# CPSC 436D
# Video Game Programming

***Basic Gameplay***



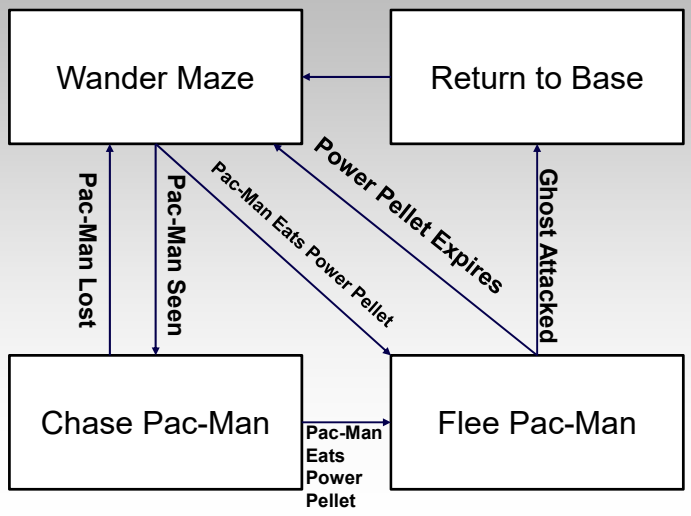

## Basic gameplay

- AKA Enemy Logic
- How do enemies “choose” their actions?

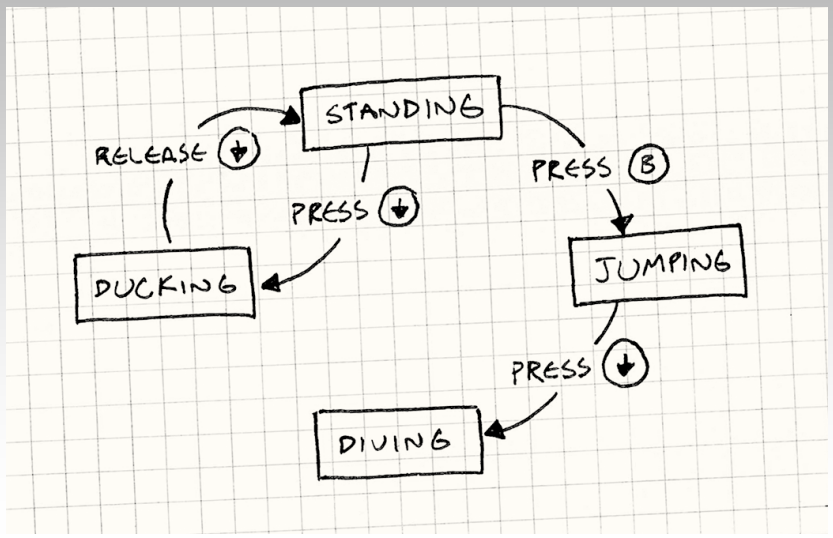
## FSM Example: Pac-Man Ghosts




## FSM Example: Pac-Man Ghosts




# Finite State Machines: States + Transitions






# FSM Pseudo-Code

```
// start
if (!walking && wantToWalk)
{
    PlayAnim(StartAnim);
    walking = true;
}

// walk loop
if (IsPlaying(StartAnim) && IsAtEndOfAnim())
{
    PlayAnim(WalkLoopAnim);
}

// stop
if (walking && !wantToWalk)
{
    PlayAnim(StopAnim);
    walking = false;
}
```

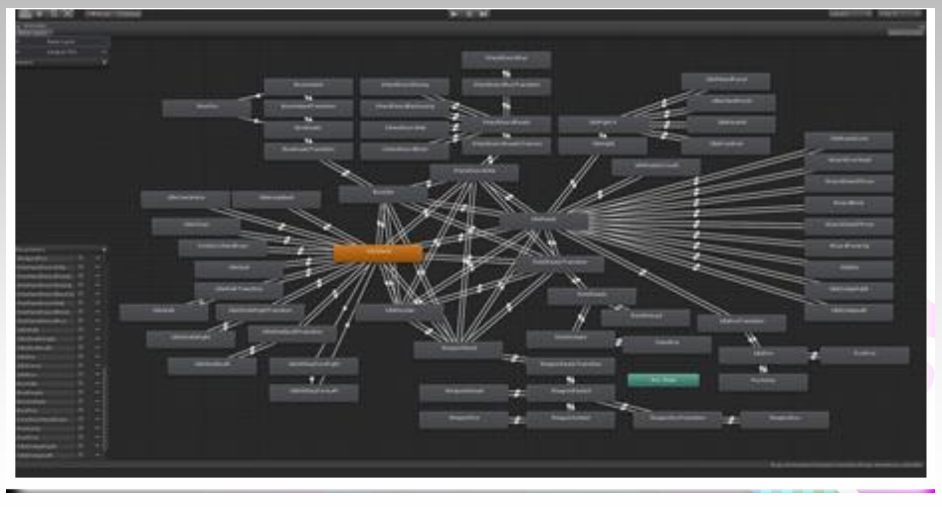
From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

## FSMs:

- ***Each frame:***
  - Something (the player, an enemy) does something in its state
  - It checks if it needs to transition to a new state
    - *If so, it does so for the next iteration*
    - *If not, it stays in the same state*
- ***Applications***
  - Managing input
  - Managing player state
  - Simple AI for entities/objects/monsters etc.



## FSMs: States + Transitions



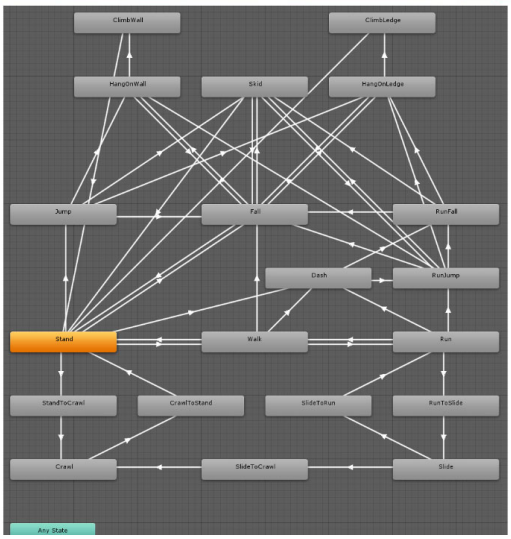
From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf

## FSMs: States + Transitions

```
if (speed > 3.0f)
{
    PlayAnim(RunAnim);
}
else if (speed > 0.0f)
{
    PlayAnim(WalkAnim);
}
else
{
    PlayAnim(IdleAnim);
}
```

From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf © Alla Sheffer

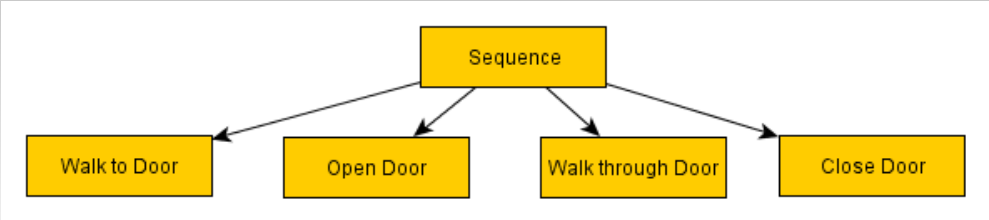
## FSMs: Failure to Scale




- *No way to do long-term planning*
- *No way to ask "How do I get here from there?"*
- *No way to reason about long-term goals*
- *FSMs can get large and hard to follow*
- *Can't generalize for larger games*

From http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf © Alla Sheffer

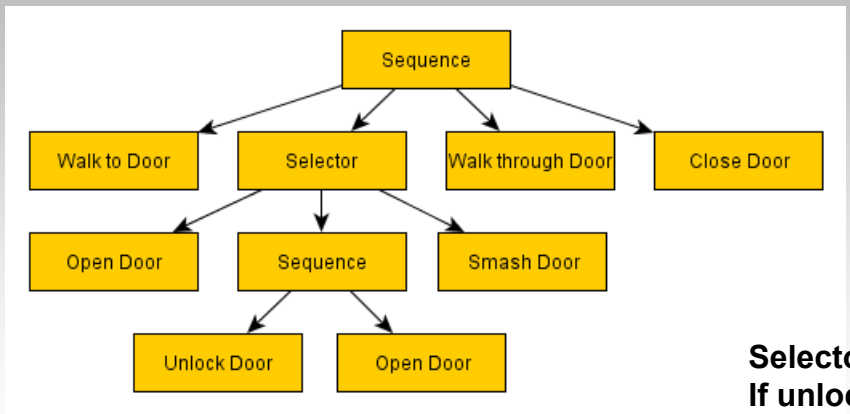
## Schematic examples




https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php



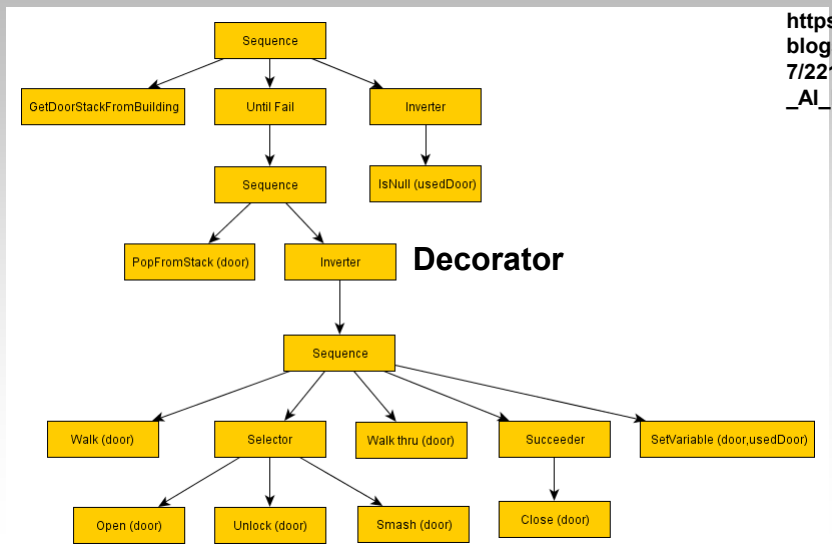
## Schematic examples




**Selector:**
**If unlocked,…**
**Or if have key,…**
**else**

https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

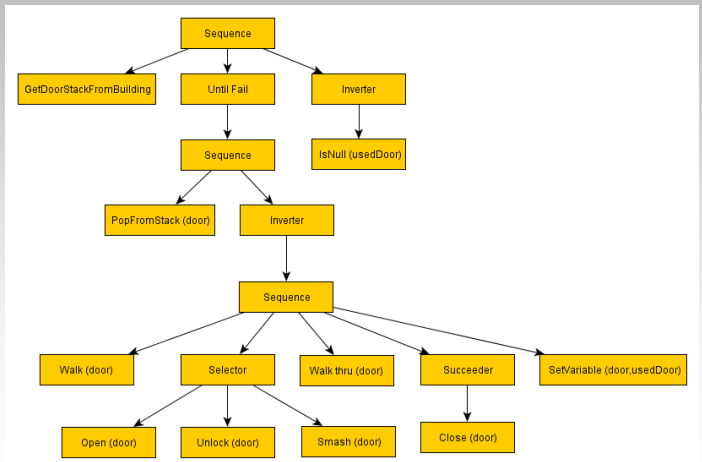
# And more complex…

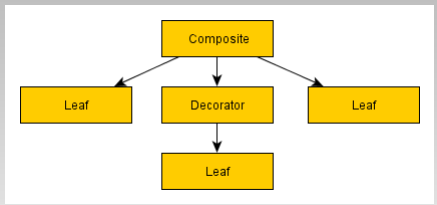



https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

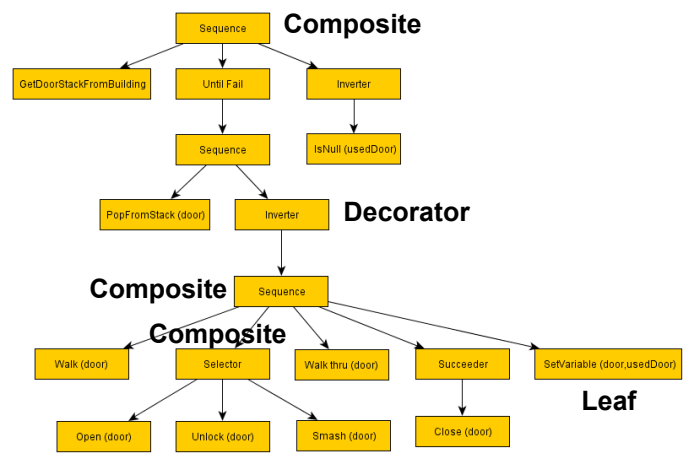


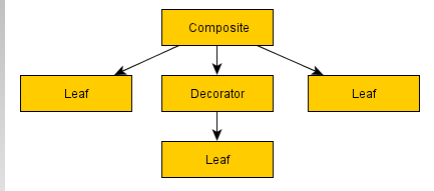
# Types



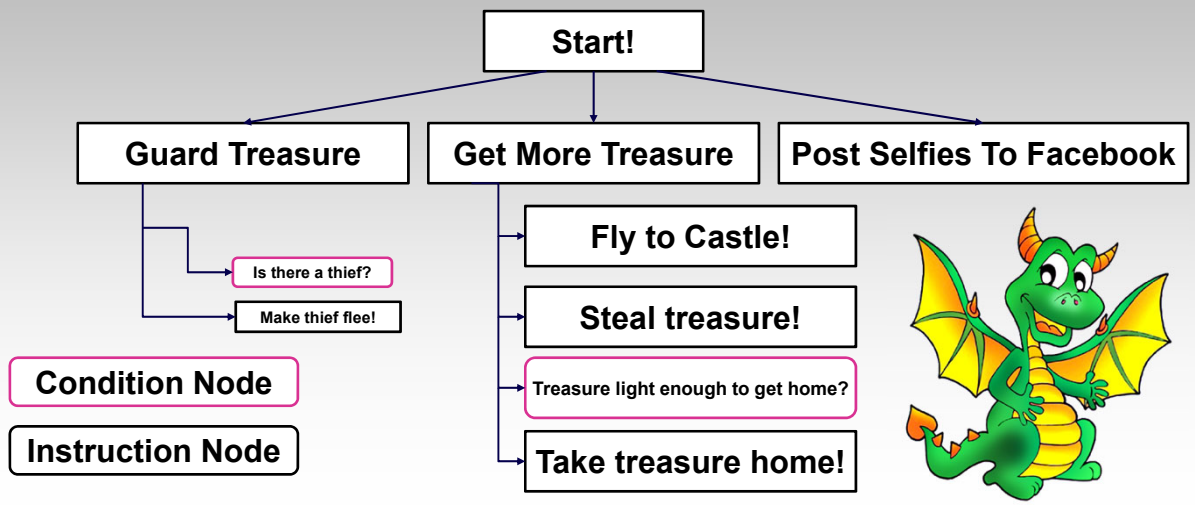
# Types




© Alla Sheffer

# Behaviour Trees: How To Simulate Your Dragon



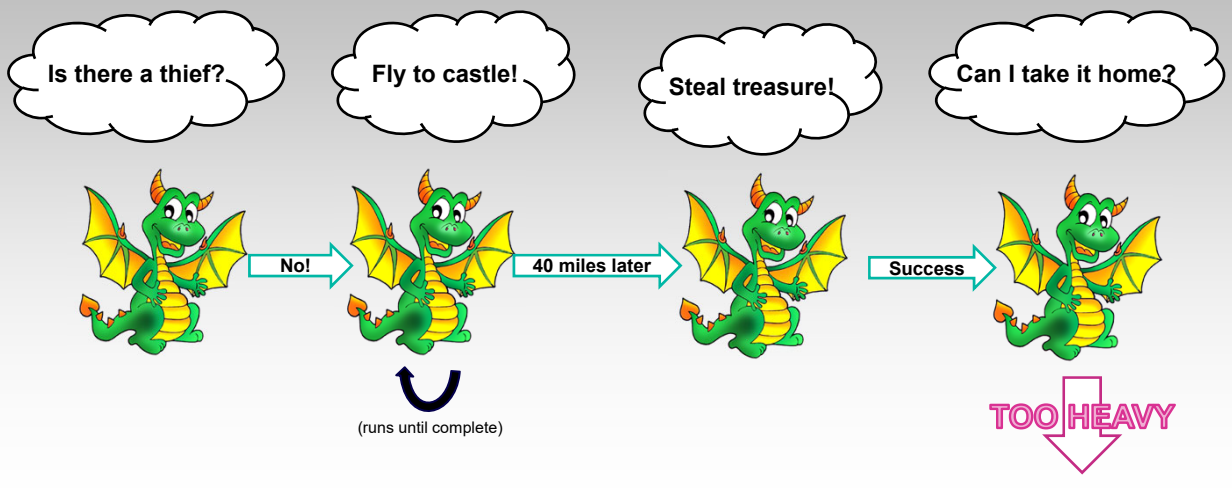

© Alla Sheffer

## Behaviour Trees

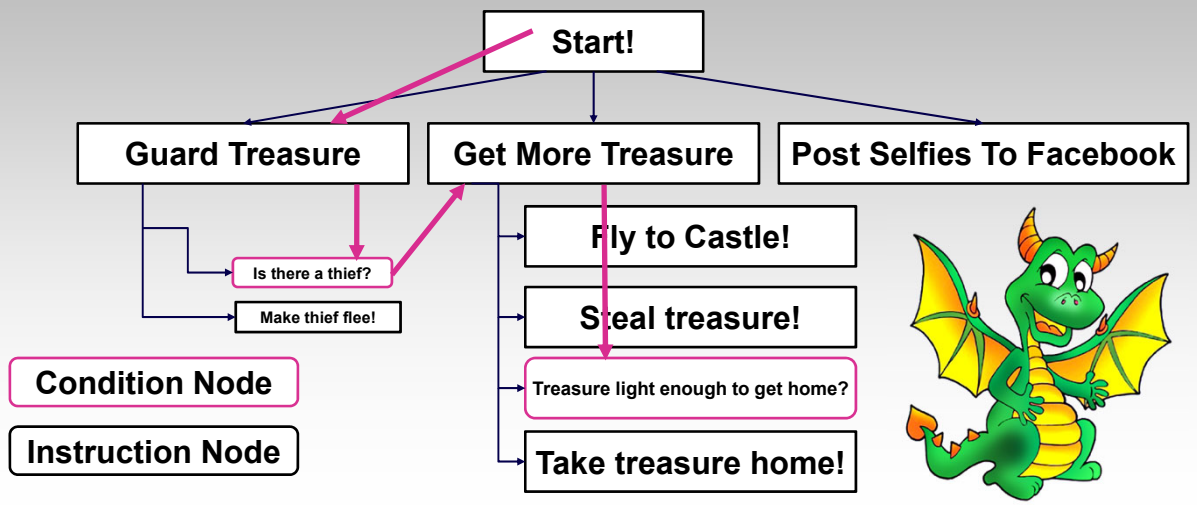
- ***Each frame:***
  - Visit a node
  - See if any **higher priority** nodes now run
    - *If so, execute them instead*
  - See if my currently running node fails
    - *If so, return to the root of the behaviour tree! Start again!*
  - See if the currently running node is done
    - *If so, run the **lower priority** node in the current branch of the tree*



## Start!



# Behaviour Trees: How To Simulate Your Dragon



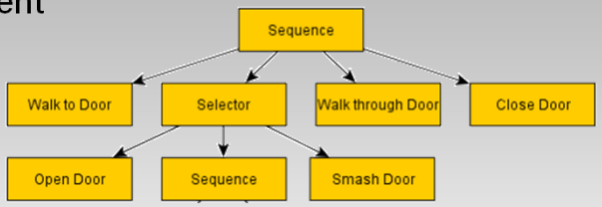



# Behaviour Trees are Modular!

- Can re-use behaviours for different purposes
- Can implement a behaviour as a smaller FSM
- Can be data-driven (loaded from a file, not hard coded)
- Can easily be constructed by non-programmers
- Can be used for *goal based programming*

## Behaviour Trees

- flow of decision making of an AI agent
- tree structured
- ***Each frame:***
- Visit nodes from root to leaves
  - *depth-first order*
  - *check currently running node*
    - succeeds or fails:
      - return to parent node and evaluate its Success/Failure
      - the parent may call new branches in sequence or return Success/Failure
    - continues running: recursively return Running till root (usually)

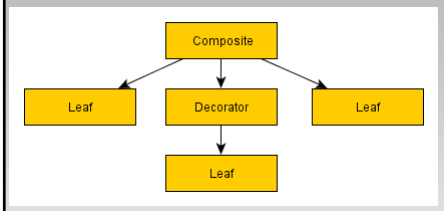


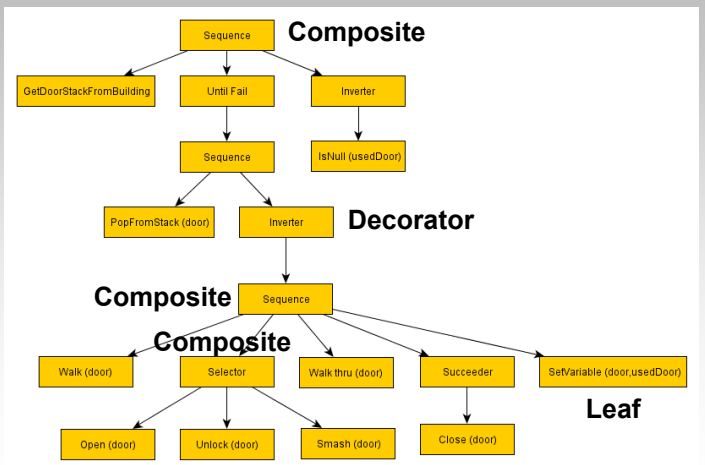



## Behaviour Tree Elements

- Leaves: commands that control AI entity
  - upon tick, return: Success, Failure, or Running
- *Branches: utility nodes that control tree traversal*
  - loop through leaves: first to last or random
  - inverter: flip Failure <-> Success
  - Produce command sequence best suited to situation
- Can be extremely deep
  - nodes calling sub-trees of reusable functions
  - libraries of behaviours chained together

## Types

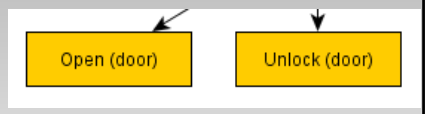


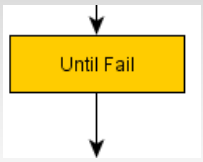



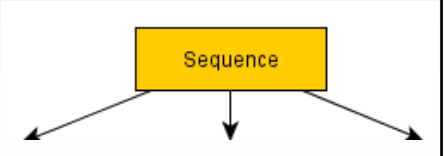



## Behaviour Tree Elements

- Leaf node
  - *Custom function, does actual work*
  - *Returns Running/Success/Failure*
- Decorator node
  - *has one child*
  - *Passes on Running/Success/Failure from child*
  - *may invert Success/Failure*
- Composite node
  - *has >= 1 children*
  - *returns 'Running' until children stopped running*

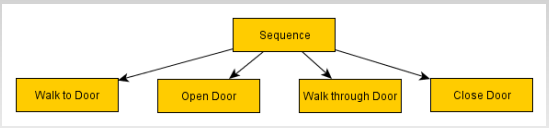


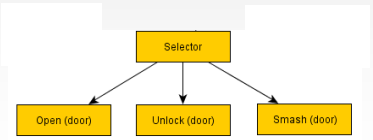



## Useful Composites

- Sequence
  - *execute **all** children in order*
  - *Success if all children succeed ( = AND)*
- Selector
  - *execute **all** children in order*
  - *return Success if any child succeeded ( = OR)*
- Random Selectors / Sequences
  - *Randomized order of above composites*

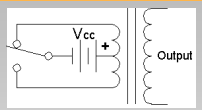





## Useful Decorators

- Inverter
  - *Negates success/failure*
- Succeeder
  - *always returns success*
- Repeater
  - *Repeat child N times*
- Repeat Until Fail
  - *Repeat until child fails*




**return "Success";**


# Leaf Nodes

***Functionality***

- ***init(…)***
  - *Called by parent to initialize*
  - *Sets state to Running*
  - *Not called again before returning Success/Failure*
- ***process()***
  - *Called every frame/tick the node is running*
  - *Does internal processing, interacts with the world*
  - *Returns Running/Success/Failure*

- Example: Walk to goal location
  - *Sets goal position for path finding*
  - *Computes shortest path*
  - *Sets character velocity*
  - *Returns*
    *- success: Reached destination*
    *- failure: No path found*
    *- running: En route*