Guion Ejercicio Behavior Trees en Unity

En este ejercicio vamos a crear una IA básica usando el plug-in gratuito BehaviorBkicks de BehaviorTrees en Unity.

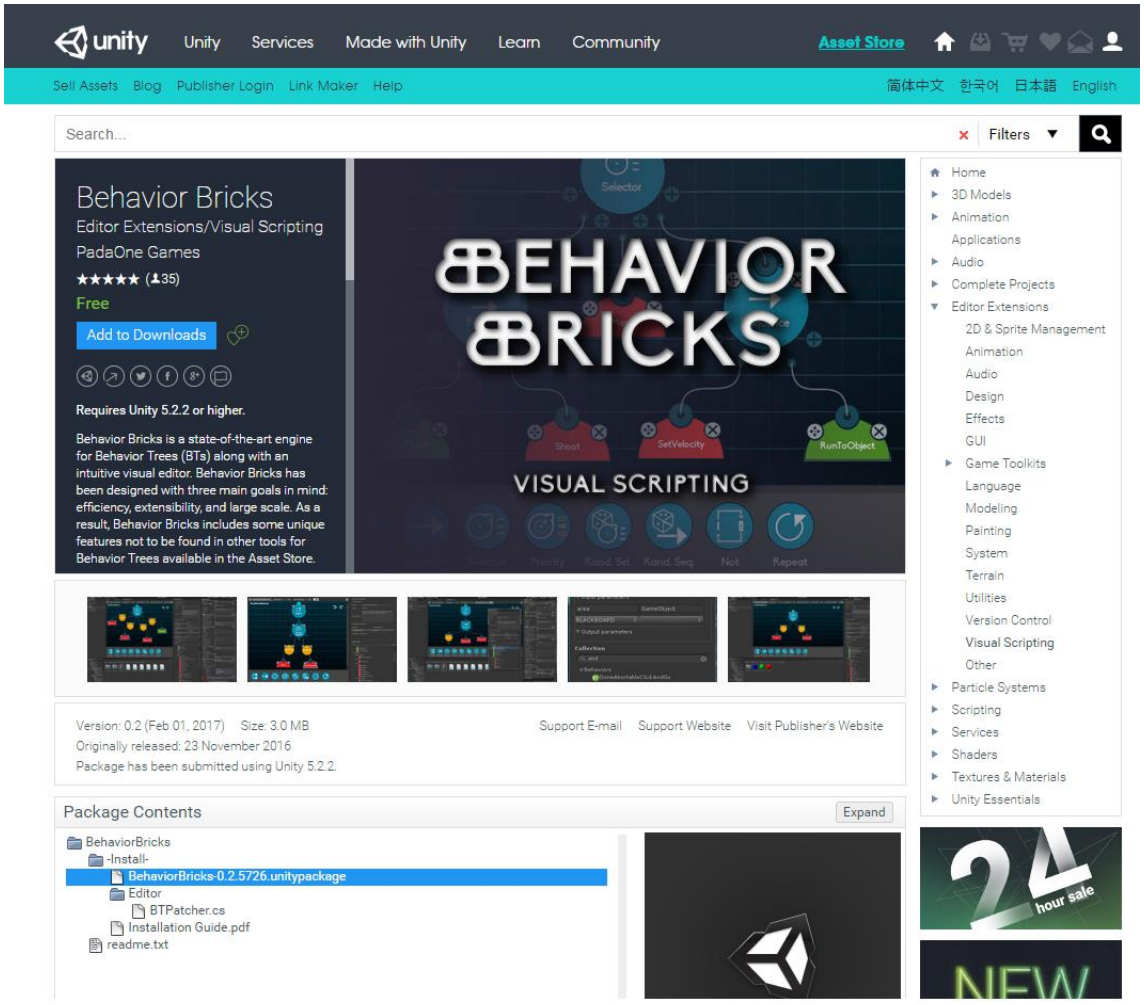
Primero instalaremos el plug-in BehaviorBricks que nos proporcionará una interfaz gráfica de creación de BehaviorTrees así como unas cuantas funcionalidades básicas en integración con Unity.

A continuación, añadiremos comportamientos básicos a un enemigo y, gracias a la versatilidad de los BehaviorTrees, controlaremos un personaje jugador.

NOTA: Debido a las actualizaciones (o no actualizaciones, mejor dicho) de BehaviorBricks con la última versión de Unity puede que el plugin no os funcione. Importante usar una versión de Unity soportada.

Parte 1: Setup BehaviorBricks

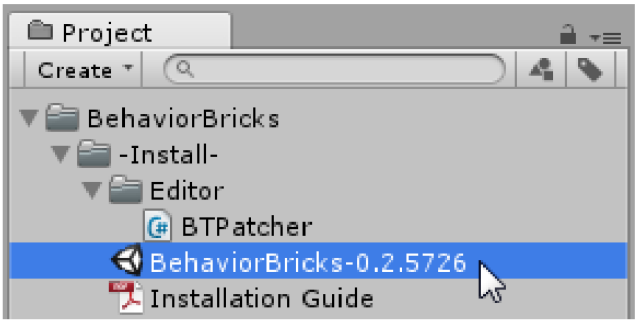
- Creamos un nuevo proyecto en Unity: File/NewProject de tipo 3D
- Lo primero que haremos será ir al AssetStore para descargar el *PlugIn* con el que trabajaremos en este ejercicio: BehaviorBricks

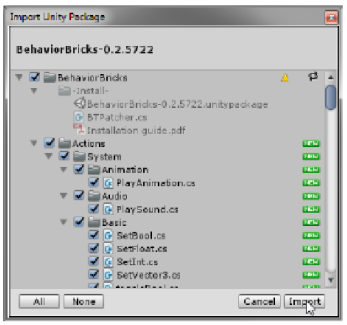


(https://www.assetstore.unity3d.com/en/#!/content/74816)

- Lo abrimos con Unity y tendremos en nuestro proyecto la carpeta BehaviorBricks

- Hacemos doble clic en el archivo de Paquete de Unity e importamos todo el contenido





- No pedirá confirmación de haber hecho un backup del proyecto (como acabamos de empezar no será un problema no tenerlo :P), así que “clicamos” en “I Made a backup, Go Ahead!”

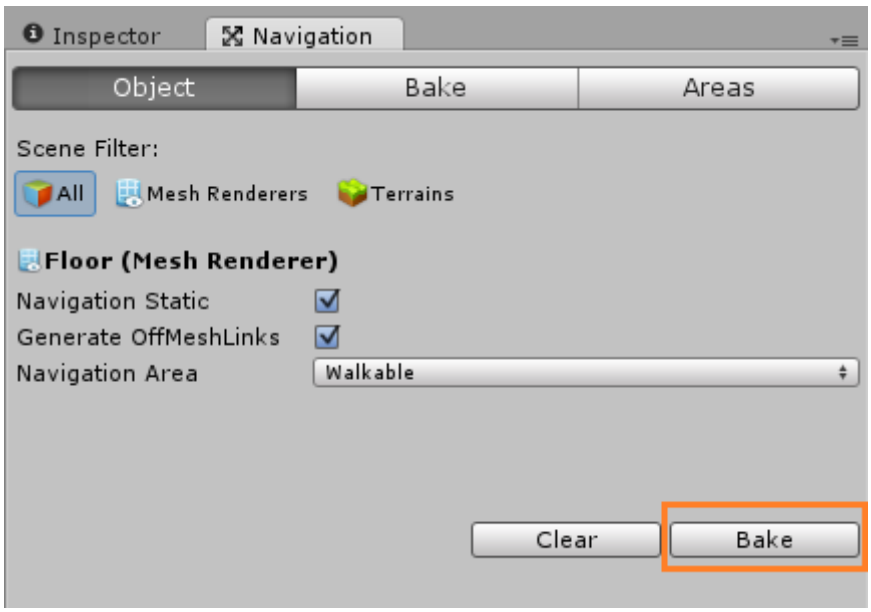
- Si tenemos algún problema podemos seguir la guía de instalación.

- Dependiendo de la versión de Unity que tengamos instalada puede que la interfaz del código de las clases de Navegación sea obsoleta en BehaviorBricks, por lo que nos aparecerán una serie de errores de compilación que nos advertirán de dichos problemas. Unity es muy majete y nos aconsejará cómo arreglar dichos problemas para adaptarse a los cambios de código en la Navegación.

- Tras corregirlos todos ya estaremos listos para comenzar a usar el plug-in!

Parte 2: Crear la escena base

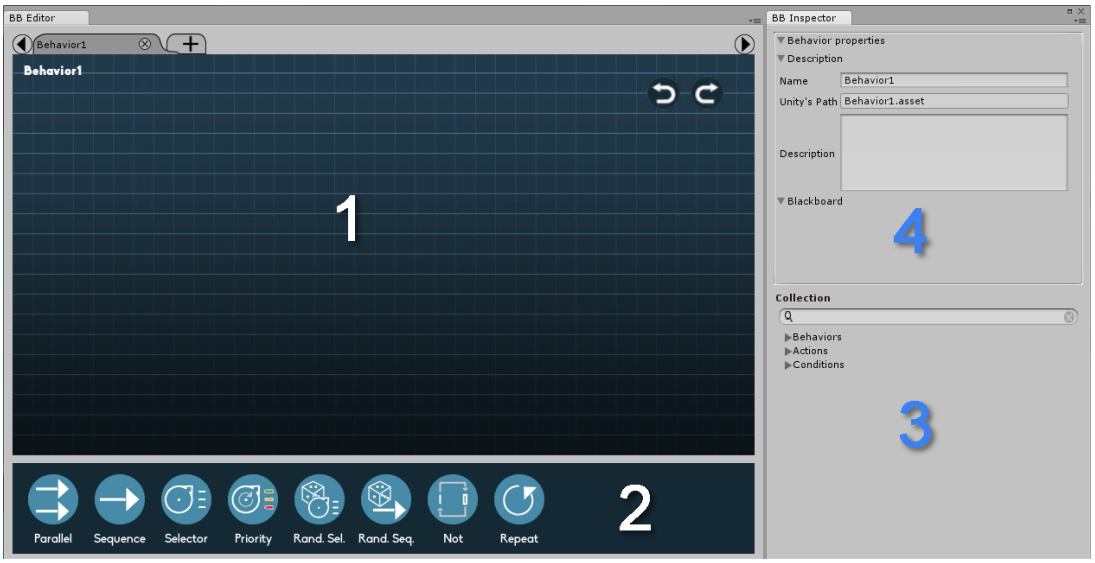
- Lo primero vamos a crear el suelo de nuestra escena: Por ejemplo, desde el menú GameObject/3DObject/Plane. Lo renombramos a Floor, lo recolocamos en la posición (0,0,0) y cambiamos su escala a (5,0,5). Para que podamos crear una *Navigation Mesh* que use este suelo como zona transitable deberemos activar su CheckBox *Static* en el *Inspector* (está colocado justo al lado del nombre)
- Vamos a colocar la cámara principal que Unity nos ha creado en la escena de manera que podamos ver todo lo que está ocurriendo. La movemos a la posición (0, 20, -30) y ajustamos su rotación a (45,0,0). De esta manera debería quedar visible todo el plano que creamos en el punto anterior.
- Añadimos una luz direccional y la mantenemos con los valores por defecto que trae.
- Ahora vamos a crear al personaje jugador y al enemigo.
    - Para el jugador elegimos una esfera, la renombramos como *Player*. La colocamos en la posición (0, 0.5, 0), es decir, en el centro del plano. Recordad que por defecto los elementos que creemos tienen tamaño de 1 unidad, con lo que al colocarla en la coordenada Y 0.5 estamos dejando la esfera apoyada sobre el suelo.
    - Para el enemigo creamos un cubo, la renombramos como *Enemy*. La colocamos en la posición (20, 0.5, 20), es decir, cerca de una esquina del plano.
    - Creamos tres materiales nuevos con los que “pintaremos” nuestro suelo, al jugador y al enemigo.
        - Uno lo llamaremos Green, seleccionaremos este color en el Albedo de su Inspector y lo utilizaremos para el suelo. (Recordad que para aplicar el material basta con arrastrarlo hasta el objeto que queramos “pintar”)
        - Otro será Blue y se lo asignaremos al player.
        - Por último, tendremos Red, con el que teñiremos al enemigo.
    - Para terminar de configurar la escena vamos a crear una malla de navegación que utilizará tanto el enemigo como el jugador para su búsqueda de caminos.
        - Para ello seleccionaremos el objeto Floor y vamos a Window/Navigation. Nos abrirá la pestaña de creación de NavMesh, donde, con la configuración por defecto, pulsaremos el botón Bake.



- Para que nuestros personajes *Enemy y Player* puedan navegar haciendo uso de la NavMesh debemos añadirles el componente que se encargará de dicha tarea. Este será el NavMesh Agent. Bastará con añadírselo a ambos y dejar la configuración por defecto.
- Ya tenemos la escena preparada para poder implementar los comportamientos mediante la herramienta de BehaviorTrees.

Parte 3: Primer comportamiento

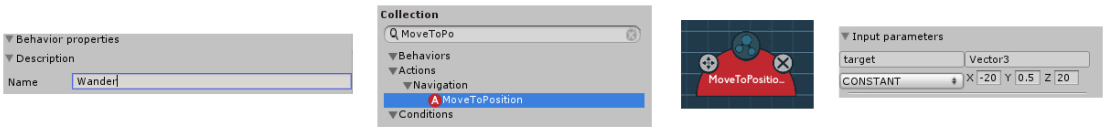
- Vamos a crear un primer comportamiento sencillo para el enemigo. Comenzaremos aprendiendo a utilizar la herramienta visual de BehaviorTrees así como a hacer uso de la Blackboard para poder utilizarla como comunicación entre los distintos nodos del árbol.
- Abriremos el editor de BehaviorBricks: Window/BehaviorBricks. La ventana del editor está dividida en 4 partes:



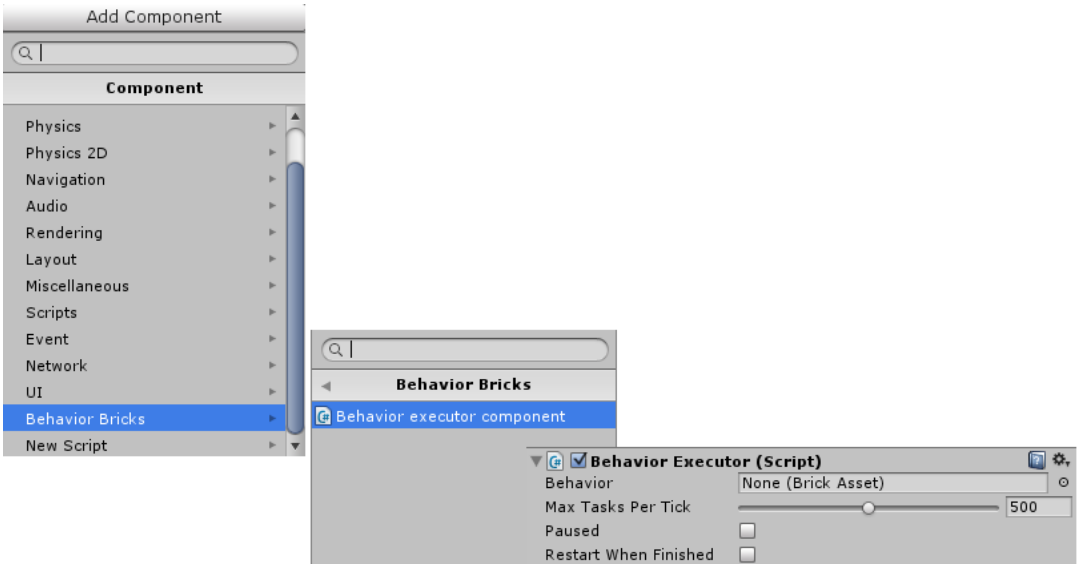
1. El panel gráfico de edición del árbol en sí. Tenemos un sistema de pestañas para cada árbol diferente que editemos.
2. Los nodos “composite” disponibles
3. El panel de propiedades, donde veremos tanto las propiedades del árbol que estemos editando, si pinchamos en una zona vacía del panel gráfico, como las propiedades del nodo del árbol si tenemos alguno seleccionado.
4. Behaviors Collection: Aquí estarán todas las acciones y condiciones que nos da por defecto BehaviorBricks, así como las que pudiéramos crear para ampliar funcionalidad.

Como primer ejemplo vamos a hacer un comportamiento que mueva al enemigo a una posición fija preestablecida.

- Creamos un nuevo BehaviorTree pulsando en el botón + del panel gráfico y lo renombraremos desde el *BB Inspector* a *Wander*. Al renombrarlo aquí veremos cómo cambia de nombre en la *Collection*. El nombre del *Asset* y el que vemos en el Inspector no tienen por qué coincidir.
- En la *Collection* buscamos el nodo *MoveToPosition*. Podemos buscarlo con la barra de búsqueda o navegar hasta encontrarlo. Lo arrastramos al editor gráfico para crear el nodo.
- Podemos ver sus propiedades en el inspector. En la zona de *Input Parameters* tendremos, para cada nodo, la información que requiere para funcionar. En este caso vamos a establecer el valor de la variable Target, que nos indica que espera recibir un Vector3, a (-20, 0.5, 20)



- Los cambios en los BehaviorTrees se salvan automáticamente.
- Para que nuestro *Enemy* pueda ejecutar BehaviorTrees debemos añadirle el componente *BehaviorExecutor* que podemos encontrar en la carpeta de BehaviorBricks.



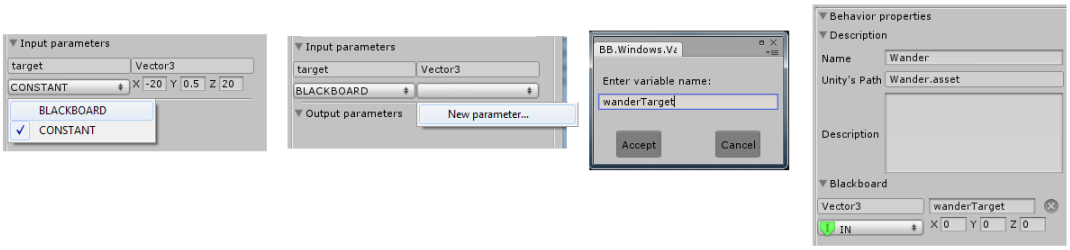
- En las propiedades del Behavior Executor vemos el campo *Behavior* donde deberemos arrastrar el árbol que queremos que se ejecute.
- En los *AssetFolders* veremos el árbol “New Brick Asset”, si es que no lo habíamos renombrado antes, y lo renombramos a *Wander*. Lo arrastramos hasta el campo Behavior del *Behavior Executor* de Enemy.
- Ya podemos probar nuestro primer comportamiento simplemente haciendo *Play* de la escena. Veremos que el enemigo se desplaza hasta la posición que hemos fijado en el nodo *MoveToPosition*. Si se nos hubiera olvidado añadir el componente *NavMesh Agent* a nuestro *Enemy*, Unity, que es muy listo, lo habría hecho por nosotros mostrándonos un bonito Warning advirtiéndonos que los necesitaba para navegar y que lo había añadido.

Parte 4: Usando la BlackBoard

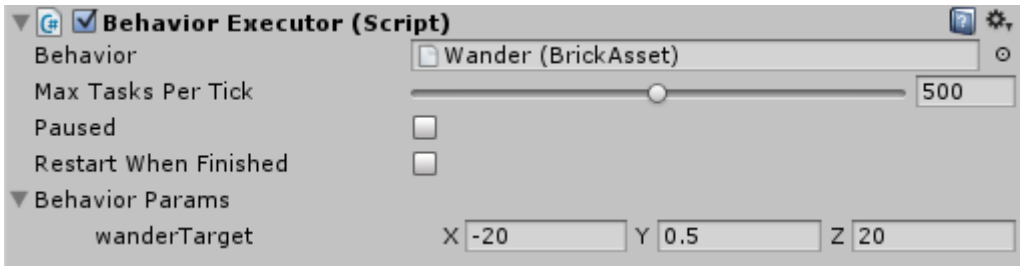
- Lo que tenemos ahora es un Behavior Tree con un solo nodo en el que hemos "hardcodeado" una posición. Para aprender a usar la Blackboard, que nos servirá como memoria y/o comunicador del BehaviorTree:

Vamos a hacer que la posición de destino prefijada para el Enemy se cree desde fuera del nodo, a través de la Blackboard.

- Abrimos el árbol de *Wander* que acabamos de crear. Seleccionamos el nodo MoveToPosition. En el *BB Inspector* vemos que donde antes añadimos la posición *Target* hay un selector en el que pone CONSTANT. Vemos que podemos cambiarlo por BLACKBOARD. Seleccionamos *New Parameter* para crear nuestra variable en la BlackBoard. La llamaremos WanderTarget.

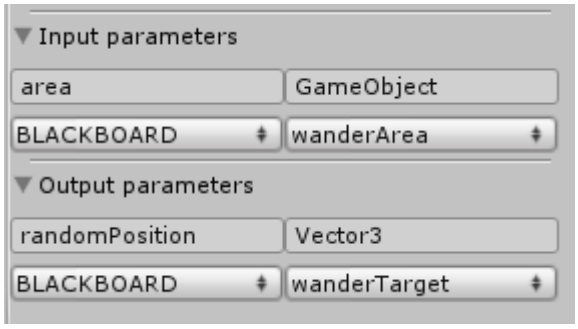


- Si hacemos clic en una zona vacía del editor gráfico veremos en el *BB Inspector* las propiedades del Behavior vemos que podemos editar el parámetro *WanderTarget* que hemos "publicado" a través de la Blackboard.
- Pero más interesante es que podemos editarlo en el componente *BehaviorExecutor de Enemy.*
  - Seleccionamos Enemy en la escena.
  - Vemos sus propiedades del componente *BehaviorExecutor.*
  - Escribimos el valor (-20,0.5,20)
  - Al ejecutar veremos que se comporta igual que antes.

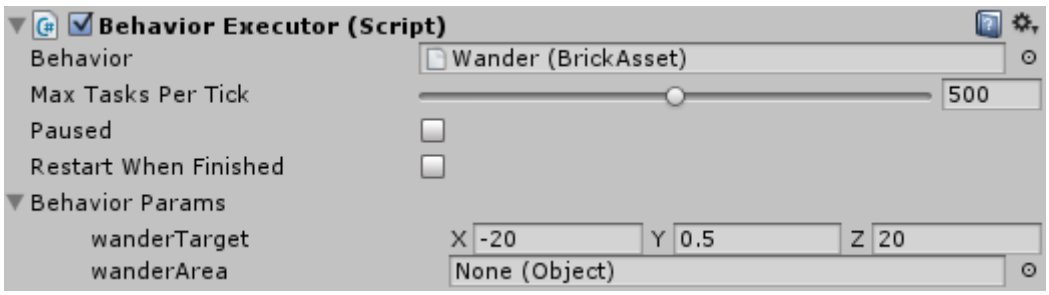


Parte 5: Crear un verdadero árbol (y un verdadero comportamiento Wander)

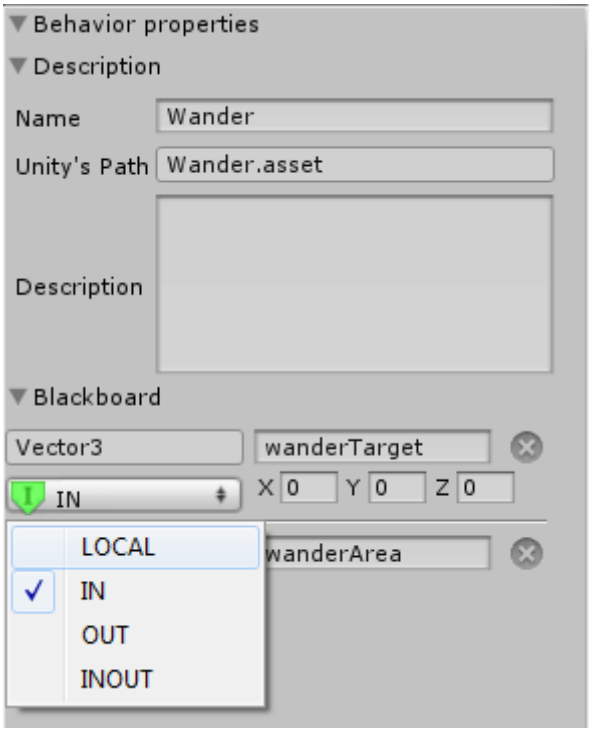
- Vamos a editar de nuevo el Behavior Tree: Wander.
- Buscamos y añadimos al árbol la acción *GetRandomInArea,* que encuentra una posición aleatoria de un GameObject que se le introduzca y la escribirá en la BlackBoard.
- Editamos los parámetros de la acción de manera que *area* pase a ser una variable de la Blackboard y la llamaremos *WanderArea*. Para *RandomPosition* seleccionaremos la variable de la Blackboard que creamos antes llamada WanderTarget. Con esto lo que hemos hecho ha sido "linkar" la acción de GetRandomInArea con MoveToPosition a través de la blackboard.



- Seleccionamos el objeto *Enemy*, en el componente *Behavior Executor;* vemos que además del antiguo parámetro WanderTarget ha aparecido el nuevo parámetro WanderArea. El caso es que WanderTarget ya no es un parámetro que queramos introducir desde esta interfaz, sino que será el valor de comunicación entre los nodos GetRandomInArea y MoveToPosition.

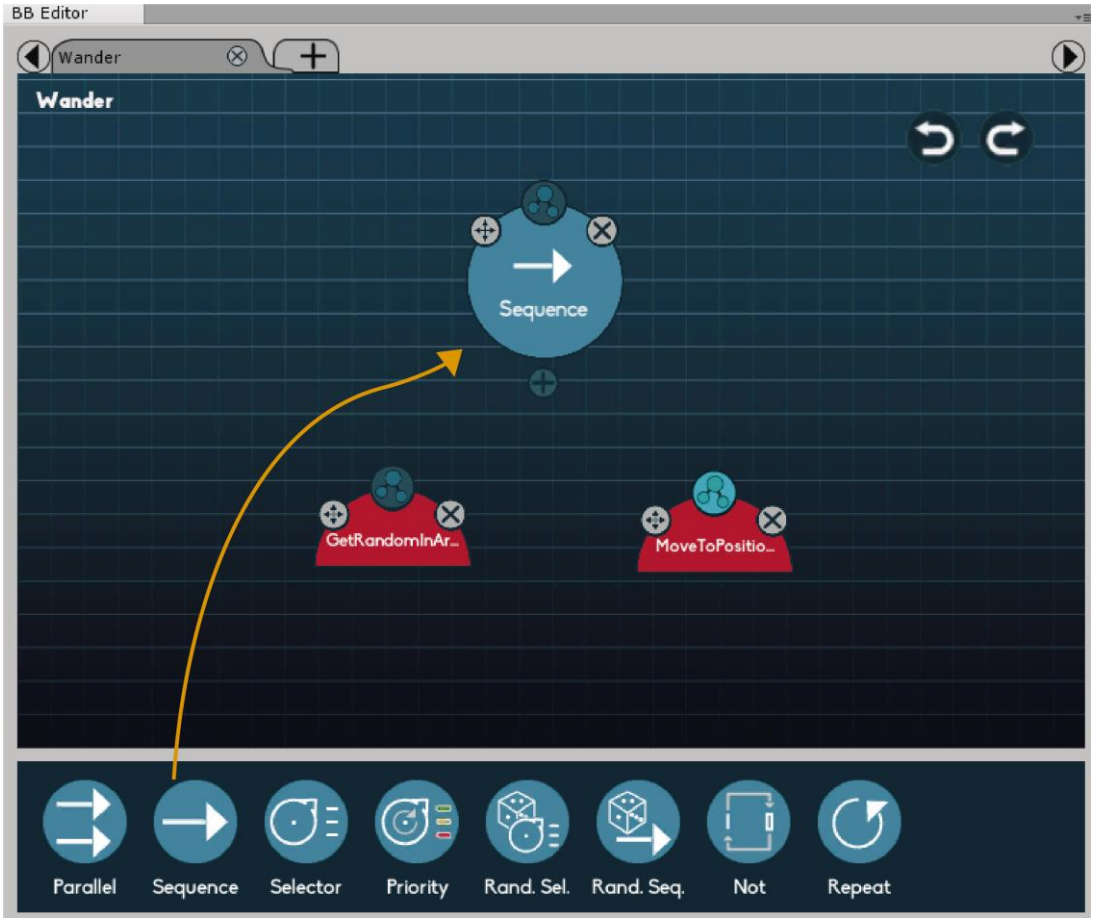


- Para hacer que sea un parámetro local editaremos el Behavior Wander. En las propiedades del árbol, es decir, clicando sobre una zona vacía de la parte de edición gráfica, veremos en la pestaña de Behavior Properties que ambas variables son de tipo IN. Cambiamos Wander Target para que sea de tipo LOCAL. Con esta acción habrá desaparecido de la parte de edición del componente de *Behavior Execution del Enemy.*

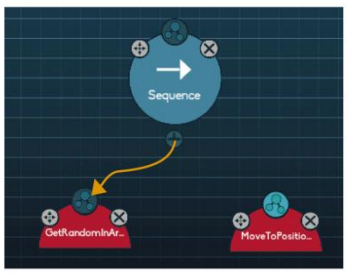


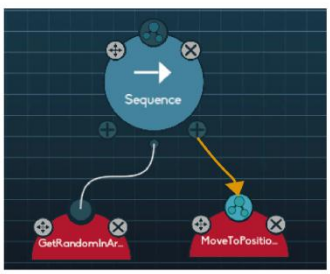
- Ahora mismo tenemos dos acciones en nuestro BehaviorTree Wander, pero ¿cómo se ejecutarán? Sólo una de las dos será ejecutada de hecho, la que esté marcada con el circulo de color azul, mientras que la otra acción no llegará a ser ejecutada; no es esto lo que queremos, así que tendremos que añadir nodos composite para decidir el flujo de ejecución.

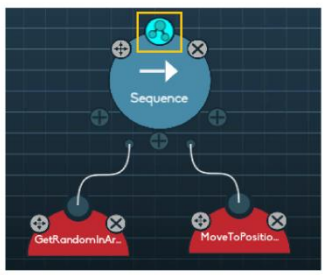
- Vamos a añadir un nodo Secuencia al árbol arrastrándolo desde la zona de “Composites” hasta el área de edición.



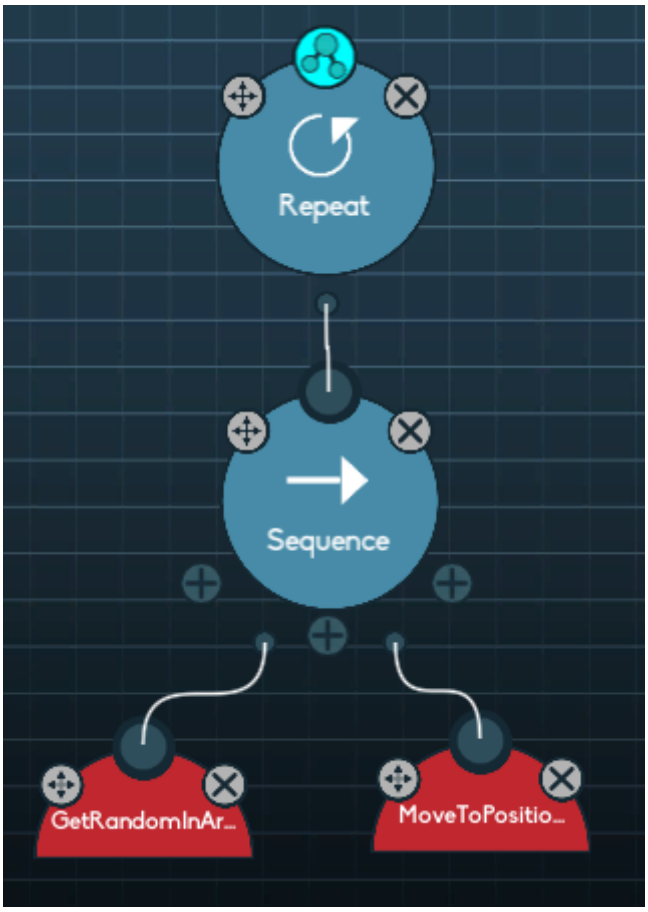
- Ahora uniremos las dos acciones para que se ejecuten en secuencia. Tras colocar las acciones como hijas de la secuencia veremos que el nodo que se ejecutará primero será el de la Secuencia. Con esto ya tendríamos un árbol válido pero que se ejecutaría una sola vez y después terminaría.







- Por tanto, para conseguir que nuestro enemigo no sólo vaya a una posición aleatoria una vez debemos añadir al árbol un nodo Repeat. Con esto conseguiremos que una vez terminada la secuencia ésta se vuelva a ejecutar, con lo que tendremos al enemigo dirigiéndose consecutivamente a posiciones aleatorias.



- Para que podamos probar nuestro árbol sólo nos queda un pequeño paso, que es establecer el GameObject que utilizará la acción GetRandomInArea para elegir su posición aleatoria.
    - Para ello debemos añadirle un BoxCollider al GameObject Floor de dimensiones (10,0,10)
    - Y arrastrar dicho GameObject Floor al parámetro WanderArea del componente de BehaviorExecution de nuestro Enemy.

- ¡Ya podemos probar el comportamiento!

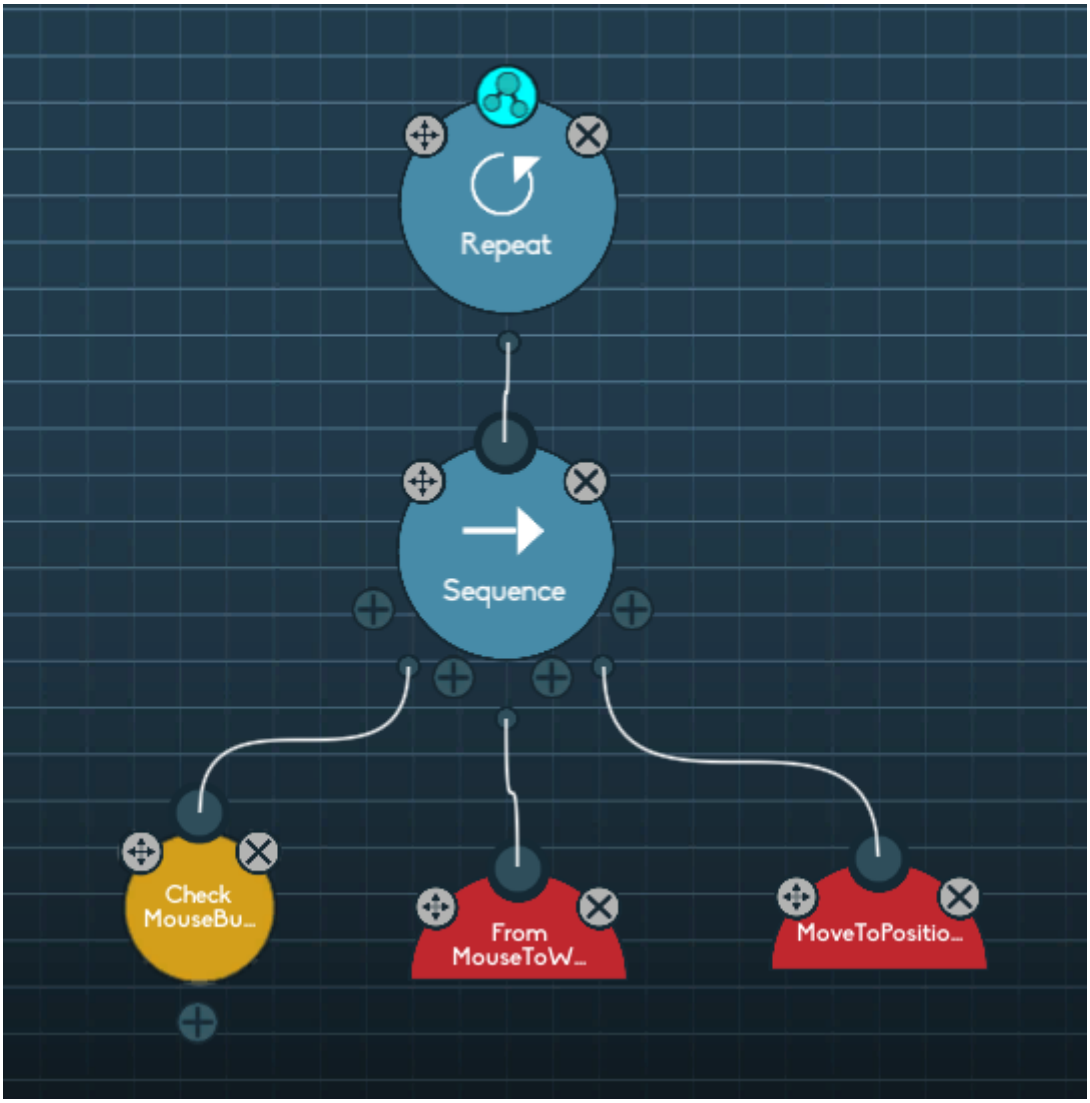
Parte 6: Controlar al Player

- Con los BehaviorTrees además de controlar a IAs también podemos controlar jugadores, flujo de misiones, de interfaz, de sonidos, etc. A continuación:

Vamos a aprender a controlar al jugador con el ratón por medio del árbol.

- Abrimos el editor de BehaviorBricks y creamos un nuevo BehaviorTree al que llamaremos ClickAndGo
- Buscamos y añadimos la condición CheckMouseButton. Como parámetro por defecto debe tener el parámetro Button a Left, con lo que esta condición devolverá True cuando el botón izquierdo del ratón sea pulsado.
- Añadimos la acción FromMouseToWorld. Esta acción convierte la posición del Mouse de coordenadas de pantalla a coordenadas de mundo lanzando un rayo desde la cámara para encontrar el punto de colisión en el mundo. Tiene varios parámetros:
  - Camera: Crearemos una variable de la Blackboard con este mismo nombre.
  - Mask: Crearemos una variable de la Blackboard con este mismo nombre.
  - SelectedGameObject: No lo necesitaremos, lo dejamos vacío.
  - SelectedPosition: Creamos una variable de la blackboard con este mismo nombre; aquí se nos almacenará el resultado de la operación.
- Añadimos una acción MoveToPosition. En el parámetro de entrada Target seleccionaremos la variable de la Blackboard recién creada: SelectedPosition, con lo que moveremos a la entidad a la posición resultado de la operación de la acción FromMouseToWorld.
- En las propiedades del árbol cambiaremos el parámetro de entrada SelectedPosition de OUT a LOCAL, ya que será un parámetro de uso interno del árbol.
- Añadimos un nodo Secuencia para llamar primero a la condición de comprobación de clic del ratón, a continuación, la acción de obtener la posición mundo del ratón y por último la acción de movimiento a dicha posición.
- Para que esta Secuencia se pueda repetir la colocaremos bajo un nodo Repeat. Cabe destacar que si esta Secuencia recibiera FALSE en algún momento (en esta escena

podría ocurrir si se pulsa el ratón en el primer frame antes de que todo esté configurado, algo realmente raro y difícil de conseguir…), incluso el nodo Repeat devolvería Fallo y dejaría de ejecutarse. Para evitar esto podemos cambiar la política del nodo estableciendo: Continue When Child Fails



- Ya tenemos el árbol de control del jugador listo. Ahora se lo asignamos al GameObject Player.
    - Como hicimos con Enemy debemos añadir al Player un componente de navegación NavMeshAgent.
    - Asimismo le añadimos un componente Behavior Execution.

- Le añadimos el Behavior ClickAndGo.
- Añadimos al parámetro camera el GameObject MainCamera.
- Establecemos el valor de Mask a Everything.

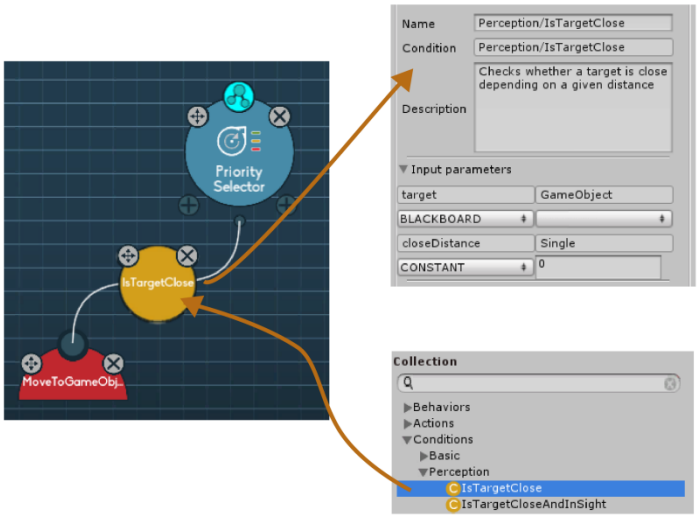
- ¡Ya podemos probar la escena y comprobar que podemos hacer moverse al jugador!
- Para dar ventaja al jugador en la siguiente parte vamos a subir el valor Speed a 7.0 en el NavMeshAgent component del player.

Parte 7: El enemigo persigue al player

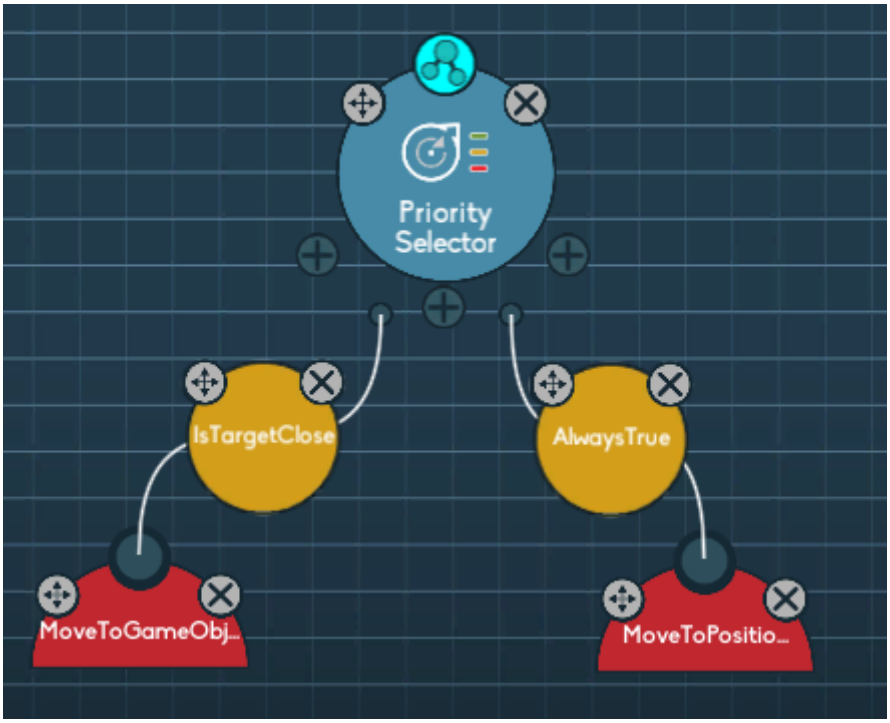
- Hasta ahora tenemos al enemigo moviéndose por posiciones aleatorias y al jugador siendo controlado con el ratón.

Vamos a añadir al enemigo la posibilidad de seguir al player cuando esté suficientemente cerca y que, si se aleja demasiado, vuelva a su posición inicial (casa)

- Lo que necesitamos entonces es ejecutar un comportamiento (perseguir) mientras se cumpla una condición (estar en distancia), para pasar a ejecutar otro cuando deje de cumplirse (volver a casa), pero que si se vuelve a cumplir la condición de cercanía la persecución se vuelva a ejecutar.
- Para conseguir esto utilizaremos un Selector de Prioridad. La prioridad va implícita en el orden de los hijos. El primer hijo será el que tenga una condición que debe comprobarse continuamente para ejecutarse o pasar al siguiente hijo; mientras, continúa ejecutando su condición para poder abortar a los siguientes hijos.
- Vamos a crear un nuevo BehaviorTree llamado EnemyBehavior.
    - Creamos un Priority Selector.
    - Añadimos la acción MoveToGameObject. Añadimos al parámetro Target una nueva variable de la Blackboard llamada Player.
    - Unimos el PrioritySelector con la acción recién creada. Se nos creará automáticamente el nodo condición que será comprobado para ejecutar dicha acción. Por defecto esta condición tendrá el valor AlwaysTrue.
    - Buscamos en la colección la condición IsTargetClose y la arrastramos y soltamos directamente en nuestra condición del PrioritySelector (¡no en la zona gráfica vacía, directamente en la condición!)



- La condición IsTargetClose tiene dos parámetros:
  - En el parámetro Target seleccionamos la variable de la Blackboard Player.
  - En CloseDistance lo establecemos a 15.
- Unimos la acción MoveToPosition como segundo hijo del Priority Selector. El parámetro target de dicha acción lo establecemos como (-20, 0.5, 0)



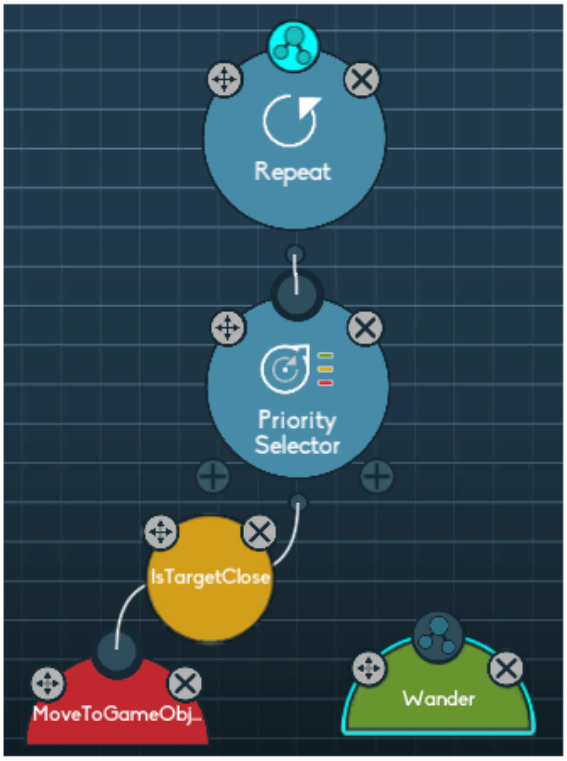
- Editamos el componente Behavior Executor de Enemy para que ahora utilice este nuevo árbol EnemyBehavior.

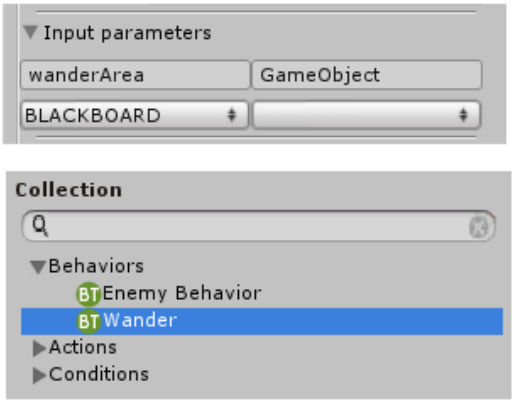
- Establecemos el parámetro Player arrastrando el GameObject Player.

- ¡Probamos! Al estar cerca del enemigo éste nos perseguirá y una vez que nos alejemos de él volverá a su posición casa. ¡Pero sólo lo hará una vez! Debemos hacer, una vez más, que el PrioritySelector sea hijo de un nodo Repeat!

Parte 8: Rehusar comportamientos (Subárboles)

- Como ya hemos visto los subárboles son una pieza clave en la creación de comportamientos cuando se usan BehaviorTrees.
- Editamos EnemyBehavior
- Eliminamos el nodo MoveToPosition.
- Buscamos en la colección el BehaviorTree que creamos antes, Wander, y lo añadimos a EnemyBehavior.

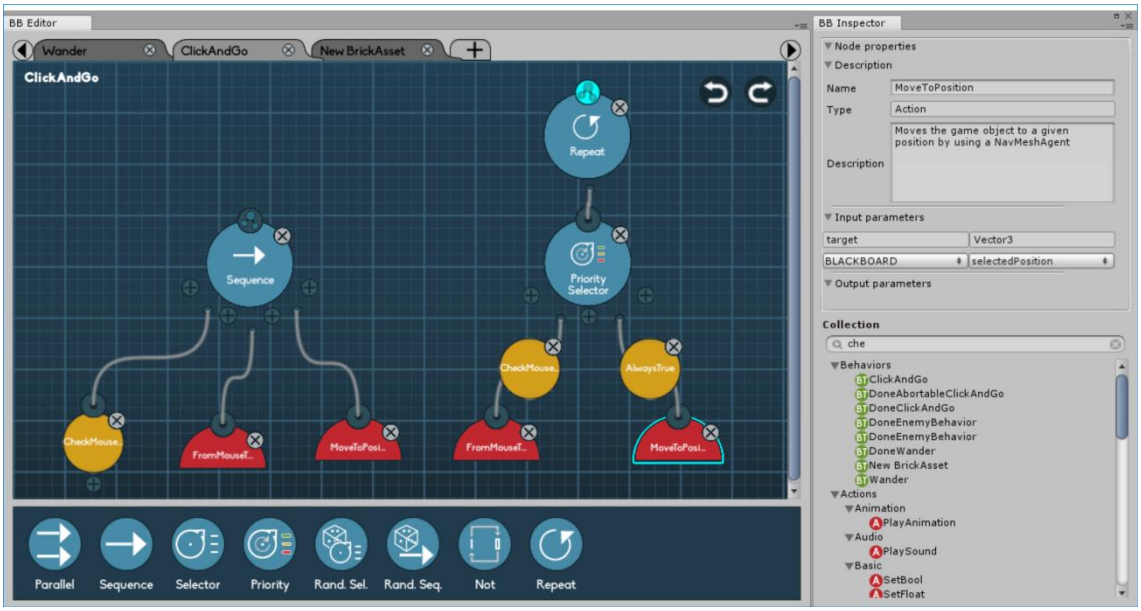




- En el subárbol debemos configurar que el parámetro WanderArea será la variable de la BlackBoard WanderArea. Será configurada en el Enemy en su Behavior Executor component, así como lo está la variable Player. Para WanderArea seleccionamos, igual que antes, el GameObject Floor.
- ¡Probamos la escena! El enemigo hará su patrulla aleatoria hasta que nos “vea” y entones nos perseguirá, volviendo a hacer su patrulla aleatoria tras alejarnos lo suficiente.

Parte 9: Mejora del control del jugador

- Como habréis notado, cuando controlamos al jugador, éste no vuelve a hacer caso a nuestros clics hasta haber llegado al punto de destino del clic anterior. Ahora que habéis aprendido a usar PrioritySelectors... ¿Se os ocurre cómo podríamos solucionar este problema?
- Intentad resolverlo... en la página siguiente os dejo cómo quedaría el árbol con ello solucionado.



Como veis ahora sí responde a nuestros clics correctamente.

Hasta aquí este ejercicio de BehaviorTrees en Unity. ¡Os animo a seguir investigando con las funcionalidades ya implementadas, o incluso a añadir nuevas acciones y condiciones que podáis necesitar en vuestros proyectos!