Exercise – Behaviour Trees

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 game objects.

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, i.e. Was X pressed? And are we jumping? Do air attack.

Try and design on paper simple behaviour trees for the following A.I. controlled agents   
(be creative, but try to 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:

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

**YES**

**YES**

Target within 200?

Wander

Target within 50?

Attack

Move to Target

**NO**

**NO**

You should already have a Behaviour class that we can use as the base class for our Behaviour Tree:

class Behaviour

func execute() = 0

To be able to create the tree structure you will need a Composite class, and then derive a Sequence class and a Selector class:

// OR node for running a list of behaviours consecutively until one succeeds

class Selector : Composite

func execute()

for each child in children

if child.execute() == Success

return Success

return Failure

// AND node for running a list of behaviours consecutively until one fails

class Sequence : Composite

func execute()

for each child in children

if child.execute() == Failure

return Failure

return Success

// abstract base class for sequence and selector nodes

class Composite : Behaviour

list children //the behaviours to select/sequence

func execute() = 0

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 Attack, Wander and Move To Target. You will need conditions for checking distance to the target.

Conditions and Actions are simply classes that also derive from the base Behaviour class so that they can be added to Sequences and Selectors.

***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.***

When drawing a Behaviour Tree, Sequences are typically draw as either of the following (in this example green circles could represent Condition leaf nodes and yellow could represent Action leaf nodes):

Selectors are then often drawn like the following:

**?**

**?**