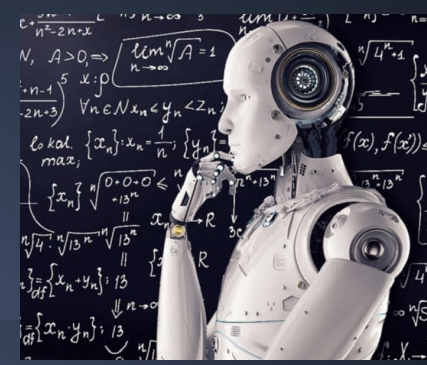




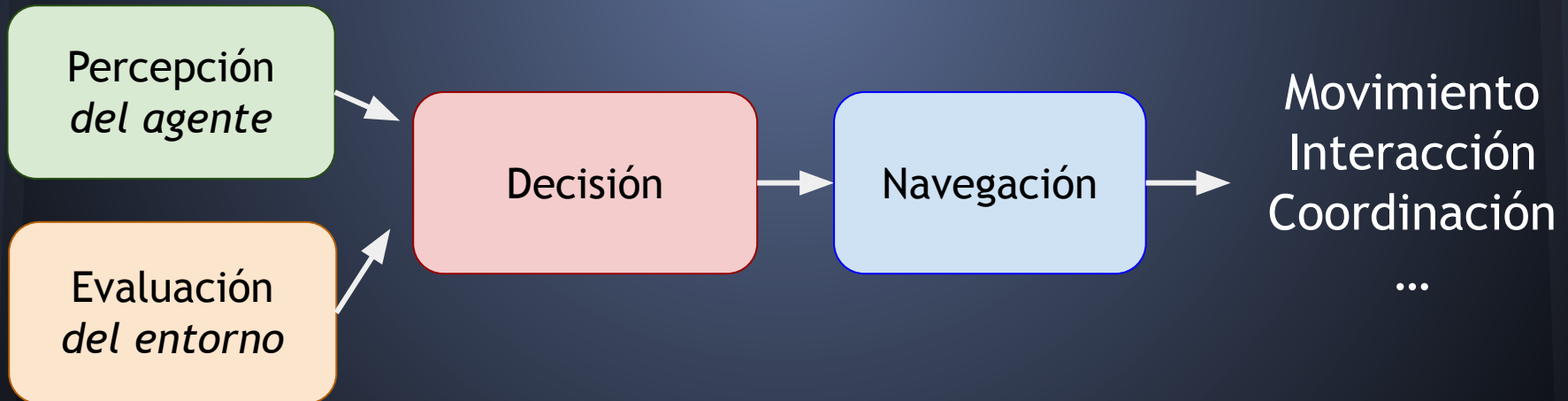
Desarrollo de Videojuegos

Herramientas para crear
jugabilidad avanzada
Inteligencia artificial

Motivación

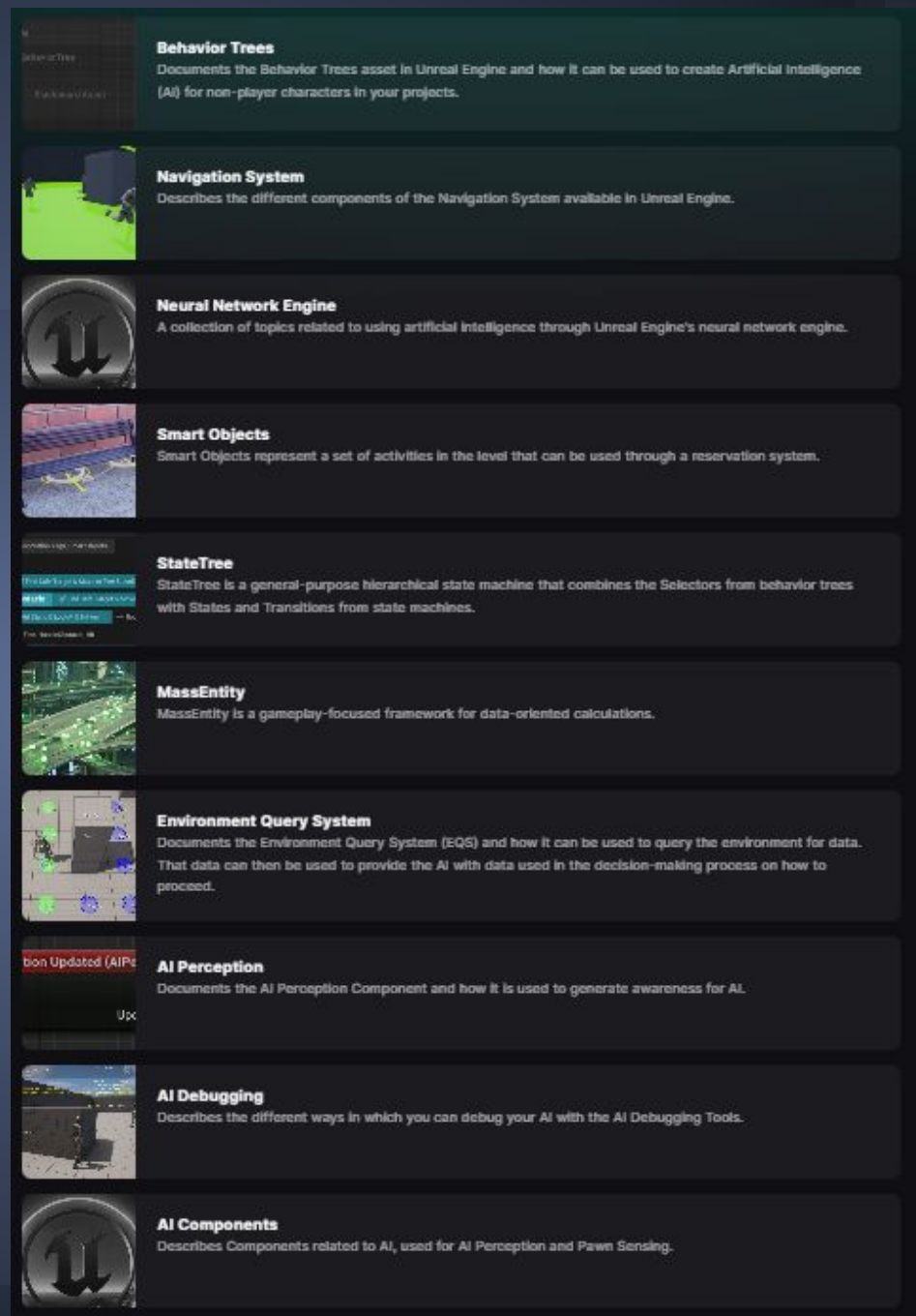


- La Inteligencia Artificial (IA) en juegos no es algo “*académico*” sino un conjunto de **soluciones de programación a problemas muy concretos**
 - Ej. Comportamiento de un agente enemigo

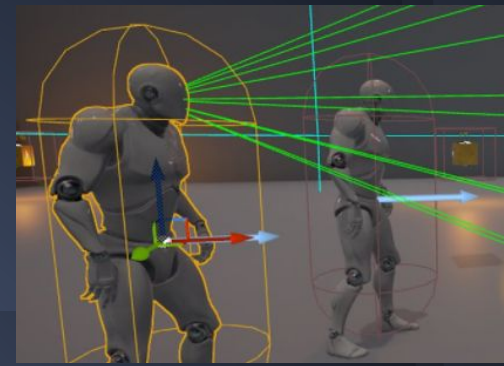


Motivación

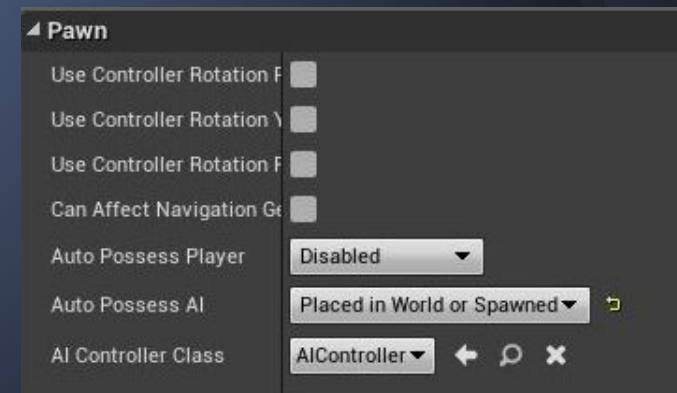
- En Unreal Engine, existen *diversos sistemas* que nos *ayudan a realizar muchas tareas*



Requisitos

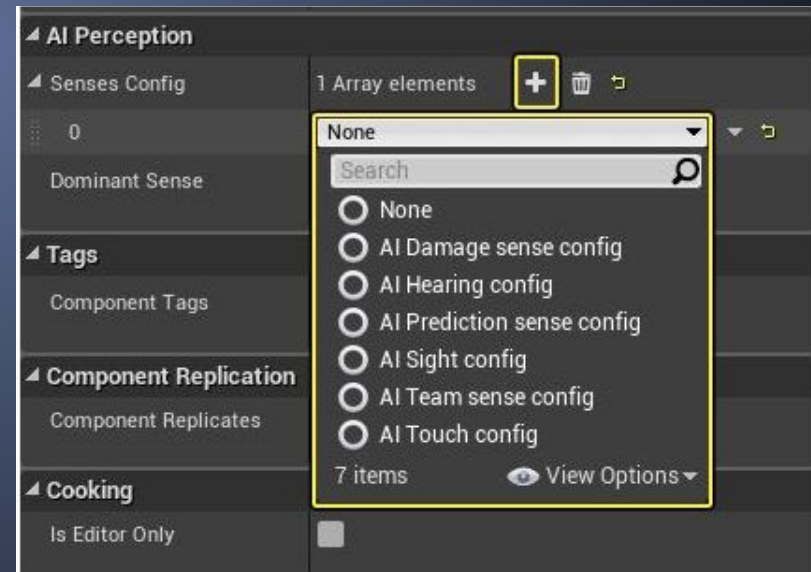


- Todo agente inteligente necesita un **peón** y un **controlador de IA** que lo “*posea*”
 - Ambos actores deben existir en el mundo, así que:
 - o usamos **Spawn Actor** (concretamente **Spawn AI from Class**), o permitimos que el peón *crea* y *sea poseído* por un controlador automáticamente
 - Si vamos a tener enemigos ya puestos en el mundo y otros generados en tiempo real, marcamos:
Placed in World or Spawned



Percepción del agente

- Los agentes pueden tener **sentidos** y percibir **daño**, **ruido**, **presencia** de alguien (por **vista**, **tacto**...), **anticiparse al movimiento de otro**...



<https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/AI Perception/index.html>

Percepción del agente

- Ej. Percepción de daño

- Unreal Engine tiene un **sistema de daño** para actores (cualquiera puede hacerlo con **Apply Damage** y recibirlo con el evento **Any Damage**)
- Además de la cantidad y el tipo de daño se puede indicar el **instigador del evento** (*controlador del jugador o bot que quiere dañar*) y el **causante del daño** (*proyectil, arma de mano o algo similar*)

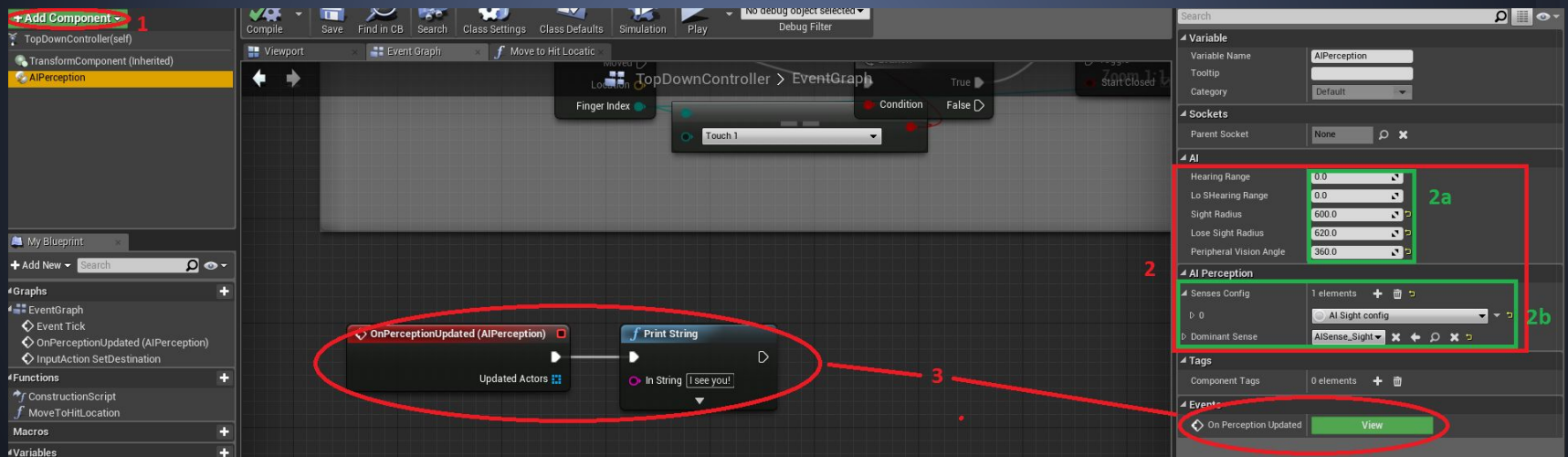
EVENT
INSTIGATOR

DAMAGE
CAUSER



Percepción del agente

- Añade lo siguiente al controlador del NPC
 - El componente **AI Perception**, los *sensidos* que necesites en **SensesConfig** y eventos del tipo de **On Perception Update**



- Todos los actores “**percibibles**” se deben *autoregistrar* en el *sentido* correspondiente

Percepción del agente

- Un actor es “percibible” llamando a `RegisterPerceptionStimuliSource` o mediante el componente `AIPerceptionStimuliSource` con `Auto Register as Source` activado
 - Ej. Añadir `AI_Sense_Sight` en `Register as Source for Senses` si lo que quieres es que sea percibible a través de la vista
- Internamente **Unreal Engine distingue entre amigos, enemigos o neutrales**; como todo es neutral por defecto, conviene marcar `Detect Neutrals` en el componente `AIPerception` de los NPCs
- Muy importante comprobar que la colisión de rayos está *bien configurada*, que el NPC no está *bloqueando su propia vista* y que no estamos trazando rayos entre *pivotes que están demasiado cerca del suelo*

Evaluación del entorno

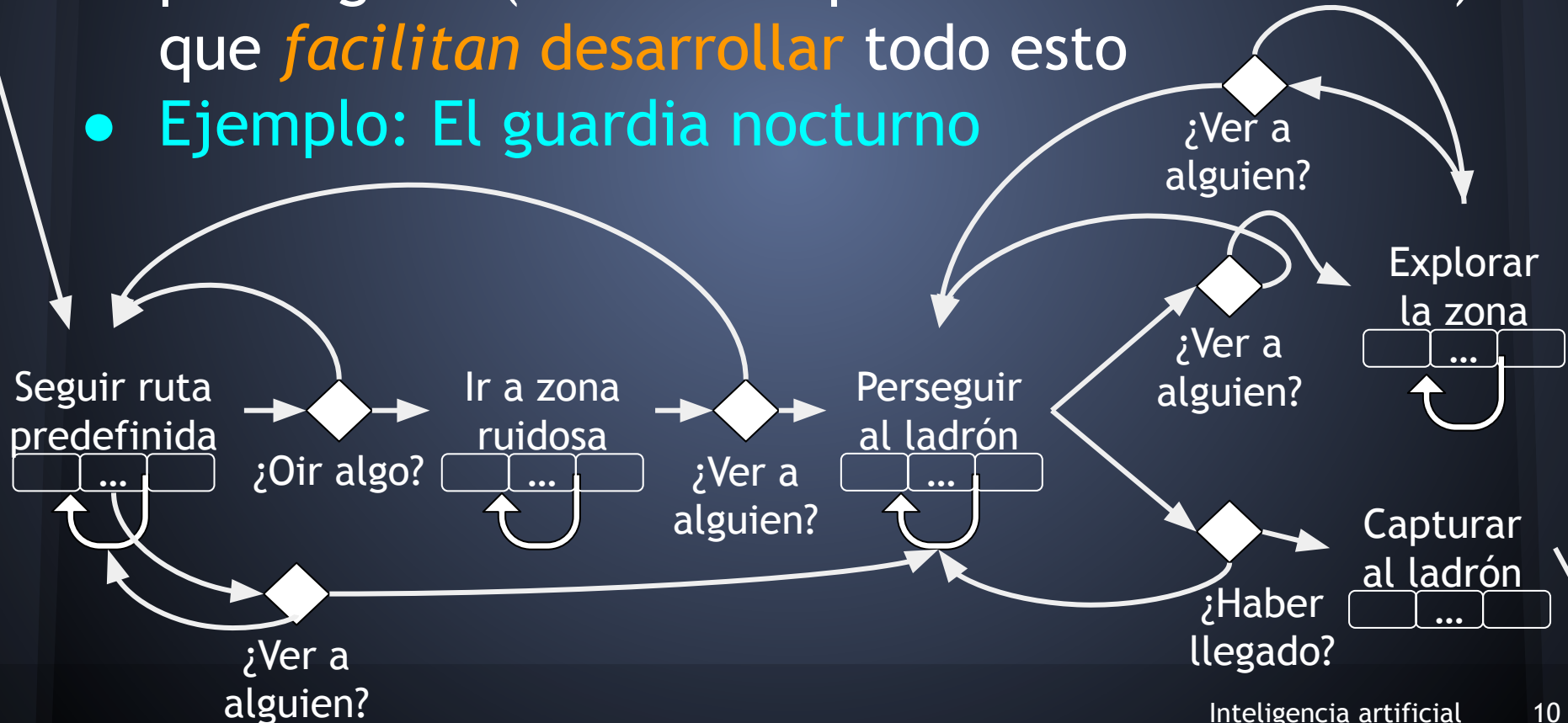
- El sistema EQS (**Environment Query System**) es una **interfaz con el entorno** que sirve para **consultar si se dan ciertas condiciones**
 - ¿Quien domina a nivel táctico el campo de batalla?
 - ¿Dónde hay una buena *cobertura* para cierta posición?
 - ...



<https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSQuickStart/index.html>

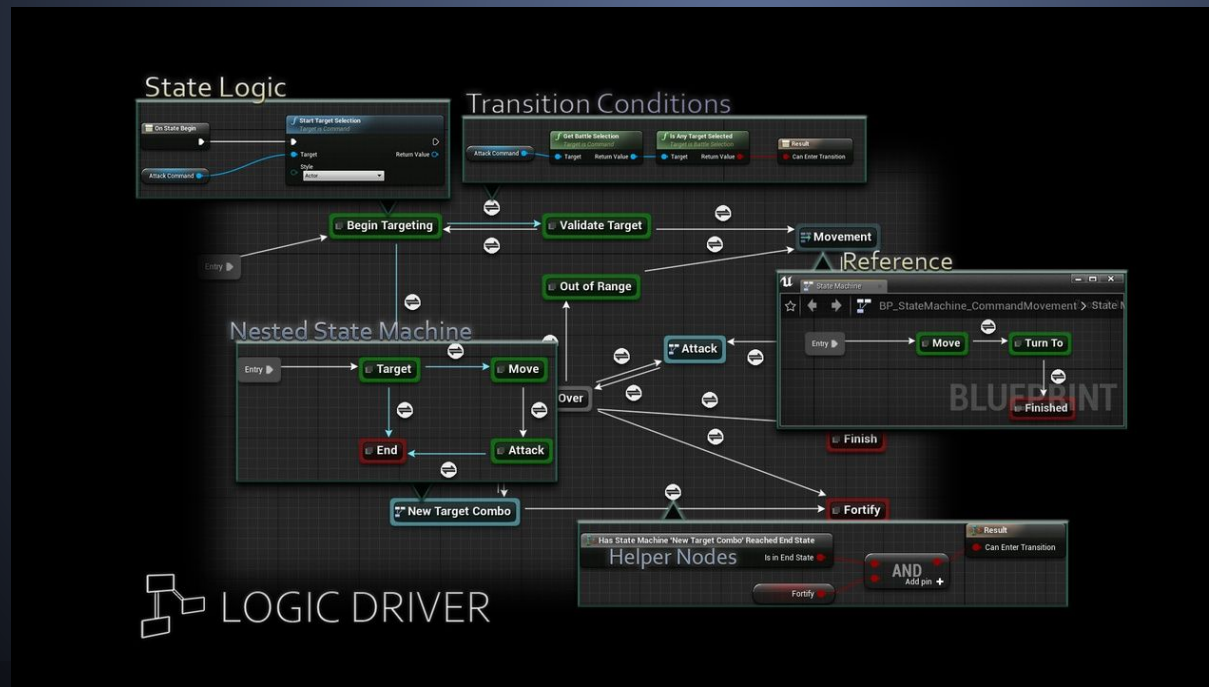
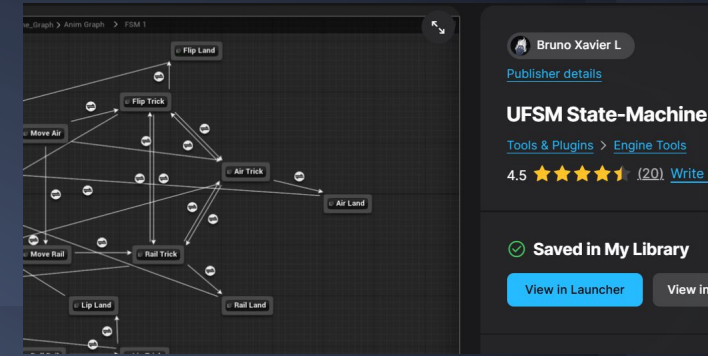
Decisión

- Se puede programar *normalmente*, aunque hay paradigmas (con sus respectivas herramientas) que *facilitan desarrollar* todo esto
 - Ejemplo: El guardia nocturno
- 
- ```
graph LR; A(()) --> B{¿Ver a}; B --> A;
```

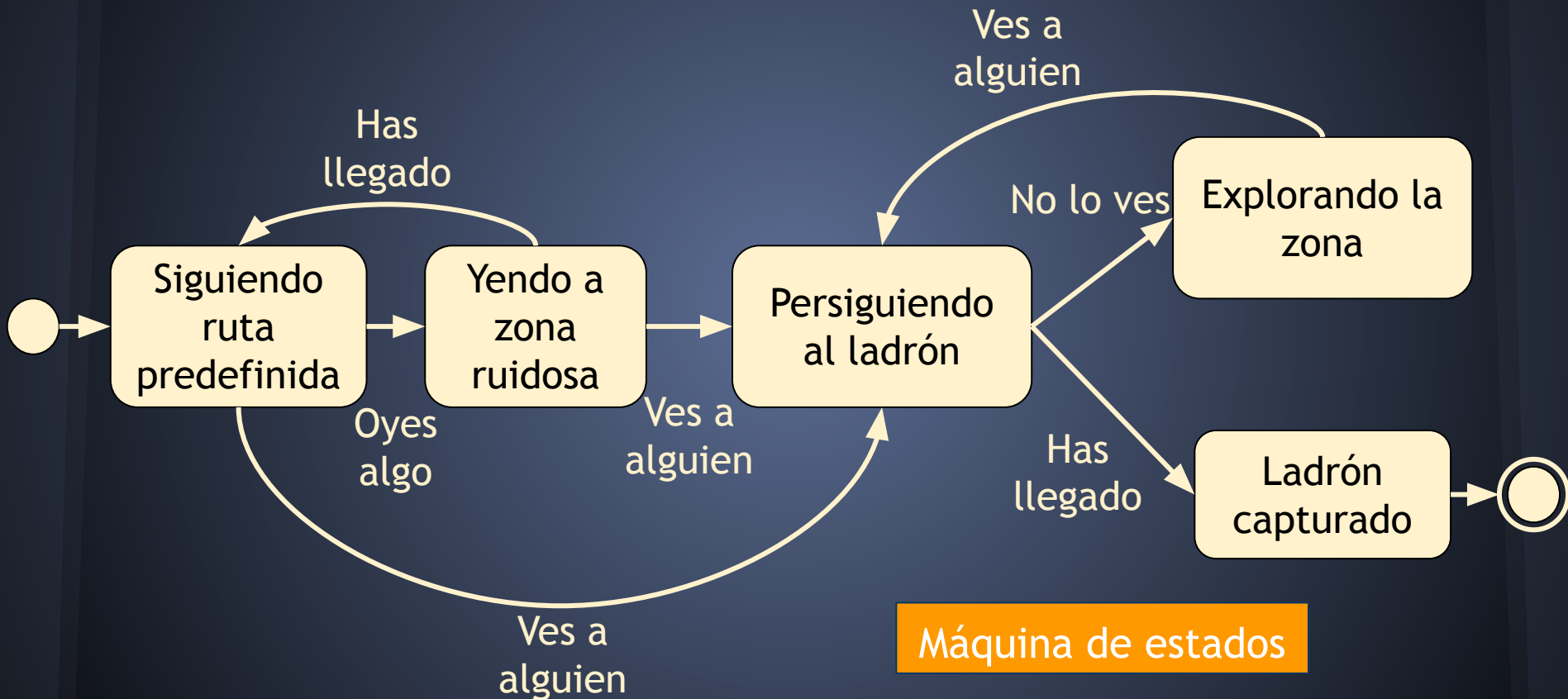


# Máquina de estados

- La **programas** tú (ej. con **Gameplay Tags**) o reutilizas la de controlar animaciones
- Usas alguna **herramienta disponible** en **Fab**

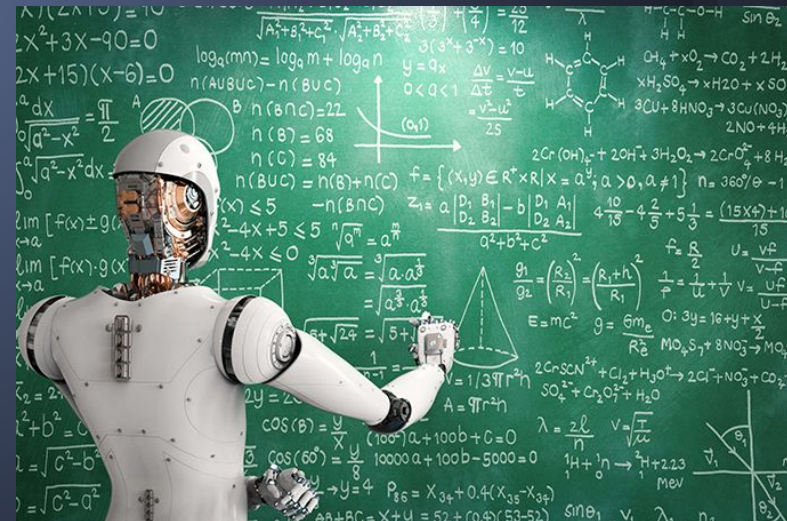
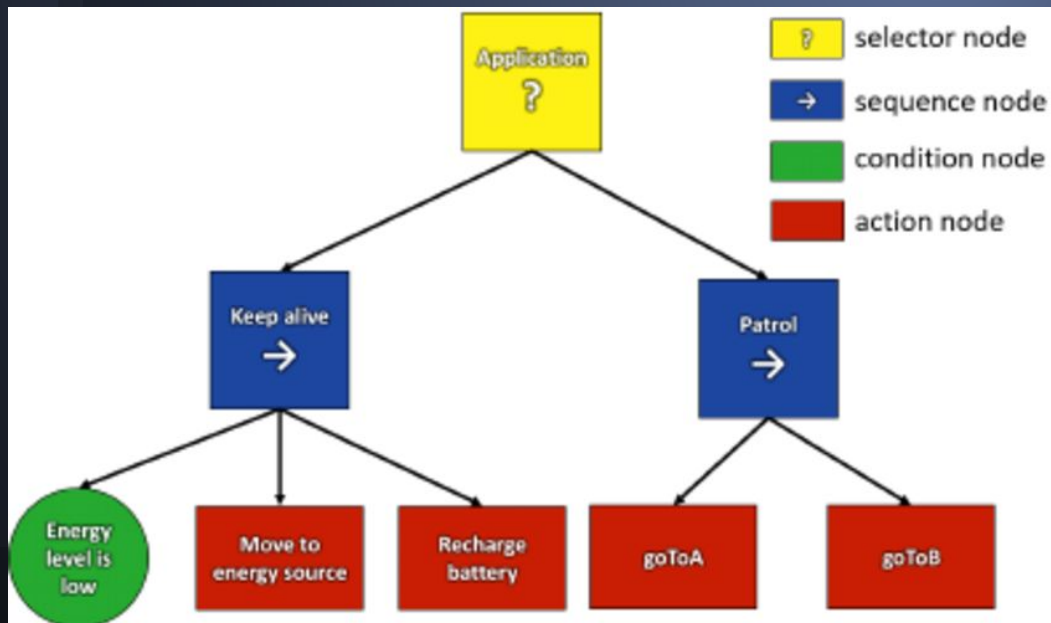


# Ejemplo



# Árbol de comportamiento

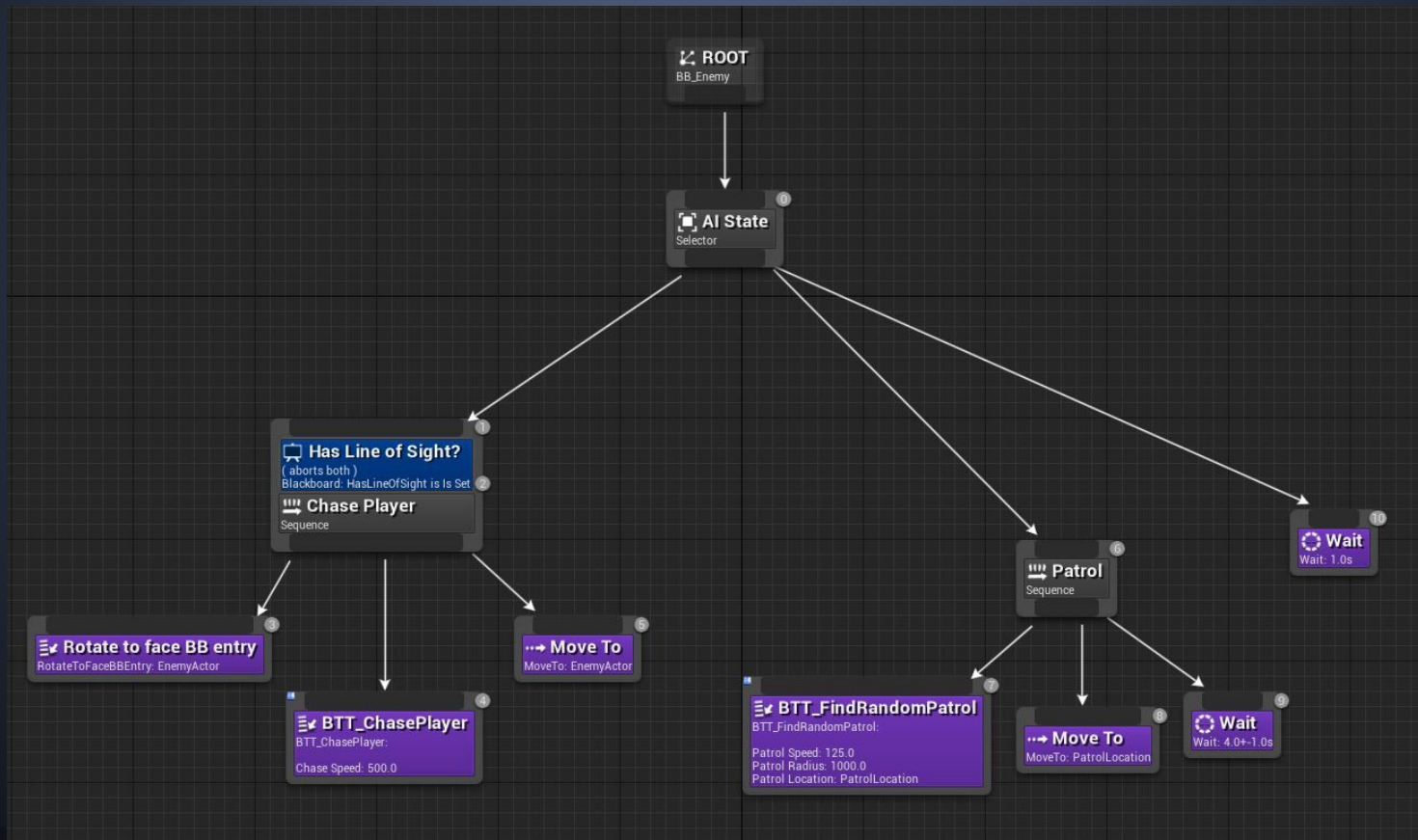
- Para decidir, el paradigma por defecto que ofrece Unreal son los **árboles de comportamiento** (para *razonar*) junto a las **pizarras** (para *compartir conocimiento*)



# Árbol de comportamiento

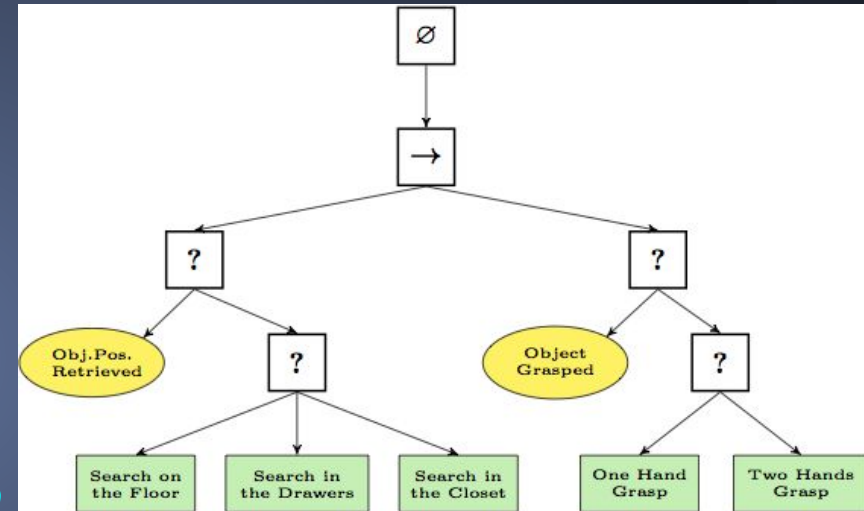
## BEHAVIOR TREE

- Jerarquizan tareas, con este aspecto:



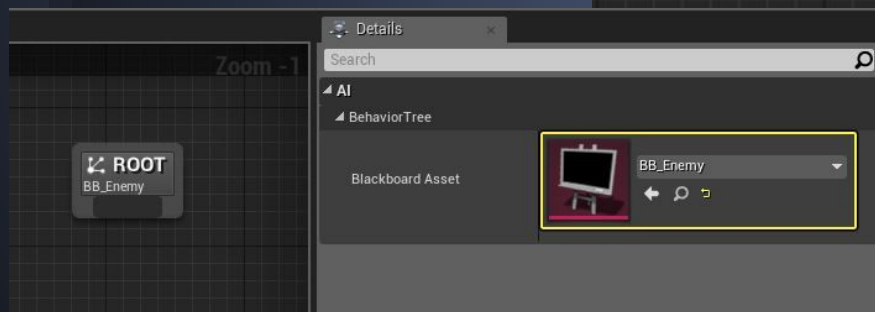
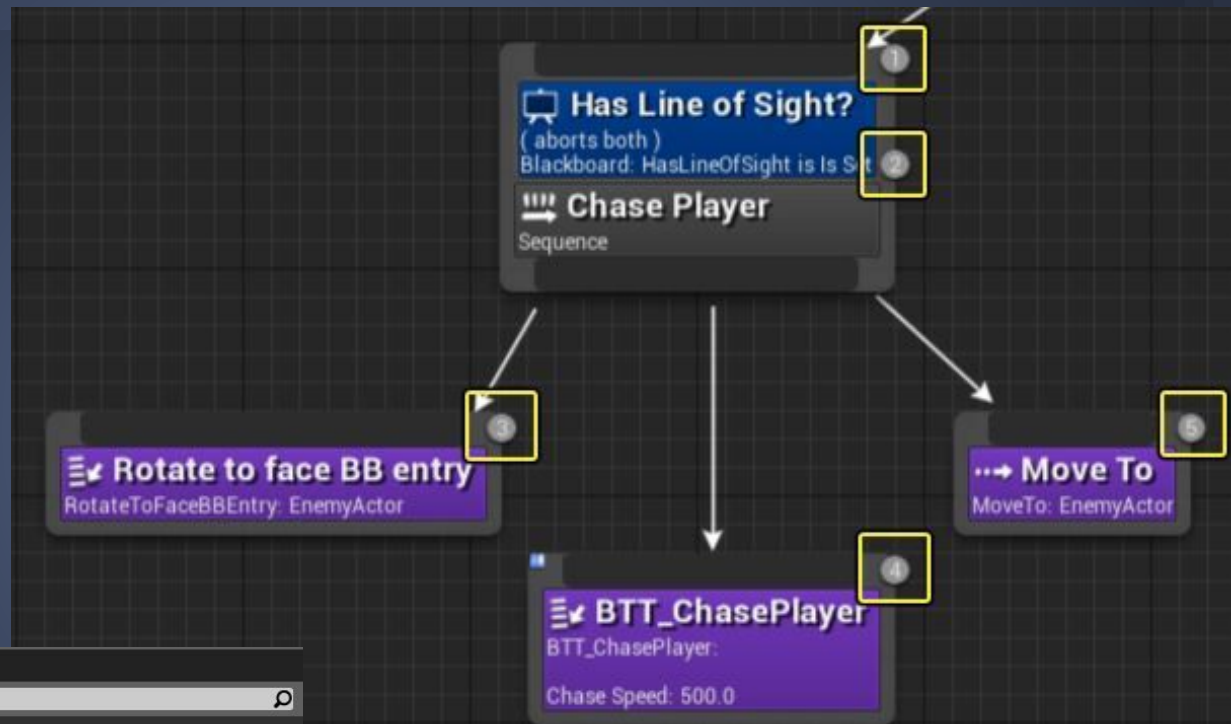
# Árbol de comportamiento

- Son **árboles dirigidos**
  - Tienen un **nodo raíz**, **nodos de control de flujo** y **nodos de ejecución** (las **tareas**)
    - \* Que pueden estar *en ejecución* o *terminar con éxito o con fracaso*



- Dos nodos principales de control de flujo:
  - **Nodo Selector** ( $?$ ), tiene éxito si uno de sus hijos tiene éxito, probando de izquierda a derecha
  - **Nodo Secuencia** ( $\rightarrow$ ), tiene éxito si todos sus hijos tienen éxito, de izquierda a derecha

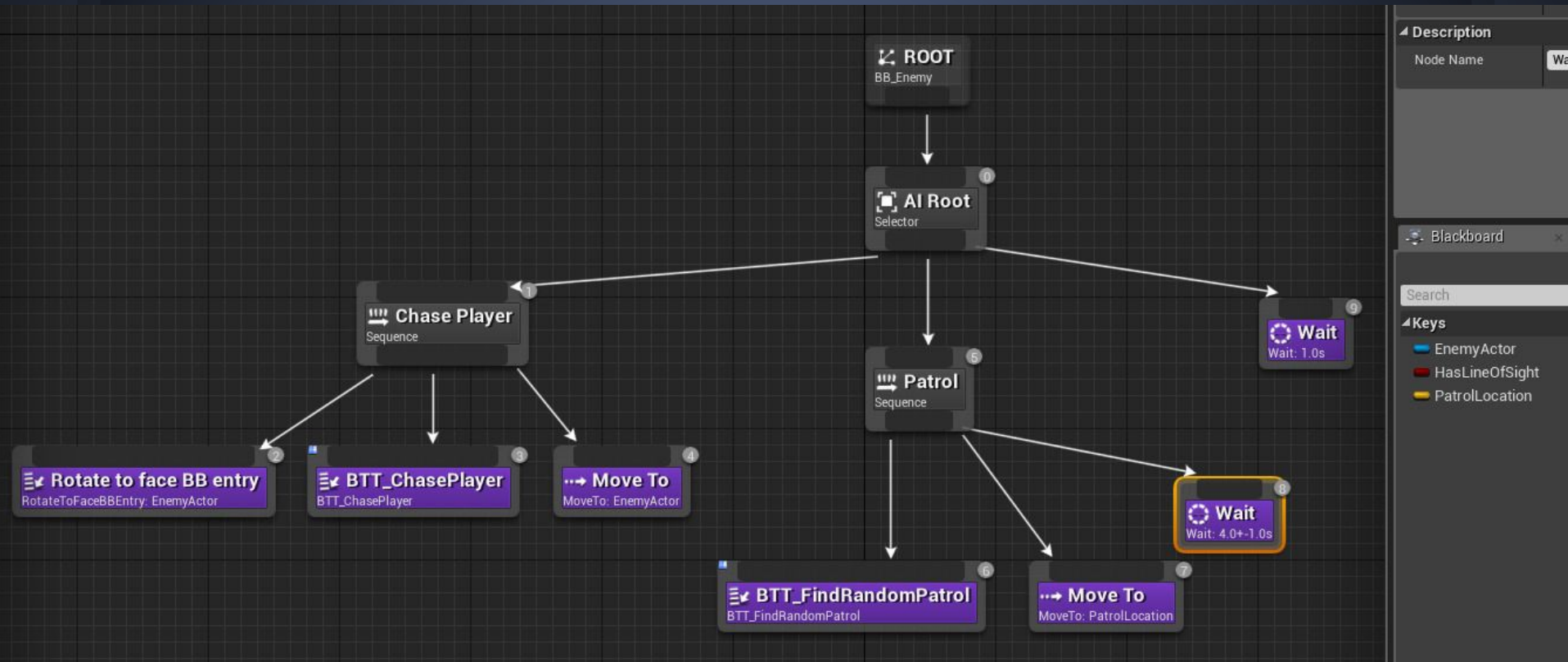
# Árbol de comportamiento



La pizarra tiene **registros clave-valor** que se usarán en el BT (son las variables locales)

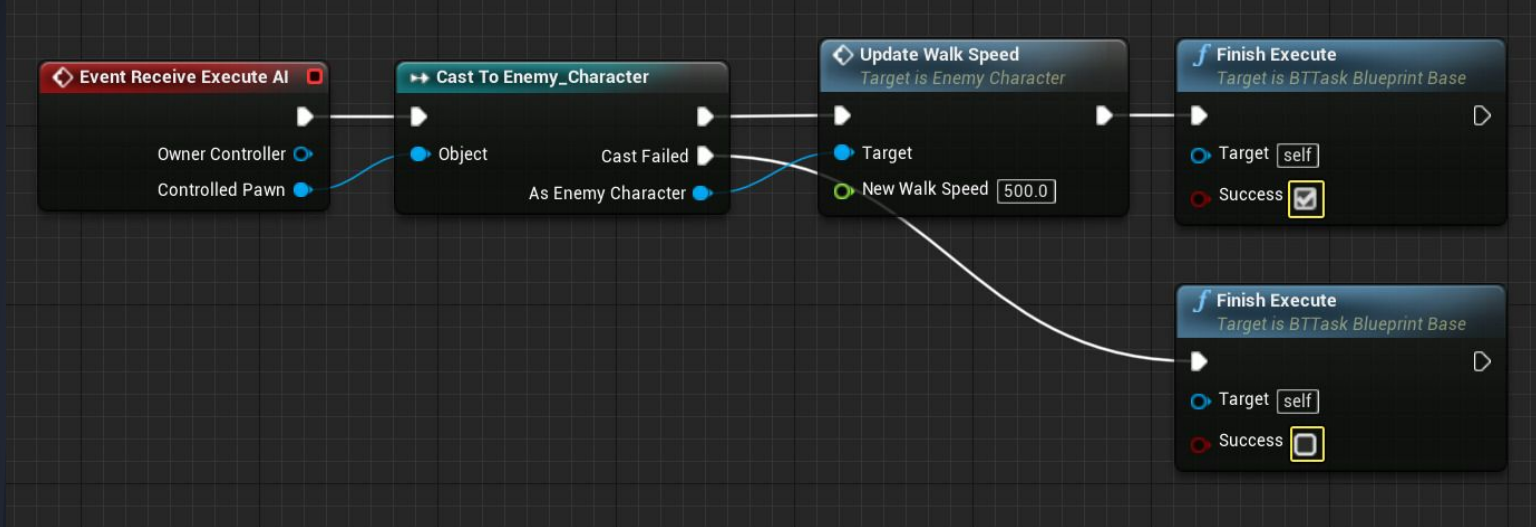
Nodos de **secuencia** o **selección**, alternan el orden de ejecución

# Ejemplo



# Árbol de comportamiento

Las tareas se programan en respuesta a eventos del BT y *pueden usar de todo*



Como dijimos, tendremos un *AIController* y este lanza el BT

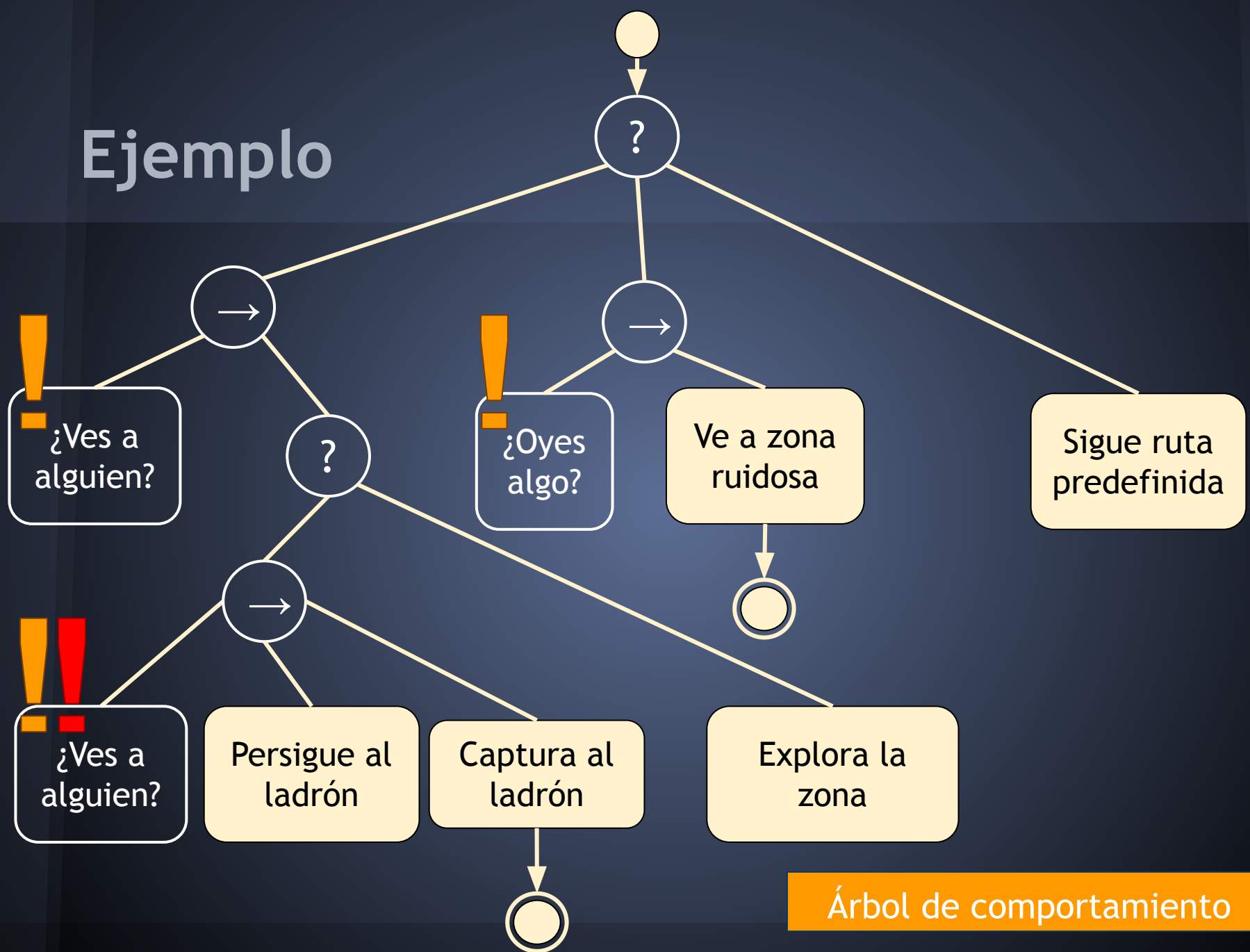
**Pawn**

|                                  |                                     |
|----------------------------------|-------------------------------------|
| Use Controller Rotation Pitch    | <input type="checkbox"/>            |
| Use Controller Rotation Yaw      | <input checked="" type="checkbox"/> |
| Use Controller Rotation Roll     | <input type="checkbox"/>            |
| Can Affect Navigation Generation | <input type="checkbox"/>            |
| Auto Possess Player              | Disabled                            |
| Auto Possess AI                  | Placed in World                     |
| AI Controller Class              | BP_EnemyController                  |

```
graph LR; A[Event On Possess] --> B[Run Behavior Tree];
```

<https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html>

# Ejemplo



Árbol de comportamiento

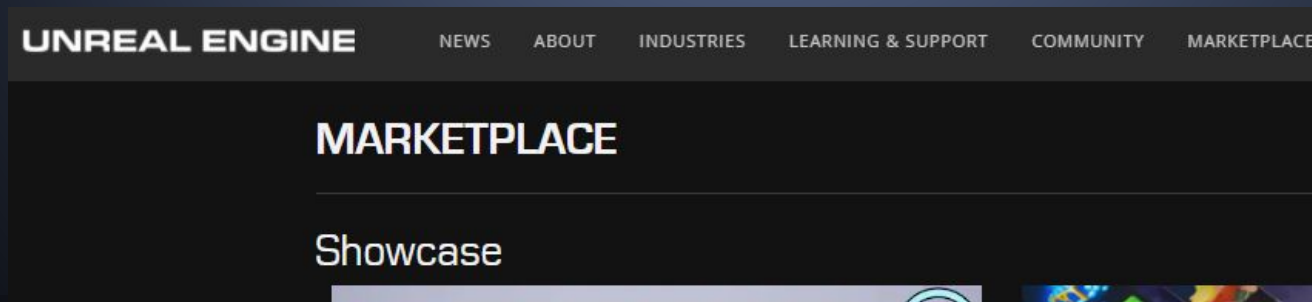
# Participación

- ¿Cuál es la diferencia entre un **nodo selector** y un **nodo secuencia**?
  - A. El primero es una “Y” lógica y el segundo una “O”
  - B. El primero evalúa a derechas y el segundo al revés
  - C. El primero es nodo de flujo y el segundo ejecución
  - D. El primero es una “O” lógica y el segundo una “Y”
  - E. El primero evalúa a izquierdas y el segundo al revés



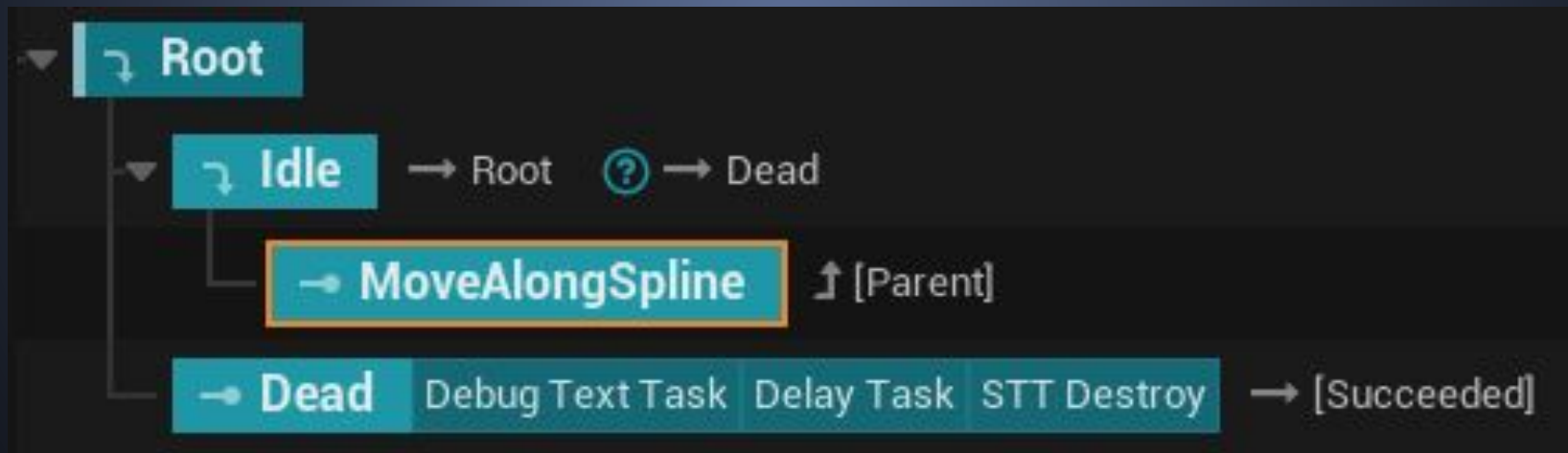
# Otros paradigmas

- Existen otras formas de *decidir*:
  - Árbol de estado
  - Teoría de la decisión/utilidad
  - Sistema de reglas
  - Planificación automática
  - ...
- Implementarlas es difícil y se suele recurrir a **herramientas de terceros**



# Ejemplo: Árbol de estado

- Activando los *plugins* **StateTree** y **GameplayStateTree**, podemos combinar estados y transiciones de FSMs con la estructura de árbol y los selectores de los BTs



<https://dev.epicgames.com/documentation/en-us/unreal-engine/statetree-quick-start-guide>

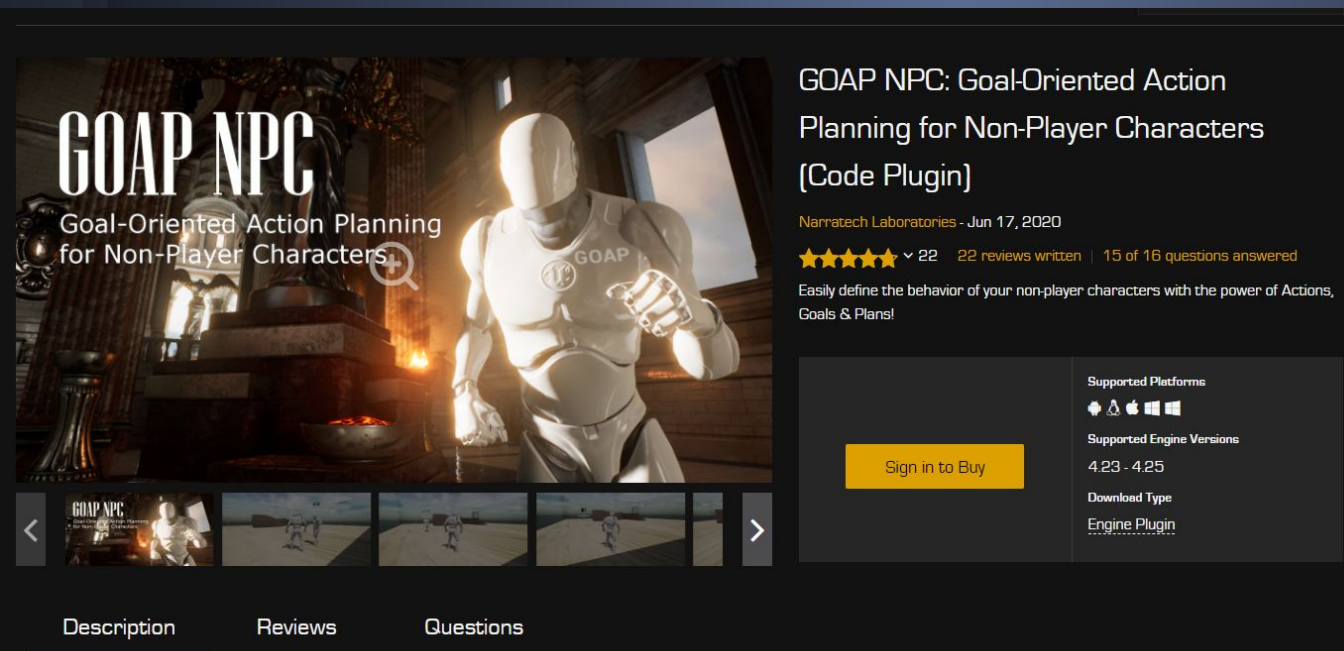
# Ejemplo



Árbol de estado

# Ejemplo: Planificación de Acciones Orientada a Objetivos (GOAP)

- Se basa en dar **objetivos** a los personajes, que construirán sus propios **planes** según las **acciones** que pueden realizar
  - Falta resubir **GOAP NPC** a **Fab...**



**GOAP NPC**  
Goal-Oriented Action Planning for Non-Player Characters

GOAP NPC: Goal-Oriented Action Planning for Non-Player Characters [Code Plugin]

Nanratech Laboratories - Jun 17, 2020

★★★★★ ~ 22 22 reviews written | 15 of 16 questions answered

Easily define the behavior of your non-player characters with the power of Actions, Goals & Plans!

Supported Platforms  
Windows, macOS, Linux

Supported Engine Versions  
4.23 - 4.25

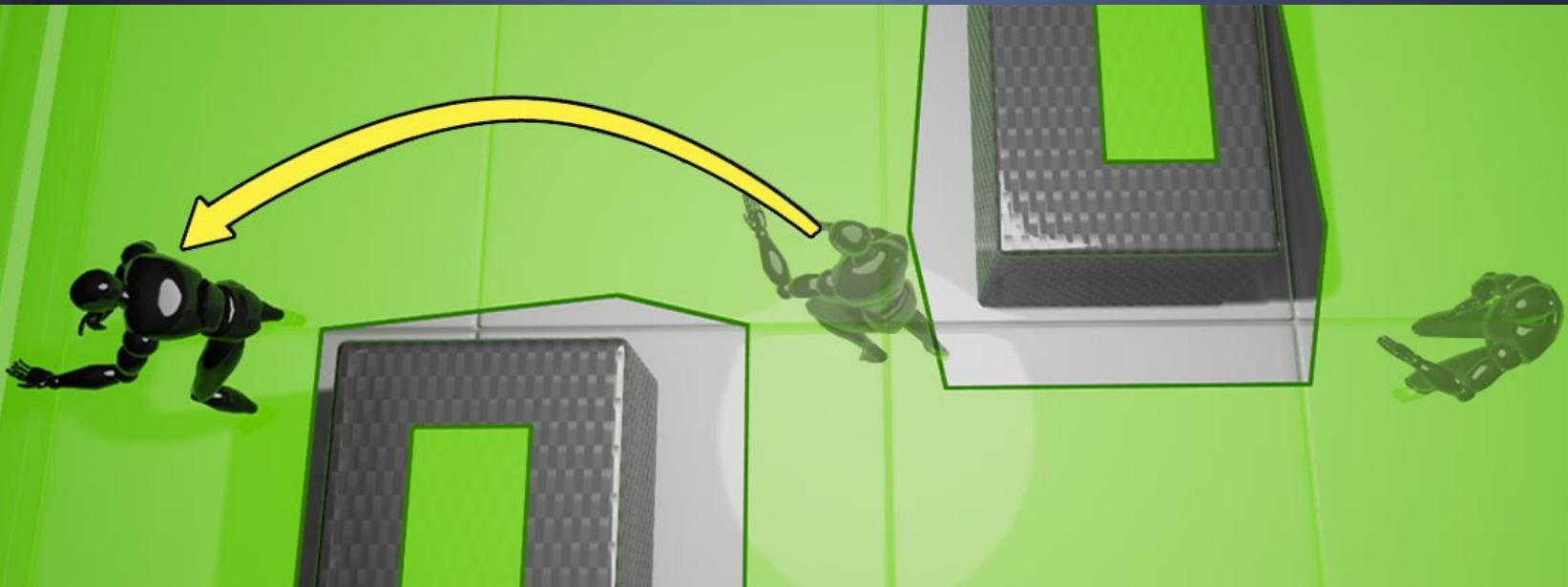
Download Type  
Engine Plugin

Sign in to Buy

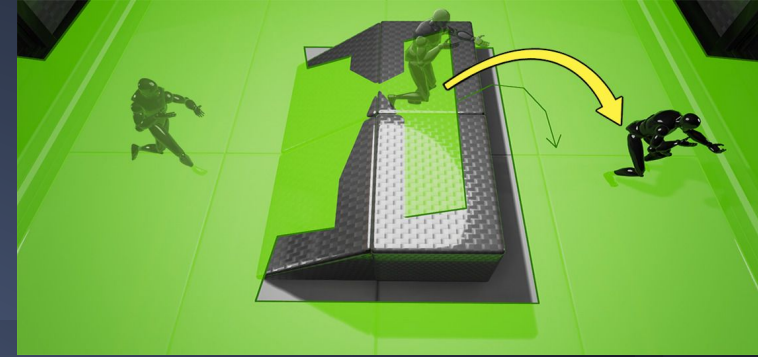
Description Reviews Questions

# Navegación

- Tener **mallas de navegación** en los niveles y **agentes capaces de navegarlas** es esencial
  - *Parte* de la inteligencia está en el mundo, y *parte* en la navegación por ruta óptima del propio agente



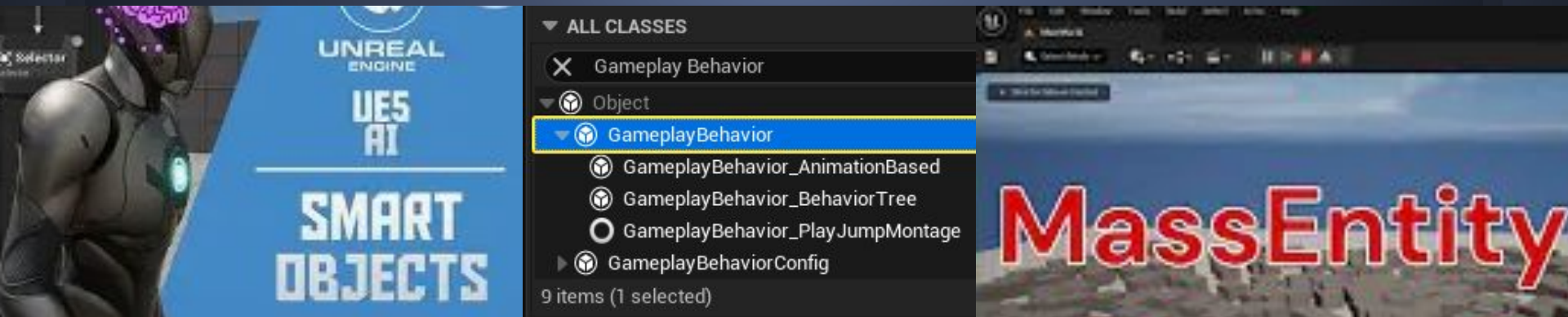
# Navegación



- El espacio **potencialmente navegable** se cubrirá con un **NavMeshBoundsVolume**
  - Pulsando **P** vemos la **mall**a de navegación *autogenerada* en base al suelo y los obstáculos
  - ¡Los controladores de IA ya *saben navegar* aquí!
- Colocando actores **NavLinkProxy** es posible **saltar o dejarse caer** de un punto a otro
- El componente **Nav Modifier** puede cambiar el **coste** u otras propiedades de una zona
  - Lo normal es que las IAs busquen *caminos mínimos en términos de coste*

# Movimiento, Interacción, Coordinación...

- Las **acciones** de los agentes inteligentes se programan como es habitual
  - Aunque también existen **herramientas de Unreal Engine** para facilitar todo esto...
  - ... incluso para hacerlo *a nivel masivo*



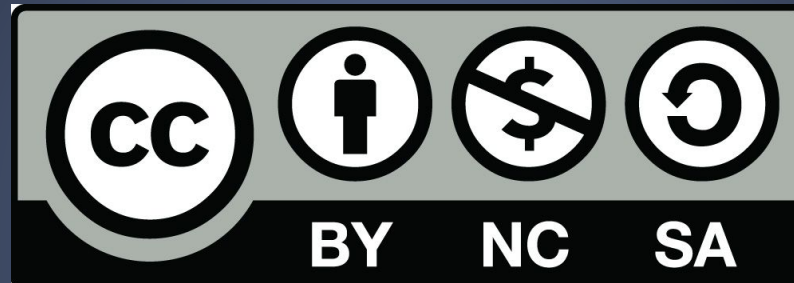
# Herramientas de depuración

- Ej. Si usamos *BTs* o percepción, se activan con **apóstrofe ('/?)**... y luego **control numérico**
  - ¡Son **extremadamente útiles** para depurar!



<https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/AIDebugging/index.html>

# Críticas, dudas, sugerencias...



*\* Licencia sólo aplicable al texto original de estas diapositivas*

Federico Peinado (2019-2026)

[www.federicopeinado.es](http://www.federicopeinado.es)