UNIVERSIDAD DE MURCIA

4º de Grado en Ingeniería Informática

IA para el desarrollo de juegos

Documento de Prácticas

José Cerezo Egea – Grupo 1.2

Jose Antonio Nicolás Navarro – Grupo 1.3

Camilo Caparrós Laiz – Grupo 1.2

Convocatoria Junio - Curso 2020/2021

Índice de contenidos

0. Manual de uso

En este manual de uso vamos a explicar los pasos a seguir para poder ejecutar nuestro proyecto correctamente.

1) Abrimos Unity Hub y creamos un proyecto nuevo de tipo “Universal Render Pipeline”.
2) Cerramos el nuevo proyecto.
3) Accedemos a la carpeta del proyecto que acabamos de crear.
4) Borramos todos los archivos que haya en esta carpeta.
5) Pegamos aquí el contenido de la carpeta “Proyecto”.
6) Ya estamos listos para arrancar Unity de nuevo y probar el proyecto.

1. Introducción

En esta memoria de prácticas se explicarán y detallarán todas las ideas de diseño que hemos ido tomando a lo largo del desarrollo del proyecto. La información la dividiremos en dos bloques, pues así se encuentra la asignatura y creemos que es lo más oportuno.

Para el primer bloque comentaremos cada uno de los escenarios de los que disponemos, así como de qué representan y cómo han de ser probados, aunque no nos centraremos en comentar las ideas básicas de ninguno de los algoritmos que se explican en la asignatura, pues no es relevante.

Para este mismo bloque, también comentaremos cada uno de los apartados opcionales que hemos decidido implementar y en qué escenarios se encuentran.

En el segundo bloque hablaremos de todo lo relacionado con el mapa, de los tipos de NPCs que tenemos, de la toma de decisiones, etc. Al igual que en el primer bloque, también dedicaremos unos apartados especiales a ver qué apartados opcionales hemos implementado.

2. Bloque 1

2.1. Escenario de los steering básicos

En esta primera escena de steerings, mostramos el funcionamiento de los steerings básicos desarrollados que son: *Seek*, *Flee*, *Arrive*, *Leave*, *Align*, *AntiAlign* y *Velocity Matching*. Cada uno de estos steerings se pone a prueba en un NPC cuyo nombre es similar al del steering, de cara a su fácil localización.

En la escena hay, además, un muñeco de color verde que representa al *player*, el cual se mueve mediante las teclas *w*, *a*, *s* y *d*. De cara a ejecutar las pruebas en la escena, hay que tener en cuenta que todos los steerings mencionados tienen como target el player, por tanto, realizarán sus acciones en base a este.

Destacar que tanto en este como en otros escenarios del bloque 1, es posible que salten ciertos mensajes de warning sobre atributos que no se utilizan, pero esto es normal y no supone ningún problema, pues son atributos utilizados en el bloque 2.

2.2. Escenario de los steering delegados

Para esta segunda escena de steerings, ilustramos el comportamiento de algunos steerings delegados como son: *Pursue*, *Face*, *Wander*, *Path Following* y *Offset Pursuit*. Al igual que la anterior escena, cada uno de estos steerings es aplicado a un NPC distinto.

Además, volvemos a tener un player respecto al cual funcionan los NPC Pursue, Face y Offset Pursuit. Estos tres steerings tienen como target un agente auxiliar, que se trata de un agente invisible (visible en *modo debug*, es decir, al pulsar *Tab*) el cual será al que modifiquemos según el player (que se encontrará como *explicit target*). Por ejemplo, en el caso de Face, lo que hacemos es utilizamos el explicit target para calcular hacia dónde tenemos que mirar, pues queremos mirar hacia el player, esa orientación se le pone al agente auxiliar y delegamos en Align para rotar. Comentar que Offset Pursuit tiene el vector de offset en el inspector, por tanto, se puede modificar en cualquier momento por si se quiere.

Para el caso del NPC Wander, tanto el target como el explicit target serán el agente auxiliar pues se delega de Face para realizar la rotación y, por tanto, para que funcione correctamente, debe mirar al mismo punto al que se tenga que desplazar.

En Path Following lo que hacemos es utilizar el agente auxiliar para hacerle Seek a cada uno de los puntos del camino a seguir. Este camino se le pasa al script por el inspector y se compone de un conjunto de agentes, formando el camino deseado. Comentar que decidimos que el funcionamiento sea cíclico, es decir, cuando el NPC llega al final del camino va directamente al primer nodo del camino.

2.3. Escenario detección de colisiones

En este escenario hemos incluido las pruebas del *Collision Avoidance* con un obstáculo móvil y otro inmóvil. En esta escena, para poder poner a prueba el código, es necesario controlar a un jugador mediante teclado, usando las teclas W, A, S y D. El agente que tiene el script de Collision Avoidance, tiene además un Seek, el cual tiene como objetivo el jugador, para lograr así que el NPC lo persiga. La idea es moverse detrás de los obstáculos, en movimiento o no, para observar cómo el agente evita atravesar y chocar contra ellos.

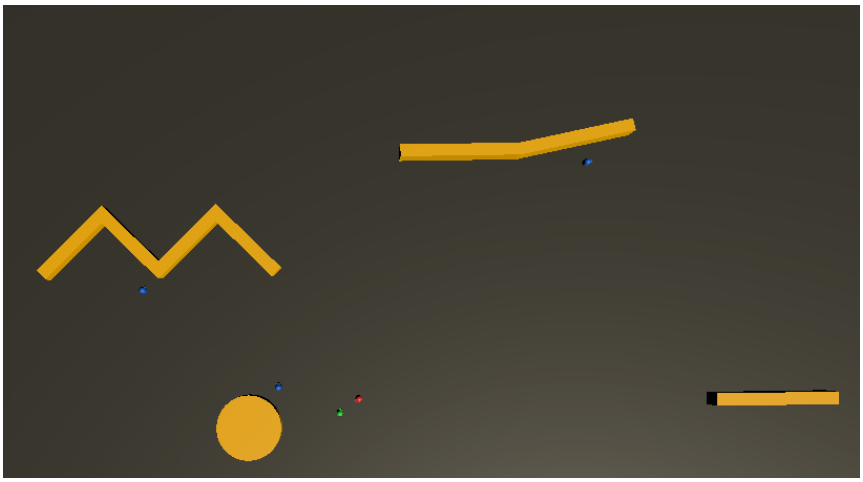

Figura 1. Muestra escenario detección de colisiones.

2.4. Escenario formaciones

En este escenario nos preocupamos de mostrar la funcionalidad de las formaciones. Estas funcionan seleccionando un número (cualquiera) de NPCs, mediante las teclas Ctrl + Click izquierdo (si no se pulsa Ctrl solo se puede seleccionar un NPC a la vez). Si el número seleccionado de NPCs es mayor que el posible dentro de una formación, esos restantes no formarán, pero seguirán estando seleccionados. Como ejemplo de esto, veamos nuestra primera formación fija:

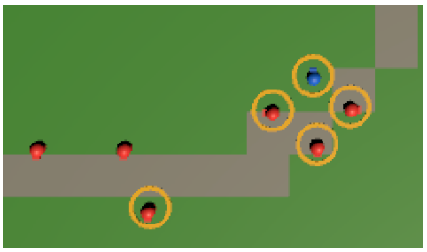

Figura 2. Ejemplo de formación fija (Tecla Z).

El funcionamiento de las formaciones es el siguiente: el controlador tiene una lista de agentes NPC que han sido seleccionados, al pulsar la tecla Z, se llama al método *hacerFormacion*

que recibe el número de formación que se ha seleccionado. A continuación, se llama al script Formaciones que será el encargado de devolver una lista de tuplas (de tamaño igual al de los slots de la formación, si es fija, y sino de tamaño igual al número de seleccionados) que contienen *(posición, orientación),* siendo estos valores relativos a los del líder de la formación. Es designado líder el primer NPC que se encuentre en la lista de seleccionado, pues realmente no es demasiado relevante quien lo sea. Una vez tenemos la lista de tuplas, para cada agente NPC seleccionado se llama a su función *formar* con una tupla distinta, que, al recibirla, creará los steerings *OffsetAlign* y *OffsetArrive* que le permitirán estar siempre orientado y situado de forma relativa al líder.

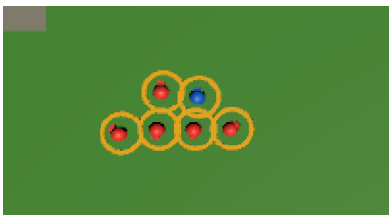

Figura 3. Ejemplo de formación fija (Tecla X).

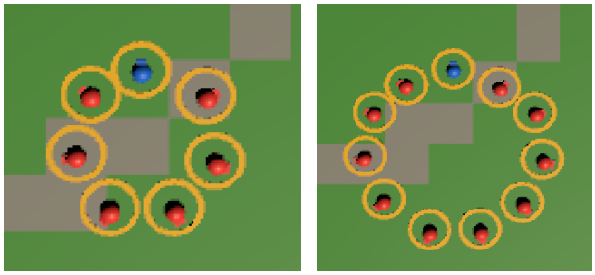

Figura 4. Ejemplo de formación escalable (Tecla C).

El funcionamiento del click derecho (mover uno o varios NPCs) mezclado con las formaciones consiste en comprobar si todos los NPCs seleccionados se encuentran o no en formación (en el controlador), en caso afirmativo, solo se desplazará al líder, pues el resto, mediante los steerings comentados anteriormente, serán capaces de seguirlo sin problema. En caso negativo, lo que hacemos, para cada NPC, es calcular un punto aleatorio de destino alrededor de dónde se ha hecho click (para que no todos vayan al mismo punto y se acumulen) y llamar al método *newTarget*. Dentro de este método (de la clase AgentNPC) lo que haremos es indicar a todos los NPCs que ya no están en formación (les anulamos los steerings) y para cada uno calculamos mediante LRTA* su path hasta el destino.

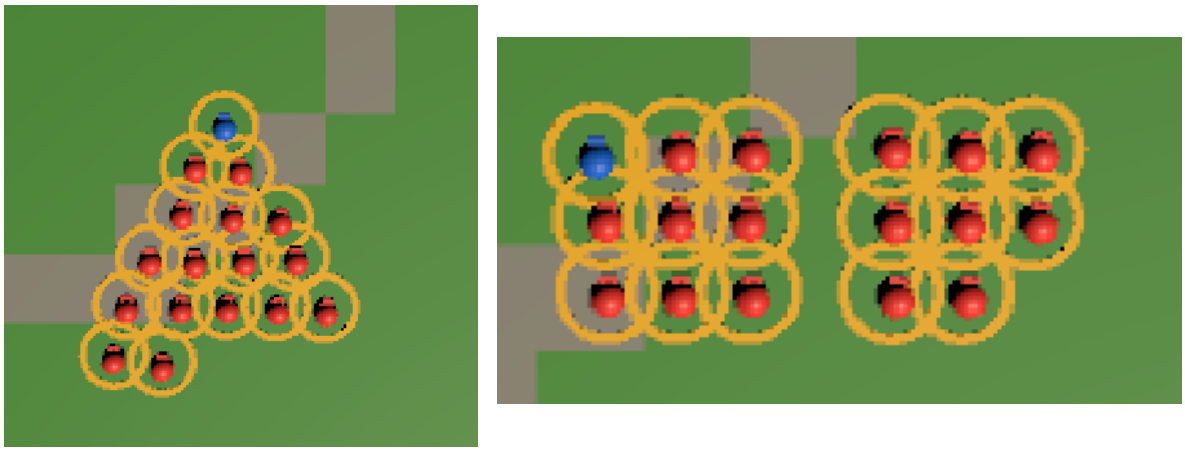

Figura 5. Otras formaciones escalables (Teclas V y B).

2.5. Pathfinding

Para la creación del pathfinding con LRTA* de subespacio minimal, lo primero que creamos fue el terreno basado en 'tiles' o casillas, de tamaño 30x30, de forma que cada celda representa un nodo. Como posición representativa del nodo se han tomado las caras.

Este terreno consta de dos zonas divididas que pueden ser atravesadas utilizando los pasos disponibles. Lo hemos dividido de esta manera también para poder mostrar las deficiencias del algoritmo al considerar el subespacio de búsqueda mínimo. El agua no puede ser atravesada y además hemos añadido un obstáculo que tampoco puede ser atravesado y que no forma parte del terreno per sé, para demostrar que se tiene en cuenta esta situación y que el algoritmo ignora los nodos bajo este obstáculo. Para hacer evidente que el agua no puede ser atravesada hemos creado un pequeño islote rodeado de nodos marcados con el tag 'Agua'.

Hemos tenido en cuenta las situaciones en las que no exista un camino también y se puede comprobar en este último caso. A continuación, mostramos una captura sobre el terreno:

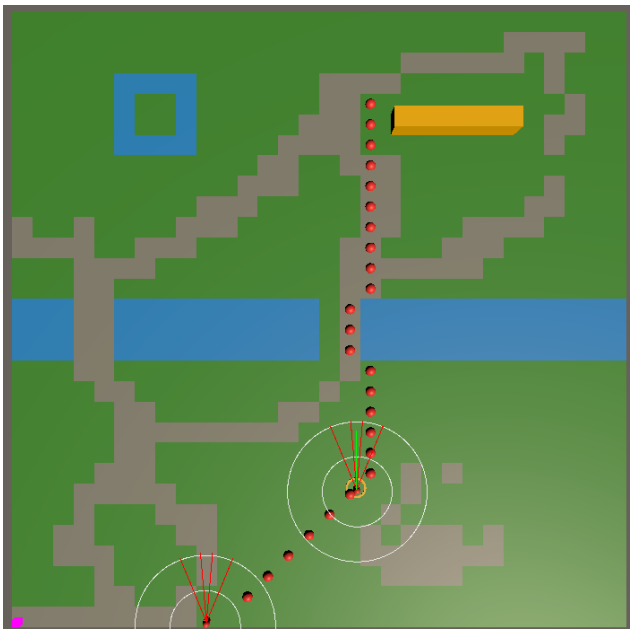

Figura 6. Ejemplo funcionamiento pathfinding.

Teniendo en cuenta que había que implementar otro tipo de pathfinding en el Bloque 2, creamos una clase 'Pathfinding' que contiene ambos algoritmos. Los únicos 2 parámetros de esta clase son el tipo de heurística, que se puede seleccionar desde el inspector y el 'Grid' con los nodos. El propio algoritmo simplemente recibe el nodo del cual se parte y nodo al que se quiere llegar. Sabemos que este algorimo devuelve un solo nodo cada vez, pero está optimizado para poder guardar los nodos calculados en una lista que es lo que devuelve finalmente. Así facilitamos la integración con el steering que va haciendo 'Seek' a los nodos para que los agentes se muevan.

2.6. Árbitro ponderado

Los agentes NPC pueden tener múltiples steering behaviours, por tanto, necesitamos un método para combinar todos los steerings resultado de estos steering behaviours. Una forma de resolverlo es el arbitraje ponderado. Nuestra implementación no difiere de la de [1], excepto que es una simple función.

ArbitroPonderado(npc, behaviours)

- npc: agente NPC al que aplicar los steering behaviours.
- behaviours: lista de steering behaviours a aplicar al NPC.
- return: steering resultado.

Lo único que hace es aplicar los steering behaviour, acumular los resultados mediante la suma ponderada y limitar el resultado a la aceleración máxima del NPC.

2.7. Leader following

Para la escena leader following tenemos un agente player al cual le siguen una serie de agentes NPC. Estos tienen los steering behaviours OffsetArrive (para seguirle, pero con una distancia de separación), Separation y Leave.

El esquema era:

1. Grupo Following con quantum 3.
2. Grupo Fleeing con quantum 1.
3. Grupo Collision con quantum 3.
4. Grupo Fleeing con quantum 1.

Vemos que con mayor prioridad haremos OffsetArrive y Separation y entre medias haremos Leave al líder, con esto conseguiremos que eviten al líder, pero sobre todo que le sigan y se separen entre ellos.

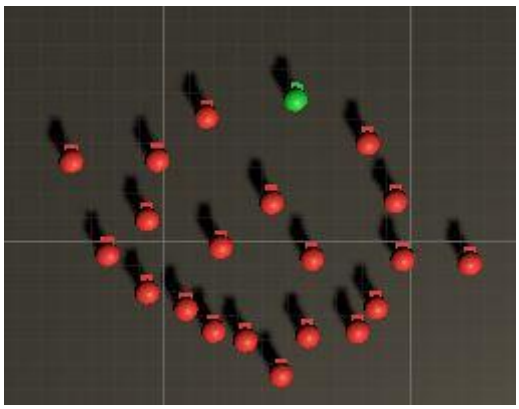

Figura 7. Ejemplo funcionamiento leader following.

2.8. Flocking

Nosotros hemos modificado el flocking para que sea realmente lo que hemos llamado un Shark Fleeing, que es lo mismo, pero también huyen de un tiburón. En la escena, tenemos un NPC blanco con Wander, varios NPC rojos con steering behaviours de flocking, Leave huyendo del tiburón y Seek siguiendo al NPC Wander, y el NPC tiburón que tiene Seek a algún NPC de los que realiza el flocking.

Recordemos el esquema:

5. Grupo Fleeing con quantum 3.
6. Grupo Flocking con quantum 2.
7. Grupo Following con quantum 1.

Vemos que la mayor prioridad es huir del tiburón, después es flocking, que tiene los pesos 3 para Separation, 2 para Cohesion y 1 para Alignment. Por último, tiene un Seek. Al haber tantos NPC en esta escena, es mejor no activar el debug.

Cuando no está el tiburón cerca, realizan el flocking y siguen al wander.

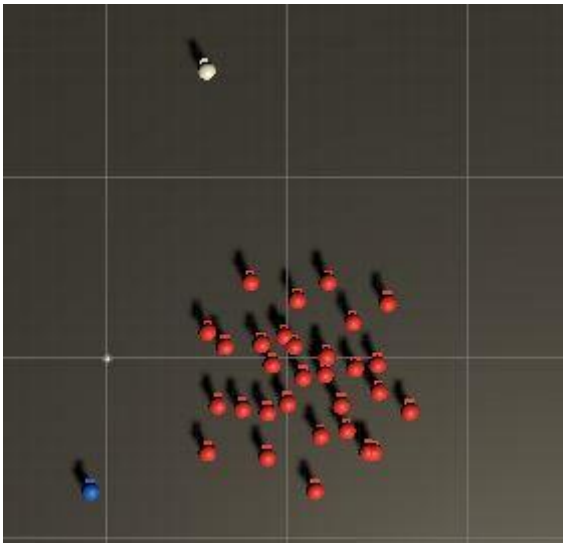

Figura 8. Ejemplo funcionamiento de flocking (parte 1).

Cuando el tiburón pasa por delante, se paran. Cuando los atraviesa, se apartan.

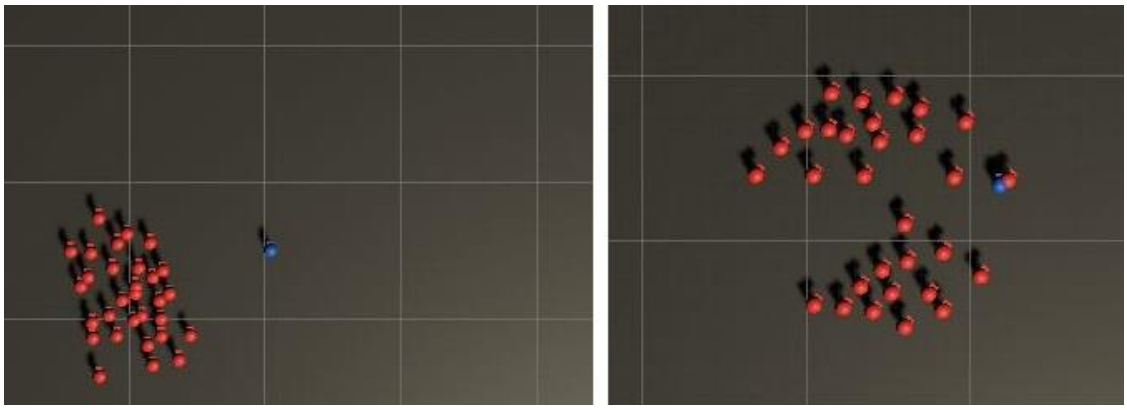

Figura 9. Ejemplo funcionamiento de flocking (parte 2).

2.9. Elementos opcionales

2.9.1. Steerings opcionales

Como parte opcional, decidimos implementar el paquete de steerings, que contienen *Collision Avoidance* basado en predicción, *Path Following* con offset predictivo, *Interpose*, *Wall Following* y *Look where you going*.

Path Following con offset predictivo, Interpose y Look where you going se encuentran en la segunda escena que comentamos, la de los steerings delegados. Esto lo hemos decidido hacer para poder comparar el comportamiento de ambos Path following dado el mismo camino y puesto que todos son steerings delegados. En la ejecución de la escena, se ve claramente como Path following predictivo se comporta de una forma distinta al que no predice.

En el caso de Interpose, tenemos como targets (NPCs entre los que se tiene que interponer) el player y el wander, por tanto, cuando se inicia la escena, interpose se estará moviendo continuamente gracias al wander, aunque se podría mover también al player para ver su funcionamiento.

Para Look where you going, lo más sencillo es ejecutar el juego y en tiempo de ejecución (mediante el inspector) cambiarle el vector velocidad para ver como varía su orientación.

Por otro lado, Wall following se encuentra en la escena *DeteccionColisiones*, ya comentada previamente, en la que tenemos tres pruebas distintas. La primera consiste en darle vueltas a un objeto circular, la segunda es un ejemplo sencillo de pared recta y el último ejemplo es un muro con forma de M, para intentar ponérselo lo más complicado posible al NPC. En ambos casos, se observa como el NPC consigue realizar su función sin problema.

Por último, el steering Collision Avoidance basado en predicción se encuentra en una escena separada llamada *PredictiveCollisionAvoidance* en la que se sitúan dos targets, uno frente al otro, para forzar la colisión. Cuando se ejecuta la escena, se ve como los NPCs se esquivan correctamente a pesar de que luego no se comportan del todo bien, pero esto es debido a que no saben qué hacer a continuación, pero esto se solucionaría mediante la combinación con otros steerings para realizar una vez evita la colisión, como podría ser un wander.

2.9.2. Árbitro prioritario dinámico

Sabemos que en el arbitraje prioritario agrupamos los steerings en ciertos grupos y aplicamos el árbitro ponderado solo a los steerings del grupo más prioritario. Para hacerlo dinámico, necesitaremos una planificación de los grupos. Veamos los grupos definidos:

- **Flocking:** Separation, Cohesion, Alignment, Wander.
- **Following:** Seek, Arrive, VelocityMatching, OffsetArrive, Wander, WallFollowing.
- **Fleeing:** Flee, Leave.
- **Collision:** WallAvoidance, Separation, CollisionAvoidance.

- **All:** todos.

Ahora, los esquemas son unas listas de grupos con sus quantums asociados.

- **Esquema Shark Fleeing:**
    1. Grupo Fleeing con quantum 3.
    2. Grupo Flocking con quantum 2.
    3. Grupo Following con quantum 1.
- **Esquema Leader Following:**
    1. Grupo Following con quantum 3.
    2. Grupo Fleeing con quantum 1.
    3. Grupo Collision con quantum 3.
    4. Grupo Fleeing con quantum 1.
- **Esquema All:**
    1. Grupo All con quantum 1.

Nuestra planificación es parecida a un Round Robin. Consiste en consumir siempre el primer grupo hasta que su quantum sea 0, en ese caso, se quita de la cabeza y se devuelve a la cola con su quantum original (definido en los esquemas anteriores). Los quantums se consumen cada ciclo. Esta planificación es eficiente mediante una lista enlazada porque el acceso a la cabeza es constante y la inserción al final de la cola también.

Lo bueno de este algoritmo es que es equitativo y conserva el orden inicial de los grupos, ya que son rotaciones y no reordenaciones. Lo malo es que quantums muy altos y desbalances en los mismos podrían dar malos resultados. No obstante, se puede remediar repitiendo grupos en el esquema. Los parámetros son:

ArbitroDinamico(npc, esquema, steerings)

- npc: agente NPC al que aplicar los steering behaviours.
- esquema: lista de grupos, da la estructura que deben formar los behaviours.
- behaviours: lista de steering behaviours a aplicar al NPC.
- return: steering resultado.

El algoritmo agrupa la lista de behaviours según la estructura dada por esquema y aplica el árbitro ponderado al primer grupo. Hay que tener en cuenta que el parámetro esquema se refiere a la instancia particular de esquema del agente NPC, no a la instancia inicial.

Los dos esquemas usados se explican a continuación.

2.9.3. Formaciones con roles

Para la creación de formación con roles, hemos definido la siguiente estructura compuesta por cuatro roles diferentes; Tanque, Melee, Apoyo Lateral y Retaguardia:

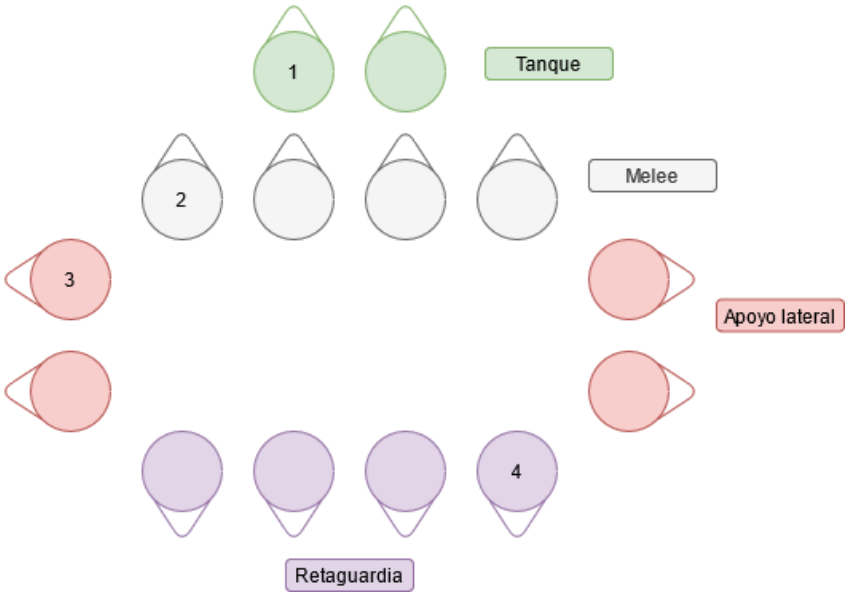

Figura 10. Diagrama formaciones con roles.

Para la versión de la formación con roles duros hemos definido la tabla de la izquierda que indica qué roles pueden ocupar qué unidades (marcados con una cruz verde). Para los roles suaves, como las unidades pueden ocupar más de un rol, damos unos valores de adecuación al rol, indicando el 0 que la unidad es muy adecuada para ese rol y cuanto más alto sea este valor, menos adecuado es para el rol:

Rol/ Unidad	Melee	Apoyo Lateral	Retag.	Tanque
Ligero	✕		✕	✕
Pesado	✕			✕
Espía	✕	✕		
Ranged		✕	✕	

Roles duros

Rol/ Unidad	Melee	Apoyo Lateral	Retag.	Tanque
Ligero	0	50	0	0
Pesado	0	100	150	0
Espía	0	0	30	120
Ranged	50	0	0	60

Roles suaves

Figura 11. Ejemplo 1 formación roles duros.

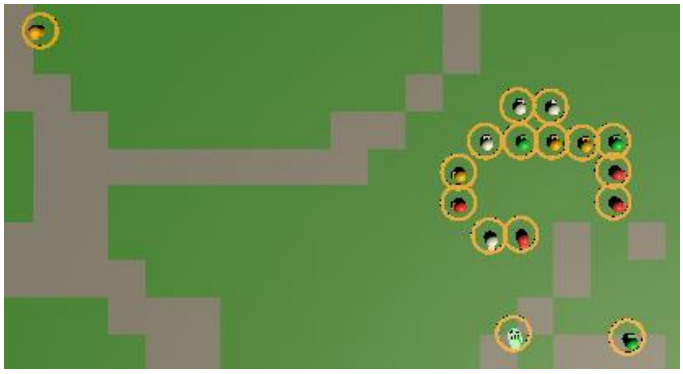

Figura 12. Ejemplo 2 formación roles duros.

En la siguiente imagen se muestra un ejemplo de la formación con roles suaves, que no es tan restrictiva con las unidades que ocupen los roles y siempre se va a completar (mientras haya suficientes agentes para formar):

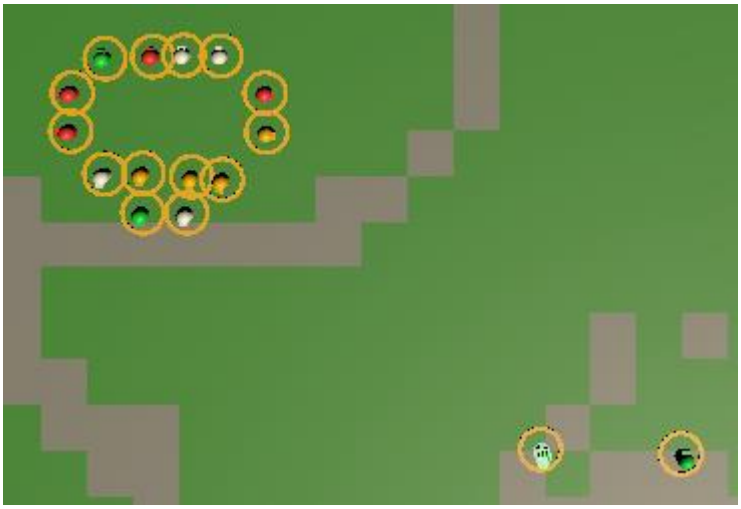

Figura 13. Ejemplo formación roles suaves.

3. Bloque 2

3.1. Diagrama de clases

Para la creación de distintos tipos de tropas hemos partido de la clase 'AgenteNPC' y creado la clase 'Soldado', de la que heredan no solo los tipos de tropa sino también la clase 'Estructura'. La justificación de que 'Estructura' herede de 'Soldado' es que, para facilitar el diseño del sistema de combate, era conveniente tratar a los edificios como agentes a los que se pueda detectar, perseguir y atacar.

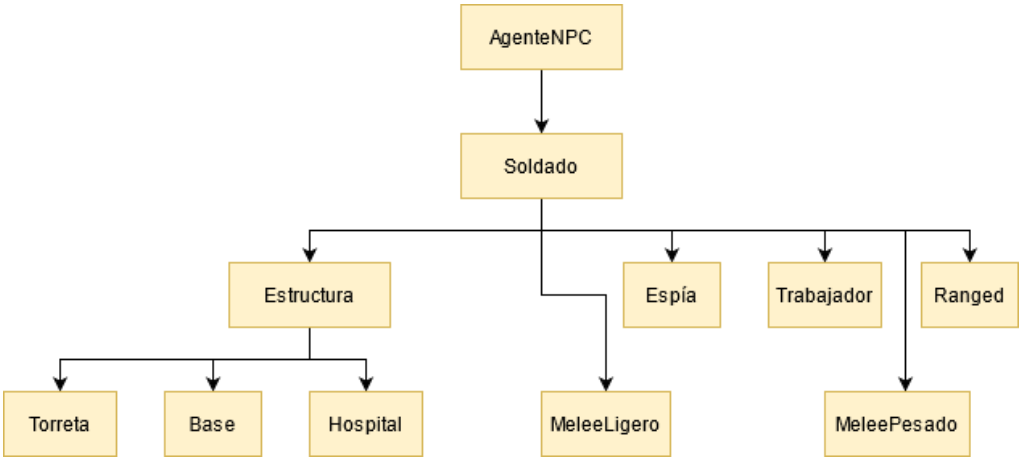

Figura 14. Diagrama de clases.

Como puede verse, se han creado 3 tipos de estructuras; Torretas, Bases y Hospitales. Las torretas forman parte del apartado de la creación de un RTS parcial y se mencionarán más adelante. Las bases son los núcleos de cada bando y es necesario destruirlas para poder ganar la partida. Los hospitales forman parte de las zonas seguras que se piden en este bloque, y su cometido es sanar a las unidades que se encuentren en su radio de curación.

En cuanto a tipos de soldado, se han creado 4. El trabajador forma parte de la creación de un RTS parcial, pero su función es la de conseguir recursos para poder construir torretas que protejan a su bando. Si no tiene recursos, va a buscarlos. Si ya tiene y existen defensas construibles, se dirige a ellas para construirlas. Si no existen defensas que se puedan construir y dispone de recursos, el trabajador vuelve a la base. El trabajador no tiene preferencia por ningún terreno en especial. El melee ligero es una unidad estándar de rango reducido y velocidad normal que tiene preferencia por los caminos. El melee pesado es una versión de un melee más lenta, con más vida y con más defensa. El ranged es una unidad a distancia con más rango que los melees y más rápida con preferencia por terreno donde haya hierba. El espía es un despiadado y entrenado asesino que trata de adentrarse en las líneas enemigas sin ser detectado. Es por esto que tiene preferencia por terrenos alejados de los caminos, como los bosques.

3.2. A* táctico

Para hacer que el pathfinding sea táctico en el Bloque 2, hemos utilizado una implementación de A* (obtenida de [2]) en C#, puesto que se comporta mejor que LRTA* al calcular todo el camino de golpe y evitar zig zags en esquinas, para evidenciar así los comportamientos de los agentes y que este pathfinding no lo esconda en cierto sentido.

Dentro del algoritmo, hemos sumado al valor de las heurísticas el valor de la influencia ejercida sobre los nodos que el algoritmo considera, así como un coste que se calcula en función de lo fácil que le resulta a cada unidad moverse sobre cada terreno, de forma que cuanta más influencia se detecte en un nodo ejercida por el bando contrario, menos coste tendrán los nodos que se consideren. Esto corresponde con el apartado f) de los elementos opcionales.

3.3. Descripción del terreno

A modo de juego final y de cara al bloque 2 de la práctica, creamos un escenario (*Bloque2*) que consta de múltiples asuntos relevantes a comentar. El primero es que la escena está creada de la misma forma en la que fue creada la de Pathfinding, es decir, el terreno consiste en un grid que contiene un conjunto de nodos, cada uno con un tag distinto, indicando qué terreno es: Camino, Hierba, Bosque o Agua.

Figura 15. Mapa del bloque 2.

En primer lugar, el mapa se encuentra dividido por un río que no es posible de atravesar salvo por los dos puentes que hemos hecho en el mapa. En la izquierda del río tenemos al bando azul y a la derecha el rojo. En el mapa hemos situado diferentes tipos de terreno de forma que se pueda apreciar como una tropa prefiere un tipo de terreno u otro. Además, hemos añadido algún elemento decorativo como los árboles y otros decorativos y funcionales como las rocas, pues son un obstáculo y por tanto es como el agua, no se puede atravesar.

Ambos bandos disponen de una base (torre) y un hospital (tienda de campaña). La base será el punto desde el que aparecerán los NPCs y además es la estructura a defender, pues si esta es destruida por el bando contrario, supondrá la derrota. Por otro lado, tenemos el hospital que es la estructura en la que los NPCs podrán ir a recuperar vida si están dentro del área de curación que se encuentra marcada por un círculo del color del bando.

Otra de las cosas que tenemos son torretas de cada bando, que están situadas en el puente más cercano a la base, en la orilla del río (representadas mediante un círculo, aunque cuando estén construidas cambiarán su apariencia). Esta estructura será capaz de atacar a los NPCs cuando estén construidas y se podrán destruir. Hablaremos más delante de esta mejora opcional. Algo relacionado con esto, son los recursos (barriles), que se encuentran repartidos por el mapa y no tienen bando. Estos recursos serán recolectados por los trabajadores y son necesarios para construir torretas.

Por último, de cara a mejorar el aspecto visual y la información de la que disponemos, hemos implementado unas barras de vida, tanto a las estructuras como a los NPCs. Además, las torretas dispondrán de dos barras, una de vida (cuando ya se encuentre construida) y otra de construcción (de color amarillo), sirviendo esta última para indicar el tiempo restante para que la defensa termine de construirse.

3.4. Mapa de influencias

Para el mapa de influencias partimos de que tenemos puntos de un grid y sus influencias asociadas. Lo único que habría que hacer es generar una matriz y extender las influencias de esos puntos mediante algún método. Los parámetros son:

Influencias(influencias, dimension)

- influencias: lista de tripletas (posición x, posición z, influencia en ese punto).
- dimension: par de enteros con el tamaño del mapa generado.
- return: matriz de influencias

Las influencias positivas son del bando rojo y las negativas las del bando azul. El algoritmo primero genera toda la matriz con valores neutrales. Seguidamente, para cada punto de influencia se obtiene otra serie de puntos de influencia que acumulamos en el mapa neutral inicial. Esta serie de puntos podrían ser tantos como el tamaño de la matriz, pero no es necesario si definimos radios de influencia. Nosotros definimos el radio como el valor absoluto de la influencia en el punto inicial.

Por último, necesitamos el método de cálculo de la influencia alrededor del punto. Sabemos que hay varios en [1]. Nosotros hemos implementado uno lineal, es decir, la influencia se reduce directamente con la distancia. La razón es porque es más fácil de entender por un usuario final y, además, es más eficiente al usar sumas y no multiplicación ni división.

La generación del mapa de influencias es costosa aun definiendo radios de influencia, así que la ejecutamos en una corrutina cada cierto tiempo.

3.5. Toma de decisiones

Árbol soldado

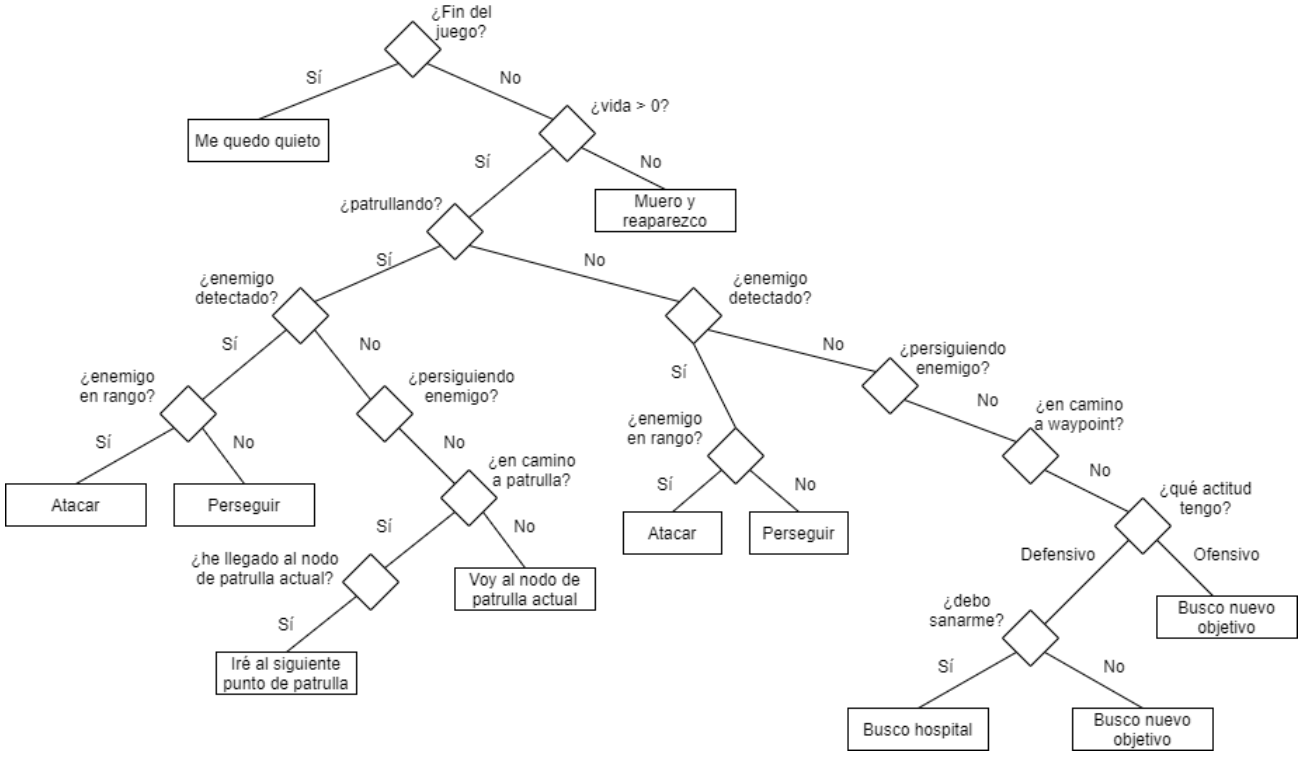

Figura 16. Mapa de decisiones de la clase soldado.

Árbol espía

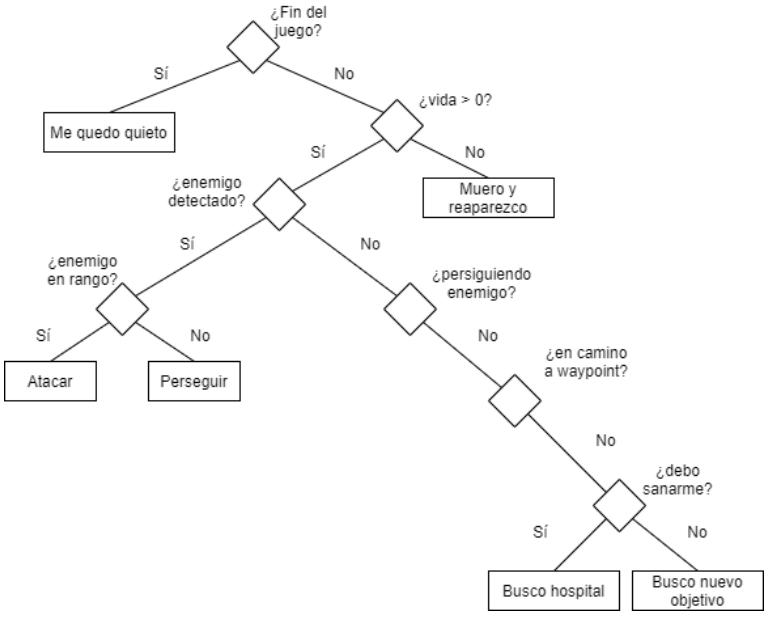

Figura 17. Mapa de decisiones de la clase espía.

Árbol trabajador

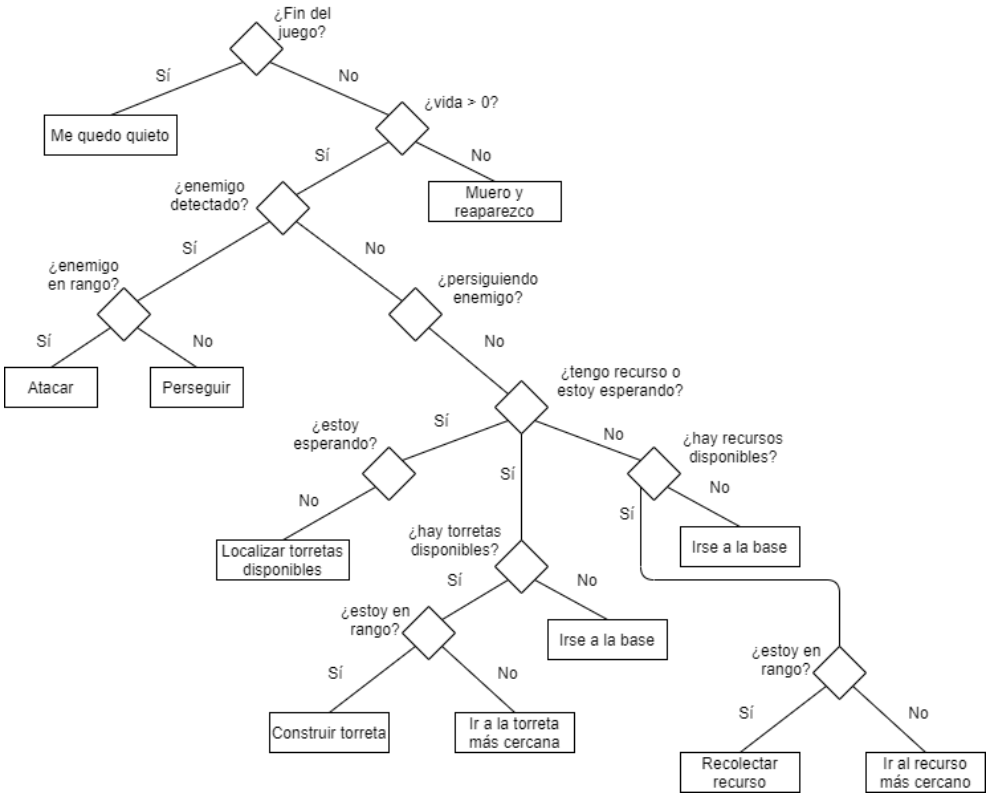

Figura 18. Mapa de decisiones de la clase trabajador.

3.6. Elementos opcionales

3.6.1. Unidades patrulleras

Las patrullas tienen un camino fijo predeterminado en la escena, mediante el uso de waypoints (objetos empty). Los NPC que patrullen tendrán el steering behaviour de PathFollowing y el camino se calcula con A* táctico. Para que reaccionen a otros NPC hay una condición en el árbol de decisión que permite atacar un enemigo si es detectado.

3.6.2. Extensión del juego a un RTS parcial

Uno de los apartados opcionales que decidimos implementar fue la extensión del juego a un RTS parcial, pues era una idea que nos atraía. Este apartado fue llevado a cabo primeramente mediante la creación de script que hereda de Soldado, llamado Trabajador. Este trabajador será un NPC el cual se encargará tanto de recoger los recursos disponibles por el mapa, así como de construir las torretas defensoras.

Para los recursos se creó el script *Recurso* que contiene la información de los tiempos de respawn y recolección del mismo, además de la funcionalidad necesaria para su correcto funcionamiento. En el caso de las defensas, se creó el script *Torreta* que tiene algo más de funcionalidad que un Recurso, pues una torreta tiene barra de vida y de construcción, además de poder atacar, ser destruida, etc.

El comportamiento del trabajador se basa en buscar recursos para poder construir las defensas. Cuando tiene un recurso, busca un sitio disponible para construir la torreta (cada bando dispondrá de dos ubicaciones posibles para construir torretas, ya prefijados) y si no lo hubiera vuelve a la base a esperar, pero si mientras transporta un recurso se muere, perderá el recurso y al reaparecer deberá ir a por otro nuevo.

Si mientras un trabajador se encuentra construyendo una torreta es atacado o visualiza a un enemigo, parará esta construcción y se irá a por él, teniendo así que volver a construir la torreta de nuevo si quisiera.

Cuando una torreta es construida, esta tiene el poder de atacar a otros NPCs enemigos, así como de ser atacada o incluso destruida. Cuando una torreta es destruida, si el trabajador quiere restaurarla, deberá utilizar otro recurso.

3.6.3. Toma de decisión con varios niveles de información táctica

En la toma de decisiones se usa información tanto interna al personaje (rangos, vida, visión) como información táctica, principalmente para el A* (mapa de influencias, información de terrenos, waypoints, hospitales). Algunos personajes como el trabajador también usan otra información como la posición de recursos y torretas, y si están construidas o no.

La información del personaje se usa más en las condiciones de la toma de decisiones mientras que la información táctica se usa más en la ejecución de acciones.

4. Mapa de teclas

Nuestro juego consta de las siguientes acciones que involucran al usuario:

- Si en una escena, como la de steerings básicos, aparece un Player (tiene color verde), este se moverá mediante las teclas W, A, S, D.
- Para la selección de NPCs, si solo se quiere seleccionar uno bastará con pulsar click izquierdo, pero si se quisieran seleccionar más, habría que utilizar el control a la vez que el click izquierdo.
- En la escena de formaciones, se pueden mandar a formar los NPCs una vez estén seleccionados y mediante las teclas Z (fija1), X (fija2), C (escalable círculo), V (escalable triángulo) y B (escalable cuadrados).
- Para la escena de Pathfinding, mover un NPC o varios será posible si estos se encuentran seleccionados y si se pulsa click derecho en el punto de destino deseado.
- Por último, en la escena de formaciones con roles, estas se pueden ejecutar seleccionando los NPCs deseados y pulsando N (roles duros) o M (roles suaves).

Figura 19. Mapa de teclas.

5. Interfaz de usuario

En el bloque 1 no se dispone de interfaz de usuario, simplemente las escenas son ejecutadas y el usuario puede iniciar ciertas acciones utilizando el mapa de teclas comentado anteriormente.

Para el bloque 2 sí disponemos de una interfaz de usuario sencilla. Esta interfaz consta de una serie de botones que servirán para modificar la *actitud* de los NPCs, para así ser capaces de modificar el comportamiento de cada bando en tiempo de ejecución.

Otro botón que está implementado es el de salir, pero no funciona, pues en Unity, la llamada a Application.Quit() solo funciona cuando el juego se ha exportado.

Además, en la parte izquierda hemos situado el mapa de influencias que nos servirá para entender el porqué de las decisiones tomadas por los NPCs en tiempo de ejecución.

Figura 20. Interfaz de usuario del bloque 2.

6. Conclusión

En esta práctica hemos podido implementar varios paquetes de steering behaviours tanto individuales como de grupo y ver cómo componerlos y orquestarlos.

También hemos implementado dos algoritmos de búsqueda de caminos, LRTA* y A*, táctico viendo sus puntos débiles y fuertes.

Por último, hemos tenido la oportunidad de ver técnicas para simular el comportamiento inteligente de NPCs usando árboles de decisión e información táctica como waypoints y mapas de influencias.

7. Bibliografía

[1] Ian Millington and John Funge. Artificial Intelligence for Games 2nd Edition. Morgan Kaufmann.

[2] https://github.com/danielmccluskey/A-Star-Pathfinding-Tutorial/blob/master/Assets/Pathfinding.cs