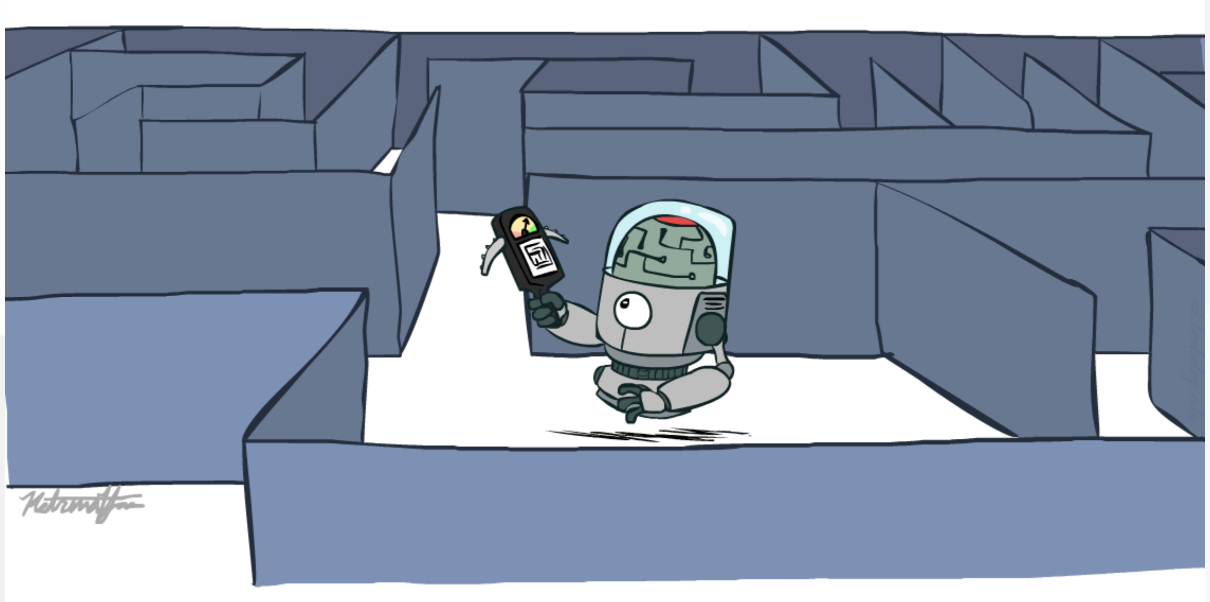
CS 6460: Artificial Intelligence

Informed Search



Instructor: George Rudolph
Utah Valley University Spring 2025

[These slides adapted from Dan Klein and Pieter Abbeel at UC Berkley]

Learning Outcomes

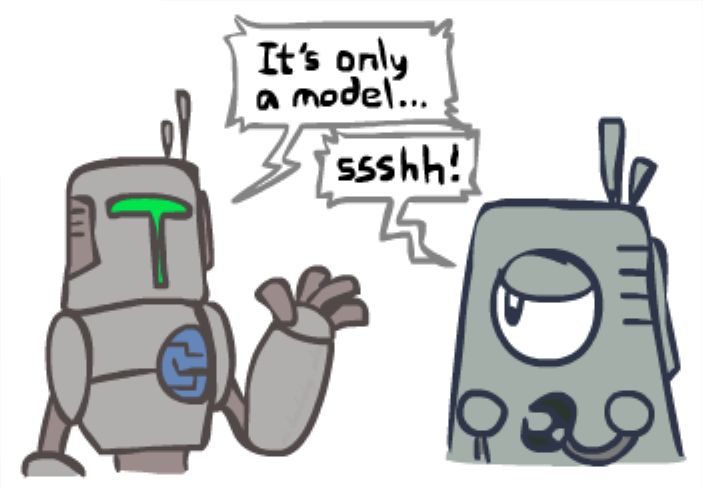
1. Solve Problems using Informed Searches

- Heuristics
- Greedy Search
- A* Search

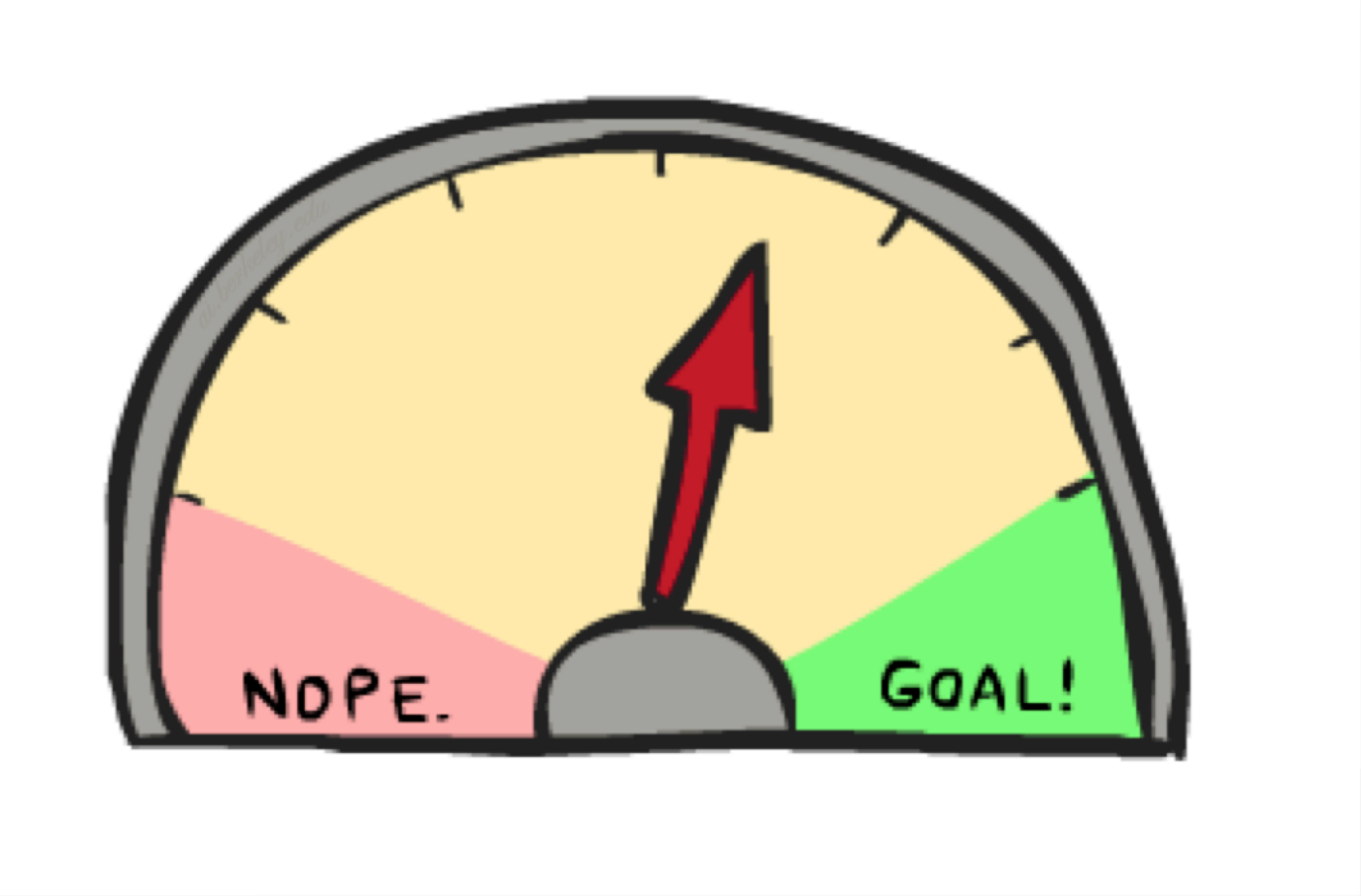
2. Model Problems as Graph Search

Search and Models

- Search operates over models of the world
- The agent doesn't actually try all the plans out in the real world!
- Planning is all in simulation
- Your search is only as good as your models…



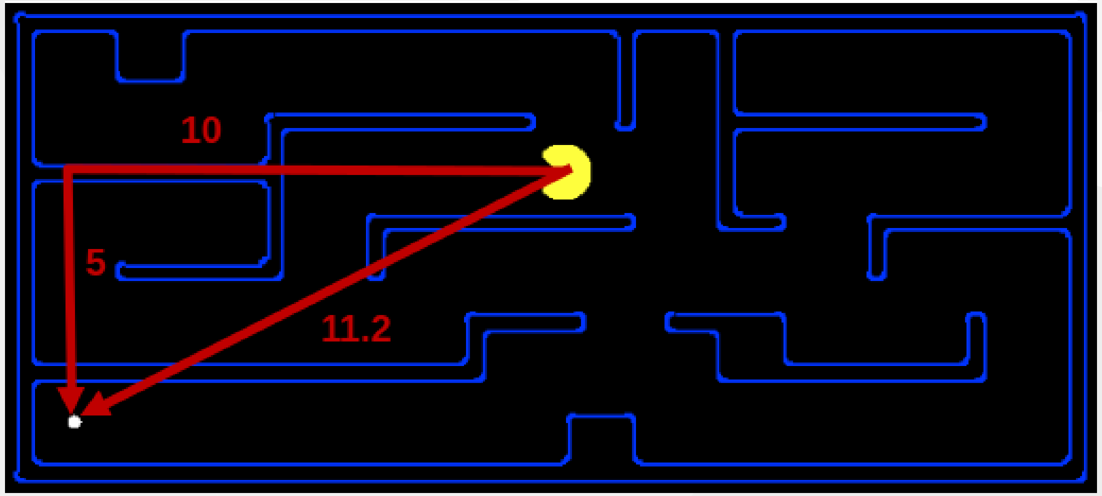
Informed Search

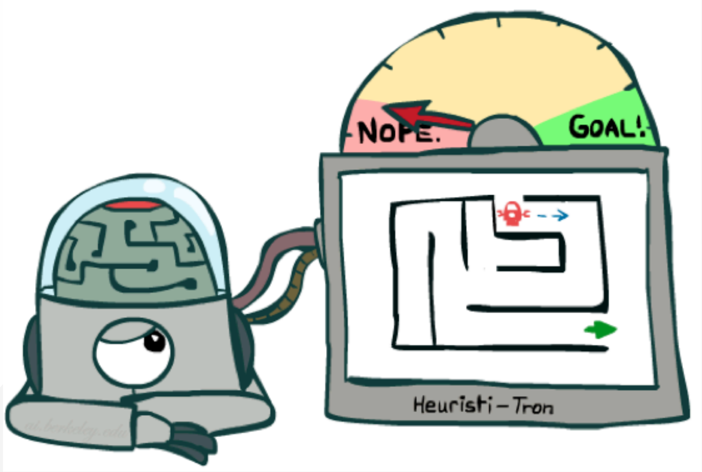


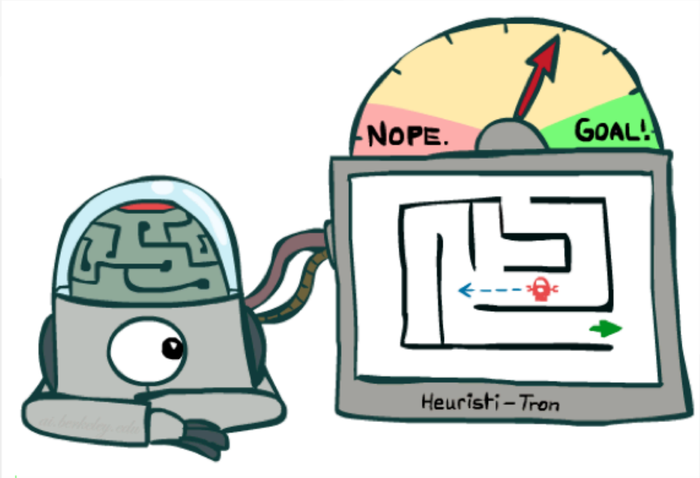
Search Heuristics

A heuristic is:

- A function that **estimates** how close a state is to a goal
- Designed for a **particular** search problem
- Examples: Manhattan distance, Euclidean distance







Example: Heuristic Function

- h(x)

  Example_Heuristic_Function

  Converted shape

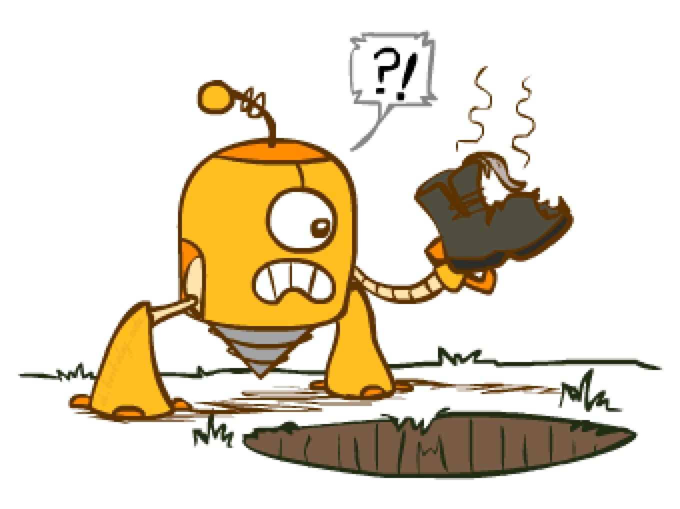
  h(x)

Greedy Search

Greedy Search

- Expand the node that seems closest…
- What can go wrong?

Greedy_Search
Greedy_Search
Greedy_Search



Greedy Search

- Strategy: expand a node that you think is closest to a goal state
- Heuristic: estimate of distance to nearest goal for each state
- A common case:
- Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS
- …
- b
- …
- b

Converted shape

…

Video of Demo Contours Greedy (Empty)

Video of Demo Contours Greedy (Pacman Small Maze)

A* Search

Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- A* Search orders by the sum: f(n) = g(n) + h(n)
- S
- a
- d
- b
- G
- h=5
- h=6
- h=2
- 1
- 8

When should A* terminate?

- Should we stop when we put a goal in the fringe?
- No: only stop when we pull a goal off the fringe
- S
- B
- A
- G
- 2
- 3
- 2
- 2
- h = 1
- h = 2
- h = 0

Is A* Optimal?

- What is wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!
- A
- G
- S
- 1
- 3
- h = 6
- h = 0
- 5
- h = 7

Converted shape

Admissible Heuristics

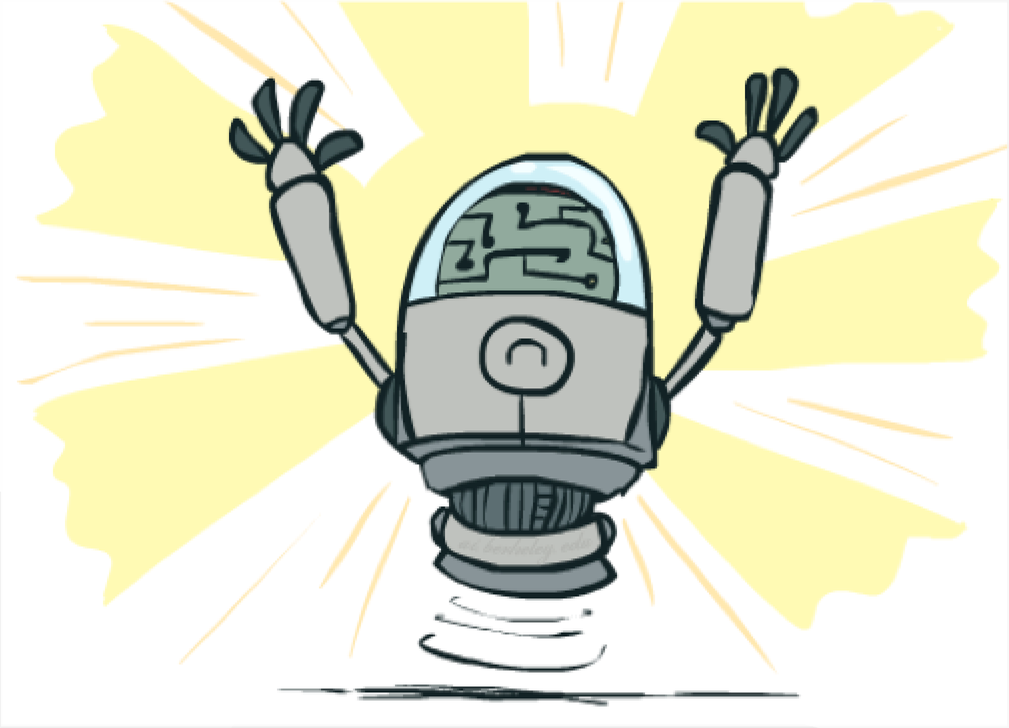
Admissible Heuristics

- A heuristic h is admissible (optimistic) if:
- where is the true cost to a nearest goal
- Examples:
- Defining admissible heuristics is the biggest effort in using A* in practice

$$0 \leq h(n) \leq h^*(n)$$

$$h^*(n)$$

Optimality of A* Tree Search



Optimality of A* Tree Search

- Assume:

Optimality of A* Tree Search: Blocking

- Proof:
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
- f(n) is less or equal to f(A)
- Definition of f-cost
- Admissibility of h
- …
- h = 0 at a goal

$$f(n) = g(n) + h(n)$$

Converted shape

Optimality of A* Tree Search: Blocking

- Proof:
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
- f(n) is less or equal to f(A)
- f(A) is less than f(B)
- B is suboptimal
- h = 0 at a goal
- …

$$f(A) < f(B)$$

Optimality of A* Tree Search: Blocking

- Proof:
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
- f(n) is less or equal to f(A)
- f(A) is less than f(B)
- n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal
- ...

$$f(n) \leq f(A) < f(B)$$

Properties of A*

Properties of A*

- …
- b
- …
- b
- Uniform-Cost
- A*

Converted shape

…

Converted shape

b

UCS vs A* Contours

- Uniform-cost expands equally in all “directions”
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality
- Start
- Goal
- Start
- Goal

Converted shape

Start

Converted shape

Goal

Video of Demo Contours (Empty) -- UCS

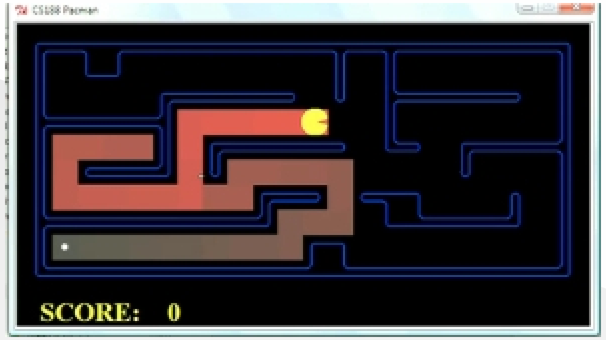
Video of Demo Contours (Empty) -- Greedy

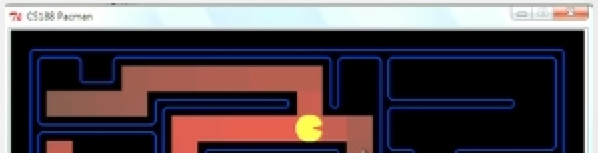
Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

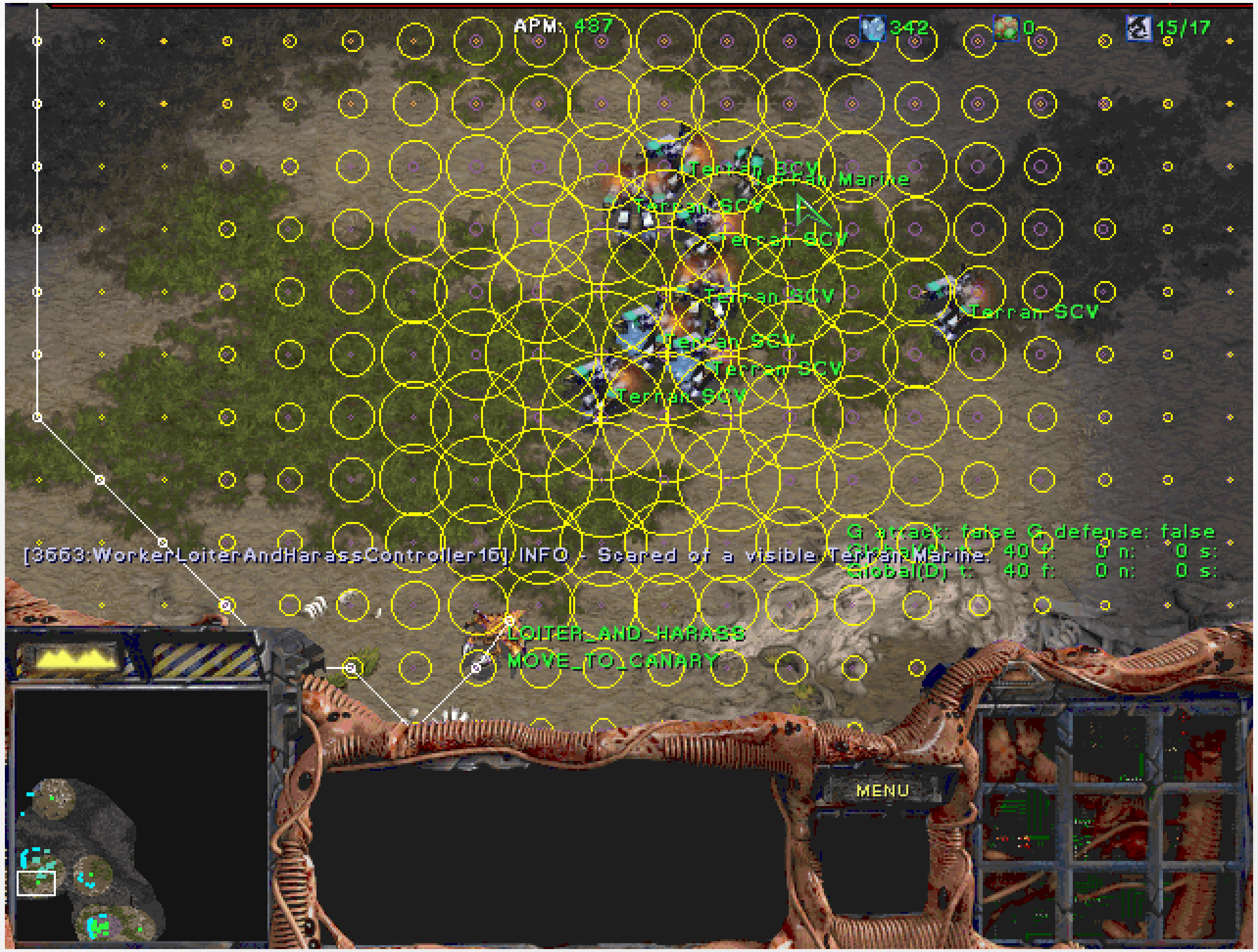
Comparison

- Greedy
- Uniform Cost
- A*





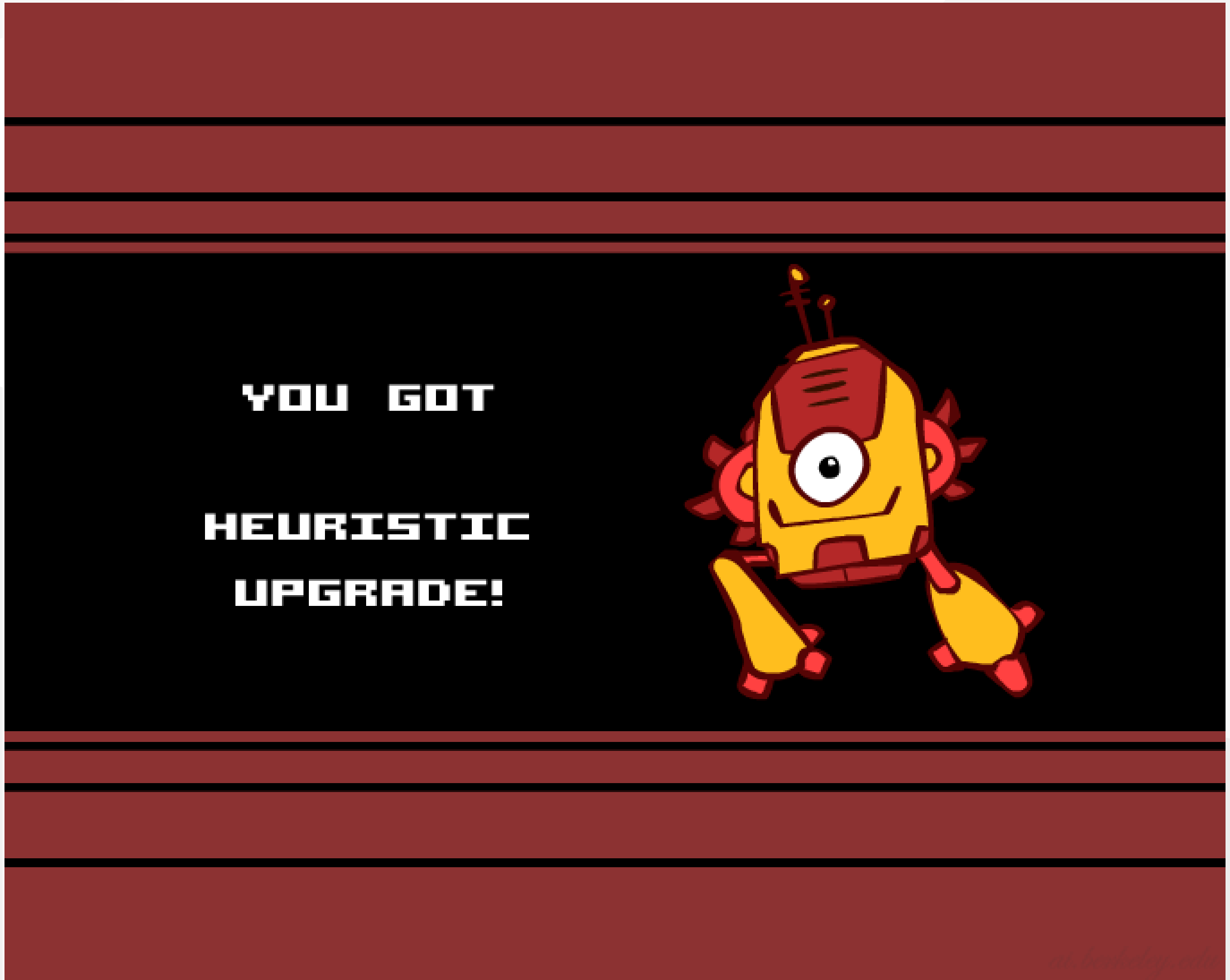
A* Applications



Video of Demo Pacman (Tiny Maze) – UCS / A*

Video of Demo Empty Water Shallow/Deep – Guess Algorithm

Creating Heuristics



Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?
- Start State
- Goal State
- Actions

Converted shape

Group of shapes

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) =
- This is a relaxed-problem heuristic
- 8
- Statistics from Andrew Moore

Converted shape

8

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total Manhattan distance
- Why is it admissible?
- h(start) =
- 3 + 1 + 2 + … = 18

Converted shape

3 + 1 + 2 + … = 18

Converted shape

Start State Goal State

8 Puzzle III

- How about using the actual cost as a heuristic?
- Would it be admissible?
- Would we save on nodes expanded?
- What's wrong with it?
- With A*: a trade-off between quality of estimate and work per node
- As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

- Dominance: ha ≥ hc if
- Heuristics form a semi-lattice:
- Max of admissible heuristics is admissible
- Trivial heuristics
- Bottom of lattice is the zero heuristic (what does this give us?)
- Top of lattice is the exact heuristic

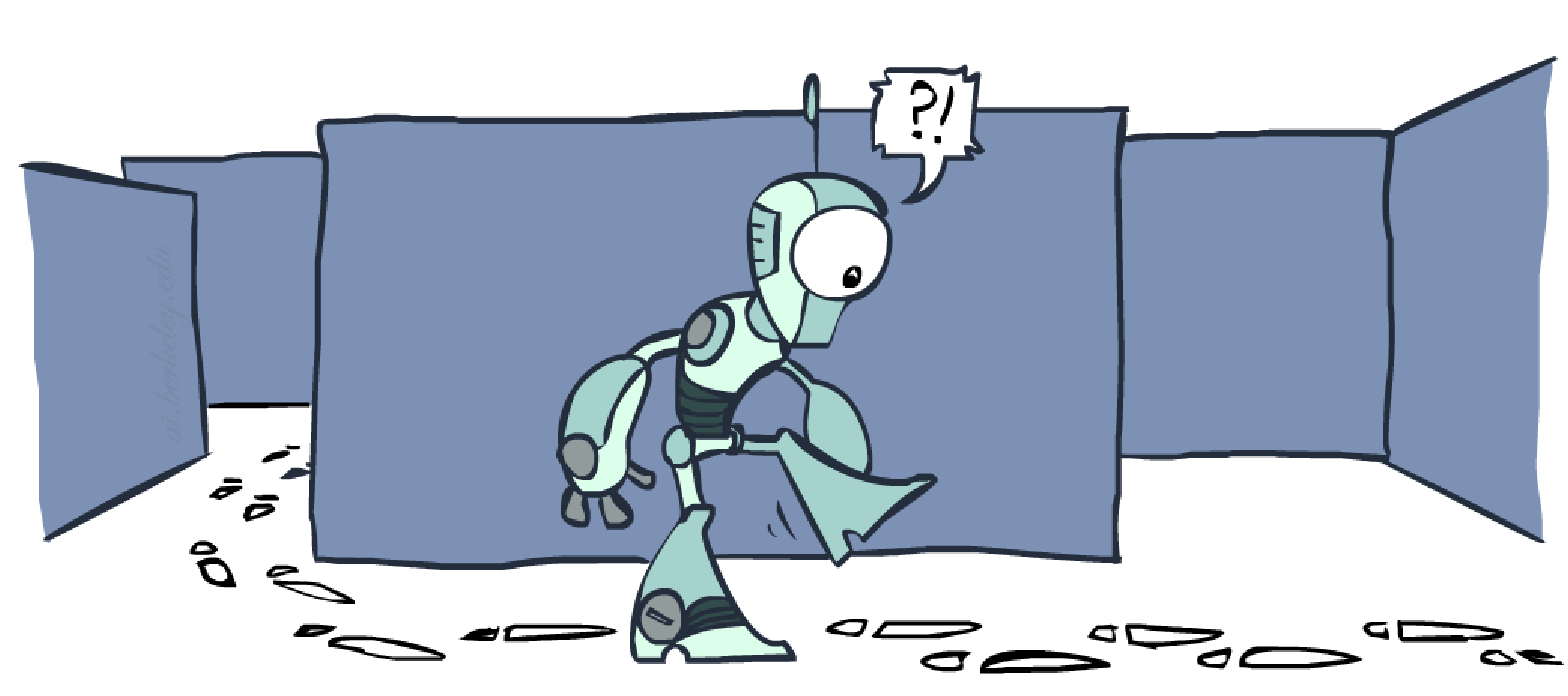
$$h(n) = max(h_a(n), h_b(n))$$

$$\forall n : h_a(n) \geq h_c(n)$$

Converted shape

Group of shapes

Graph Search



Tree Search: Extra Work!

BFS Graph Search Example

- we shouldn’t bother expanding the circled nodes: WHY?

Converted shape

S a b d p a c q

Graph Search

- Idea: never expand a state twice
- How to implement:
- Tree search + set of expanded states (“closed set”)
- Expand the search tree node-by-node, but…
- Before expanding a node, check to make sure its state has never been expanded before
- If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

A* Graph Search Gone Wrong?

- S
- A
- B
- C
- G
- 1
- 1
- 1
- 2
- 3
- S (0+2)
- State space graph
- Search tree

Consistency of Heuristics

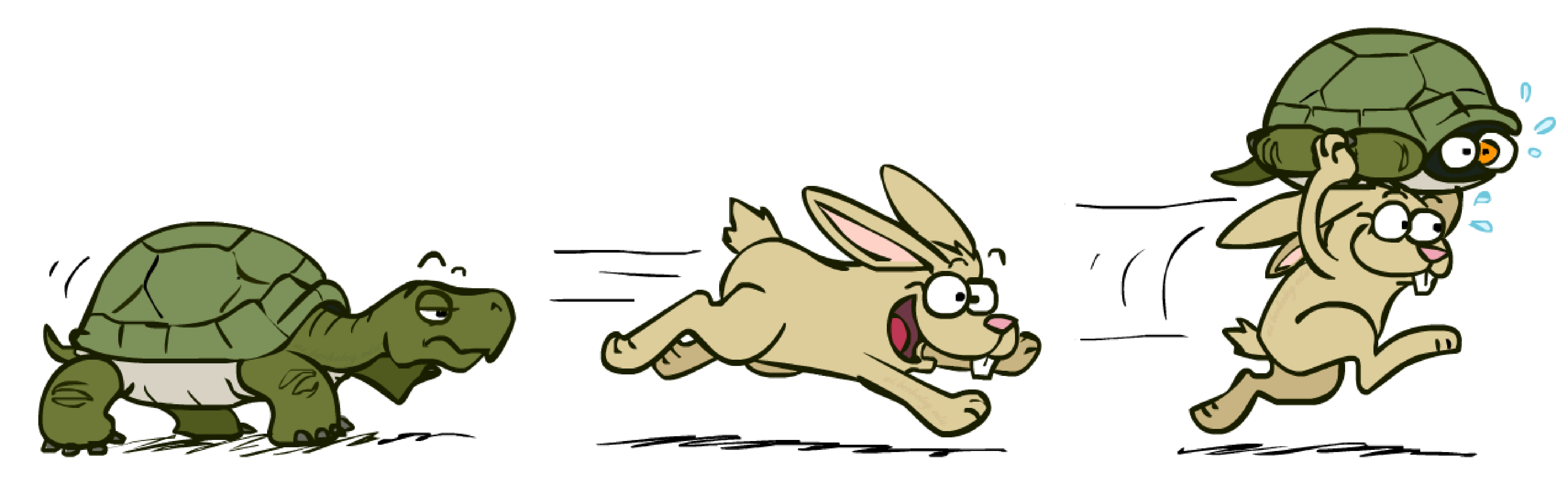
- Main idea: estimated heuristic costs ≤ actual costs
- Admissibility: heuristic cost ≤ actual cost to goal
- h(A) ≤ actual cost from A to G
- Consistency: heuristic “arc” cost ≤ actual cost for each arc
- h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
- The f value along a path never decreases
- h(A) ≤ cost(A to C) + h(C)
- A* graph search is optimal
- 3
- A
- C

Optimality

- Tree search:
- A* is optimal if heuristic is admissible
- UCS is a special case (h = 0)
- Graph search:
- A* optimal if heuristic is consistent
- UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Tree Search Pseudo-Code

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
```

Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
```