COMP 4303:

Artificial Intelligence in Games

Behaviour Trees

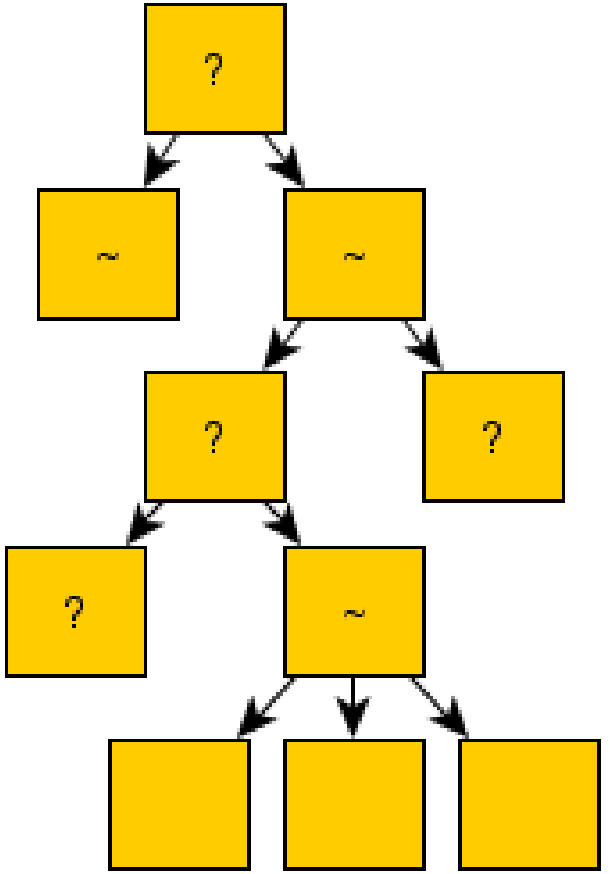
Jay Henderson (jayhend@mun.ca)

Behaviour Trees

What is a Behaviour Tree?

A **Behaviour Tree (BT)** is a hierarchical task switching structure in the form of a tree

Similar to an FSM, a **BT** is made up of nodes, but *each node is not necessarily a state*



Behaviour Tree Nodes

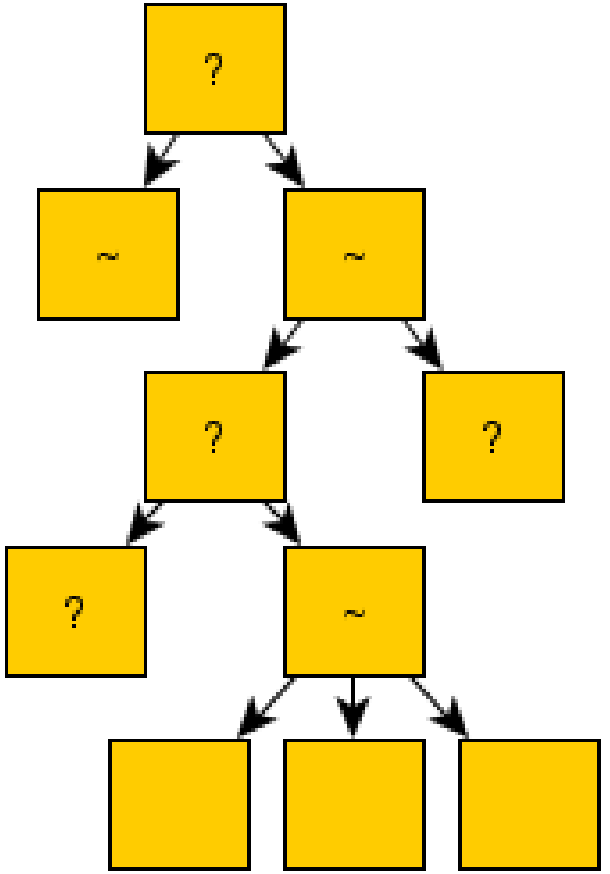
Similar to state nodes,
BT nodes share common functionality

Has one method:

Run

Returns 1 of 3 statuses:

Success
Failure
Running



Example: pathfinding + walking

Let's say we are executing some sort of pathfinding algorithm and we are instructing a character to walk along the path

If we have a walk node:

While the character is walking:

    return running

If the character cannot reach the target location or gets stuck while walking:

    return failure

If the character reaches the destination of the path:

    return success

abstract class Node

Create an "abstract class" in js:

In our constructor,

throw an error if trying to call the constructor on the abstract class (Node)

throw an error if run is undefined (i.e. not implemented)

Create an enum-like object for the 3 statuses:

Status { Success, Failure, Running }

Since javascript does not have enums:

We can use Object.freeze() to make the object immutable

We can use Symbol() for named value to ensure the value is unique

Behaviour Tree Structure

Leaf nodes are nodes that have no children (green)

These contain actual behaviours or actions that control our AI

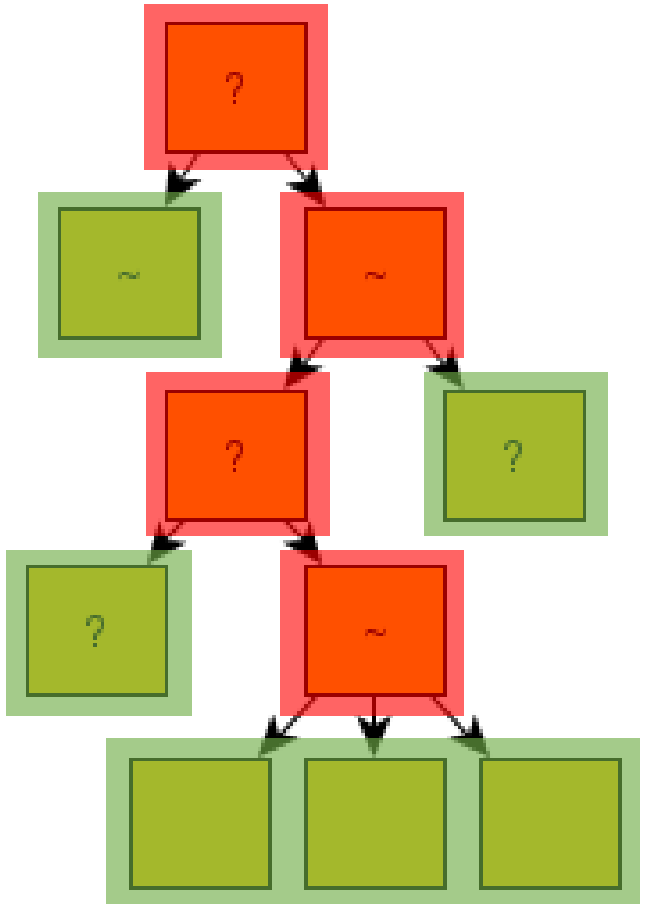
Composite nodes have children (red)

These dictate how to traverse the tree i.e. where to go next

Decorator nodes have one child

used to transform the return of the child

e.g. inverting



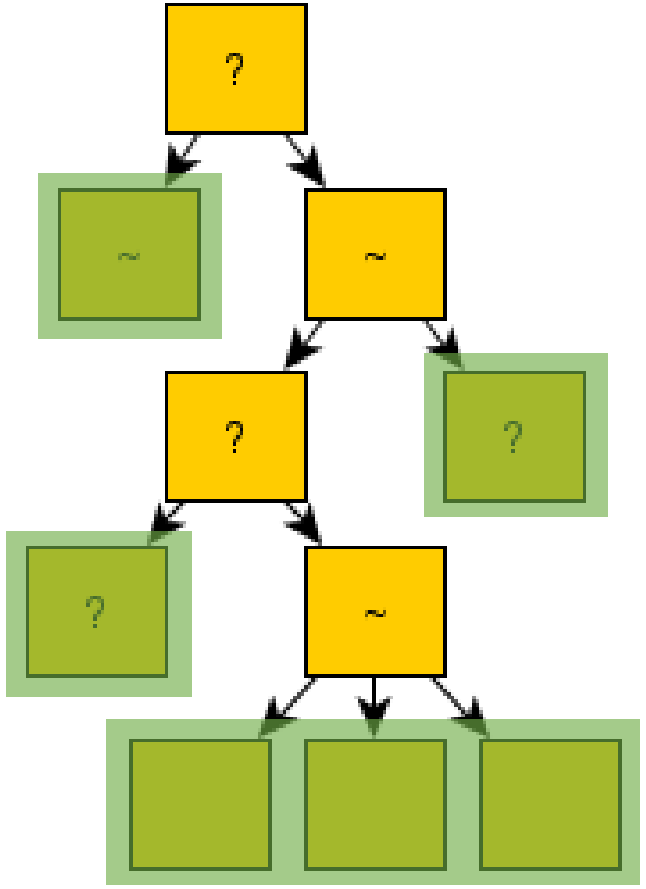
Types of Leaf nodes

Condition

- Tests something
  e.g. does the character have remaining health?
- Can only return *success* or *failure*
- Requires it's own implementation

Action

- Alters the state of the game
  e.g. raise health, move to position (x, y)
- Requires it's own implementation

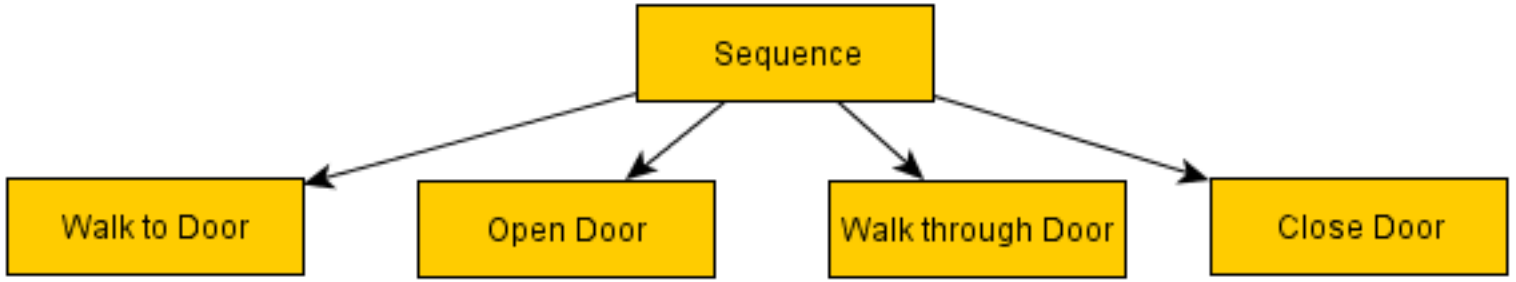


Types of Composite nodes: Sequence (→)

Sequence

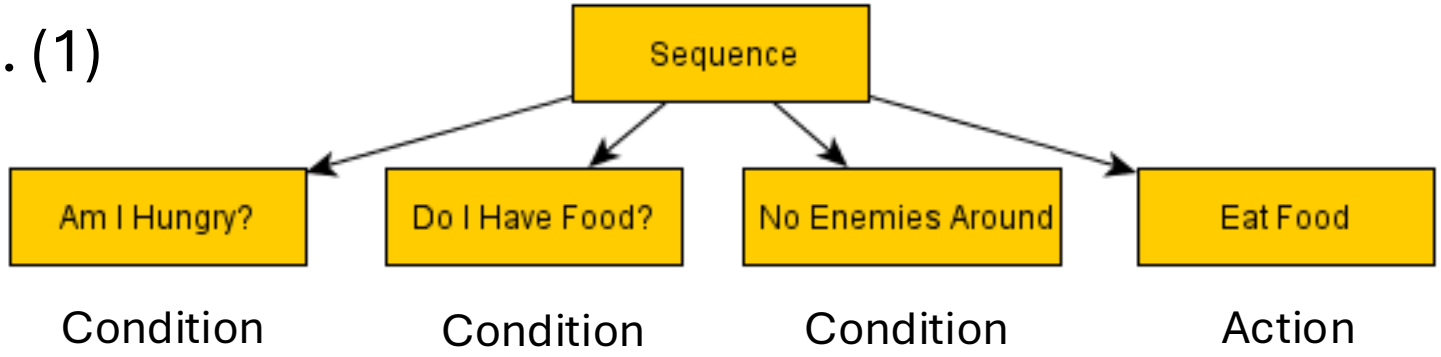
- Visit each child node in order

  e.g. start with the first child node, if that returns *success* call the second child node
- If ANY child returns *failure*, the sequence node returns *failure*
- If ANY child returns *running*, the sequence node has not failed or succeeded and returns *running* so we can revisit it next time we traverse the tree
- **success = ALL children to return *success***



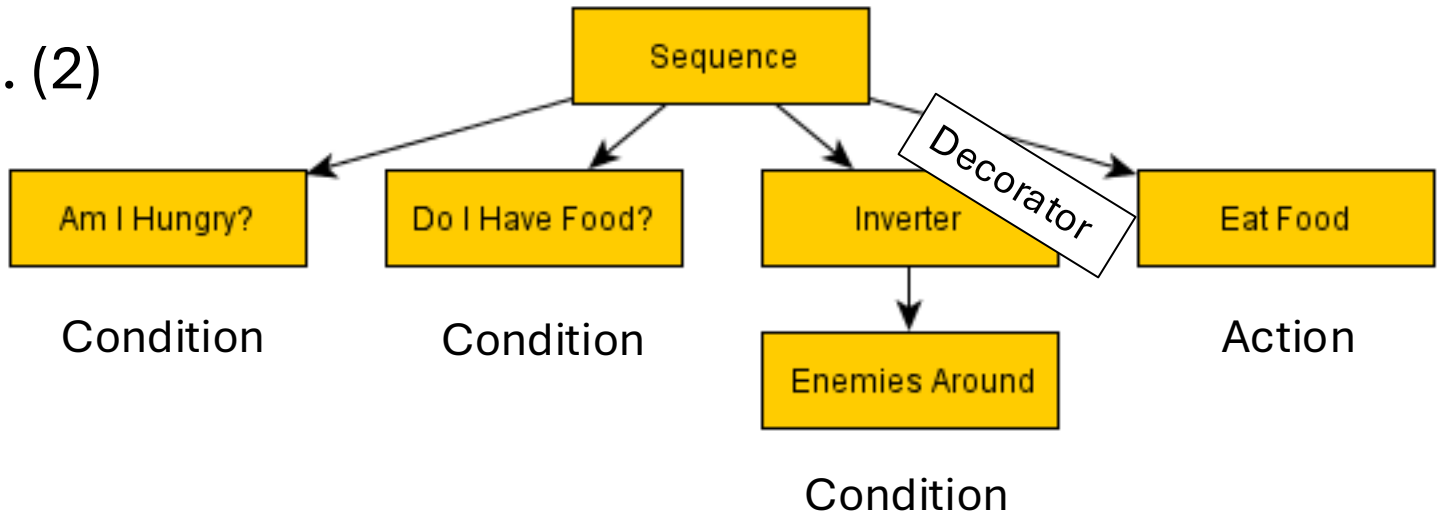
Types of Composite nodes: Sequence (→)

e.g. (1)



Types of Composite nodes: Sequence (→)

e.g. (2)



Sequence BTNode class

Instance variables:

children – an array to hold all the node's children

Methods:

run()

- Run each child node
- If a child is running, this node is running
- If one child fails, this node fails
- If all children are successful, this node is successful

Types of Composite nodes: Selector (?)

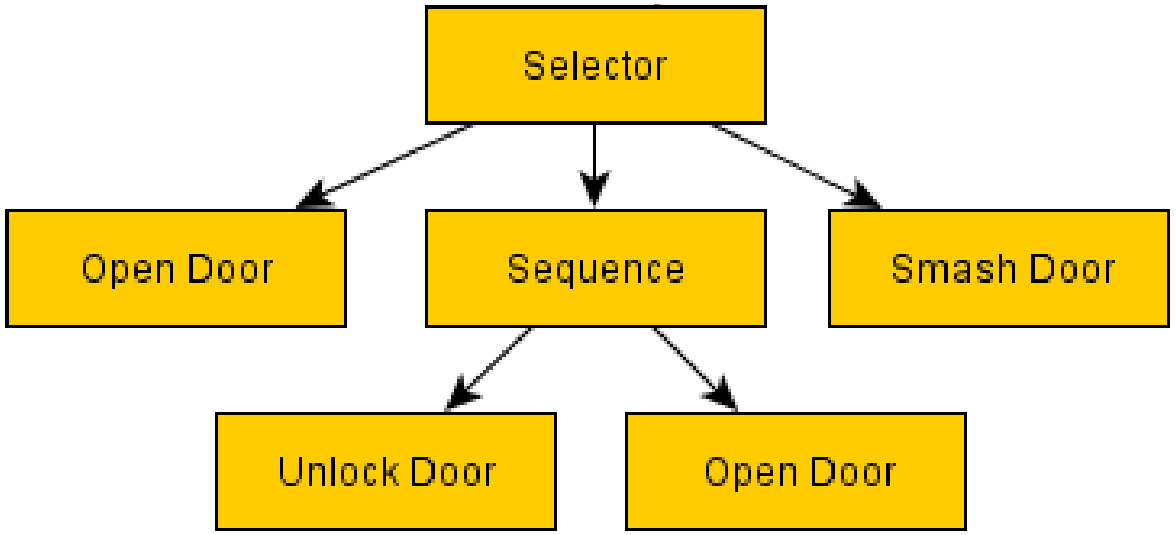
Selector

- Visit each child node in order until a success is reached
  e.g. process first child, if failure, process second child, etc.
- If ALL children return *failure*, the selector node returns *failure*
- **success = at LEAST one child returns *success***

- Running status for a selector node depends on your requirements
  For simplicity, here we can return running if a running node is hit

Types of Composite nodes: Selector (?)

e.g. Opening a door



Selector BTNode class

Instance variables:

children – an array to hold all the node's children

Methods:

run()

- Run each child node
- If one child is successful, this node is successful
- If one child is running, this node is running
- If all children fail, this node fails

Root node

Any entity which applies a behaviour tree will do so from a root node

The root node is where the BT starts from and can be of any type of node

Each BT can only have one root node

Benefits of Behaviour Trees

Modular

Few dependencies between components

Conform to a basic interface

Reactivity

Starts at a fixed rate so BTs can react in real time to events

Built in Hierarchical Structure

Model complex, compound behaviours with less code

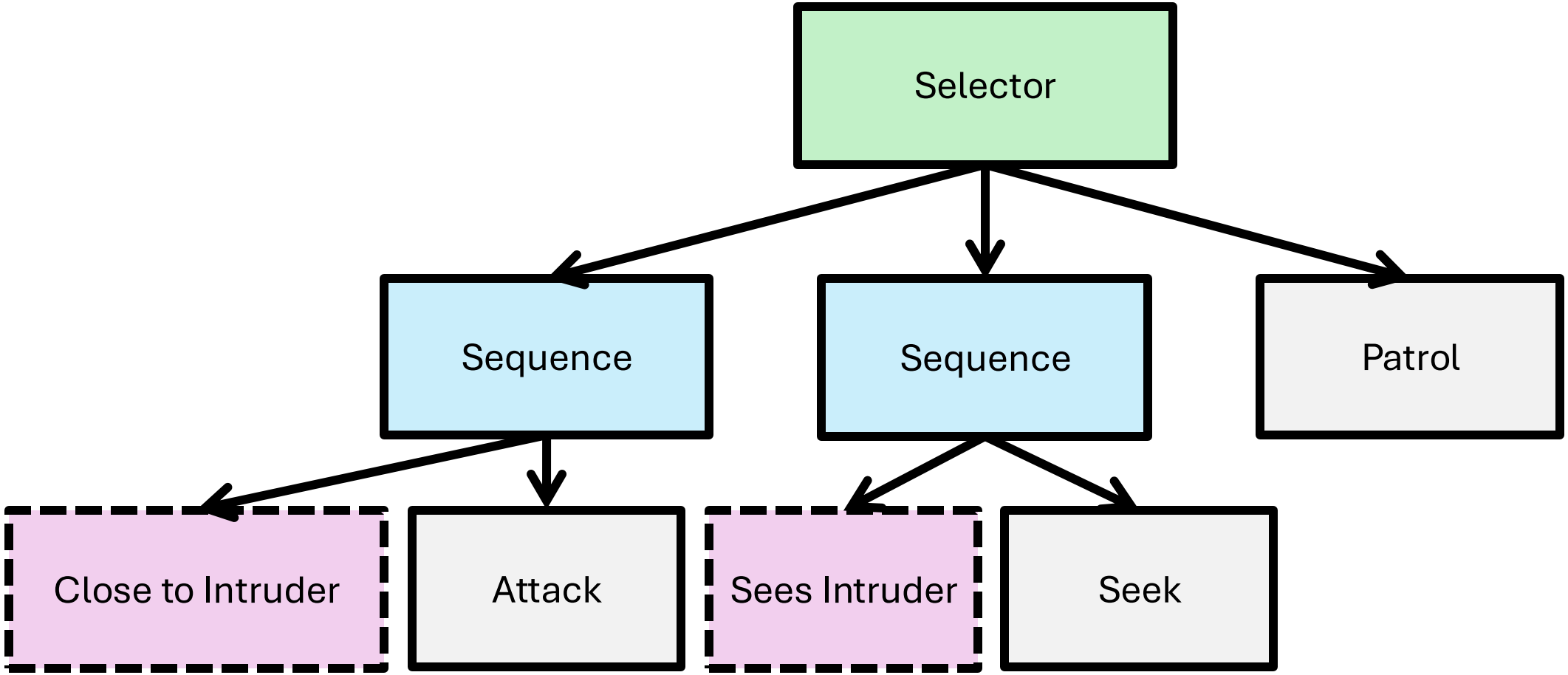
Example: Patrolling Guard

A guard is patrolling an area by wandering

If the guard sees an intruder, the guard will go to the intruder (seek)

If the guard gets close enough, the guard will attack (arrive)

Example: Patrolling Guard



InRangeToIntruder class (Condition node)

Let's create a general purpose BT node "InRangeToIntruder" that we can use to check if our guard "sees" an intruder or is "close to" an intruder

Instance variables:

guard

player

radius

Method:

run() – tests to see if our guard is within a certain radius to our player if so return success, otherwise return failure

Attack (Action node)

This class applies the attack behaviour

Instance variables:

guard

player

Methods:

run() – applies attack and sets colour to red, returns success

Seek (Action node)

This class applies the seek behaviour

Instance variables:

guard

player

Methods:

run() – applies seek and sets colour to yellow, returns success

Patrol (Action node)

This class applies the patrol behaviour

Instance variables:

guard

Methods:

run() – applies wander and sets colour to lightgreen, returns success