Universidad de Murcia
Grado en Ingeniería Informática

IA PARA EL DESARROLLO DE JUEGOS

Curso 2019 - 2020

Simón Navarro Almenzar
Alberto García Sáez
José García Ramírez

Profesores: Luis Daniel Hernández Molinero
Francisco Javier Marín-Blázquez Gómez

Julio de 2020

Índice.

1. Introducción.

En esta práctica, tenemos como objetivo la implantación de un sistema de IA enfocada al mundo de los videojuegos. Constará de dos partes en las que diferenciaremos primero una parte donde se implementan los elementos asociados al movimiento de los personajes (distintos Steerings) y una segunda parte donde se implementa un modo de IA más relacionada con la estrategia y la táctica de los personajes, desarrollado para un juego de guerra.

2. Entorno de desarrollo.

Para el desarrollo de la práctica nos hemos decantado por el motor de desarrollo de videojuegos en multiplataforma, Unity, concretamente en su versión 2019.3.2f1. Elegimos este motor, además de que se recomienda por los profesores (aunque no sea un requisito), porque también tuvimos una primera toma de contacto en el primer cuatrimestre con la asignatura de Fundamentos Computacionales de los Videojuegos.

Para el desarrollo de la práctica, hemos utilizado nuestros ordenadores personales con sistema operativo Windows 10, y hemos trabajado conjuntamente mediante la aplicación de escritorio remoto AnyDesk, así como la aplicación de comunicaciones Discord.

3.Listado de clases.

```
Agent                   Pursue
AgentNPC                WallAvoidance
AgentPlayer             Wander
Mele                    Circle
PhysicalBody            Form
Ranged                  Formation
TodoTerreno             Line
Camara                  Slot
Collision               Grid
CollisionDetector       Node
Manager                 Aligment
Menu                    Cohesion
Path                    Separation
Align                   BlendedSteering
Anti-Align              Flocking
Arrive                  LeaderFollowing
Flee                    PFLWYG
Seek                    SeekLWYG
VelocityMatching        ObstacleAvoid
Evade                   Attack
Flee                    GoToRespawn
LookWhereyouGoing       GoToWayPoint
PathFollowing           InfluenceMap
Minimap                 Steering
RunAway                 SteeringBehaviour
State                   ArriveBase
StateMachine            FleeBase
PathFinding             SeekBase
WanderBase
```

4. Bloque 1: Movimiento

4.1 Estructuras para el almacenamiento de información.

El comportamiento **Steering** se trata mediante dos clases:

- **Steering**: Contiene los parámetros *linear* y *angular*
- **SteeringBehaviour**: Es la clase padre de cualquier comportamiento Steering. Contiene un método heredable *getSteering()* y un *Target* sobre el que se realizarán algunos cálculos.

La clase principal que gestiona el comportamiento de los NPC se llama **AgentNPC**. Esta clase hereda directamente de **Agent** y **PhysicalBody** que contienen la mayoría de los atributos necesarios. Esta clase aplica el método *getSteering()* del **SteeringBehaviour** que esté utilizando y lo aplica utilizando las ecuaciones de Newton-Euler

$$e = e_o + v_o \cdot t$$
$$v_f = v_o + a \cdot t$$

$$\theta = \theta_0 + \omega_0 \cdot t$$
$$\omega = \omega_0 + \alpha \cdot t$$

4.2 Steerings acelerados.

Seek: El Agente va desde su posición hasta la posición de su *Target* calculando la distancia entre los dos. Cuando la distancia es nula, se le aplica aceleración negativa para detener su movimiento.

Flee: Al contrario que **Seek**, flee calcula la posición desde el *Target* hasta la posición del agente, de manera que avanza en dirección contraria huyendo de él.

Arrive: De forma análoga a **Seek** calcula la distancia entre su posición y la posición del *Target*, con la diferencia de que a un cierto radio del *Target* aplica una deceleración progresiva.

Align: Calcula la diferencia entre los valores de orientación del Agente y su *Target*, aplicando una aceleración angular hacia la orientación del *Target* en función de un radio angular, de la misma forma que sucede con **Arrive**.

Anti-Align: Funciona de la misma manera que **Align** pero desde la orientación contraria para conseguir que la orientación final del Agente sea la contraria a la del *Target*.

Velocity Matching: Calcula la diferencia entre las velocidades del Agente y *Target* y lo limita a la máxima aceleración del Agente.

4.3 Steerings Delegados.

Pursue: Hereda de **Seek**, y calcula una corrección en la posición del *Target* en función de una predicción de movimiento.

Face: Este **Steering** hereda directamente de **Align**, y calcula la orientación final como la orientación hacia el objetivo.

Wander: Hereda de **Face**, obteniendo su velocidad angular, y a su vez obtiene la velocidad lineal en función de su aceleración y de su orientación. Calcula un objetivo ficticio que se sitúa en los márgenes de una circunferencia imaginaria, y aplica **Face** hacia esa posición.

PathFollowing: El steering sigue un Path establecido avanzando de *targetParam* en *targetParam* las posiciones del **Path**. Hereda de **Arrive** y calcula un *Target* ficticio como cada posición del Path.

4.4 Steering Compuesto (árbitros).

Mezcla Ponderada: El sistema de árbitros se basa en una clase llamada **BlendedSteering** que combina los steerings que se quieren usar aplicando un peso a cada uno. Esta clase hereda directamente de **SteeringBehaviour** y se trata desde el agente como el único **Steering** activo.

4.5 Steerings coordinados o Steerings en grupo.

Separation: Permite que los personajes mantengan una distancia mínima entre ellos. Para ello calcula la distancia con cada uno del resto de personajes y aplica una corrección a su movimiento alejándose de ellos si su distancia es menor.

Alignment: Mediante los vectores cabecera de los personajes, basados en su orientación, aplica una desviación en su orientación para que todos vayan miren en la misma dirección.

Cohesion: Al contrario que **Separation**, **Cohesion** intenta hacer que los personajes no se alejen demasiado. Calcula un centro de masas mediante la posición del resto de personajes y se dirige a ese centro.

Flocking/Bandada: Este steering combina los 3 comportamientos anteriores, con preferencia ***Separation***>***Cohesion***>***Alignment*** y además aplica un **Arrive** hacia un Target invisible, el cual se mueve de forma aleatoria mediante un **Wander**. Hereda directamente de **Blended Steering**, que actúa como árbitro. Con este **Steering** de grupo se obtiene un comportamiento que simula una bandada, en la que los personajes van juntos hacia la misma dirección sin solaparse.

4.6 Formaciones.

Formaciones Fijas: Las formaciones se basan en situar a los personajes en torno a un líder que no forma parte directa de ella. Para realizar las formaciones se trabaja con 3 clases básicas, **Formation**, **Form** *y* **Slot**.

- **Formation** es una clase que calcula un Target, que es hacia donde deben ir los personajes. Para ello aplica una matriz de orientación según su posición concreta dentro de la formación.

- **Form** es el **SteeringBehaviour** que se encarga de mover a los personajes en función del Target obtenido con la clase **Formation**. Es la clase padre de cada tipo de formación, y mediante un identificador numérico crea un tipo de formación u otro.

- **Slot** es la clase de apoyo que le indica a cada personaje cual tiene que ser su posición y su orientación relativa al líder de la formación.

Line: Esta formación sitúa a los personajes en una línea recta, mirando en la dirección a la que mira el líder. Para ello se sitúan en las posiciones (-1,0), (1,0) y (2,0) relativas al líder y con orientación 0.

Defensive circle: Esta formación hace que los personajes se coloquen formando un círculo, y cada personaje mira en una dirección distinta, hacia fuera. Se colocan en las posiciones (-1,-1), (1,-1) y (0,-2), con orientaciones -90, 90 y 180 respectivamente.

4.7 Algoritmo de Pathfinding LRTA*.

LRTA es un algoritmo en tiempo real que intenta obtener el camino más corto (siguiendo un grid) hacia una posición determinada. Para ello recorre los nodos adyacentes al nodo actual y avanza hacia el nodo de menor coste, actualizando el coste del nodo anterior sumándole 1.

Hemos implementado detección inmediata de obstáculos, así como un límite máximo de coste para bloquear el movimiento hacia zonas inaccesibles o demasiado costosas. En caso de no poder ir a una posición determinada, el algoritmo devuelve un Path cuya única posición es el nodo actual.

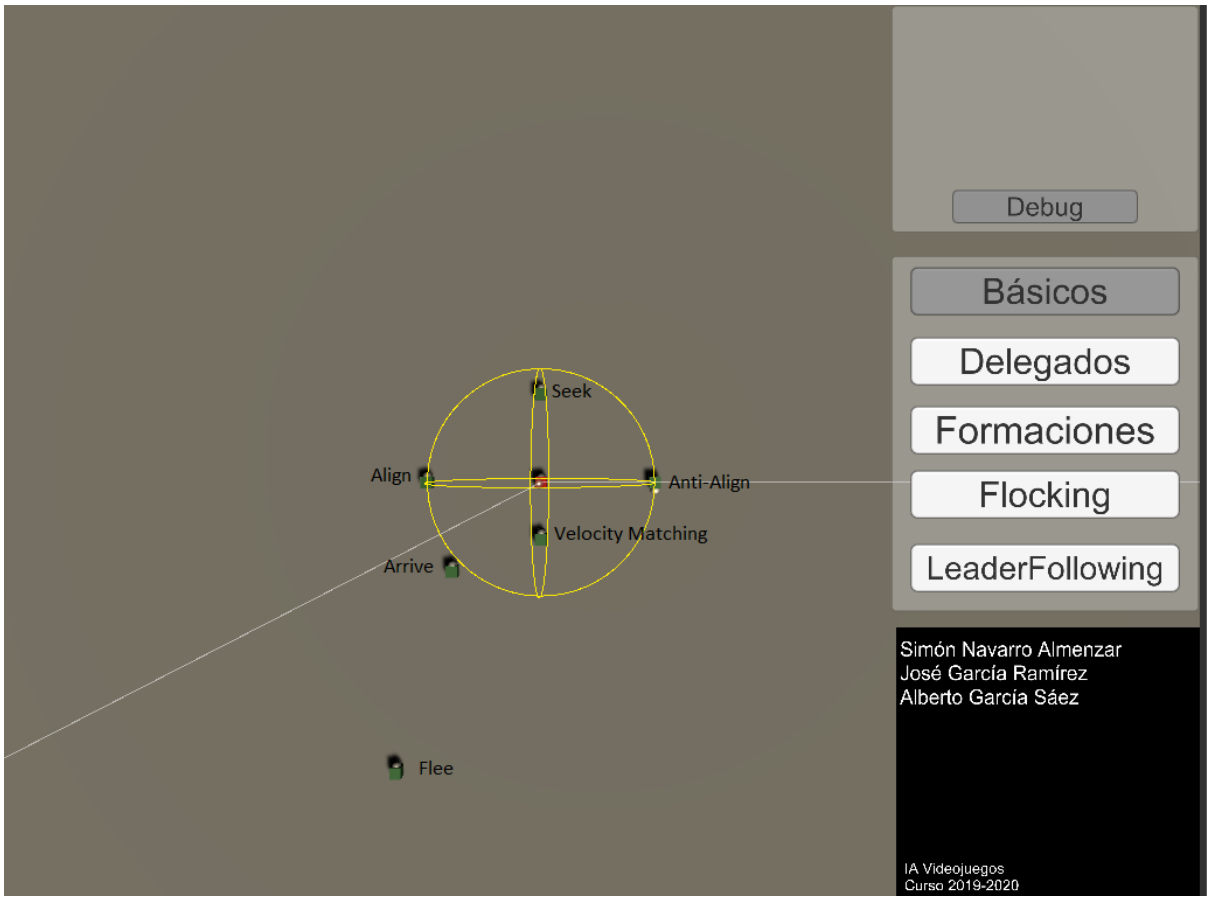
El **PathFinding** necesita, antes de hacer cálculos, inicializar un grid. Esto lo hace mediante una heurística basada en la *distancia de Manhattan*, y le aplica una variación en función del terreno en el que se sitúa el nodo, de manera que los personajes con preferencia sobre algunos terrenos tratan ese nodo con coste menor que los demás, y de la misma manera los terrenos a los que es reacio los trata como nodos con demasiado coste.

Para la segunda parte también hemos implementado detección de influencia. Si los nodos evaluados tienen influencia del equipo contrario, no se contemplan como opción. También existe una versión de este algoritmo que no trata influencia, para casos determinados.

Por último, el algoritmo trata de minimizar el **Path** creado buscando repeticiones de patrones en el **Path** y eliminándolas de éste.

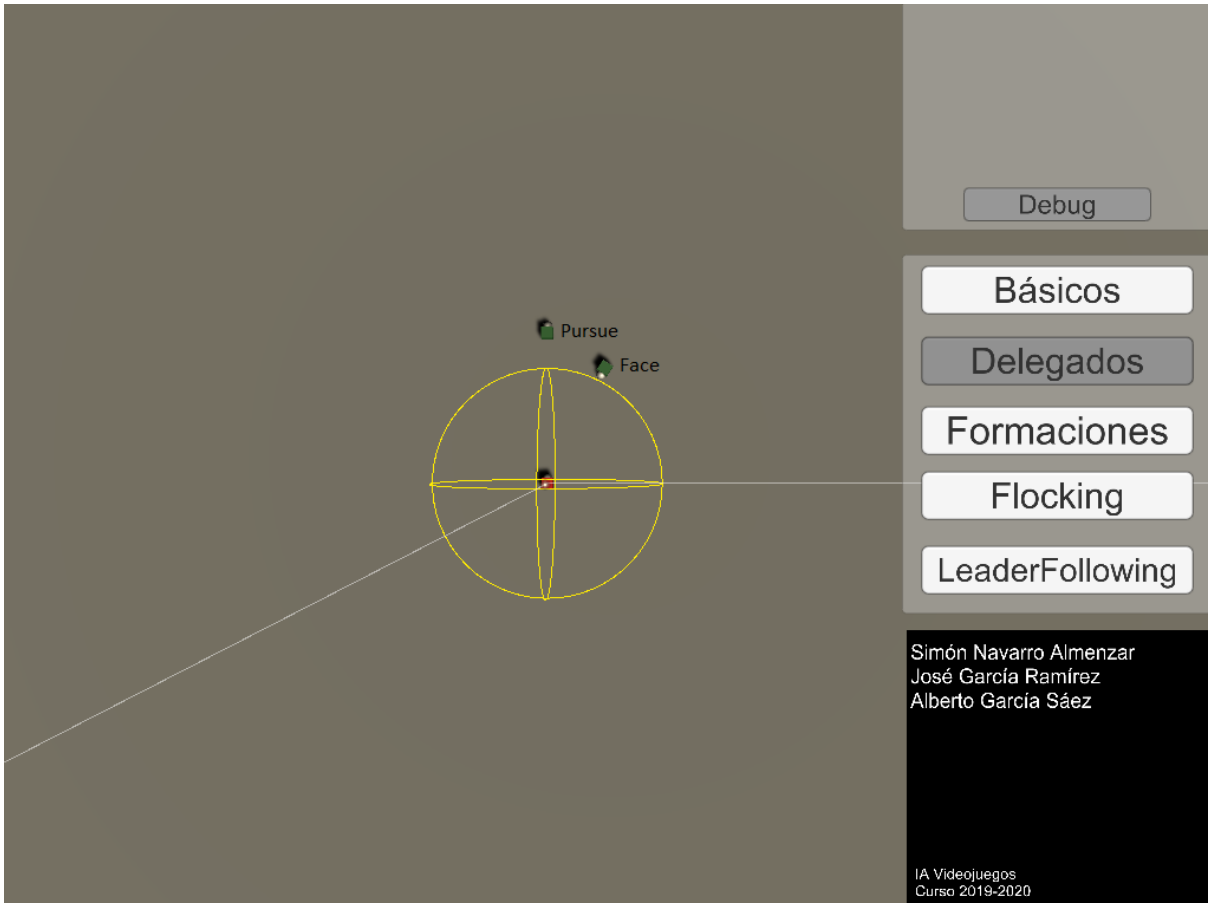
Nuestro algoritmo no funciona en tiempo real porque no conseguimos familiarizarnos con las corutinas, así que decidimos implementarlo de esta forma.

4.8 Escenas de prueba del Bloque 1.



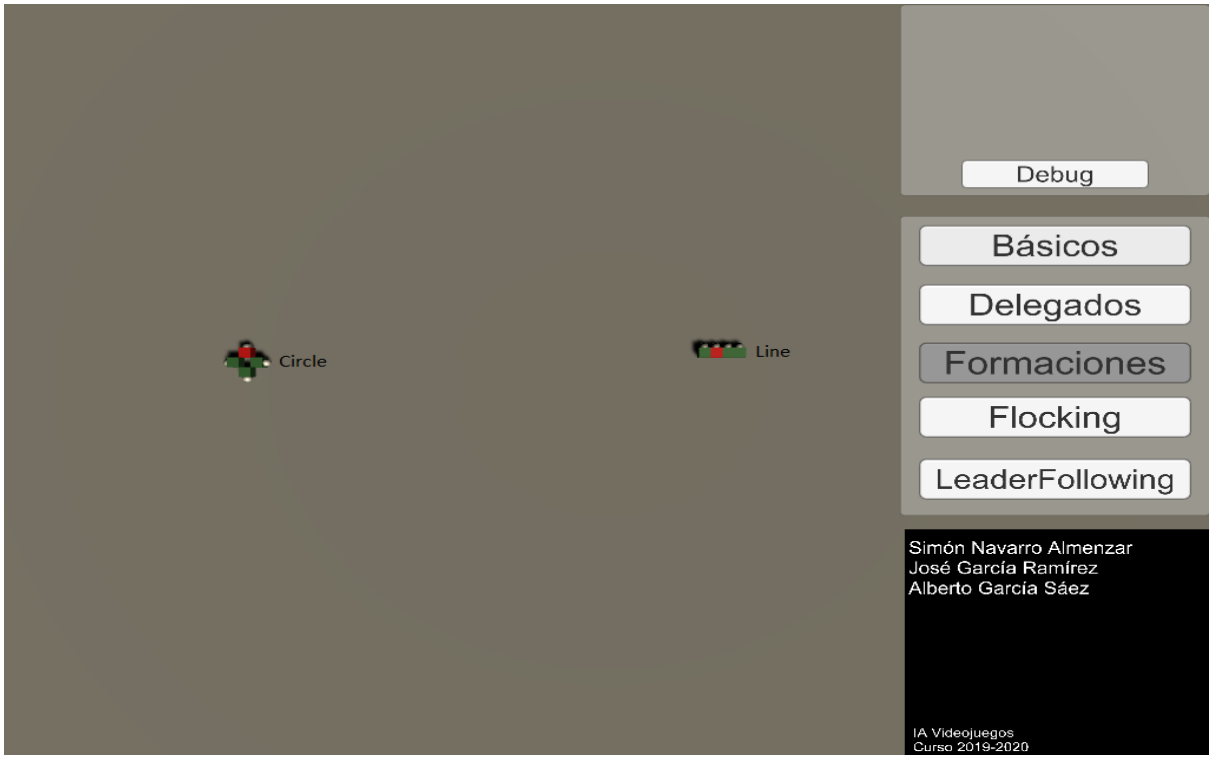
En esta escena podemos observar:

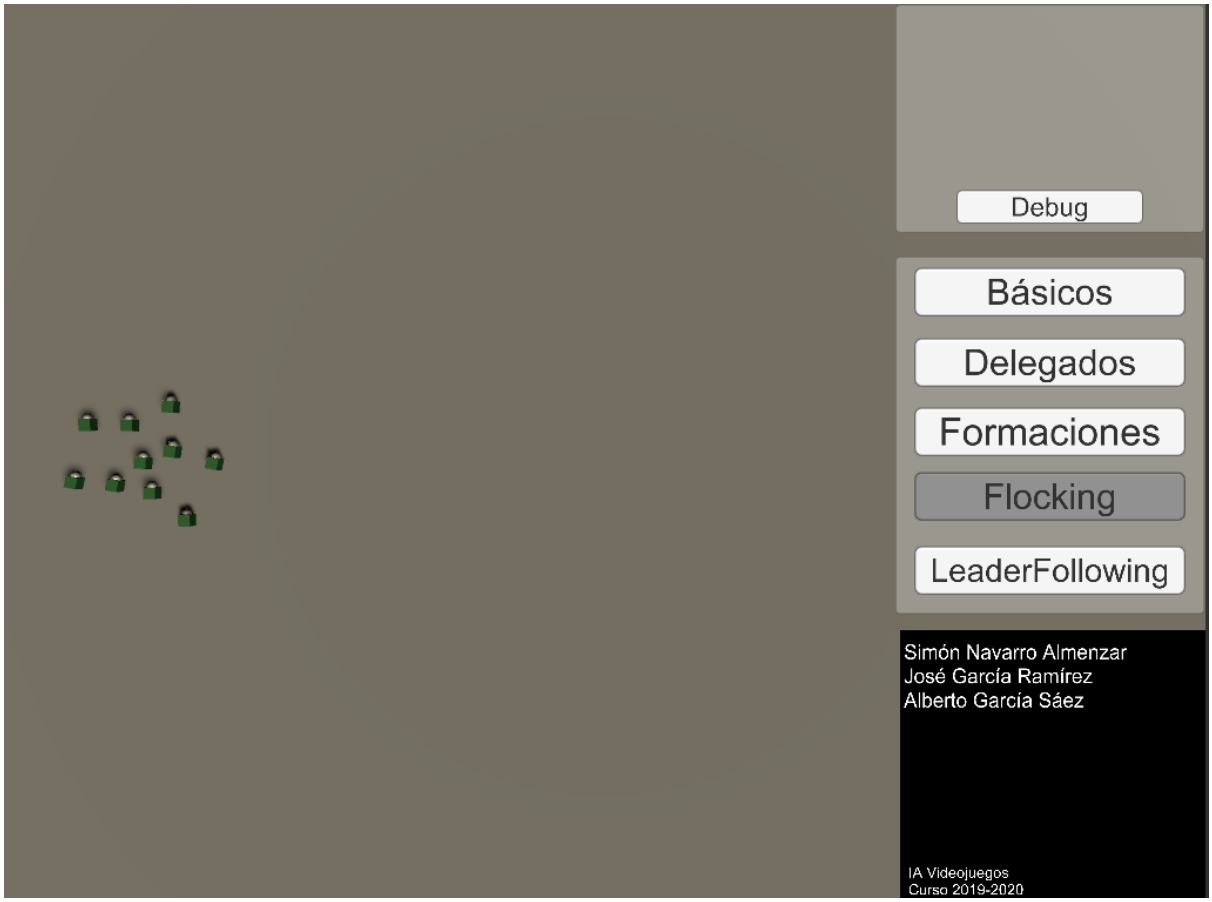
- **Seek** avanza hacia el objetivo.
- **Arrive** avanza hacia el objetivo y al entrar en el radio exterior (amarillo) comienza a decelerar.
- **Align** mira hacia donde mira el objetivo.
- **Anti-Align** mira en dirección contraria a donde mira el objetivo.
- **VelocityMatching** copia la velocidad del objetivo.
- **Flee** huye en dirección contraria a donde está el objetivo.



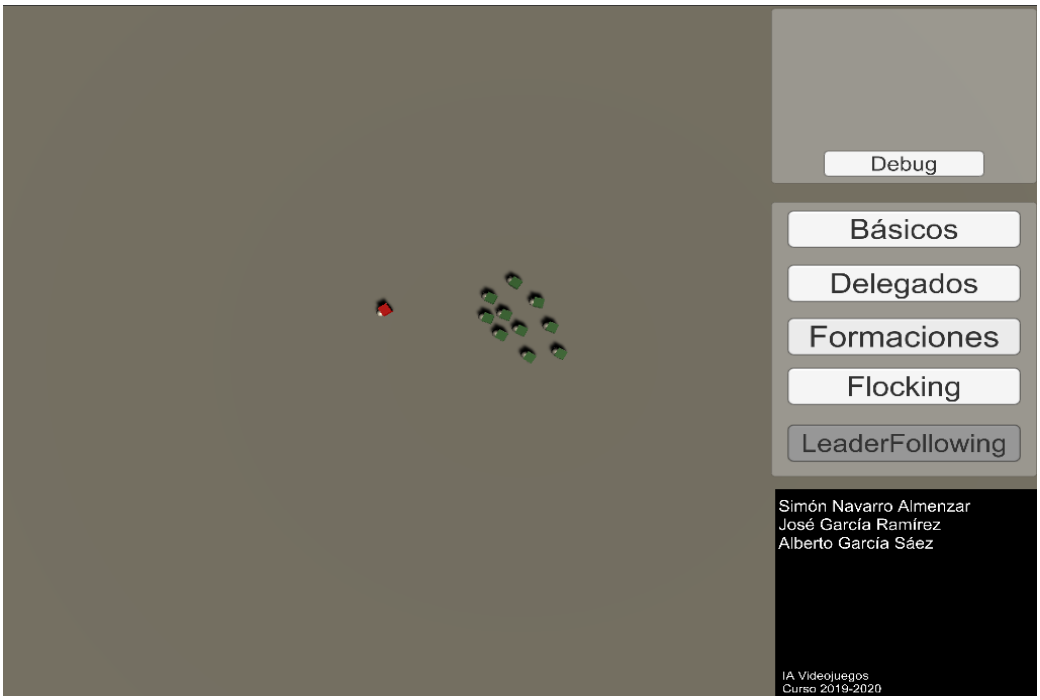
Aquí podemos ver:

- **Face** mira en dirección al objetivo.
- **Pursue** avanza hacia el objetivo.

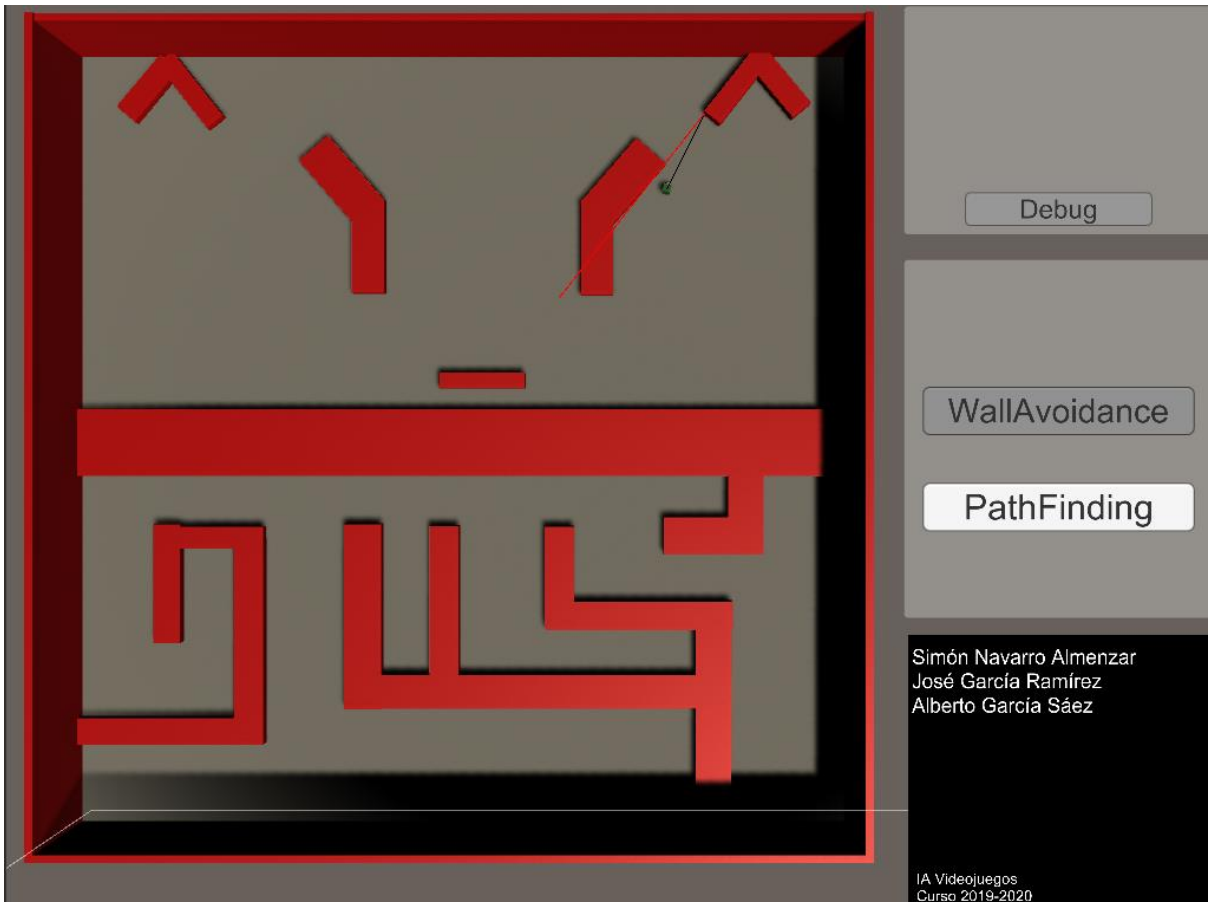




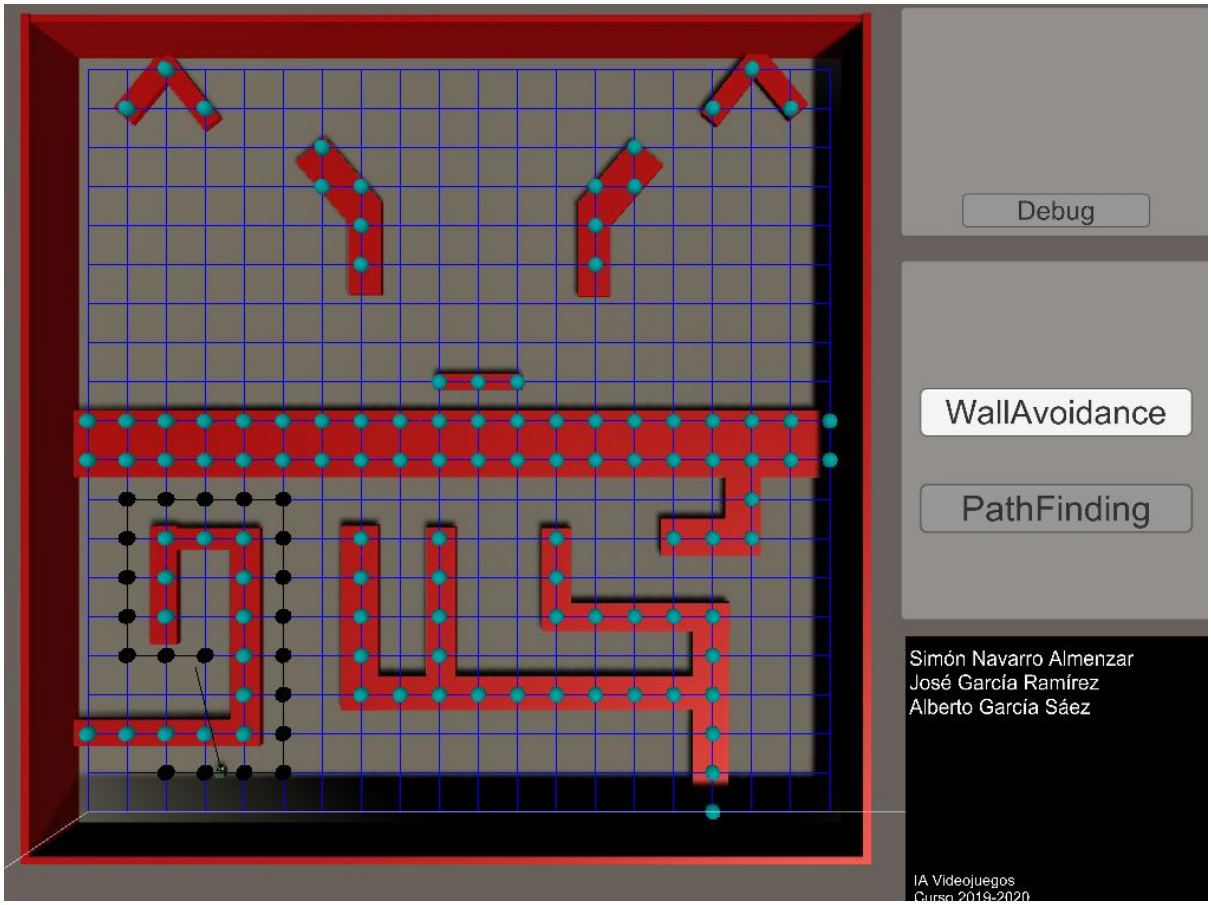
Esta escena muestra cómo los personajes avanzan hacia una misma dirección manteniendo distancias.



Aquí podemos observar cómo los personajes van siguiendo todos a un líder manteniendo distancia.



El NPC trata de evitar acercarse al muro y se muestra la normal del punto de impacto.



En esta escena podemos observar el camino obtenido mediante **PathFinding**, así como el **Grid** y la detección de obstáculos del mapa.

Todas estas escenas se demostrarán en la exposición de una manera más clara, pues mediante fotografías es más complicado observar lo que está sucediendo.

4.9 Opcionales.

4.9.1 Presentar mejoras sobre los steering obligatorios.

- Mejora sobre PathFollowing

Una mejora implementada ha sido la detección de esquinas del **Path** al ejecutar **PathFollowing** para evitar balanceos por exceso de velocidad. Para ello hemos decidido establecer como Targets las posiciones del **Path**, de manera que si detectamos una esquina aplicamos un radio de frenada a ese Target pero si está en una línea recta no se aplica deceleración. Para ello también ha sido necesario que **PathFollowing** herede de la clase **Arrive** haciendo que los personajes tengan movimientos mucho más controlados.

4.9.2 Steerings basados en velocidad, delegados y en grupo.

Steering Uniforme:

- **Seek**
- **Flee**
- **Arrive**
- **Wander**

Los steerings basados en velocidad funcionan igual que con aceleración, pero aplican en todo momento la velocidad máxima, en vez de una aceleración progresiva.

Steering Delegado:

-**LookWhereYouGoing:** Este **Steering** hace que tu personaje mire en la dirección en la que efectúa el movimiento, para ello es necesario hacer uso de **Align**, que obtiene la orientación futura de nuestro personaje a partir de la tangente del vector velocidad del mismo.

Steering en grupo:

-**LeaderFollowing**: Este **Steering** intenta simular que los personajes sigan a un líder huyendo de él previamente si están en su camino. Los personajes tienen un líder invisible al que siguen mediante un **Arrive**, se separan utilizando **Separation**, e intentan huir del líder principal mediante **Evade**, con esa preferencia, utilizando la clase **BlendedSteering** como actuador.

4.9.3 Modo depuración.

El modo depuración muestra detalles de velocidad y rotación de los personajes en un cuadro en la esquina superior derecha, así como rango de sus radios.

Para el caso de **ObstacleAvoidance**, muestra también la detección de los muros y la normal aplicada, y para el caso de **PathFollowing** muestra el **Grid**, el **Path** que está siguiendo y una línea hacia la posición final.

El juego consta de un menú inicial que contiene instrucciones para utilizar este modo Debug.

Es importante resaltar que este modo Debug funciona a base de Gizmos, por tanto no se verá desde el ejecutable, pero sí que se puede ver desde el editor de Unity si se activa la función de Gizmos.

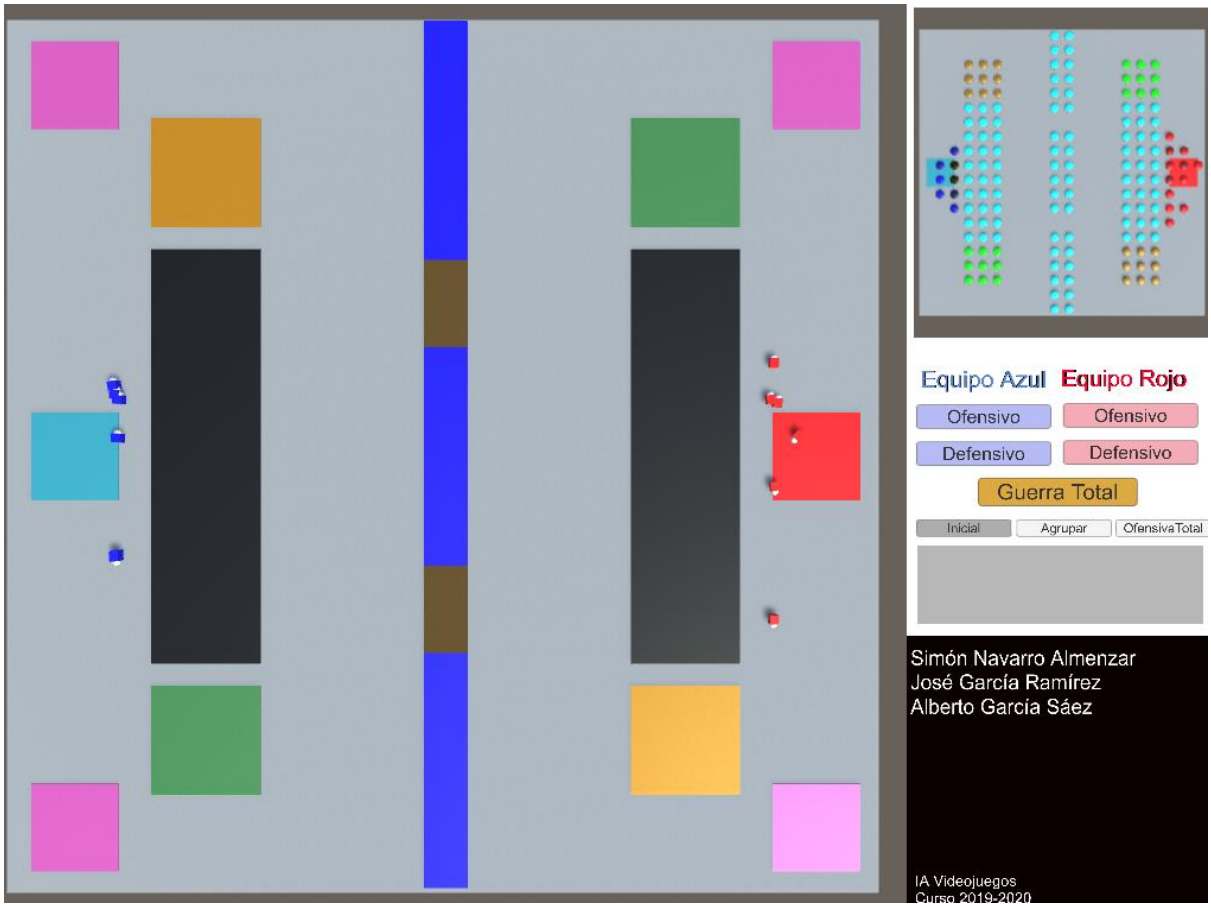
5. Bloque 2: IA Estratégica y táctica.

En este segundo bloque se explican todos los elementos utilizados para el desarrollo de este juego de guerra con Inteligencia Artificial, en el cual hay distintos personajes, con distintos roles y con sus respectivas preferencias a la hora de pasar por zonas del mapa, cuyo objetivo será el de poder controlar la base enemiga durante 10 segundos para alcanzar la victoria.

5.1 Interfaz y Controles

Para la interfaz de nuestro proyecto hemos creado un menú en el cual podemos ver diferentes opciones:

Podemos ver las **Instrucciones de nuestro juego** y la información de este mismo. Si pulsamos el botón comenzar, podremos elegir varias opciones relacionadas con el desarrollo de los dos bloques. Seleccionamos la parte 2 y se nos muestra el siguiente mapa con su respectiva interfaz que se explica a continuación:



Hemos seguido las recomendaciones de la asignatura y hemos creado un mapa con dos "países" separados por un río el cual tiene dos puentes por donde los agentes no controlados (NPCs), podrán pasar para atacar las bases enemigas (roja y azul). En la parte superior derecha se muestra el mapa de influencia que generan los propios NPCs y se muestran también distintos botones para que se pueda seleccionar el modo de juego y la táctica de ataque/defensa para que se desarrolle la guerra.

Para poder ver la **información de los NPCs**, basta solo con hacer click sobre ellos mismos y en la parte derecha del juego se mostrará su información respecto a qué tipo de personaje es (Mele, Ranged, Todoterreno o patrullero), su estado actual, su salud, el waypoint hacia dónde se dirige y finalmente su velocidad, y si está en un combate, la información acerca de su enemigo. Finalmente, si pulsamos la tecla Esc volvemos al menú principal y todo volvería a comenzar de nuevo.

Al iniciar el juego los dos equipos se colocan en posiciones defensivas por defecto, esperando una orden por parte del jugador. Una vez seleccionado un modo, no se puede cambiar por claridad de la ejecución. Sin embargo, sí que se puede cambiar de táctica durante la ejecución.

En el apartado Instrucciones del menú principal se muestra un mapa de las teclas que se pueden utilizar.

En la escena relativa a la Parte 1 hay implementado un algoritmo de cámara que permite controlar la cámara con W,A,S,D, rotarla con Q,E y hacer zoom con la rueda del ratón. Especialmente útil para ver el comportamiento de **Flocking** y **LeaderFollowing.**

5.1.1 Controladores

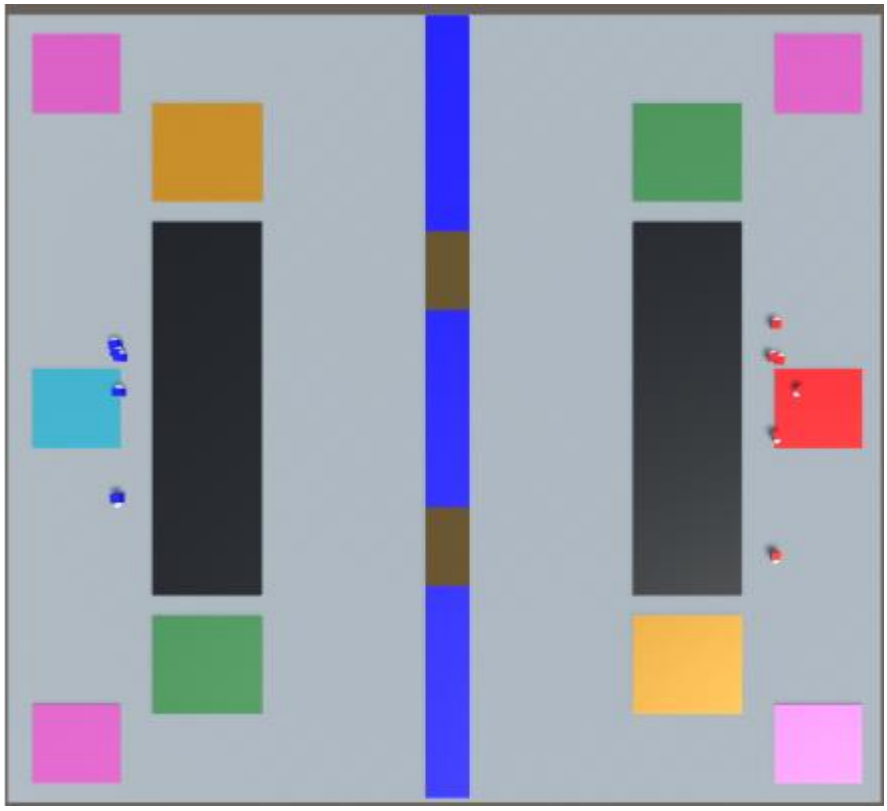
El juego consta con dos clases que actúan de manager del juego, gestionando información importante.

Manager: Gestiona todo lo relativo a la gestión de wayPoints, personajes muertos, input del sistema, cambios de modo, cambios de tácticas y modo debug, así como de toda la interfaz gráfica. Esta clase se encarga de la Parte 2 de la práctica.

ControladorParte1: Esta clase se encarga de todo lo relativo a los steerings de la Parte 1 de la práctica. Gestiona el modo debug y la interfaz gráfica.

5.2 Grid y mapa.

El mapa está dividido en las siguientes zonas:



En este mapa podemos observar diferentes lugares diferenciados por colores.

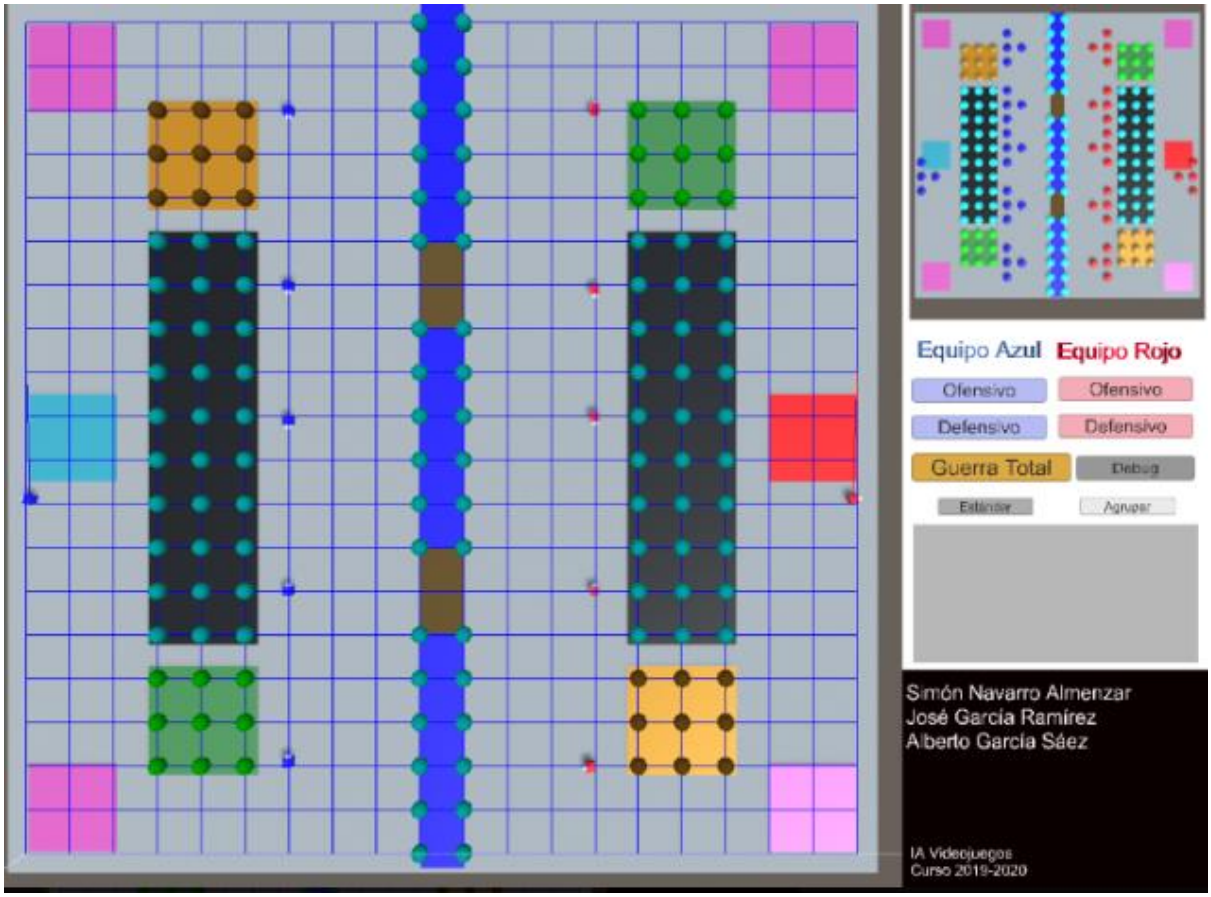
- La zona de bosque se representa con el color verde.
- La zona de tierra se representa con el color marrón.
- Las zonas negras son obstáculos que deben evitar los agentes.
- La base del equipo está representada con el mismo color de este.
- Las zonas de cura se representan con el color rosa, situadas en las esquinas del mapa.
- Se puede observar también el río que divide el mapa en dos "países" y sus dos respectivos puentes por donde cruzan los agentes.

Para la colocación de los waypoint en el mapa, hemos decidido colocarlos a mano, indicando sus distintas posiciones en la clase Manager. Un ejemplo de los waypoints defensivos es:

```
WayPointAzulDefensivo = new Vector3[5];
WayPointAzulDefensivo[0] = new Vector3(30, 0, 85);
WayPointAzulDefensivo[1] = new Vector3(30, 0, 65);
WayPointAzulDefensivo[2] = new Vector3(30, 0, 50);
WayPointAzulDefensivo[3] = new Vector3(30, 0, 30);
WayPointAzulDefensivo[4] = new Vector3(30, 0, 10);

WayPointRojoDefensivo = new Vector3[5];
WayPointRojoDefensivo[0] = new Vector3(65, 0, 85);
WayPointRojoDefensivo[1] = new Vector3(65, 0, 65);
WayPointRojoDefensivo[2] = new Vector3(65, 0, 50);
WayPointRojoDefensivo[3] = new Vector3(65, 0, 30);
WayPointRojoDefensivo[4] = new Vector3(65, 0, 10);
```

A continuación, se muestra el grid que se crea para el mapa, identificando las zonas de obstáculos y terrenos diferentes gracias al uso de un vector de tipo RAY, el cual lanza un rayo desde 5 puntos más alto que el terreno con dirección (0,-1,0) y así podemos identificar, comparándolo con el tag, qué tipo de terreno es:

```
ray.origin = new Vector3(node.Posreal.x, node.Posreal.y + 5, node.Posreal.z);
ray.direction = new Vector3(0, -1, 0);
RaycastHit[] hit = Physics.RaycastAll(ray, 10);

foreach (RaycastHit h in hit)
{
    if (h.transform.CompareTag("Pared"))
    {
        node.Obstacle = true;
    }
    else if (h.transform.CompareTag("Bosque"))
    {
        node.Bosque = true;
    }
    else if (h.transform.CompareTag("Tierra"))
    {
        node.Tierra = true;
    }
```



5.3 Equipos, unidades y comportamiento.

5.3.1 Equipos.

Hay dos equipos, un equipo azul y un equipo rojo. Los dos equipos son completamente simétricos y poseen:

- 2 **Meles**
- 2 **Rangeds**
- 1 **TodoTerreno**
- 1 **Patrullero**

Cada equipo tiene un modo ofensivo y un modo defensivo. En modo ofensivo los personajes tratarán de llegar a la base enemiga y en modo defensivo tratarán de adoptar posiciones estratégicas en su territorio para defender.

Existe otro modo, llamado Guerra Total, que actualiza los dos equipos a modo Ofensivo y con táctica **OfensivaTotal**.

5.3.2 Unidades.

En este apartado describimos cada uno de los cuatro tipos de unidades empleados en la práctica y su relación con respecto al resto de personajes y los terrenos.

- Ranged: Este es el personaje con mayor velocidad y aceleración máxima, además posee una vida base menor al resto (ya que es un personaje mejor que el resto gracias a su rango), es un personaje que prefiere ir por tierra y es penalizado al ir por zonas de bosque, además es eficaz sobre personajes Todoterreno, intermedio sobre Ranged y tiene un daño reducido sobre personajes Mele.

- Mele: Este personaje, junto al personaje TodoTerreno, tiene vida máxima superior a Ranged, prefiere ir por zonas de bosque y es penalizado al ir por zonas de tierra, además es eficaz atacando a personajes Ranged, intermedio a Mele y reducido a personajes TodoTerreno.

- TodoTerreno: Este personaje tiene vida máxima igual al Mele, además no tiene zona predilecta por la que ir, es eficaz sobre personajes Mele, intermedio sobre TodoTerreno y tienen un daño reducido sobre personajes Ranged.

- Patrullero: Este personaje es el más especial de todos (no es un personaje en sí), cualquiera de los tres tipos de personajes descritos anteriormente puede ser un patrullero ya que se le aplica el tipo patrullero por medio de tags de Unity. En el caso de nuestra práctica decidimos que el Patrullero fuera un personaje Mele.

Factor de tipo de atacante/defensor (FAD)

Rol/Atributos	Ranged	TodoTerreno	Mele
Ranged	x1	x1.25	x0.75
Todoterreno	x0.75	x1	x1.25
Mele	x1.25	x0.75	x1

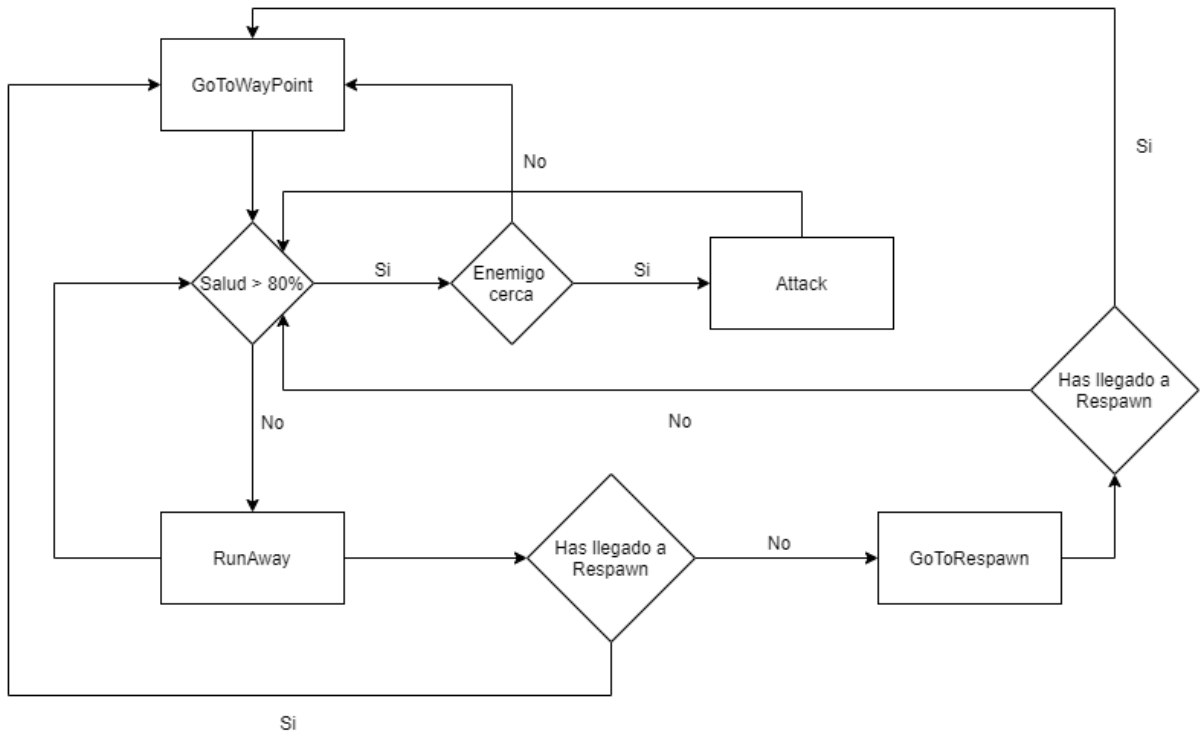
Factor por terreno del atacante (FTA)

Terreno/Unidad	Ranged	TodoTerreno	Mele
Suelo	x1	x1	x1
Bosque	x0.1	x2	x0.25
Tierra	x2	x1	x1

Factor por terreno del defensor (FTD)

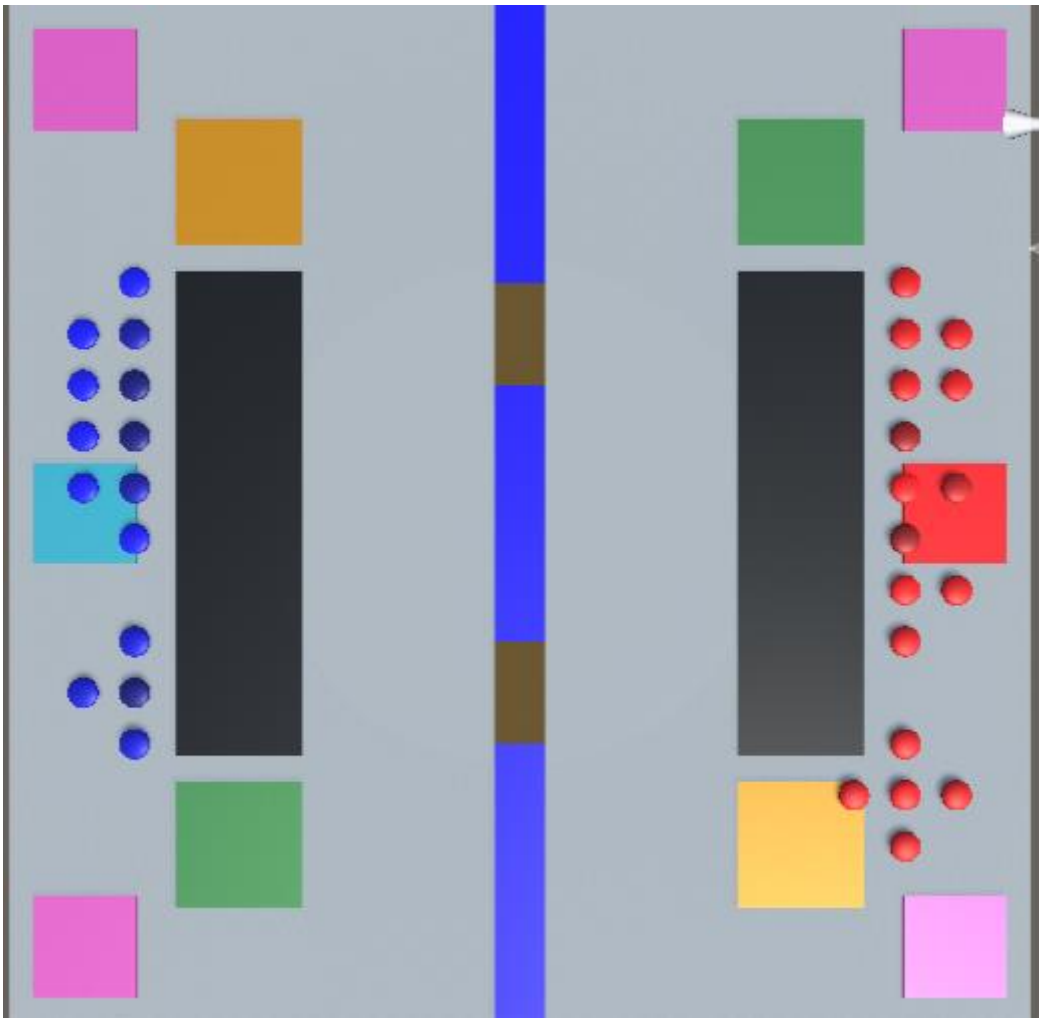
Terreno/Unidad	Ranged	TodoTerreno	Mele
Suelo	x1	x1	x1
Bosque	x1.25	x2	x0.5
Tierra	x0.75	x0.5	x1

5.3.3 Comportamiento



- **GoToWayPoint**: Los personajes avanzan hacia los wayPoints establecidos. Según la táctica empleada, estos cambian.
- **GoToRespawn**: Cuando un personaje muere, ha de ir primero a la posición donde murió.
- **Attack**: Al encontrarse a un enemigo, empieza una batalla.
- **RunAway**: Cuando el personaje posee menos del 20% de la vida tiene que huir hacia la zona de cura de su equipo más cercana.

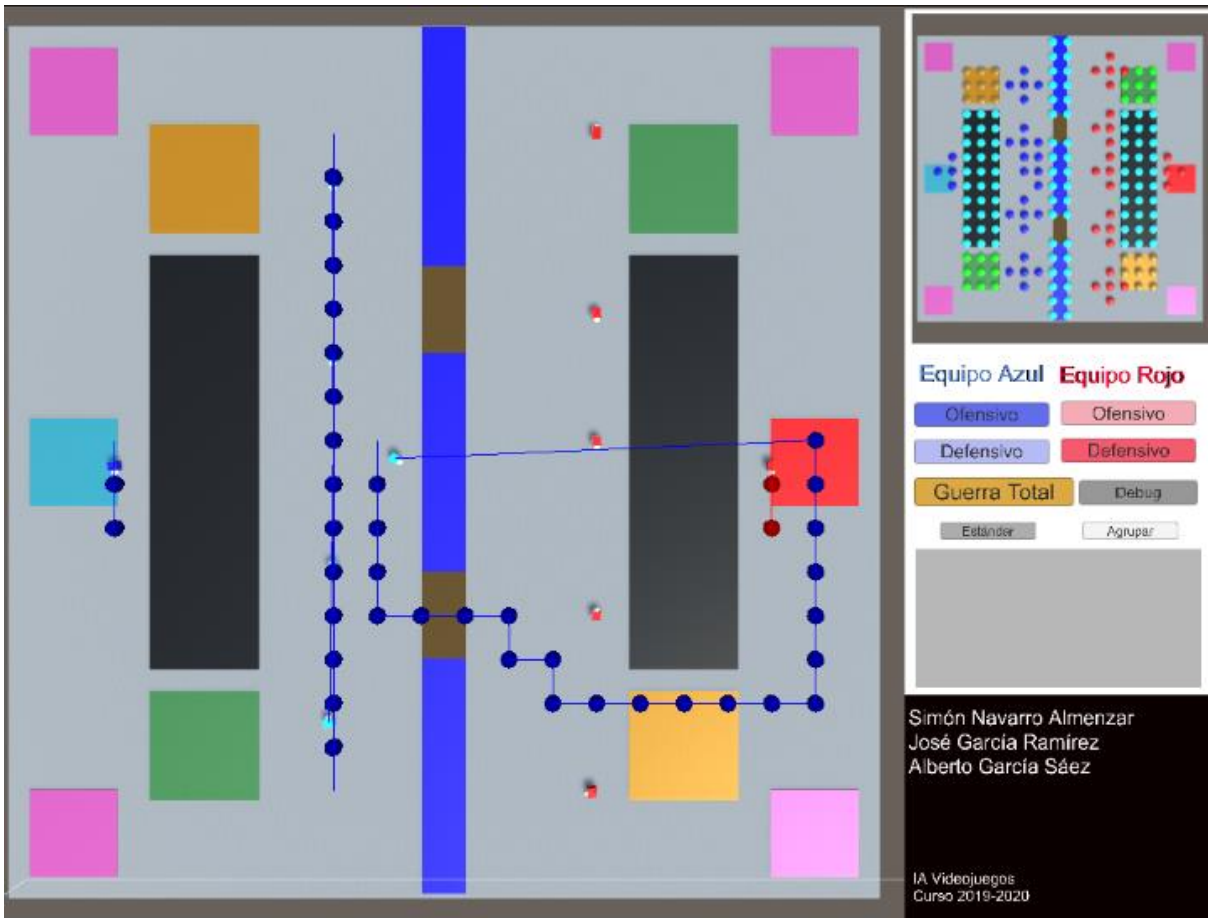
5.4 Mapa de influencias.



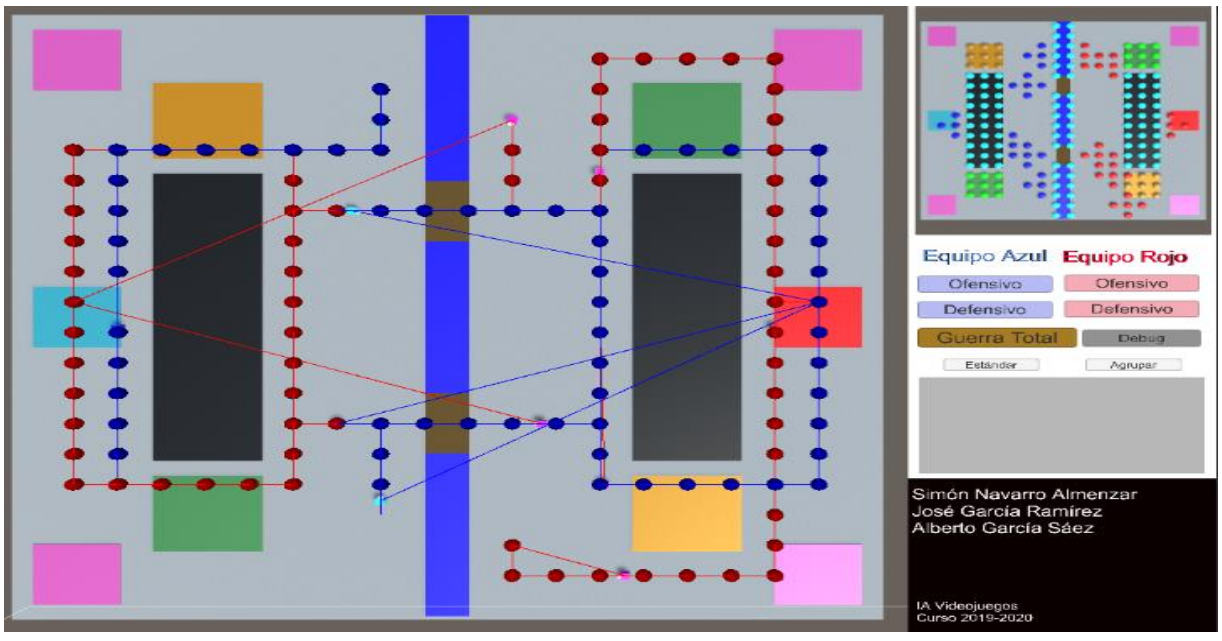
El minimapa proporciona una representación gráfica del mapa de influencia. Un personaje que no está rodeado por obstáculos genera influencia en 5 nodos. El suyo propio, y una influencia de la mitad en los 4 nodos adyacentes. Esta información se actualiza cada 2 segundos.

Cada nodo guarda información acerca de la influencia de cada equipo, información que más tarde utilizará el **PathFinding** táctico para decidir qué nodos visitar y cuáles no.

Podemos ver un ejemplo de cómo los personajes utilizan esta información para intentar evitar los nodos con influencia enemiga:

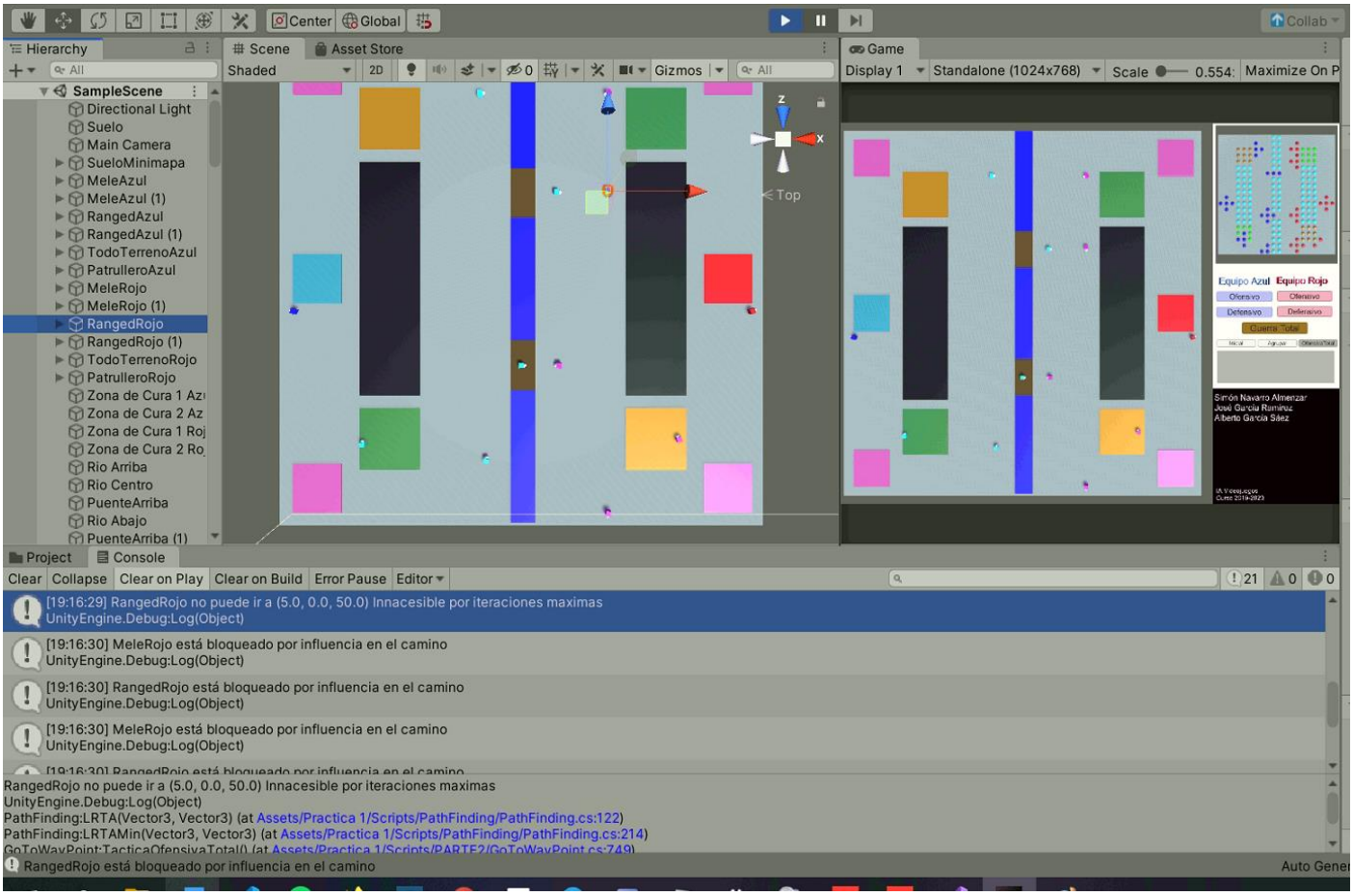


Podemos ver la diferencia de este comportamiento respecto a un comportamiento sin influencia cuando ejecutamos la táctica OfensivaTotal, que ignora influencia:



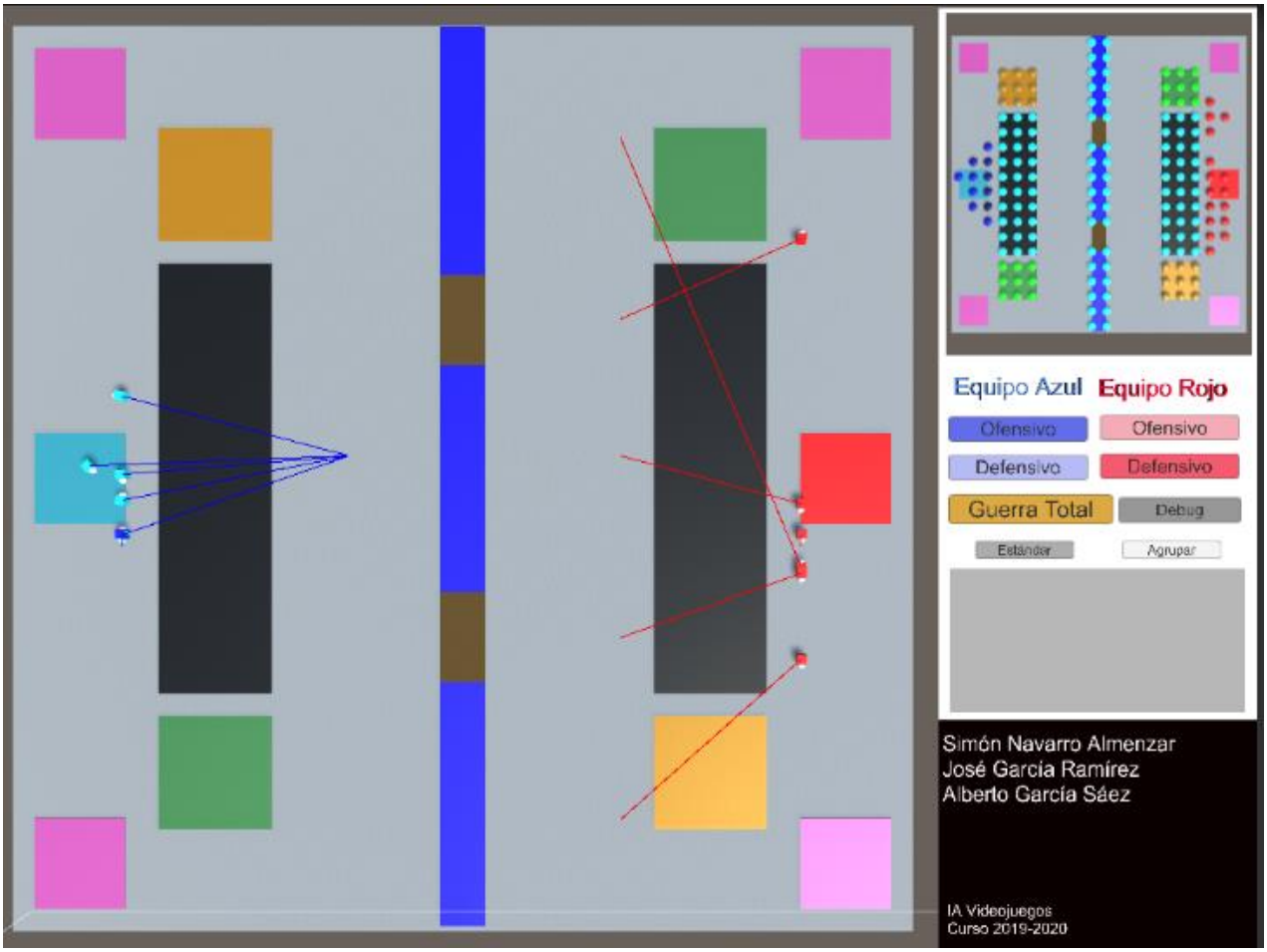
5.5 Pathfinding táctico.

En esta imagen podemos observar cómo los dos puentes tienen influencia azul, de manera que los personajes rojos cuando calculan el PathFinding no pueden encontrar ningún acceso hacia la base enemiga, así que mediante mensajes de Debug muestra que se han agotado las comprobaciones máximas de los costes de los nodos y se bloquean hasta que el mapa de influencia se actualice.

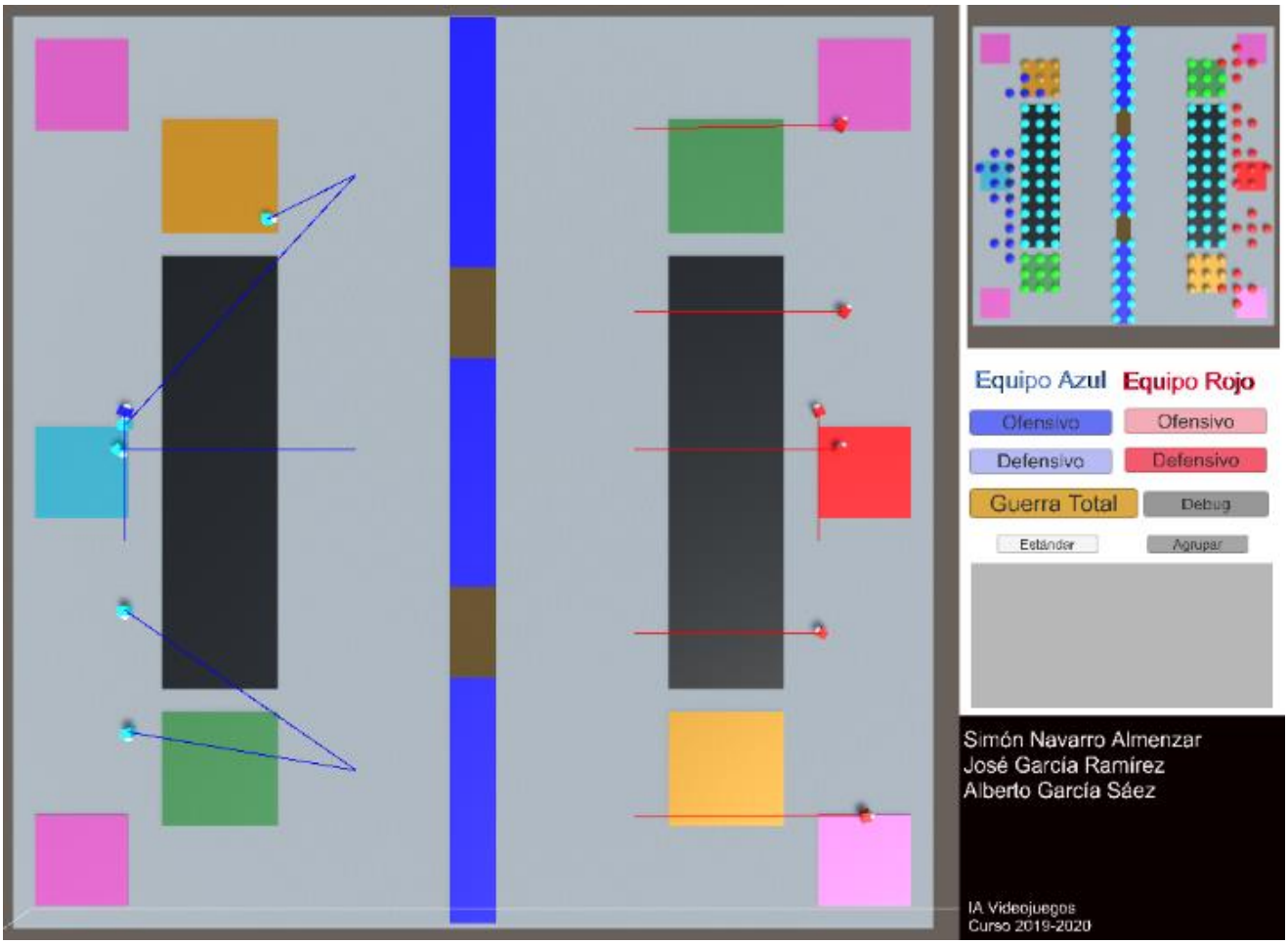


5.6 Tácticas.

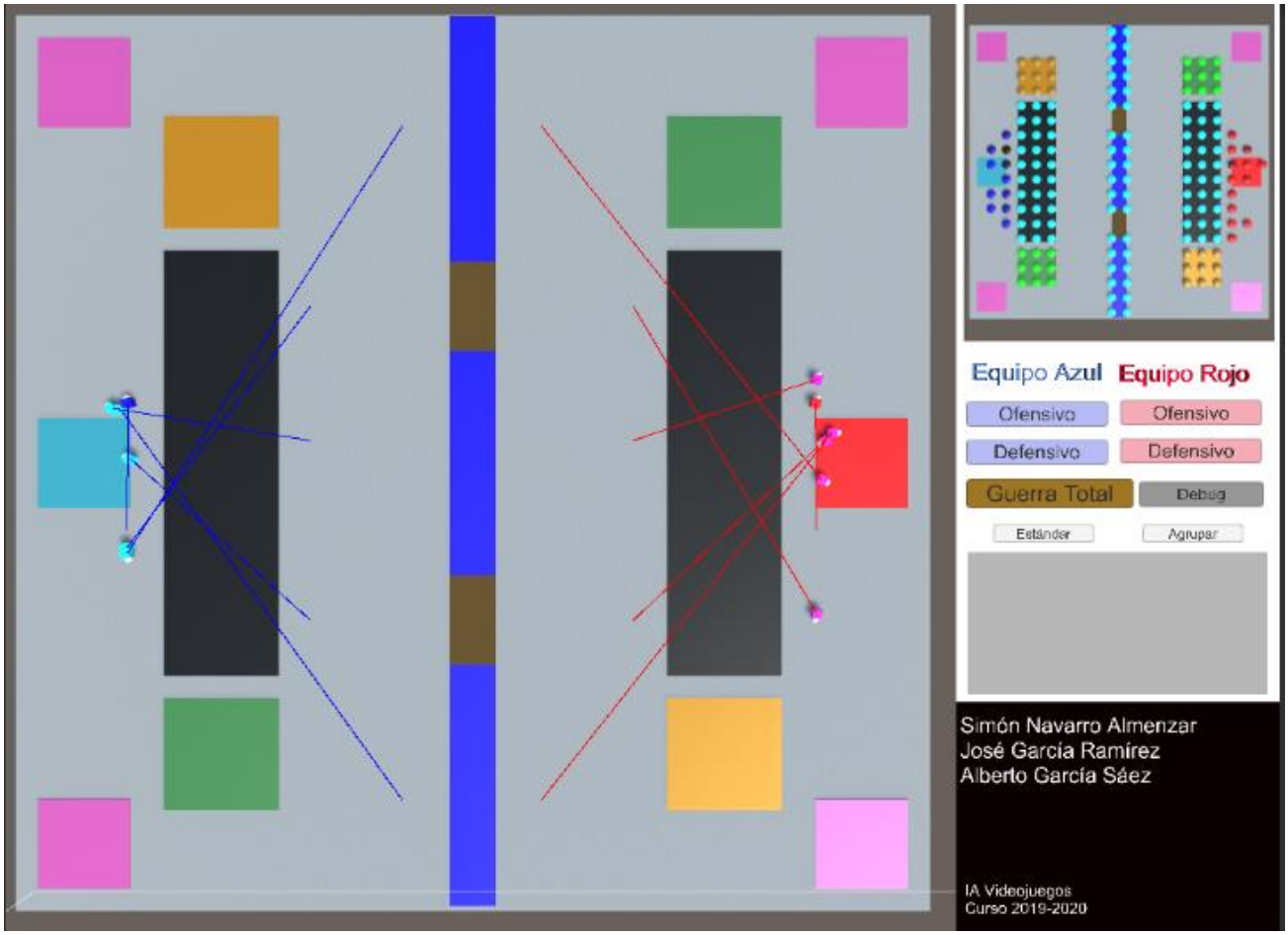
Estándar: Los personajes tratan de ir a un punto común a su propio ritmo y a continuación avanzan hacia la base enemiga.



Agrupar: Cada tipo de personaje se reúne en un punto. Cuando están todos los personajes de un tipo reunidos, avanzan hacia la base enemiga.



Ofensiva Total: Todos los personajes se reúnen en diversas posiciones de su territorio. Cuando todos han llegado, avanzan a la vez hacia la base enemiga. Ignora influencia enemiga. Es un ataque a la desesperada.



5.7 Opcionales.

5.7.1 Implementar otras propiedades de información táctica y usarlas.

Hemos implementado que los personajes utilizan unos wayPoints u otros en función de su tipo de personaje. Esto se utilizará en el apartado 4.7.3.

5.7.2 Implementar ataques coordinados/especializados usando información táctica.

Hemos realizado 3 tácticas diferentes que coordinan los personajes antes de realizar el ataque. (Ver 5.6)

5.7.3 Implementar pathfinding táctico a nivel de grupo.

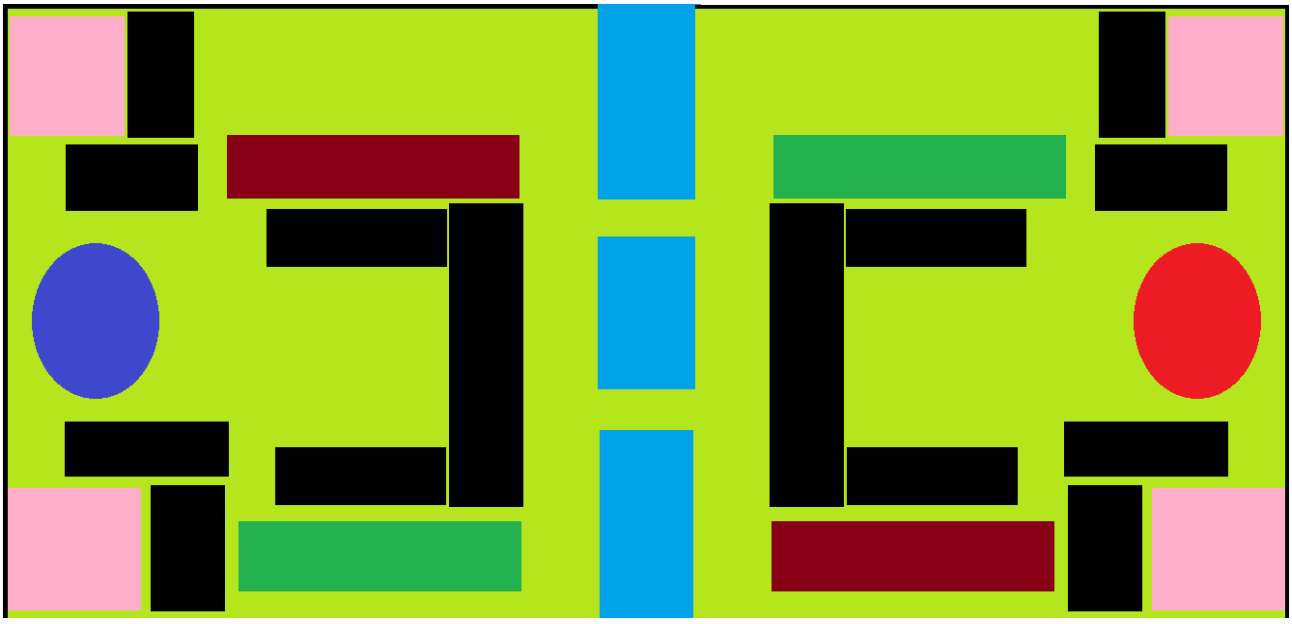
Las tácticas OfensivaTotal y Agrupar tratan de colocar los personajes en diversos puntos y no avanzar hasta que no estén todos colocados.

6. Anotaciones varias

Algunas anotaciones que hemos conservado durante el desarrollo de la práctica con ideas para realizarla:

```
d) idea> cada nodo tiene prop bool que al inicilaizar el grid que indica zonas de interes del mapa.

/**
Tras inicializar el grid, llamar a funcion de establecer propiedades tacticas que ponga los nodosX.informacionTacticaX = true. Para ello crear listas de WayPoints en Manager
que guarden las posiciones con esa información táctica. Por ejemplo, Vector3[] nodosCobertura = [(5,0,10), ((10,0,40))]....
**/

e) tacticas> si pulsas boton tactica actula igual a x, entoces el go to weypoint en funcion de la tactica los waypoint cambian dependiendo de la tactica

/**
Al pulsar un botón determinado, la táctica cambia. GoToWayPoint tiene que comprobar la táctica actual y a partir de ahí ejecuta LRTA sobre unos WayPoints u otros.
- Táctica agrupar: Cada tipo de personaje se reune en un lugar. Cuando están todos, avanzan el wayPoint hacia la base enemiga.
- Táctica ofensivatotal: Ejecutar LRTA sin influencia hacia la base enemiga. Hay que abrir los cuadrados y poner un camino en medio custodiado por un defensor. Con esta táctica
todos intentarán ir por el centro hacia la base enemiga y se enfrentarán al defensor todos juntos.
- Táctica inicial: Es la táctica con la que empieza el juego. Todos los personajes intentan ir a un punto de su base (a su propio ritmo) y después todos van a la base según sus
preferencias de terreno.
**/

a) idea> si eres ranged tu waypoint sera una zona abierta, si eres mele zona cerrada y si eres todoterreno igual, compruebas waypoints y su influencia y decides a donde ir

/**
Incluido en e). La táctica agrupar hace que los personajes vayan a wayPoint distintos en función de su tipo de personaje.
**/
```

Ideas para opcionales y tácticas



Idea de mapa inicial

7. Conclusiones.

Realizar esta práctica nos ha llevado mucho tiempo, especialmente por las dificultades especiales relacionadas con la pandemia del COVID-19. Nos hemos sentido perdidos durante gran parte del desarrollo y hemos tenido que emplear más tiempo del que pensábamos. Creemos que la práctica es demasiado larga, es la práctica más larga que hemos realizado durante el grado, con diferencia, y muchos alumnos deciden no realizar esta asignatura por la dimensión de la práctica.

Como posibles mejoras, en primer lugar, como hemos mencionado, recortar la práctica. Otro detalle que facilitaría mucho el asunto sería facilitar más documentación útil sobre desarrollo de videojuegos, uso de componentes en unity, lenguaje, etc.

Respecto a la parte 2 de la asignatura, se valoraría un poco menos de ambigüedad en los requisitos. Si bien la libertad a la hora de desarrollar proyectos únicos es importante, hay algunos requisitos que no quedan demasiado claros. Por ejemplo, los requisitos opcionales de la parte 1 están bastante claros, como “implementar otros delegados y en grupo”, sin embargo, en la parte 2 se pide “Implementar otras propiedades de información táctica y usarlas”, lo cual vemos un concepto demasiado amplio y que no sabíamos por dónde abordar.

Como conclusión final, la parte 2 es más interesante y llevadera que la parte 1, en parte gracias a la libertad de desarrollo que permite, pero los requisitos de la parte 1 están bastante mejor explicados que los de la parte 2. Dar flexibilidad no tiene por qué provocar ambigüedad. Aun así, hemos disfrutado la asignatura, hemos aprendido bastante y es probable que dediquemos tiempo a profundizar en este tema.

8. Bibliografía.

BillWagner. (n.d.). Documentos de C#: Inicio, tutoriales y referencias. Retrieved from https://docs.microsoft.com/es-es/dotnet/csharp/

Technologies, U. (n.d.). Welcome to the Unity Scripting Reference! Retrieved from https://docs.unity3d.com/ScriptReference