Exercise – Behaviour Trees - Part 1

The goal of this tutorial is to design and better understand Behaviour Trees. This tutorial makes use of the material from previous tutorials. In the previous tutorial you implemented Decision Trees to make the decisions for your agents. In this tutorial you will instead spend some time thinking about and designing Behaviour Trees on paper, and then implement a Behaviour Tree in code to replace your Decision Tree. ***You will need a pen / pencil and paper to begin with.***

Exercise 1:

Behaviour Trees are a structure controlled by a few very simple classes that are used to construct the complex logic.

Using just Sequence and Selector composites we can arrange custom Actions and Conditions to create almost any sort of A.I. decision system for our game's NPCs and even for our players. Player characters for games can be setup just like an A.I. controlled NPC, making use of a Behaviour Tree, but the decisions are derived from the player's input.

In pairs, try and design on paper simple behaviour trees for the following A.I. controlled agents (be creative, but consider all Actions and Conditions you might need):

- An ant gathering food for its hive.
- A soccer goal keeper.
- A pet dog or cat.
- A bot in a shooter game.

Exercise 2:

You are now to implement a Behaviour Tree in code to replace the Decision Tree from the last tutorial.

You will need to implement a Behaviour class, a Composite class, a Sequence class and a Selector class:

```
//the interface class for all behavior nodes
class IBehaviour

    func execute(agent) = 0 //abstract func
```

```
//base class for sequence and selector nodes
class Composite : IBehaviour
    list childBehaviours //the behaviors to select/sequence

    func execute(agent) = 0
```

```
//'AND' node for running a list of behaviours consecutively until one fails
class Sequence : Composite

    func execute(agent)
        foreach child in childBehaviours
            if child.execute(agent) == Failure
                return Failure
        return Success
```

```
//'OR' node for running a list of behaviours consecutively until one succeeds
class Selector : Composite

    func execute(agent)
        foreach child in childBehaviours
            if child.execute(agent) == Success
                return Success
        return Failure
```

```
//agent object with a behavior tree
class Agent
    IBehaviour behaviourTree //root node of tree

    func update()
        behaviourTree.execute(this)
```

With the base classes for a Behaviour Tree created, try to think of the Actions and Conditions required to replace your Decision Tree. You will need Actions for Flee, Wander and setting the timer. You will need conditions for checking distance between the agents and checking if the timer is active.

Before implementing your entire tree in code use pen / pencil and paper to design the logic of your tree to check if it is correct.

References:

- An excellent, in-depth explanation of behaviour trees can be found on gamasutra here.
- AltDevBlogADay also have a video tutorial on behaviour trees here.