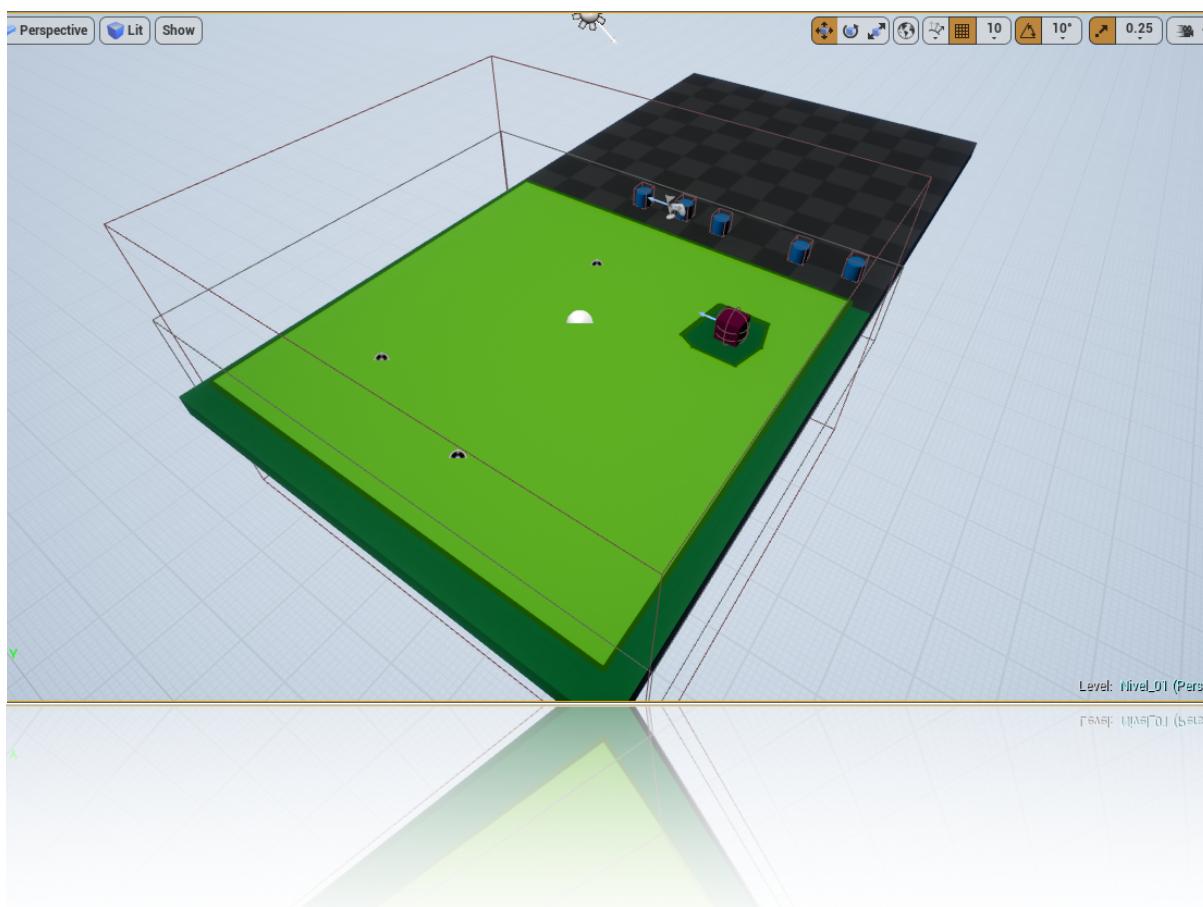


Persecución IA - UE4

Diseño de Pre-Producción y Concept-Art



Nicolás Tapia Sanz

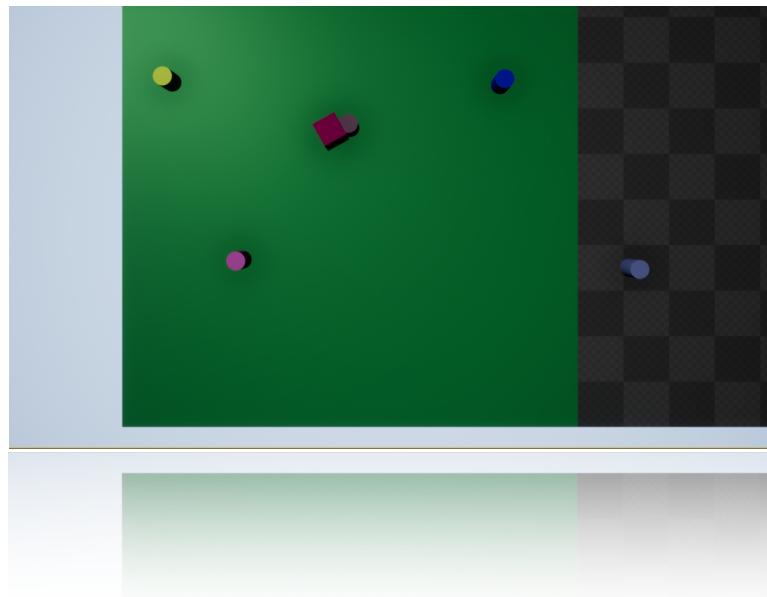
3.3. - DVJU

ESNE 2017/2018

Persecución IA - UE4

Diseño de Pre-Producción y Concept Art

Introducción



A continuación se procede a explicar el diseño de la práctica 2 de la asignatura de Diseño de Pre-Producción y Concept Art llamada Persecución IA, desarrollada en Unreal Engine 4, a nivel de desarrollo, funcionamiento y planteamiento.

Objetivo y Motivación

El objetivo de esta práctica es la implementación de una escena basada en la utilización de árboles de comportamiento en la que un agente sigue una ruta predefinida en ella.

En dicha escena también habrá varios elementos que se pueden arrastrar con el cursor del ratón y que, si entran en la zona de influencia de el agente, éste abandonará su ruta y perseguirá al elemento.

Si uno o más elementos se encontrasen en la zona de influencia del agente, éste siempre seguirá al primero que haya entrado. En caso de que este saliera, seguirá al segundo que entrara y así sucesivamente, siempre siguiendo al elemento que más tiempo lleve en la zona.

En caso de que todos los elementos desaparezcan de su zona de influencia, el agente retomará su ruta habitual.

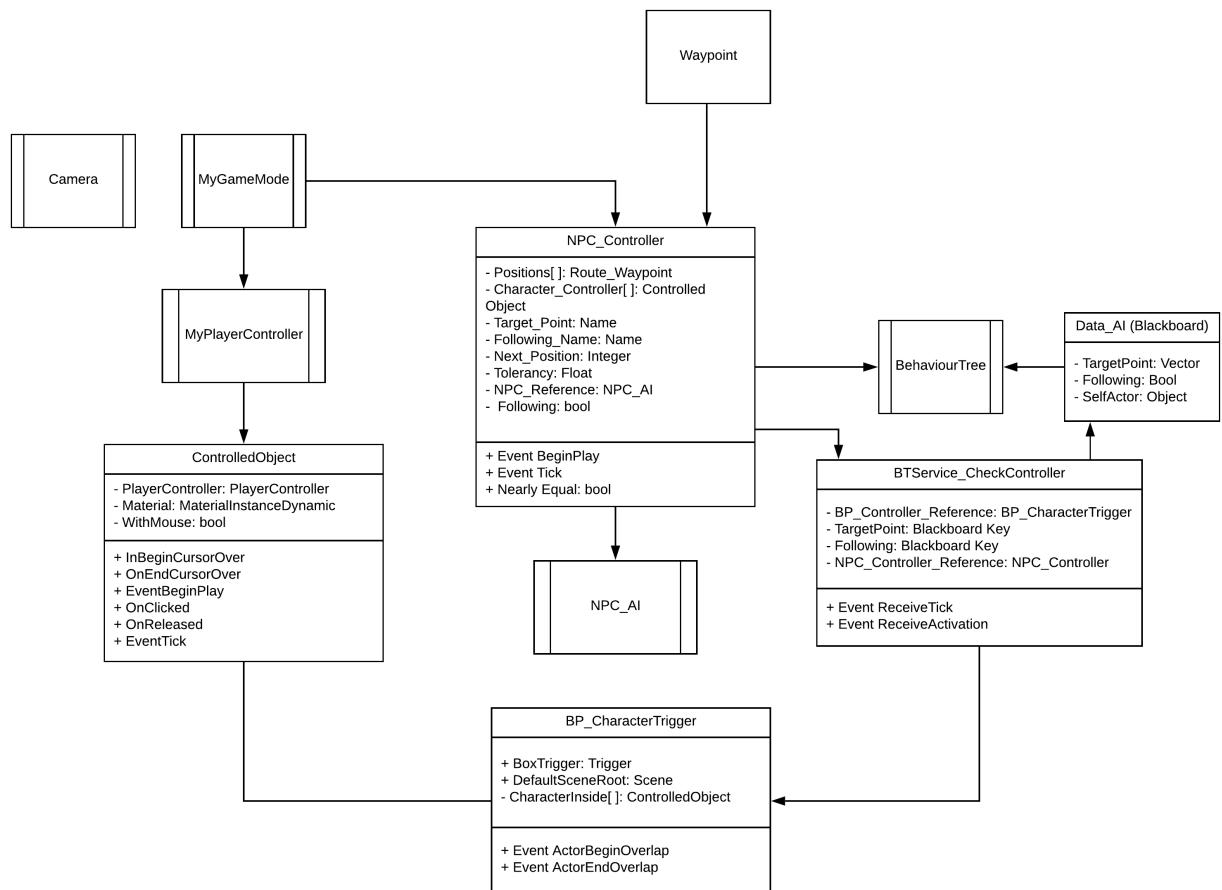
Fechas

Comienzo: 24/01/2018

Entrega: 21/02/2018

Descripción Técnica

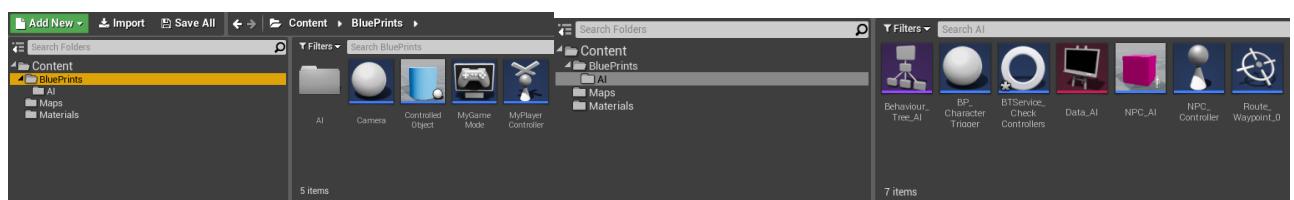
En la imagen siguiente se puede ver observar un diagrama UML con las clases de las que esta compuesto el proyecto.

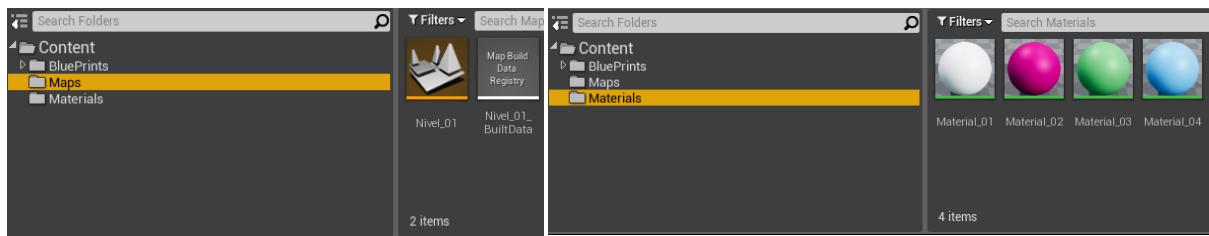


A continuación se explican los componentes del juego:

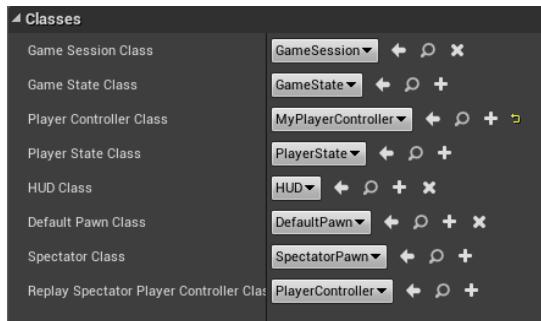
Estructura

A continuación se muestra en imágenes la estructura del proyecto. La carpeta Content es la carpeta padre. Esta dividida en Blueprints, Maps y Materials. La carpeta Blueprint a su vez contiene distintos elementos a la par que otra carpeta con más Blueprint enfocados a la IA de la práctica.



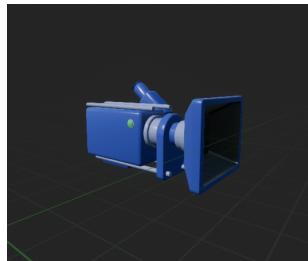


MyGameMode



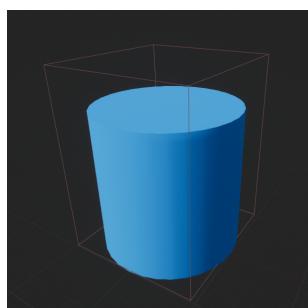
En la imagen de la izquierda se muestran las clases por defecto a utilizar implementadas en MyGameMode.

Cámera



La cámara en si misma no contiene código ya que se mantiene en su posición estática constantemente.

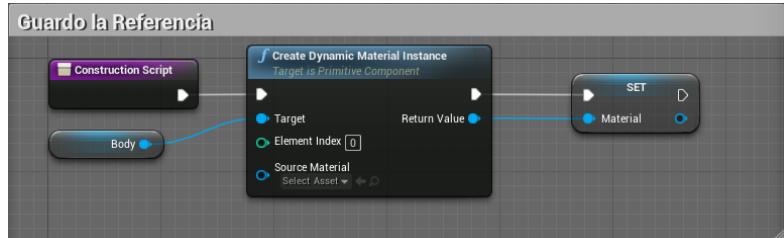
ControlledObject



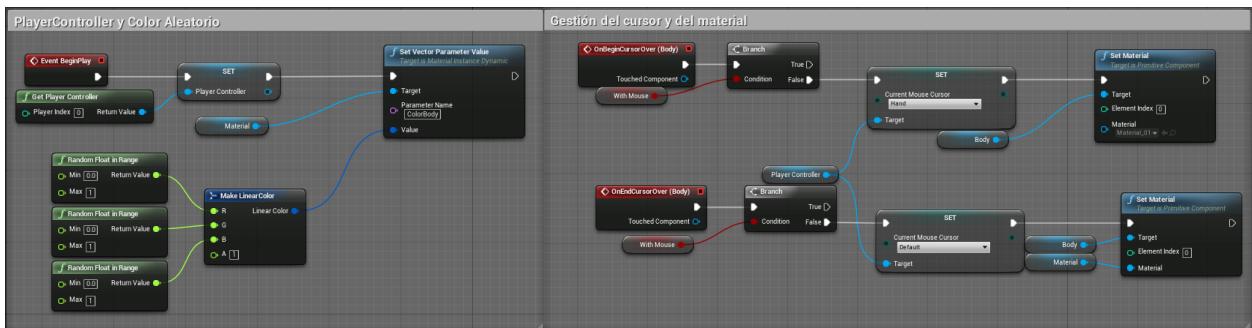
ControlledObject son todos aquellos objetos que se pueden seleccionar en la escena y arrastrar hasta la zona de acción del Agente con inteligencia artificial. Al iniciar la escena, cada uno tendrá un color aleatorio distinto cada vez.



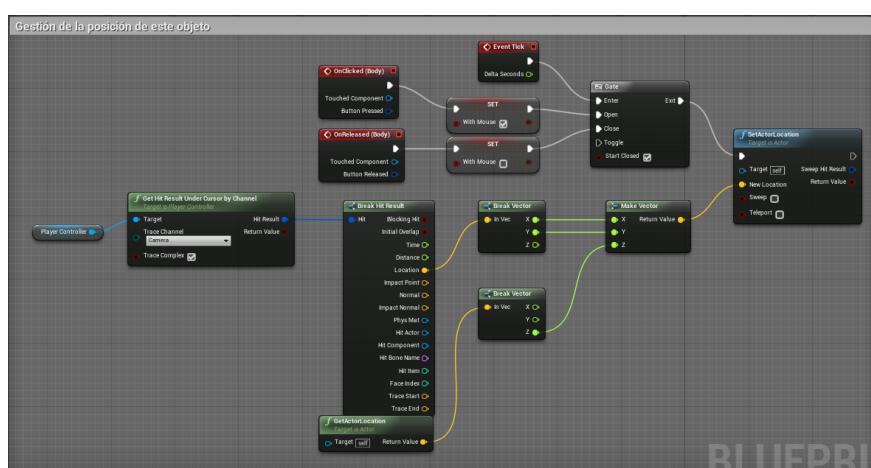
En la imagen siguiente se puede observar la estructura de este elemento. Cuenta con un Body y un Collider. Se encarga de controlarse a si mismo básicamente. Al crearse recoge la referencia al material que tiene sobre si mismo para asignarse un color aleatorio más tarde.



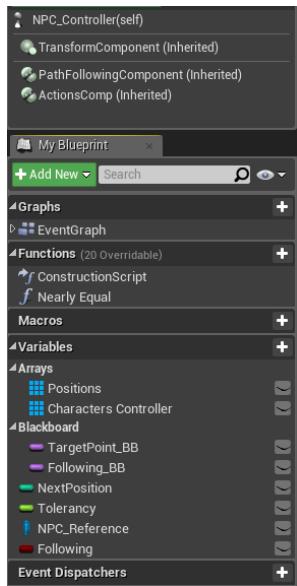
Al arrancar la escena, recoge a su propio controlador y establece un color de manera aleatoria al material que él mismo posee. En cada partida, su color será diferente. También se encarga de controlar si el usuario ha colocado el ratón sobre él. Si es así, cambia el cursor y cambia su propio material para hacerse ver claramente sobre quien tiene el mouse el usuario.



Si el usuario hace click sobre él, el objeto seguirá al mouse constantemente hasta que vuelva a hacer click, momento en el que lo dejará en ese punto.

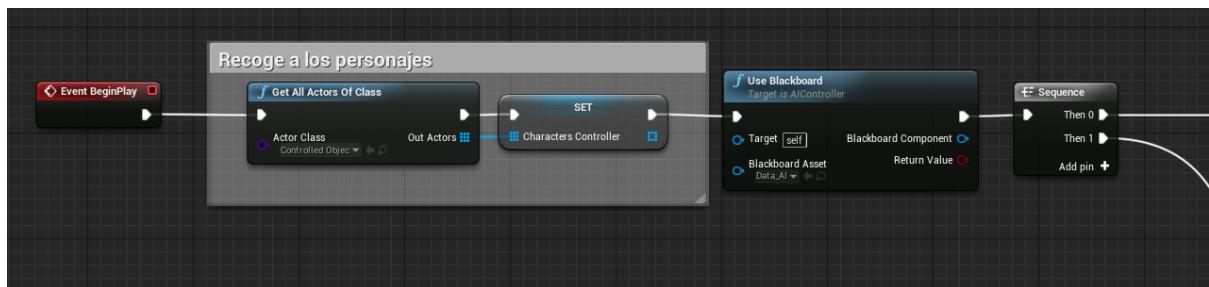


NPC_Controller

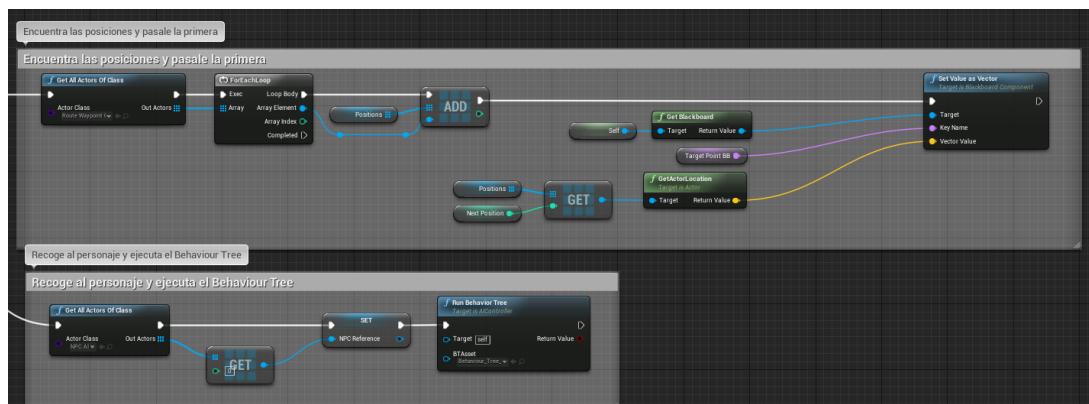


Es el controlador del NPC_ AI, el agente que se mueve de un punto a otro y que persigue a los ControlledObject. La estructura del objeto es la que se puede ver en la imagen de la izquierda. Controla las posiciones a las que debe moverse el NPC, los ControlledObject que hay, los puntos a los que debe moverse, si debe seguir a alguien o no, a que posición debe ir y si se encuentra o no en la siguiente posición.

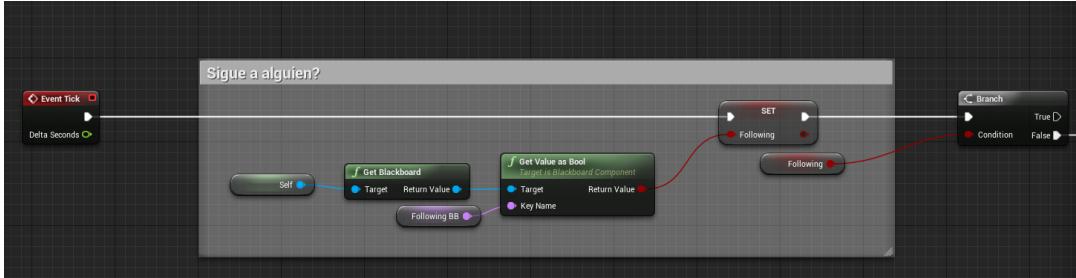
Al arrancar la escena lo primero que hace es coger la referencia a todas las cosas importantes que debe controlar. Primero recoge referencias de todos los ControlledObject que hay en la escena y se le indica que Blackboard (algo que veremos más adelante) debe utilizar.



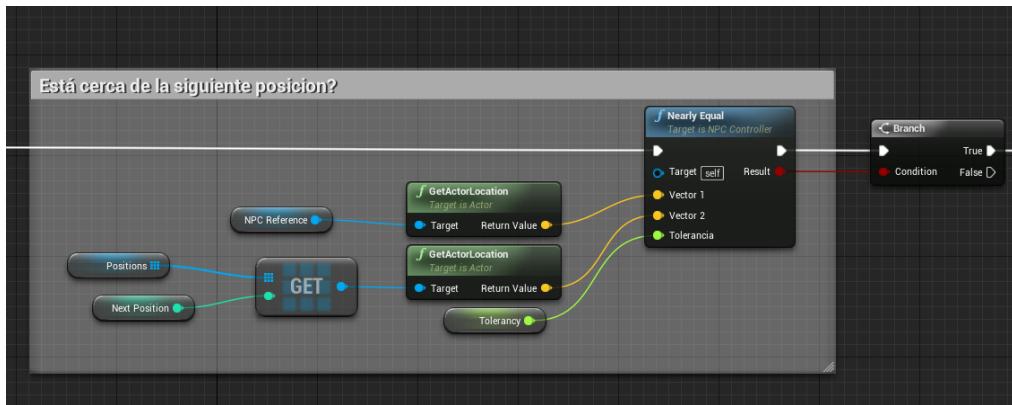
Una vez hecho esto realiza dos tareas: Recoger todos los Waypoints que hay en la escena y marcar el primero al que debe ir en el Blackboard y recoge al NPC_AI de la escena (que veremos más adelante). Tras esto, comienza a ejecutar el Behaviour Tree.



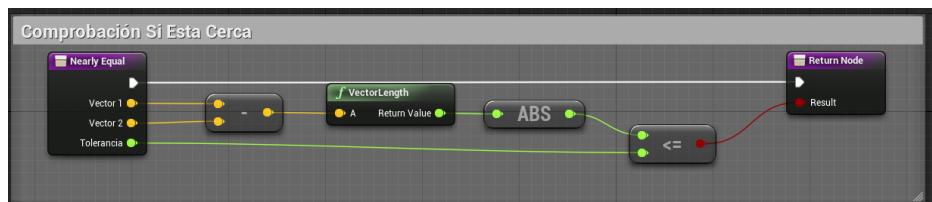
En cada frame realiza una serie de comprobaciones. Comprueba si se encuentra siguiendo a alguien. Si fuese así no realizará nada, pero en caso contrario realizará la comprobación de si ha llegado al punto siguiente.



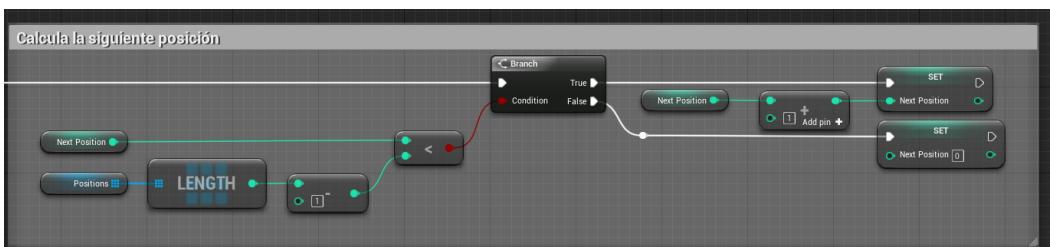
El método para comprobar si se encuentra cerca se llama Nearly Equal.



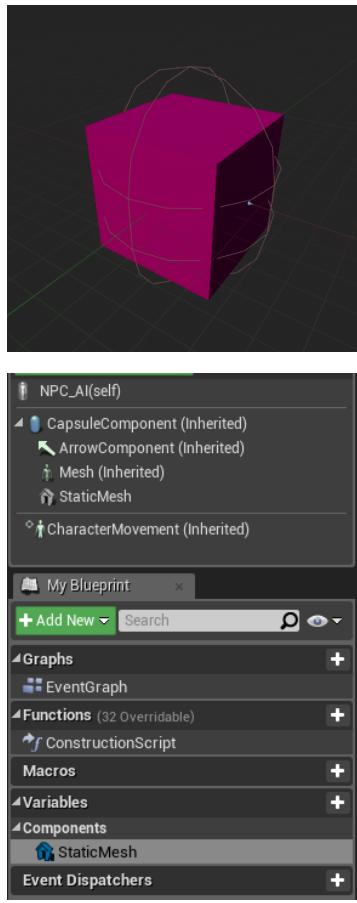
Este método recibe la posición del Actor y la posición a la que tiene que alcanzar, así como un valor de tolerancia mínima para ver si ha llegado hasta dicho punto o no.



En caso de haber llegado correctamente hasta la posición (o un punto muy cercano a ella) se calculará la siguiente posición y se le pasará esta.



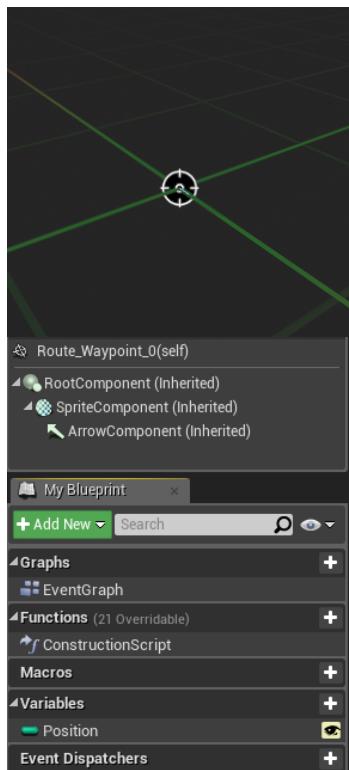
NPC_AI



Es el actor que se encuentra en la imagen de la derecha. Es la representación gráfica del agente con IA. Como tal no realiza ninguna tarea más que la de representar. Todas las tareas las realiza su controlador, anteriormente explicado.

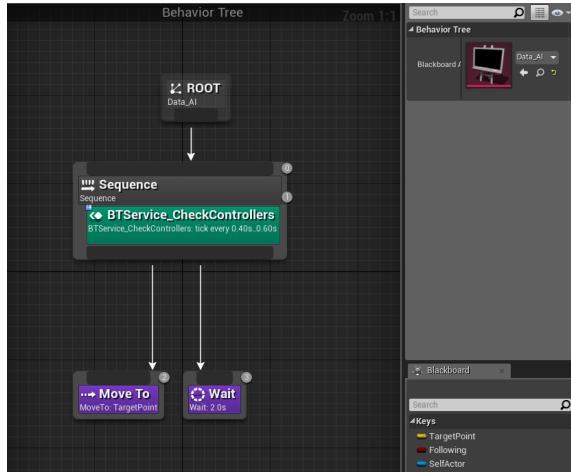
Esta es la estructura del actor. Dado que no tiene ninguna tarea que hacer, los principales elementos de los que se componen son un StaticMesh y una capsula a modo de Collider de físicas.

Waypoint



Es uno de los puntos que se coloca en la escena para que el agente con IA recorra en su patrón de recorrido. La representación gráfica que tiene no se verá en el juego final. Tan solo es una manera de que el desarrollador pueda observar donde se encuentra. En la imagen inferior podemos ver la estructura del objeto.

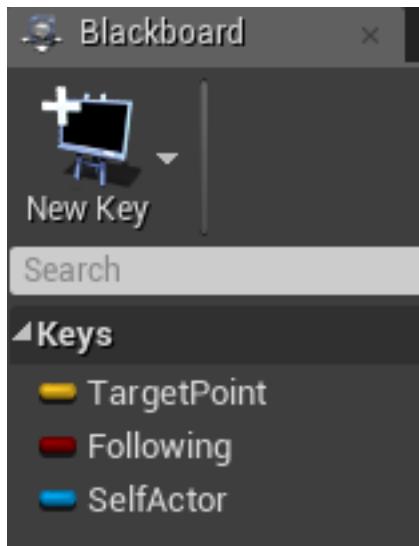
Behaviour Tree



Este es el árbol de comportamiento del cual hace uso el NPC_Controller. Básicamente es una secuencia con un Service y dos ramas. La secuencia ejecuta las ramas de izquierda a derecha. El orden de ejecución será el siguiente: BTService_CheckControllers, Move To y Wait. El sentido de esto es el siguiente: El Servicio se encarga de actualizar los valores y comprobar distintas cosas. Una vez realizado esto,

ordena al NPC/Agente a la posición que almacena en su Blackboard. Después de esto le ordena esperar 2 segundos.

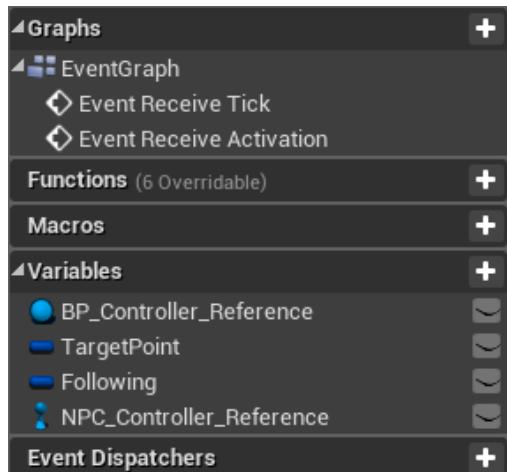
Blackboard



Este es el blackboard del Behaviour Tree explicado antes. El blackboard sirve para comunicar los datos entre el Behaviour Tree y el resto de Blueprints. Tiene tres variables: TargetPoint, posición a la que deberá acudir; Following, booleano que le indica si esta siguiendo a alguien; y SelfActor, una referencia a si mismo como Actor.

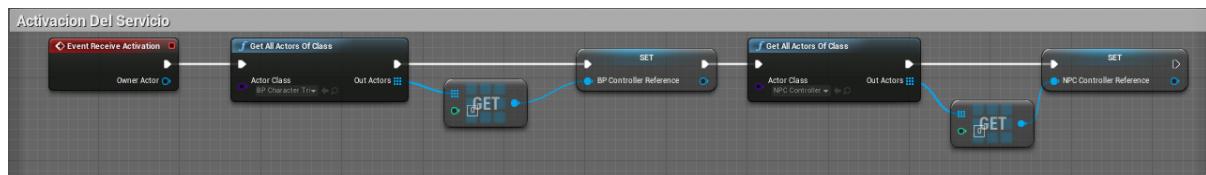
BTService_CheckControllers

Un servicio es un Blueprint que se encarga de actualizar y comprobar datos dentro de un BehaviourTree.

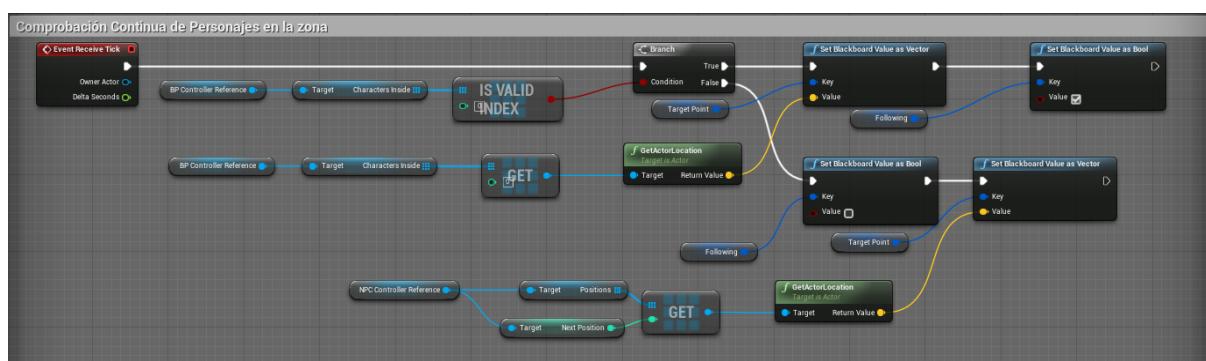


Esta es la estructura de este servicio. La variables que tiene son: BP_Controller_Reference, una referencia al Trigger que detecta la entrada de ControlledObjects en la zona (que se explicara más adelante); TargetPoint, referencia a la variable del Blackboard; Following, referencia a la variable del Blackboard; y NPC_Controller_Reference, referencia al controlador del Agente.

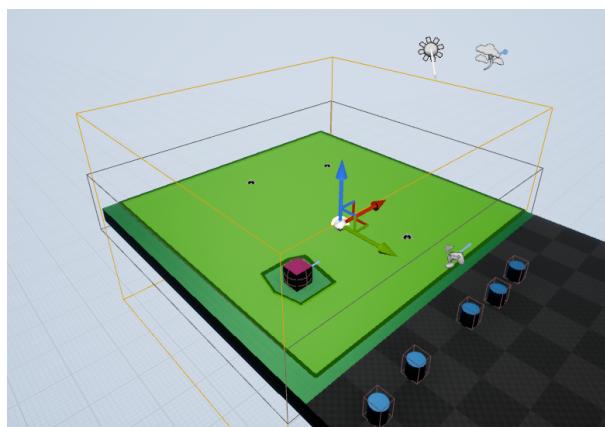
Al activarse el servicio por primera vez, lo primero que hacer es obtener la referencia al Trigger y al controlador del NPC y almacenarlas en sus variables correspondientes.



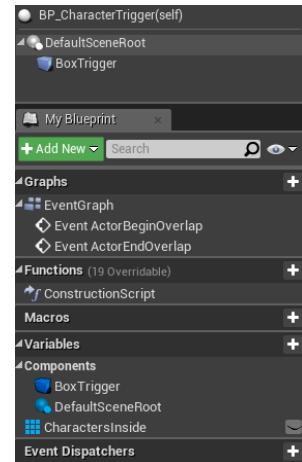
En cada frame el Servicio comprueba si hay algun ControlledObject dentro de la zona, si es asi, le marca la posición como la nueva dirección a la que debe ir. En caso negativo, recoge la posición que le debería tocar como la siguiente y se la marca al Blackboard para que, cuando se ejecute el Move To, acuda al lugar correcto.



BP_Character_Trigger

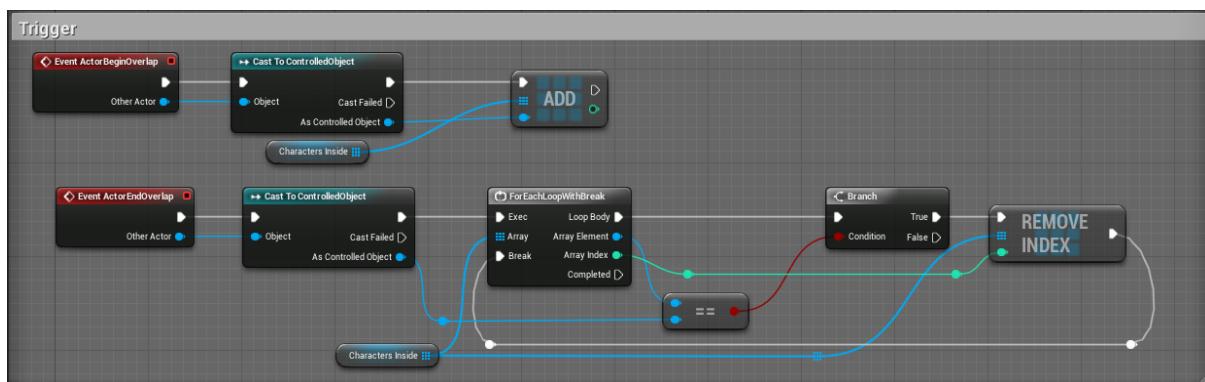


Es un trigger que se encuentra en la escena y que controla los ControlledObject que entran en la zona. Podemos verlo en la imagen de la izquierda.



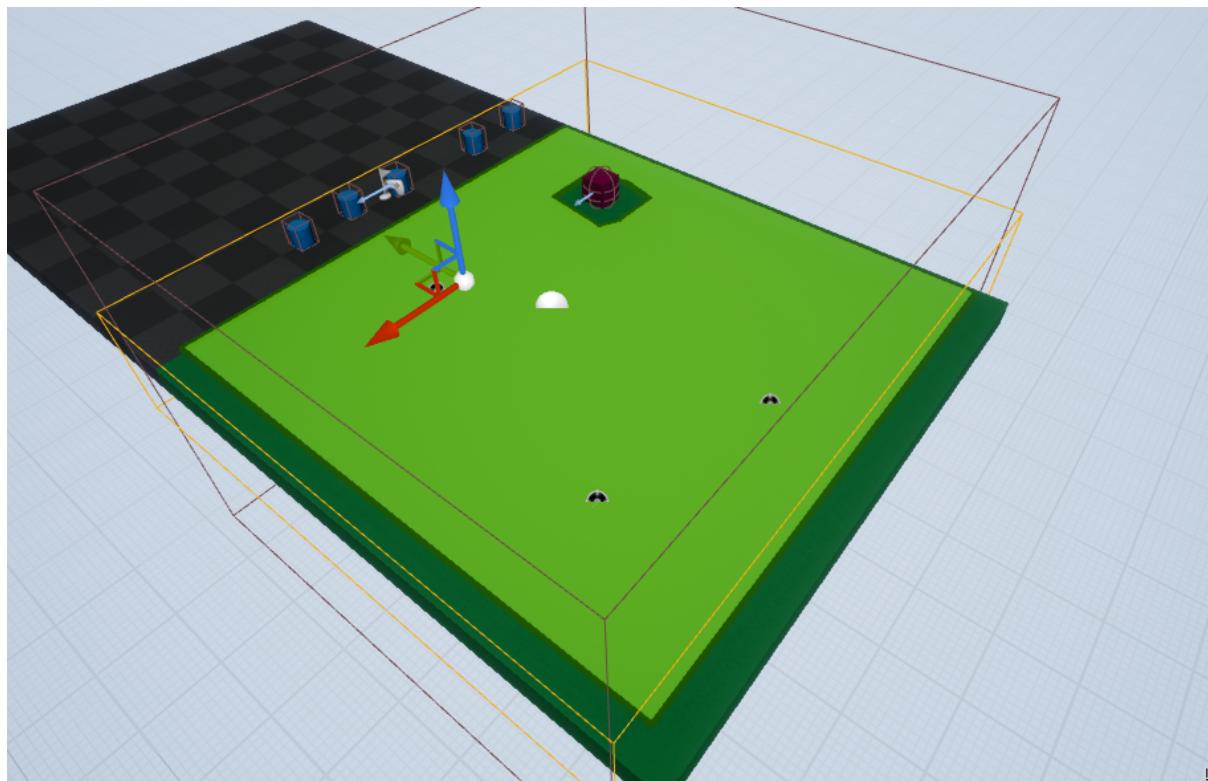
Podemos ver la estructura que tiene en la imagen de la derecha. Principalmente es un BoxTrigger que almacena en un array los elementos que entran en su campo de acción.

Si un Actor entra en la zona, el BP lo castea a ControlledObject. En caso de serlo, se añade al array. Si un Actor sale de la zona, se comprueba que sea un ControlledObject, se busca en el Array y se elimina. Al eliminarlo, todos los demás elementos ocupan el hueco vacío que haya quedado.



NavMeshBoundsVolume

Es un elemento que se saca del panel lateral izquierdo. Se utiliza para definir a una inteligencia artificial que zonas puede recorrer y cuales no. Podemos ver cómo es el de la escena en la imagen inferior.



Diario de Desarrollo

Desarrollé esta práctica durante las navidades de 2017. Partí del proyecto ya creado por el profesor que se encontraba en campus para seleccionar los objetos y arrastrarlos. A partir de ahí, y dado que no había realizado ningún ejercicio sobre BehavioursTree en Unreal, decidí informarme sobre el comportamiento de estos. Utilicé el documento que se encontraba en campus y busqué información en Internet para comprender el sentido de Task, Decorator y Service y todos los modos de ejecución de los BehaviourTree. Una vez comprendidos, en papel, desarrollé como sería la lógica de funcionamiento del árbol de comportamiento.

Primero coloqué las plataformas donde se iba a producir el juego, ajuste la cámara y preparé los BP que iba a necesitar.

El primer objetivo que me planteé fue el de que el Agente se moviera de un punto a otro y se repitiera el bucle sin parar. El primer problema lo encontré al colocar el NavMesh y es que no veía la malla de navegación en la superficie. Esto se debe a que normalmente la pre visualización viene desactivada. Pulsando una tecla del teclado ya se puede ver. Coloqué los puntos por la malla y en pocos intentos conseguí que funcionara. Sorprendido y motivado por el hecho de haberlo conseguido con relativa facilidad comencé a implementar la introducción de los ControlledObject.

Esta nueva fase me tomó más tiempo. Estuve durante mucho tiempo tratando de buscar un método para detectar si hay un ControlledObject dentro de la malla de navegación. Sin embargo, acabé por descubrir que no hay ningún método en Blueprints que permita hacer esto. La única manera sería a través de C++.

Debido a esto, planteé la utilización de un Trigger que coincidiera con el tamaño del NavMesh para comprobar si había alguno en la zona. Una vez hecho esto, me resultó bastante fácil continuar con el desarrollo ya que tan solo implicaba algunos cambios en lo ya creado y el movimiento con ratón de los ControlledObject ya estaba creado.

Tiempo después decidí implementar que los objetos aparecieran con colores aleatorios cada vez que se jugara, a modo de extra.

Un fallo que he tenido y que no me he dado cuenta de él hasta haber desarrollado la memoria es que no he aplicado correctamente la manera de trabajo que pide Unreal en cuanto a Actors y Controladores se refiere. En mi caso, centrado en los ControlledObject, desarrollé el código en el actor cuando debería haberlo hecho en su controlador y no a la contra. Sin embargo, si que he cumplido esta regla en el Agente con IA. Dado que me dí cuenta demasiado tarde de este fallo y todo funcionaba correctamente, decidí no hacer el cambio.

He tenido otros problemas a la hora de comprender como se pasan correctamente algunas variables entre los Blueprints y el Blackboard. Me parece un sistema algo tedioso y complicado de ver. Por ejemplo, en el Servicio del Behaviour Tree tenemos variables de tipo Blackboard Key para comprobar o almacenar datos, mientras que en el NPC_Controller tenemos variables de tipo Name y se las establecemos al Blackboard mediante "Set Value as ...". Creo que es un sistema difícil, que pueda dar muchos problemas y que, para los que estamos empezando, puede ser muy lioso incluso para los ya experimentados si tuvieran que asimilar el código de alguien.

Enlace a Repositorio

<https://github.com/Nikoooo95/LittleAIUnreal.git>