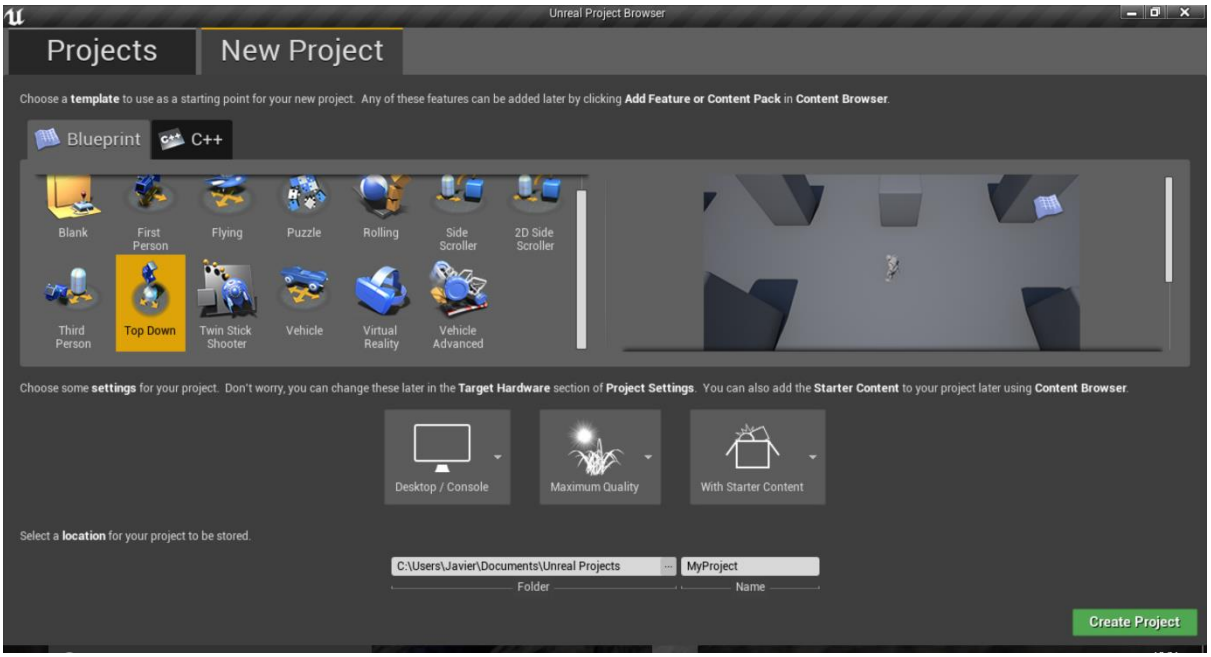
Ejercicio Utilities

Enviroment Query System

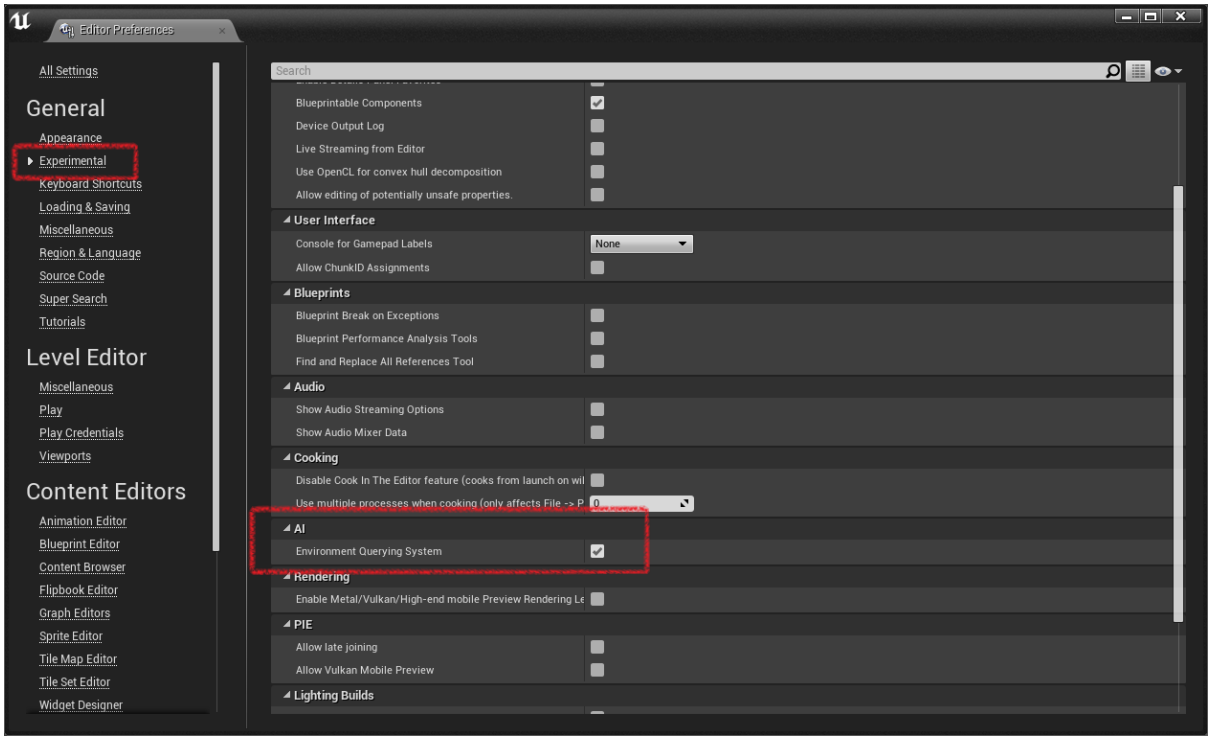
El objetivo de esta práctica es conocer qué es *Enviroment Query System* (en adelante EQS) y cómo utilizarlo. EQS es el sistema que gestiona las *Utility Functions* de Unreal. Nos permite generar un conjunto de elementos y realizar un filtrado y evaluación de los mismos. En esta práctica crearemos dos consultas, una para elegir un objetivo del que huir y otra para elegir una posición que nos permita estar a cubierto de dicho objetivo. En esta ocasión implementaremos este comportamiento utilizando *Behavior Trees.*

1. Comienzo

a. Crear proyecto: Como base vamos a usar el ***template de Blueprint "Top Down".***



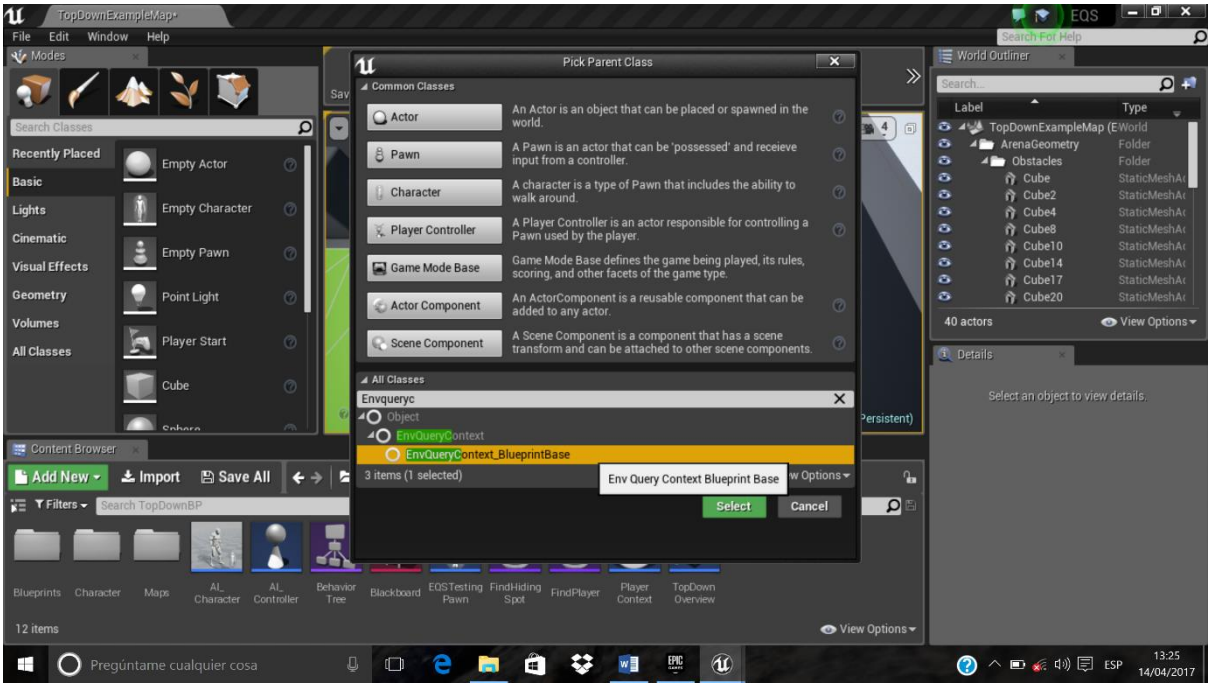
b. De momento EQS es considerado una característica experimental de Unreal, por lo que deberemos activarla para poder usarla.



c. Creamos una *NavMesh*, como hemos visto en prácticas anteriores. (Anexo 3)
d. Creamos los *blueprints* necesarios para el ejercicio:
    - Creamos un *AIController*.
    - Creamos un *Character*; podemos crearlo como vimos en prácticas anteriores, de cero, o directamente crear desde el menú de creación de clases típicas: *CreateCharacter*.
    - Asignamos el *AIController* a dicho *Character*.
    - Colocamos dicho AICharater en el mapa, arrastrando para crear una nueva instancia.
    - Creamos un *BehaviorTree*:
        - *Add New -> Create Advanced Asset -> Artificial Intelligence -> Behavior Tree*
    - Creamos una *Blackboard*:
        - *Add New -> Create Advanced Asset -> Artificial Intelligence -> Blackboard*
    - Creamos dos *Enviroment Query System*:
        - Add *New -> Create Advanced Asset -> Artificial Intelligence -> Enviroment Query System*. Uno de ellos será el que utilicemos para encontrar objetivo: FindPlayer, el otro será para encontrar un lugar en el que estar a cubierto del objetivo: FindHidingSpot.

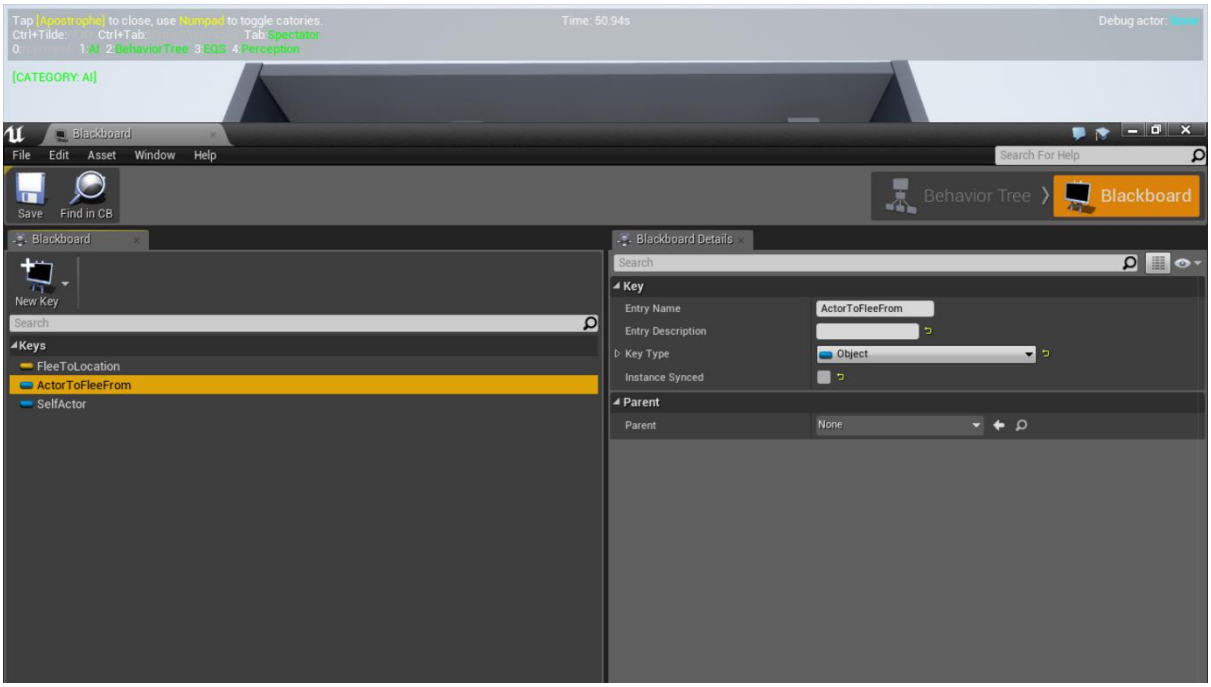
e. Por último, crearemos un *EnvQueryContext_BlueprintBase*

- Los llamaremos *PlayerContext*
- Cuando vayamos a realizar consultas con EQS necesitaremos hacerlas dentro de un contexto. Por defecto las *Queries* se realizarán sobre la propia IA que genera las consultas: *TheQuerier*
- Pero en ocasiones querremos hacer las consultas sobre contextos más complicados, por ejemplo, teniendo como marco de referencia al jugador al cual se enfrentan las IAs. Para esto servirá, como veremos más adelante, este nuevo contexto que crearemos, para poder realizar consultas con respecto al jugador.

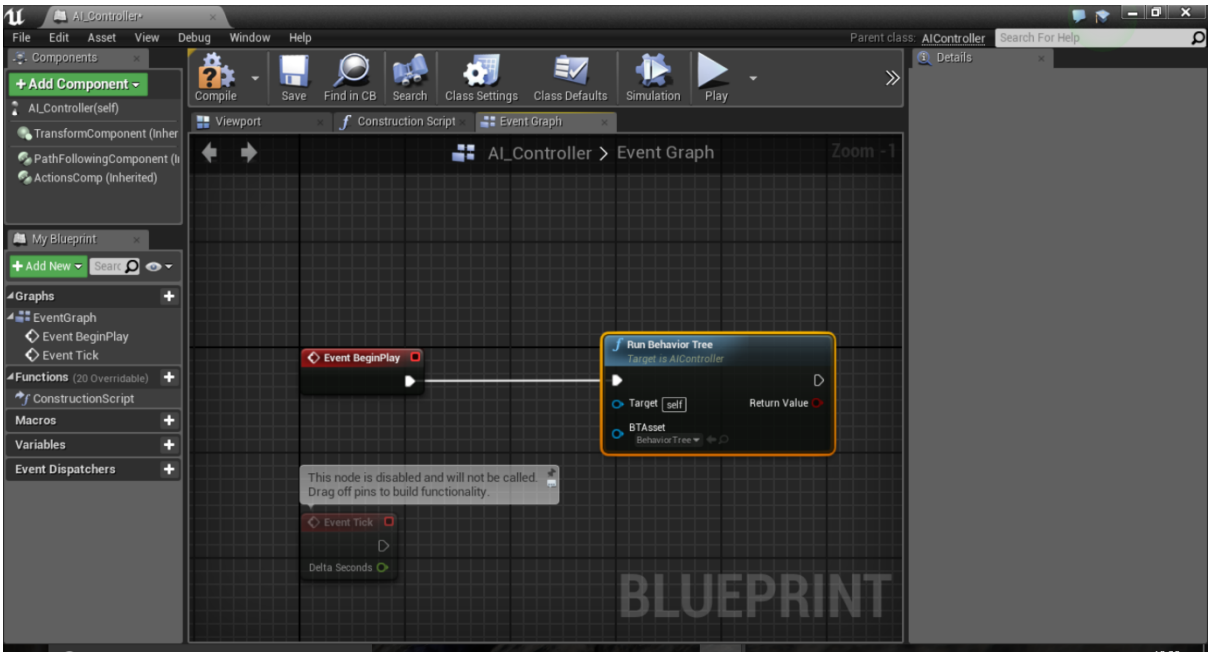


f. Vamos a configurar la *Blackboard*.

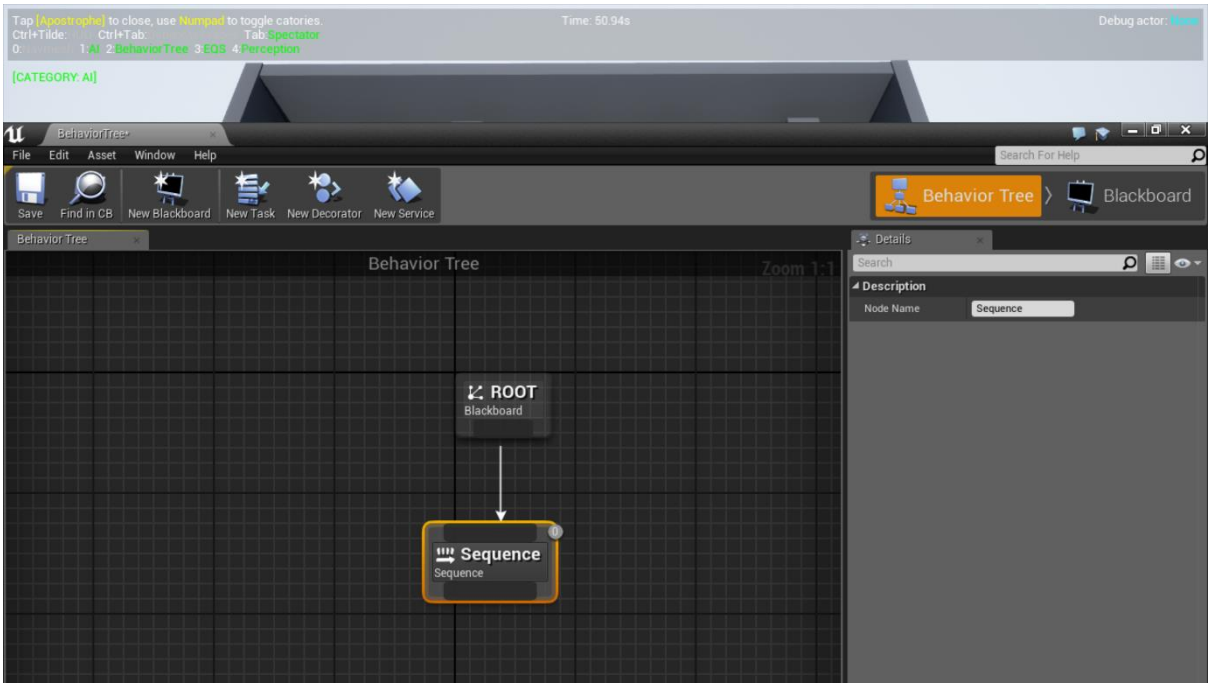
- *Doble clic* en la *Blackboard* para editarla
- Creamos un par de nuevas *"keys"*
- La primera será de tipo *vector* y la llamaremos *FleeToLocation;* en ella almacenaremos la posición a la que deberá huir nuestra entidad.
- Creamos otra *"key"* de tipo *Object* y la llamaremos *ActorToFleeFrom; aquí tendremos la referencia a la entidad de la cual deberá huir nuestra IA.*



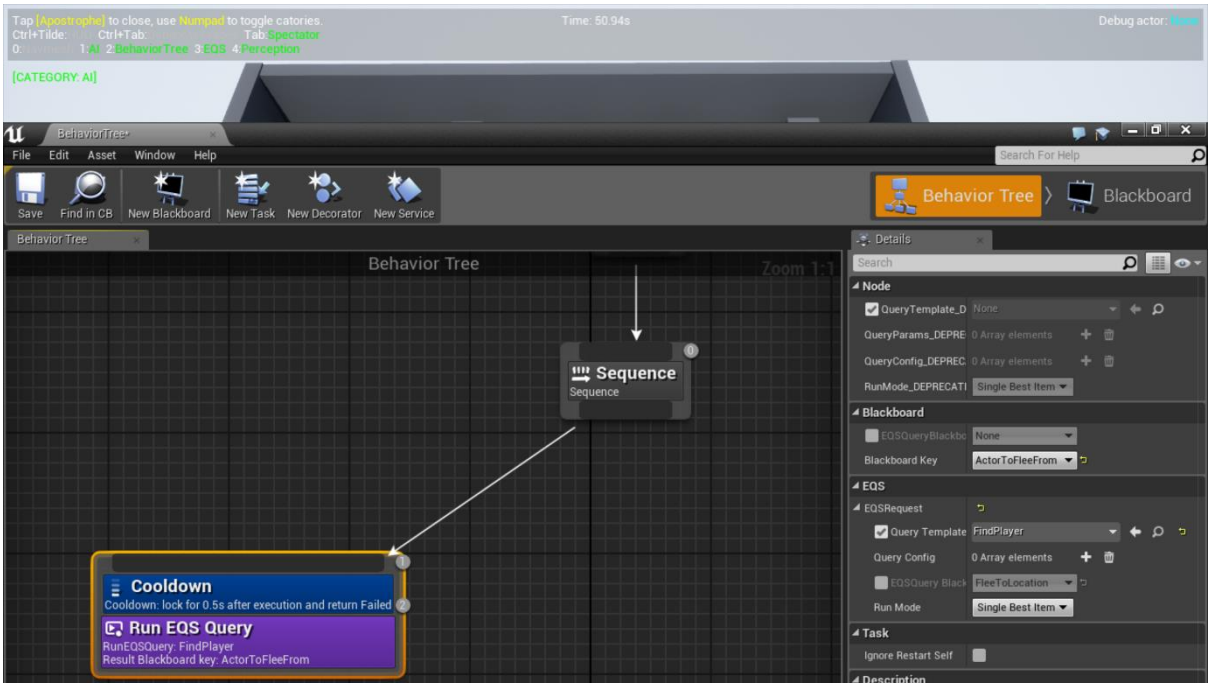
g. Ahora modificaremos nuestro *AIController* para que ejecute el *Behavior Tree* que controlará nuestra IA.
   - En el *EventGraph* buscamos el evento *BeginPlay.*
   - Creamos a continuación un nodo *RunBehaviorTree.*
   - En este nodo seleccionaremos el árbol que creamos previamente.
   - Así, el control pasa a tenerlo el árbol de comportamiento que configuraremos a continuación.



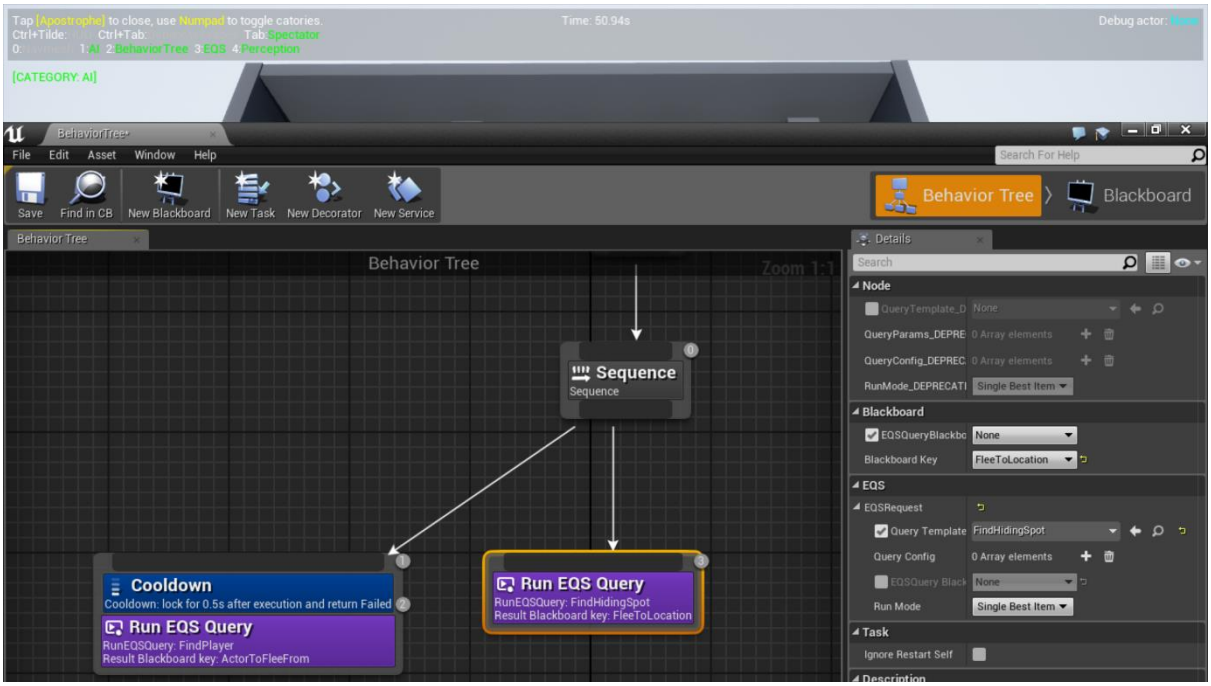
h. Vamos a construir el *BehaviorTree* que implementará el comportamiento de huida mediante las consultas de *EQS*
   - Como sabéis todo *BehaviorTree* en Unreal tiene un punto de entrada que es el nodo *Root*. A partir de él se ejecutarán todos los nodos que añadamos. Será un árbol sencillo, simplemente tendremos una secuencia que ejecutará, de izquierda a derecha los nodos que añadiremos mientras sus hijos se ejecuten satisfactoriamente. Estos hijos serán dos consultas de *EQS*, una para encontrar al jugador del que escondernos y la siguiente para encontrar un lugar que nos permita permanecer ocultos de él. Cuando ambas consultas den un resultado positivo entonces mandaremos a la entidad moverse hasta el punto que la segunda *EQS* nos ha calculado para permanecer allí escondidos.
   - Editamos el *BehaviorTree* con *doble clic*.
   - Añadimos un nodo secuencia por debajo del *Root*
     - Botón derecho y seleccionamos *Composites -> Sequence*
     - Conectamos el *Root* y la secuencia haciendo clic y arrastrando desde el nodo raíz hasta el nodo secuencia.



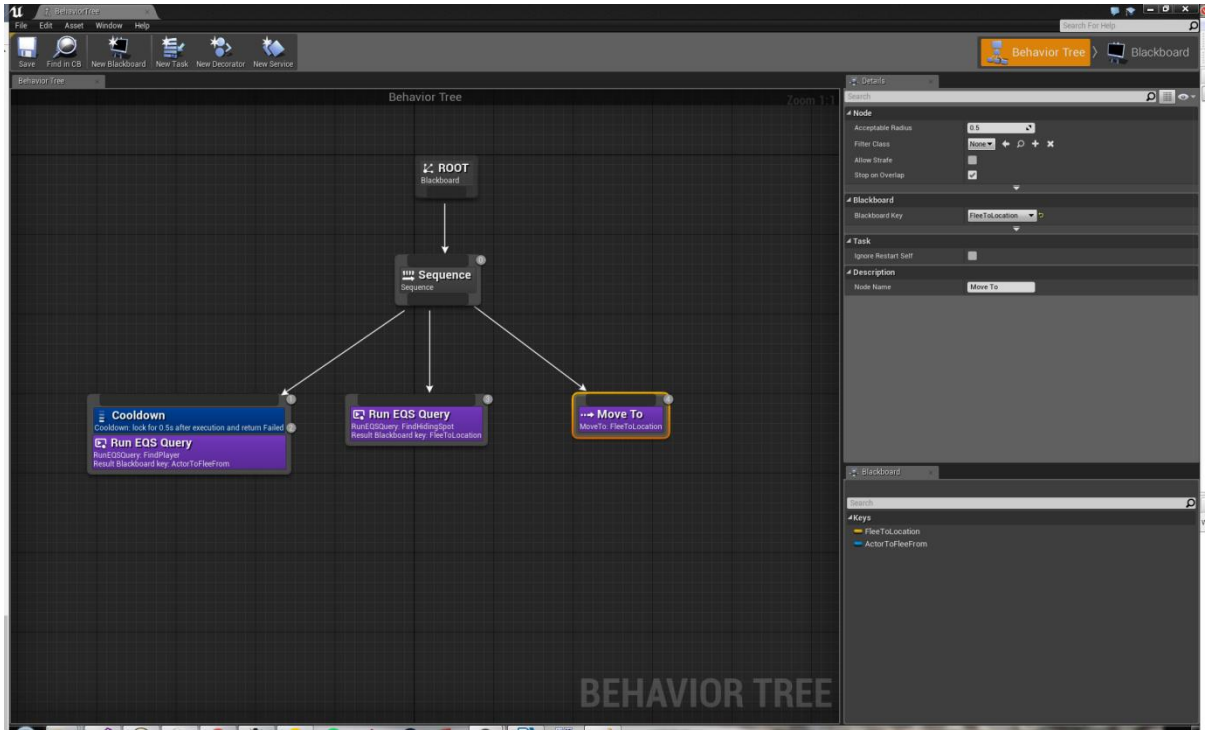
- Añadiremos ahora la primera de las consultas *EQS*
  - Botón derecho y seleccionamos *Task->Run EQSQuery*
  - La conectamos por debajo de la secuencia
  - La seleccionamos para editar sus propiedades en el panel de detalles
  - En el campo *QueryTemplate* seleccionamos la consulta *FindPlayer* que habíamos creado anteriormente
  - En el campo *BlackboardKey* seleccionamos la variable *ActorToFleeFrom*
  - Vamos a añadirle un decorador a este nodo para evitar que se ejecute cada constantemente:
    a. Botón derecho en el propio nodo y añadimos *AddDecorator->Cooldown*
    b. Configuramos su tiempo de *cooldown time* a 0.5 segundos



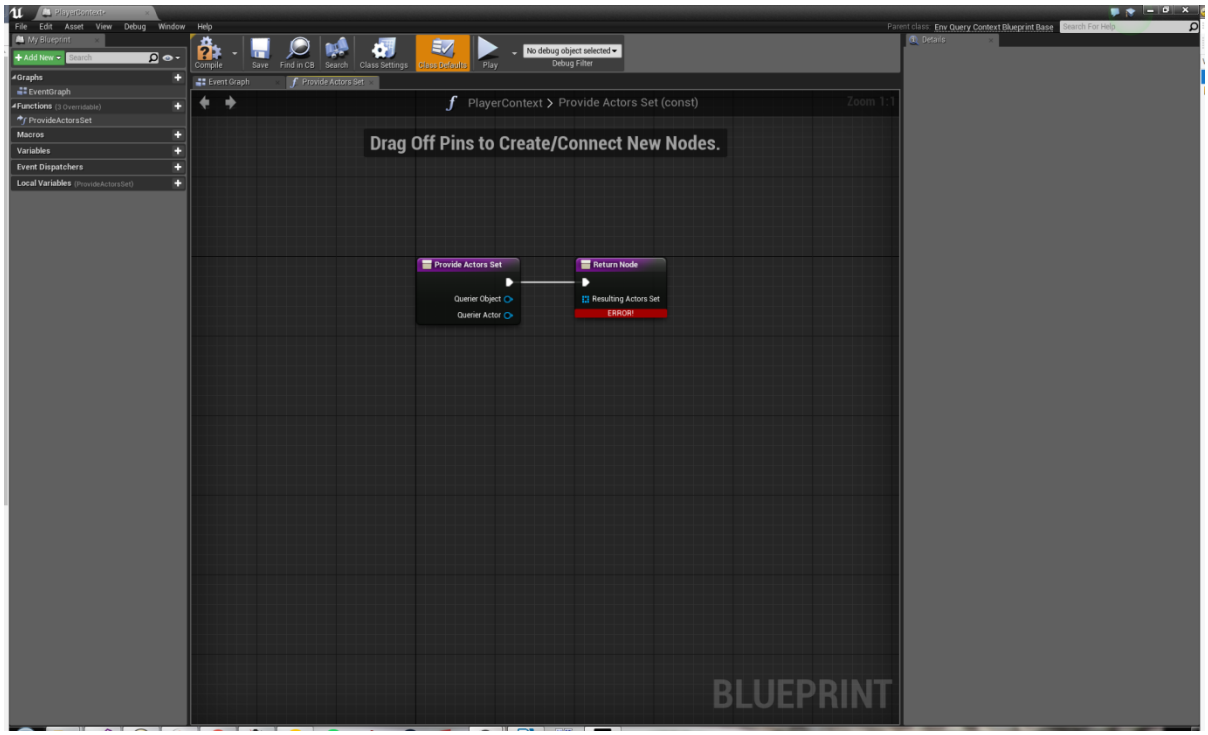
- Añadimos nuestra segunda consulta *EQS*
  - La creamos y la añadimos por debajo de la secuencia tras la *EQS* de *FindPlayer*
  - En sus propiedades seleccionamos como *QueryTemplate FindHidingSpot*
  - En esta ocasión seleccionamos como Blackboard Key la variable *FleeToLocation*



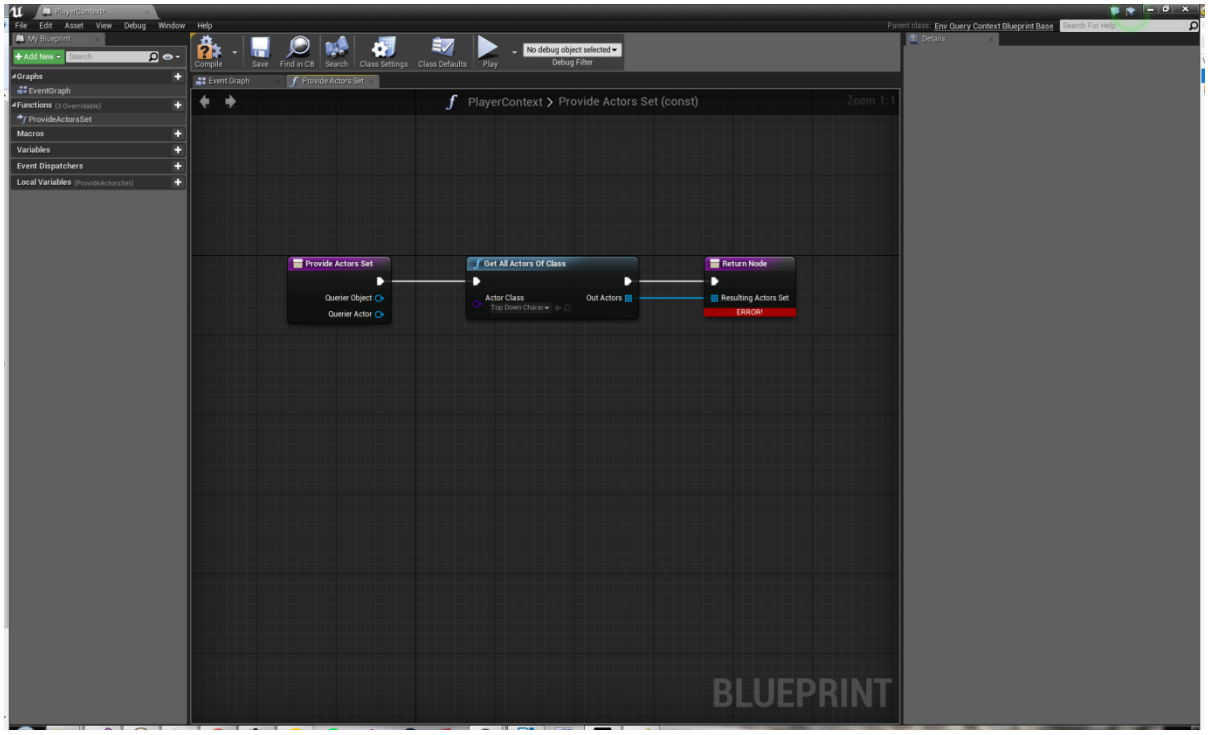
- Por último, creamos el nodo que moverá a nuestra IA
  - Creamos un nuevo nodo en el *Behavior Tree: Task->Move To*
  - Lo colocamos bajo la secuencia después de las dos consultas
  - Configuramos su *Blackboard Key* con la variable *FleeToLocation*



i. Editaremos a continuación *PlayerContext*, nuestro *EnvQueryContext_BluePrintBase*
   - *Doble clic* en *PlayerContext* para editarlo.
   - Tenemos que sobrescribir la función que generará el conjunto de elementos.
   - Para ello hacemos *clic* en *Functions->Override.*
   - Seleccionamos *ProvideActorsSet* como *OverrideFunction.*
   - Nota: Dicho botón estará oculto mientras no pasemos por encima con el ratón de la parte *Functions* del panel *MyBlueprint.*

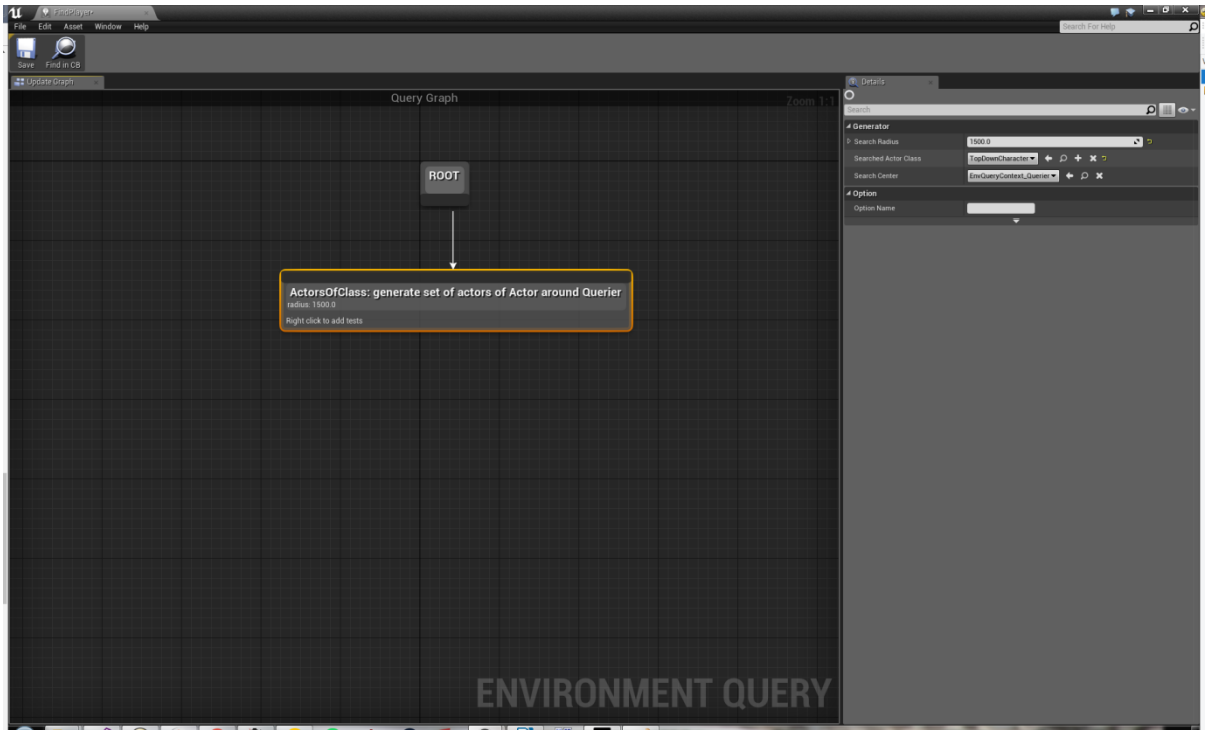


- Debemos añadir un nuevo nodo *GetAllActorsOfClass* en este *blueprint.*
- Seleccionamos como *ActorClass: TopDownCharacter.*
- Colocamos este nuevo nodo entre *ProvideActorSet* y el *ReturnNode*.

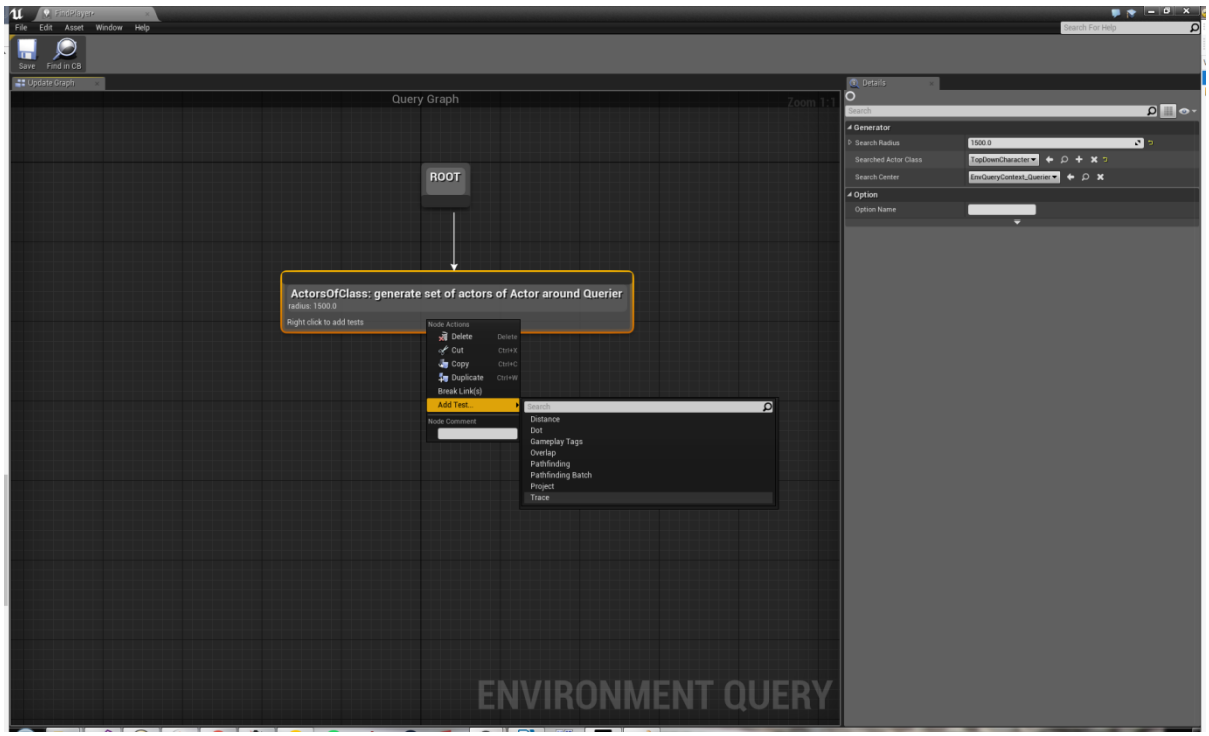


j. Vamos a configurar nuestras consultas, primero *FindPlayer*

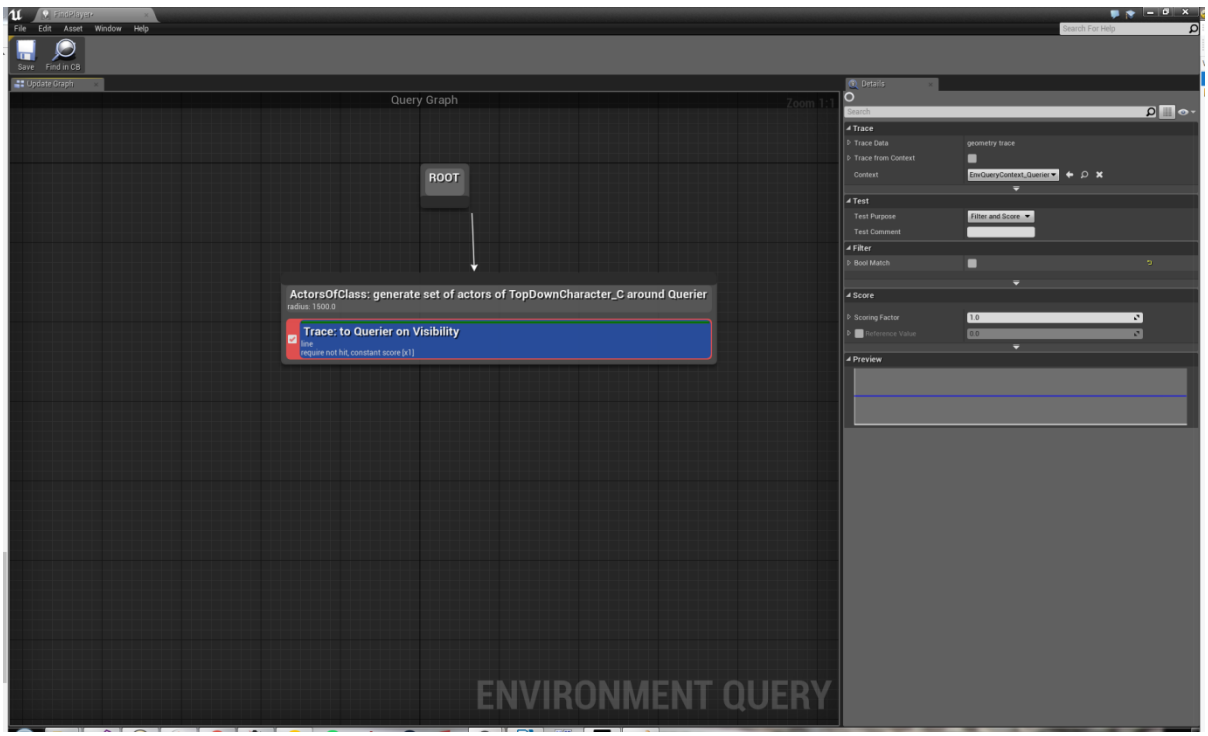
- Esta consulta, que será la más simple de las dos que crearemos, seleccionará al jugador del que queremos escondernos. Para ello nos devolverá una lista con los *TopDownCharacters* que tengamos a menos de X distancia y filtrará los que no pueda ver nuestra IA. La ventaja de hacer esto con una *EQS* en lugar de hacerlo de otra manera es que posteriormente podremos añadir nuevos filtros y evaluadores para seleccionar el mejor de estos personajes según otros criterios.
- Hacemos *doble clic* en el *asset FindPlayer* que creamos previamente
- Nos abrirá un editor, igual que el de *BehaviorTrees*, pero que servirá para editar las *queries*.
- Como tal veremos que nos crea un nodo *Root* desde el que partirá nuestra consulta
- Vamos a añadir un nuevo generador, que creará el conjunto de elementos sobre los que seleccionaremos al mejor
- Dicho generador lo colocaremos por debajo del nodo *Root*, igual que hacíamos al unir nodos de nuestro *BehaviorTree*.
- Sólo puede haber un generador por consulta
- Vamos a editar el generador
    - Modificamos el *SearchRadius* a 1500
    - Como *SearchActorClass* seleccionamos *TopDownCharacter*
    - Con esto el generador creará ítems para su posterior filtrado y evaluación con todos los *TopDownCharacters* en el radio indicado



- Vamos a crear en esta consulta un elemento que nos servirá de filtro y de evaluador al mismo tiempo
    - Creamos un *Trace Test* para comprobar si hay línea de visibilidad entre nuestra IA y cada uno de los ítems a filtrar/evaluar: Botón derecho en el generador -> *AddTest -> Trace*

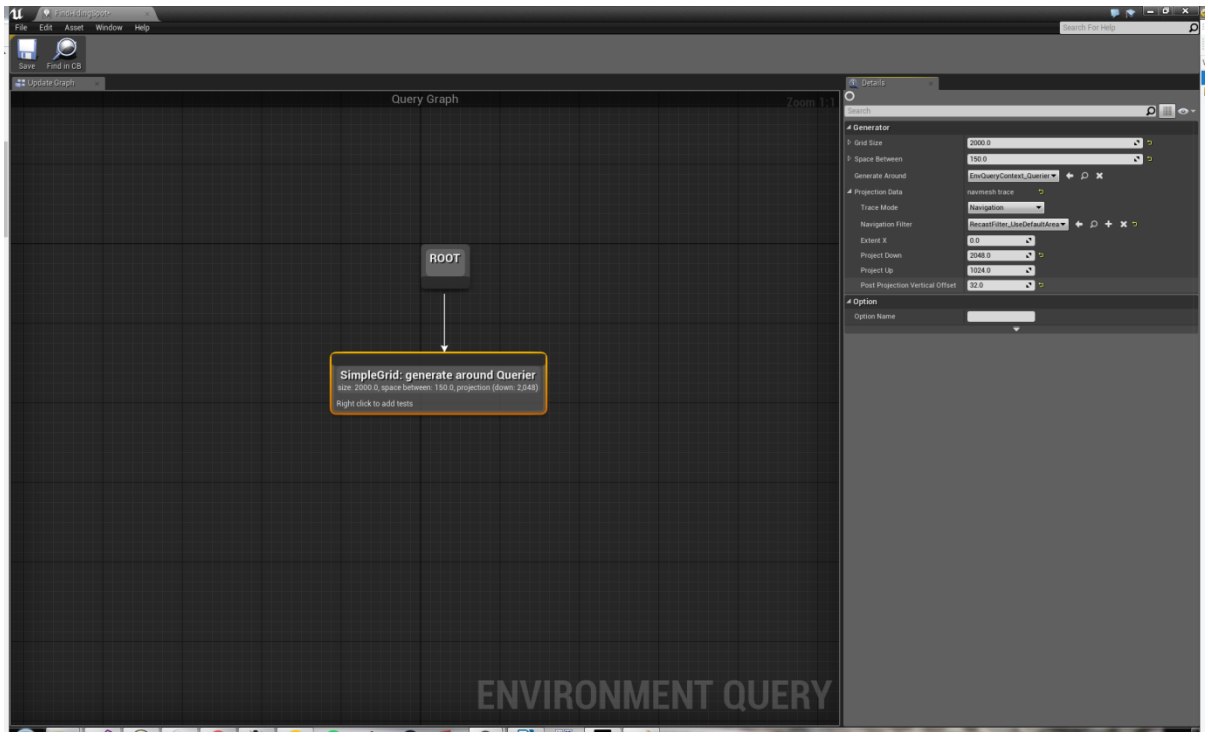


- Cambiamos la propiedad en el campo Filter->Bool Match a false. La variable BoolMatch a true indica que queremos que filtrar las consultas en las que SÍ haya visibilidad, por lo que al ponerlo a false dejaremos que pase a evaluarse, pudiendo obtener así al jugador.

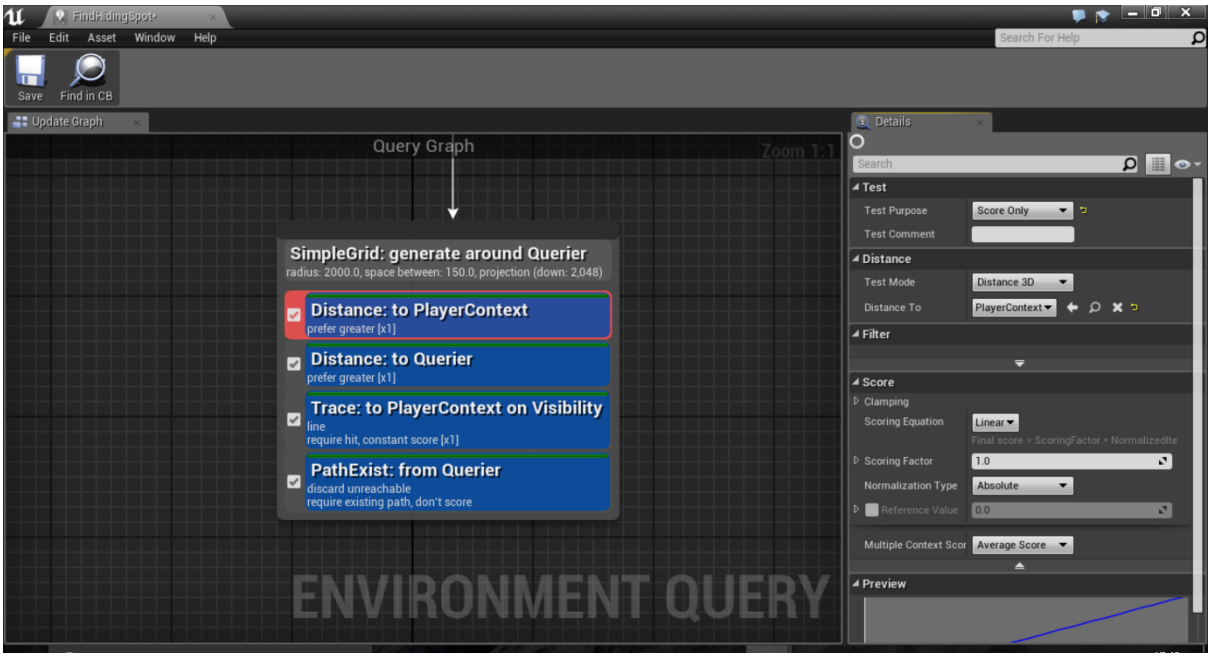


k. Ahora configuraremos la consulta *FindHidingSpot*

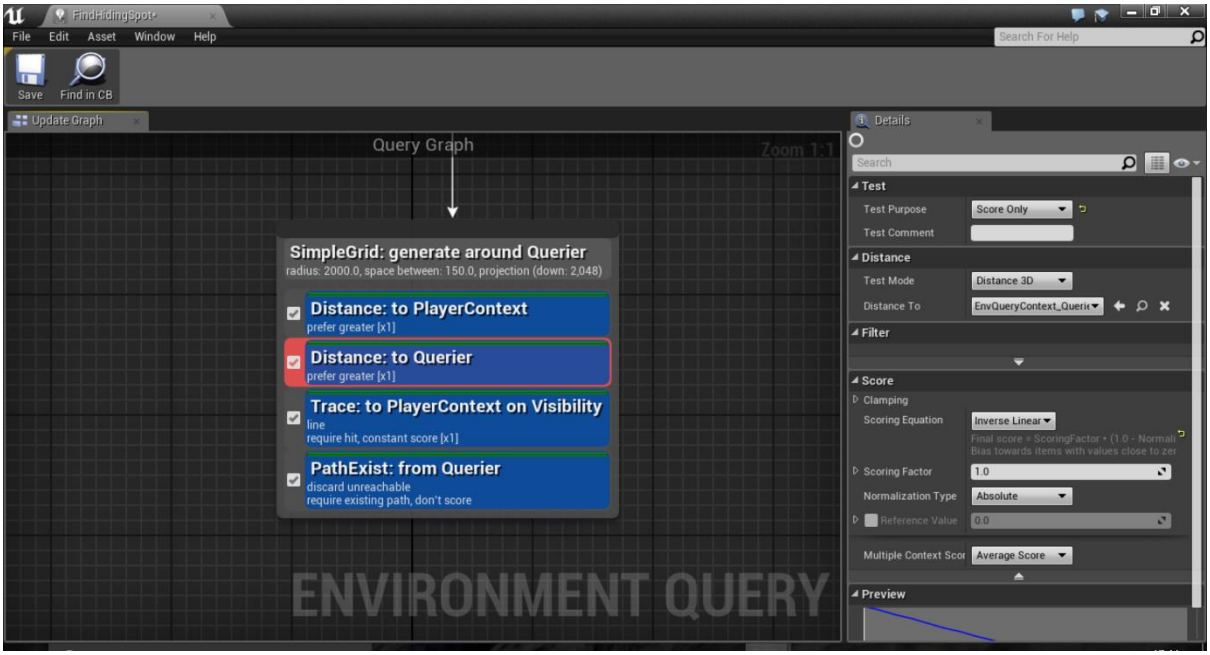
- En este caso generaremos una rejilla de puntos, que serán proyectados en la *NavMesh*, sobre los que pasaremos una serie de filtros y evaluadores para seleccionar uno de ellos, que nos permita estar a cubierto del jugador seleccionado en la consulta anterior.
- Editamos la *EQS FindHidingSpot*
- Creamos un generador de tipo *SimpleGrid*
- Vamos a configurar el generador para obtener un número manejable de ítems sobre los que iterar, pero suficientemente grande. Los valores de proyección cuadran con los tamaños de los personajes que manejamos en esta práctica.
    - *Grid size:* 2000
    - *Space Between*: 150
    - *Navigation Filter: RecastFilter_UseDefaultArea*
    - *Proyect Down*: 2048
    - *Post Proyection Vertical Offset*: 32



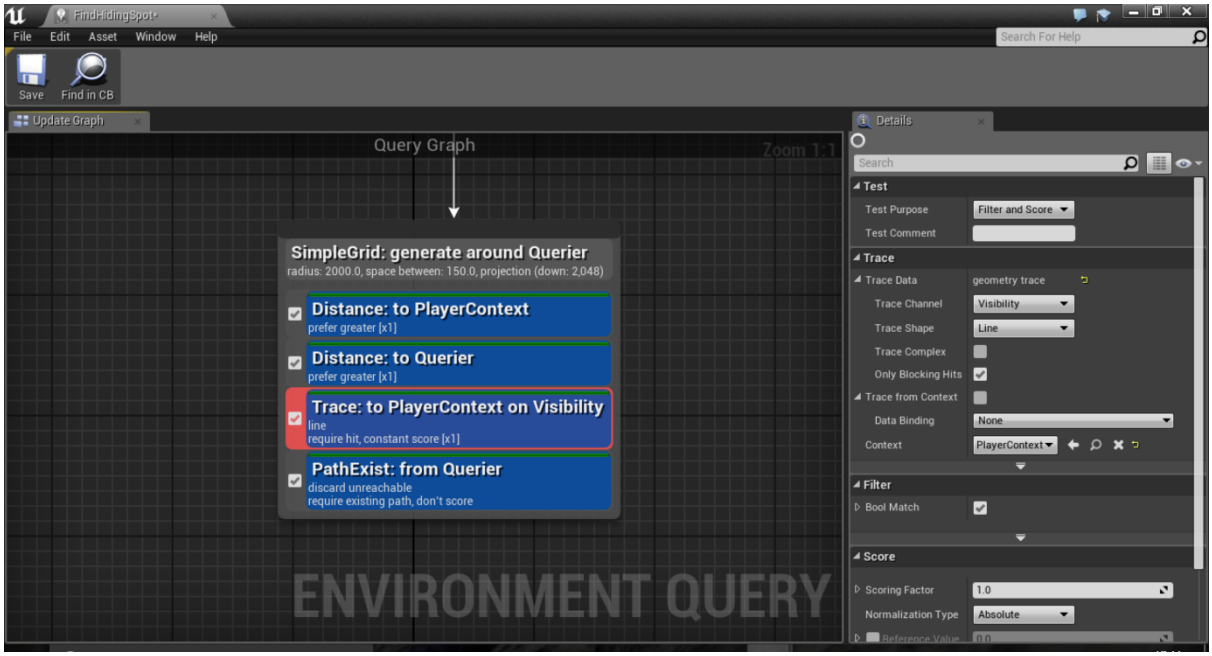
- Ahora vamos a añadir los "*Tests*" que permitirán filtrar y evaluar el mejor punto para esconderse
    - Añadimos un *Test* de distancia
        a. *Distance To: PlayerContext*
        b. *Test Purpose: ScoreOnly*, con lo que no será un filtro sino simplemente un evaluador
        c. Con esto estaremos evaluando: cuanto más lejos del jugador mejor



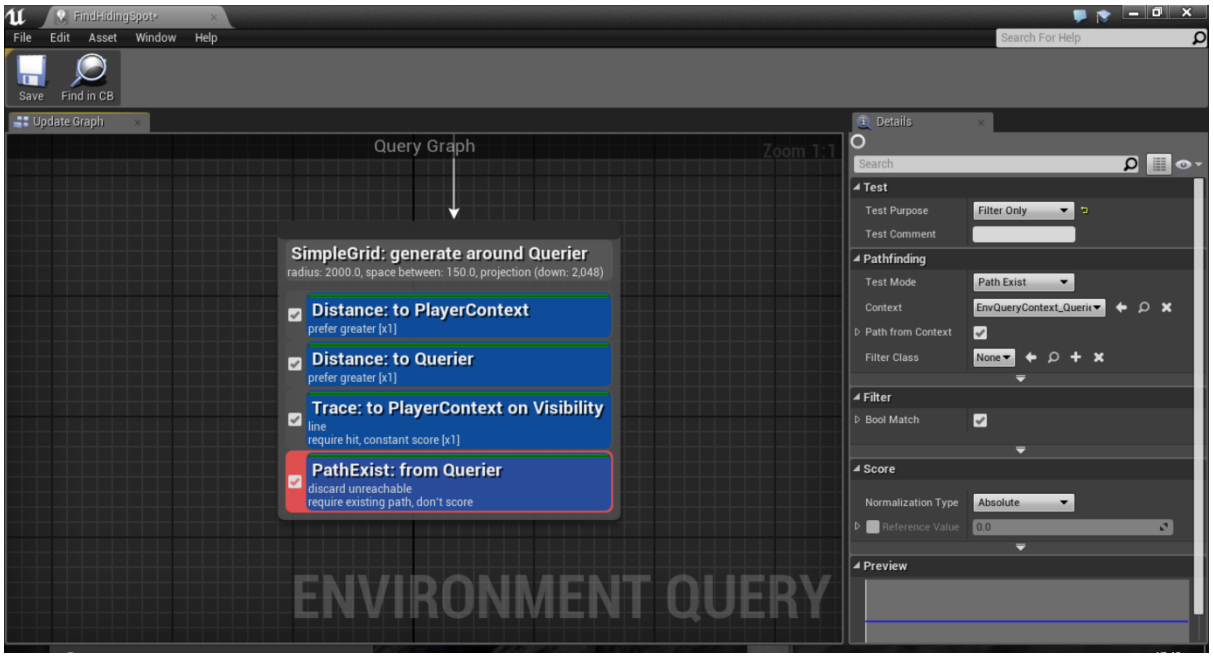
- Añadimos un segundo *Test* de distancia:
    a. Dejamos *DistanceTo* como está: *EnvQueryContext*, con esto haremos las consultas de la IA al ítem, al contrario del caso anterior que era del player al ítem.
    b. *Test Purpose: ScoreOnly.*
    c. En el campo *Score->ScoringEquation: InverseLinear.*
    d. Con esto favoreceremos los *items* que estén cerca de nuestra IA para evitar que se desplace innecesariamente.



- Añadimos un *test* de *Trace*
    a. Este test lanzará rayos que colisionarán con la geometría desde cada ítem al contexto que se elija para determinar si hay o no visibilidad.
    b. Cambiamos *Context* a *PlayerContext* para eliminar los ítems visibles por el jugador.



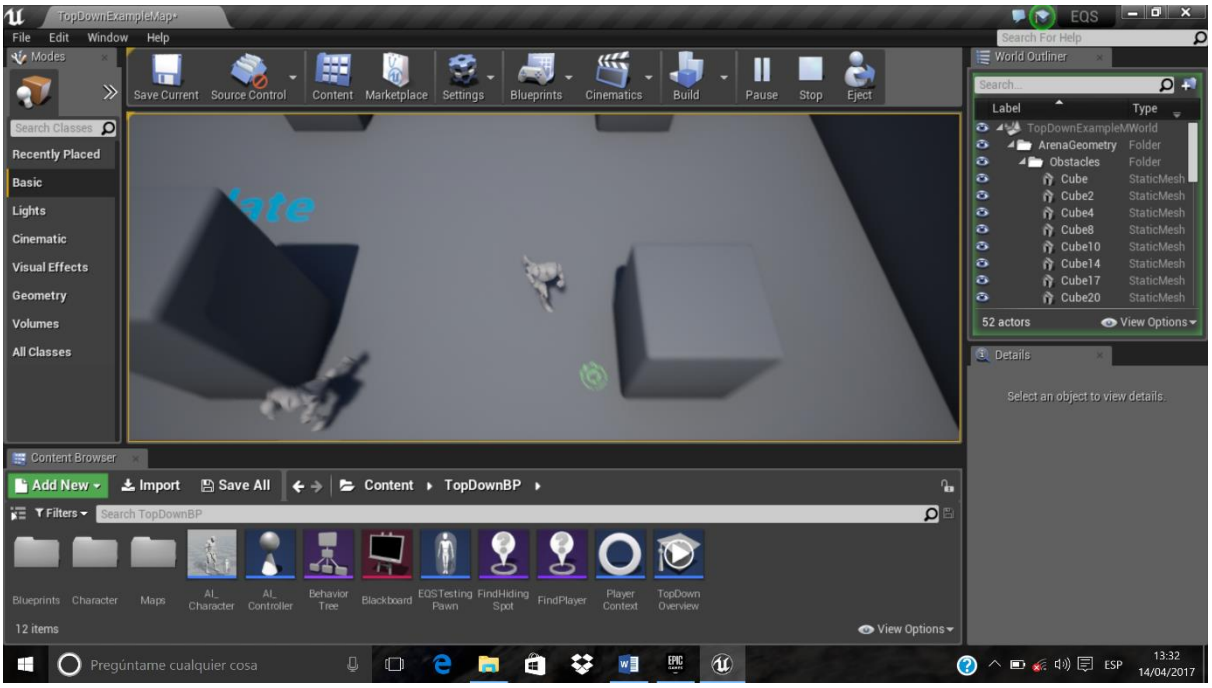
- Añadimos un *Test Pathfinding*, con esto aseguraremos que nuestra IA tendrá un camino válido hasta el punto en el que se esconderá.
    a. *Test Purpose: Filter Only*. Si no filtráramos podríamos tener un problema, ya que nuestra IA podría seleccionar, en caso de que no hubiera ninguno navegable, alguno al cual no pudiera encontrar un camino para llegar a él. Esto es debido en parte a que la generación de *NavMesh* de Unreal no elimina los triángulos que hay dentro de la geometría, como ocurre en el ejemplo que estamos manejando, el *TopDownCharacter template*, dentro de las columnas que hay en el nivel. Dentro de las mismas hay *NavMesh* y por tanto, algún punto del *Grid* que cayera dentro de éstas podría llegar a ser evaluado, generando el problema.



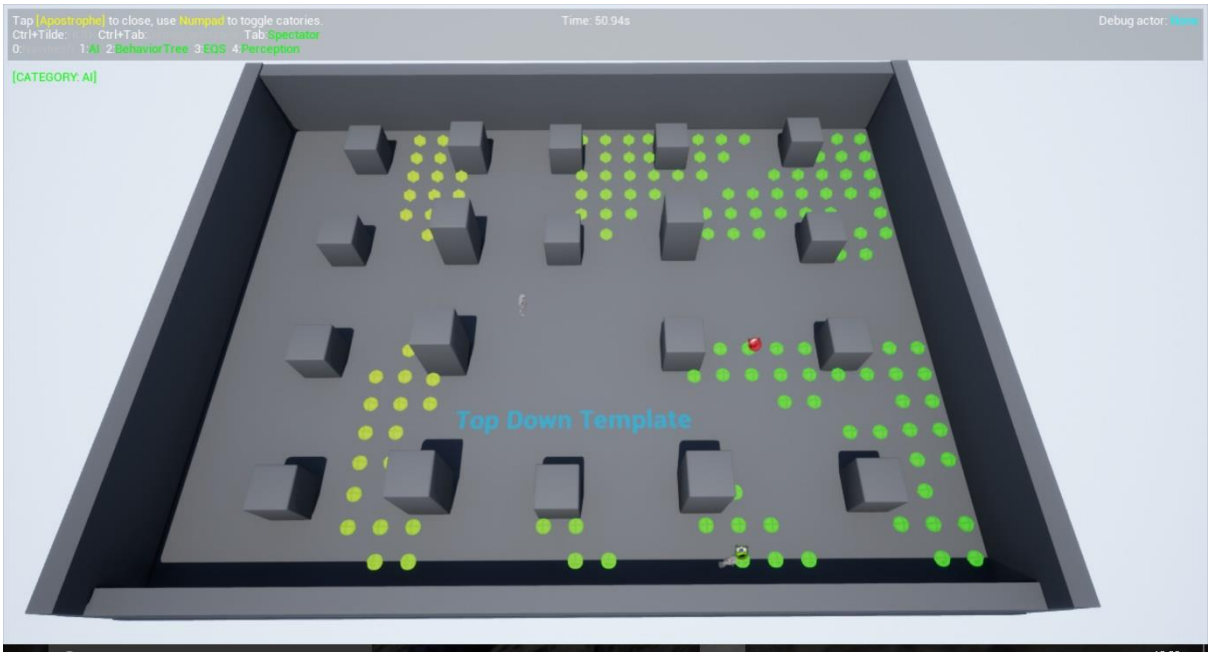
l. ¡Probad! Veremos cómo la IA huye de nuestro jugador hasta situarse en un punto desde el que no "le veríamos". Si le flanqueamos volverá a huir seleccionando un nuevo punto que cumpla con los criterios de las consultas.

m. Daros cuenta que estamos filtrando y evaluando un conjunto de posiciones relativamente grande; además alguna de nuestros Test son costosos, pues implican testeos de visibilidad. Podéis comprobar que, si disminuyerais, por ejemplo, el parámetro *Space Between* del generador de *FindHidingSpot* un poco cada vez que nuestra IA tuviera que volver a elegir un punto al que dirigirse se produciría un pequeño parón, lo que significa que nos hemos pasado con la densidad de puntos. Ojo: no bajar mucho este valor que podríamos dejar colgado Unreal o el PC (😁)

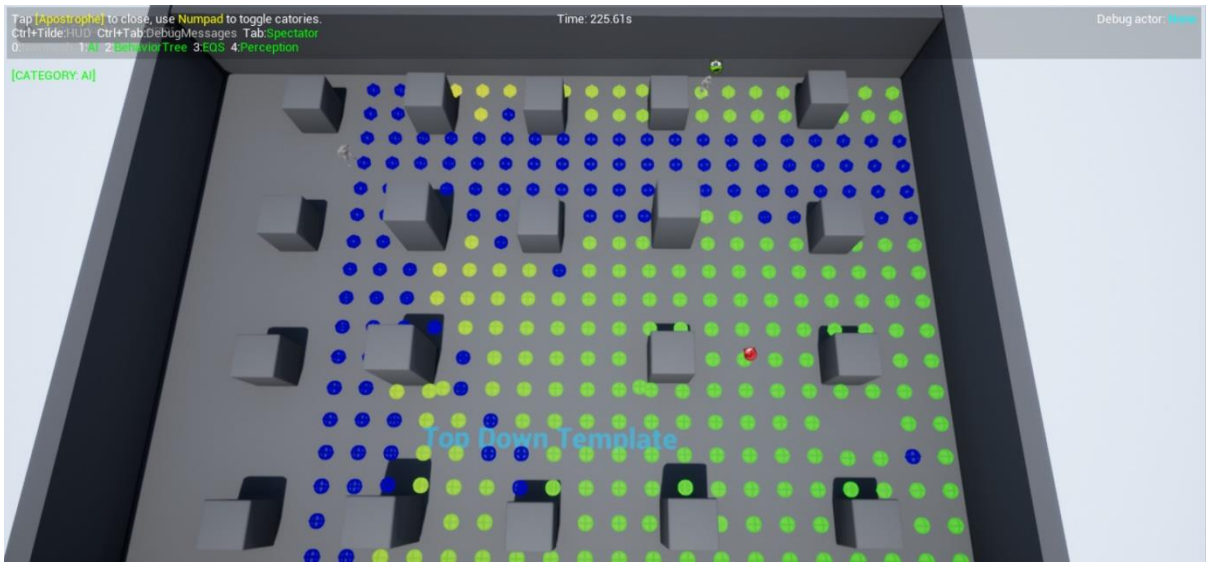
n. Se pueden crear nuevos generadores y *Contexts* tanto en *Blueprint* como en C++, pero para crear nuevos *Tests*, de momento, sólo lo podremos hacer utilizando C++.



o. Para poder visualizar los resultados de nuestras consultas Unreal nos proporciona el *EQS Testing Pawn*
   - Nos permite visualizar qué está haciendo nuestras *Queries*
   - Su representación será en forma de esferas, una por cada ítem que componga la *Query*, que variarán de color en función del rojo al verde para indicar los valores de evaluación de nuestras consultas.
   - Las esferas azules nos indicarán qué ítems han sido filtrados y no han sido evaluados.
   - Para utilizarlo debemos crear un *EQS Testing Pawn*
     - *New Blueprint Class -> EQS Testing Pawn*
     - Creamos una instancia del mismo en el nivel
     - En el campo de configuración *Query Template* seleccionaremos la *EQS* de la que queremos visualizar la información
     - Para poder ver durante el juego la información deberemos seleccionar la instancia de *EQS Testing Pawn*
     - Para refrescar los valores que se muestran para que sean los de la última consulta realizada debemos hacer clic en el botón + de *QueryConfig* en el apartado *EQS* para que regenere la información



- Para ver los elementos filtrados seleccionaremos *Draw Failed Items*

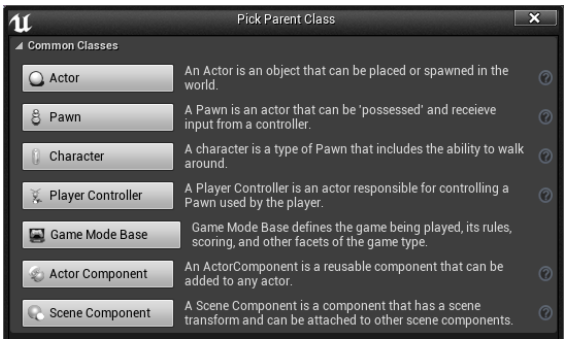


- Para ver los valores de puntuación de los elementos seleccionamos *Draw Labels*

ANEXOS

ANEXO 1: Creación de un Character "desde cero" tal y como hicisteis en la práctica 1

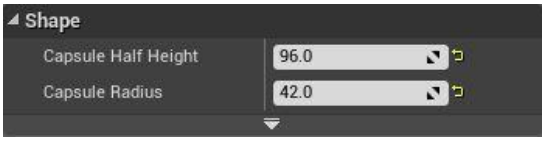
Ahora tenemos que crear un *Character* para que nuestra IA lo maneje, para ello en la carpeta de los *blueprint* del proyecto, en el panel ***Content Browser***, hacemos clic con el botón derecho y se abrirá un menú desplegable con opciones para crear tipos de *blueprints*.



Seleccionamos ***Character*** y esto nos creara un nuevo *Character* en nuestra carpeta de blueprints al que llamaremos ***"AI_Character"***.

Para configurar el ***AI_Character*** hacemos *doble clic* sobre él y se abrirá el editor:

1. Seleccionamos el componente ***Mesh*** en el panel de **Componentes**.
2. En el panel ***Details*** buscamos la categoría ***Mesh*** y asignamos el ***SK_Mannequin*** *al* ***Skeletal Mesh***. De esta forma nuestro *Character* usará esta representación visual.
3. Movemos la ***Mesh*** hacia abajo para que este centrada en la capsula y la **rotamos** de forma que mire hacia donde apunta la flecha azul.
4. Ahora tenemos que configurarle las animaciones, buscamos en el panel ***Details*** la sección ***Animation*** y le asignamos el ***ThirdPerson_AnimBP*** a la propiedad ***Animation Blueprint Generated Class.***
5. Seleccionar el ***CapsuleComponent*** del panel ***Components*** y ajustar en el panel ***Details***, la ***Capsule Half Height*** y el ***Capsule Radius*** para que se ajusten al ***Skeletal Mesh***.

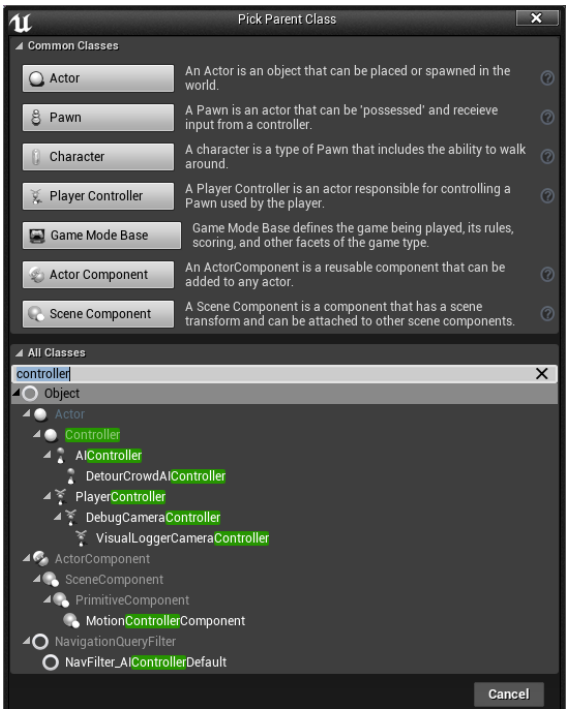


Por ultimo en ***Components*** seleccionamos el componente ***Character Movement*** y en el panel ***Detail*** configuramos el ***Nav Agent Radius* = 42** y el ***Nav Agent Height* = 192** (lo que mide la capsula de alto).

ANEXO 2: Creación AIController

Una vez tenemos nuestro ***AI_Character*** lo arrastramos al mapa para crear una instancia y la llamamos "*MyCharacter*". Además, vamos a necesitar un ***AIController***, en UE4 existe una jerarquía de clases que Actor/Controller que permiten controlar un *Pawn*, así es como el player puede controlar cualquier par mediante su *PlayerController*, en el caso de la IA tenemos el *AIController*.

El ***AIController*** nos ofrece funcionalidad para mover al *Pawn* y usar un *Behavior Tree* para controlar las decisiones (pero eso lo veremos más adelante). Creamos el AIController y lo llamamos "*EnemyPatrol*". Hacemos lo mismo que para el *Character*, botón derecho en la carpeta de blueprints, pero en este caso no vamos a crear una ***Common Class*** sino que vamos a tener que buscar en ***All Classes*** *el AIController:*



Una vez tenemos nuestro *AIController* tenemos que asignárselo a su *Character*, para ellos seleccionamos la entidad "*MyCharacter*" y ponemos en el buscador de propiedades (panel ***Details***) *AIController* y le asignamos "*EnemyPatrol*". Con esto lo que conseguimos es que cuando al *Pawn* no se le haya hecho *Possess* de ningún controlador se le haga con una instancia de "*EnemyPatrol*".

ANEXO 3: Creación NavMesh

Necesitamos tener una representación del mundo más sencilla que la capa física para poder calcular caminos de un punto a otro, hay varias formas de representar el espacio (*grid, waypoints, navmesh*, etc.) y depende mucho del tipo de juego y del tiempo que tengamos para implementarlo (en el caso de no usar UE4). UE4 utiliza *NavMesh* que es prácticamente el estándar en la industria en representación espacial, en el pasado se utilizaba mucho más *waypoints* porque son más fáciles de implementar y ocupan menos espacio. Pero gracias a mucha investigación en generación de *NavMesh* a lo largo de los años se conoces técnicas bastante sencillas y conocidas para generarlas por lo que cualquiera puede hacer un algoritmo de generación de cero.

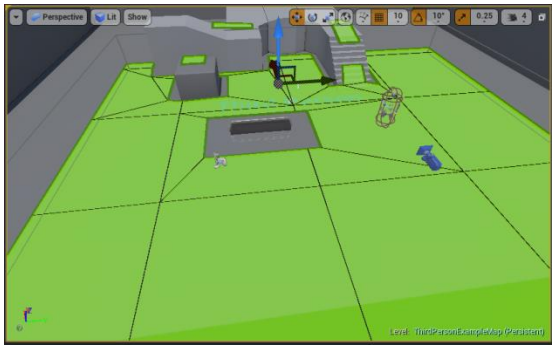
La *NavMesh* nos aporta más porque tiene información de los bordes de la zona navegable.

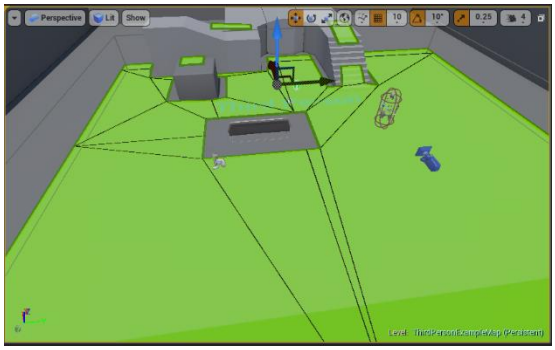
Para que se genere una *NavMesh* en el nivel actual tenemos que crear un ***NavMeshBoundsVolume*** que abarque toda la zona donde queremos que la IA pueda navegar. Este tipo de volumen se encuentra en ***Modes/Volumes*** y simplemente tenemos que arrastrarlo a nuestra escena

Para mostrar la información de navegación podemos marcar la opción ***Navigation*** o usar el atajo de teclado **P**. Si queremos verla *ingame* sacamos la consola y ponemos **'show navigation'**.

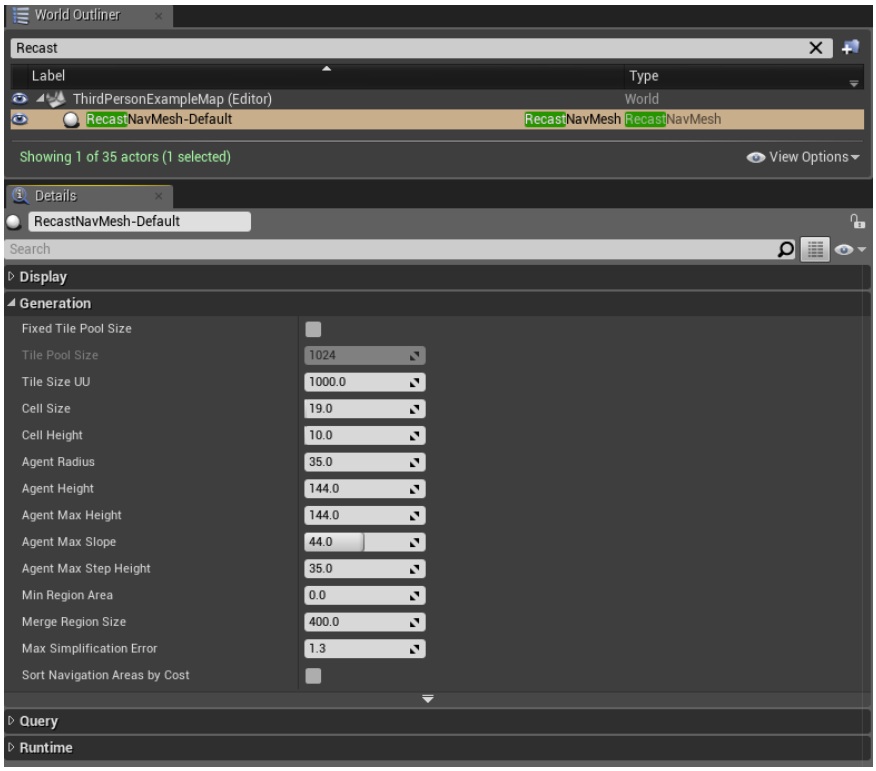
La configuración de la generación de la *NavMesh* está en un nodo que se llama ***RecastNavMesh-Default*** en el ***World Outliner***. En él se pueden configurar los valores por defecto para la generación de la NavMesh en este nivel, los parámetros más importantes son:

- Cell Size: Es el tamaño del *voxel* (un pixel en 3d, un bloque de minecraft ☺) que representa un espacio navegable. Este tamaño se refiere al tamaño del suelo del *voxel* es decir ancho y profundidad. Esto lo que controla es la resolución de la *NavMesh*, cuanto más pequeño sea más fiel a la colisión real será la *NavMesh* pero más costosa de generar. Al ser la generación offline esto nos da un poco igual a no ser que queramos regenerar la *NavMesh* en caliente por destrucción de objetos dinámicos, por ejemplo.
- *Cell Height*: Altura del voxel, este valor se debe corresponder con la máxima altura que puede subir una entidad andando, esto quiere decir que todo escalón más bajo que esta altura no existe para la *NavMesh*.
- *Tile Size UU*: Tamaño de celda. Si bajamos este valor el tiempo de generación aumenta, pero la NavMesh generada es más limpia, a la izquierda vemos una *NavMesh* generada con 560 y a la derecha con 1500.





- *Min Region Area*: Quita de la *NavMesh* las regiones que están aisladas que sean menores que el tamaño configurado.



En esta ocasión no hace falta que creéis obstáculos :P