serán las curvas usadas de tal forma que podremos observar que aunque a simple vista parecen hacer lo mismo, su comportamiento tiene algunas diferencias en función de nuestra interacciones con ellas y de como se defina en su curva de utilidad, la utilidad de realizar ciertas acciones en según que momentos.

3.2.5. Tutoriales

De cara a enseñar como crear una inteligencia artificial usando *UtilityNetwork* hemos creado un tutorial que estará disponible en español y en ingles, y en el que se detallarán los pasos a seguir para crear una inteligencia artificial. Constara de una serie de diapositivas que detallarán los pasos a seguir y resaltaran las clases y funciones a las que nos referimos en cada caso. Este tutorial también sera utilizado a la hora de realizar las pruebas con usuarios para comprobar las dificultades que se puedan encontrar a la hora de seguirlo, pues uno de los objetivos es que pueda UtilityNetwork pueda ser utilizado tanto con personas con experiencia previa en Unreal Engine la cual podría hacerse una idea rápida de como se usa simplemente explicando la función de las clases creadas, pero también esta orientada a servir de guía para personas con poca o ninguna experiencia que están empezando que lo que necesitan es poder seguir una guía rápida de como hacerlo sin necesidad de dedicar mucho tiempo a aprenderlo.

Unreal Engine ya cuenta con tutoriales sobre el uso de árboles de comportamiento por lo cual aunque inicialmente planteamos crear un tutorial también de árboles de comportamiento no creemos poderlo crear igual de completo y lo que es mas importante no seria capaz de simplificarlo lo suficiente como para que una persona con pocos conocimientos pudiese verlo y ser capaz de crear una inteligencia artificial lo suficientemente compleja como para equipararse a una creada con *UtilityNetwork*, de tal forma que recurriremos a los tutoriales que proporciona Unreal Engine en este campo o la la experiencia previa de los usuarios si es que la tienen cuando preguntemos por la comparativa entre UtilityNetwork y arboles de comportamiento.

Capítulo 4

Metodología y herramientas

De cara a la realización del proyecto nos planteamos que tipo de metodologías deberíamos usar si ágiles o tradicionales. Podríamos usar perfectamente un método tradicional de trabajo dado que la fecha de finalización es conocida desde el inicio del proyecto, pero dado que tendríamos reuniones de forma regular para revisar el trabajo pensamos que podríamos utilizar esto para decidir seguir una metodología ágil que pensábamos podría ayudarnos mas a llevar el trabajo a buen termino.

4.1. Metodología

A la hora de decidir que tipo de metodología elegir nos inspiramos en metodologías ágiles y tomamos Scrum como referencia. Aunque decidimos no seguir Scrum fielmente, si que decidimos usar algunos de los aspectos que definen esta metodología.

El motivo que nos llevo a afrontar este proyecto mediante este tipo de metodología fue que aunque teníamos claro donde queríamos llegar en un primer momento, sabíamos que era posible que los objetivos cambiaran dado que no sabíamos que problemas nos encontraríamos o como afrontar el llegar donde queríamos, de tal forma que pensamos que la mejor forma de conseguir llegar al objetivo final era irnos marcando diferentes hitos o subproblemas hasta llegar al final y dado que teníamos estos pequeños objetivos podíamos reaccionar a posibles cambios que se produjesen. De esta forma intentábamos conseguir el hito marcado cada vez y en cada reunión evaluábamos los problemas que habíamos tenido para llegar al objetivo marcado y en caso de no haber podido llegar formas de solucionarlo para conseguir llegar e ir un paso mas allá, dando lugar a periodos en los que los objetivos de la reunión siguiente era arreglar o llegar fielmente a los objetivos del sprint anterior en caso de que nos fuese imposible llegar.

Así, como parte de la metodología, teníamos que considerar los siguientes elementos que son usado en Scrum y que nosotros decidimos usar en nuestro desarrollo.

- ***Product Owner*:** Es la persona responsable del producto, tiene una visión clara de lo que se va a construir y se encarga de gestionar el trabajo del equipo intentando conseguir los mejores resultados en la finalización del producto . En nuestro caso, ese rol lo cumple el director del TFG.

- **Reunión de planificación del Sprint:** Se deciden las funcionalidades que se incluirán en el sprint, propuestas por el *Product Owner* y estimadas por el equipo de desarrollo.

- **Desarrollo del Sprint:** Cada uno de los desarrolladores coge una tarea del sprit y trabaja en ella hasta que está completa o surge una dificultad importante. Trabajando de esta forma hasta la finalizacion de esta fase.

- **Reunión de revisión:** Los resultados obtenidos cuando finaliza la fase anterior deben ser presentados al *Product Owner* para ser evaluados.

- **Reunión de retrospectiva:** Se evalúa el desarrollo del sprint, problemas encontrados con el proceso y las herramientas..., indicando el grado de satisfacción obtenido de los resultados de forma que se pueda organizar los objetivos del siguiente hito.

4.1.0.1. Flujo de trabajo: El desarrollo de un Sprint

Para poder concretar lo explicado anteriormente en nuestro proyecto, describiremos cómo aplicamos las metodologías en nuestro proyecto describiendo a continuación el desarrollo genérico de un par de nuestros sprints, Aunque estos sprint los realizábamos durante todo el año he decidido poner estos dos que pertenecen a los últimos meses del desarrollo e decidido coger estos dos porque en este momento del desarrollo decidimos hacer sprint mas cortos y también porque surgieron algunos problemas durante su realización, no cumpliendo completamente alguno de los hitos. De todos lo hitos realizados podríamos dividirlos en dos partes, los primeros que se realizaron durante la primera mitad del proyecto que consistían en su mayoría en reunir documentación de cara a prepararnos para el trabajo a realizar, y los sprint realizados durante la segunda mitad del proyecto cuando realmente empezamos con la creación de nuestro sistema de control de bots.

De esta forma es como abordamos dos de nuestros últimos sprint: **Sprint del 30/6/2018**

1. **Comienzo del sprint: reunión** Lo primero que hacemos es en función de como acabamos el sprint anterior plantear los objetivos que tendremos que conseguir para la próxima reunión. En este caso al vernos obligados a presentar mas tarde el proyecto, decidimos hacer sprint mas cortos con hitos muy concisos, para el hito de este sprint planteamos que una de las cosas que teníamos que hacer para facilitar el uso del sistema era crear una serie de acciones ya definidas que pudiesen ser utilizadas por los usuarios en las inteligencias artificiales creadas.

2. **Desarrollo del sprint** Durante el tiempo hasta la próxima reunión intentamos realizar lo planteado en la reunión intentando cumplir fielmente con los objetivos. Inicialmente realizamos estas acciones como *custom event* dentro del propio controlador de la inteligencia artificial, pensando que al crear hijos de este controlador el usuario ya tendría acceso a estas acciones, mas tarde se decidió cambiarlo poniéndolas en una librería externa para facilitar la creación de mas acciones por parte de los usuarios.

3. **Cierre del sprint** Al comienzo de la reunión, se sumariza lo hecho desde la anterior reunión, se comentan los resultados y se validan. También se comentan las dificultades encontradas tanto con el producto como con el proceso. Equivale a la Reunión de Revisión y a la Reunión de Retrospectiva. En este caso el desarrollo no nos dio muchos problemas y fuimos capaces de llegar al objetivo propuesto y creamos varias acciones de forma satisfactoria.

Sprint del 7/7/2018

1. **Comienzo del sprint: reunión** Como objetivo para este sprint propusimos hacer una pequeña interfaz que facilitase el ver cuando se realizan las distintas acciones en los mapas donde comparamos las inteligencias artificiales, otro de los objetivos era subir el proyecto a *GitHub* pues al trabajar yo solo no vi inicialmente la necesidad de usar ningún control de versiones mas allá de hacer una copia de seguridad por si tenia algún problema serio.

2. **Desarrollo del sprint** Durante el desarrollo en este caso si nos encontramos con algunos problemas entre ellos la comunicación entre la interfaz y los NPC, no eran capaces de transmitir bien sus datos y por lo tanto no se mostraba en pantalla, al final se consiguió solucionar y mostraba todo correctamente. Por otro lado hubo problemas a la hora de subir el proyecto a a *GitHub*, debido al limite de 100 MB que pone *GitHub* ha la hora de subir ficheros, pues no deben ser superiores a dicha capacidad.

3. **Cierre del sprint** Se hablo de los problemas mencionados anteriormente,con los problemas respecto a las interfaces al solucionarse, se tomo nota de como hacerlo y ya, pero el problema con *GitHub* fue mas complicado, se decidió usar *GitHub* junto *Git LFS* que permitía la subida de archivos con capacidades superiores al limite permitido por *GitHub*. Finalmente se acabaría decidiendo usar *Perforce* dado que al usar *Git LFS* se limita el proyecto a 1 GB no pudiendo rebasarse este limite

Una vez que teníamos decido que metodología usaríamos, lo siguiente era pensar que programas y herramientas nos ayudarían en el desarrollo del proyecto, claro que aunque inicialmente teníamos claro algunas de las herramientas que usaríamos, algunas se modificaron durante el desarrollo del proyecto y no se usaron o se cambiaron por otras que nos convenían mas.

4.2. Herramientas

Durante el desarrollo del proyecto, y para cubrir algunos aspectos necesarios del proyecto como la comunicación, el desarrollo del software, el control de versiones, la compartición de recursos decidimos emplear las siguientes herramientas:

4.2.1. Desarrollo

Para poder desarrollar el sistema de control de bots había algunas herramientas que eran imprescindibles para poder llevarlo a cabo a continuación presentaremos dichas herramientas explicando algunos detalles sobre ellas y su función durante el desarrollo.

4.2.1.1. Unreal Engine

Es el motor gráfico creado por Epic Game para el cual decidimos crear nuestro sistema de control de bots mediante el uso de una inteligencia artificial que utilice la teoría de la utilidad.

Este motor gráfico esta a disposición de todo el mundo de forma gratuita para poder desarrollar videojuegos principalmente, cuenta con una interfaz gráfica que hace fácil su uso y es uno de los motores gratuitos mas conocidos junto con Unity.

Respecto a Unreal Engine decidimos usar la versión 4.19 del motor, esto se debe a que es una versión desarrollada después de la creación de Fortnite un juego creado por Epic Game con este motor, después del cual la compañía mejoró muchos aspectos del motor, dando como resultado una versión bastante robusta y en la que se solucionaban problemas con los que ellos mismo se habían encontrado a la hora del desarrollo.

Dentro de Unreal Engine, para la parte mas visual se utilizaron materiales, texturas y objetos del juego Paragon, que también fue creado con Unreal Engine. Estos recursos fueron liberado por parte de la compañía para poder ser utilizados de forma libre y gratuita por los usuarios.

4.2.1.2. Visual Studio

Es un entorno de desarrollo integrado que soporta diferentes lenguajes, en nuestro caso hemos seleccionado este entorno porque aunque Unreal Engine en su interfaz gráfica permite programar sin utilizar código de forma directa mediante un conjunto de“cajas” o nodos que actúan como las funciones y el uso de *“Blueprint”*, para hacer algunas cosas más complejas necesitaremos programarlas directamente en el motor y como el lenguaje de programación que usa internamente Unreal Engine es C++ y por otro lado la familiaridad con este entorno que ya habíamos usado anteriormente fue un factor determinante para elegirlo frente a otros.

Aunque finalmente no fue muy utilizado pues fuimos capaces de hacer todo mediante la interfaz gráfica del propio Unreal Engine y las cosas que nos planteamos crear mediante código, dificultaban el uso de cara a los usuarios que tendrían que trabajar con él, un ejemplo de este hecho era la creación de la estructura usada como nodo de utilidad inicialmente la cree escribiendo directamente código, pero nos dimos cuenta que haciendo esto el usuario tenia que inicializar él mismo esta estructura y todos sus campos mediante "cajas" a la hora de crear las inteligencias artificiales, pero si creaba la estructura usando la interfaz gráfica de Unreal Engine a la hora de crear una variable de ese tipo el usuario podría inicializarla directamente en la propia interfaz. Puede no parecer muy problemático la inicialización de dichas variables y realmente si estamos usando pocas no lo es pero cuando el numero de nodos se incrementa nos encontramos con que tenemos que hacer *set* de todas ellas modificando con los valores iniciales y no es muy realista el pensar que tendríamos que hacer esto, por ello la programación directa en C++ quedo en un segundo plano.

Por otro lado visual si fue usado para ejecutar el proyecto de forma que si fallaba, teníamos un mayor control sobre las excepciones capturadas pudiendo evitar las acciones que provocaban fallos en el programa.

4.2.2. Comunicación

Otro aspecto importante a la hora de desarrollar el proyecto era la comunicación entre los distintos miembros, como en muchas ocasiones no es posible la comunicación presencial hay que recurrir al uso de herramientas como las explicadas a continuación.

4.2.2.1. Slack

Es una herramienta poco conocida (aunque su uso va en aumento) bastante útil para la comunicación en un equipo de desarrollo. Gracias a esta herramienta, el director del proyecto es capaz de aunar en un solo servidor a todos los proyectos que tiene a su cargo; pero ofrece canales diferenciados por temas; posibilitando la comunicación de todos los grupos de forma separada con chats sobre temas comunes. De esta forma podíamos comunicarnos con el director del proyecto en caso de surgir algún problema o tener alguna consulta que realizarle durante el desarrollo del sprint.

4.2.2.2. Skype

En algunos casos era difícil por cuestiones ajenas al proyecto el ir presencialmente a las reuniones y una buena solución era utilizar Skype que ademas al contar con una función que permite la compartir la pantalla con el resto de usuarios, nos permitía poder enseñar el código directamente o explicar algo directamente mostrándolo y no solo de palabra a los demás.

4.2.3. Alojamiento compartido y control de versiones

También es necesario el ser capaz de compartir los recursos necesarios para la documentación y el desarrollo, así como los avances producidos en el proyecto durante su desarrollo y para eso las herramientas que decidimos usar son las siguientes.

4.2.3.1. Google Drive

Como sistema de alojamiento y compartición de ficheros utilizamos Google Drive, ya que para ficheros de documentación o para material de investigación que encontrábamos era mucho más cómodo. Además también fuimos alojando las distintas presentaciones que hicimos a lo largo del año para mostrar el avance del proyecto.

Hemos elegido esta herramienta sobre otros servicios de alojamiento en la nube debido a que posee herramientas de edición de documentos *on-line* con las que estábamos familiarizados y porque poseemos almacenamiento ilimitado gracias a que contamos con cuentas proporcionadas por la Univesidad Complutense de Madrid, que tiene un acuerdo con *Google* para ofrecer ventajas a los alumnos de dicha universidad.

Fue este también el método elegido para compartir el proyecto en si, dado que pudimos aprovechar ese espacio ilimitado solucionando las limitaciones que encontramos en otras herramientas a la hora de subir el proyecto.

4.2.3.2. *Perforce*

Es un sistema de control de versiones, inicialmente pensamos en usar *Github* pero nos encontramos con algunas dificultades como la limitación de no poder subir ficheros superiores 100 MB, esto se podía solucionar usando a su vez *Git LFS*, pero al usarlo no permitía que el proyecto fuese superior a 1 GB, requeria la compra de mas espacio, al hablarlo con el director del proyecto nos comento que el contaba con un servidor que podía ser usado por *Perforce* y estas fue la razón determinante por la que elegimos finalmente *Perforce*.

Ademas *Perforce* es capaz de conectarse directamente con Unreal Engine permitiendo subidas desde el propio proyecto facilitando y agilizando así su uso.

4.2.4. Redacción de la memoria

Aunque mediante notas a mano o archivos alojados en *GoogleDrive* tuviéramos la información necesaria para redactar la memoria, veíamos necesaria una herramienta para maquetar de forma sencilla y que permitiese el uso simultaneo de varios usuarios, muy necesario para que nos indicaran ciertos errores o consejos mientras la redactábamos de forma que se agilizara el trabajo.

4.2.4.1. *Overleaf* y *LaTeX*

Para maquetar el texto empleamos *LaTeX*, que permite la redacción y maquetación de documentos de carácter académico-científico en forma de código fuente. *Overleaf* es una herramienta online de edición de proyectos en *LaTeX*, que permite la implantación de plantillas preconstruidas.

Utilizamos esta herramienta de escritura y edición de documentos como aplicación principal para desarrollar esta memoria. *Overleaf* es una herramienta *on-line* del conocido sistema de composición de textos *LaTeX* y que además permite la implantación de plantillas como por ejemplo la que está siendo usada, que es la que propone la Facultad de Informática de la Universidad Complutense de Madrid.

Capítulo 5

Análisis, diseño e implementación

A la hora de la implementación que nos permitirá hacer uso de la teoría de la utilidad en Unreal Engine las partes mas importantes son la creación de la clase *MasterUtilityController* y de las estructuras de datos *UtilityNode* y *UtilityNodeComplex*, a continuación analizaré y explicaré el uso de estas clases y estructuras así como de las funciones, variables y campos que contienen. También explicare los pasos necesarios para poder usarlo así como las distintas formas de usar los elementos proporcionados, de tal forma que sea mas fácil hacerse una imagen de como funciona todo.

5.1. Clases y Estructuras

Para ser capaces de desarrollar un sistema de control de bots que era el objetivo final, antes teníamos que crear las pequeñas piezas que lo componen.

A continuación pasaremos a explicar cuales son estas pequeñas partes y en que consisten.

5.1.1. *UtilityNode*

Esta es una estructura para poder crear nodos donde guardaremos la información necesaria para poder determinar la mayor utilidad y poder ejecutar la acción correspondiente. Esta estructura también se puede usar para crear lo que hemos llamado ámbitos de acción, de tal forma en vez de ejecutar una acción seleccionaremos un ámbito al que meternos, de esta forma reduciremos el numero de nodos que tendremos que evaluar en cada vuelta pues solo evaluaremos los del ámbito.

A continuación daré una descripción de cada uno de los campos contenidos dentro de esta estructura.

Figura 5.1: UtilityNode

- ID: Campo de tipo *string* define el nombre del propio nodo y es la forma de identificarlo a la hora de determinar las acciones que se tienen que ejecutar.

- UtilityCurve: Campo de tipo *curvefloat* define la curva que sera usada para calcular la utilidad de un nodo, la curva sera creada fuera de esta estructura pero aquí indicaremos cual es la curva que usaremos de las que tengamos creadas.

- Mod: Campo de tipo *float* nos permitirá modificar la utilidad después de que haya sido calculada, de tal forma que se pueda pueda personalizar ligeramente el comportamiento de dos inteligencias artificiales que tengan exactamente las mismas Curvas. Pudiendo hacer que para un bot, un nodo llegue a tener mas valor o menos que los de otro bot que utiliza el mismo nodo con la misma curva, de esta forma si tenemos un modificador de 0,23 en un nodo que indique el miedo de un personaje ha ciertos estímulos, nuestro bot después de calcular la utilidad con la *utilityCurve*, el valor de la utilidad se vera incrementado en 0,23 siendo mas probable que se ejecute la función que queremos asociar a dicho nodo. Como se intenta normalizar los valores entre 0 y 1 es recomendable que los valores del modificador estén en un rango entre 1 y -1.

- Value: Campo de tipo *float* es utilizado para guardar el valor de utilidad del nodo en cuestión, inicializado a 0 al principio, se modificara cada vez que evaluemos la utilidad del nodo por el valor resultante de la curva de utilidad y será el valor con el que determinemos que nodo tiene mayor utilidad.

- Active: Campo de tipo *boolean* si su valor es *false*, el valor resultante del nodo siempre sera 0, por otro lado si lo que hacemos es ponerlo a true, se calculara realmente el valor resultante respecto a la curva. De esta forma Si es 0 no sera necesario que se realice el calculo de la utilidad en la curva.

Al observar los distintos campos del *UtilityNode*, se puede ver que usamos una única curva de utilidad, las repercusiones de esto es que a la hora calcular la utilidad se calculara sobre una única curva de tal forma que se calculara en base a un único valor, siendo este una única variable o la suma de distintas variables, para poder ejemplificarlo pensemos que queremos calcular la utilidad de un nodo llamado *Fear* que usaremos para calcular si un bot tiene miedo y de ser así huya o se mantenga alejado del objeto que le produzca temor, para esto algunos de los posibles factores que podrían aumentar este miedo serian el numero de estos objetos que vea el NPC y la distancia respecto a ellos, con este nodo a la hora de evaluar la utilidad utilizaríamos o bien la cantidad de objetos o bien la distancia, también podríamos sumar la distancia y el numero de objetos y usarlo en una única curva, pero resulta extraño y es erróneo sumar variables distintas que determinan distintas cosas como distancias o cantidades para usarlo en una única curva, para poder dar solución a este problema pensé en crear *UtilityNodeComplex* que explicare a continuación.

5.1.2. *UtilityNodeComplex*

Esta estructura es similar a la anterior y tiene la misma función que la anterior pero cuenta con algunos campos más para poder solucionar algunos problemas que tenia la estructura anterior, de esta forma los campos de esta estructura son los siguientes:

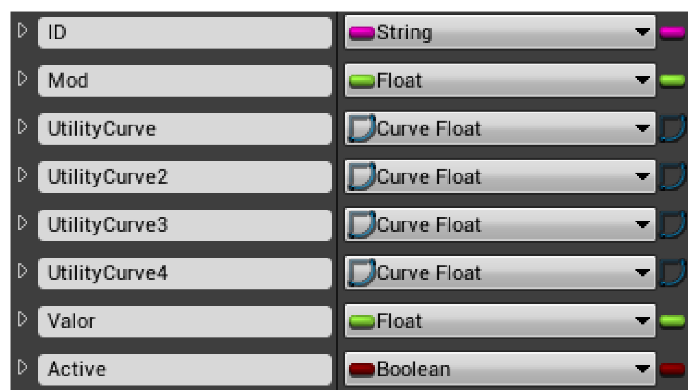


Figura 5.2: UtilityNodeComplex

No explicaremos los campos que tienen el mismo nombre que los de la estructura anterior pues su función es la misma, pero por otro lado esta estructura cuentan con tres campos mas que la anterior y todos ellos de tipo *curvefloat* estos campos existen para solucionar el problema mencionado anteriormente, de esta forma y siguiendo con el ejemplo anterior utilizaremos el nodo *Fear* y utilizaremos para calcular el valor de su utilidad el número de enemigos y la proximidad, ahora contamos con varias curvas de utilidad pudiendo usar una de ellas para ver la utilidad respecto al numero de enemigos y otra para ver la distancia, ya no hace falta sumar los distintos valores y cada curva esta orientada a un valor determinado de tal forma que los incrementos de utilidad de la curva de utilidad que usemos para ver la distancia respecto al objeto del miedo y la que usemos para ver el numero de objetos que nos dan miedo serán distintos, pues el incremento o descenso de utilidad entre estar por ejemplo a 1 metro del objeto y tener un enemigo mas no tiene porque ser el mismo que es lo que pasaba en el ejemplo anterior si elegíamos sumar las distintas variables, de esta forma podemos comparar variables distintas porque son evaluadas bajo condiciones diferentes en distintas curvas, pero resulta en un valor de utilidad que se puede sumar al resto de valores de las otras curvas.

5.1.3. *MasterUtilityController*

Es un controlador creado heredando de *IAController* y su función es la misma pero en ella será donde implementemos nuestra *UtilityNetwork*, de esta forma al utilizar esta clase como base donde implementar la inteligencia artificial he creado una serie de funciones que son necesarias para poder hacer uso de la utilidad que proporcionamos.

5.1.3.1. Funciones

A continuación explicare cada una de las funciones y variables creadas:

- EvaluateUtility: Esta función consta de dos parámetros de entrada y un parámetro de salida:

1. Utility
2. Input
3. Out Utility

Utility es el nodo de utilidad simple (*UtilityNode*) que se va a evaluar dentro de la función. *Input* es de tipo *float* y lo usaremos para calcular la utilidad del nodo. *Out Utility* es el nodo de utilidad simple (*UtilityNode*) que devuelve la función cuando ya se ha calculado todo.

En esta función lo que hacemos es comprobar si el nodo esta activo o no, si no esta activo en vez de calcular la utilidad modificaremos el campo *value* de la estructura poniéndolo siempre a 0, en caso de estar activo lo que haremos será calcular el valor correspondiente de la curva respecto al *float* de entrada, sumándole al valor resultante el valor contenido en el campo *mod* y cambiando con dicho resultado el campo *value* del nodo de utilidad y una vez modificado todo lo necesario el (*UtilityNode*) resultante sera el que sacaremos por *Out Utility*

- Max: Esta función cuenta con dos parámetros de entrada y uno de salida, los parámetros de entrada son de la clase *UtilityNode* son los nodos de utilidad que serán comparados y por otro lado el Out Utility es el nodo resultante de calcular cual es el que tiene el valor máximo entre los dos anteriores.

- *EvaluateUtilityComplex*: Esta función cumple la misma función que EvaluateUtility pero trabaja con *UtilityNodeComplex* por lo tanto en vez de contar con un único parámetro de entrada llamado input, este cuenta con cuatro y cada uno de ellos sirve para calcular el valor de una de las curvas del nodo el resultado de cada una de las cuatro curvas es sumado dando asi el que sera el valor que tome el campo value del nodo.

- *MaxComplex*: Al igual que el caso anterior esta función es igual a la función Max mencionada anteriormente pero trabaja con *UtilityNodeComplex* en vez de con *UtilityNode*.

5.1.3.2. Variables

Esto respecto a las funciones creadas en *MasterUtilityController*, en lo que respecta a las variables creadas en esta clase y que son necesarias tenemos:

- *Owner*: De tipo *character* que es la clase padre de todos los *characters y pawns* que podemos crear, si queremos acceder a las variables particulares de un personaje en concreto tendremos que hacer un *"cast to"* es decir hacer un casting a la clase que necesitemos. Esta variable nos da una referencia rápida del personaje para poder acceder por ejemplo a la posición de este o a sus variables como vida, enemigos marcados etc y de esta forma poderlas usar a la hora de evaluar la utilidad. Si en vez de crear un hijo de *MasterUtilityController* para trabajar, decidimos copiar la clase donde podremos cambiar el tipo de *owner* al que necesitemos en función del personaje creado, de esta forma no tendremos que hacer casting y podremos acceder directamente a sus variables.

- ID: Es una variable de tipo *string*, que nos indicará el nombre del nodo con el mayor valor de utilidad que a sido seleccionado para ejecutar la acción correspondiente.

- IDprev: Esta variable al igual que la anterior es de tipo *string* y nos indica el nombre de la utilidad que fue elegida en la ejecución anterior, de esta forma podremos hacer que una acción no se ejecute desde el inicio cada vez que se calcule la acción a realizar si se mantiene una misma acción durante varias vueltas. Hay que tener cuidado con el uso de esta variable pues podría bloquear la ejecución haciendo que el personaje se quede estático mientras no cambie la utilidad.

5.1.4. *UtilityActions*

Esta clase es una librería de funciones donde hemos implementado algunas de las acciones que podrán ser ejecutadas por las inteligencias artificiales así como las funciones necesarias para estas acciones, si el usuario necesita mas funciones serán añadidas a esta clase incrementando el catalogo de acciones que se puedan realizar. Las funciones con las que contamos actualmente son:

- MoveTo:Esta función hará que el NPC se mueva a un punto determinado. Cuenta como parámetros de entrada con un vector que actuara como destino al que nos queremos mover, y un tipo *AIController* que normalmente tomara el valor *self* en los controladores en los que sea ejecutada la acción.

- AttackTo: Esta función hace que el NPC se mueva hasta la posición de de un un enemigo y le aplique una cierta cantidad de daño. Los parámetros de entrada con los que contaremos son controller que es de tipo *AIController* e indica el controlador en el que ejecutaremos esta acción, enemy de tipo *actor* nos indica el objetivo al que atacaremos, damage de tipo *float* nos indica la cantidad de daño que haremos al objetivo y por ultimo me de tipo *actor* también nos indica el NPC que esta atacando.

- RunAway: Esta función hace que el NPC huya de un cierto personaje u objeto y sus parámetros son los mismo que los de la función anterior salvo damage que no existe en esta función y sus funciones son las mismas salvo enemy pues en este caso no atacaremos si no que huiremos de el.

- Patrol: Esta función hace que el NPC vaya de un punto a a un punto b constantemente y de forma alterna y sus parámetros son dos valores de tipo *vector* que indican los distintos puntos entre los que patrulla, un valor tipo *bool* que indicara en que ocasión va a que punto y controller que nos indicara el *AIController*.

- MinDistande: Esta función calcula la distancia mínima entre el NPC y un array de objetos de tipo *BPBotiquin*, siendo sus parámetros el array de tipo *BPBotiquin* y me de tipo *actor* que nos indica al NPC.

- Heal: Esta función busca el botiquín mas cercano para curarse, para ello llama a otras funciones que también están implementadas en *UtilityActions*. Sus parámetros de entrada son me de tipo *Actor* que hace referencia al NPC, controller de tipo *AIController* indica el controlador que esta ejecutando esta función y radius de tipo *float* indica el radio en el que busca un botiquín.

- Scan: Esta función es la que usamos para poder buscar los botiquines y sus parámetros de entrada son únicamente radius de tipo *float* que nos indica el radio de búsqueda y me de tipo *actor* que es necesario para sacar la localización del NPC. Si queremos que busque otro tipo de objetos tendremos que modificarla o crear una nueva función.

5.1.5. Curvas de Utilidad

Son *curvesFloat* cuando creemos una de estas curvas definiremos las curvas y puntos que contenga, podremos crearla como queramos, pero tenemos que tener en cuenta que intentamos normalizar los valores entre 0 y 1 de tal forma que hay que definirlas con esto en mente. Las curvas que sean usadas en los *UtilityNodeComplex* tenemos que tener en cuenta que pueden sumarse hasta cuatro valores por lo que si el valor máximo de las curvas es de uno el valor total seria cuatro y por lo tanto no siempre seriamos capaces de normalizar entre varias utilidades, por esta razón tendremos que tener en cuenta cuantas curvas usaremos para crearlas en consecuencia de tal forma que si usamos 2 curvas en un mismo nodo las definiremos para que los valores limite que puedan tener sean 0 y 0.5, lo mismo tendremos que hacer si se usan 3 o las cuatro curvas. Por lo tanto tendremos que crear las curvas en función de nuestra necesidades en cada momento.

5.2. Creación de Bots

Una vez explicadas las distintas estructuras y clases creadas para poder introducir la utilidad en Unreal Engine continuaremos explicando los pasos a seguir para poder crear una inteligencia artificial que sea capaz de usar lo explicado anteriormente, pero antes de explicarlo detalladamente en esta imagen podremos observar un diagrama en el que mostramos los pasos que tiene que ir recorriendo nuestro controlador.

Figura 5.3: Diagrama del controlador

Las fases por las que pasamos están claramente definidas:

- **Inicio de la ejecución** En Unreal Engine este inicio se hará desde un *event tick* por lo tanto perderemos la forma de bucle observada en el diagrama.

- **Evaluación de Utilidad** Aquí se evaluaran los distintos nodos determinando la utilidad de ejecutar cada uno de ellos.

- **Barrera** Encargada de bloquear el flujo de ejecución mientras no se evalúen todas las utilidades.
- **Máximo** Donde calcularemos cual es el valor máximo de utilidad de entre los distintos nodos que tenemos.
- **Seleccionar Acción** Ya calculado el máximo determinaremos a que nodo pertenece para poder determinar la acción a realizar.
- **Ejecutar Acción** Ejecutamos la acción correspondiente
- **Reinicio de variables** Reiniciamos las variables que utilizamos para bloquear el flujo en la barrera y también actualizamos IDPrev.

Una vez visto de forma general las fases por las que pasara la ejecucion de nuestro sistema de control de bots, procederemos a explicar paso por paso y de forma detallada como tendremos que crear la inteligencia artificial y junto con las otras clases relacionadas para que funcione correctamente de forma que siguiendo estos pasos no tengamos ningún problema, pudiendo actuar esta explicación como un tutorial para el usuario.

El primer paso es crear un *character* que se convertirá en el Npc que hará uso del controlador, el siguiente paso lo podemos tomar de dos maneras distintas, podemos o bien crear un hijo de la clase *MasterUtilityController* por lo tanto contaremos con las funciones y variables contenidas en esta clase pudiendo añadir las que nosotros necesitemos en el hijo, si usamos esta opción nos veremos obligados a hacer casting en los casos en los que necesitemos recurrir a las variables del NPC pues *owner* es de la clase padre *character* y por lo tanto no contara con las variables y o funciones que queramos añadir al NPC, la otra opción es copiar y usar la clase *MasterUtilityController* como si fuese una especie de platilla, haremos una copia de esta y cambiaremos el clase de la variable *owner* por la de nuestro NPC, de esta forma lo podremos usar directamente. Por otro lado en el NPC creado tendremos que cambiar el controlador que va a ser utilizado, esto lo podremos hacer en la pestaña *Pawn*, en *AI Controller Class* donde indicaremos la clase que usaremos, después de hacer esto accederemos al controlador comprobaremos que es el que queremos usar y accederemos a su variable *owner* haciendo un set y pasándole una referencia *self*. A continuación en el NPC tendremos que añadir las funciones y variables que necesitemos tales como la vida, los enemigos vistos, la modificación de la vida respecto a los daños ...

Una vez acabado ese paso todo lo que haremos sera en el controlador creado, procederé a relatar lo que hay que hacer paso a paso junto con algunas imágenes, pues una de las metas era crear un método que pudiese ser seguido paso a paso por cualquier persona y que le permita poder crear una inteligencia artificial que haga uso de la utilidad, por lo que los pasos a seguir son los siguientes:

1. Crearemos por cada acción distinta que queramos que se realice dos variables, una de tipo *UtilityNode* o *UtilityNodeComplex* y otra de tipo bool,hay que recordar que no se pueden mezclar las clases *UtilityNode* y *UtilityNodeComplex* pues no se puede calcular el máximo entre una de un tipo y otra del otro solo entre dos de la misma clase, por lo tanto tendremos que usar todas de una misma clase.

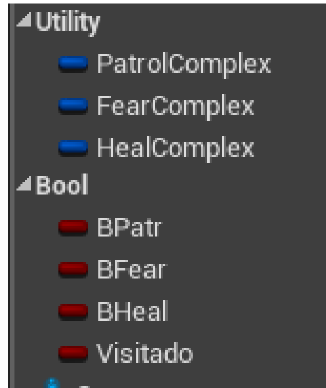


Figura 5.4: Variables necesarias

En la imagen podemos observar que seis de las variables parecen estar emparejadas dos a dos por lo tanto contamos con 3 acciones (Patrol, Heal y Fear)

2. Usaremos *EventTick* junto con un *sequence* para poder dividir el flujo, y en cada una de las ramas creadas por el *sequence* usaremos la función *EvaluateUtility* o *EvaluateUtilityComplex* según corresponda pasándole los datos necesarios un *UtilityNode* o un *UtilityNodeComplex* y las los datos que necesitemos para calcular la utilidad.

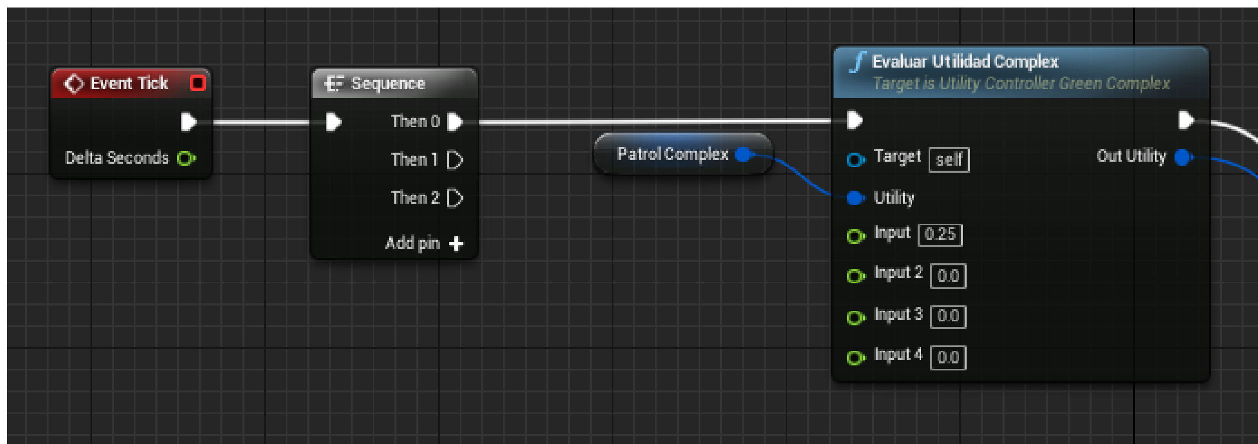


Figura 5.5: Evaluación de Utilidad

En esta imagen podemos ver lo explicado anteriormente como la función que usa es *EvaluateUtilityComplex* vemos que podemos poner cuatro posibles input y que uno esta metido directamente mediante un valor numérico. Otra forma de pasar estos valores y la que usaremos mas comúnmente es la mostrada en la siguiente imagen.

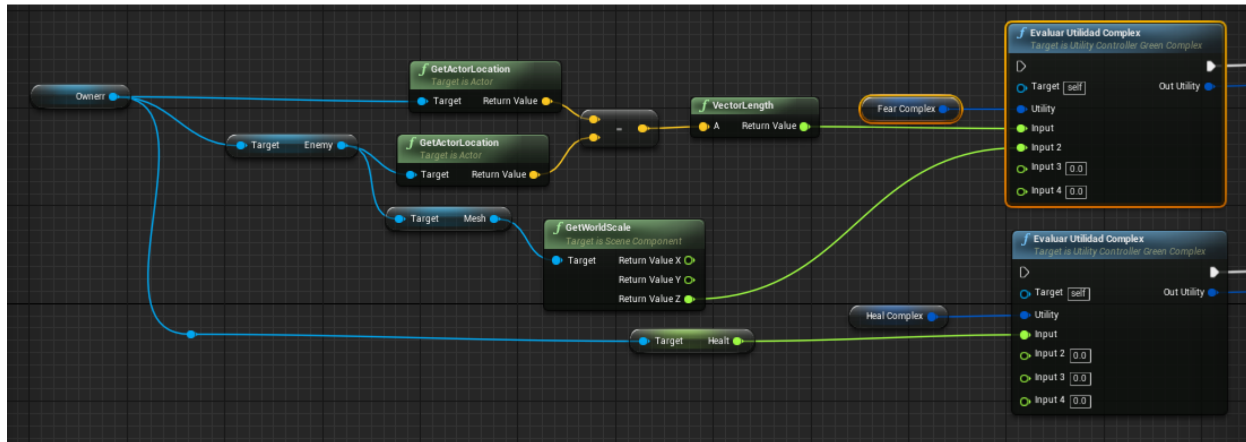


Figura 5.6: Paso de variables

Podemos observar que sacamos los datos necesarios desde la variable *owner* vinculada al NPC

3. El siguiente paso es modificar los nodos y los valores de tipo *bool* después de haber evaluado la utilidad, el valor del nodo sera tomado de la salida de la función mientras que el tipo *bool* se pondrá siempre a cierto en este paso indicando que a sido visto y evaluado.

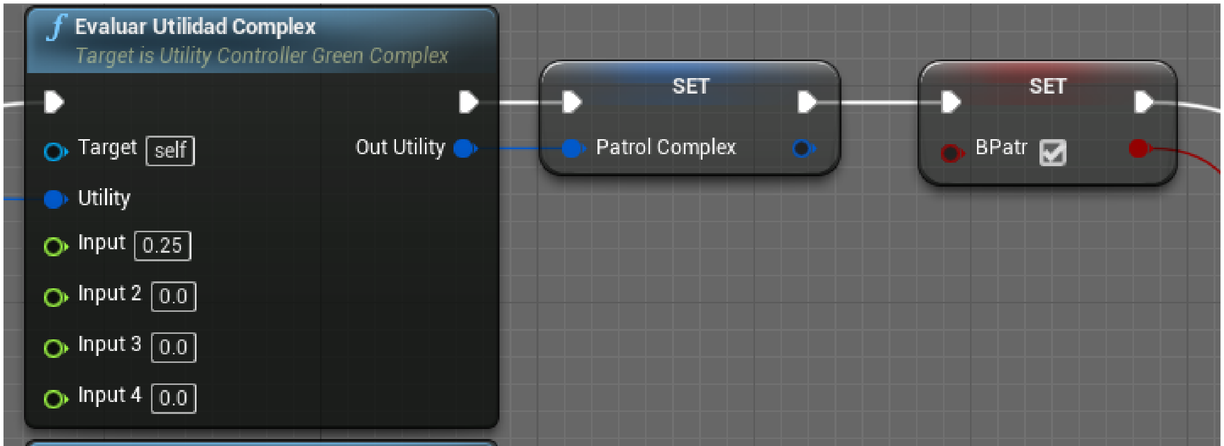


Figura 5.7: Modificación de variables

4. Continuaremos creando una barrera que bloquee el flujo hasta que todos los nodos se evalúen, esta barrera la crearemos fácilmente con un *and* de las variables de tipo *bool*

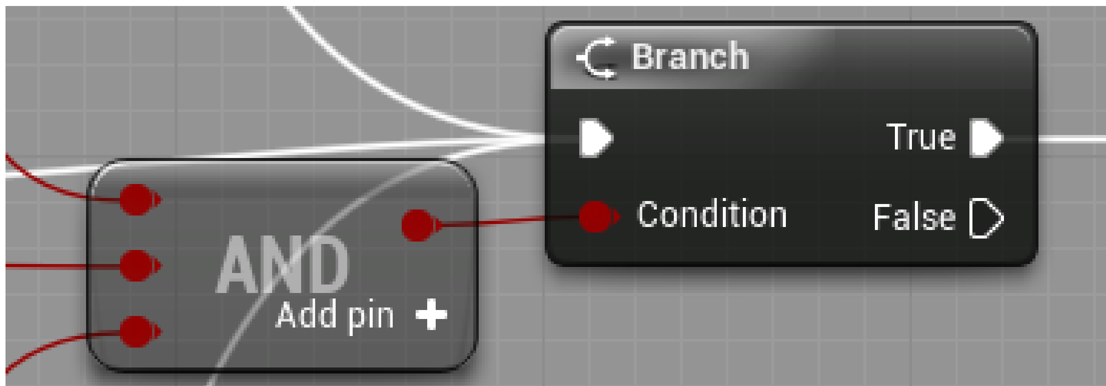


Figura 5.8: Barrera

5. Lo siguiente que haremos sera calcular el máximo entre los distintos nodos, recordando que solo podremos calcularlos dos a dos.

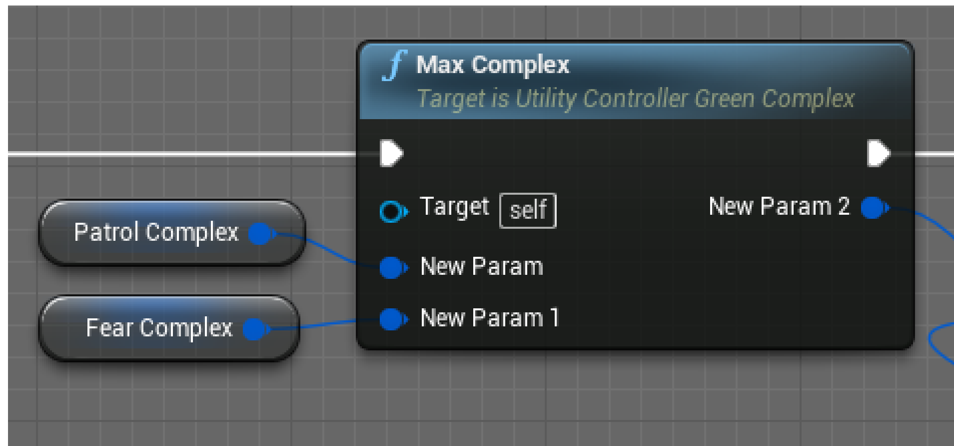


Figura 5.9: Máximo

6. A continuación modificaremos la variable ID con el valor resultante de calcular el máximo y evaluaremos con un *switch* cual es el valor de ID.

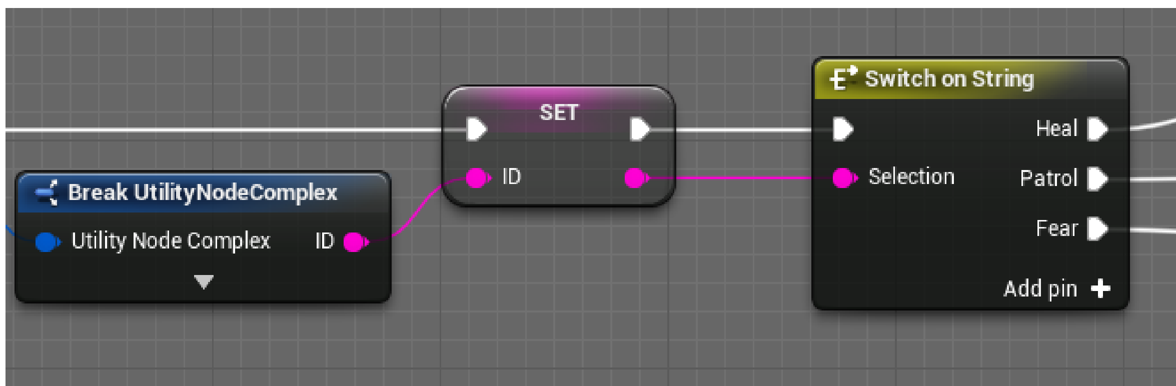


Figura 5.10: Switch de acciones

7. Lo siguiente es ejecutar las acciones correspondiente al valor que contenga ID, pasando los valores necesarios a estas funciones.

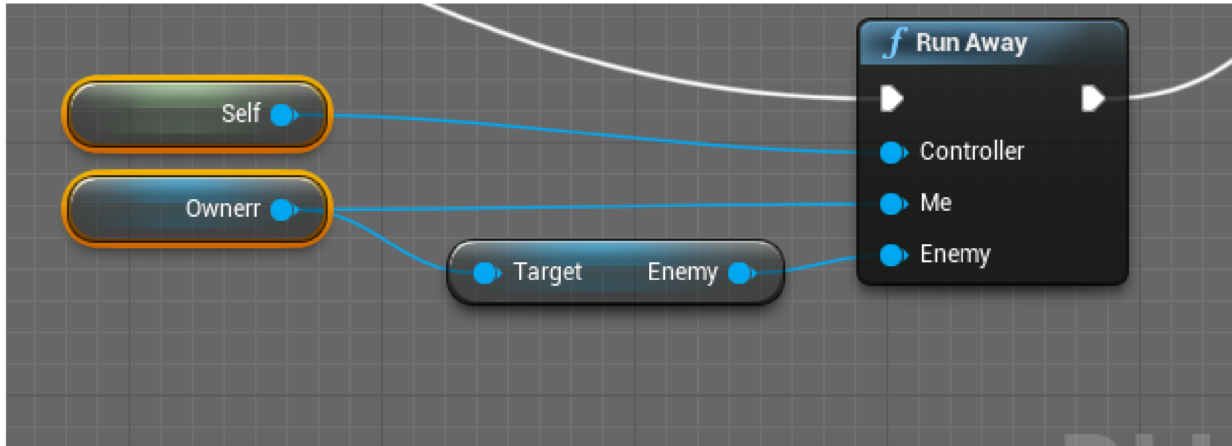


Figura 5.11: Ejecución de acciones

En la imagen podemos ver que la acción a ejecutar es *RunAway* para la que necesitamos saber cual es el controlador, el valor de bot (*Owner*) y cual es el enemigo del que huiremos.

8. El ultimo paso es poner a falso todos los valores de tipo *bool* y cambiar el valor de IDPrev que indica el valor de ID en la anterior ejecución por el valor de ID.

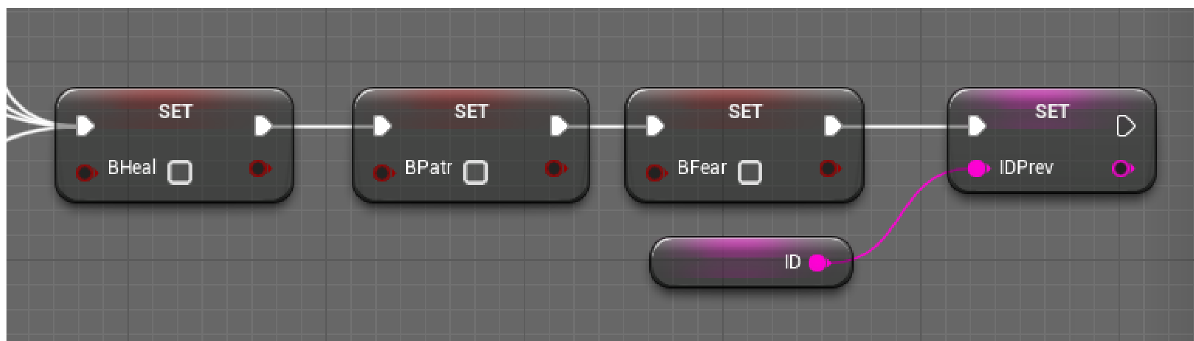


Figura 5.12: Reinicio de variables

Siguiendo este método seremos capaces de crear una inteligencia artificial que sea capaz de hacer uso de la utilidad en Unreal Engine.

5.2.1. Métodos Alternativos

A parte de la forma anteriormente explicada, ciertos recursos creados pueden ser usados de otras formas añadiendo funcionalidades como es el caso de la variable IDPrev y de los nodos de utilidad.

El IDPrev esta pensado para usarse de tal forma que si coinciden los valores de ID e IDPrev no se ejecute la acción que fuese a ejecutarse, en algunas ocasiones es necesario pues nos interesara acabar la ejecución de una acción antes de volverla a ejecutarla en vez de que se este ejecutando constantemente hasta que cambie el valor de ID.

Respecto a los nodos otra forma de usarlos es utilizarlos para crear lo que he denominado "ámbitos de acción"de tal forma que no hace falta que un nodo este vinculado a una acción, si no que estará vinculado a un ámbito, esta forma se usa para evitar que se tengan que calcular las utilidades de todos los nodos. A parte de los nodos usados para ejecutar las acciones crearemos nodos para distintos ámbitos por ejemplo: combate, interacción, supervivencia ..., se calculara la utilidad de meterse en estos ámbitos y el que tenga mayor utilidad sera en el que nos metamos a evaluar las utilidades de las acciones que pertenezcan a ese ámbito. Siguiendo el ejemplo anterior si se metiese en el ámbito combate podría evaluar la utilidad de atacar a distancia o a meleé y así no tener que evaluar si tiene que curarse o huir, de esta forma se ira creando una red con los distintos ámbitos y acciones que podremos ampliar tanto como queramos o necesitemos. Es recomendable usar los ámbitos cuando la cantidad de nodos que queramos usar en la inteligencia artificial sea muy grande y fácilmente clasificable en los distintos ámbitos dando lugar a ámbitos con mas de un nodo contenidos en él, quedando de la forma mostrada en el siguiente diagrama.

Figura 5.13: Diagrama usando ámbitos

Otra de las funciones que ha resultado de como se ha diseñado *MasterUtilityController* es la posible utilización de arboles dentro del controlador, dando la oportunidad de poder combinar la utilización de árboles de comportamiento con la posibilidad de usar la utilidad, esto se debe a que *MasterUtilityController* un hijo de *AIController* que es donde se usan comúnmente los arboles de comportamiento, por lo tanto el árbol de comportamiento puede ser llamado como si fuese una acción mas.

Capítulo 6

Pruebas y resultados

Para comprobar la utilidad y la facilidad de poder aplicar esta funcionalidad a la hora de crear una IA en Unreal Engine hemos decidido hacer dos tipos de pruebas, por una lado pruebas que permitan visualizar ejemplos de inteligencias artificiales creada mediante el sistema de control de bots que hemos creado, viendo las diferencias y los distintos comportamientos y por otro lado pruebas en las que hacemos a los usuarios visualizar un tutorial para comprobar después de verlo el grado de confianza que tendrían al crear la inteligencia artificial para un bot.

6.1. Pruebas internas

Para las pruebas internas hemos creado tres mapas distintos en los que podremos observar distintos aspectos de las inteligencias artificiales creadas y distintos métodos.

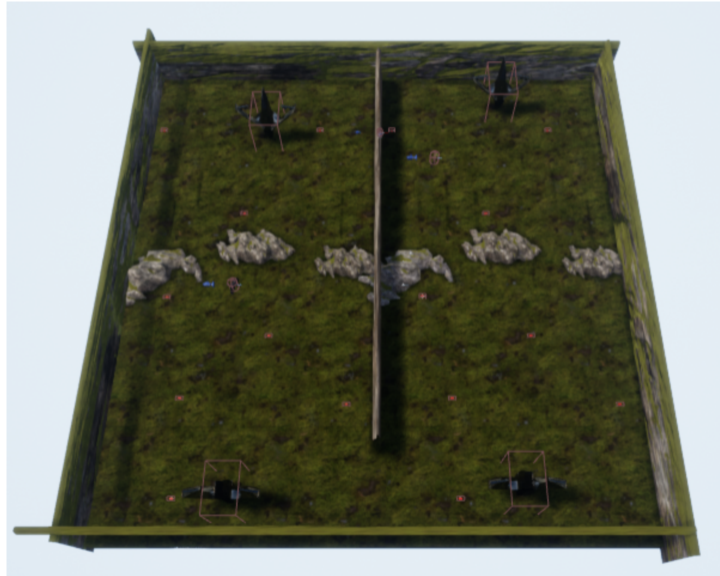


Figura 6.1: Mapa con uso de UtilityNodeComplex

6.1.1. MapUtilityComplex

En este mapa compararemos dos inteligencias artificiales que hacen uso de la utilidad utilizando los nodos complejos que permiten el uso de varias curvas de utilidad en un mismo nodo, los dos tienen comportamientos similares que solo se diferencian en las curvas de utilidad usadas y el numero de variables que utiliza, para ver las diferencias explicare los distintos comportamientos que tiene cada NPC.

6.1.1.1. Green

Este Bot usa el controlador *UtilityControllerGreenComplex*, consta de tres *UtilityNodes-Complex*, siendo este el nodo de utilidad que nos permite el uso de varias curvas para calcular la utilidad, las utilidades implementadas son *fear, heal y patrol.* Los comportamientos de estas utilidades son las siguientes:

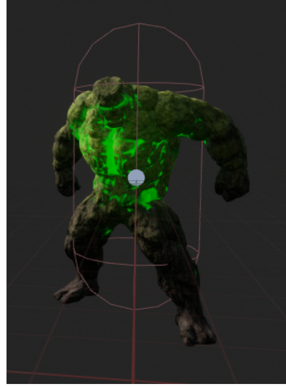


Figura 6.2: NPC Green

- Patrol: La curva usada es una recta constante, por lo tanto da igual el valor que le pongamos como entrada en *EvaluateUtilityComplex* porque mantendrá el mismo valor en todo momento.

- Fear: En este nodo haremos uso de dos curvas de utilidad, por lo tanto la utilidad total sera la suma de los valores resultantes de calcular la utilidad en estas dos curvas, estas curvas evaluarán la distancia entre el bot y el objeto de su miedo y por otro lado evaluará la altura del enemigo al que teme. El comportamiento definido para este NPC es que si ve a nuestro personaje no pasara nada pero cuando nos vea y ademas seamos grandes en función de la distancia entre los dos tendera a alejarse de nosotros o pararse y no seguir avanzando en caso de estar en el camino del NPC.

- Heal: Esta utilidad indicara la necesidad de un NPC de curarse y solo hemos usado una curva de utilidad que utilizara la variable Health para calcular lo útil que es para nosotros curarnos en el momento de evaluar la utilidad. En este NPC hemos utilizado una curva que provoca que cuando es mínimamente herido recurra a buscar un botiquín.

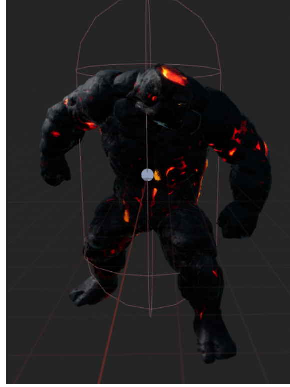


Figura 6.3: NPC Red

6.1.1.2. Red

Este Bot usa el controlador *UtilityControllerRedComplex*, consta de tres *UtilityNodes-Complex*, siendo este el nodo de utilidad que nos permite el uso de varias curvas para calcular la utilidad, las utilidades implementadas son *fear, heal y patrol.* Los comportamientos de estas utilidades son las siguientes:

- Patrol: La curva usada es una recta constante por lo tanto da igual cual sea el valor del input introducido en el *EvaluateUtilityComplex*, aun asi hemos decidido ponerle un valor usado para calcular la utilidad es constante.

- Fear: En este casos al igual que en el NPC anterior se asusta de nosotros pero en este caso no evaluara la altura, únicamente tomara como referencia las distancia entre en NPC y nosotros.

- Heal:La función de esta utilidad es la misma que en el bot anterior aunque la diferencia es que usa una curva diferente y por lo tanto su comportamiento también es diferente del anterior, siendo esta una inteligencia artificial que solo recurrirá a curarse cuando su vida corra realmente peligro contando con muy poca vida.

6.1.2. MapUtilitySimple

En este mapa compararemos dos inteligencias artificiales que hacen uso de la utilidad utilizando los nodos simples que permiten el uso de una única curva de utilidad por nodo, los dos tienen comportamientos similares que solo se diferencian en las curvas de utilidad usadas, para ver las diferencias explicare los distintos comportamientos que tiene cada NPC.

6.1.2.1. GreenSimple

Este NPC usa el controlador *UtilityControllerGreenSimple*, consta de tres *UtilityNodes*, siendo este el nodo de utilidad que usa una única curva de utilidad para calcular la utilidad, las utilidades implementadas son fear, heal y patrol. Los comportamientos de estas utilidades son las siguientes:

- Patrol: La curva usada es una recta constante por lo tanto en cualquier valor mantendrá el mismo valor, aun así en la llamada a la función *EvaluateUtility* el valor usado para calcular la utilidad es constante.

- Fear: En este nodo haremos lo que hace es comprobar la distancia al al objeto de su miedo para determinar si tiene que huir o no de el, de tal forma que cuanto mas se aproxima al enemigo mayor es la utilidad de huir.

- Heal: Esta utilidad indicara la necesidad de un NPC de curarse usando la variable Health para calcular lo útil que es para nosotros curarnos en el momento de evaluar la utilidad. En este NPC hemos utilizado una curva que provoca que cuando es minimamente herido recurra a buscar un botiquín.

6.1.2.2. RedSimple

Este Bot usa el controlador *UtilityControllerRedSimple*, consta de tres *UtilityNodesComplex*, siendo este el nodo de utilidad que nos permite el uso de varias curvas para calcular la utilidad, las utilidades implementadas son fear, heal y patrol. Los comportamientos de estas utilidades son las siguientes:

- Patrol: La curva usada es una recta constante por lo tanto en cualquier valor mantendrá el mismo valor, aun así en la llamada a la función *EvaluateUtilitySimple* el valor usado para calcular la utilidad es constante.
- Fear: Esta utilidad se comporta exactamente igual que la anterior salvo que es capaz de acercarse mas al objeto de su miedo.
- Heal:La función de esta utilidad es la misma que en el NPC anterior aunque la diferencia es que usa una curva diferente y por lo tanto solo recurrirá a curarse cuando su vida corra realmente peligro contando con muy poca vida.

6.1.3. Utility VS Tree

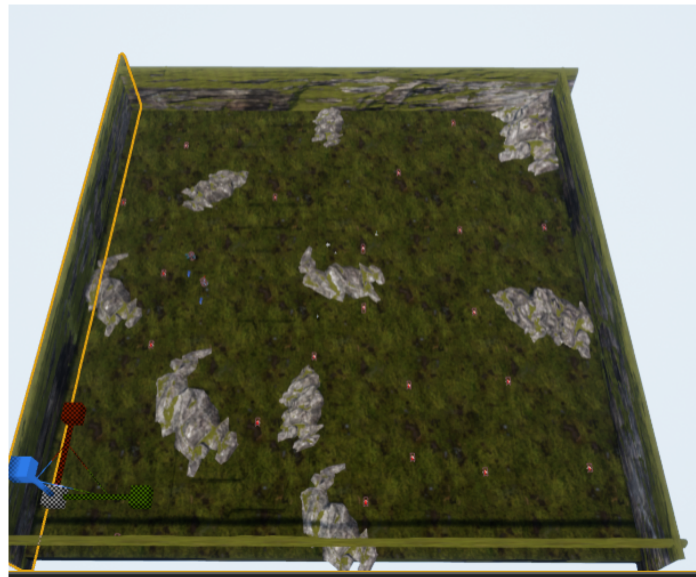


Figura 6.4: Mapa Utility VS Behavior Tree

En este mapa compararemos dos inteligencias artificiales que intentan hacer lo mismo pero que han sido implementadas mediante dos métodos diferentes, una con arboles de comportamiento y otra mediante nuestro sistema *UtilityNetwork*. El mapa únicamente tiene rocas y botiquines, de tal forma que los NPC rondaran por el mapa cogiendo los botiquines pero huirán de nosotros o se pararan al acercarnos. De esta forma podremos observar que cuando se usa UtilityNetwork el resultado es el esperado, pero cuando usamos árboles de comportamiento aunque la parte de coger los botiquines puede funcionar bien en algunos casos se atasca y no es capaz de progresar cuando no puede acceder al botiquín que tenga como objetivo, por otro lado no es capaz de huir de nosotros cuando nos acercamos porque esta ejecutando otra acción y cuando tiene que realizar la acción de huir de nosotros puede que no estemos en rango para que huya.

6.2. Pruebas con usuarios

En lo que respecta a las pruebas con usuarios, lo que he hecho a sido mostrarle a los usuarios un tutorial explicativo sobre como crear IAs con nuestro sistema de control de bots y una vez que lo han visto nos han rellenado un formulario con algunas preguntas. Algunas de las preguntas eran simplemente de control como la edad y el sexo de los usuarios el resto de preguntas estaban orientadas a dar a conocer los conocimientos previos y las dificultades que veían o bien en el método en si o bien en comparación con las herramientas ya creadas en Unreal Engine, otra pregunta importante a sido si después de ver el tutorial se veían preparados para intentar crear una IA, en caso de responder afirmativamente les di la posibilidad de intentarlo. Para hacerse una mejor idea de las preguntas podemos encontrar el formulario en la pagina:

6.2.1. Análisis

La población de la encuesta puede parecer reducida dado que solo se ha realizado a 5 personas, esto se debe a que aunque nuestro proyecto plantea como uno de los objetivos el poder ser utilizado por personas que acaban de iniciarse en el uso de Unreal Engine y carecen de conocimientos o sus conocimientos del motor son muy básicos. Por esta razón las encuestas se hicieron a gente relacionada con el desarrollo de videojuegos, tanto a programadores como a algún modelador. Pero a pesar de que la población de la encuesta es solo de 5 personas podemos empezar a analizar las respuestas de los usuarios y observar que con solo 5 personas ya podemos observar algunas coincidencias dentro de los datos obtenidos en los formularios.

Las primeras preguntas son de control no aportan información sobre el proyecto sino de la población que ha respondido al formulario en este caso las preguntas han sido referentes al sexo y la edad de los usuarios y los resultados son los siguientes:

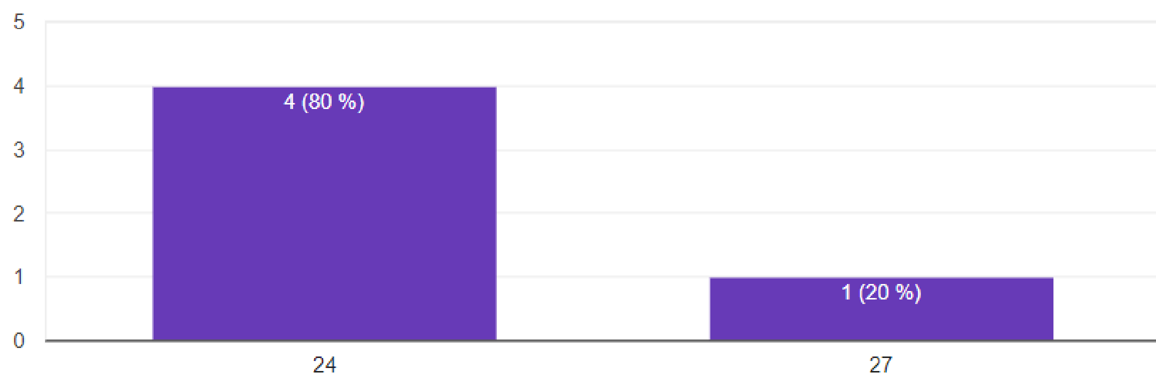


Figura 6.5: 2ª Pregunta

En lo referente al sexo de los usuarios en este caso han sido todos varones y respecto a la edad todos los encuestado tienen una edad de 24 años excepto uno de ellos.

En la siguiente pregunta ya empezamos a indagar en los conocimientos previos de la persona en cuestión, en este caso si contaban con experiencia previa usando Unreal Engine, dando como resultado:

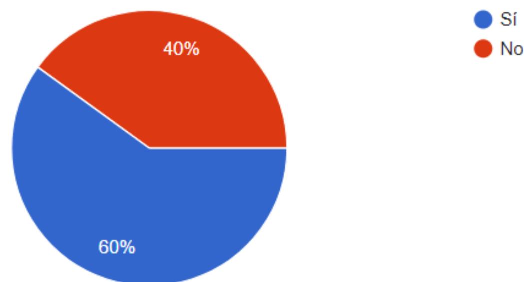


Figura 6.6: 3ª Pregunta

Una mayoría con experiencia previa a excepción de dos usuarios.

Una vez que han visto el tutorial les preguntamos sobre su opinión respecto a lo que han visto

¿Que te ha parecido Utility Network?

5 respuestas

Permite hacer AIs con respuestas complejas facilmente

Es interesante, y tiene muchos usos

Sencillo si se van siguiendo los pasos y tienes algunos conocimientos previos

Parece una utilidad muy interesante para crear IAs sofisticadas, que ya de por si en Unreal tiene bastantes complicaciones. El sistema de utilidad parece bastante interesante.

Parece interesante, tendría que ponerme más con unreal para entenderlo a fondo, porque he trabajado con IAs pero la implementación a fondo de esta herramienta se me escapa un poco. El concepto lo pillo y parece bastante útil.

Figura 6.7: 4ª Pregunta

Destacan sobre todo que parece sencillo e interesante, se puede deducir que piensan que la utilidad es un concepto interesante y que aporta algo que no tenia Unreal Engine o que hace mas fácil algunas funciones de Unreal Engine. También nos interesaba saber si verían posible el crear una inteligencia artificial después de ver el tutorial dando como resultado:

¿Crees que con la información aportada serias capaz de crear una IA?

5 respuestas

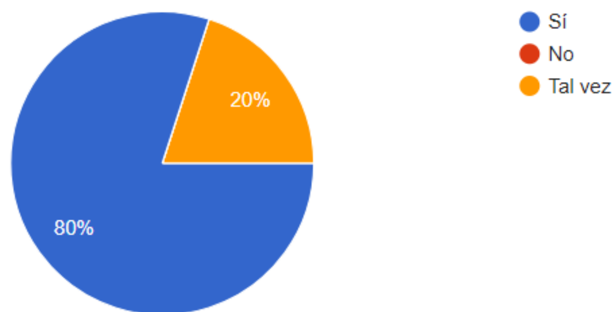


Figura 6.8: 5ª pregunta

A parte del formulario hablamos un poco con los usuarios y en esta pregunta nos transmitieron que en el tutorial daba la sensación de estar muy sistematizado y que en rasgos generales veían posible el crear una inteligencia artificial después de ver el tutorial, también comentaron que en caso de ver alguna dificultad creían que siguiendo el tutorial de forma paralela paso por paso seria n capaces de hacerlo.

Como uno de los puntos era la comparativa con las herramientas de Unreal Engine, en este caso los árboles de comportamiento, les preguntamos si tenían experiencia previa usándolos resultando de la siguiente manera:

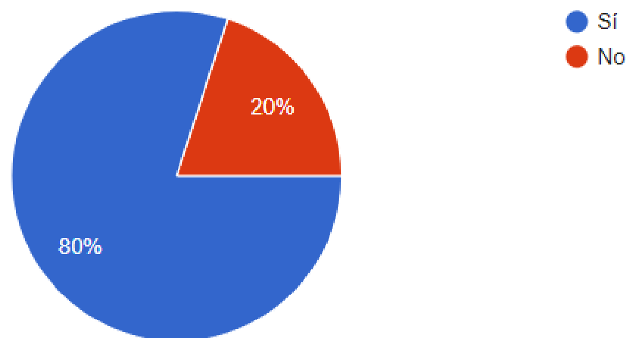


Figura 6.9: 6ª pregunta

En su mayoría habían trabajado con arboles de comportamiento pero indicaban que no tenían mucha experiencia, al menos en Unreal Engine dado que la curva de aprendizaje se hacia muy dura y acababan usando lo justo para crear inteligencias artificiales simples.

La siguiente pregunta obligada era la comparación entre arboles de comportamiento y *UtilityNetwork* siendo estas las respuestas:

Si has trabajado con arboles de comportamiento ¿Que te parece UtilityNetwork en comparación?

4 respuestas

Cómodo e intuitivo.

Como hay que crear menos clases para trabajar, es mas sencillo saber donde va cada cosa y donde hay que hacer las modificaciones

Parece bastante interesante. Con algo de pulido puede quedar una herramienta muy chula.

PArece más complejo y permite crear situación más enrevesadas de lo que te permiten en general los árboles.

Figura 6.10: 7ª pregunta

Nos indicaban que parecía fácil de usar pero en algún caso indicaron que faltaría pulir algunos detalles. Por otro lado una de las respuestas indicaba que parecía complejo, al ser una respuesta contraria al resto preguntamos para saber mas detalles y nos dio a entender que la complejidad no la veía en el uso si no en la idea sobre la que se planteaba pues ademas permitía comportamiento complejos.

Por ultimo nos interesaba saber que veían mas complicado sin comparativas, simplemente que dificultades encontraban después de ver el tutorial.

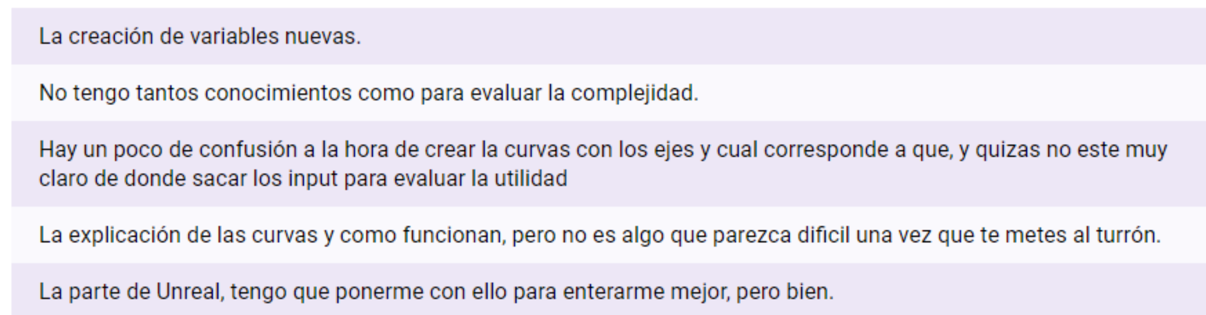


Figura 6.11: 8ª pregunta

Los puntos mas repetidos fueron las curvas y las variables, creemos que esto se debe a una falta de explicación en el tutorial por nuestra parte. Las curvas las veían complicadas porque no se explicaba en el tutorial como hacerlas y al ser algo que se creaba fuera de los *blueprint* no se veía muy claro. Otro de los puntos complicado que observaban era el paso de variables como input para calcular la utilidad, creemos que al igual que en caso anterior tendríamos que haber explicado mejor esta parte en el tutorial.

Con estos resultado podemos ser capaces de sacar algunas conclusiones respecto al sistema de control de bots, tanto propias como respecto a la percepción que tienen los usuarios.

Capítulo 7

Conclusiones

Tras las diferentes pruebas realizadas hemos comprobado que realmente el trabajo realizado añade una nueva funcionalidad a Unreal Engine para el desarrollo de inteligencias artificiales de una forma satisfactoria. Dando la opción de elegir otro método diferente mas allá de los arboles de comportamiento incluidos en Unreal Engine, siendo capaz por otro lado de añadirles un comportamiento complejo de forma mas sencilla e intuitiva que los arboles de comportamiento que se construyen normalmente con el motor. Por ejemplo, no teniendo que modificar todas las *task* (hojas del árbol, acciones) creadas para reaccionar a distintos comportamientos mientras se ejecuta una acción.

También es cierto que aunque, siguiendo los pasos propuestos en la creación de inteligencias artificiales, una persona con pocos, o ningún, conocimiento puede llegar a crear una, se vera limitado a los recursos creados, no pudiendo acceder a nuevas acciones al no ser capaz de desarrollarlas. Sin embargo una persona que cuente con conocimientos previos podrá crear sus propias acciones, e incluso usar los recursos proporcionados de formas en las que no se había pensado previamente, llevando de esta forma a mejorar el trabajo realizado.

Tras describir las sensaciones que tenemos analizando las pruebas realizadas, compararemos los objetivos que inicialmente teníamos como metas, si realmente hemos sido capaces de cumplirlos y en que medida.

Los objetivos eran los siguiente:

1. Conseguir crear una clase que nos permita definir el nodo que contara con los campos necesarios para que pueda ser calculada la utilidad.

2. Conseguir que la clase anterior pueda evaluar la utilidad tanto con una única variable como con varias.

3. Crear otra clase que nos permita trabajar con los nodos creados, de cara a que sean evaluados en su totalidad y nos devuelva la opción a tomar.

4. Permitir la creación de gráficas que nos permita poder calcular la puntuación que se consigue en un nodo en función de ciertas variables.

5. Proporcionar una serie de acciones básicas que podrá ser aumentada por el usuario.

6. Crear mapas donde poder visualizar los distintos comportamientos.

Y respecto a esos objetivos los resultados han sido:

1. Realmente se ha conseguido crear este nodo, están definidos como las estructuras de datos *UtilityNode* y *UtilityNodeComplex*. Tiene todos los campos necesarios para poder calcular la utilidad, por lo tanto este objetivo ha sido totalmente cubierto.

2. Este objetivo se ha completado también pero no como se planteó inicialmente, pues tenemos una función para comprobar la utilidad de los nodos mencionados anteriormente pero esta función no esta implementada en la estructura.

3. La clase creada para gestionar el tratamiento de los nodos también fue creada con éxito y cumple su función. Se creo como un hijo de la clase *AIController* y dentro de ella definimos las funciones necesarias para trabajar con los nodos, por lo tanto este objetivo también ha sido cumplido satisfactoriamente.

4. Este objetivo también se cumplió pues aunque inicialmente no lo sabíamos Unreal Engine contaba con la posibilidad de crear curvas como si fuesen otro *blueprint* mas, sin tener que recurrir a *timelines* para dibujar las gráficas.

5. La forma en la que conseguimos alcanzar este objetivo es ligeramente diferente a como se pensó inicialmente pero la solución final nos parece mas satisfactoria pues se crearon estas acciones en una librería, fuera de las clases anteriormente mencionadas, de forma que no es necesarios que el usuario modifique la clase padre *MasterUtilityController*, para añadir acciones simplemente tendrá que acceder a la librería y crear ahí la función correspondiente.

6. También se ha cumplido este objetivo, los mapas están creados y se pueden ver y compara los distintos comportamientos, aunque en este caso no se ha conseguido llegar al objetivo de una forma completamente satisfactoria, pues algunos comportamientos no se pueden apreciar tan claramente como nos gustaría

Para comprobar las dificultades de la gente en el uso de esta herramienta les hicimos rellenar un formulario con algunas preguntas sobre las dificultades encontradas y ha la hora de analizar las encuestas también podemos observar que a la gente le resulta relativamente sencillo crear inteligencias artificiales con este método, únicamente viendo un tutorial y siguiendo sus pasos mientras crean la inteligencia artificial, por otro lado algunas de las dificultades mencionadas están relacionadas con los conocimientos previos en Unreal Engine, o por alguna explicación del tutorial que no ha sido del todo entendida por los usuarios.

Por estas razones creo que la realización de este trabajo realmente tiene sentido para los creadores de contenido que utilicen Unreal Engine, tanto para los que tienen conocimientos previos como para aquellos que acaban de iniciarse en el uso de Unreal Engine, a pesar de esto también veo necesario el uso continuado de esta herramienta y el reporte de errores que viene acompañado de ese uso continuado para poder pulir o arreglar estos errores que surjan y de los que inicialmente no se es consciente y conseguir que la herramienta siga mejorando en un futuro, que es una de las razones por las que el uso de arboles de comportamiento a ido mejorando hasta llegar a lo que es hoy en día.

7.1. Trabajo Futuro

En lo que respecta al desarrollo futuro de esta herramienta, mas allá de pulir los pequeños errores que puedan surgir, veo tres caminos en los que se podría enfocar par mejorar la herramienta.

La creación de una interfaz diferenciada del resto de forma que al igual que cuando creamos los arboles de comportamiento cambia ligeramente la interfaz adaptándose al uso de los arboles de comportamiento, de la misma forma al crear una clase de tipo *UtilityController* a la hora de trabajar en ella podría cambiar la interfaz facilitando su uso ayudando tanto a aquellos con conocimientos como haciendo mas fácil su uso para aquellos que están empezando. Este camino supone tocar el código en el núcleo del motor gráfico lo cual no es un trabajo fácil cuando hablamos de Unreal Engine pues en algunos casos es difícil encontrar la documentación de algunos aspectos necesarios para este trabajo.

Otro posible camino de mejora de esta herramienta es el uso dual de árboles de comportamiento y de la utilidad, aunque no era un objetivo ni algo pensado consciente mente por la forma en la que decidí diseñar esta herramienta es posible utilizar arboles de comportamiento como una posible acción sujeta a una cierta utilidad. De forma que una acción podría ser ejecutar un árbol previamente creado, diseñado para realizar unas acciones determinadas se podría por ejemplo fusionar el uso de los ámbitos explicados anteriormente con los arboles de comportamiento de forma que cada ámbito ejecute una árbol de comportamiento, una de las cosas a estudiar si se sigue este camino es el rendimiento pues es posible que el uso de la utilidad no compense el tener que usar varios arboles de comportamiento en un mismo controlador, es un camino interesante que intenta fusionar aun mas esta herramienta con las ya proporcionadas por Unreal Engine.

El ultimo camino posible que veo de mejora es la inclusión de la teoría de la probabilidad, la teoría de la utilidad es una de las dos teorías usadas en la teoría de decisión, la otra es la teoría de la probabilidad, de esta forma si somos capaces de incluir esta ultima junto con el uso de la utilidad podríamos crear inteligencias artificiales que hagan uso de la teoría de decisión.

Al incluir el uso de la probabilidad junto con el uso de la utilidad, nuestras inteligencias artificiales no solo reaccionarían a estímulos externos reaccionando en consecuencia, si no que las acciones a realizar se elegirían en función de utilidad de las posibles consecuencias de dichas acciones, es decir que si al dispara a un enemigo hay pocas posibilidades de matarle, que seria la consecuencia que tendría mayor utilidad, que herirle que cuenta con mas posibilidades, pero lanzar una granada tiene muchas posibilidades de matar al enemigo, la acción tomada seria lanzar la granada en la mayoría de los casos y si no contamos con ninguna otra variable que intervenga, de esta forma las inteligencias artificiales en vez de reaccionar a estímulos directamente, tienen en cuenta la utilidad de lo que pueda pasar es decir tienen en cuenta los posibles futuros derivados de su acción.

Estos son los tres caminos en los que creo que se puede mejorar, pero al ser una herramienta nueva y no basarse en otra anterior, aun esta abierta a muchas posibilidades por lo tanto es posible que alguien que la empiece a usar desde cero sea capaz de ver otros posibles caminos, que no me había planteado inicialmente por lo tanto creo que lo mas importante a futuro es ser capaz de usarla y recibir *feedback* para poder mejorarla, arreglarla y poder descubrir todas sus futuras posibilidades.

Referencias

Desjardins, J. (2017). *How video games became a $100 billion industry.* https://www.businessinsider.com/the-history-and-evolution-of-the-video-games-market-2017-1?IR=T.

Día internacional del gamer, una industria con 15,8 millones de jugadores habituales en españa. (2018). http://www.europapress.es/portaltic/videojuegos/noticia-dia-internacional-gamer-industria-158-millones-jugadores-habituales-espana-20180829180358.html.

Goldberg. (2011). *All your base are belong to us: How 50 years of videogames conquered pop culture.* Three Rivers Press. Descargado de `https://books.google.es/books?id=CKQXVbY19PQC`

Kent, S. (2001). *The ultimate history of video games: From pong to pokémon and beyond : the story behind the craze that touched our lives and changed the world.* Prima Pub. Descargado de `https://books.google.es/books?id=C2MH05ogU9oC`

McCarthy, J. (1956). (Conferencia de Dartmouth de 1956)

OxfordDictionaries. (s.f.). https://es.oxforddictionaries.com/definicion/inteligencia.

Sagredo, I. (s.f.). *Aplicación de técnicas de aprendizaje automático supervisables por el diseñador al desarrollo de agentes inteligentes en videojuegos* (Tesis Doctoral no publicada). Universidad Complutense de Madrid.

Tassi, P. (2014). *Are video games becoming too focused on multiplayer?* https://www.forbes.com/sites/insertcoin/2014/11/10/are-video-games-becoming-too-focused-on-multiplayer.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460.

von Neumann J. Morgenstern O. (1944). *The theory of games and economic behavior.* Princeton University Press.

Apéndice A

Introduction

Video games have now become something that goes beyond being simple computer applications for entertainment. Although giving them a precise definition is complex, among the software that is traditionally cited when talking about *first video games in history* we found the missile simulator that Goldsmith and Mann programmed in 1947, the classic NIM game that Ferranti created in 1951, and noughts and crosses that appeared in the Douglas thesis in 1952 (Goldberg, 2011; Kent, 2001).

Despite the uncertainty of knowing exactly which one is really the first video game, we know that the appearance of the first idea of video game corresponds to the year 1951 and was introduced by Ralph H. Baer when he suggested the idea of making an interactive game for a television after a commission (Kent, 2001).

After, in the 70s, we can talk about the first revolution of video games in the American society with the appearance of the Magnavox Odyssey, whose main creator was Ralph H.Baer, and the game Pong produced by Atari, considered by many as the first commercial video game production, being something different than an experimental project. The idea of video game has been changing over the years, adapting to the needs of each time and reaching what it is today: the largest entertainment product consumed in the world (Desjardins, 2017).

The growing demand for video games, and above all, the demand of users, has led the video game industry to constantly improve the main aspects of these: gameplay, visual characteristics, story and artificial intelligence, among others. This evolution has led to this industry to be able to invoice figures that are around 100000 million dollars (in 2017).

We can distinguish multiple aspects that are necessary to establish when creating a video game from scratch, such as design, mechanics, context, graphics, target audience, etc. Some of these aspects have varied their level of importance through time.

In the current era, for example, it has become a very important aspect that the game is oriented to be multiplayer, (Tassi, 2014) reaching the point that in many video game productions have tended to reduce the time devoted to development of the modes known as individual campaign, or *singleplayer* (with respect to previous games of the same sagas) to give more importance to the multiplayer modes (or *multiplayer*). A clear example of this case is found in the genre of *shooter* (or action games with gun shots), or sports games; genres that today we do not conceive without a multiplayer mode.

This fact has caused that the creation of artificial intelligences in games is an aspect that has been gaining more and more importance. This is because the habitual players, once accustomed to playing with and against other human players, feel more complicated their immersion in the universe of the game if they happen to face again automatic players, with non-human or illogical reactions.

It also happens that artificial agents are used to control non-human characters, so a human reaction in a character of this type would also be illogical. The objective, therefore, is to be able to create agents with artificial intelligences that have a satisfactory relationship with the context of the game and are able to react to the environment in the way the developer wishes.

For these reasons artificial intelligence in video games is an increasingly important aspect that must continue to evolve and implement the different theories that arise in this area of information technology such as decision theory, utility theory, trees of behavior, Bayesian networks ...

To be able to work in the inclusion of these theories and other aspects in the video games we need to have development tools that are integrated and work properly with game engines. These modern engines like Unreal Engine or Unity, have almost come to monopolize the development of video games in the last 10 years, but long before its appearance, the game engines were written in a single entity with certain methodologies. For example, a game of Atari 2600 should be designed with a methodology *bottom up* to get the most out of the system.

The fast advance of the recreational ones made rethink this method of development thus passing to use a base of rules *hard-coded* with a small number of levels and graphical data. Thus arose the first 2D game engines developed and used by companies.

The rise of 3D in 1990 with games like Doom and Quake already posed the division between " game content " (specific rules of the game and data) and " game assets " (collision detection and game entity). In 1998 and with the idea of separation between engine and content emerged Unreal Engine [1] , the use of these engines has significantly helped both small developers and global companies in the sector, thus helping the development of video games in an efficient and fast way.

[1] important to say that Unreal Engine was released a few years ago, thus making it a graphic engine that facilitates its use both in education and in other fields due to its easy access

As I explained earlier, the recent growth of multiplayer games in the industry has made the importance of artificial intelligence in games increased, because when we play against the game environment we need more human reactions in such a way that they do not seem strange to us. the behaviors of the *NPC's*, making sure you do not miss the immersive experience of the games. This growing demand for an immersive gaming experience and intelligent interactions with *NPC's* focuses the focus on the techniques used by developers to define artificial intelligence. A very widespread method within video game development are the behavior trees (*Behavior trees*), a mathematical model (hierarchical) of execution that defines the different finite paths that can be executed if a certain criterion is met or sequentially depending of the case. The demand for more complex intelligent agents makes this technique not so effective, in this context utility arises as an extrapolation of the one already used in economics. The main idea of this new paradigm is that every possible action or state given by a model can be described with a parameter called "utility" that describes how important that action is in a certain context and for a certain character. This idea is intended to address a large number of existing problems in BT and that can be solved in this way.

There are various tools for the development of artificial intelligence in game engines to provide agents with different behaviors, however, most of these tools are paid, and the difficulty curve to learn to use them satisfactorily is usually very pronounced. Thus, the assessment of its use by developers in productions of a discrete or even medium size is very complex, since it is necessary to evaluate if it really is worth undertaking an economic outlay in expensive tools to master.

This is a problem, both for small developers and for large companies that have to dedicate time and effort in the creation of tools that, on the other hand, due to the time and effort involved in their development, do not usually release later. Unreal Engine footnote Unreal Engine does not have a base with any tool that uses Utility Theory independently to create artificial intelligences or improve them, on the other hand independent developers if they have created plugins that improve or add certain aspects to the creation of artificial intelligences it is the engine to which we intend to assign our tool. Basic account with a tool for creating behavioral trees to implement artificial intelligences, but lacks other tools for the development of more complex agents or to improve said artificial intelligences, causing them to react to stimuli or learn. Getting them to have these more advanced capabilities is not possible using only behavior trees, this is a problem that will eventually reduce the ability of this engine to solve complex situations. It is therefore important to start developing software that unifies these ideas regarding artificial intelligence in textit UtilityNetwork game engines like Unreal Engine as our bot control system, textit UtilityNetwork.

Apéndice B

Conclusions

After the different tests carried out, we have verified that the work done really adds a new functionality to Unreal Engine in what to create artificial intelligences, giving the option to choose another different method beyond the behavior trees proposed by Unreal Engine, being able on the other hand to add a complex behavior to them in a simpler and intuitive way than the behavior trees for example not having to modify all the *task* (leaves of the tree, actions) created to react to different behaviors while An action is executed.

It is also true that although following the steps proposed in the creation of artificial intelligences a person with little or no knowledge can create one, it will be limited to the resources created, unable to access new actions because they are not able to create them, however a person with prior knowledge will be able to create their own actions and even use the resources provided in ways that had not been previously thought, leading to improve the work done.

Although these are the sensations that we have analyzing the tests done, let's see having the objectives as a goal if we have really been able to fulfill them and to what extent. The objectives were the following:

1. Create a class that allows us to define the node that will have the necessary fields so that the utility can be calculated.

2. Getting the previous class to evaluate the utility with a single variable or with several.

3. Create another class that allows us to work with the nodes created, in order to be evaluated in its entirety and return us the option to take.

4. Allow the creation of graphs that allow us to calculate the score obtained in a node based on certain variables.

5. Provide a series of basic actions that can be increased by the user.

6. Create maps where you can visualize the different behaviors.

And with respect to those objectives, the results have been:

1. This node has been created, it is defined as the data structures textit UtilityNode and UtilityNodeComplex, it has all the necessary fields to be able to calculate the utility therefore this objective has been totally covered.

2. This objective has been completed also but not as initially thought because we have a function to check the utility of the nodes mentioned above, but this function is not implemented in the structure.

3. The class created to manage the treatment of the nodes was also created successfully and fulfills its function. It was created as a child of the class *AIController* and within it we defined the functions necessary to work with the nodes, therefore this objective has also been satisfactorily fulfilled.

4. This objective was also fulfilled because although initially we did not know Unreal Engine had the possibility of creating curves as if they were another *blueprint* more, without having to resort to *timelines* to draw the graphs.

5. The way in which we achieve this goal is slightly different from what was initially thought but the final solution seems more satisfactory because these actions were created in a bookstore, outside the aforementioned classes, so it is not necessary that the user modifies the parent class *MasterUtilityController*, to add actions you simply have to access the library and create the corresponding function there.

6. This objective has also been met, the maps are created and you can see and compare the different behaviors, although in this case you have not been able to reach the goal in a completely satisfactory way, because some behaviors can not be seen as clearly as we would like

To check the difficulties of the people in the use of this tool we made them fill out a form with some questions about the difficulties encountered and when it comes to analyzing the surveys we can also observe that it is relatively easy for people to create artificial intelligences with this method , only watching a tutorial and following their steps while creating artificial intelligence, on the other hand some of the difficulties mentioned are related to the previous knowledge in Unreal Engine, or for some explanation of the tutorial that has not been fully understood by the users.

For these reasons I think that the realization of this work really makes sense for content creators who use Unreal Engine, both for those who have previous knowledge and for those who have just started using Unreal Engine, despite this I also see necessary the continued use of this tool and the reporting of errors that comes with that continued use to be able to polish or fix these errors that arise and of which initially you are not aware and get the tool to continue to improve in the future, which is one of the reasons why the use of behavioral trees has been improving until it reaches what it is today.

B.1. Future Work

Regarding the future development of this tool, beyond polishing the small errors that may arise, I see three ways in which you could focus to improve the tool.

The creation of an interface differentiated from the rest so that, like when we create the behavior trees, the interface changes slightly adapting to the use of the behavior trees, in the same way when creating a class of type *UtilityController* to the Time to work on it could change the interface facilitating its use by helping both those with knowledge and making it easier to use for those who are just starting out. This path involves touching the code in the core of the graphical engine which is not an easy job when we talk about Unreal Engine because in some cases it is difficult to find the documentation of some aspects necessary for this work.

Another possible way to updrade this tool is the dual use of trees of behavior and utility, although it was not an objective or something consciously thought by the way in which I decided to design this tool it is possible to use behavioral trees as a possible action subject to a certain utility. So that an action could be to execute a tree previously created, designed to perform certain actions, for example, merge the use of the areas explained above with the behavior trees so that each field executes a behavior tree, one of the things to study if you follow this path is the performance because it is possible that the use of the utility does not compensate having to use several trees of behavior in the same controller, is an interesting way that tries to merge this tool even more with those already provided by Unreal Engine.

The last possible way that I see of improvement is the inclusion of the theory of probability, the theory of utility is one of the two theories used in decision theory, the other is the theory of probability, in this way if we are able to include the latter together with the use of utility we could create artificial intelligences that make use of decision theory. By including the use of probability together with the use of utility, our artificial intelligences would not only react to external stimuli reacting accordingly, but that the actions to be carried out would be chosen based on the utility of the possible consequences of said actions. say that if you shoot an enemy there is little chance of killing him, that would be the consequence that would be more useful, than hurt him that has more possibilities, but throwing a grenade has many chances to kill the enemy, the action taken would be to throw the grenade in most cases and if we do not have any other variable that intervenes, in this way the artificial intelligences, instead of reacting directly to stimuli, take into account the usefulness of what may happen, ie they take into account the possible future derivatives of his action.

These are the three ways in which I think it can be improved, but being a new tool and not based on a previous one, it is still open to many possibilities therefore it is possible that someone who starts using it from scratch will be able to see other possible ways, which I had not initially considered therefore I think that the most important thing in the future is to be able to use it and receive *feedback* in order to improve it, fix it and be able to discover all its future possibilities.

Apéndice C

Instructions for use

In order to open the project and be able to test the maps in which it interacts with the different bots to verify their reactions, the first thing we have to do is download it, we can do it from:

https://drive.google.com/open?id=1si6fuvBAm4UA0ENAuYZsvBBoY1PX8Pcb

In this address we will access the google drive where it is stored and where we can freely access it. When we run to keep in mind that we must have installed Unreal Engine to version 4.19, which is the version that has been implemented.

Once we open it we will find that it does not start any map, for that we will have to access *Content ->ThirdPersonCPP ->Maps* where we will find the three maps of which we have spoken, in the moment in which we execute one of the three and we give to the play the program will begin to run. In order to move around the world we will use the *WASD*, with the left button we will shoot the bots and with the Q key we will increase or decrease in size.

If what we want to see are the created classes, we can find them in *Content ->TFG ->UtilityNetwork* where we will also find tutorials that will explain how to create an artificial intelligence using the bot control system we have created.