Roaming, Chasing and Pawn Sensing using Behavior Trees

Add a navmesh bounds volume in the scene and set it to the level where you want the navigation to be possible. Make a blueprint class of type Character called controlling AI. Create a behavior tree called controlling AI BT, a blackboard called AI BB and an AI Controller called My\_AIC.

Open the character BP and add the static/skeletal mesh which you want for the character. Set the collision to character mesh maybe and in that case if you put it in the world you can see that it doesn’t affect the navmesh. Click on the class defaults and set the AI controller class to My\_AIC and set the auto possess AI to placed in world or spawned so whenever it enters the world or is spawned the My\_AIC possess it.

Create a variable called BT of type behavior tree. Make a function/event called setup AIC. This is going to be used for setting values to variables etc. This is makes setting variables and changing parameters easier. Extend the setup AIC node and add a set BT node and extend the input pin into the setup AIC node so that the setup AIC node now takes BT as an input parameter. Extend the output exec pin of the Set BT node and add a run behavior tree node and set the BT Asset pin to the value of the BT variable.

Go to controlling AI BP and add a get AI controller node, extend it and add a cast to My\_AIC node. Hook the input exec pin of the cast to My\_AIC node to the output exec pin of the Event Begin play node. Extend the as My\_AIC pin and add a setup AIC node and set the BT to controlling AI BT. Don’t forget to set the controlled actor pin in the get AI controller node. In this case set it to self.

This setup also helps us to use the AIC for lots of kinds of enemies that run different behavior trees while they use the same classes and pawn sensing etc. If you want you can promote the BT pin to a variable inside the Controlling AI BP and then set the value to Controlling AI BT too.

Open the Controlling AI BT and set the blackboard to AI BB. Extend the root and add a sequence node. Extend the sequence node and add the wait node. drag the wait to the far right of the sequence.

Now on simulation, go to the BT and we can see the control goes from root to wait node which means that so far the setup is working.

Now we add the roaming behavior.

In the controlling AI BT replace the sequence node with a selector node. A selector node goes to the far left and checks if the condition is satisfied if yes it’s completed else it goes to the next left-most node and so on. Extend the selector node and add a sequence node. To find the location we use a task so click on the create task button and rename it to Find Roaming Location BTT. Open it and add an event execute AI node, extend the controlled pawn and add a get actor location node and extend the return value and add a get random reachable point in navigable radius. Extend the radius pin and promote it to a variable called Radius and set the default to say 1000 and make it public by clicking the eye icon. We need to save the output in a BB key as we are going to use it in another task. So go to the BB and add a new key called MoveToLocation of type vector. Now go back to the Find Roaming Location BTT event graph and extend the random location pin and add a set blackboard value as vector node. Extend the key and promote it to a new variable called RandomLocationOutputKey and make it public by clicking the eye icon. Hook the input exec pin of the set blackboard value as vector node to the output exec pin of the event receive execute AI node. Extend the output exec pin of the set blackboard value as vector node and add a finish execute node and check the success pin. The finish execute node tells the BT that the task is complete so it’s very important.

So now whenever the Find Roaming Location BTT is fired it gets a random location and sets the key of the blackboard’s value to a random location. Go to Controlling AI BT and extend the sequence node and add the Find Roaming Location BTT node. In the default section in the right-hand window you can see that the radius and Random Location Output key variables are visible. Set the random location output key to Move To Location. Extend sequence node and to the right of the find roaming location BTT add a Move To node and set the move to parameter to MoveToLocation node. Extend the sequence node and add a Wait node to the right of the move to node. Set the wait time to 5s with a random deviation time of 2s.

Now on simulation, the control goes from selector to the sequence and stays there. The character starts to roam around and wait for intervals.

Next, we setup pawn sensing. We do the setup for pawn sensing in the AI controller and this allows us to use the same setup for multiple enemies of different classes. Open the My\_AIC and in the viewport add a pawn sensing component. The problem with pawn sensing is that the default one provided by UE can’t be made a child of the mesh so if we have animations on the mesh like head turning etc. the pawn sensing doesn’t turn along with it. To get around this we can make our own pawn sensing and this gives greater freedom and functionality. However, in this tutorial we are going to use the default pawn sensing. Click on the pawn sensing component and on the right hand window click on the on see pawn function. Once it appears in the event graph extend the pawn pin and add an equal object node. Extend the other pin and promote it to a variable called Pawn Sensing Target. We need to set the Pawn Sensing Target and then change the type to pawn object but we do all the variable setting in the Setup AIC part. So after the run behavior tree node add a set Pawn sensing target node and hook its input pin to the setup AIC node so that a new input parameter is added in the setup AIC node. It has to be after the run behavior tree as we are calling the blackboard. Go to the Controlling AI’s event graph and in the part where we call the setup AIC, extend the pawn sensing target in and add a get player node so as to make the character detect the player.

Go back to the on see pawn definition and extend it and add a branch node. Hook the return value of the equal object node to the condition pin. We are going to set this to a blackboard entry but then we find ourselves repeating the code for setting blackboard entries so instead of that we make a simple function library so that we need to write it only once.

Create a blueprint function library called My AI Functions and inside that add a function called SetBTBool. Add a get blackboard pin and hook the target into the set BTBool node so that a new input parameter gets added to it and rename the parameter to AI Controller. Extend the return value and add a set value as bool node and hook its input exec pin to the output exec pin of the set BTBool node. Extend the key name pin and add a make literal name node and hook the value to the Set BTBool node and rename the input parameter to Key Name. Hook the bool value pin to the set BTBool node too.

Now go back to the My\_AIC and extend the true pin of the branch node and add a set BTBool node and extend the AI controller pin and add a self node, set the key name to Can See Target (Enter this string in the input field) and check the bool value. Extend the output exec pin of the set BTBool node and add a retriggerable delay node. Add a reference to the pawn sensing node, extend it and add a get sensing interval node, extend it and add a float \* float node and set the other value to 2 and hook the return value to the duration pin of the retriggerable delay node. Extend the output exec pin of the retriggerable delay node and add a set BTBool node and extend the AIController and add a self node, set the key name to Can see target and uncheck the bool value.

What happens here is that whenever the pawn sees the target the retriggerable delay is activated which means that the delay is delayed so it never reaches the completed pin until the pawn no longer sees the player then it stops firing so the delay is no longer retriggered so the exec flows from completed pin and the value of the Can see target variable is set to false.

If you want to test this add a print node before and after the retriggerable delay node and to the one before set the print string to can see and to the one after set the print string to cannot see.

So now on simulation whenever the pawn can see the player the can see is printed but the moment the player is no longer in line of sight cannot see is printed just once.

Go to BB and add 2 keys called Can See Target and Recently Seen Target of bool type. Go to the Controlling BT and on the sequence node add a decorator - blackboard based condition. On the right-hand window set the blackboard key to Can See Target and set the key query to is not set. Set the notify observer to on result change and observer aborts value to Self. So whenever the Can See Target is false the sequence is activated and keeps running until the Can See Target value changes and at that moment the sequence aborts passing control back to the selector.

Extend the selector node and add another sequence node to the left of the previous sequence node and this is going to be used to follow the player. Add a decorator and set the blackboard key value to can see target, key query to is set and observer aborts value to both. Extend the sequence node and add a move to node. In the BB add a new key of type object called Target. Then change the base class of the key type from object to actor which is important else we can’t use it in the move to node. Come back to the Controlling BT and set the blackboard key value of the move to node to Target.

Now we need to set a value to Target in the My\_AIC. To do this we can either use the function library we created earlier and make a new function for use the same function as created earlier and use enums and switch to modify it to set blackboard entry according to the input type or you could do it in the My\_AIC itself. Anyways the end product must be setting the blackboard key’s Actor value to the pawn sensing target node.

Now on simulation the character chases the player when it can see the player and when it can’t it goes back to patrolling.

Finally we are going to add a feature where the character goes back to the origin when it loses track of the player. For this we use the Recently seen target we made earlier. You just need to create another BB entry called Origin and set the value to origin in the AIC using any of the methods (function library/ inline direct definition/ enum etc.) and then between the 2 sequence nodes extend the selector node and add another sequence node and add a decorator where recently seen target is set. Extend the sequence node and add a wait node to the far left and to the wait node’s right add a move to node and set the move to value to Origin. To the far left add a new task node and this task must set the value of the recently seen target to false.