CSc 165
Computer Game Architecture

14 - Artificial Intelligence & Non-Player Characters (NPCs)

Overview

- Non-player characters (NPCs)
- Managing NPCs / networking
- Managing NPCs / AI
- AI – Finite State Machines
- AI – Behavior Trees
- AI – Particle Systems
- Behavior Trees in TAGE

Non-Player Characters (NPCs)

Types:

- hostile antagonists (“MOB”, or “creep”)
- allies (or partially controlled by player)
- bystanders / decoration
- swarm (“particle system”)

Control:

- typically by an “NPC controller”
- controller can be on server, or on a client
- actions can be “dumb”, or use complex AI
- swarm can sometimes be controlled as one NPC

NPC controller

Approaches:

- controller resides on network server
- controller resides on separate server
- controller resides on first client
- control is distributed across clients

Server-based synchronization example:

- server sends updates frequently
- each client sends messages to server as they interact with an NPC

NPC AI (Artificial Intelligence)

Purpose of AI:

- attacking
- evading
- following / chasing
- patrolling / guarding
- path-finding
- learning / adapting to player behavior
- evolving strategies over time
- providing realism
- tuning playability – ease vs. difficulty (tradeoff)

NPC AI (Artificial Intelligence)

Techniques (in decreasing order of commonality):

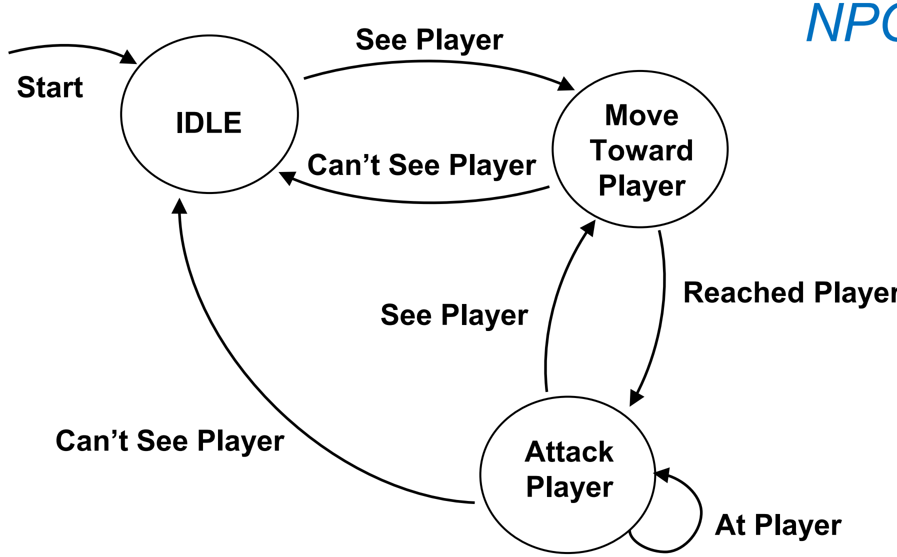
- finite state machines (FSM)
- behavior trees
- search algorithms (greedy, A*, Dijkstra, ACO)
- swarm intelligence (“boids”, PSO, ACO, etc.)
- neural networks *(growing in popularity with deep learning!)*
- rule-based “expert” systems
- genetic algorithms
- *etc…*

Can occupy as much as 50% of the game update!

Finite State Machines (FSM)

example:

Controller stores state information about each NPC and the game



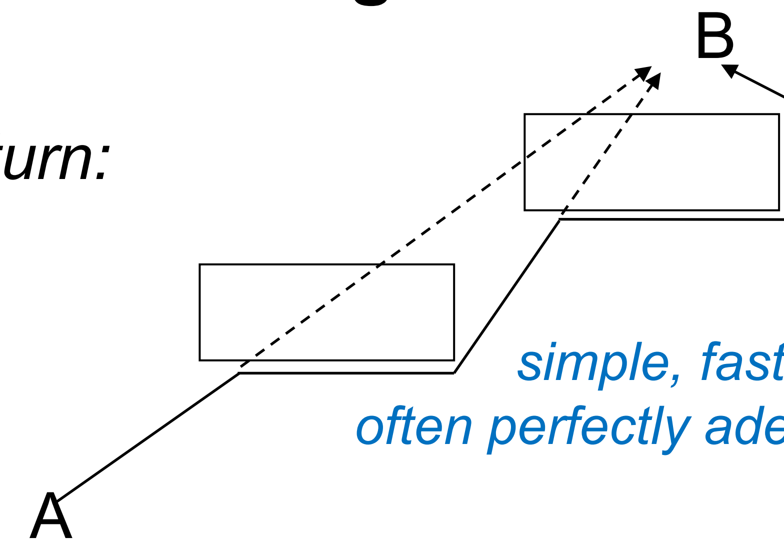
Can be implemented with Strategy design pattern

Flexible, but usually ad-hoc, i.e., specific to that game

Path-Finding algorithms

- Crash-and-Turn
- Dijkstra’s Algorithm
- A* search
- D* search
- Sample algorithm (for mazes)

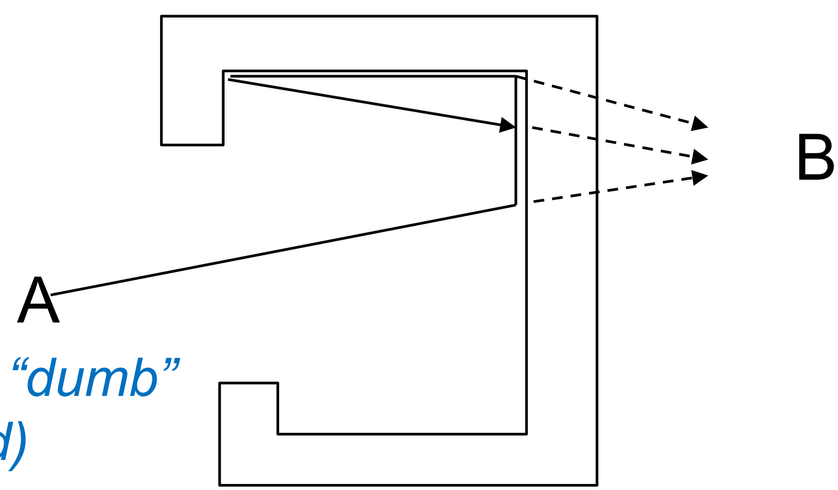
Search / Path Finding



A "greedy" method – *crash & turn:*

```
compute line of sight to target;
while (not at target)
{ if (not blocked)
    { move along line of sight
    }
    else
    { turn facing parallel to blocker
      while (adjacent to blocker)
      { move forward
      }
      compute line of sight to target
    }
}
```

simple, fast, and often perfectly adequate



but can fail (might be ok if a "dumb" NPC is preferred)

Simulated flocks / swarms

- “swarm” = large group of coordinated NPCs
- difficult to coordinate so many separate AIs
- there can be performance issues
- one solution: treat swarm as a single entity, without worrying about synchronizing each individual NPC across all clients
- examples:
    - flocks of birds (“boids”)
    - schools of fish
    - crowds of people
    - stampede of wildebeests
    - colony of ants (although often modeled by ACO)

“BOIDS”

A model for “Artificial Life”

Craig Reynolds, SIGGRAPH 1987

Simple rules, leading to complex *emergent behavior*

Each “boid” has a few simple attributes:

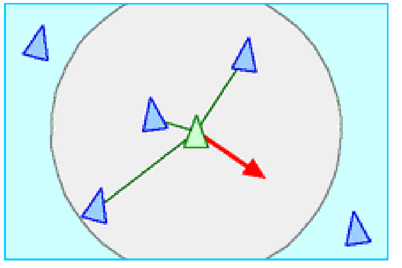
- Speed and heading (velocity)
- Position

Every “boid” runs the same, simple program.

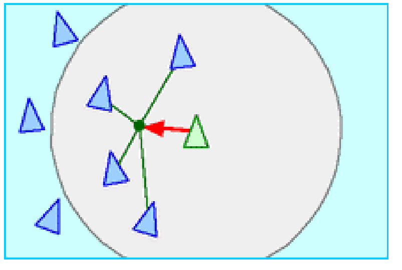
Good for modeling a flock of birds.

Basic Boid Rules

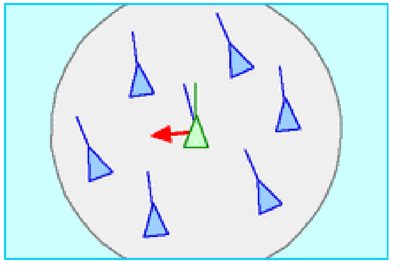
separation



cohesion



alignment



Implementing Boid Rules

```
initializeBoids();

update (time)
{  Vector v1, v2, v3;

   for (each boid b)
   { v1 = getCohesionVector(b);
     v2 = getSeparationVector(b);
     v3 = getAlignmentVector(b);

     // vector addition
     b.velocity = b.velocity + v1 + v2 + v3;

     // point + vector → new point
     b.position = b.position + b.velocity;
   }
}
```

Boid
Vector velocity Point position

Conrad Parker, http://www.kfish.org/boids/

Implementing Boid Rules (continued)

```
Vector getCohesionVector(Boid b)
{ //calculate flock center
  Point center = 0;
  for (each boid n)
  { center = center +  n.position ;
  }
  center = center.divideBy(numBoids);
  //find a vector that moves a "little bit"
  //(e.g. 1%) toward center
  Vector change = (center - b.position)/100;
  return change;
}
```

```
Vector getSeparationVector(Boid b)
{ Vector change = 0;
  for (each boid n)
  { dist = n.position - b.position;
    if ( abs (dist) < MIN_DIST)
    {   change = change - dist;
    }
  }
  return change;
}
```

```
Vector getAlignmentVector(Boid b)
{ //find flock average direction vector
  Vector avgDir = 0;
  for (each boid n)
  { avgDir = avgDir + (n.velocity);
  }
  avgDir = avgDir / numBoids ;
  //return a small fraction of the diff
  //between this boid & avg
  Vector change = (avgDir-b.velocity)/8;
  return change;
}
```

Rule-Based Systems

Example logic:

	<IF>	<THEN>
1.	next to enemy	fight
2.	next to enemy ^ strong	chase
3.	next to enemy	flee
4.	command received	execute command
5.	others fighting ^ have gun	shoot at enemy
6.	-else-	pace back and forth

Based on Core Techniques and Algorithms in Game Programming, Daniel Sanchez-Crespo Dalmau

Methods inspired by nature

Genetic Algorithms:

- behavior encoded as a “chromosome”
- NPCs with successful behavior become parents
- child chromosomes become new behavior
- behavior evolves through natural selection

Neural Networks:

- behavior modeled as a network of “neurons”
- desired behavior used for training
- network weights adjusted over time by backpropagation
- behavior learns by example

Behavior Trees

Two types:

- Systems / Software engineering
- Games (as used in Halo, Spore, etc.) *(this is what we are doing)*

Advantage over FSMs: *consistent modeling approach*

Numerous tutorials by Alex Champanard and others available online.

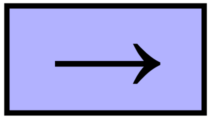
Supported in many engines such as Unreal,

… and TAGE!

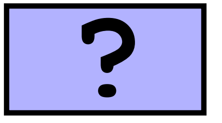
Behavior Tree components

Interior Nodes

- **Sequence node** ***(similar to AND)***



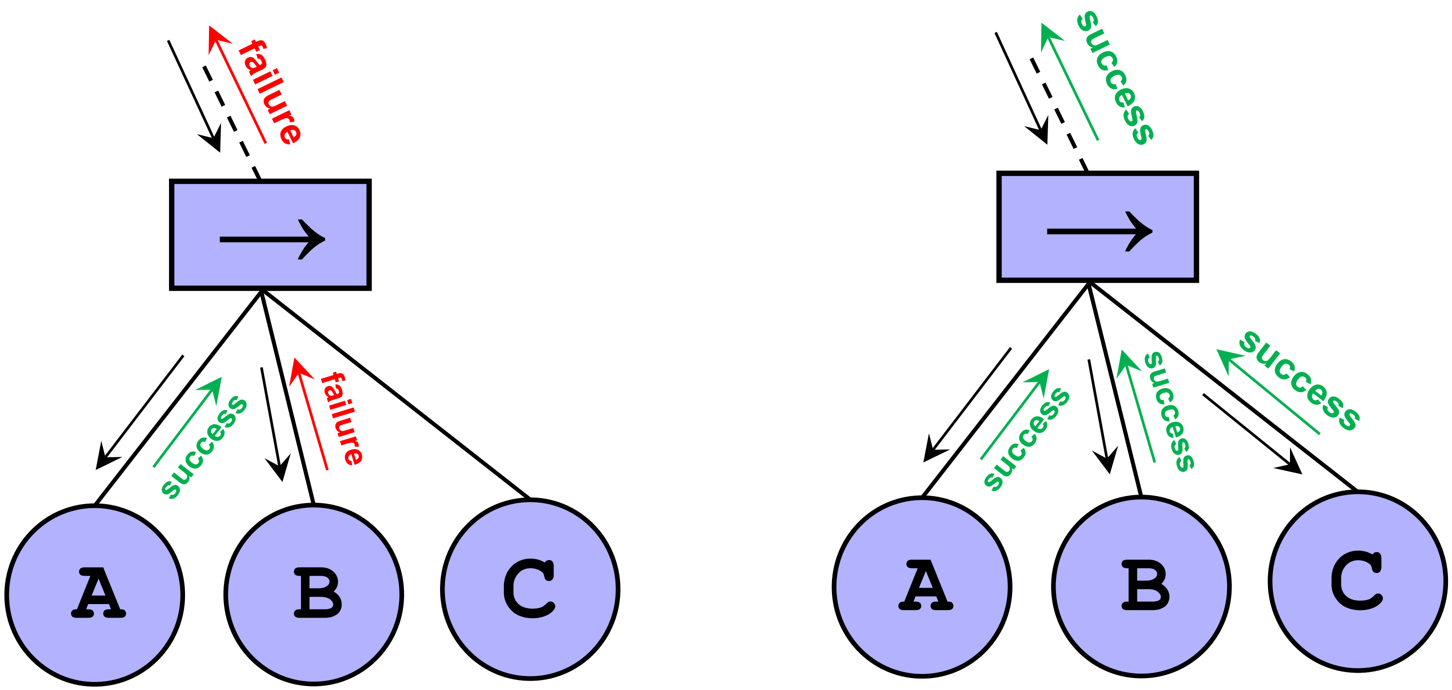
- **Selector node** ***(similar to OR)***



Leaf Nodes:

- **Can be either *Conditions* or *Actions***
- **are game-dependent**
- **Actions typically change the NPC or game state**
- **Conditions return true or false based on some check**

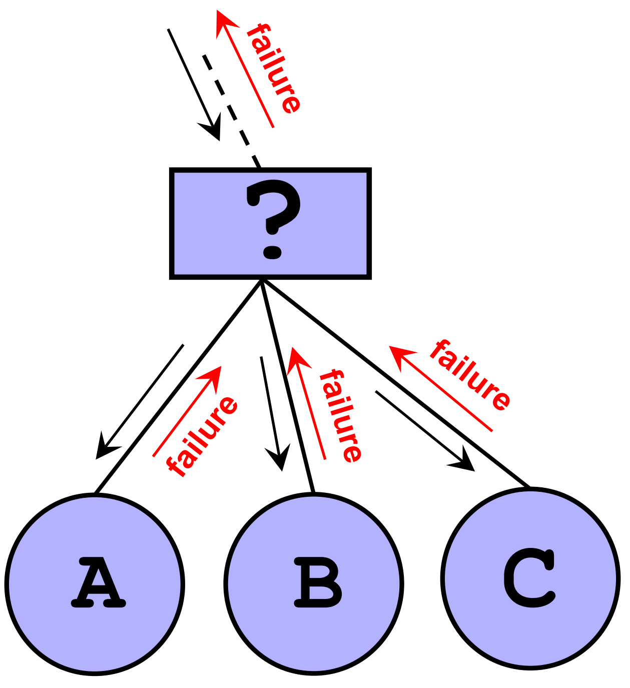
Sequence Nodes



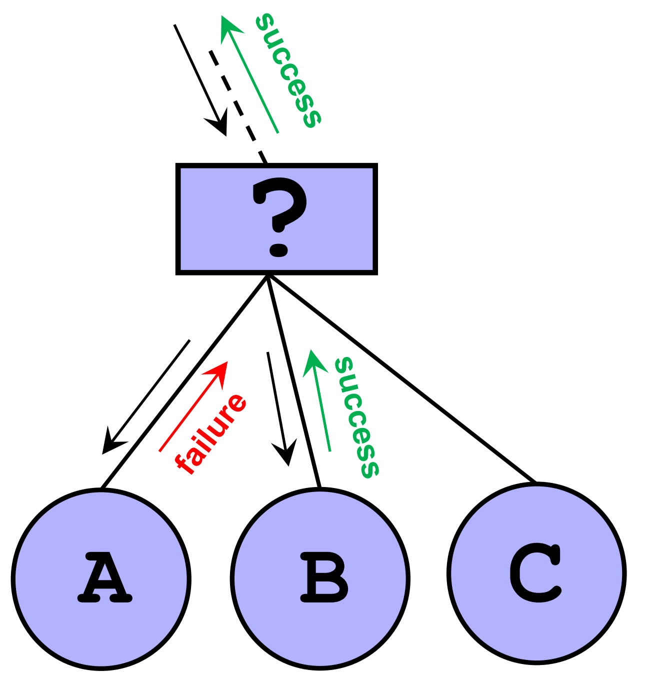
Sequence fails

Sequence succeeds

Selector Nodes

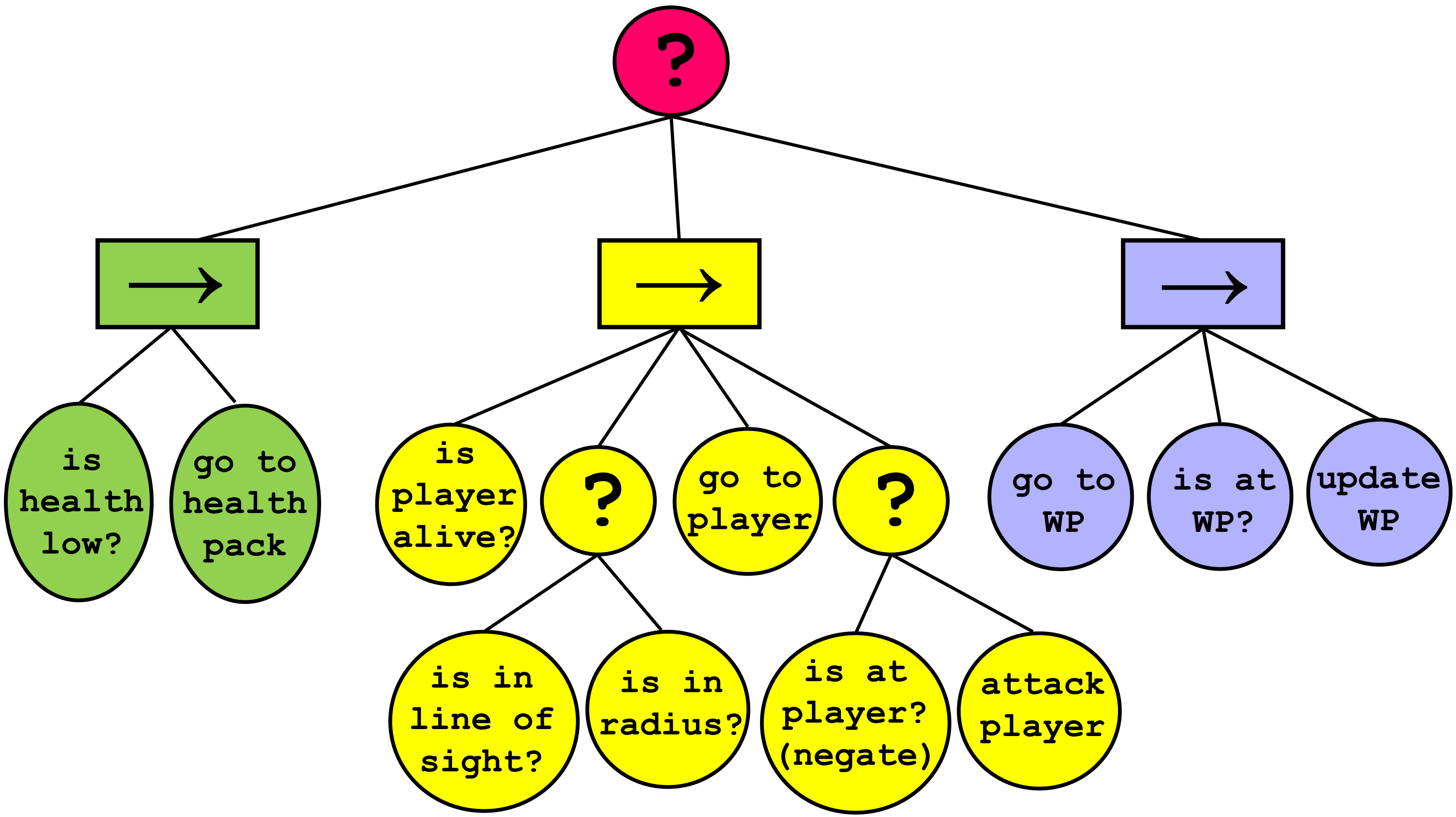


Selector fails

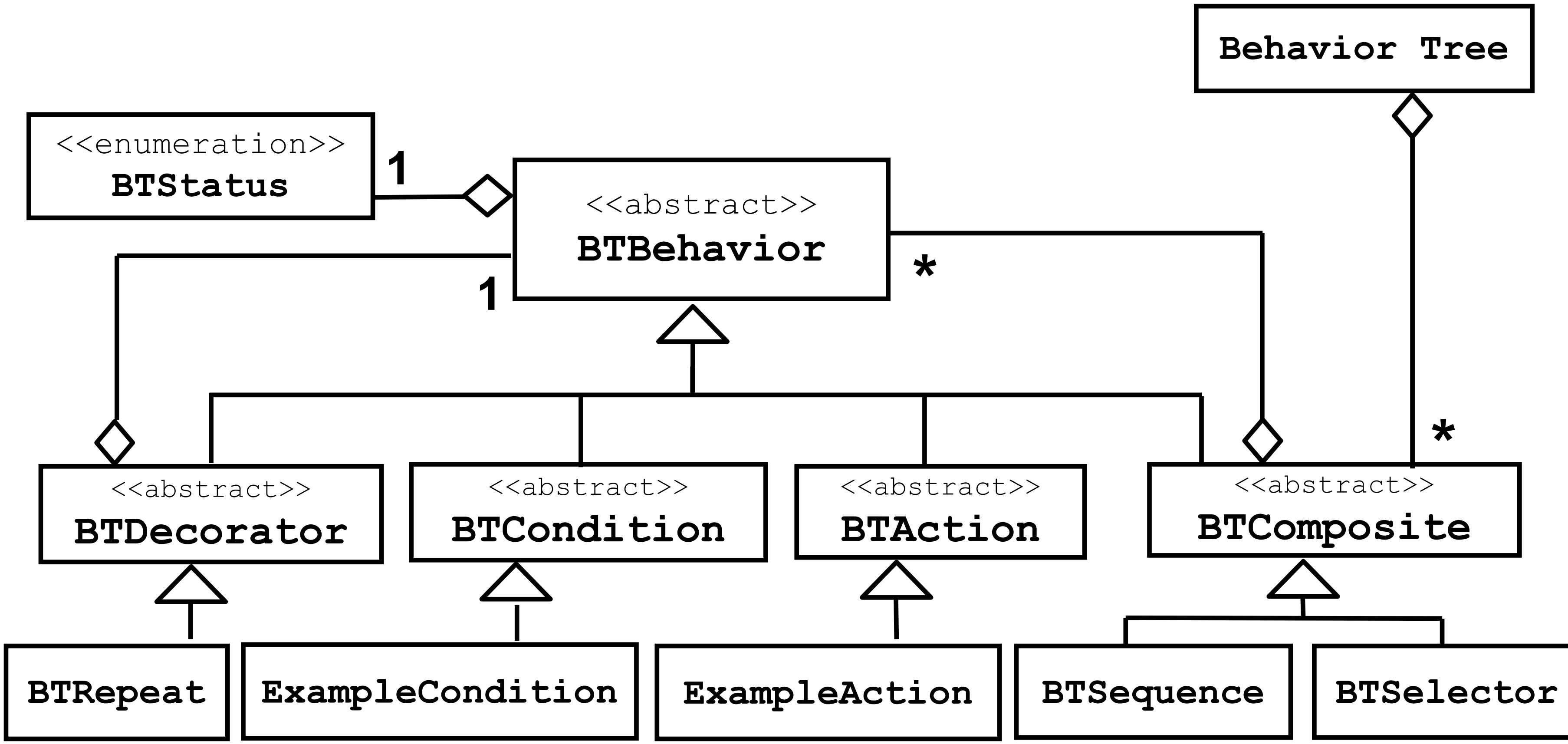


Selector succeeds

Example Behavior Tree



Behavior Trees in TAGE



Managing AI Performance

One simple approach is "tick and think":

- "Tick" phase
  - ✓ done frequently (such as every frame)
  - ✓ simple, predictable steps (such as movement) only
- "Think" phase
  - ✓ less frequent (such as once or twice per second)
  - ✓ full decision-making (such as behavior tree evaluation)
  - ✓ generally also includes a "tick"

THINK
tick
tick
tick
THINK
tick
tick
tick
THINK
…

May stagger the "Think" operations if multiple NPCs:

THINK – NPC1
tick
tick
THINK – NPC2
tick
tick
…