

06 Developing AI in Games

Contents

[Overview 3](#_Toc508201486)

[6.1. Core Concepts 4](#_Toc508201487)

[AI Basics 4](#_Toc508201488)

[Behavior Trees 4](#_Toc508201489)

[AI Information Gathering 5](#_Toc508201490)

[EQS 5](#_Toc508201491)

[Navigation Meshes 5](#_Toc508201492)

[AI “Imperfections” 6](#_Toc508201493)

[6.2. Common Mistakes 6](#_Toc508201494)

[EQS 6](#_Toc508201495)

[NavMesh Bounds 6](#_Toc508201496)

[Exercises 7](#_Toc508201497)

[Exercise 6A: AI Basics 7](#_Toc508201498)

[Exercise 6B: Behavior Tree Basics 8](#_Toc508201499)

[Exercise 6C: EQS 9](#_Toc508201500)

[Exercise 6D: EQS and Service Tasks 10](#_Toc508201501)

[Exercise 6E: EQS in Blueprint 11](#_Toc508201502)

# Overview

**PURPOSE**

The purpose of this tutorial is to outline the features and functionality of the AI systems available within Unreal Engine 4.

**SCOPE**

This section will provide an introductory overview of the following:

* Core AI concepts within the context of Unreal Engine

**PREREQUISITES**

* A basic understanding of the Engine and how to use the tools
* An introductory knowledge of Blueprint

## 6.1. Core Concepts

### AI Basics

AI can be driven in many ways; however, **State Machines** and **Behavior Trees** are the two most common approaches in Unreal.

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\icon_Blueprint_StateMachine_24x.png **State Machines** typically divert the Tick functionality into the current state’s respective Tick. State machines are set up to allow the changing between certain states from given states depending on certain transition rules.

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BehaviorTreeMode_40x.png **Behavior Trees** work by redirecting flow down a “tree” whereby branches and leaf nodes can only be accessed given certain conditions.

### Behavior Trees

Behavior Trees are the most straightforward implementation of AI in Unreal. They are powerful but require a bit of setup:

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BehaviorTreeMode_20x.png

**Behavior Tree**

Directs logic flow

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BlackboardMode_20x.png

**Blackboard**

Stores knowledge required to direct logic flow and core instructions

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Service_24x.png

**Services**

Evaluate relevant information to control Blackboard parameters

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Non_Conditional_Decorator_24x.png

**Decorators**

Used to stop Composite nodes or Leaf from being entered as well as maintain Observers to reset the Behavior Tree if required

C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Selector_24x.png C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Sequence_24x.png C:\Users\Chad Mulroney\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Simple_Parallel_24x.png

**Composite nodes**

Used for node execution flow. Sequences execute from left to right, and when one node fails, Sequences will themselves fail. Selectors execute from left to right and stop when one of their children succeeds.

**Blackboard subtrees** can be loaded in dynamically for custom behaviors. For instance, a separate Dynamic Subtree could be loaded for different held items to keep the primary tree clean with all item-specific logic being isolated out.

They typically require the creation of additional BTTasks that are specific to the item.

Hide and Seek AI example available in the Academic Examples project:  
**World'/Game/Examples/HideAndSeekAI/HideAndSeekMap.HideAndSeekMap'**

### AI Information Gathering

* At any stage, AI has the potential to access almost any information within the world. The trick is to make it something that’s understandable from which simple conclusions can be drawn and act upon.
* Sometimes it is impossible to know if information is fed to the AI to ensure that gameplay is consistent. For instance, the location of a dropped flag could be passed to a bot without it ever seeing it.

### EQS

* EQS is an incredibly powerful tool that can be used to quickly process large amounts of information in the world.
* EQS can be used for more than just having a pawn understand its environment. It could also be used for gameplay tasks, such as identifying weaponless players and “randomly” dropping items closer to those in need to keep the game more even and tense.

### Navigation Meshes

* Navigation meshes divide the world into simplified navigable polygons that can be searched to find a path to a location.
* **NavMesh** islands can be linked using **NavLinkProxy** actors. This allows your pawns to move between different **NavMeshes**. A few examples where this is useful include elevators, ledges, and jump pads.
* Navigation meshes have a range of agents set up so that giants could be on one **NavMesh**, people on another, and mice on another. This allows for a wide range of potential paths for a variety of pawn agent sizes.
* Navigation Invokers can be used as a way of creating navigation data over a large area to save on path evaluation costs and memory overhead.

### AI “Imperfections”

* A wide range of AI imperfections can be employed to make a pawn seem more “human.” These can include the following:
* Delaying data from reaching the Behavior Tree to simulate human reactions
* Missing shots that AI could otherwise hit
* Missing first shots taken so that the player takes cover and becomes aware of the enemies’ presence or position

## 6.2. Common Mistakes

### EQS

EQS is an experimental feature that **must be enabled manually from the Editor Preferences**. If it is not manually enabled, the editor will give you errors on launch.

### NavMesh Bounds

If your pawns appear to be set up but aren’t moving, you may not have a **NavMesh** bounds volume in place to generate navigation data or, if you do, it may not be positioned over the area.

# Exercises

## Exercise 6A: AI Basics

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the most basic setup of an AI Controller.

**Student Task:** Create a new AI Controller that will use the default ThirdPersonCharacter as its pawn and make it wander around to random points within 1,000 units.



**Directions**

1. Create an AI Controller.
2. Create a Custom Event called “Wander” and call it from “OnPossess”.
3. Have the Wander function implement an “AIMoveTo” that sends it to a random reachable point in a radius around the controlled pawn.
4. Add a delay on both the success and fail execution outputs to reattempt to wander to a new location.
5. Set the AI Controller on the pawn in the scene.

**Note: Make sure you have a Nav Mesh Bounds Volume.**

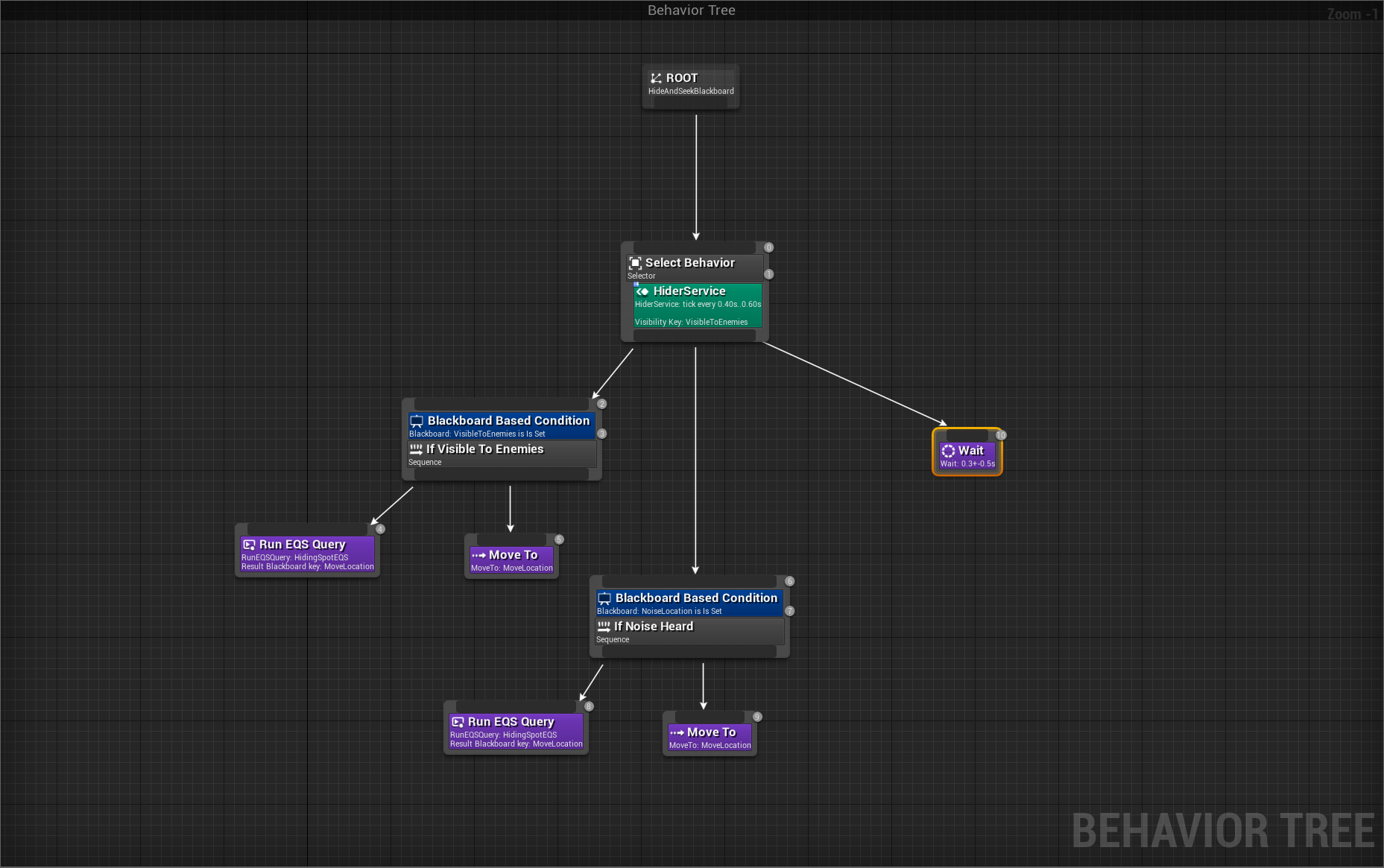
Example available here: **Blueprint'/Game/Examples/AI/AISimpleWander.AISimpleWander'**

## Exercise 6B: Behavior Tree Basics

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the basic setup of AI Behavior Trees.

**Student Task:** Create AI, using a Behavior Tree, that randomly wanders.

****

**Directions**

1. Create an AI Controller.
2. Create a Behavior Tree and Blackboard.
3. Add a Move Location Vector to the Blackboard.
4. Create a Blueprint Task to find a wander location that has a radius and Move Location Blackboard Selector variable.
5. Set the Blackboard Selector to the Vector value of the final location.
6. Move to that location using a Sequence and the two nodes.

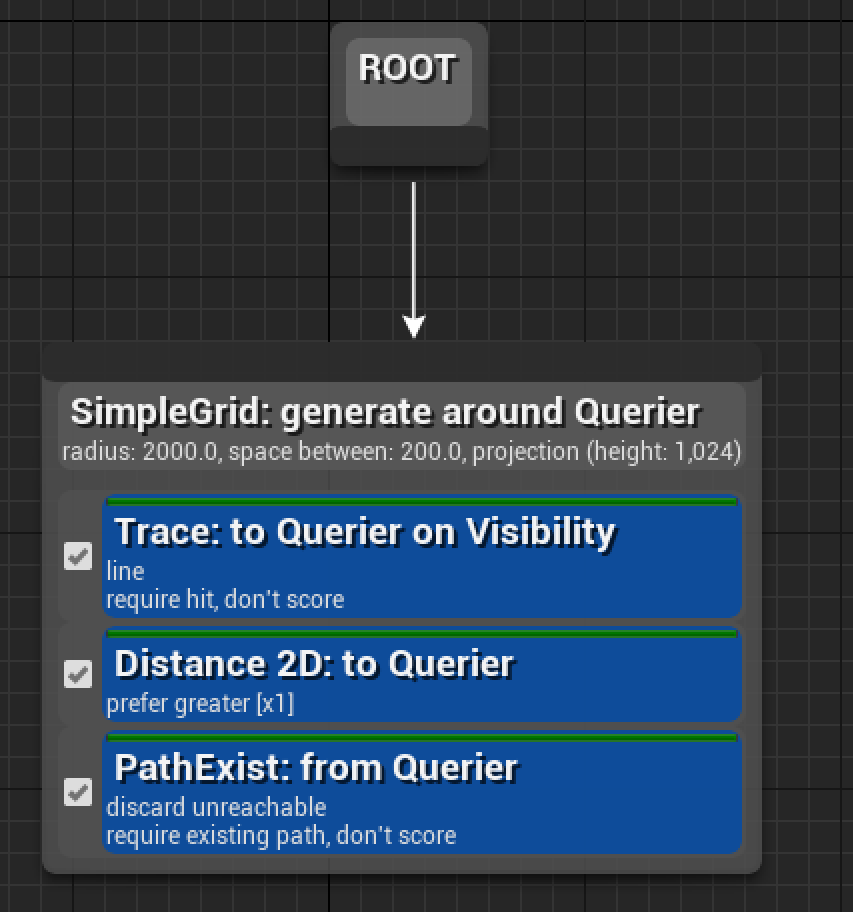
Example available here: **Blueprint'/Game/Examples/AI/AIBTWander.AIBTWander'**

## Exercise 6C: EQS

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the functionality of AI Behavior Trees when combined with EQS.

**Student Task:** Create a new AI that uses EQS to move to a location that it currently can’t see.



**Directions**

1. Create an AI Controller.
2. Create a Behavior Tree and Blackboard.
3. Create an EQS Query to find a wander location that cannot currently be seen by the bot, using a filter, and then score the possibilities according to distance from the query. **If you cannot see EQS listed please enable the Environmental Query System in the Experimental section of your Editor Preferences.**
4. Test your EQS Query by creating a Blueprint of the EQS Test Pawn and setting the EQS test to your newly developed test.
5. Move to one of the top 25% farthest away using a sequence with the EQS Query and a Move To command.

Example available here: **Blueprint'/Game/Examples/AI/AIBTIntelligentWander.AIBTIntelligentWander'**

## Exercise 6D: EQS and Service Tasks

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the functionality of AI Behavior Trees when combined with EQS and Selectors/Sequences.

**Student Task:** Create a new AI that uses EQS to move to a location that it currently can’t see until it finds an “egg” Blueprint. Once it has found the Egg Blueprint, it returns to a “home” location to return the Egg or alternatively destroys the Egg.

**Directions**

1. Create a simple “Egg” Blueprint Actor with a Static Mesh Component and enable Simulate Physics.
2. Extending upon your Behavior Tree in the previous exercise, add an Egg Target object of type Egg to the Blackboard.
3. Create a Service Task Blueprint that will do the following:
   1. Conduct a spherical overlap around the pawn for Eggs
   2. Check if an Egg is currently attached to an actor and ignore it if it is
   3. Set the Egg to be stored as a Blackboard reference
4. Add the Service to detect when an Egg actor is visible to the Behavior Tree.
5. If an Egg is visible, move to the Egg.
6. Create a BT Task to perform to either return the Egg to a destination or destroy the Egg when reached.

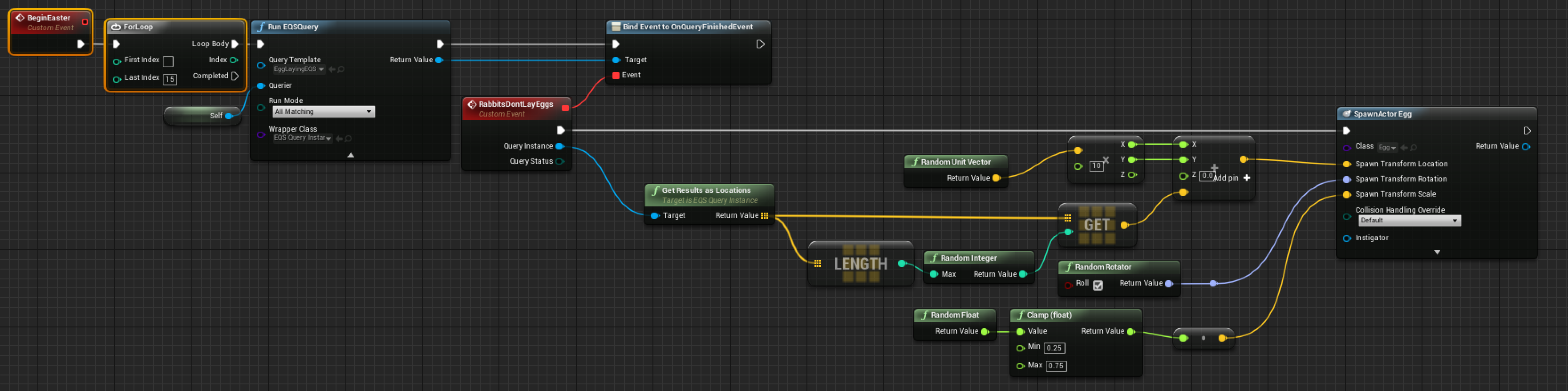
Example available here: **Blueprint'/Game/Examples/AI/AIBTEggHunt.AIBTEggHunt'**

## Exercise 6E: EQS in Blueprint

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the functionality of EQS as a tool for object placement in the world.

**Student Task:** Create a Blueprint that uses EQS to place Egg collectables around the map in positions that no pawns can see.



**Directions**

1. Create an EQS Context to return Pawns (aside from the query, if they’re a pawn).
2. Create an EQS Query to return locations that cannot be seen by any pawns.
3. Create a Blueprint that will spawn Eggs in these locations when Begin Play is pressed by calling an EQS Query. Remember that as EQS Queries can take some time to run, an event needs to be bound to their completion event.

Example available here: **Blueprint'/Game/Examples/AI/EasterBunny.EasterBunny'**