ARTIFICIAL INTELLIGENCE : INTELLIGENT BEHAVIORS

Behrouz Minaei

Iran University of Science and Technology

Basic Chasing and Evading

- the simplest chase algorithm involves correcting the predator's coordinates based on the prey's coordinates so as to reduce the distance between their positions.

- This is a very common method for implementing basic chasing and evading.

Basic Chasing

```
if (predatorX > preyX)

    predatorX--;

else if (predatorX < preyX)

    predatorX++;

if (predatorY > preyY)

    predatorY--;

else if (predatorY < preyY)

    predatorY++;
```

Basic Evading

```
if (preyX > predatorX)

    preyX++;

else if (preyX < predatorX)

    preyX--?>;

if (preyY > predatorY)

    preyY++;

else if (preyY < predatorY)

    preyY--;
```

Tile based chasing

- Tile based Chasing means chasing in 2D environments where we have a tiled map

```
if (predatorCol > preyCol)
    predatorCol--;
else if (predatorCol < preyCol)
    predatorCol++;
if (predatorRow> preyRow)
    predatorRow--;
else if (predatorRow<preyRow)
    predatorRow++;
```

Tile based Chasing

Line of Sight Chasing

- The gist of the line-of-sight approach is to have the predator take a straight-line path toward the prey

- the predator always moves directly toward the prey's current position.

- If the prey is standing still, the predator will take a straight line path. However, if the prey is moving, the path will not necessarily be a straight line.

- The predator still will attempt to move directly toward the current position of the prey, but by the time he catches up with the moving prey, the path he would have taken might be curved

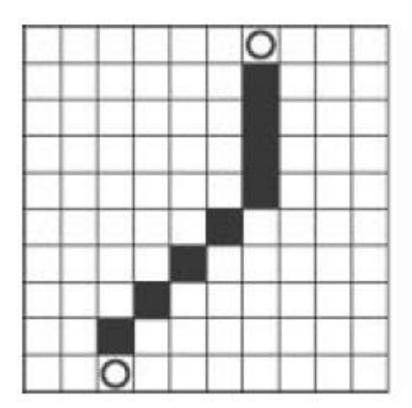# Basic Layout of Line of Sight Chasing

Line of Sight Chasing Algorithm

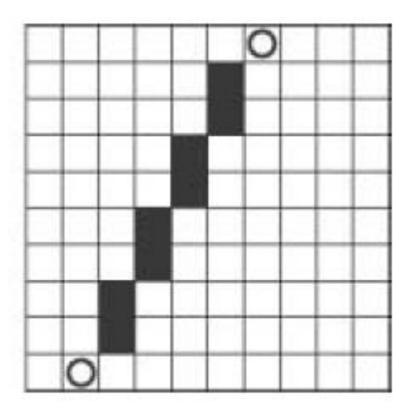☐ you can use the simple chasing algorithm to make the troll relentlessly chase the player.

☐ It will even calculate the shortest possible path to the player. So, what's the disadvantage?

  ◻ When viewed in a tile-based environment, the simple chase method doesn't always appear to produce a visually straight line.

Line of Sight Chasing Algorithm in tiled environment
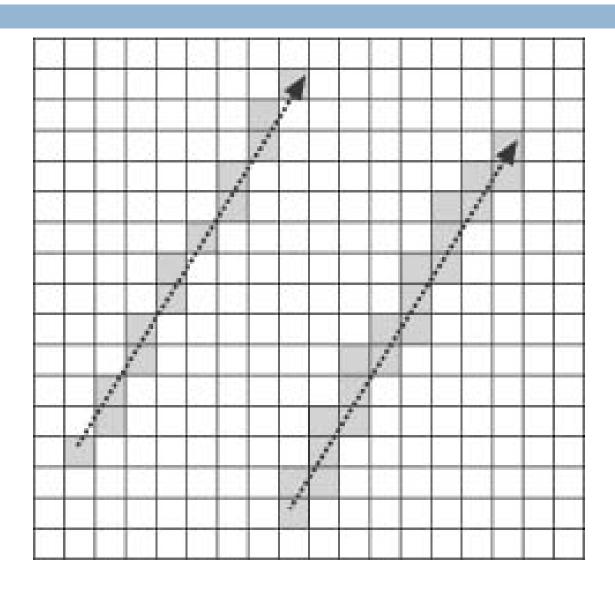
Basic Chasing in Tiled Environments
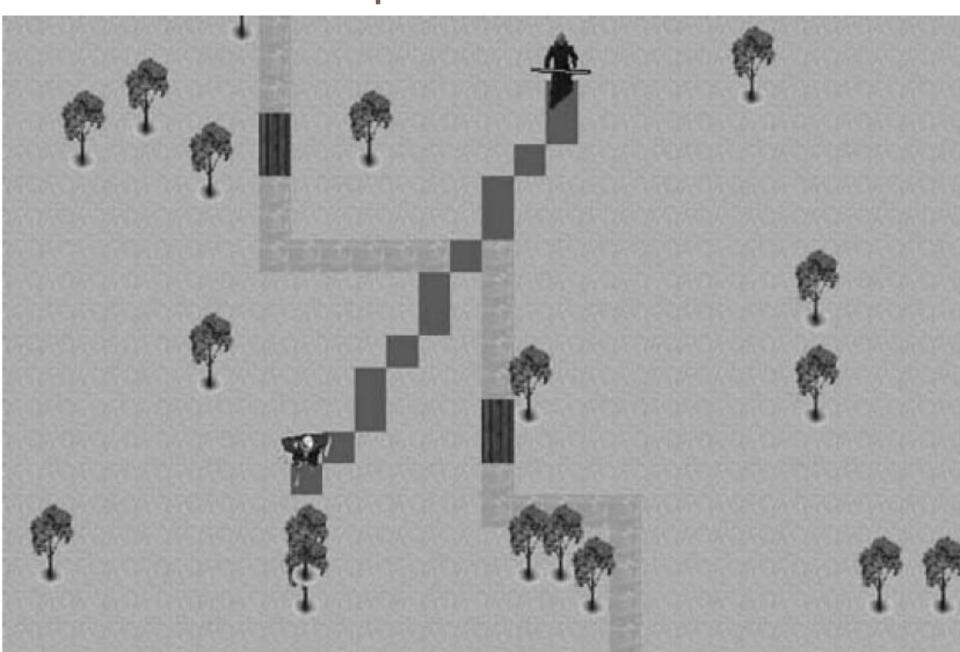
Preferred Line of chasing

Bresenham's line algorithm

- Although you can calculate the points of a line in several ways, we're going to use Bresenham's line algorithm.

- Bresenham's algorithm is one of the more efficient methods for drawing a line in a pixel-based environment, but that's not the only reason it's useful for path finding calculations.

- Bresenham's algorithm also is attractive because unlike some other line-drawing algorithms, it will never draw two adjacent pixels along a line's shortest axis.

Bersenham (Left) V.S normal (Right) algorithm

Bersenham output

Flocking

- Often in video games, non-player characters must move in cohesive groups rather than independently.

- Let's consider some examples.
  - Lets say you are creating a herd of sheep. Your sheep would appear more realistic if they were grazing in a flock rather than walking around aimlessly.

- Here again, birds that hunt in flocks rather than independently would seem more realistic and pose the challenge to the player of dealing with somewhat cooperating groups of predators.

The father of Flocking attributes!

- At the heart of such group behavior lie basic flocking algorithms such as the one presented by Craig Reynolds in his 1987 SIGGRAPH paper, "Flocks, Herds, and Schools: A Distributed Behavioral Model."
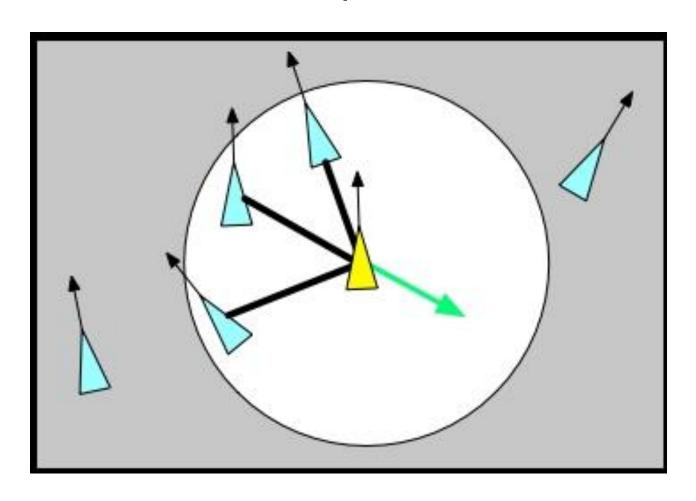
Classic Flocking

- Craig Reynolds coined the term *boids when referring to his simulated flocks.*

- *The behavior he generated very* closely resembles shoals of fish or flocks of birds.

- All the boids can be moving in one direction at one moment, and then the next moment the tip of the flock formation can turn and the rest of the flock will follow as a wave of turning boids propagates through the flock.

- Reynolds' implementation is leaderless in that no one boid actually leads the flock;

Flocking Rules

- impressive is the fact that this behavior is the result of three elegantly simple rules.

- These rules are summarized as follows:
  - *Cohesion :* Have each unit steer toward the average position of its neighbors.

  - *Alignment :* Have each unit steer so as to align itself to the average heading of its neighbors.

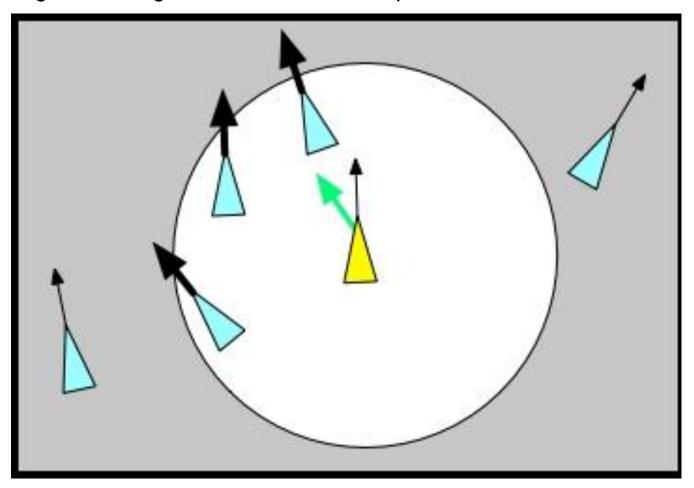 - *Separation:* Have each unit steer to avoid hitting its neighbors.
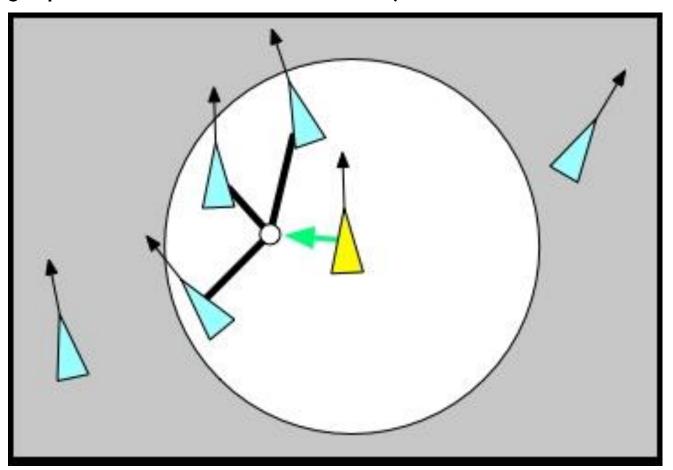
Separation

- Don't collide with nearby flock mates

Alignment

- attempt to match velocity of nearby flockmates (steer towards average heading of local flockmates)
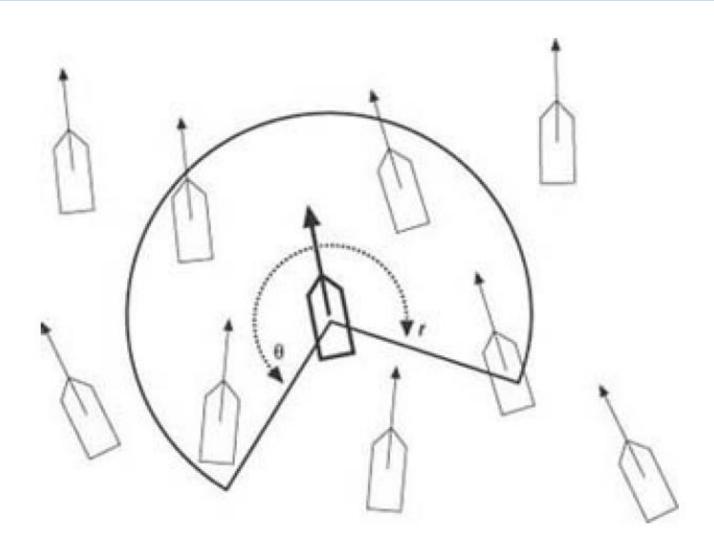
Cohesion

- attempt to stay close to nearby flockmates (steer towards average position of local flockmates)
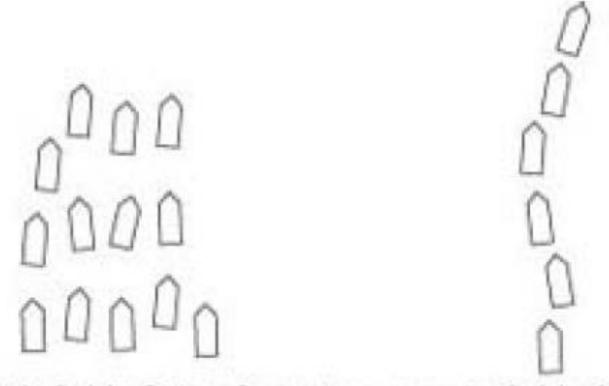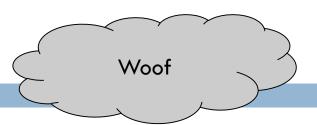
Visibility

Visibility

□ Visibility will determine behavior

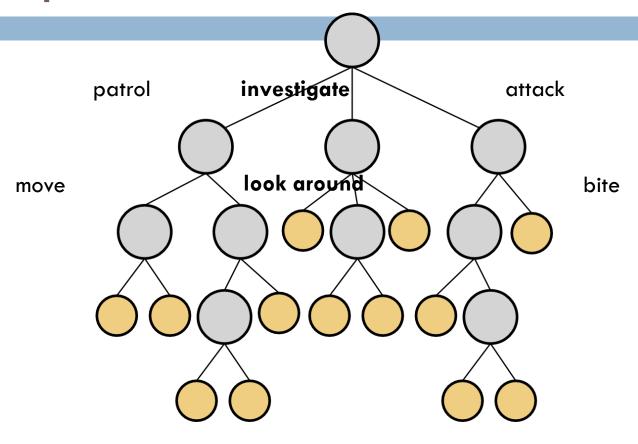Wide field-of-view formation   Narrow field-of-view formation
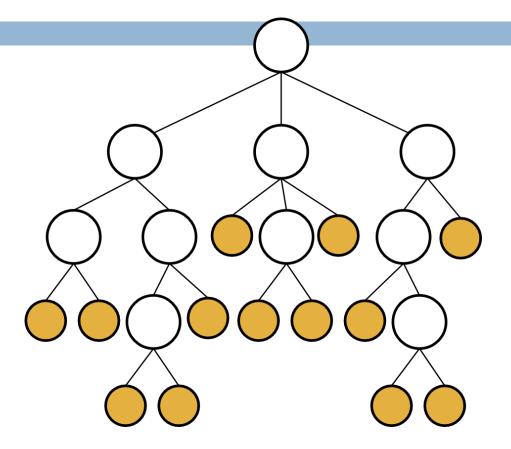
Behavior Tree

A Guard Dog's AI

Woof

Example

patrol · **investigate** · attack
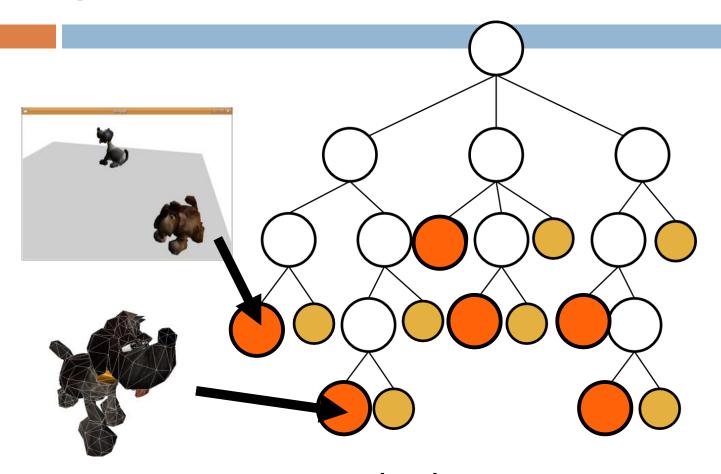
move · **look around** · bite

☑ recursive decomposition

Leaves
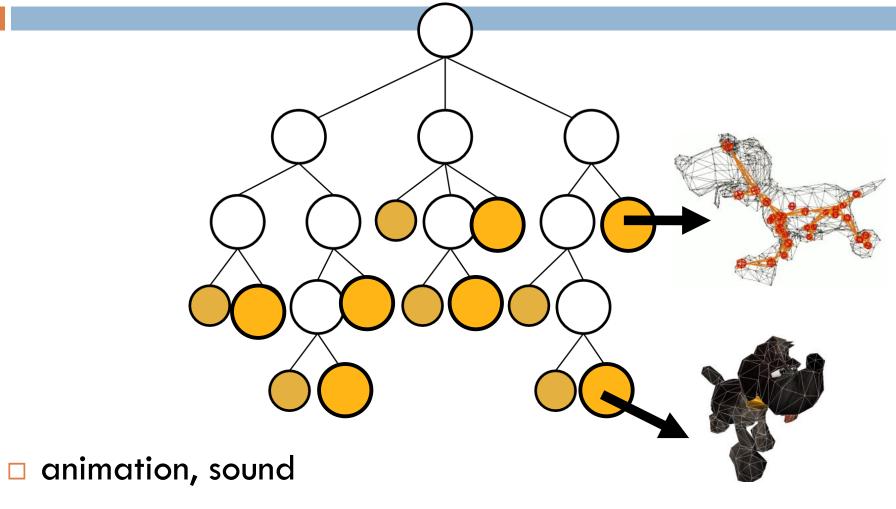
□ interface between AI and engine

Conditions
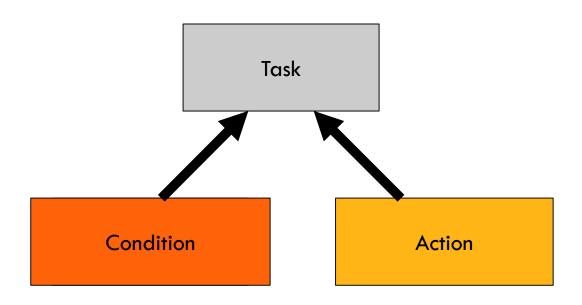
☐ actor state, meta-checks

☐ collision, entity queries

Actions
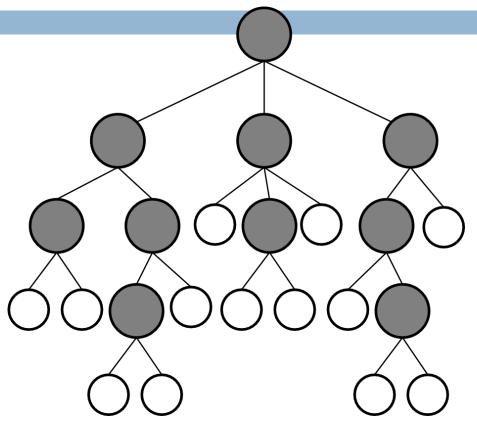

- animation, sound
- using objects, game logic
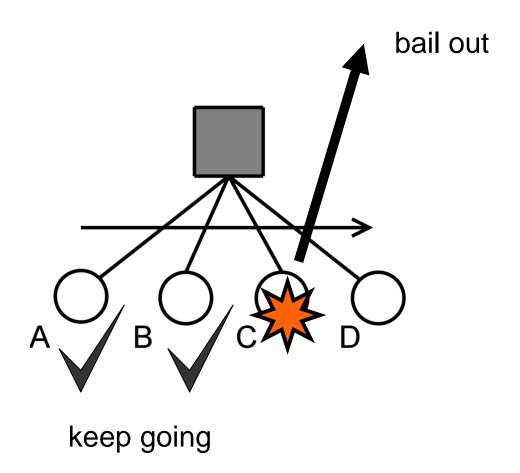
It's All About Tasks

- ☐ Latent computation
- ☐ Succeed or Fail

Building Complexity
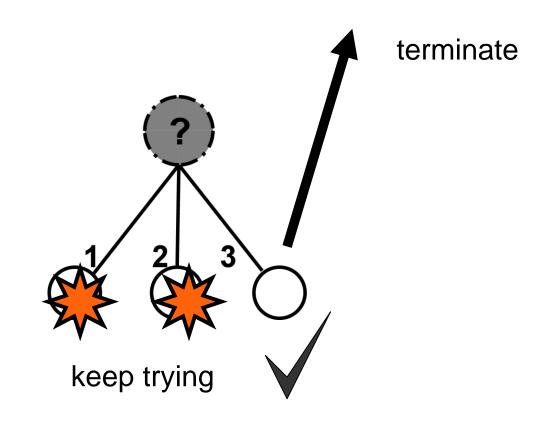
☑ branches manage the leaves
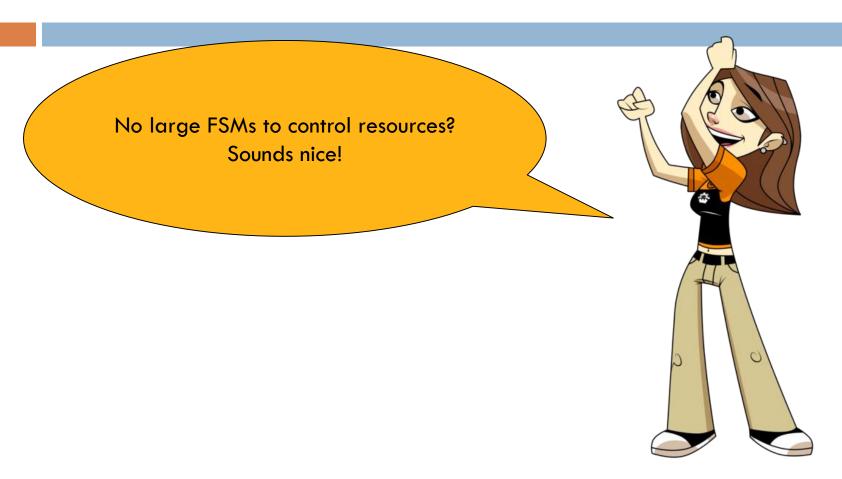
☑ done using composite tasks
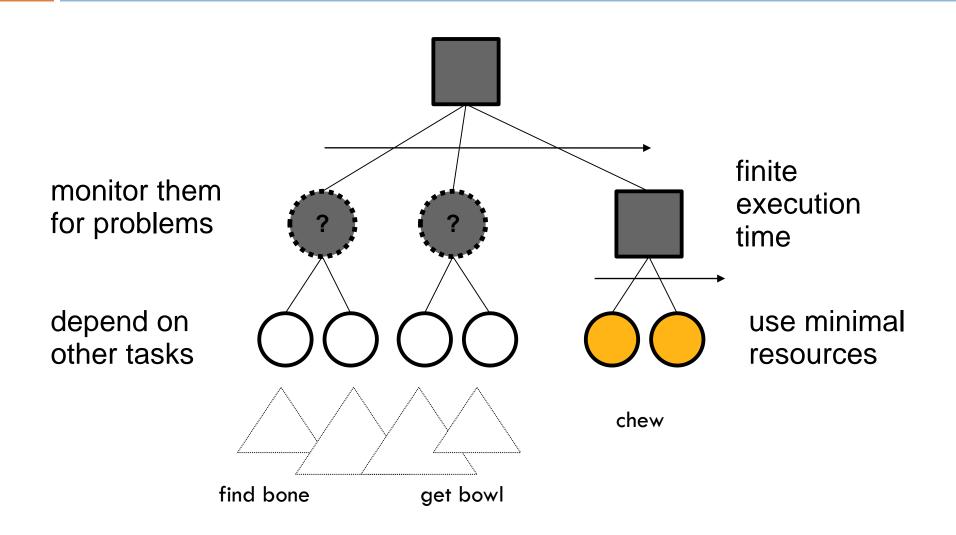
Sequences

Selectors

terminate

?

1   2   3

keep trying

Goal Oriented Architecture

Goal Directed Behaviors

No large FSMs to control resources?
Sounds nice!

bark, eat bone, walk to location, bite, jump,
sit down, hide, chase, growl

Example: Eating a Bone

monitor them for problems

depend on other tasks

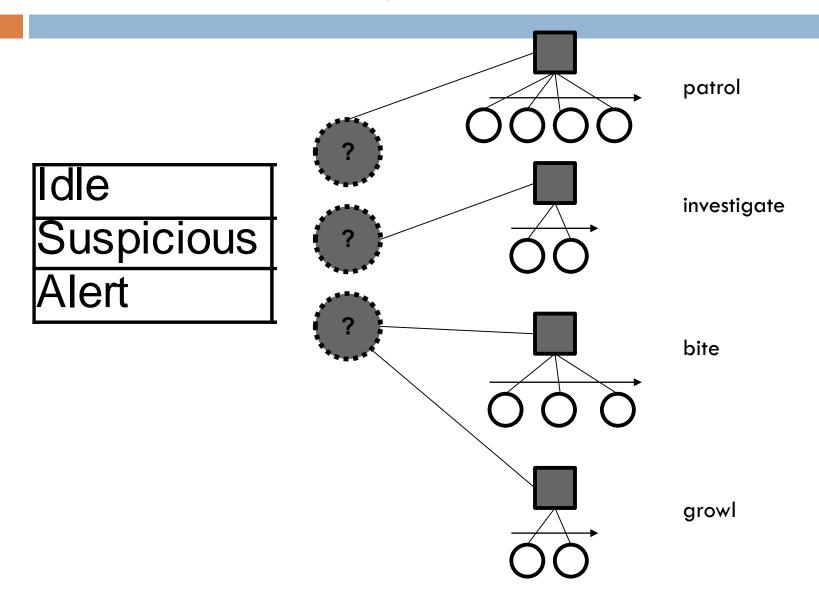find bone   get bowl

chew

finite execution time
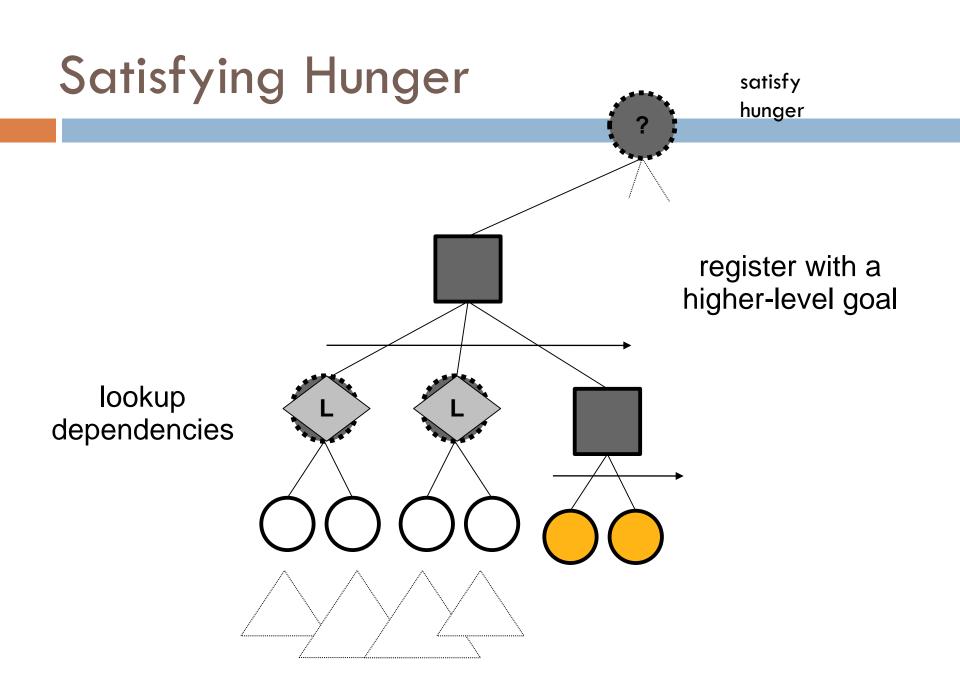
use minimal resources

A Little More Abstraction

- separate WHAT: the **goals**
- from HOW: the **behaviors**
- easier to combine trees together

It's Just a Lookup Table!

Idle
Suspicious
Alert

patrol

investigate

bite

growl

Satisfying Hunger
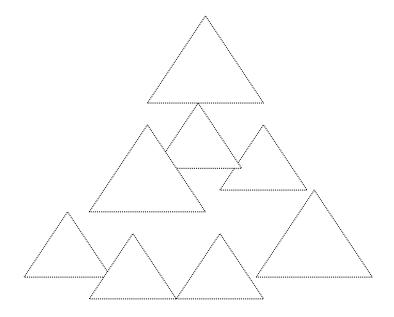
satisfy hunger

register with a higher-level goal
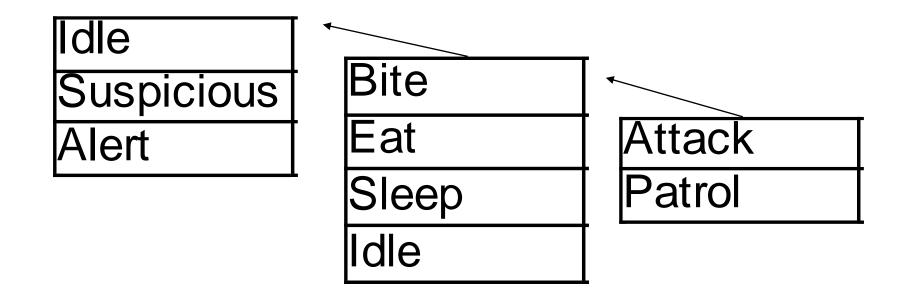
lookup dependencies

Workflow

- build lots of small trees

- connect trees via lookup decorator

- "search tree" assembled automatically

Customization

- use lookup table to customize AI

- per-character, per-group, per-type

- use simple inheritance of tables

Idle
Suspicious
Alert

Bite
Eat
Sleep
Idle

Attack
Patrol

Conclusion

- There are many more ways to create intelligent behaviors. Some of these techniques are:
  - Genetic Algorithms
  - Neural Networks
  - Expert Systems
  - Adaptive Intelligence
  - And many more
- Even Most of the current games use the algorithms that we specified in these set of lectures.
  - Maybe we see the above algorithms more in the future games.