Exercise – Decision Trees

The goal of this exercise is to implement a basic decision tree structure. We will be creating a decision behaviour and implementing individual decisions.

Setup Decisions:

We first need to create a decision class that we will use as a base class:

//base class for decisions

class Decision

func makeDecision( gameObject, deltaTime ) = 0

To be able to implement a very basic decision tree handling Yes / No we would need to derive a decision that could make use of conditions, we could be bool or conditional test, or use a Condition object:

//a decision tree node

class ABDecision : Decision

Decision A

Decision B

Condition condition

func makeDecision( gameObject, deltaTime )

if condition is true then

A.makeDecision( gameObject, deltaTime )

else

B.makeDecision( gameObject, deltaTime )

For our previously created GameObject’s to be able to make use of decisions we would need to implement a Decision Behaviour, which may use a simple execute() method like what follows:

virtual bool execute(GameObject\* gameObject, float deltaTime) {

m\_decision->makeDecision( gameObject, deltaTime );

}

We would now be able to create an implement custom decisions that perform certain actions.

For example, we might create a Seek decision that simply uses makeDecision() to implement a basic steering behaviour to move the game object towards a target.

Exercise:

With your basic Decision classes set up, try to implement the following behaviour using a Decision Tree, using whatever conditions you require which may involve creating new derived decisions (note the ‘Attack’ decision can be whatever you like, and the distances used may need to change):

**NO**

**NO**

**YES**

**YES**

**YES**

**YES**

**YES**

Target within 50?

Move to Target

Attack

Wander

Target within 200?