## Creating AI Behaviour Trees in Kiwi.js

In this article I'm going to show you how to create custom AI Trees using Kiwi.js and the AI Tree Plugin. I won't be going into detail on my game state or how the game objects are affected, I will just be covering the AI Tree and how I designed and created my patroling guard AI.

In my example I wanted to create a simple patroling guard AI, where my characters would patrol back and forth, between two specified points. To make the AI slightly more complex, I wanted another state where the guard shouldn't move and shoot something instead. You can click anywhere on the screen to create barrels for the guards to shoot. Each barrel takes three shots to be destroyed. The AI will check to see if there are any barrels in front of the guard for him to shoot, otherwise he will continue his regular patrol.

### The AI Tree

The AI Tree Plugin contains an AI container, which controls the behaviour of any object it is attached to, in the form of a behaviour tree. A behaviour tree is a directed acyclic graph. This means it's a hierarchy of nodes, and you can't create loops. When you call "update" on the AI object, the tree calls the first node, which performs some calculations and then calls some or none of its children. When a node is done, it returns a signal up the tree, which eventually returns to the AI object. This can create very complex decision-making processes in a very logical manner, somewhat like a flowchart.

The AI object is used by creating and adding a sequence of inner and outer nodes, explained below, to it as children. The AI object calls an update on each of its children every frame.

The AI tree is then attached to our game object and set to update every frame.

There are two types of nodes that can be added to an AI tree: Inner Nodes and Outer Nodes.

### Inner Nodes

The core of any AI Tree is it's inner nodes. Inner nodes are used to control the flow. They are useful for making decisions or running multiple actions in sequence.

There are five types of inner nodes:
- Selector
- Sequencer
- UntilFailure
- UntilSuccess
- UntilTime

Let's look at the two I used in my example, the sequencer and the selector.

Sequencer nodes are used to follow a set sequence of actions in order or to check multiple conditions, before performing an action.

A Sequencer node will step through each of its children in turn every update frame, calling the run function on each of them. The node will return FAILURE immediately if one of its children fails, or will return SUCCESS if it reaches the end of its children, and they all succeeded. If a child returns RUNNING then the node will be set as RUNNING and will continue from that child on the next update.

For the example, I used 2 sequencer nodes. One contains two Action nodes, explained below, which cycle between eachother. The other contains an Action and a Condition Node, also explained below.

Selector nodes are used to find a suitable action from an ordered list. Append Condition Outer Nodes to return without selecting an action.

Selector nodes step through each child, calling the run function on each. However, it is almost the opposite of a sequencer node. It will return SUCCESS as soon as a child succeeds or FAILURE if none of its children succeed. If a child returns RUNNING then the node will be set as RUNNING and will continue from that child on the next update.

In the example I used a selector node to ensure that the guard would not attempt to walk towards a waypoint if they were currently in the process, or capable, of shooting at a barrel.

### Outer Nodes

Outer nodes of our AI Tree are used to specify actions for our game objects, or to check variables within our game for conditions. Therefore, the base AITree Plugin does not provide any outer nodes. I needed to write these myself by creating specific functions to do tasks within my game. I created a new Kiwi Plugin called the BasicPatrolAI Plugin.

Every outer node has a run function which is called by the AI Tree. Within this function you need to set the status variable of the node to either STATUS_SUCCESS, STATUS_RUNNING, or STATUS_FAILURE. This will inform the AI Tree of how the node operated and what step to take next.

Action outer nodes are used to perform a task within our game, or on our game object.

During the run function we need to set our status variable. For action nodes you would usually set it to SUCCESS if the action is now complete, RUNNING if some progress was made but the action is not quite finished yet and FAILURE if something changed and the action no longer can be completed.

For the example I used 2 action nodes, a moveToDestination action and a shoot action. These took the game object and either moved it or changed its properties. Also the shoot action called functions on the game object's state which, in turn, created new game objects, the bullets.

Condition outer nodes are used to check specific values in game to determine an outcome, but don't change anything. Use them before action nodes in a Sequencer Inner Node to determine if the action should occur or not.

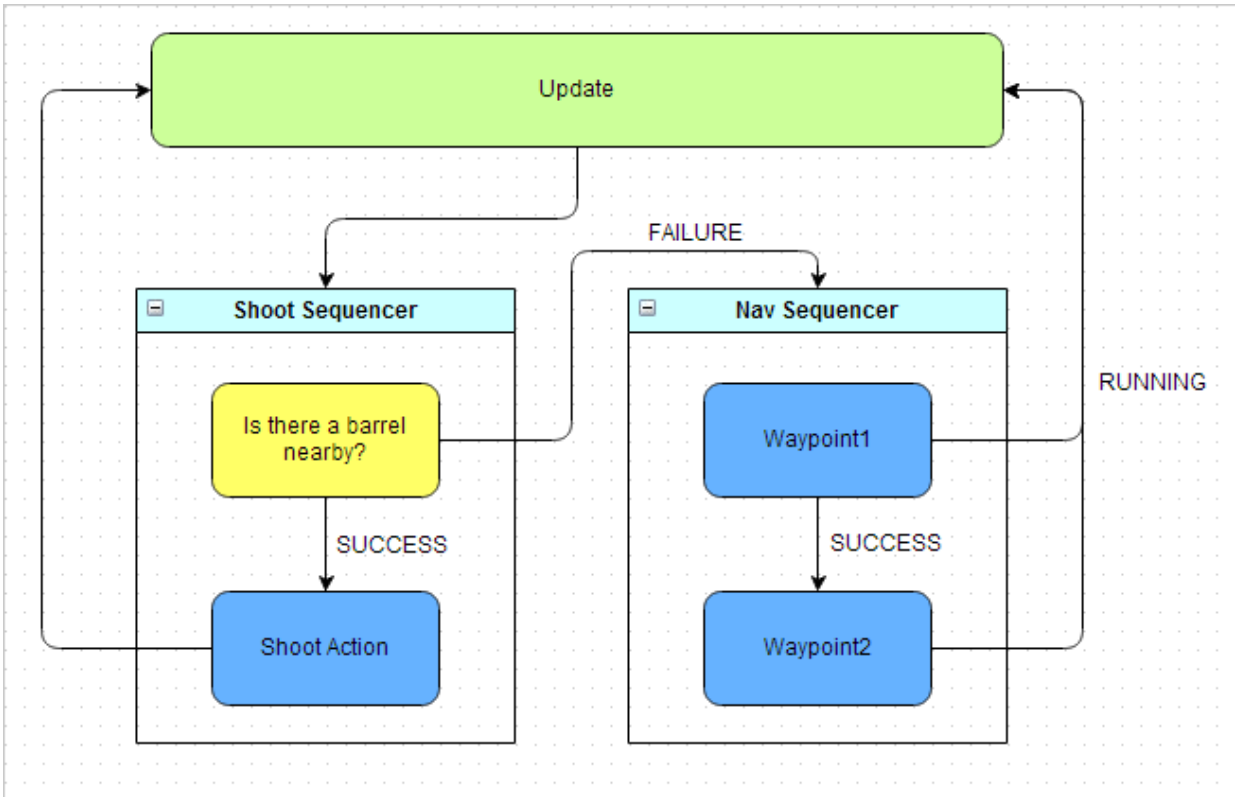
For condition outer nodes, during the run function, change the status variable to SUCCESS if your condition is true, or FAILURE if it is false. Therefore if your condition fails and is within a Sequencer Inner Node then the following actions/conditions will not be run.

In my example I used a condition outer node to check if there was a barrel within a certain range from our guard. This allowed the AI Tree to continue to the shoot action if this condition was true, or continue to the moveToDestination action otherwise.

Now that we have covered some of the basics of how the different types of nodes work, let's take a look at the tree structure I created for my example above.

## My Tree Structure

When I created my tree structure I had to think about what I wanted to happen, in what order, and how changes to the game’s state would affect it. So I laid out a simple decision tree.




The first thing to do was create the 4 outer nodes:

- BarrelDetector ( Condition Node : DetectBarrel )
- ShootAction ( Action Node : ShootAction )
- Waypoint1 ( Action Node : MoveToDestination )
- Waypoint2 ( Action Node : MoveToDestination )

Then I needed to add them to inner nodes to form the structure and order that I wanted. I needed two sequencer nodes, one for the shooting action and one for the patroling action.

- navSequencer
- shootSequencer

I also needed one Selector node to choose which sequencer to process.

- actionSelector

Then to create the structure I simply added the nodes to other nodes as children in the correct order to form this structure:

- actionSelector
    - shootSequncer
        - BarrelDetector
        - ShootAction
    - navSequencer
        - waypoint1
        - waypoint2

So let's run through a quick sequence to see how each part works.
The first thing that happens is the actionSelector node is run, which calls the run function on the shootSequencer first. This then calls run

on the Barrel Detector. If this condition returns SUCCESS then the shootAction is run, which shoots a bullet or updates the timer, and returns SUCCESS. Now, since both of the shootSequencer's children have returned SUCCESS it too returns SUCCESS which triggers the actionSelector to end and the tree is finished, until the next update cycle.

However, if the BarrelDetector node returns FAILURE then the shootSequencer instantly returns FAILURE and the shootAction is never run. Because the shootSequencer returned FAILURE the action selector continues to its next child, the navSequencer. Then it will run the waypoint1 action, which will either return RUNNING if the guard hasn't reached the waypoint yet, or SUCCESS if he has. Once it returns SUCCESS the sequncer will call run on its next child, the waypoint2 action. On subsequent updates which reach the navSequencer, the sequence will skip over waypoint1 as long as waypoint2 returns RUNNING. If it returns SUCCESS it will revert back to waypoint1.

And that's it! Whilst AI is complex and difficult to implement, you can easily break it down into smaller conditions and actions and order it into a nice behaviour tree. Using the AI Tree Plugin and Kiwi has certainly made making AI for my HTML5 games a much simpler task.

You can find more information on behaviour trees by reading this great guide http://www.altdevblogaday.com/2011/02/24/introduction-to-behavior-trees/

Also learn about and download the exciting new javascript engine Kiwi.js developed by Instinct Entertainment: www.kiwijs.org