DEVELOPING AI IN GAMES

Understanding Artificial Intelligence and Artificial Stupidity

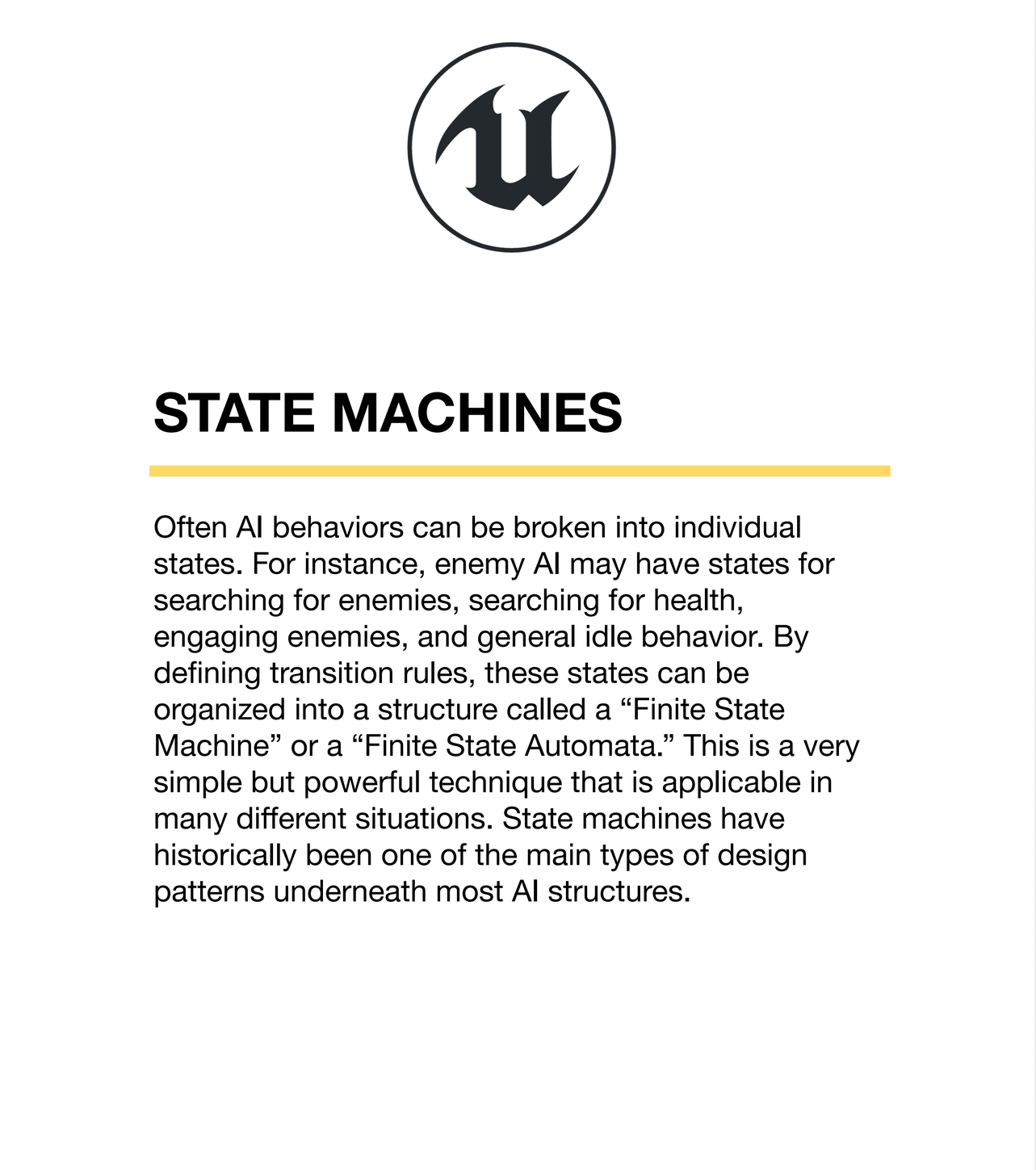
AI OVERVIEW

I think, therefore I am. —Not a robot

WHAT IS AI?

In games, AI typically refers to the implementation of seemingly intelligent behavior in various game agents. AI types can have significant differences in the following:

- How they process tasks
- How they receive and “perceive” information about the world
- How they understand the spatial rules and layout of the world
- How they are adapted to more closely approximate human behavior
- How they react to outside stimuli



STATE MACHINES

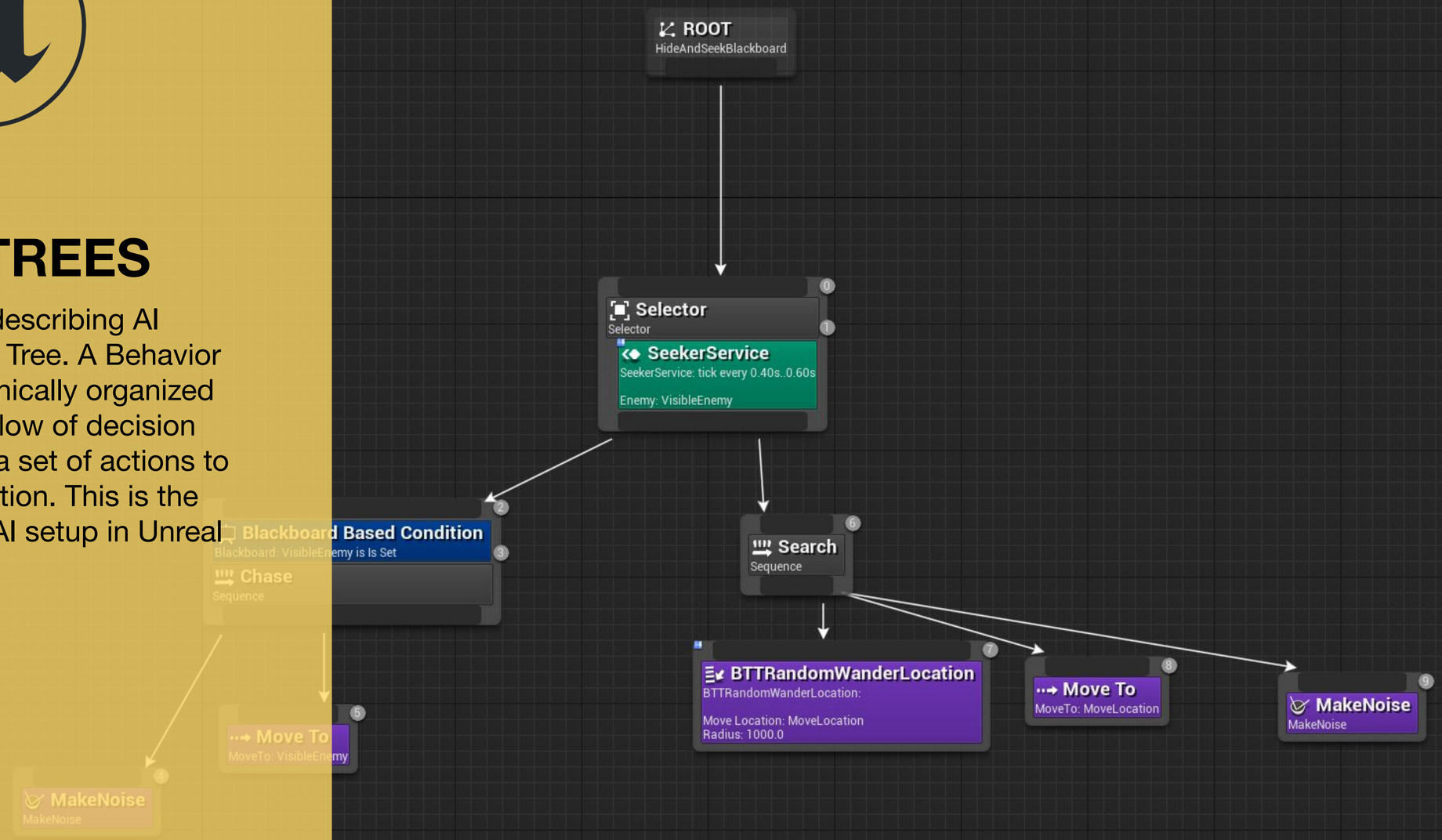
Often AI behaviors can be broken into individual states. For instance, enemy AI may have states for searching for enemies, searching for health, engaging enemies, and general idle behavior. By defining transition rules, these states can be organized into a structure called a “Finite State Machine” or a “Finite State Automata.” This is a very simple but powerful technique that is applicable in many different situations. State machines have historically been one of the main types of design patterns underneath most AI structures.

AI SYSTEMS IN UE4

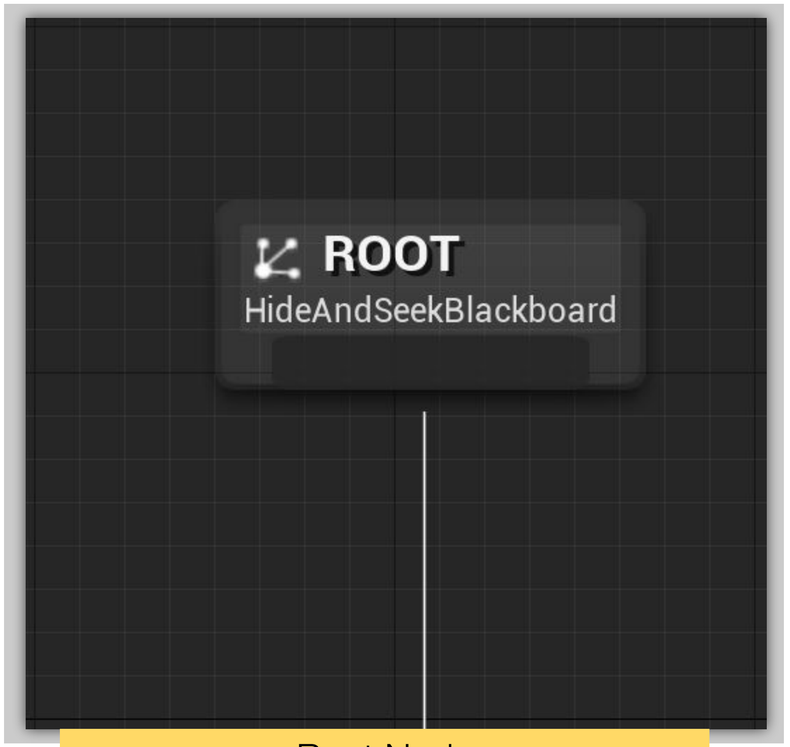
Behavior Trees and EQS

BEHAVIOR TREES

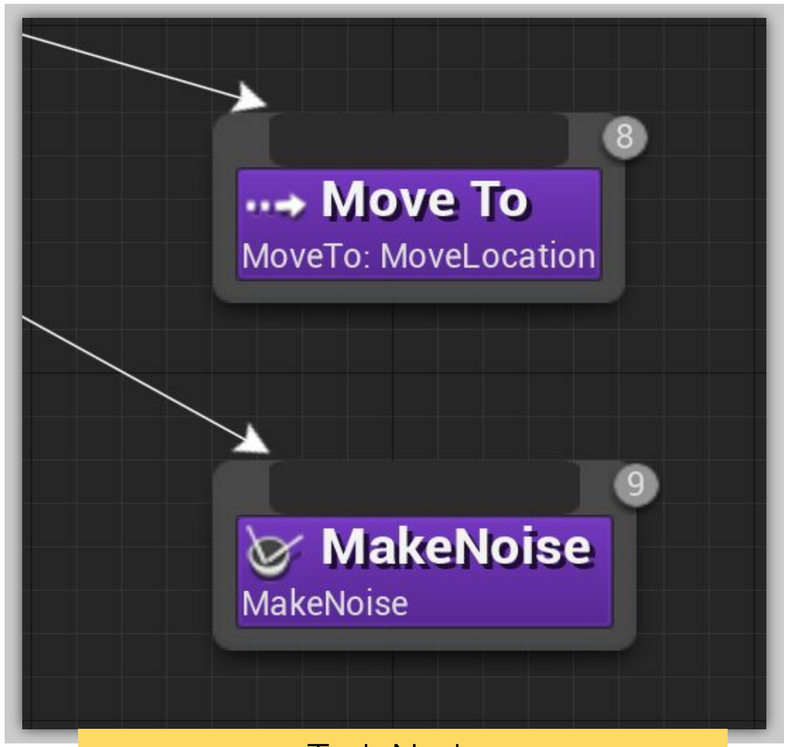
A useful technique for describing AI behaviors is a Behavior Tree. A Behavior Tree consists of hierarchically organized nodes that control the flow of decision making and determine a set of actions to execute in a given situation. This is the most common type of AI setup in Unreal Engine 4.



BEHAVIOR TREE



Root Node



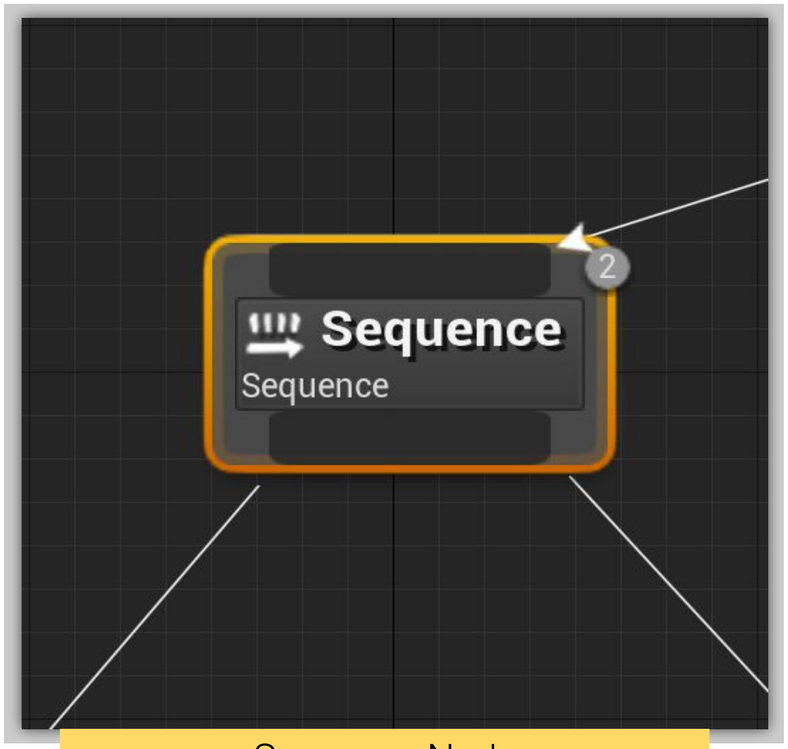
Task Nodes

Root

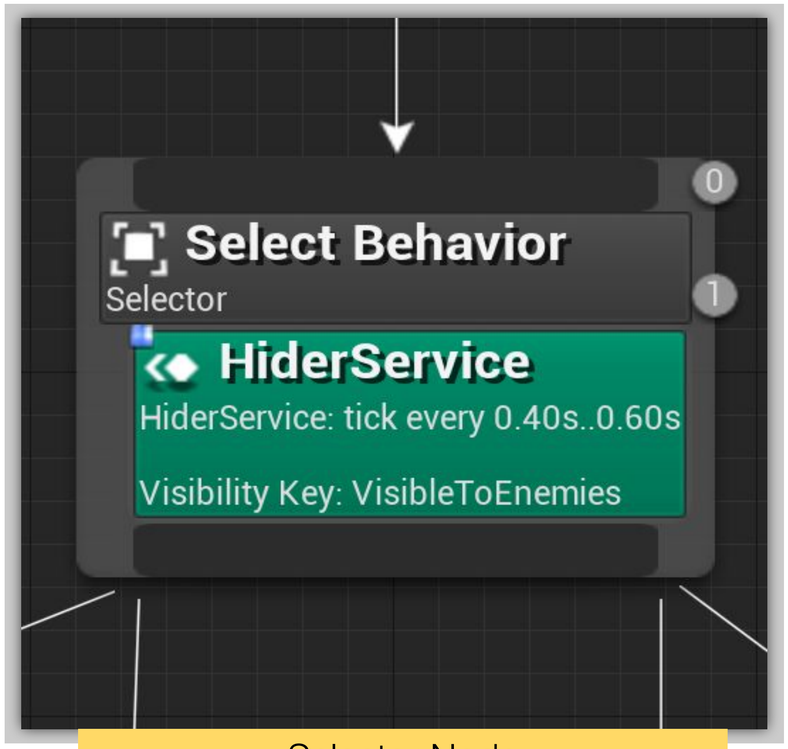
The **Root Node** is unique in the Behavior Tree and is the starting point for the Behavior Tree. It can only have one connection, and you cannot attach Decorators or Services to it.

Tasks

Task Nodes are the leaves of the Behavior Tree, the nodes that “do” things and don’t have an output connection.



Sequence Node



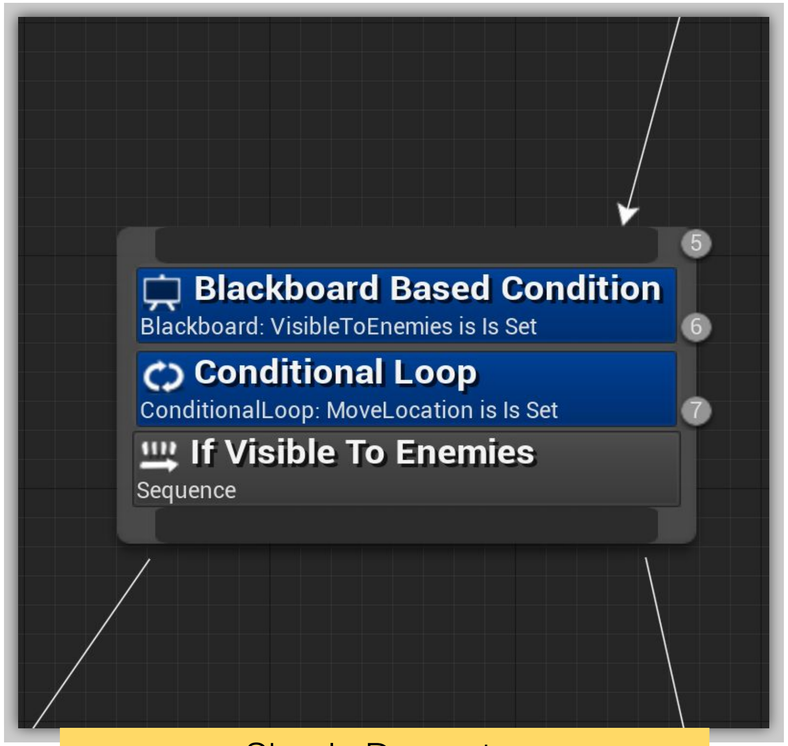
Selector Node

Composite: Sequence

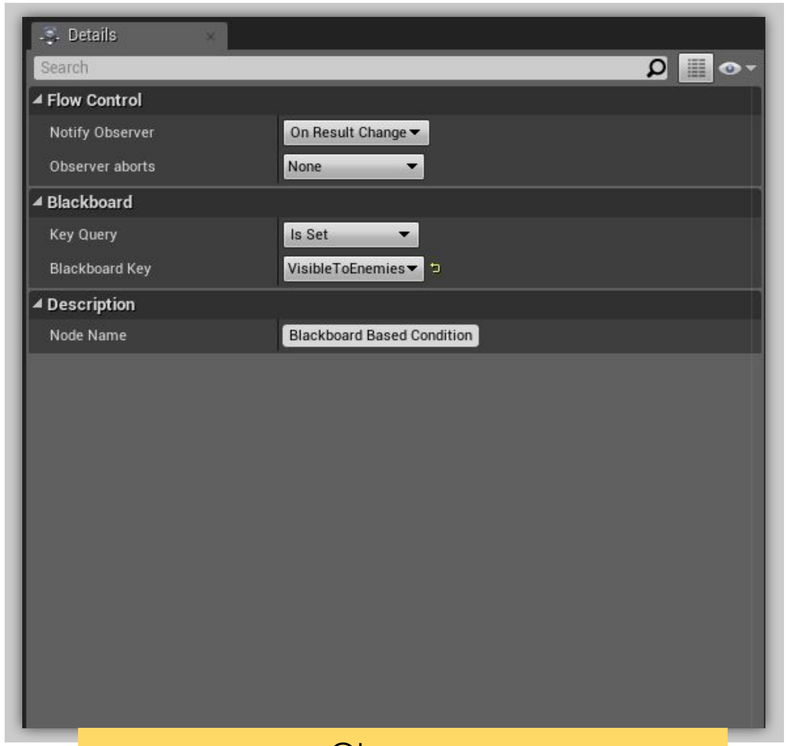
Sequence Nodes execute their children from left to right and will stop executing their children when one of their children fails. If a child fails, then the Sequence fails. If all the Sequence's children succeed, then the Sequence succeeds.

Composite: Selector

Selector Nodes execute their children from left to right and will stop executing their children when one of their children succeeds. If a child succeeds, the Selector succeeds. If all the Selector's children fail, the Selector fails.



Simple Decorator



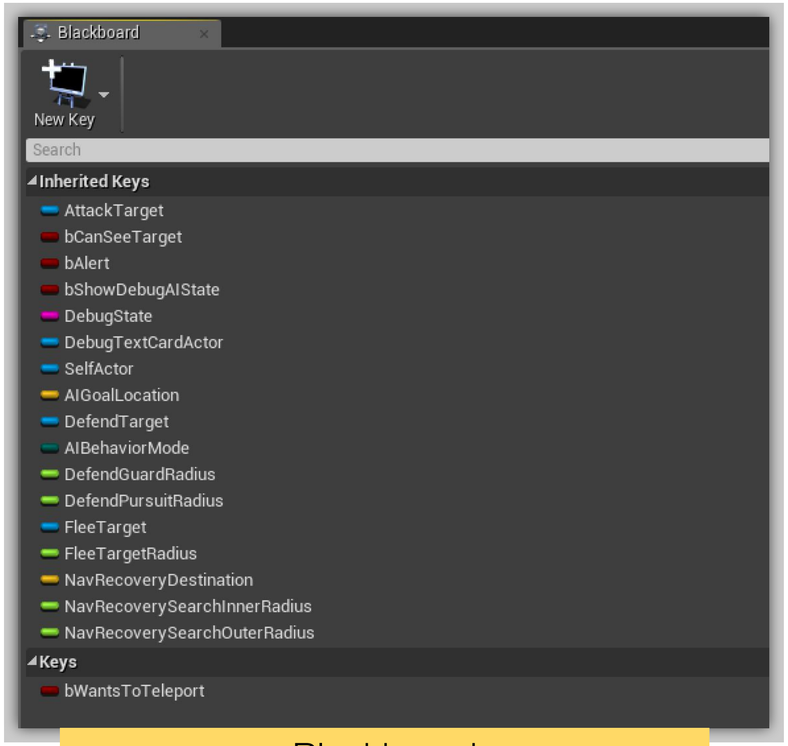
Observer

Decorators

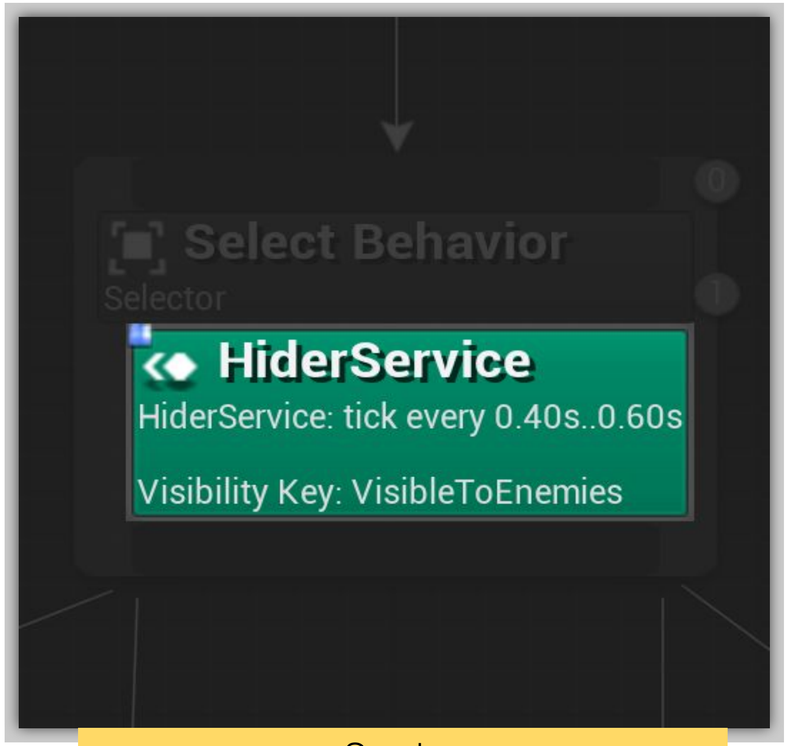
Decorators are also known as conditionals. They attach to another node and make decisions about whether or not a branch in the tree, or even a single node, can be executed.

Observers

Observers are a property of a decorator node that can observe values in the Blackboard and abort the execution of the branch if necessary.



Blackboard



Service

Blackboards

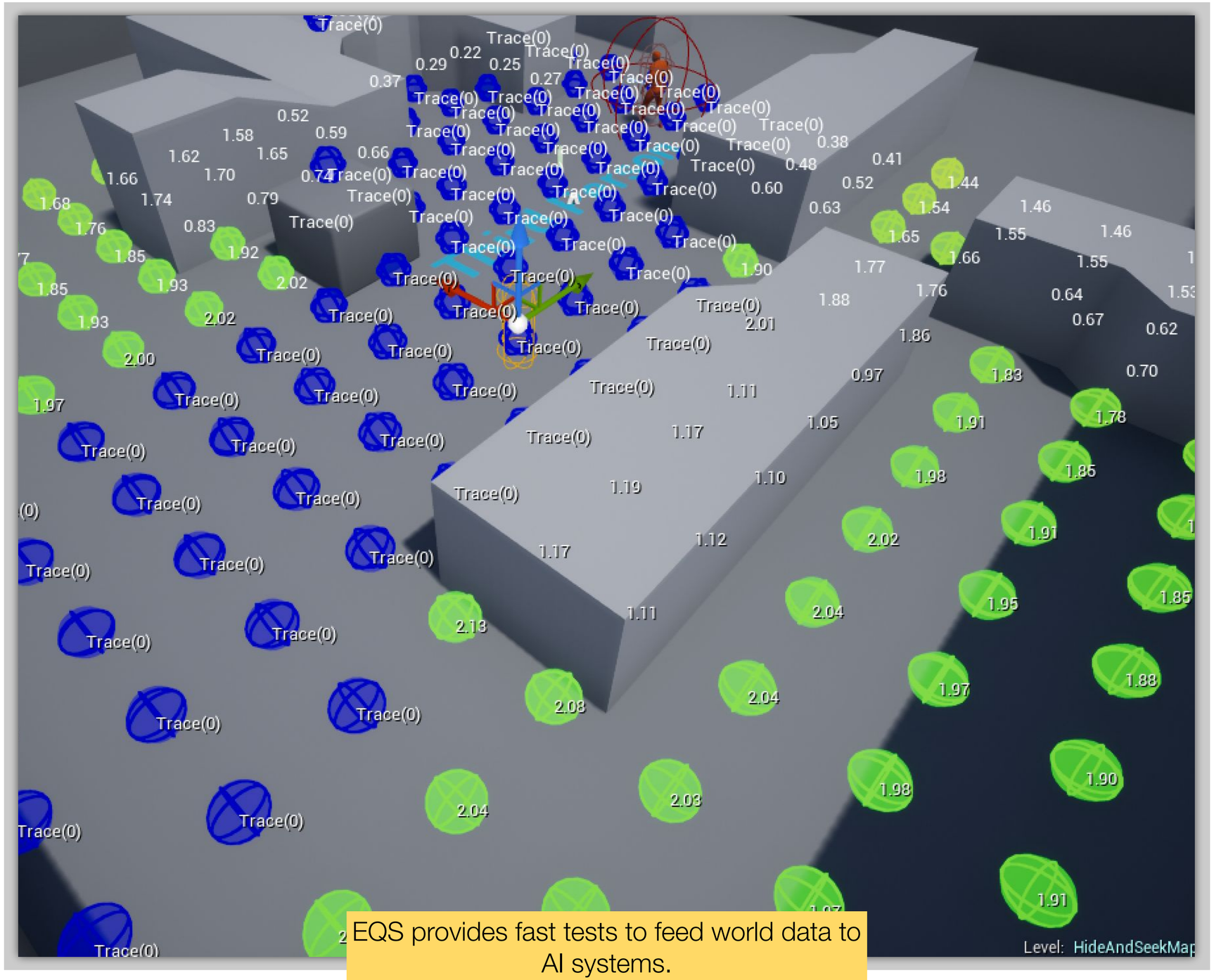
The Blackboard is the AI's memory. It stores key values for the Behavior Tree to use.

Services

Services attach to Composite nodes and will execute at their defined frequency as long as their branch is being executed. They are often used to make checks and to update the Blackboard.

AI Information Gathering

While AI can potentially access any information at any stage, it’s important to simulate human behavior and perception constraints. For this reason, it’s common to check when characters are within a certain radius of the AI and to perform intermittent traces to see if there is a clear line of sight between the player and the enemy. The enemy locations are usually compared using a dot product to ensure they’re in the same direction as the AI’s head; however, this is not always the case.

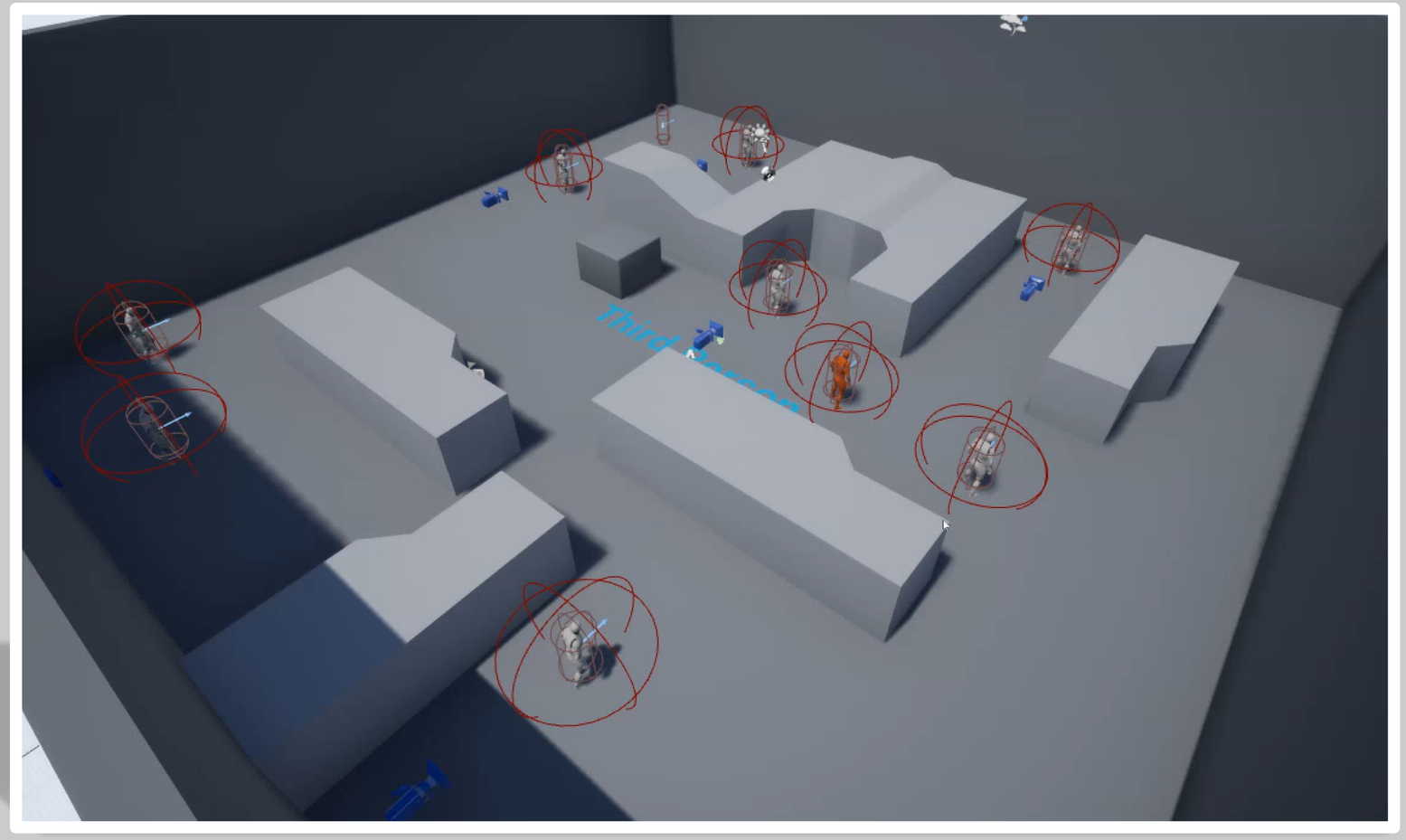


EQS provides fast tests to feed world data to AI systems.

ENVIRONMENT QUERY SYSTEM (EQS)

The **Environment Query System** is a feature of the artificial intelligence system in Unreal Engine 4 for collecting data on the environment, asking questions of the data through Tests, then returning the one item that best fits the tests performed.

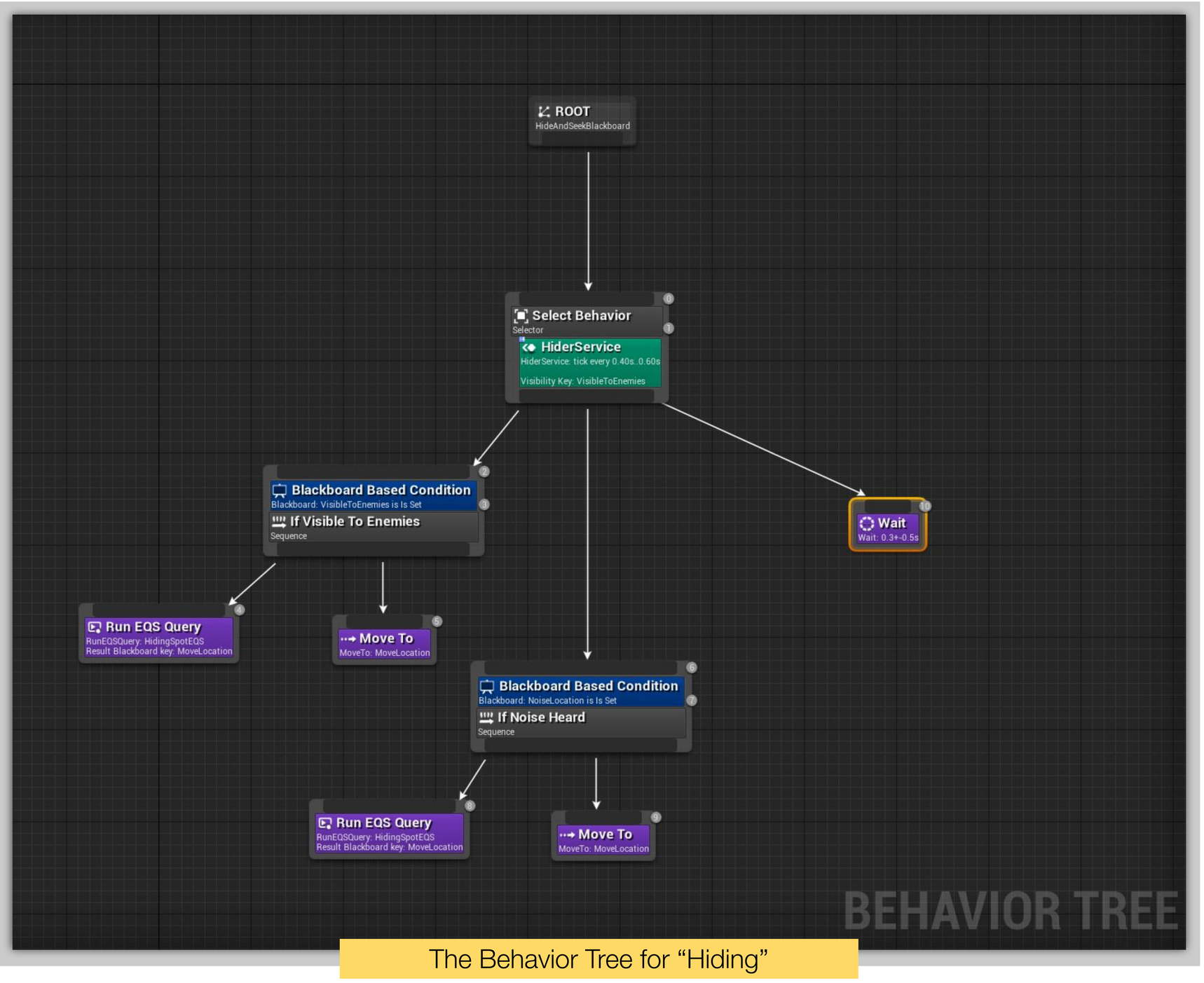
You can use it to find the closest power-up, to figure out which enemy is the greatest threat, or to find cover.



Both sets of AI characters use EQS to hide from or seek the other team.

HIDE AND SEEK AI: EXAMPLE VIDEO

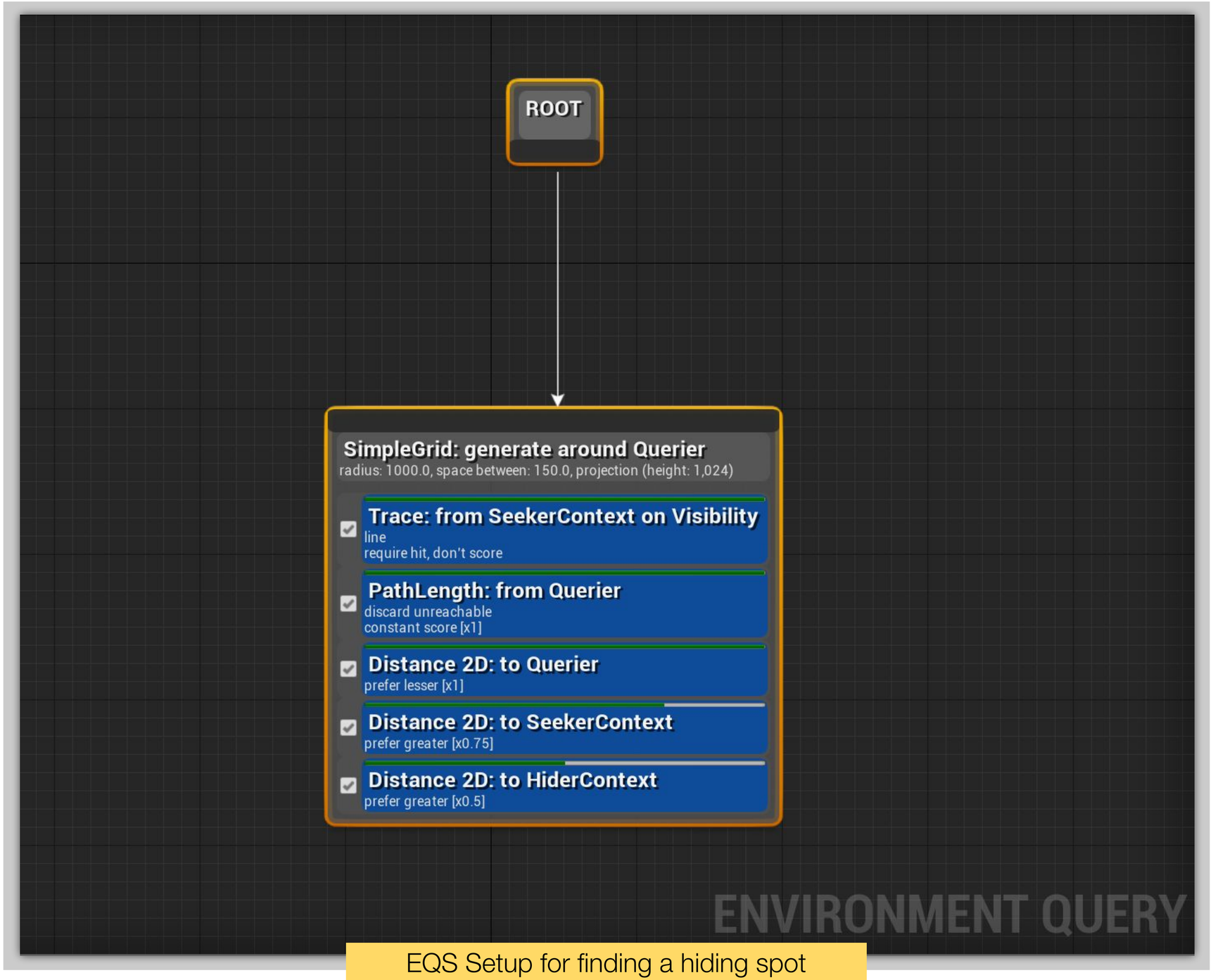
In this video, you can see a group of white AI characters using EQS to find hiding places away from any red characters.



The Behavior Tree for “Hiding”

HIDE AND SEEK AI: BEHAVIOR TREE

The Hider Service determines whether the AI can currently be seen or not. If the AI is visible or can hear an enemy coming, it will relocate itself to a better position. If it can’t hear anyone coming or is not visible to an enemy, it’ll wait before reconsidering these parameters.



EQS Setup for finding a hiding spot

HIDE AND SEEK AI: EQS

The **EQS** Query does the following:

- Traces between the point and any known “Seeker” (the term we used for the searching team in Hide and Seek)
- Discards any unreachable points in the grid
- Evaluates the distance to the player (prefers closer points)
- Evaluates the distance from any Seeker (prefers farther points)
- Evaluates the distance to any other “hiders” to look for separate hiding places

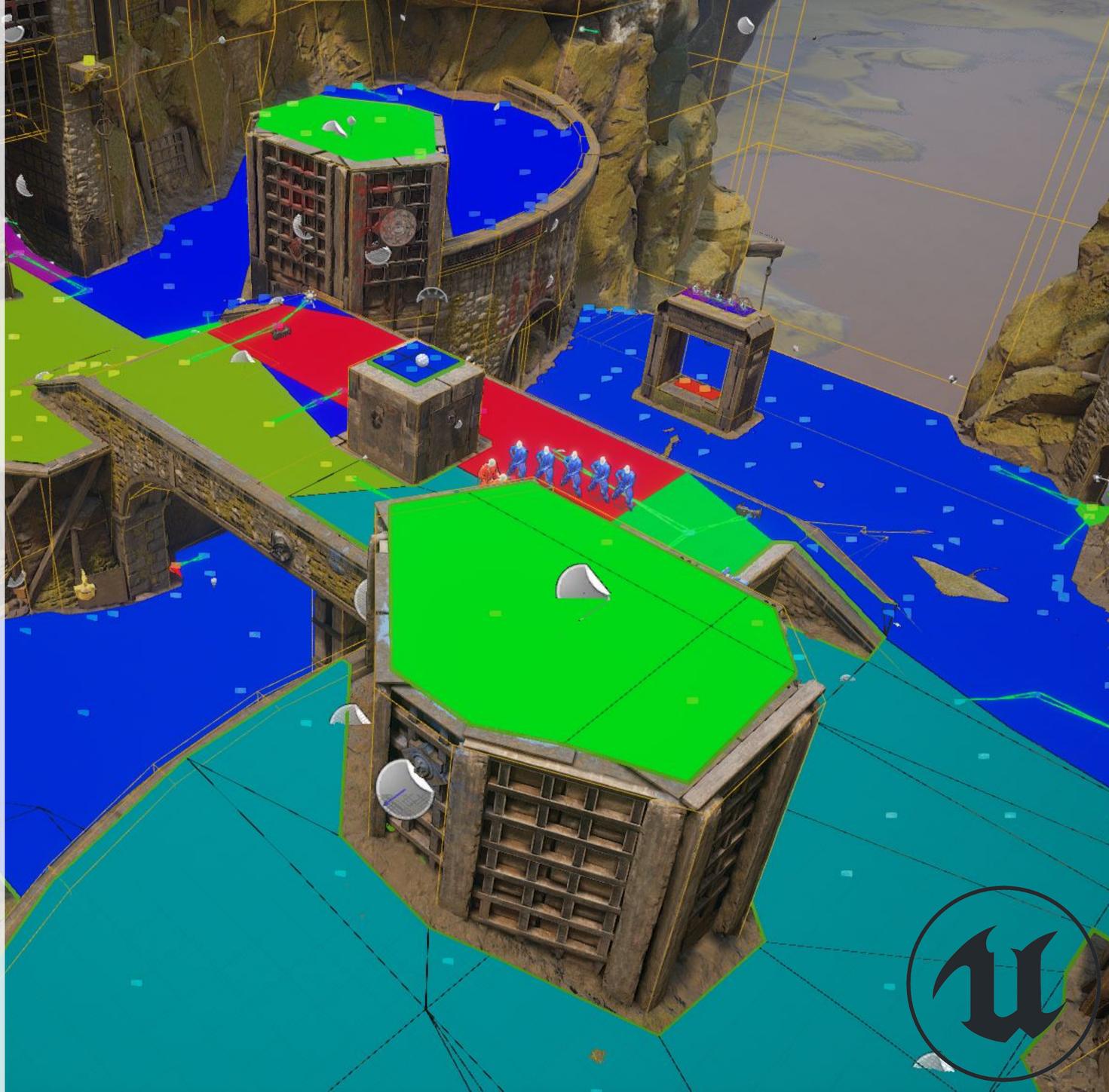
NAVIGATION

Processing and Interpreting the Important Elements of the World

NAVIGATION OVERVIEW

An AI character can only move through the world if it knows how. Unlike a human being, who can view and perceive an environment, AI requires the environment to be simplified into a format that can be more easily processed.

The process of identifying a path through an area is known as **Pathfinding**. The three most common types of navigation systems are Navigation Points, Navigation Grids, and Navigation Meshes.



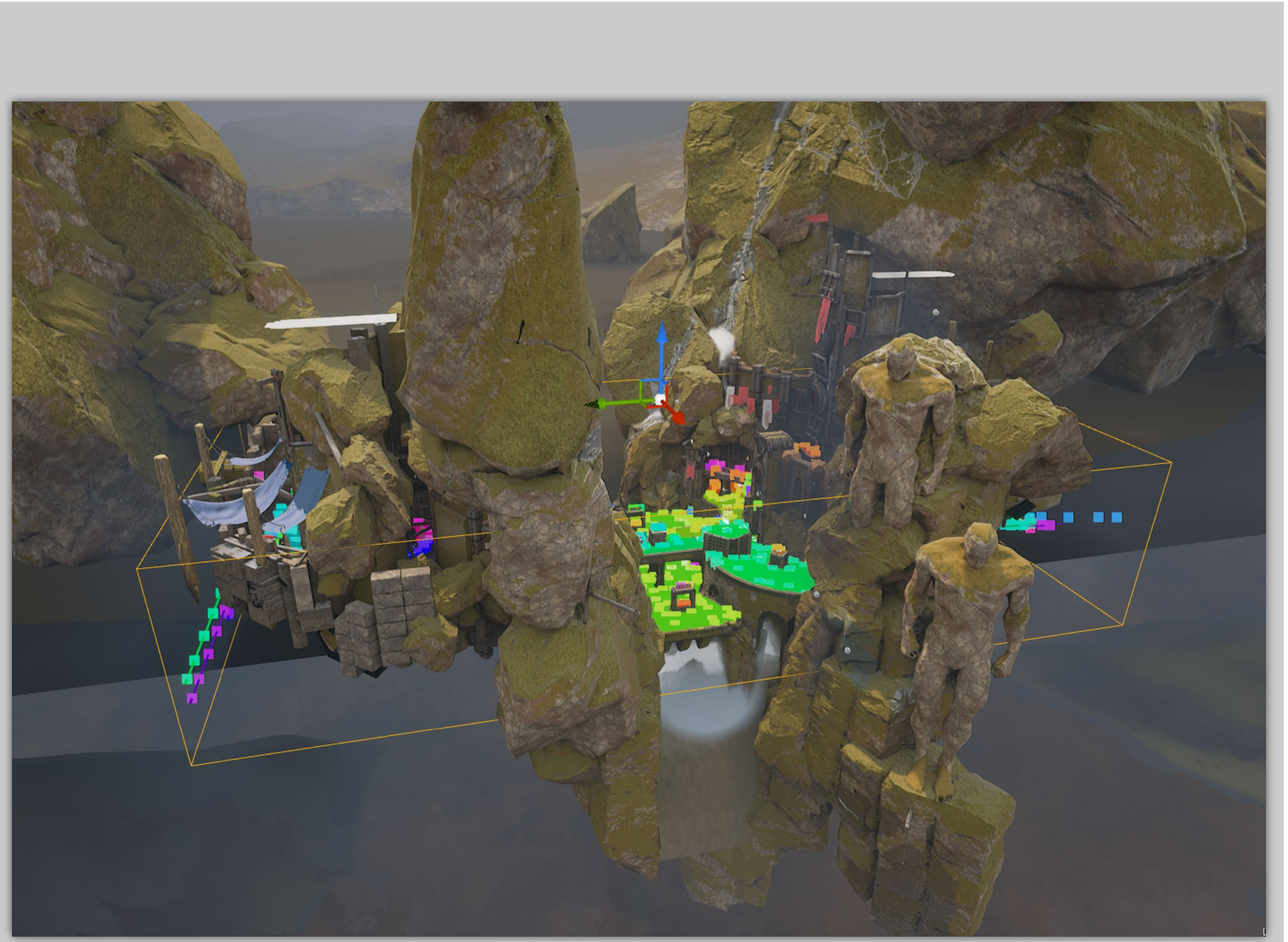
HOW DOES PATHFINDING WORK?

Dijkstra Pathfinding

Every node in a search graph knows what other nodes are connected to it. The Dijkstra pathfinding algorithm begins at the starting node in a graph and expands to any connected nodes if the distance to reach it is less than the currently known distance to reach it or if the node has never been traveled to. It expands through each node until the finishing point is found, at which point the path is discovered by walking back from the finishing node through to the surrounding nodes with the lowest score until the path back to the first node has been formed.

A* Pathfinding

The A* algorithm operates in the same way as a Dijkstra algorithm; however, it also includes the weighting of a heuristic in determining the score of a given node. This heuristic could be distance or a distance value modified by the type of ground between the two points.

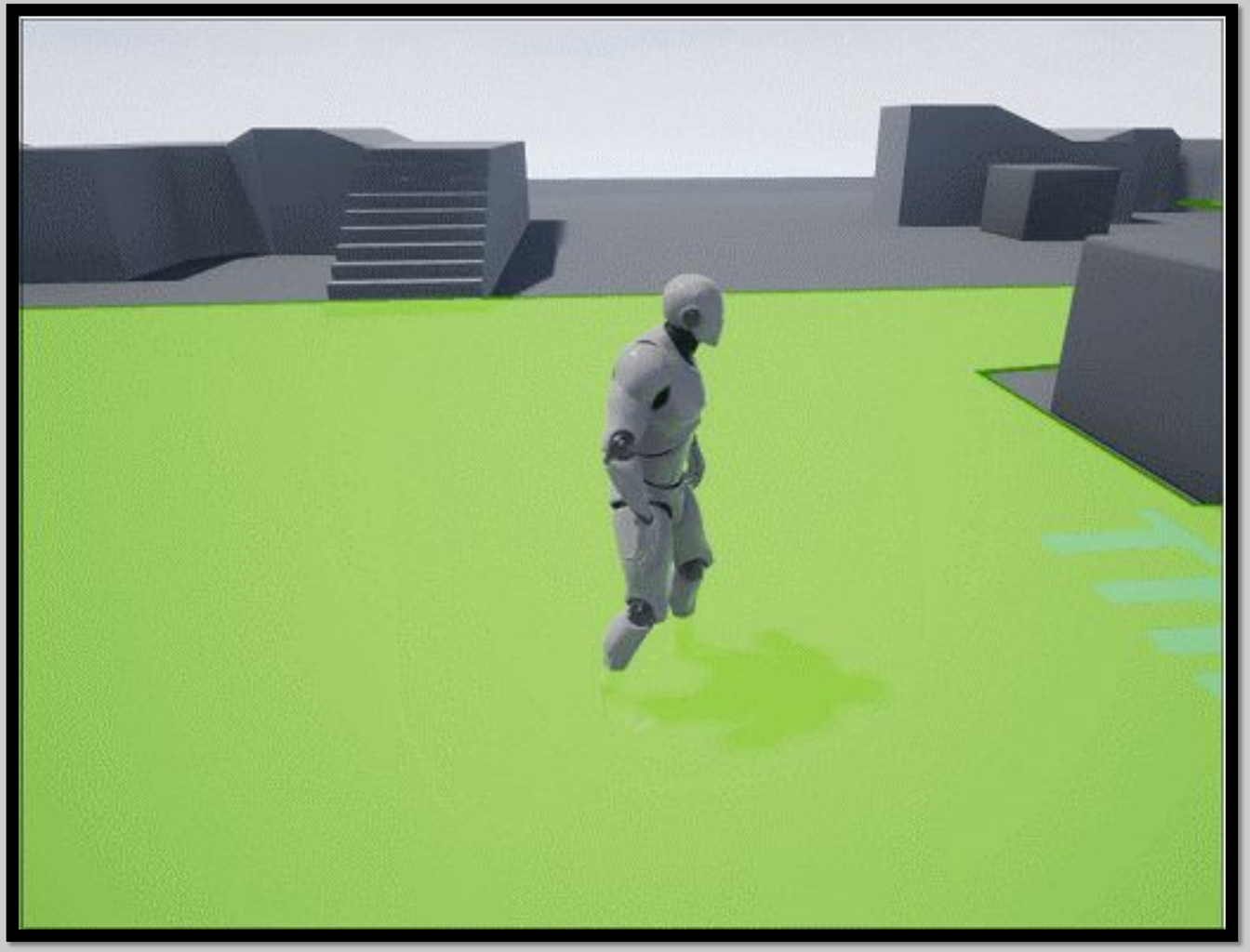


The Navmesh Bounds Volume calculates navigable areas in the map.

SETTING UP A NAVMESH IN UNREAL

To set up a navigation mesh in Unreal Engine, add a NavMeshBounds Volume to your environment. If you enable Path Viewing (P), you should see large green regions covering the map to indicate what areas can and can’t be reached by AI within the environment.

By default the NavMesh is calculated against an agent that is roughly the size of a typical human. Additional Agents can be added from the Project Settings.

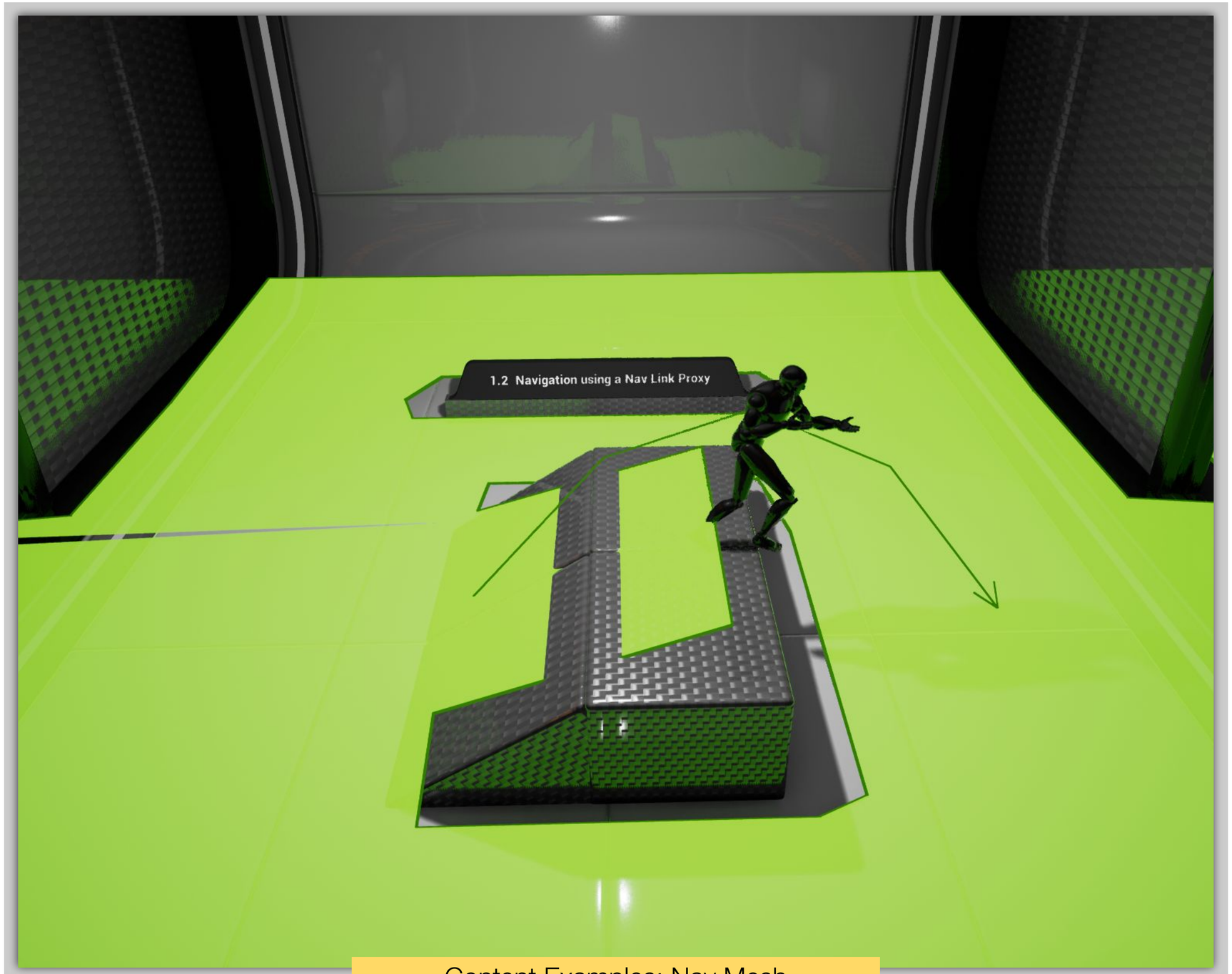


The NavMesh is extended as the Player moves

NAVMESH INVOKERS

Navigation meshes are typically built automatically during level development; however, for large-scale environments, this can take up considerable resources.

A solution is the use of navigation invokers, where navigation is only built around Actors with a NavMesh Invoker Component. When two Actors with this component walk close to one another, their navigation regions connect.



Content Examples: Nav Mesh

NAVLINK PROXY

Navigation Link Proxies are a way of connecting sections of NavMeshes that aren't automatically connected through an automatic connection. They could be used for ledges, elevators, or other environmental quirks.

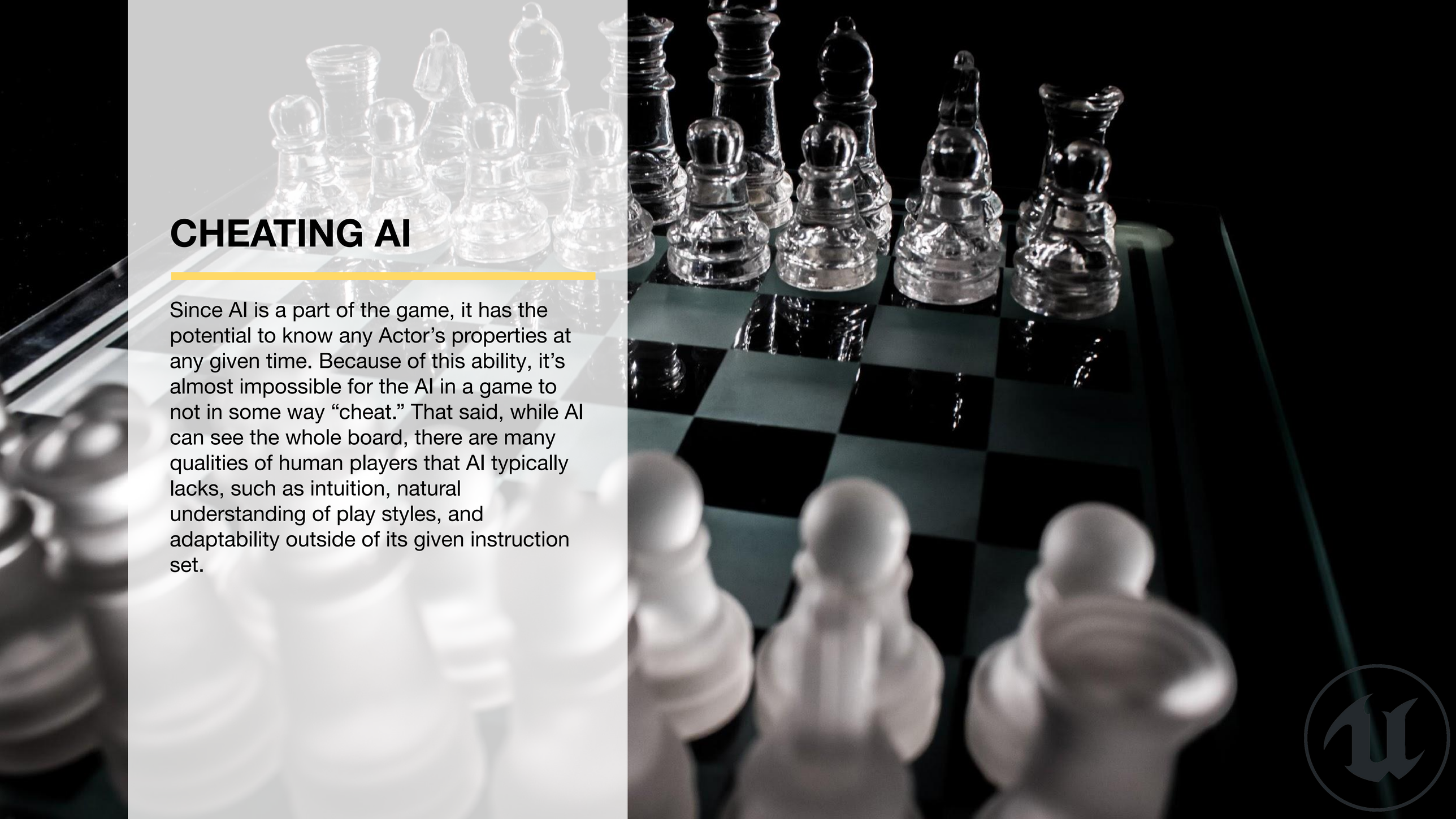
Navigation Link Proxys can be expanded with smart links to automatically trigger behaviors in Actors trying to cross them.

AI IMPERFECTION

Making Things Seem Normal,
Even When They’re Not

CHEATING AI

Since AI is a part of the game, it has the potential to know any Actor’s properties at any given time. Because of this ability, it’s almost impossible for the AI in a game to not in some way “cheat.” That said, while AI can see the whole board, there are many qualities of human players that AI typically lacks, such as intuition, natural understanding of play styles, and adaptability outside of its given instruction set.



The enemy can cheat, but you probably won’t know.

Known Friendlies

It’s common for AI to know where its friends are but to rely on a given perception system to detect enemies. The first method of developing around known friendlies is to limit perception of friendly AI to a controlled field of vision. Players are unlikely to know where enemies are at any given time, so they are unlikely to spot this semi-omniscient behavior.

Known Environment Layout

In almost all cases, AI is informed of its environment layout. Unlike players, who may look at a bridge and ask “How do I get there?” the AI is almost always informed of how to reach any given location that is reachable.

Key Item Locations

Since the perception of being “reactionary” is important, game items like flags and power-ups often have their positions known to the AI at any given time. This means that when a key event occurs, the enemies in a game are able to react to it intelligently instead of relying on perception.

Full Player Comprehension

AI often keeps a list of references to any known/perceived enemies. This list is updated when a player is seen, and when the AI has lost sight of a player there is typically an amount of time that the player and their actions continue to be tracked before the player is removed from the list. This means that the AI has a better general sense of permanence of threats in the area; it also means that it can know where the player is aiming or what they’re doing despite not having a direct view of them.

Artificial Stupidity

If an enemy were to open fire on a player and fire according to the information at its disposal, with an instant-hit weapon the enemy wouldn’t miss. With a projectile weapon it is very unlikely that the enemy would miss. Since AI is driven by variables and can instantly process almost any fActor in the game, its logic and aim are often intentionally compromised to allow for a fighting chance by players.

These intentional decreases in ability can occur in a variety of ways, including but not limited to the following:

- Artificial delays on information to simulate human reaction times
- Enemy AI missing its shots when first entering combat with a player to give the player time to react and head to cover
- Enemies decreasing in accuracy as a player reaches low health to help encourage battles that the player only just survives

Perceiving Intelligence

Part of intelligent behavior seeming intelligent is signaling intent to the player. When an enemy appears to have planned or coordinated its behavior, it appears to be drawing information from realistic sources and coordinating its actions as one would expect real players to do. This signaling can be done in various ways.

For instance, if an AI unit is in the process of approaching the player from the side, it may pass information to another AI player to shout out “Flank them!” to imply that the setup was intentional.

AI beyond Agents

AI Directors

AI can be wider reaching than a conventional game agent and used to direct a large number of game enemies or game events. An AI Director could be telling hordes of enemies which players to attack, identifying how to set up tense situations for players as they move through a level, or even determining how to add health and weapons for players so that they can feel like they’re just surviving by the skin of their teeth.

Player Assisted Guidance

AI can be used so that if a player gets lost, lights or other world aspects are triggered in a way that directs their attention toward the correct path. A more obvious way to do this is with nearby non-playable characters adding conversation elements that suggest the player move toward a goal.

Procedural Generation

AI can be used for a wide range of elements involved in procedural generation. For instance, AI could be used to intelligently automatically grow a forest with certain foliage types according to specific growth and distribution rule sets.

Game Balancing

AI can be used for automatic game balancing. By developing AI that is tracking how the player is playing the game, how erratic their movements are, or how they typically do in different engagements, AI could be used to make the game more or less difficult dynamically.

CONCLUSION

Questions?