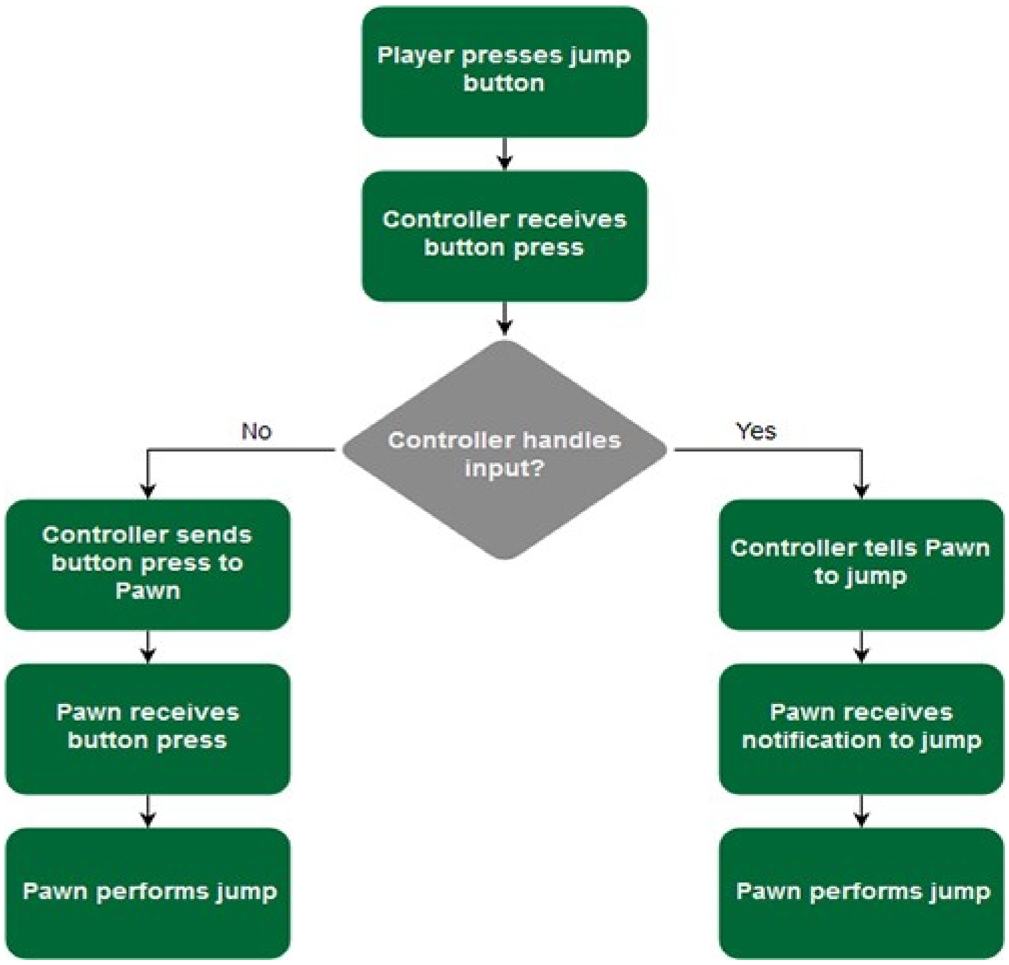
Documentation

Unreal Engine Ai Assignment

- Create an AI entity that can control a Pawn
- Create and use behavior trees and blackboards
- Use AI Perception to give the Pawn sight
- Create behaviors to make the Pawn roam and attack enemies

A controller is a non-physical actor that can *possess* a Pawn. Possession allows the controller to—you guessed it—*control* the Pawn. But here 'control' mean:

For a player, this means pressing a button and having the Pawn do something. The controller receives inputs from the player and then it can send the inputs to the Pawn. The controller could also handle the inputs instead and then tell the Pawn to perform an action.


Player presses jump button
Controller receives button press
Controller handles input?
No
Yes
Controller sends button press to Pawn
Pawn receives button press
Pawn performs jump
Controller tells Pawn to jump
Pawn receives notification to jump
Pawn performs jump

The Behavior Tree Editor

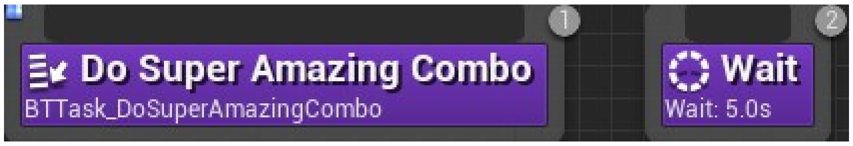
The behavior tree editor contains two new panels:

1. ***Behavior Tree:* This graph is where you will create nodes to make the behavior tree**
2. ***Details:* Nodes you select will display its properties here**
3. ***Blackboard:* This panel will show Blackboard keys (more on this later) and their values. Will only display when the game is running.**

Like Blueprints, behavior trees consist of nodes. There are four types of nodes in behavior trees. The first two are *tasks* and *composites*.

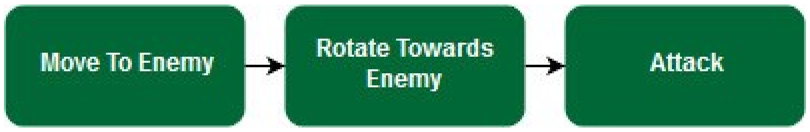
What are Tasks and Composites?

As its name implies, a task is a node that "does" something. This can be something complex such as performing a combo. It could also be something simple such as waiting.

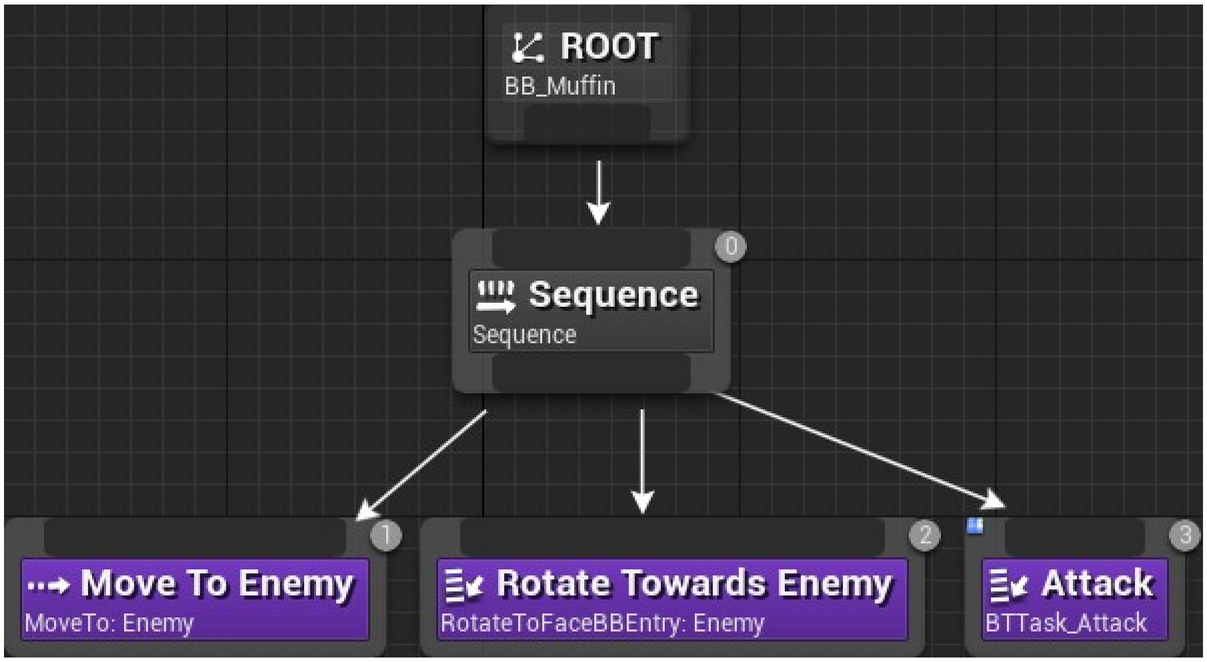


To execute tasks, you need to use composites. A behavior tree consists of many branches (the behaviors). At the root of each branch is a composite. Different types of composites have different ways of executing their child nodes.

For example, you have the following sequence of actions:



To perform each action in a sequence, you would use a *Sequence* composite. This is because a Sequence executes its children from left to right. Here's what it would look like:



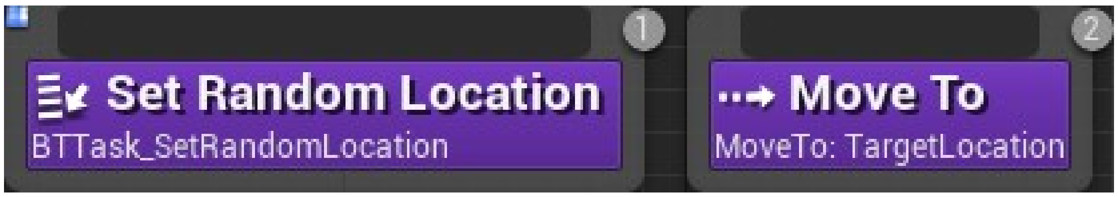
If any of a Sequence's children *fail*, the Sequence will stop executing.

For example, if the Pawn is unable to move to the enemy, *move to Enemy* will fail. This means *Rotate Towards Enemy* and *Attack* will not execute. However, they will execute if the Pawn succeeds in moving to the enemy.

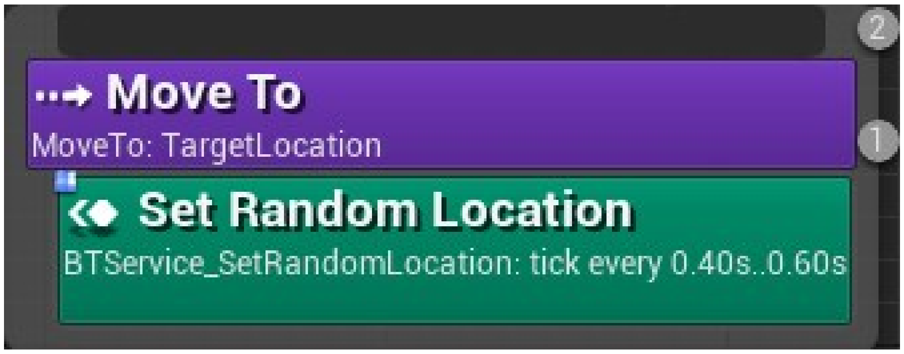
What is a Service?

Services are like tasks in that you use them to do something. However, instead of making the Pawn perform an action, you use services to perform checks or update the blackboard.

Services are not individual nodes. Instead, they attach to tasks or composites. This results in a more organized behavior tree because you have less nodes to deal with. Here's how it would look using a task:



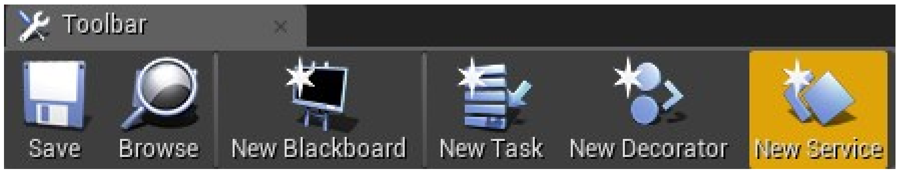
Here’s how it would look using a service:



Now, make a service that will generate a random location.

Creating a Service

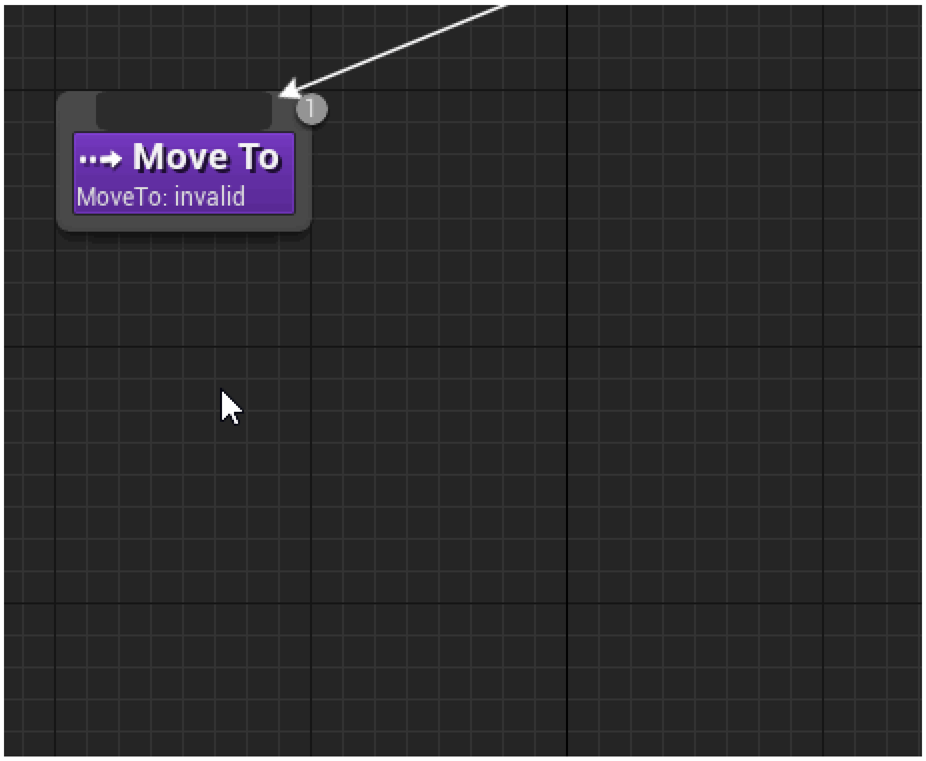
Go back to behavior tree that made *and click* New Service.



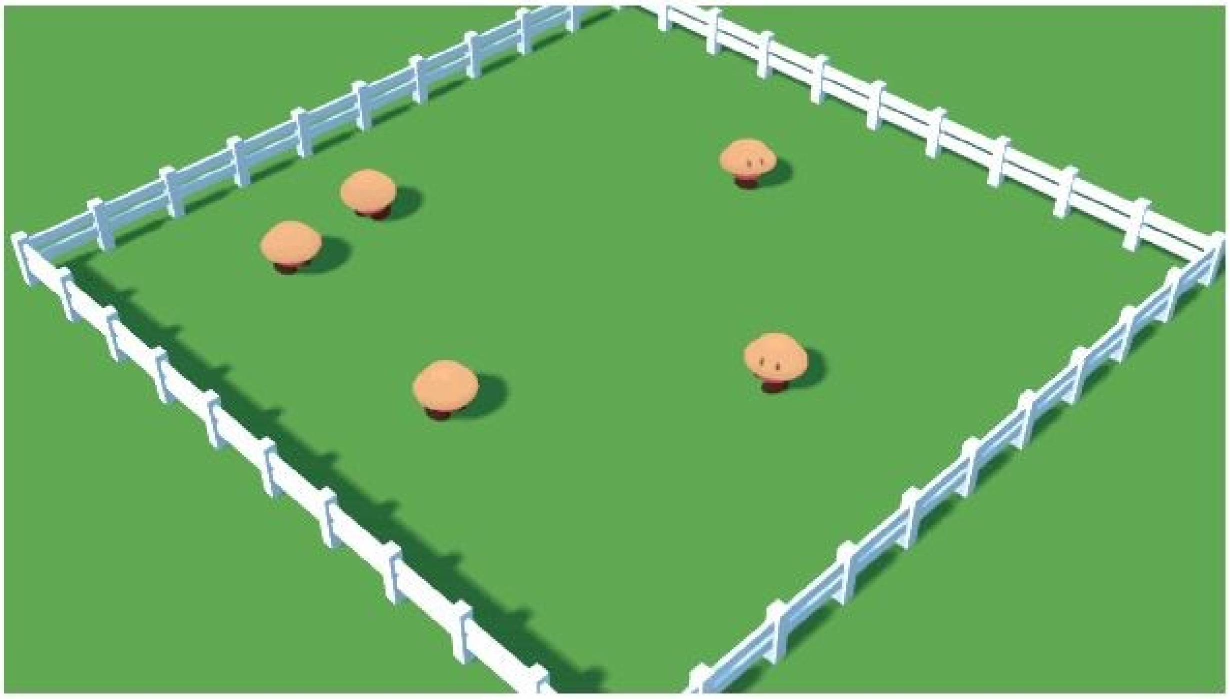
This will create a new service and open it automatically. Name it *BTService_SetRandomLocation*. You’ll need to go back to the Content Browser to rename it.

The service only needs to execute when the Pawn wants to move. To do this, you need to attach it to *MoveTo*.

Open *BT_Muffin* and then *right-click* on *MoveTo*. Select *Add Service\BTService Set Random Location*.



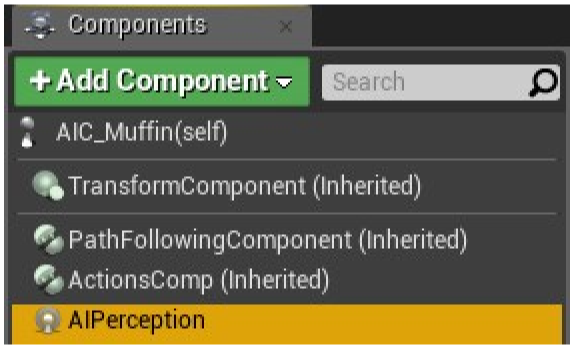
Now, *BTService_SetRandomLocation* will activate when *MoveTo* activates.



Setting Up AI Perception

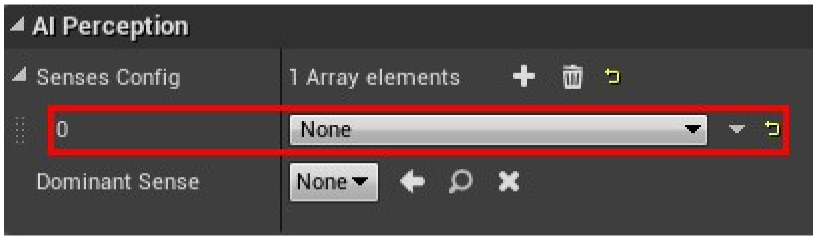
AI Perception is a component you can add to actors. Using it, you can give *senses* (such as sight and hearing) to your AI.

Open *AIC_Muffin* and then add an *AIPerception* component

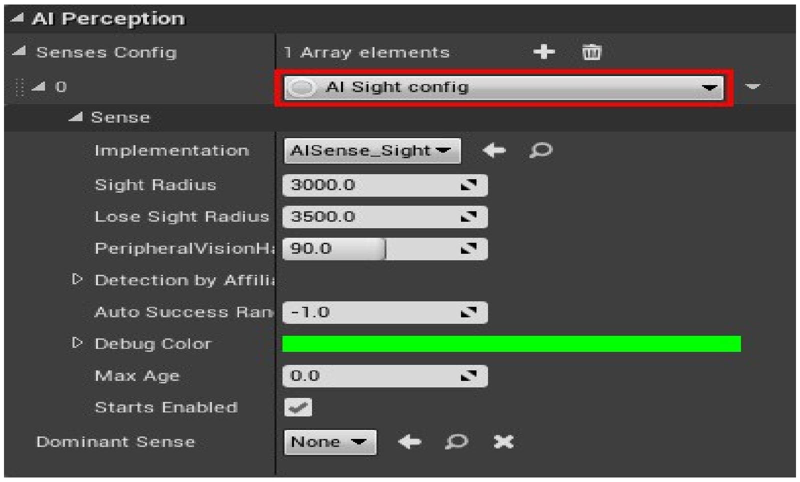


Next, you need to add a sense. Since you want to detect when another player moves into view, you need to add a *sight* sense.

Select *AIPerception* and then go to the Details panel. Under *AI Perception*, add a new element to *Senses Config*.

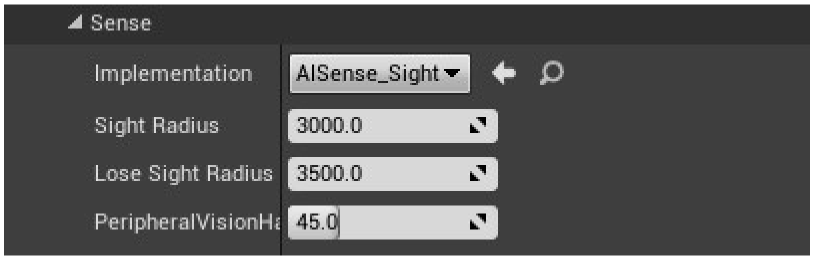

AI Perception
Senses Config 1 Array elements
0 None
Dominant Sense None

Set element *0* to *AI Sight config* and then expand it.


AI Perception
Senses Config 1 Array elements
0 AI Sight config
Sense
Implementation AISense_Sight
Sight Radius 3000.0
Lose Sight Radius 3500.0
PeripheralVisionH: 90.0
Detection by Affili
Auto Success Ran -1.0
Debug Color 🟩
Max Age 0.0
Starts Enabled ✓
Dominant Sense None

There are three main settings for sight:

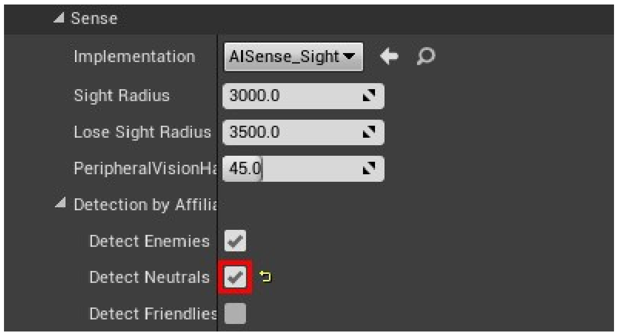
1. *Sight Radius:* The maximum distance the player can see. Leave this at *3000*.
2. *Lose Sight Radius:* If the player has seen an enemy, this is how far the enemy must move away before the player loses sight of it. Leave this at *3500*.
3. *Peripheral Vision Half Angle Degrees:* How wide the player's vision is. Set this to *45*. This will give the player a *90* degree range of vision.



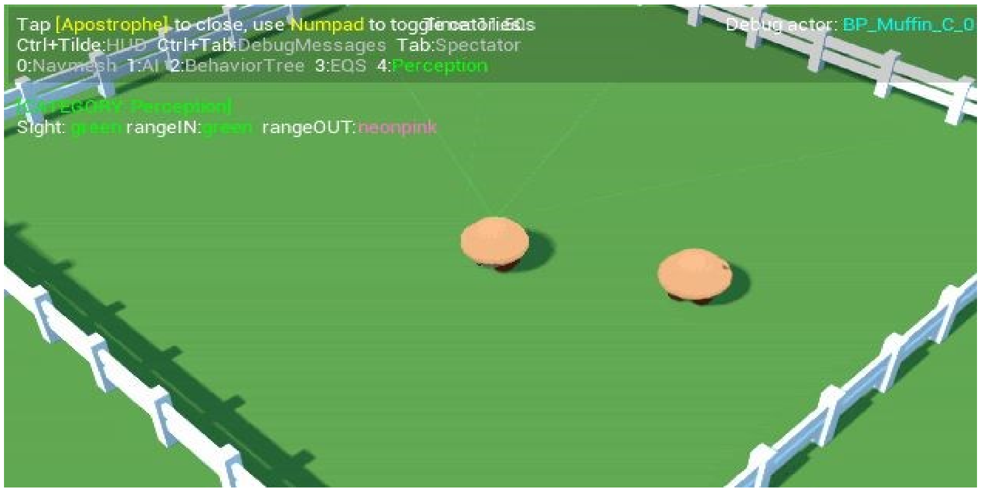
Sense
Implementation: AISense_Sight
Sight Radius: 3000.0
Lose Sight Radius: 3500.0
PeripheralVisionHa: 45.0

By default, AI Perception only detects enemies (actors assigned to a different *team*). However, actors do not have a team by default. When an actor doesn't have a team, AI Perception considers it *neutral*.

As of writing, there isn't a method to assign teams using Blueprints. Instead, you can just tell AI Perception to detect neutral actors. To do this, expand *Detection by Affiliation* and enable *Detect Neutrals*.



Sense
Implementation: AISense_Sight
Sight Radius: 3000.0
Lose Sight Radius: 3500.0
PeripheralVisionHa: 45.0
Detection by Affilia
Detect Enemies: ✓
Detect Neutrals: ✓
Detect Friendlies:



```
Tap [Apostrophe] to close, use Numpad to toggle categories                Debug actor: BP_Muffin_C_0
Ctrl+Tilde:HUD  Ctrl+Tab:DebugMessages  Tab:Spectator
0:Navmesh  1:AI  2:BehaviorTree  3:EQS  4:Perception

[CATEGORY: Perception]
Sight: green  rangeIN:green  rangeOUT:neonpink
```