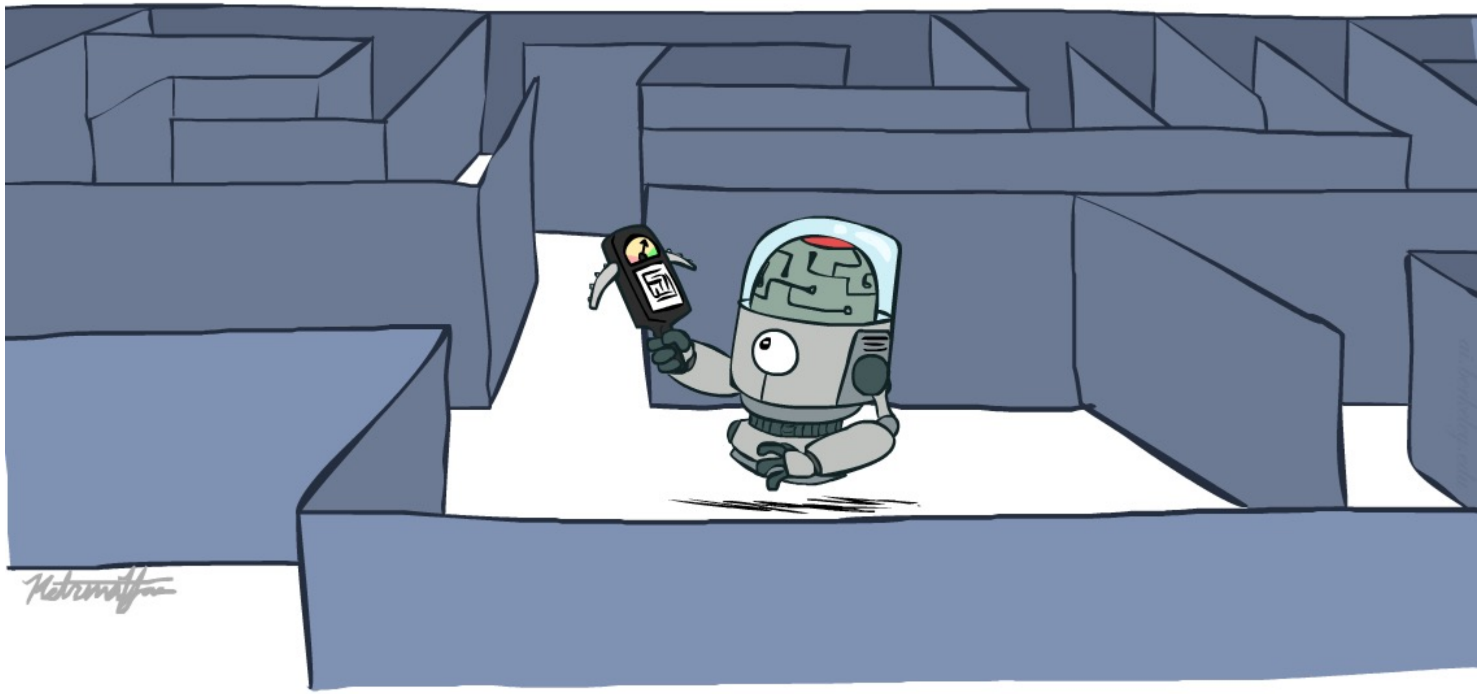
# CS 343: Artificial Intelligence

## Informed Search




Prof. Yuke Zhu

University of Texas at Austin


[These slides based on ones created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

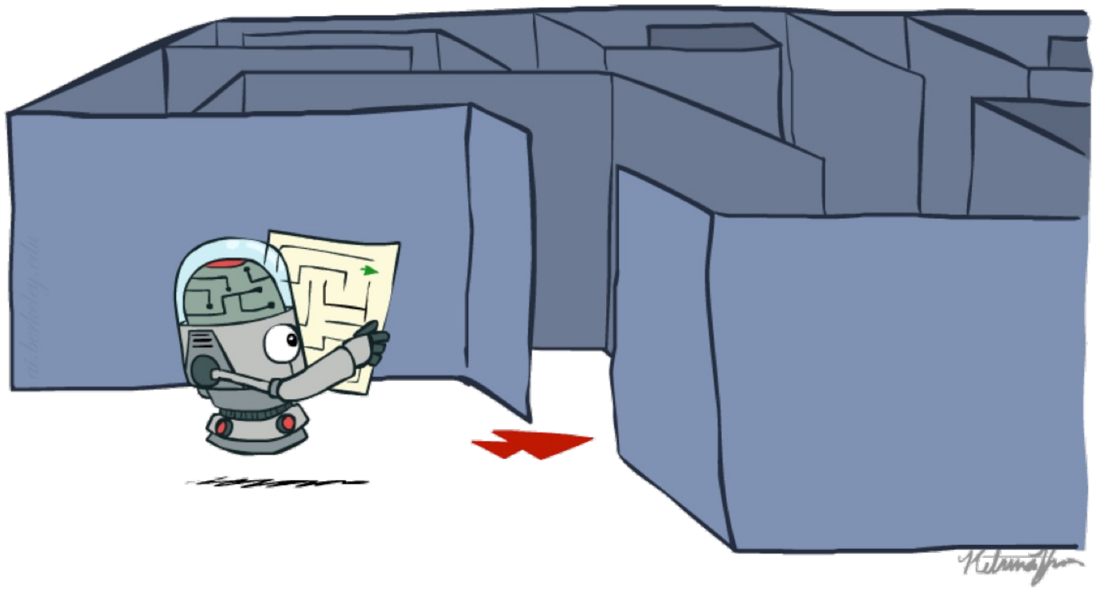
# Today

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search

- Graph Search

# Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

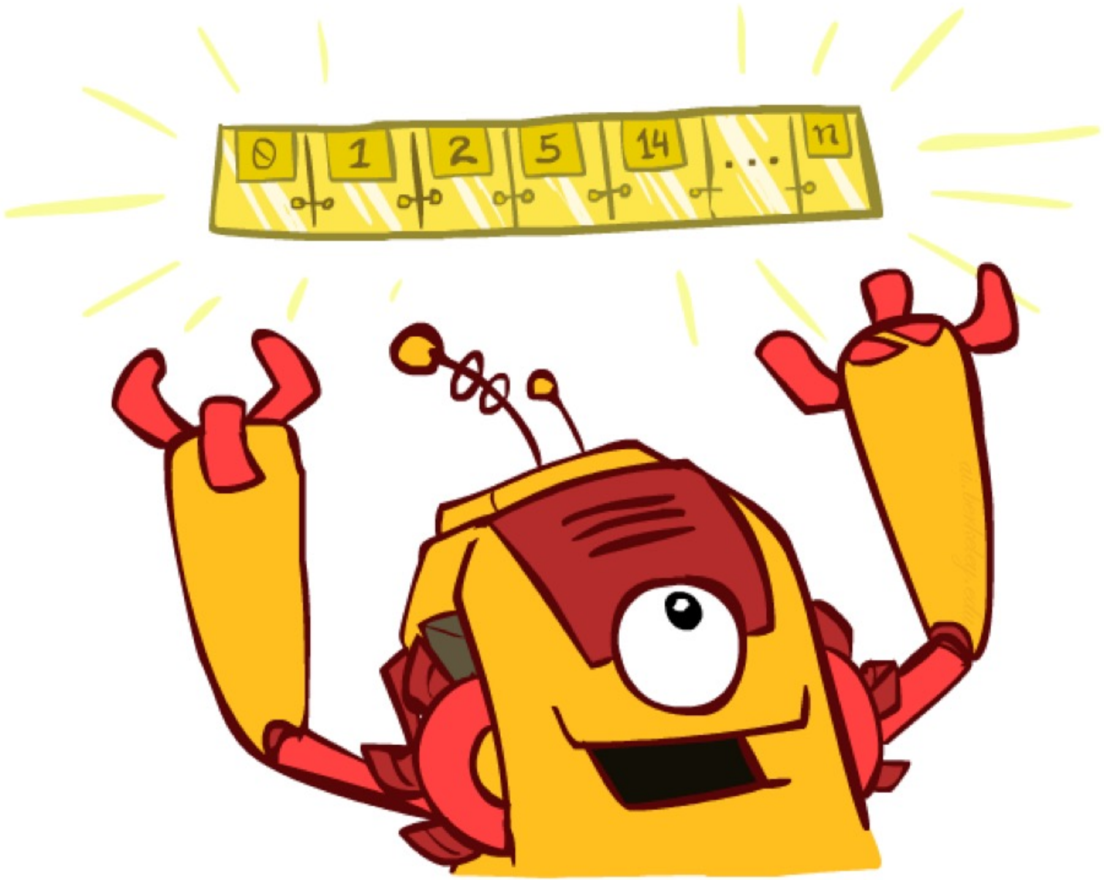


- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

# The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



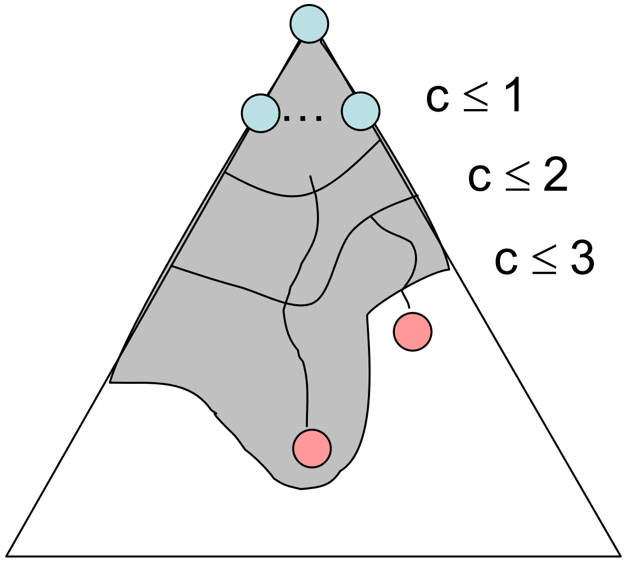
# Uninformed Search

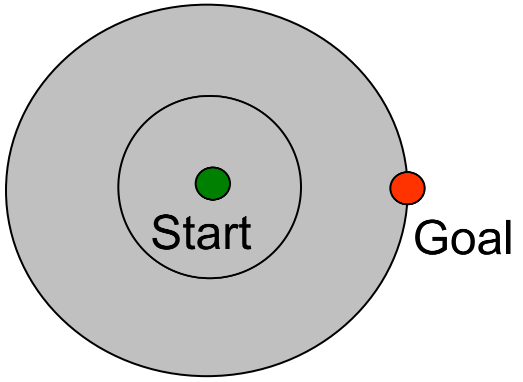
# Uniform Cost Search

- Strategy: expand lowest path cost

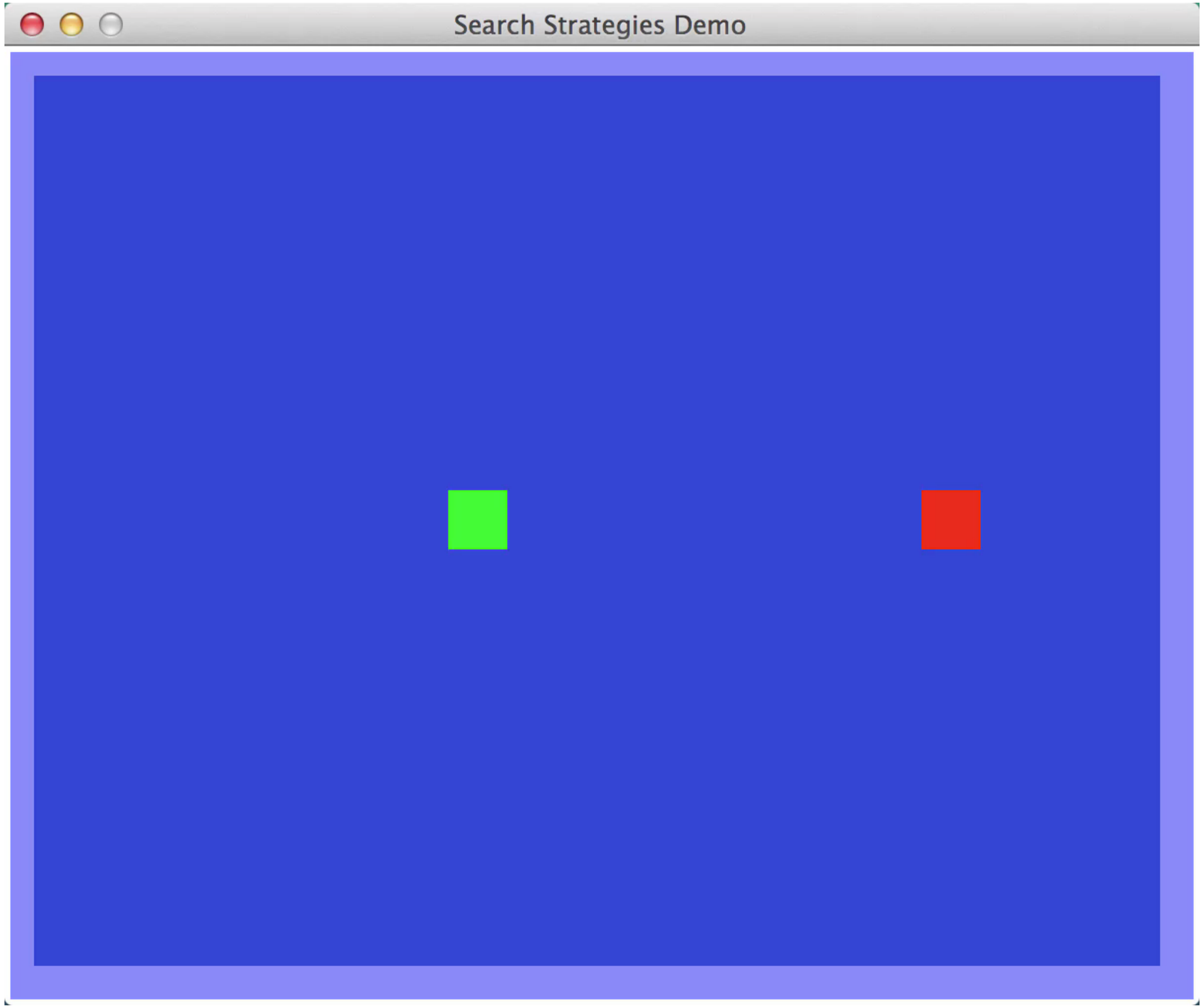
- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every "direction"
  - No information about goal location

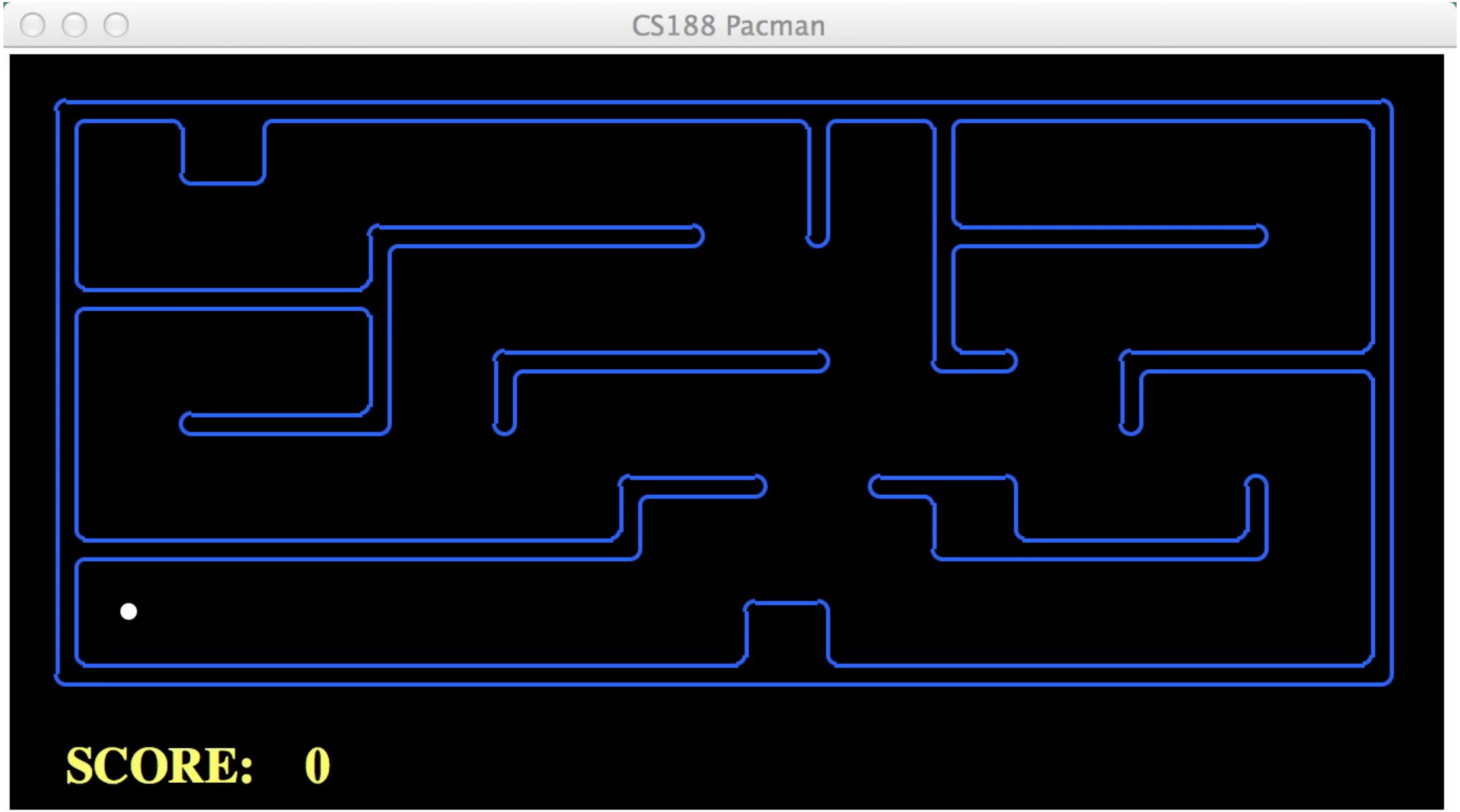


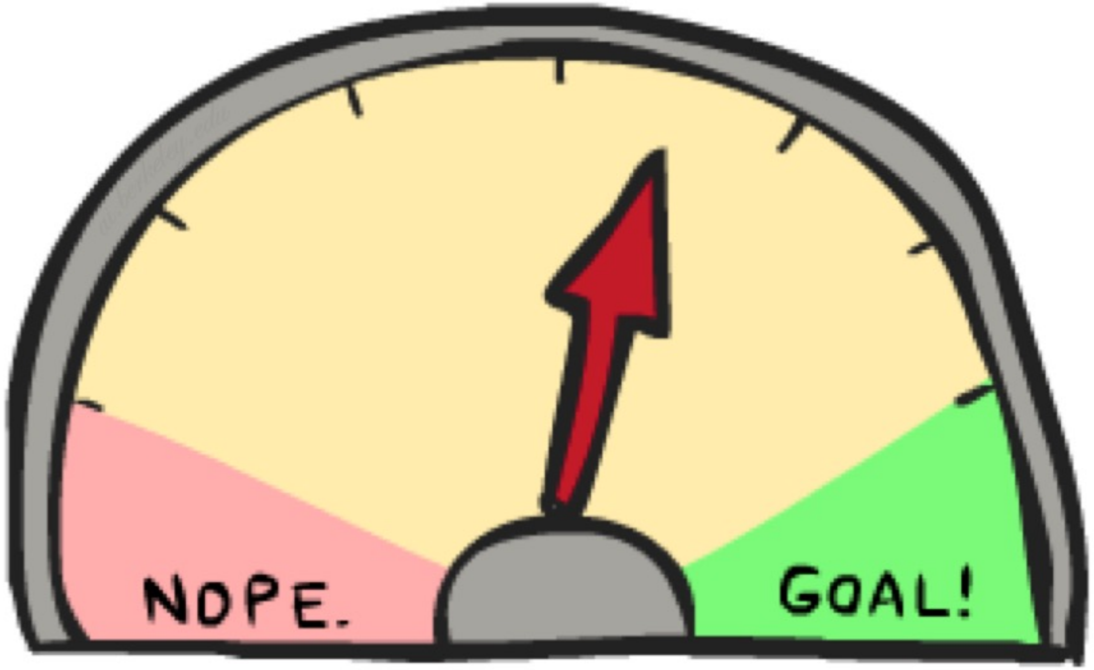
# Video of Demo Contours UCS Empty



# Video of Demo Contours UCS Pacman Small Maze

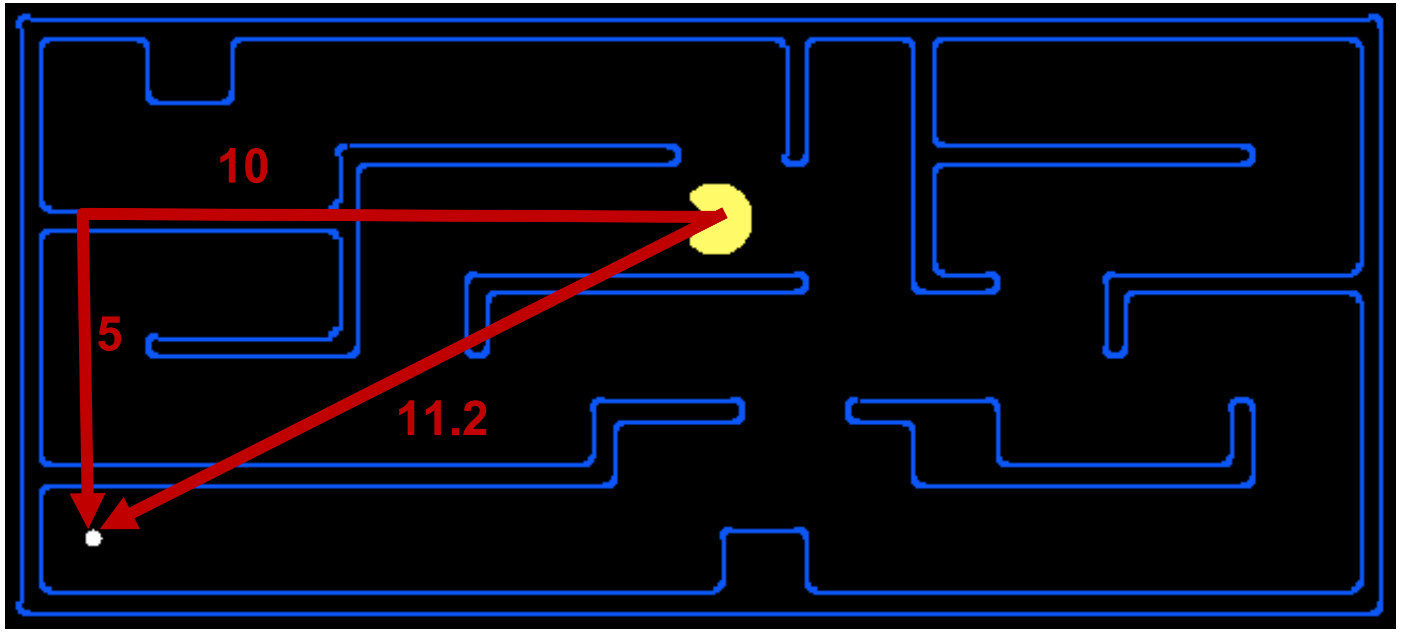


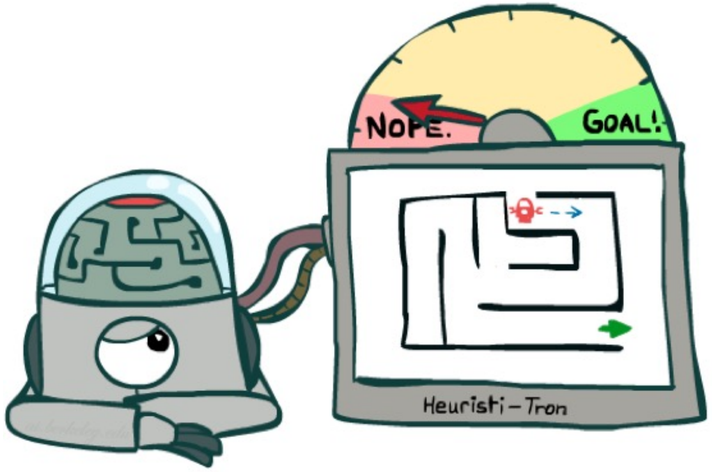
# Informed Search

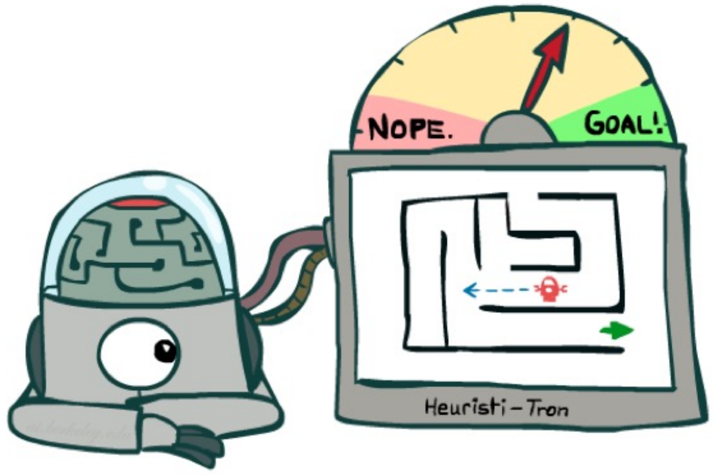


# Search Heuristics

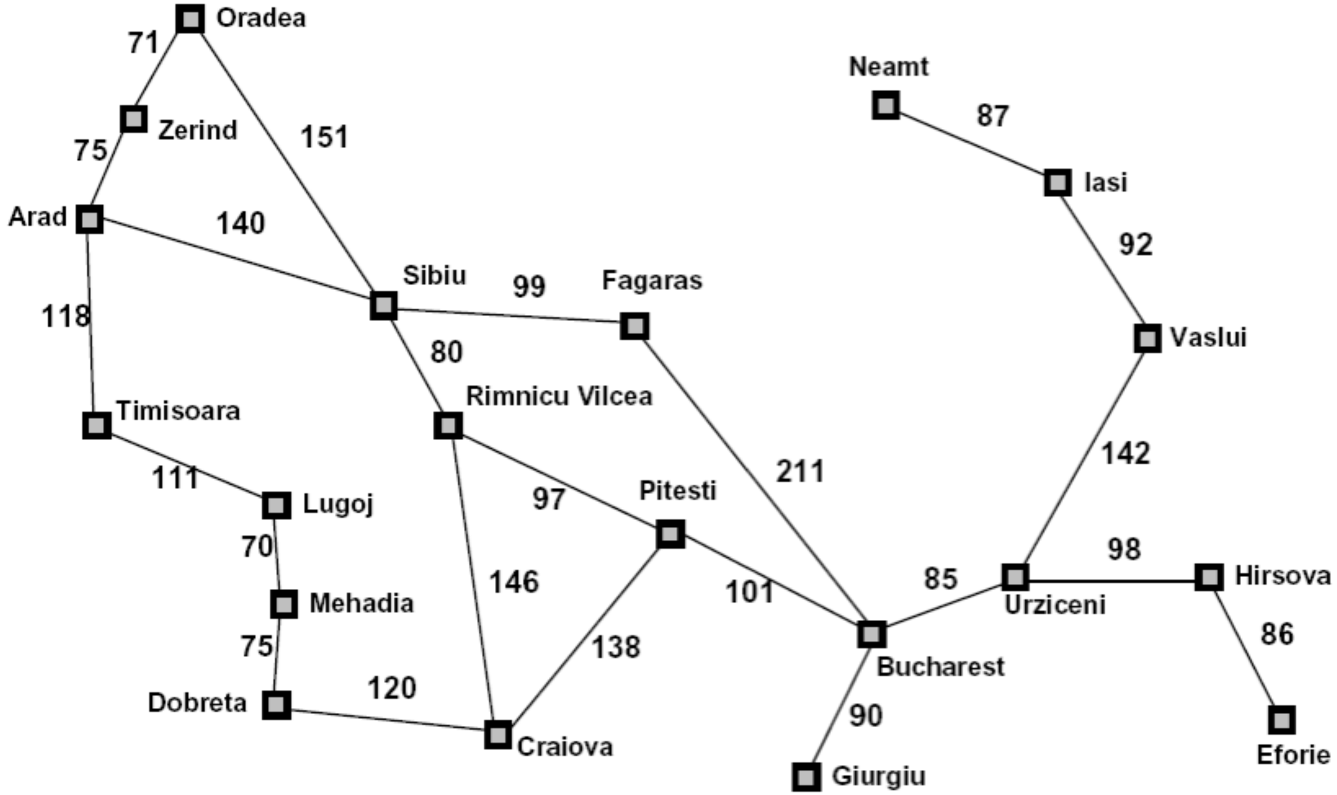
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing



# Example: Heuristic Function



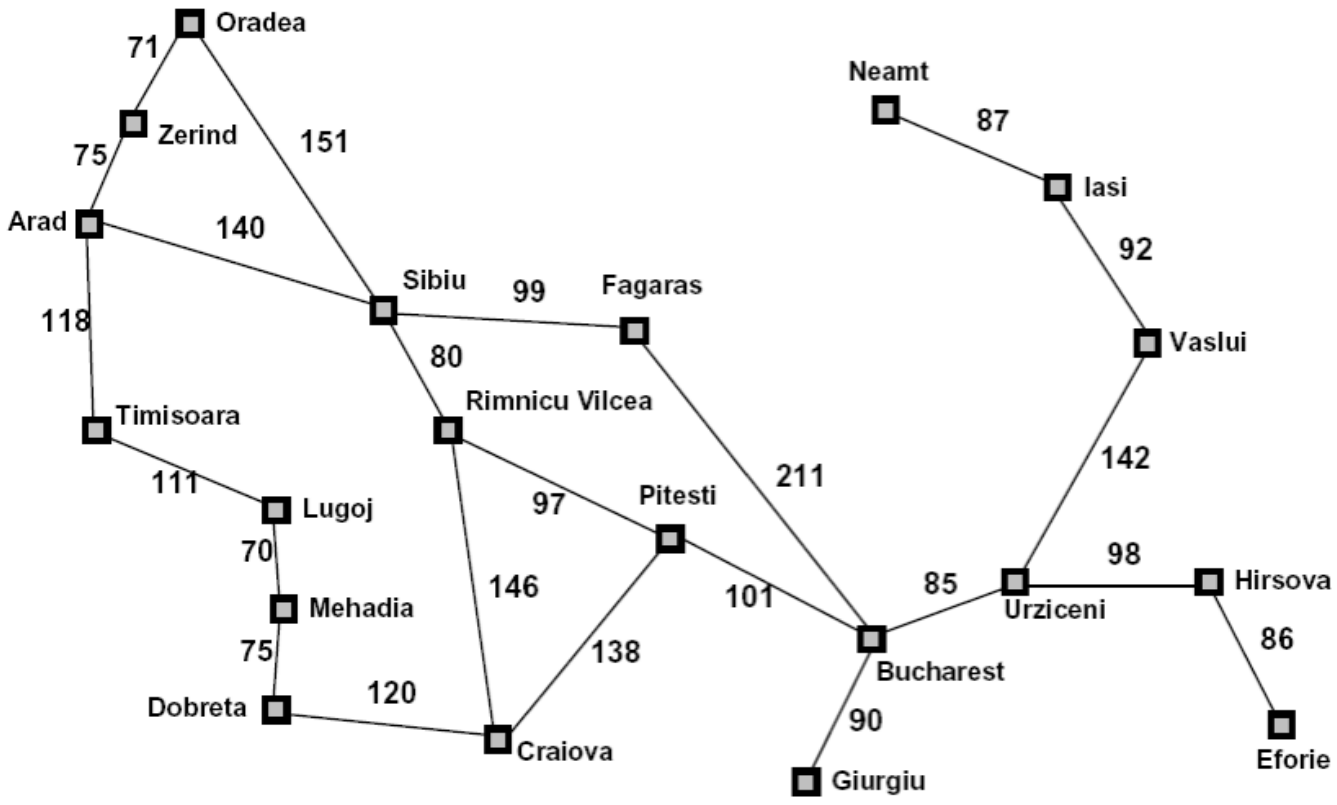

Straight−line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

h(x)

# Greedy Search

# Example: Heuristic Function



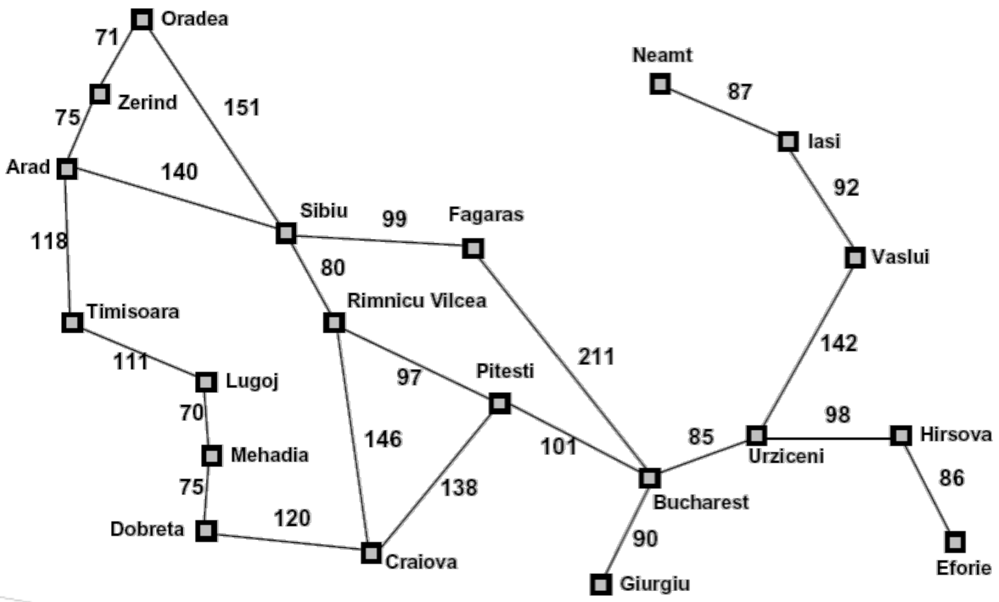

Straight−line distance to Bucharest

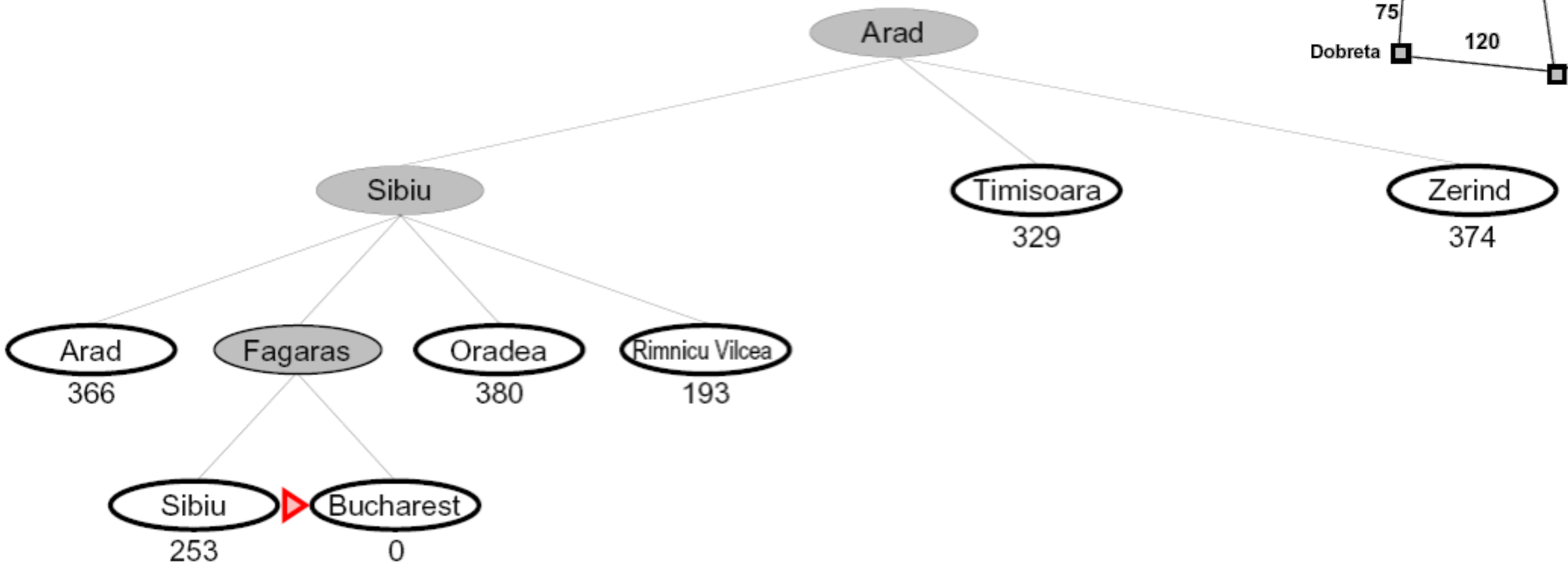
| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

h(x)

# Greedy Search

- Expand the node that seems closest…



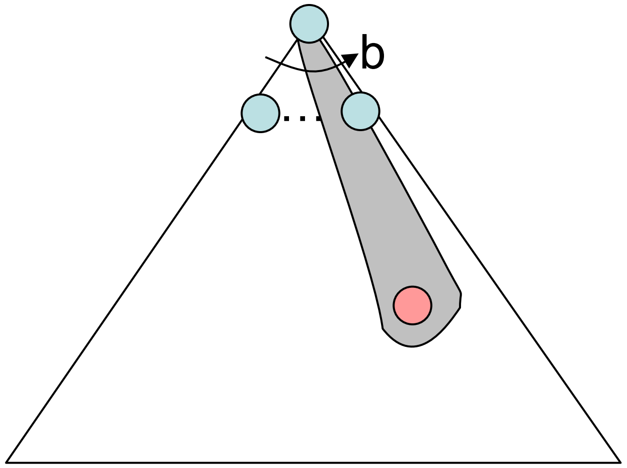

- What can go wrong?

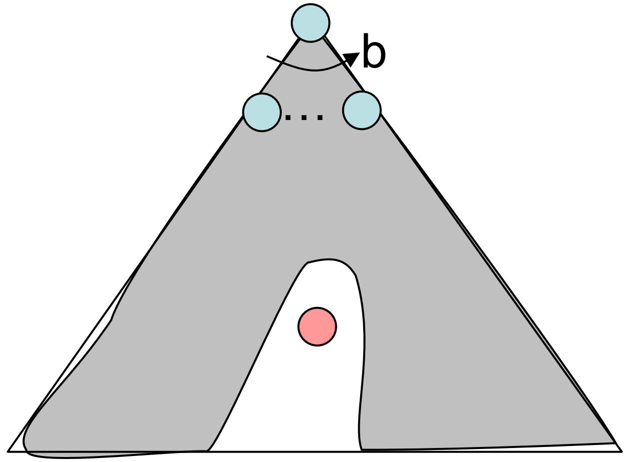
# Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state

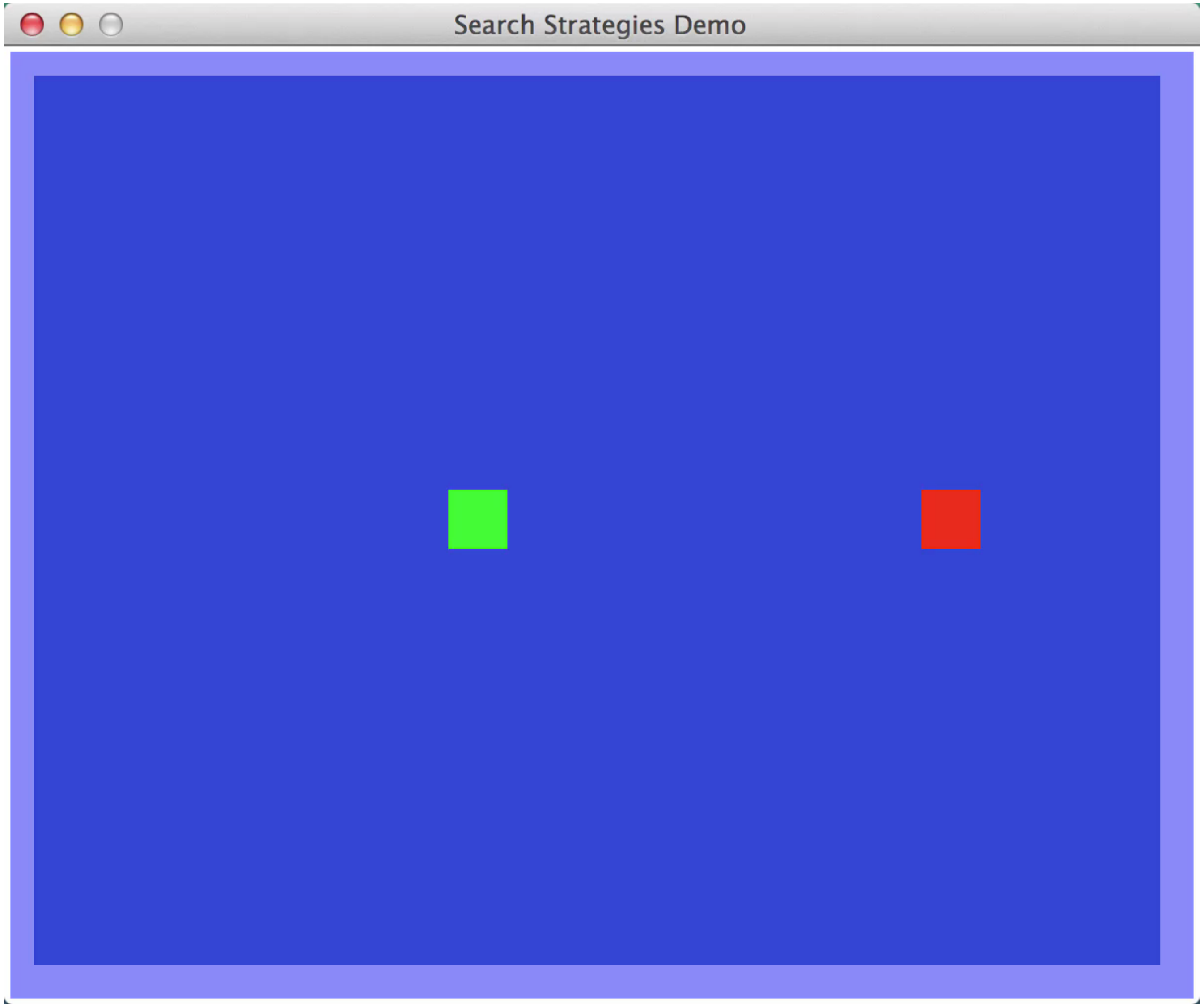
- A common case:
  - Best-first takes you straight to the nearest goal
  - Suboptimal route to goal due to imperfect heuristic
  - Does not lead to nearest goal

- Worst-case: like a badly-guided DFS

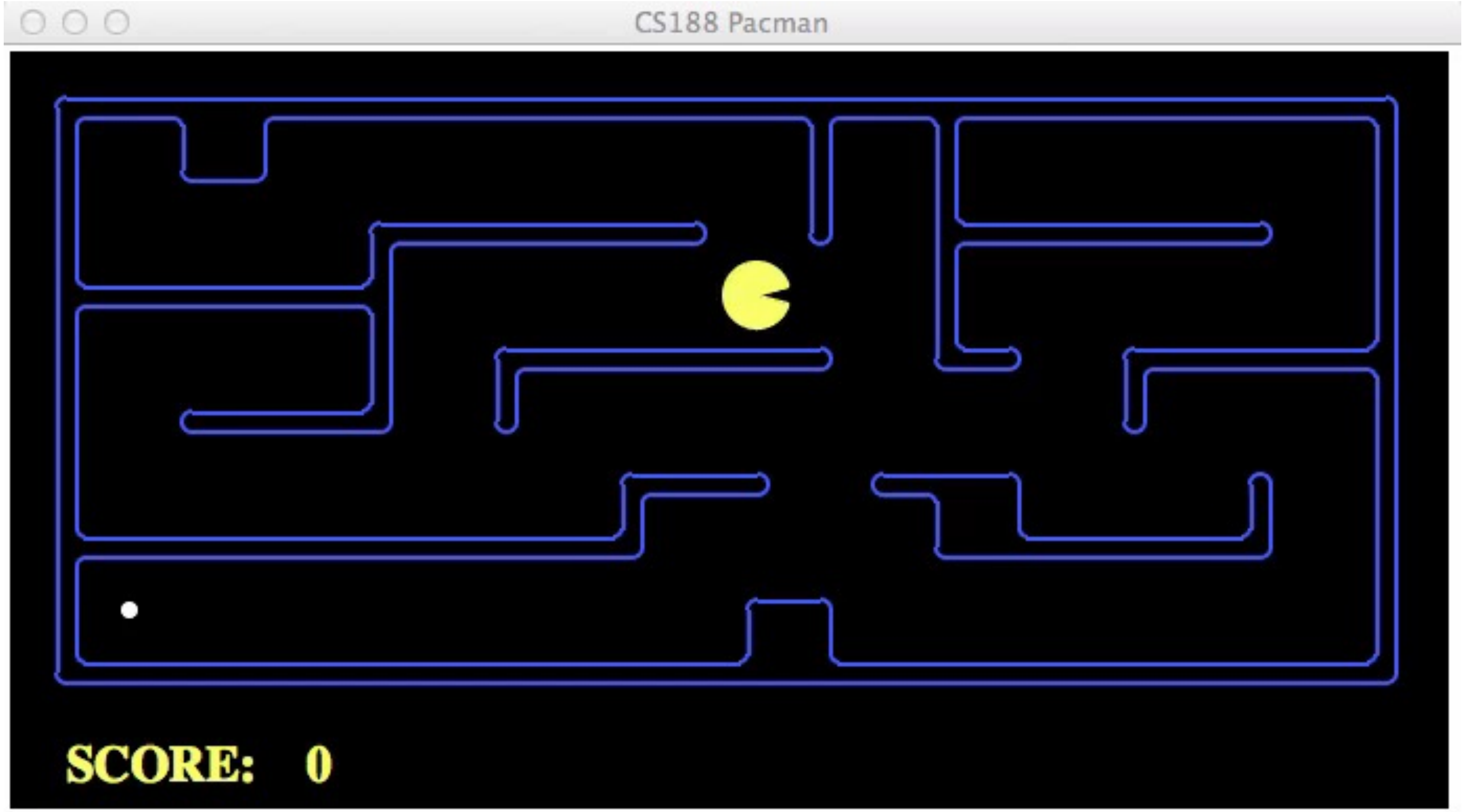


# Video of Demo Contours Greedy (Empty)



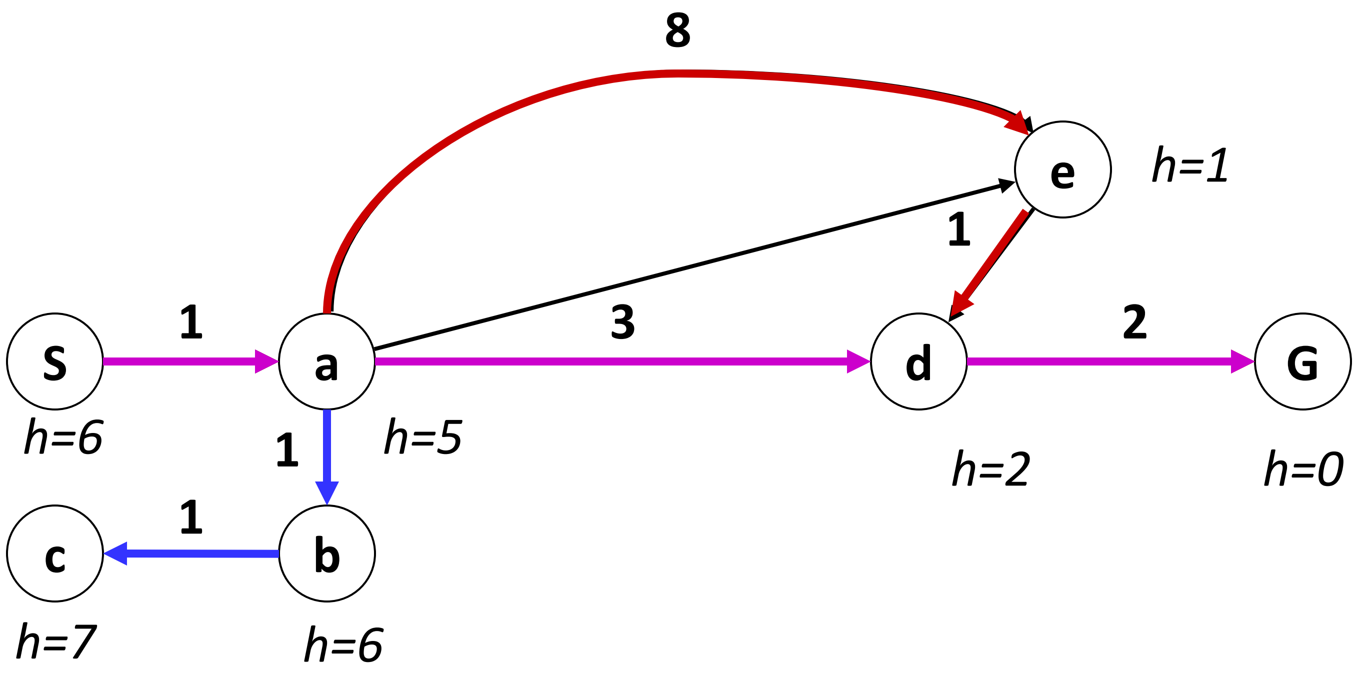
# Video of Demo Contours Greedy (Pacman Small Maze)

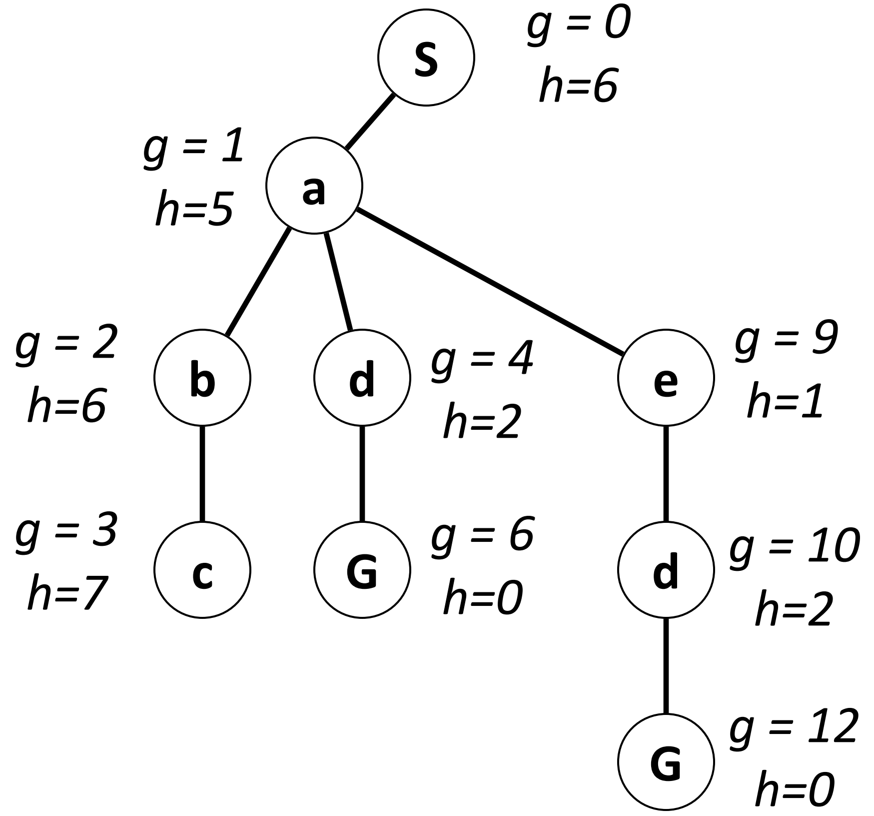


# A* Search

# 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)

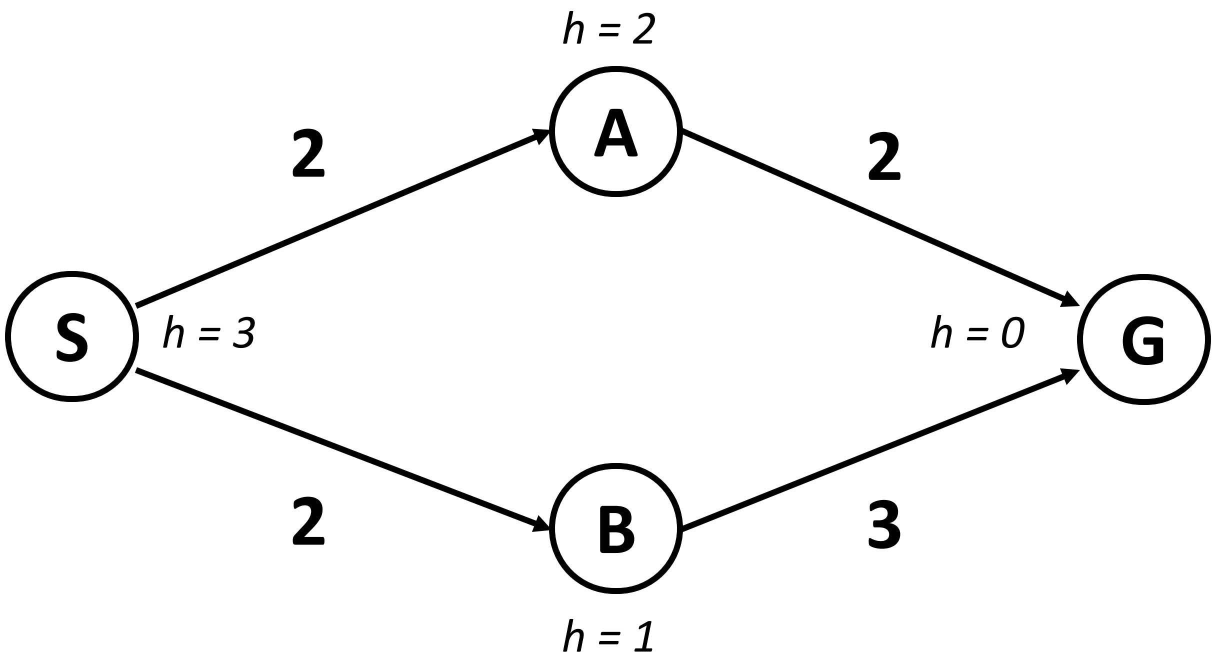



- A* Search orders by the sum: f(n) = g(n) + h(n)

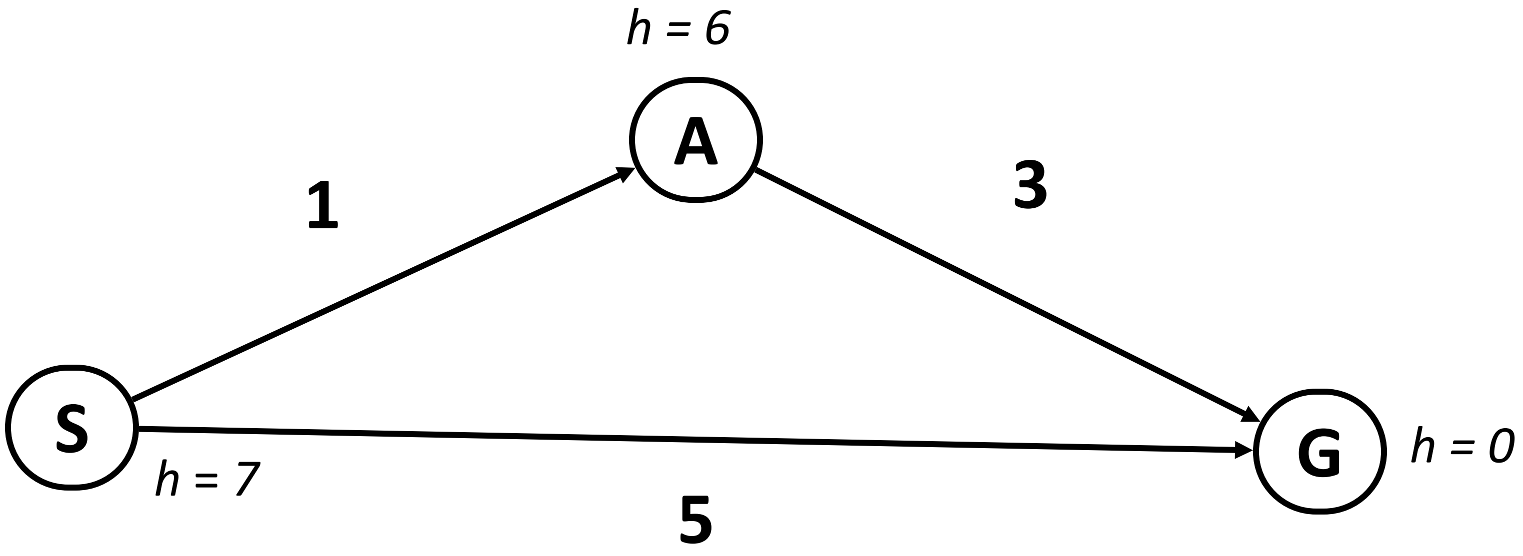
# When should A* terminate?

- Should we stop when we enqueue a goal?



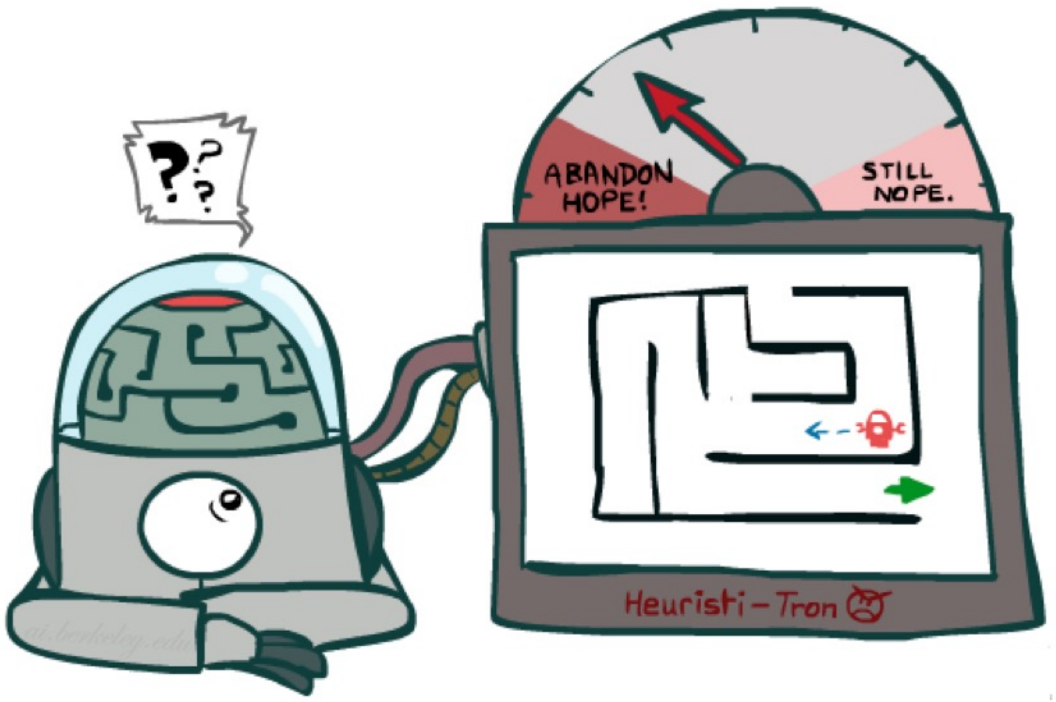

- No: only stop when we expand a goal

# Is A* Optimal?

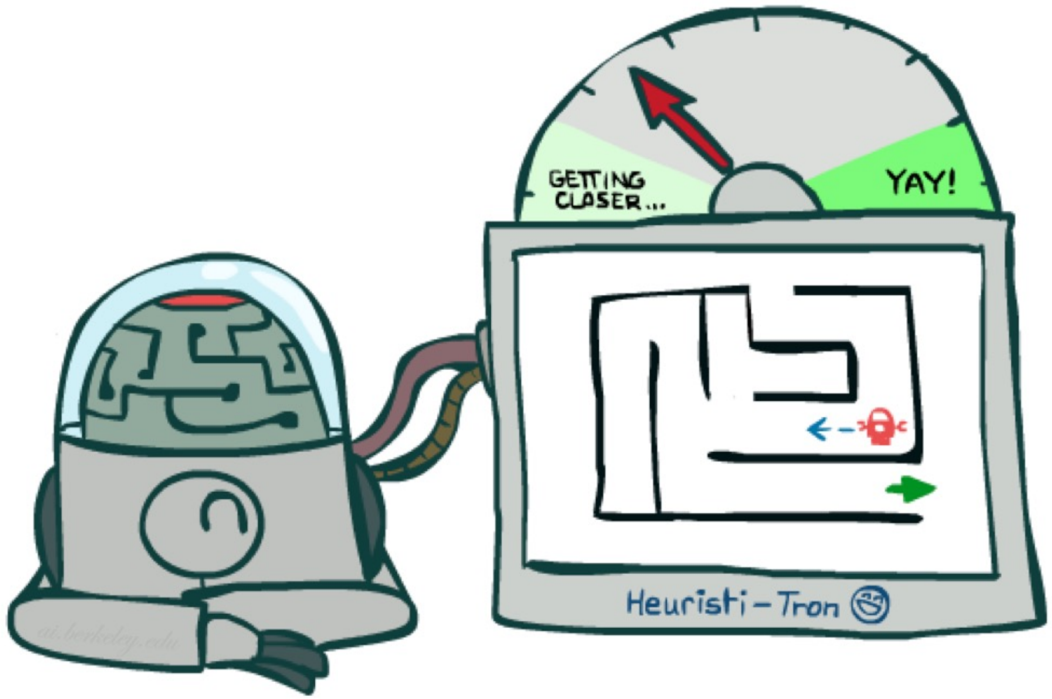



- What will A* do here?
- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

# Idea: Admissibility




Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



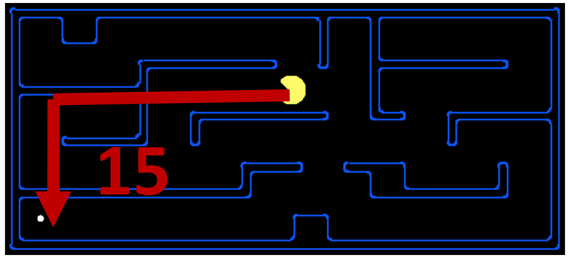

Admissible (optimistic) heuristics can still help to delay the evaluation of bad plans, but never overestimate the true costs

# Admissible Heuristics

- A heuristic $h$ is *admissible* (optimistic) if:

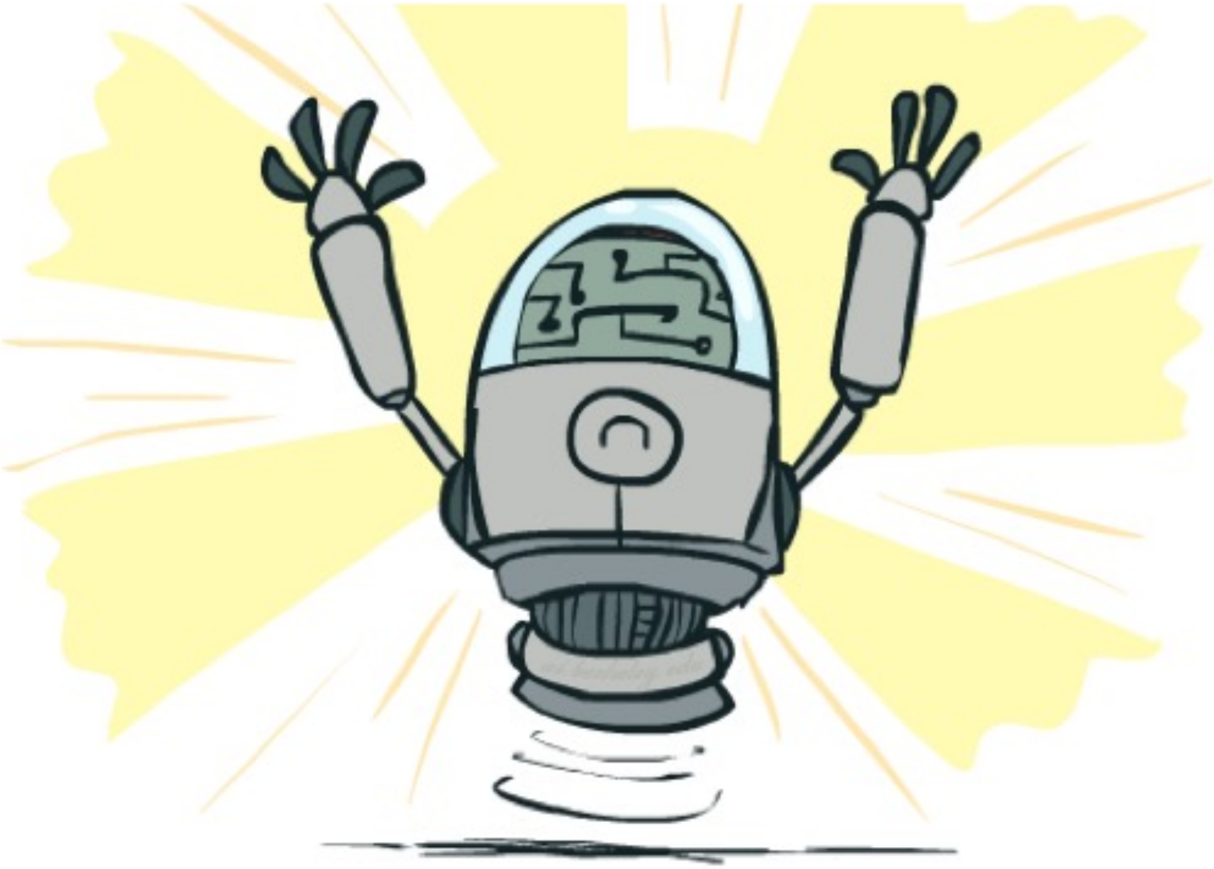
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

- Example: 


- Coming up with admissible heuristics is most of what's involved in using A* in practice.

# Optimality of A* Tree Search



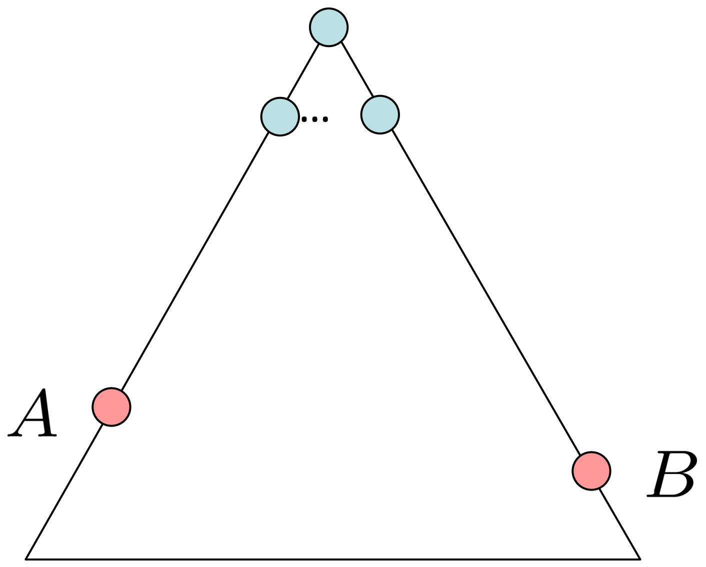
# Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

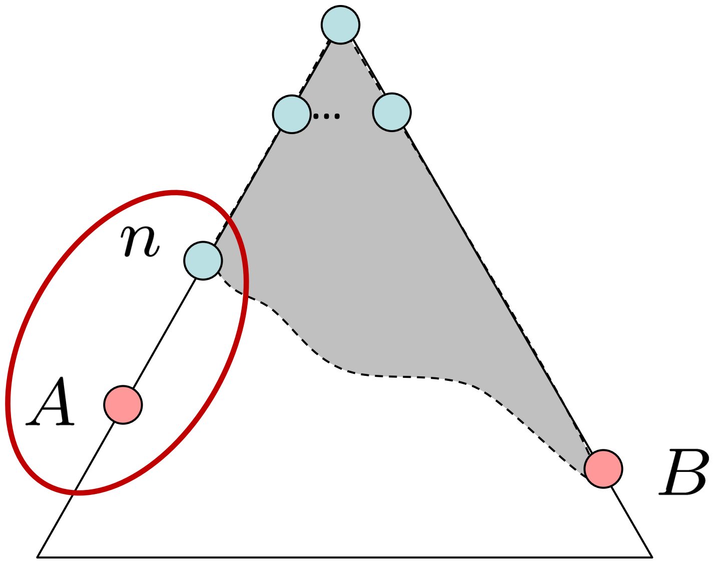
- A will exit the fringe before B



# 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
    1. f(n) is less or equal to f(A)

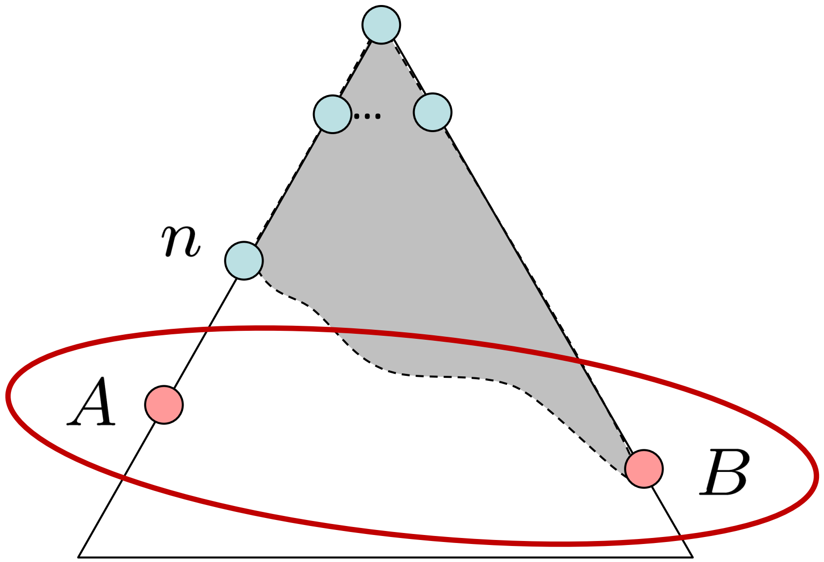



| | |
|---|---|
| $f(n) = g(n) + h(n)$ | Definition of f-cost |
| $f(n) \leq g(A)$ | Admissibility of h |
| $g(A) = f(A)$ | h = 0 at a goal |

# 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
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



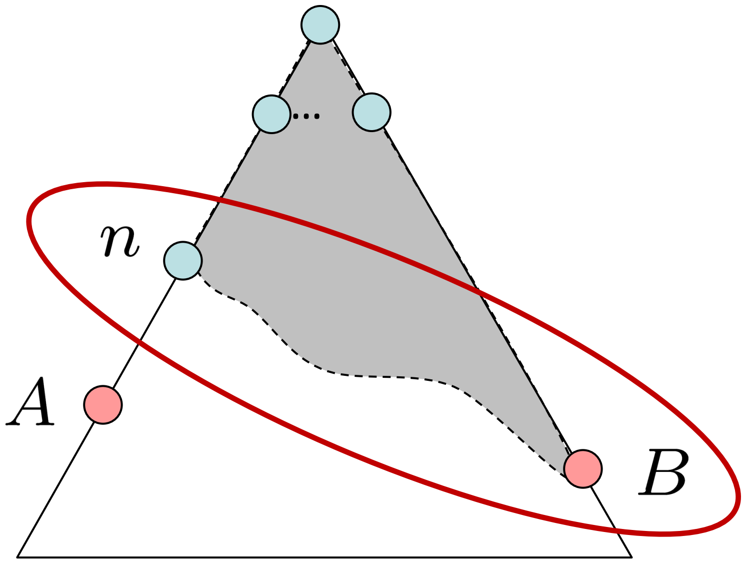

$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

# 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
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

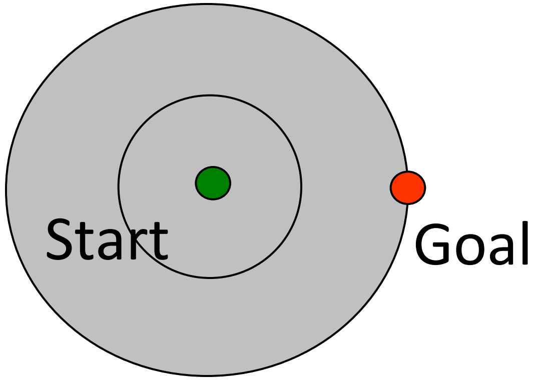



$$f(n) \leq f(A) < f(B)$$

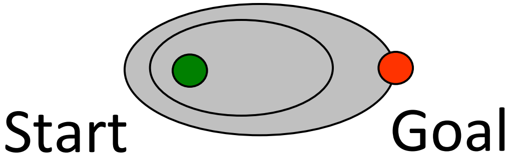
# Properties of A*

# UCS vs A* Contours

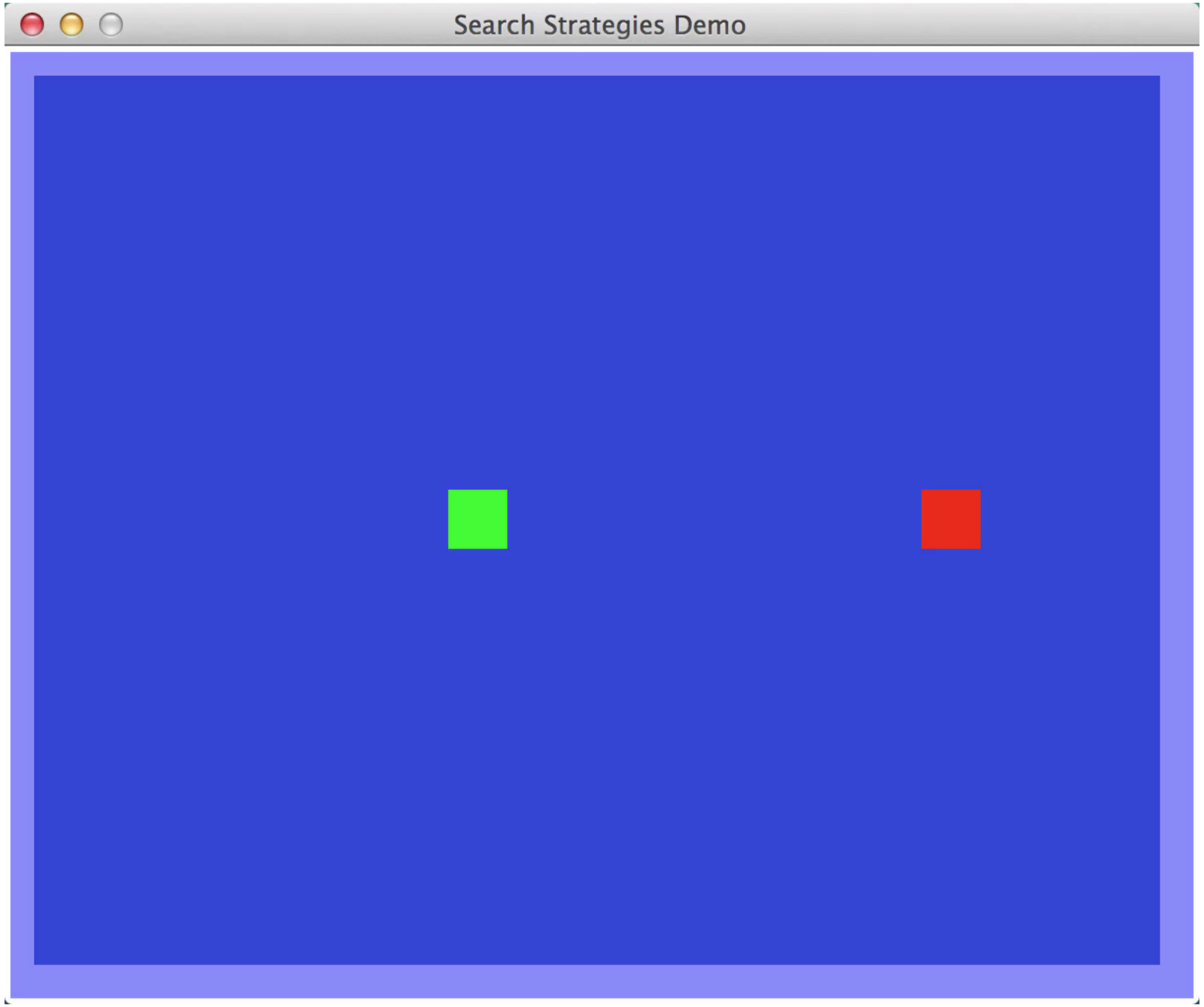
- Uniform-cost expands equally in all "directions"



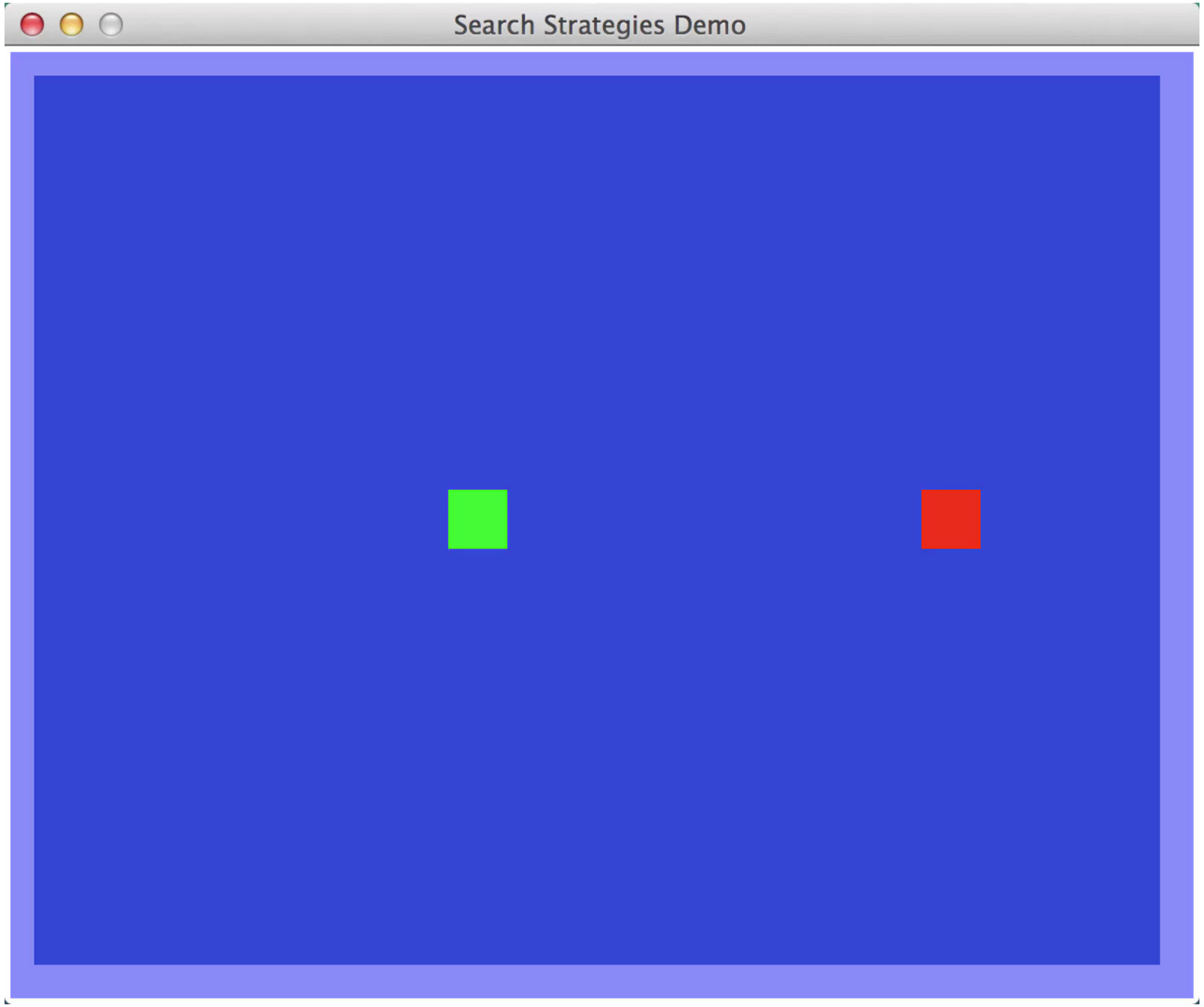

- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



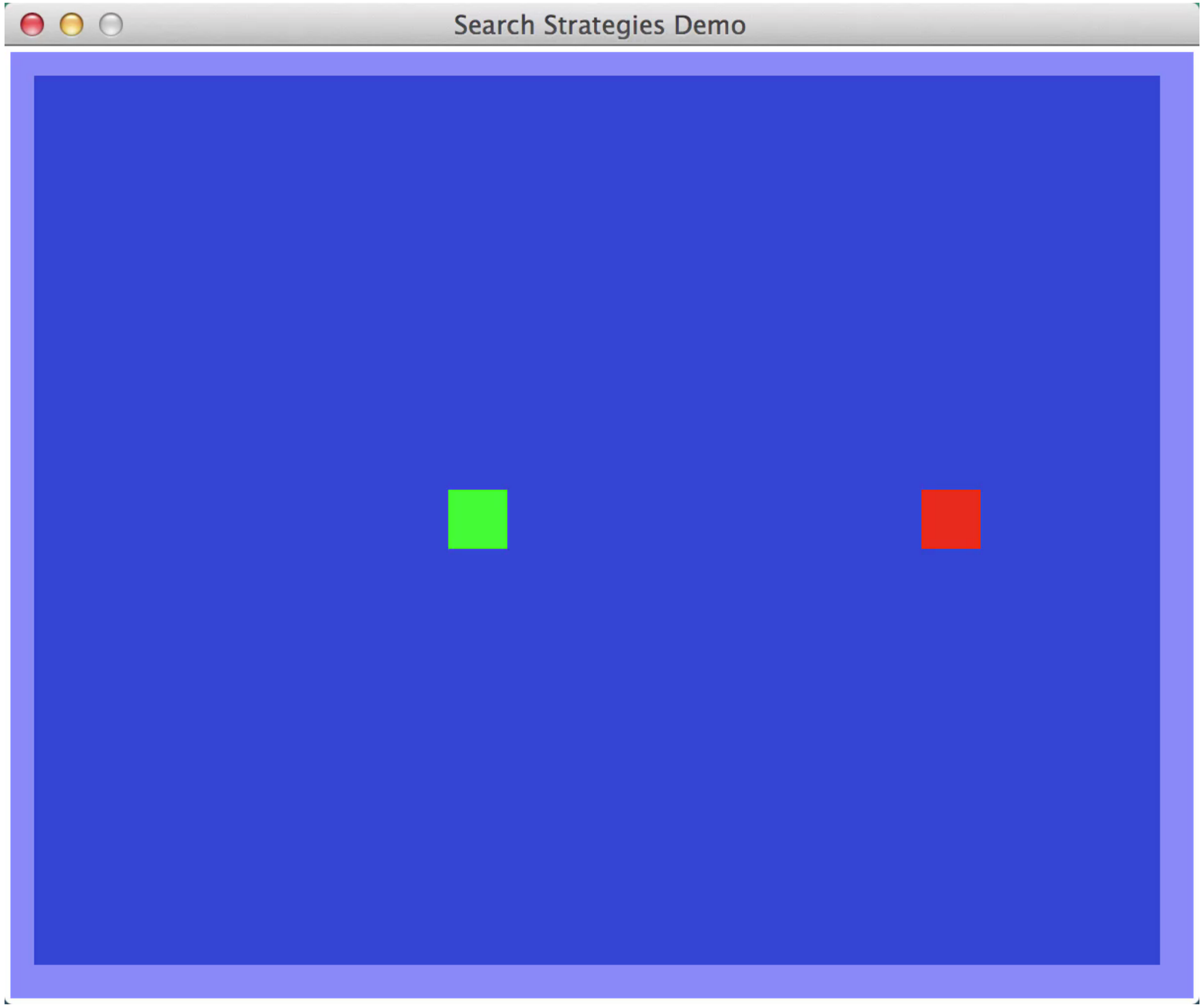
# Video of Demo Contours (Empty) -- UCS



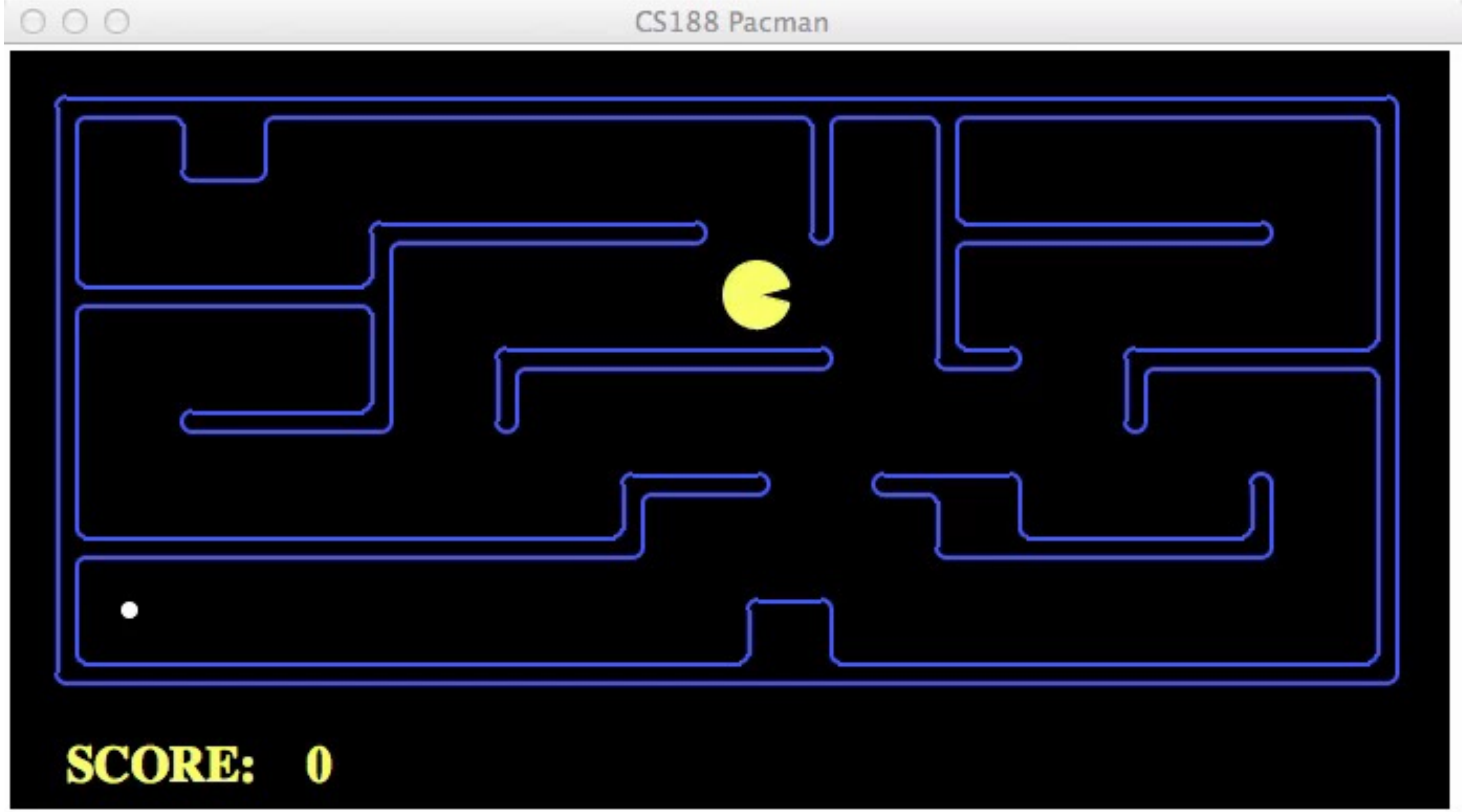
# Video of Demo Contours (Empty) -- Greedy



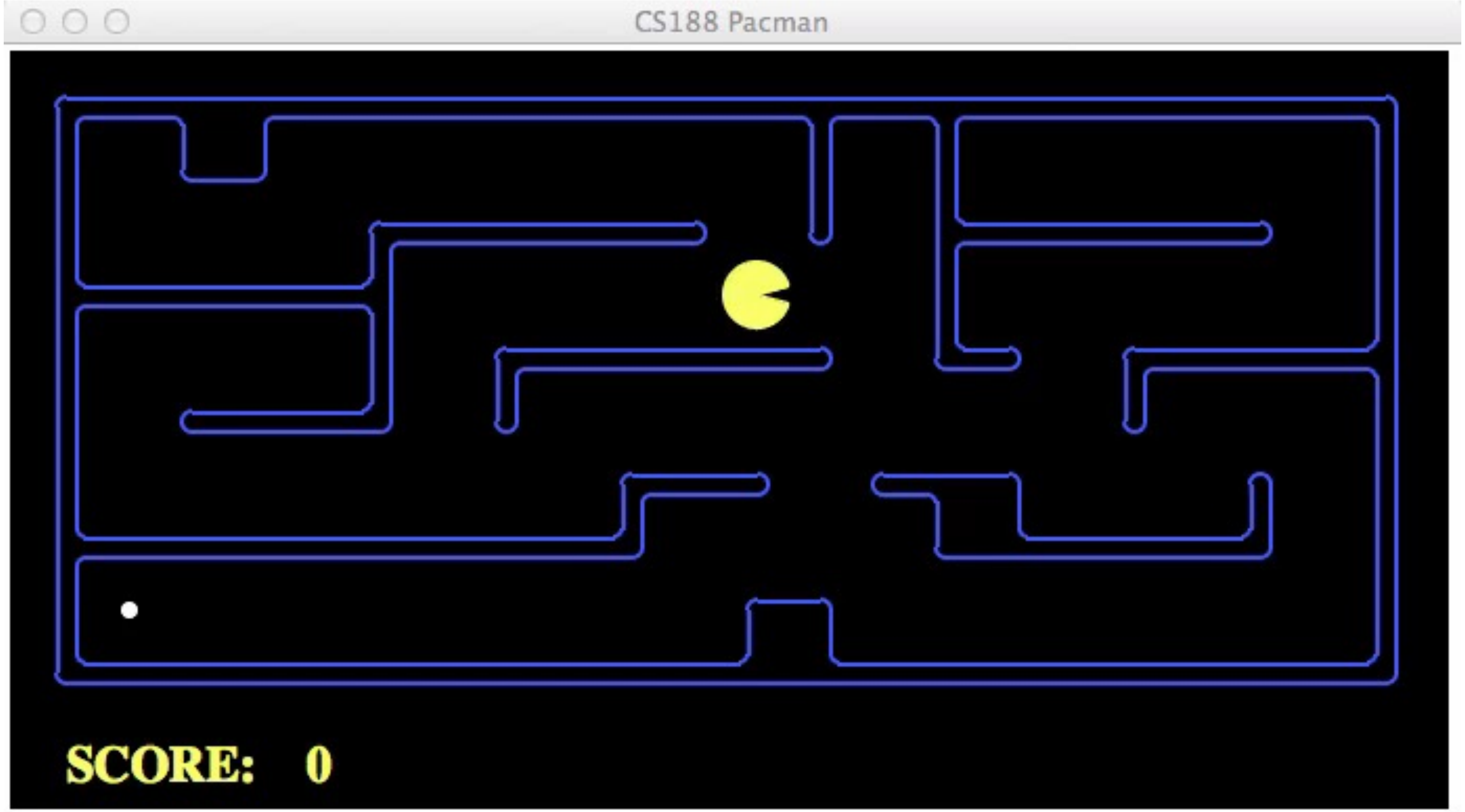
# Video of Demo Contours (Empty) – A*



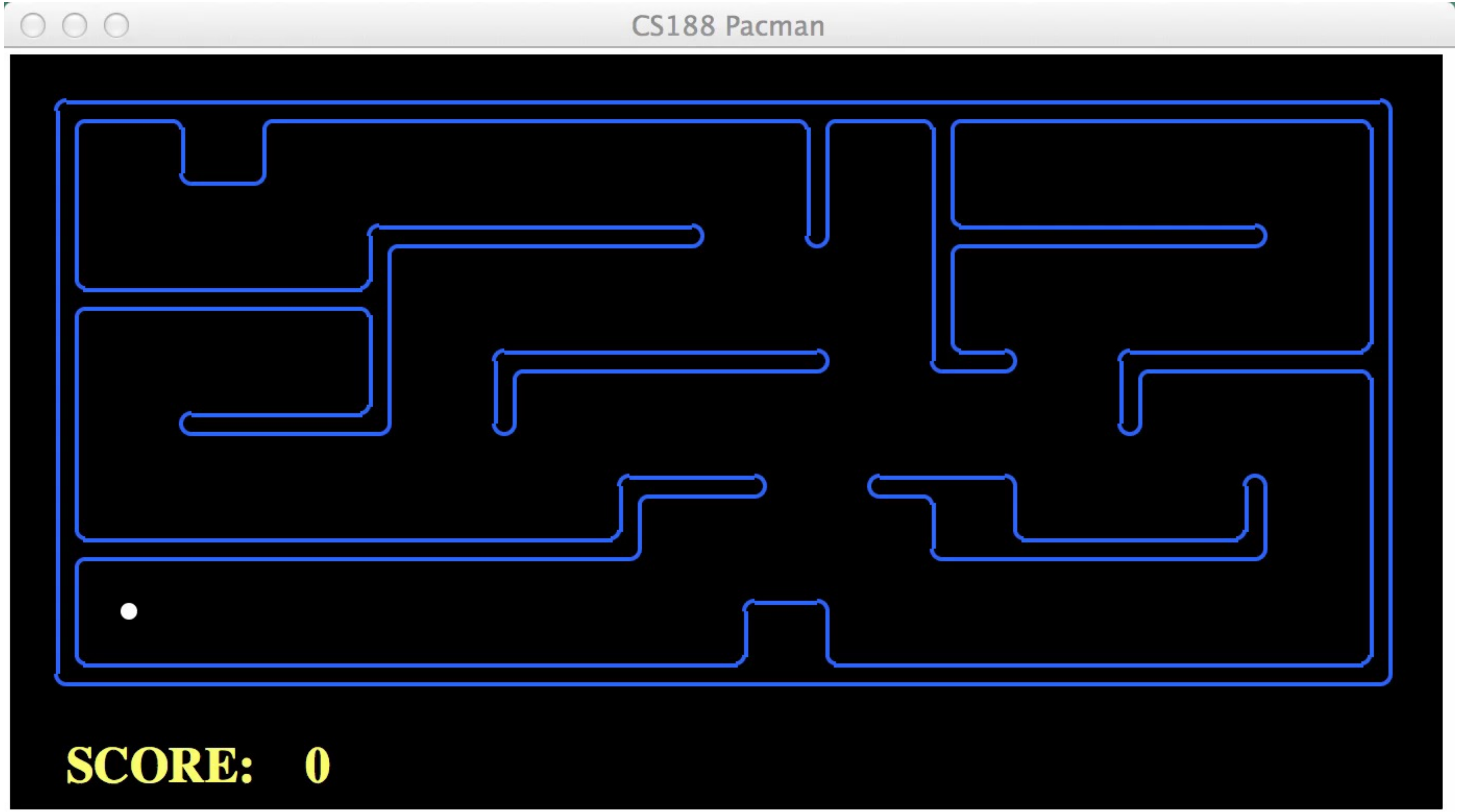
# Pacman - A*



# Pacman - Greedy



# Pacman - UCS



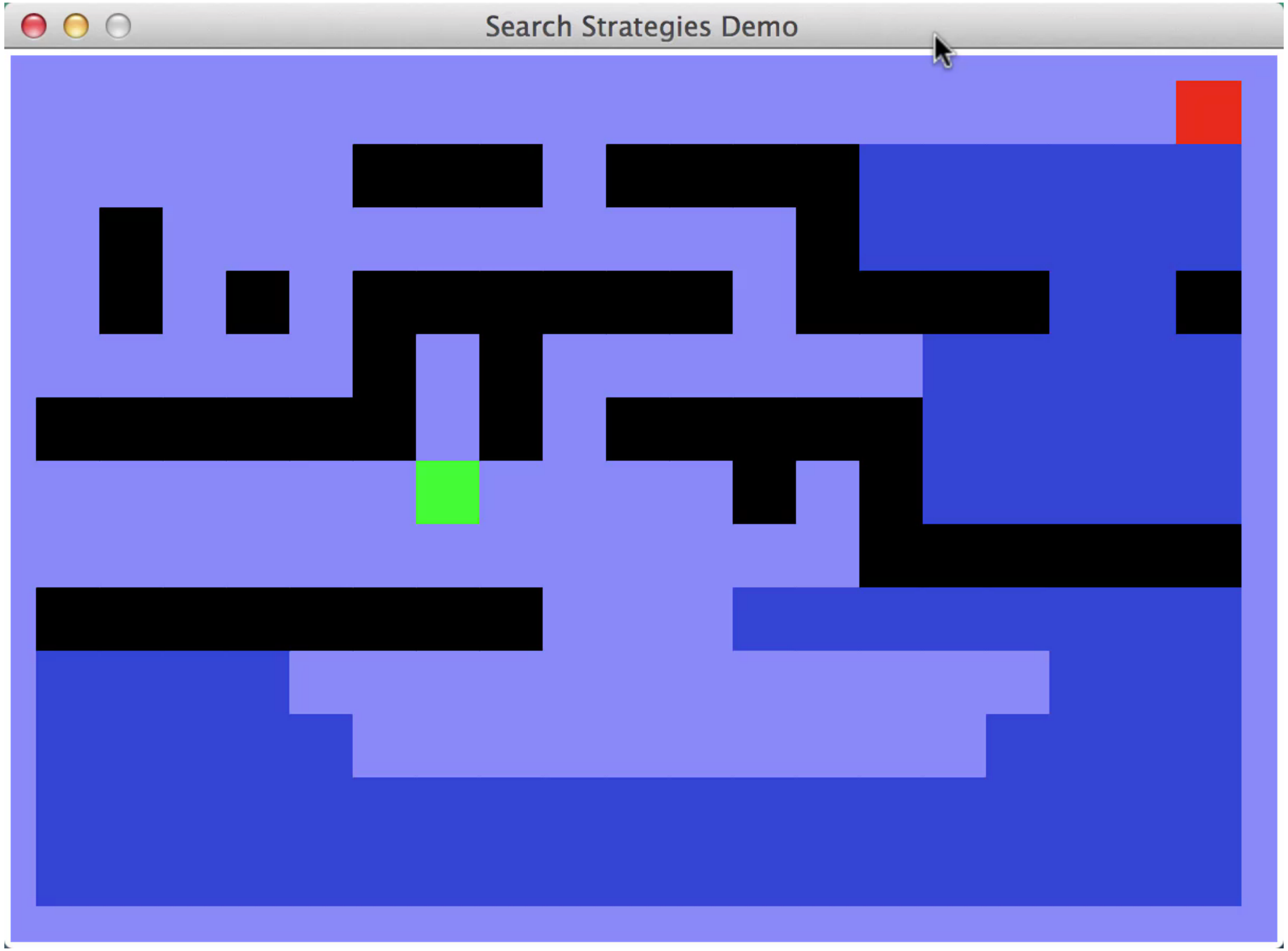
# Comparison


Greedy

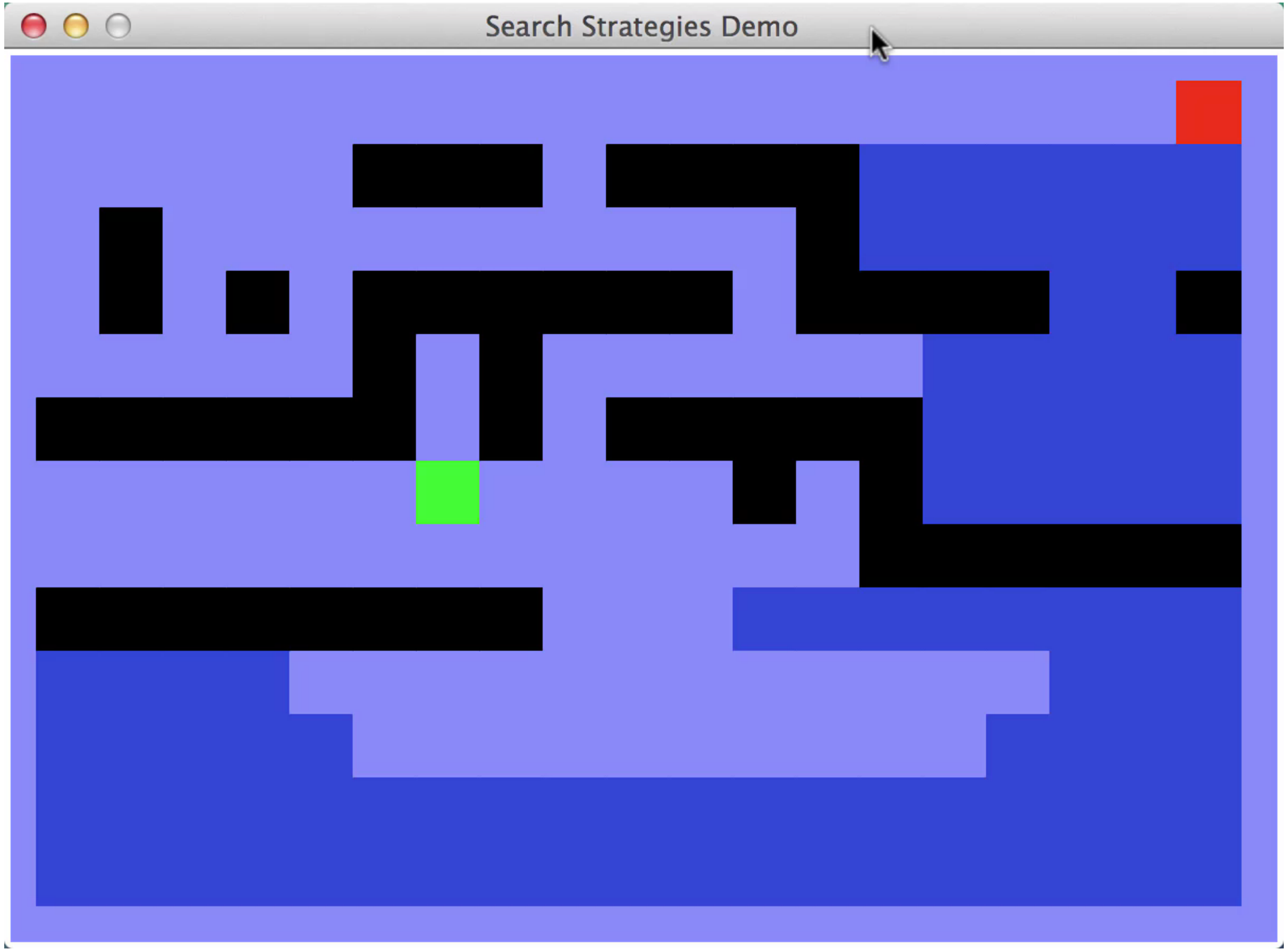

Uniform Cost


A*

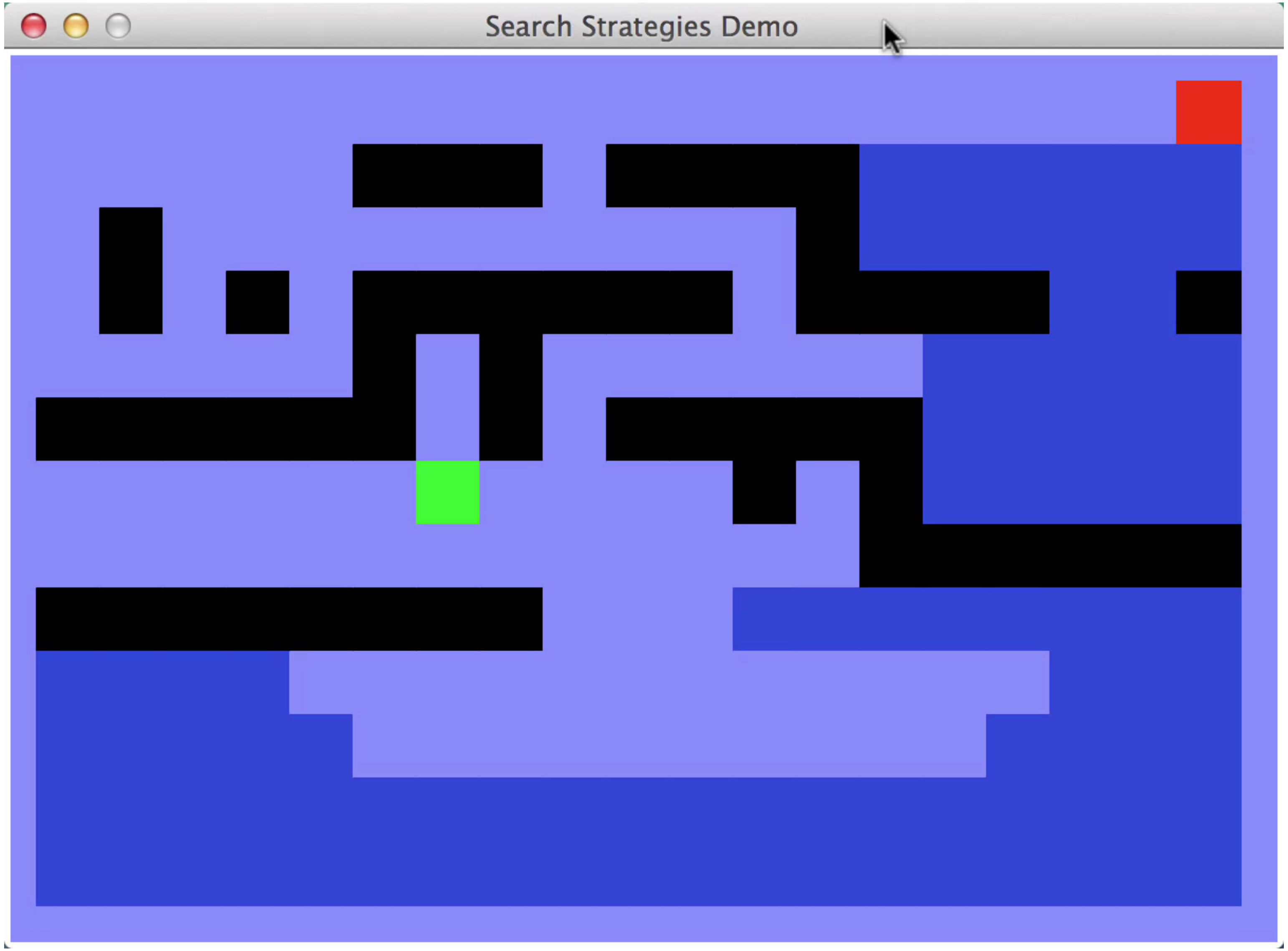
# Guess algorithm (DFS / BFS / UCS / Greedy / A*)



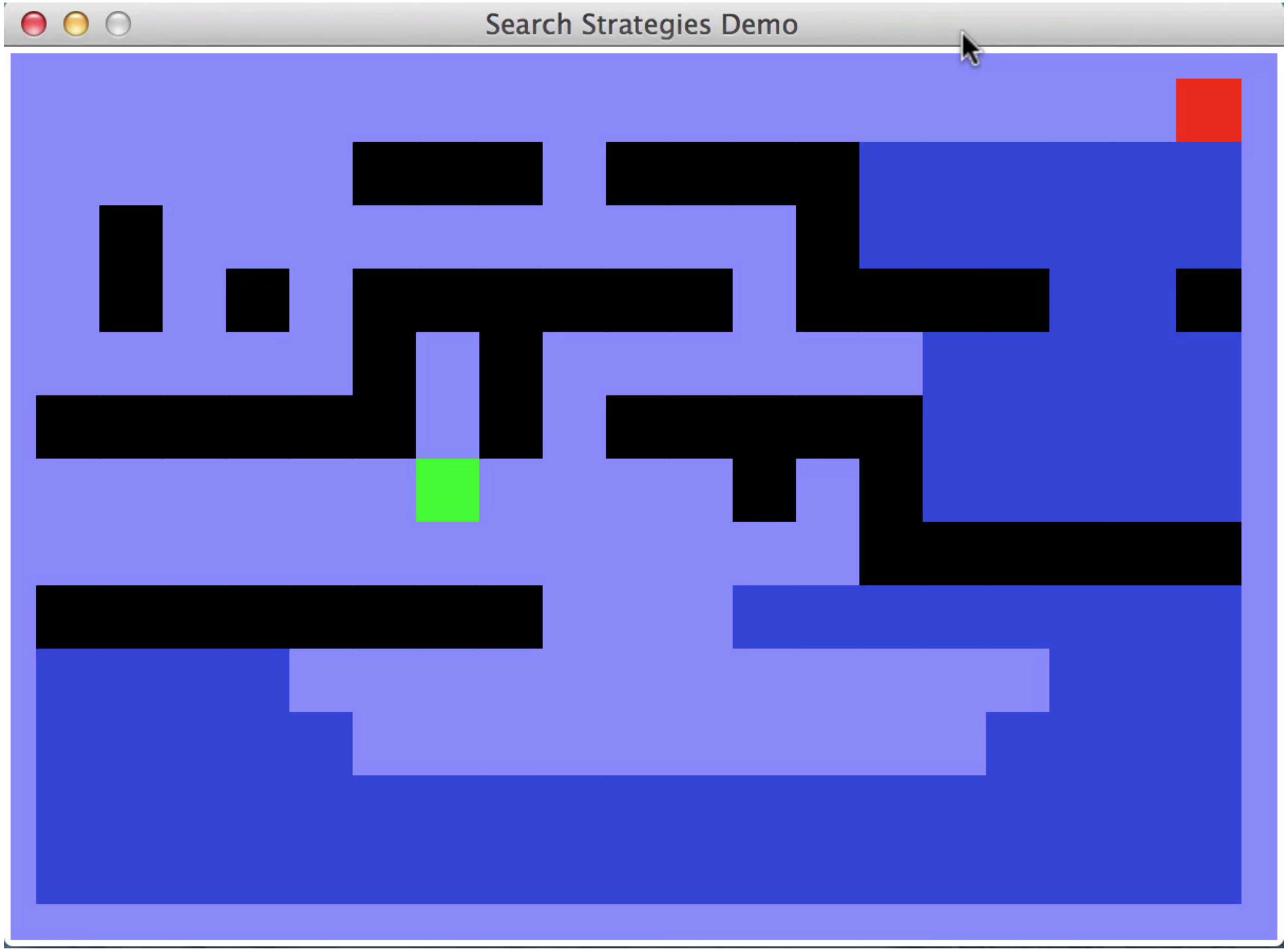
# Guess algorithm (DFS / BFS / UCS / Greedy / A*)



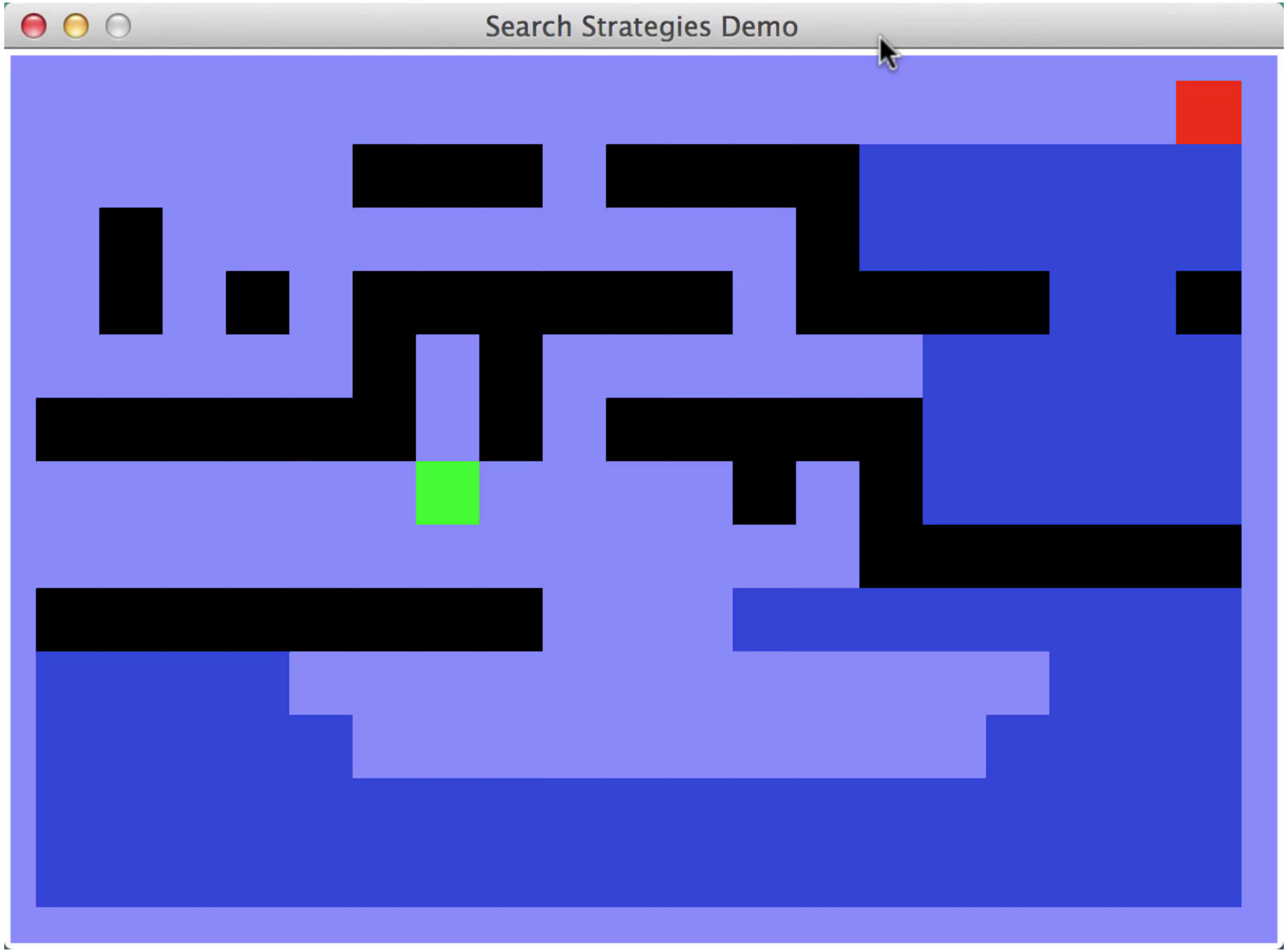
# Guess algorithm (DFS / BFS / UCS / Greedy / A*)



# Guess algorithm (DFS / BFS / UCS / Greedy / A*)



# Guess algorithm (DFS / BFS / UCS / Greedy / A*)



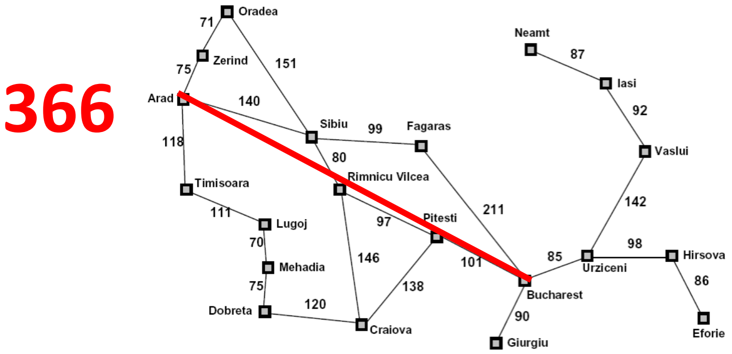
# A* Applications

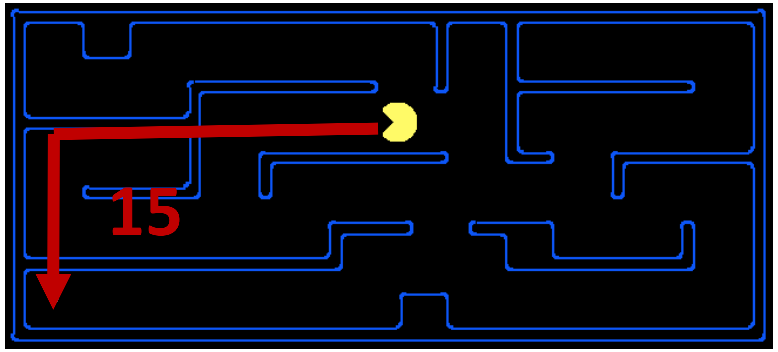
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

# Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

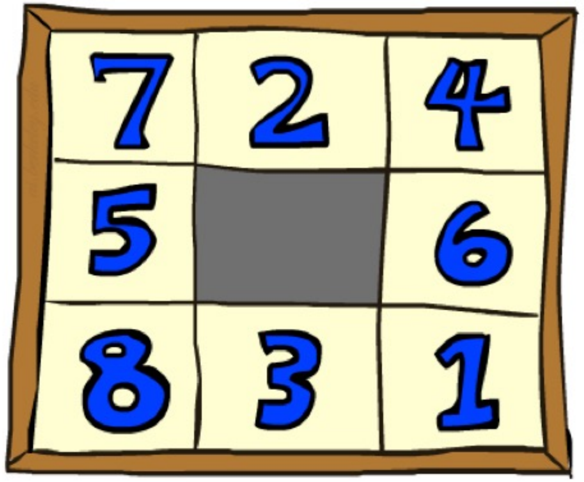
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



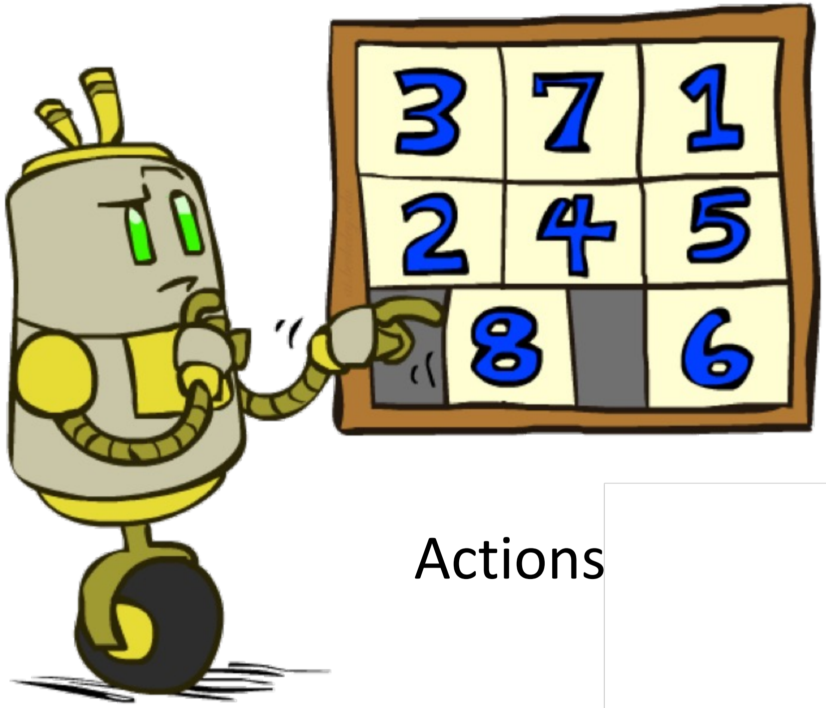




- Inadmissible heuristics are often useful too

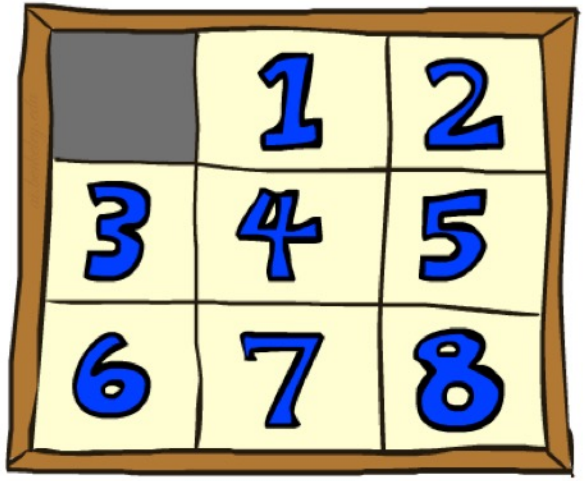
# Example: 8 Puzzle




Start State




Actions

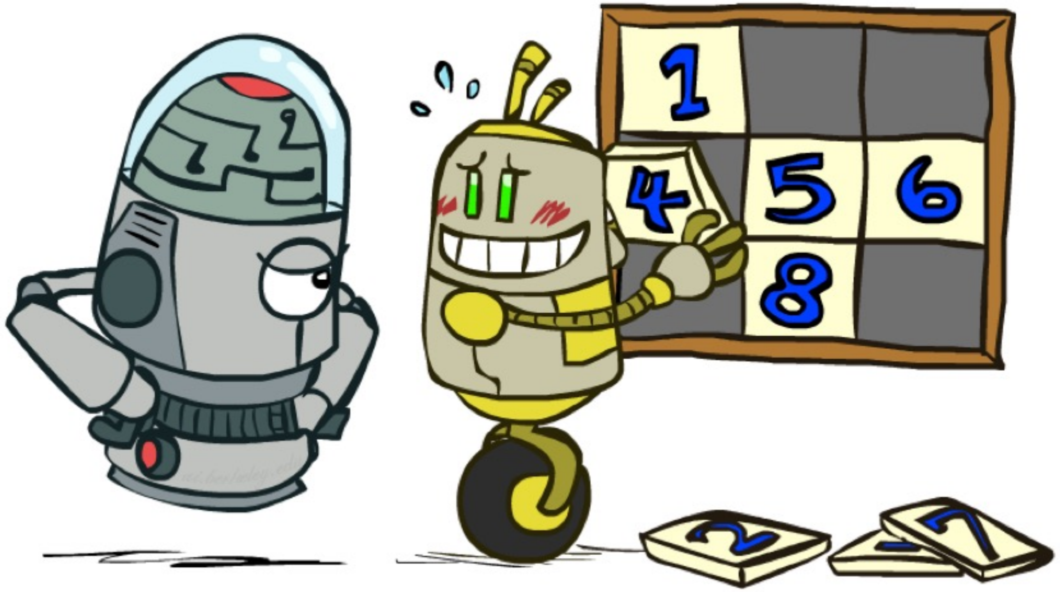


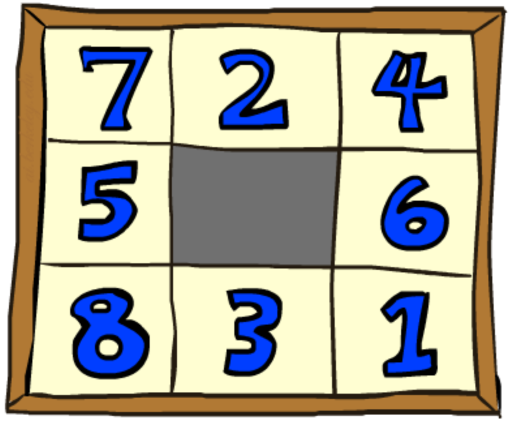

Goal State

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the heuristic be?

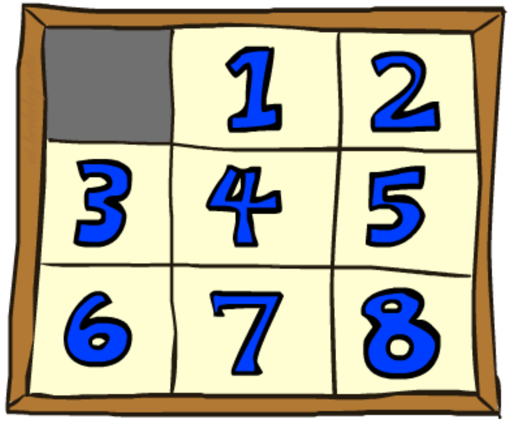
# 8 Puzzle I

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




Start State




Goal State

| | Average nodes expanded when the optimal path has… | | |
|---|---|---|---|
| | …4 steps | …8 steps | …12 steps |
| UCS | 112 | 6,300 | $3.6 \times 10^6$ |
| TILES | 13 | 39 | 227 |

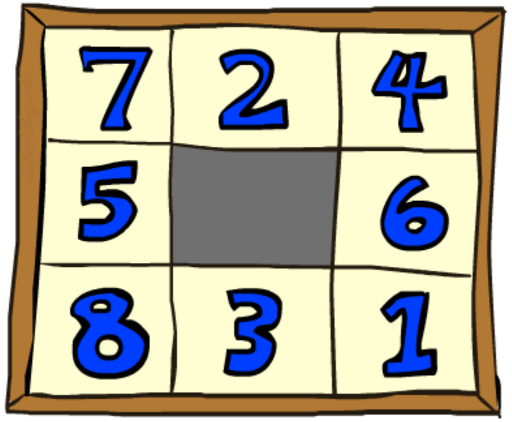
# 8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

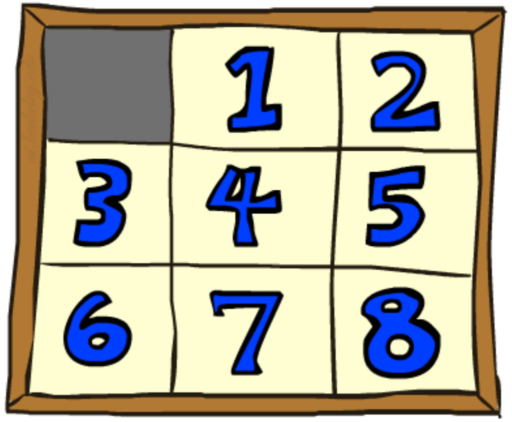
- Total *Manhattan* distance

- Why is it admissible?

- h(start) = 3 + 1 + 2 + … = 18




Start State

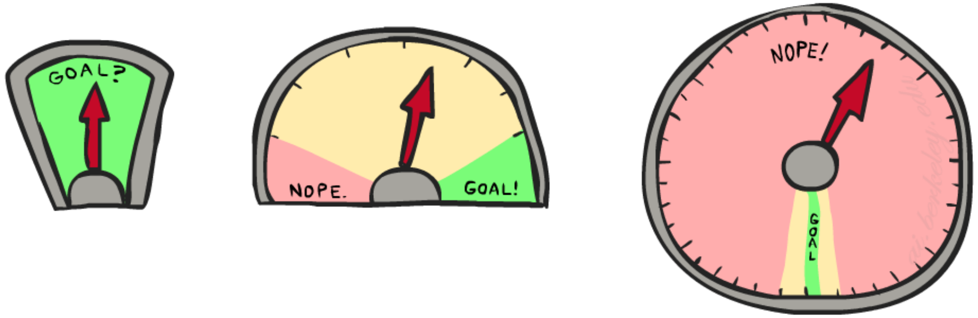



Goal State

| | Average nodes expanded when the optimal path has… | | |
|---|---|---|---|
| | …4 steps | …8 steps | …12 steps |
| TILES | 13 | 39 | 227 |
| MANHATTAN | 12 | 25 | 73 |

# 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

# Trivial Heuristics, Dominance

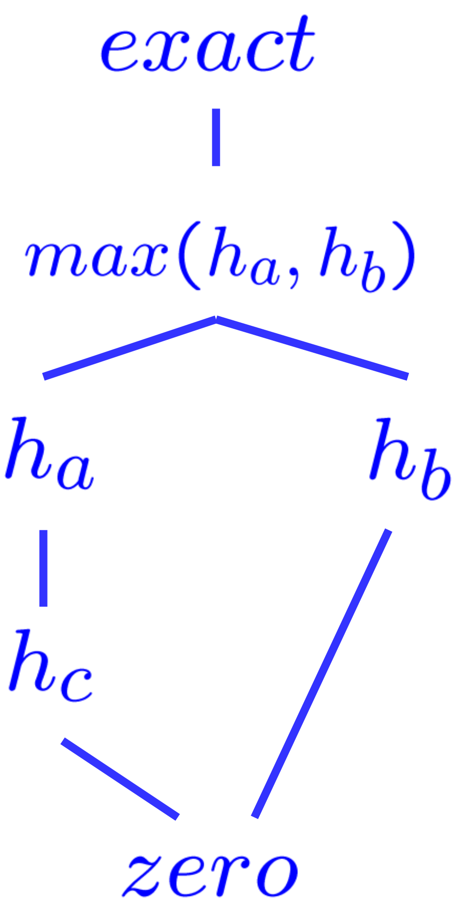
- Dominance: $h_a \geq h_c$ if

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

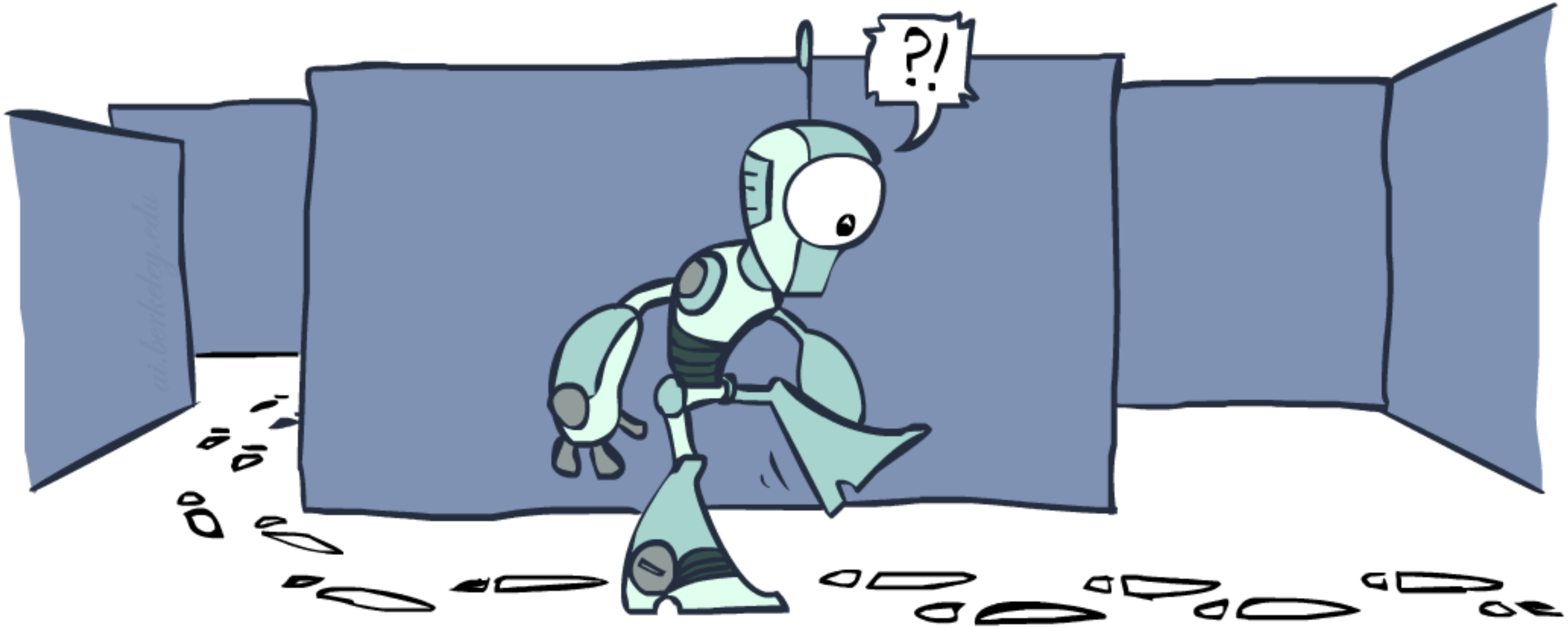
- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

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

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic

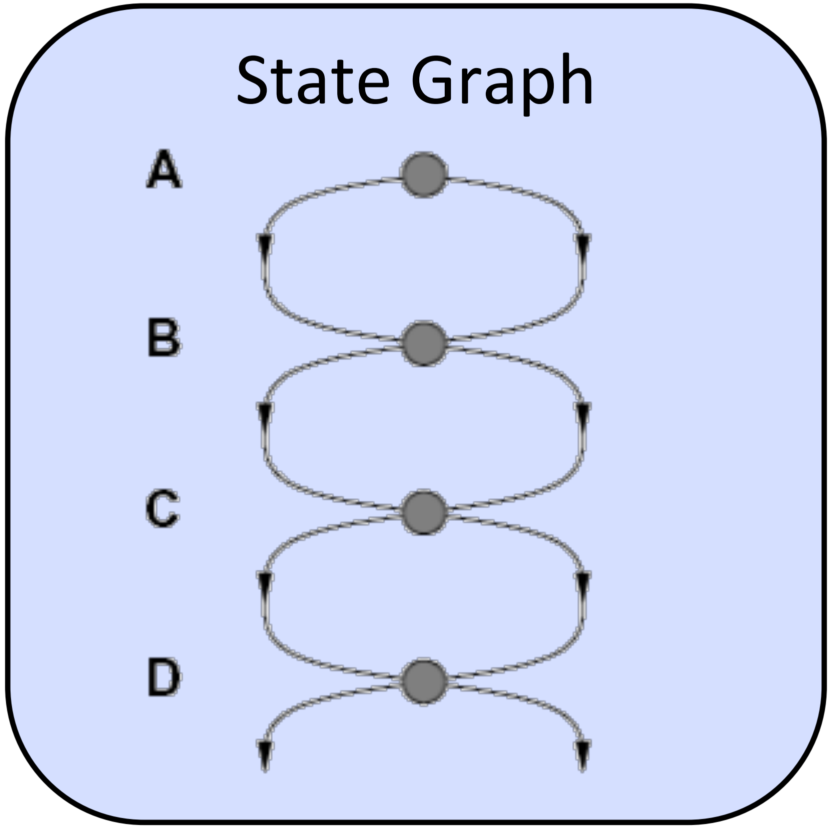


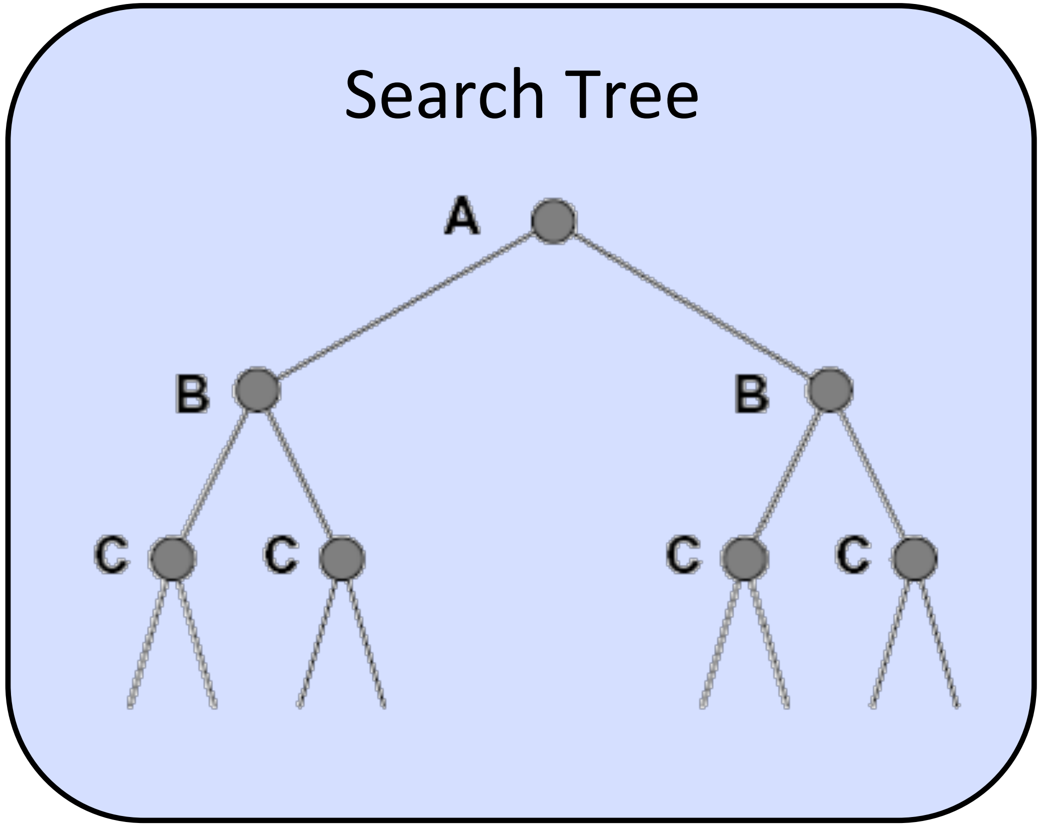
# Graph Search



# Tree Search: Extra Work!

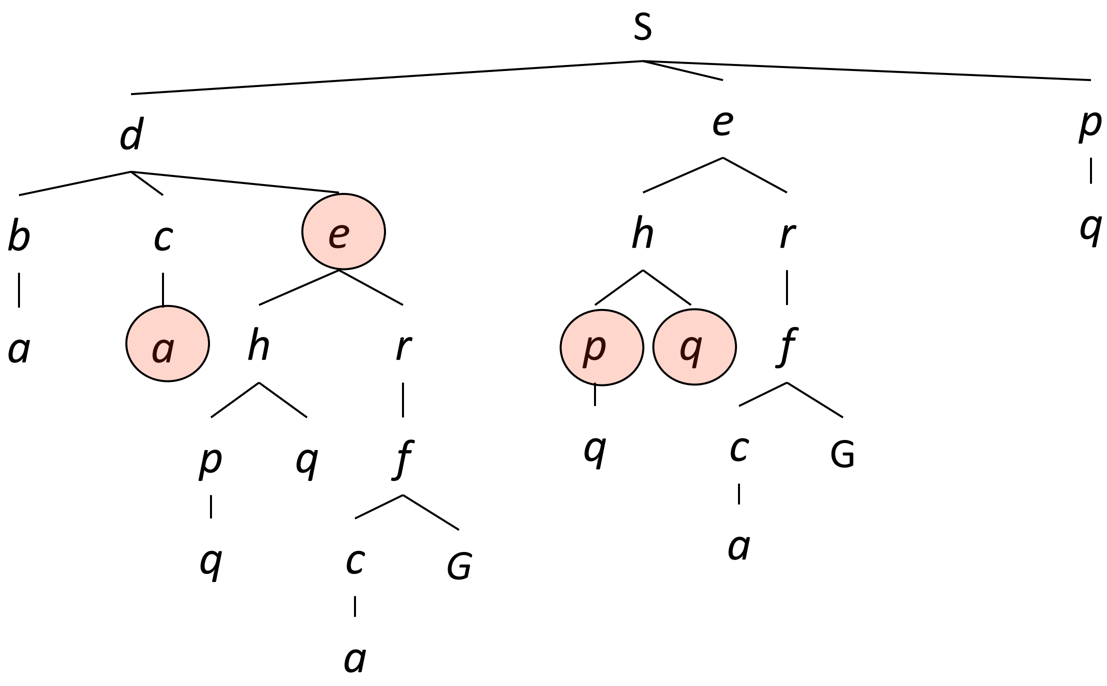
- Failure to detect repeated states can cause exponentially more work.



# Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

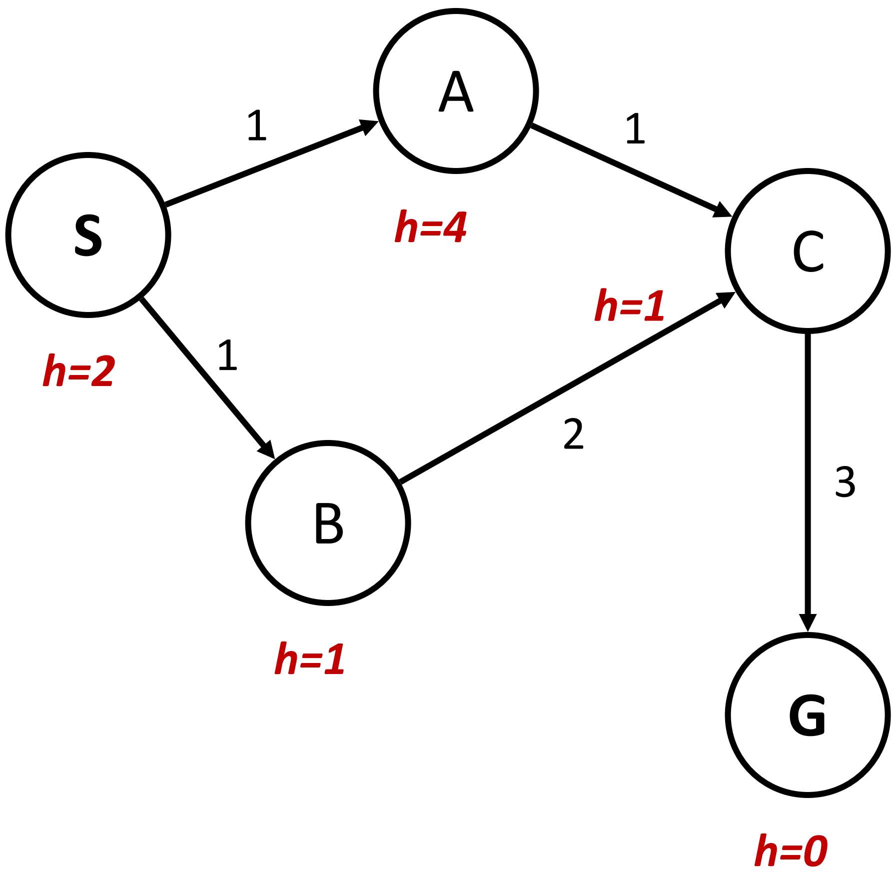


# Graph Search

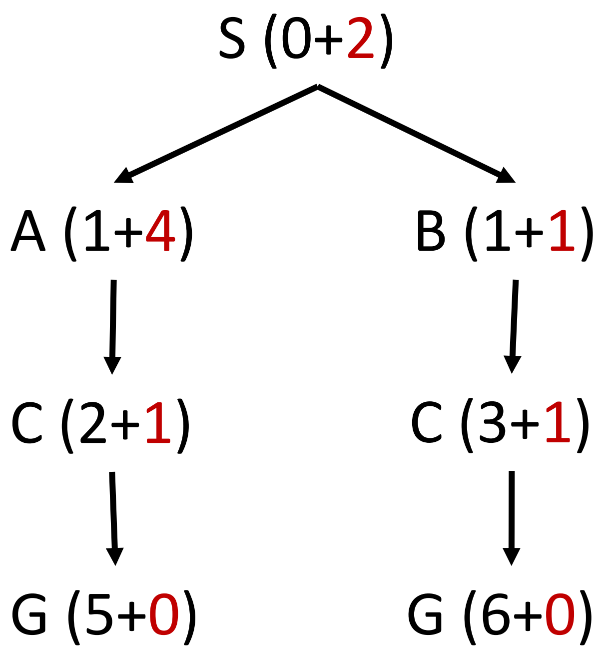
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: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

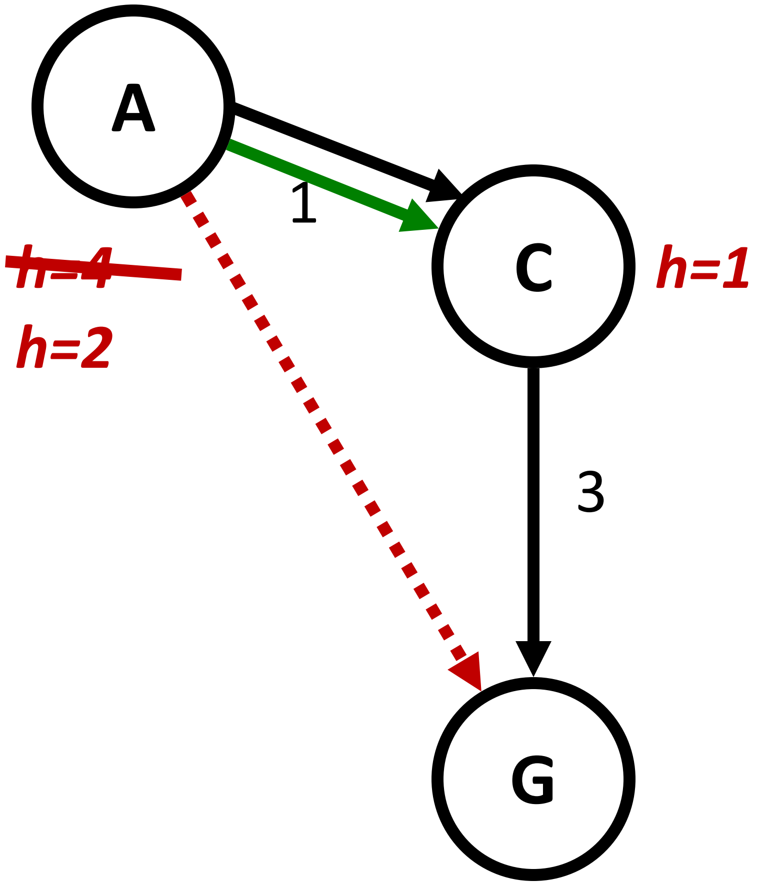
State space graph




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)

  i.e. if the true cost of an edge from A to C is X, then the h-value should not decrease by more than X between A and C.

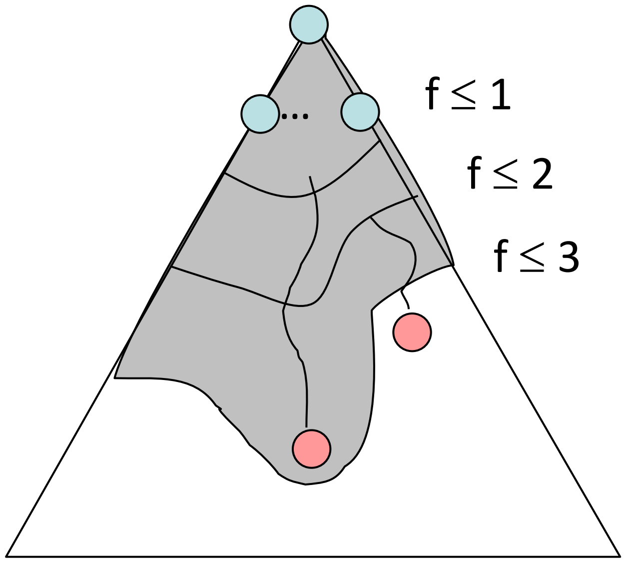
- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)

  - A* graph search is optimal

# Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal



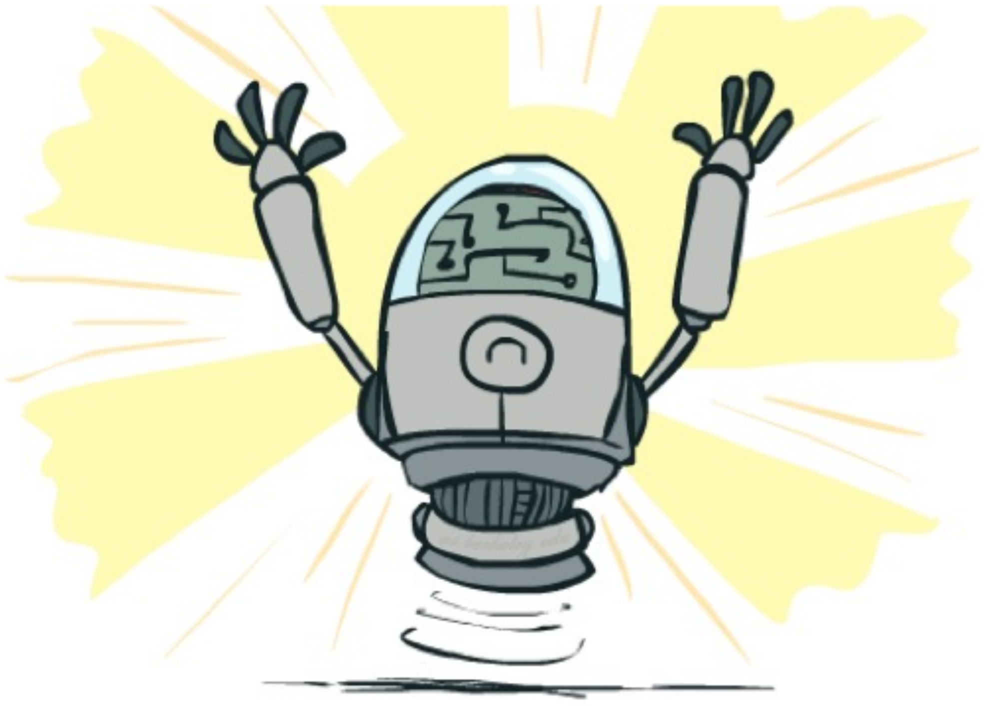
# Optimality

- 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)

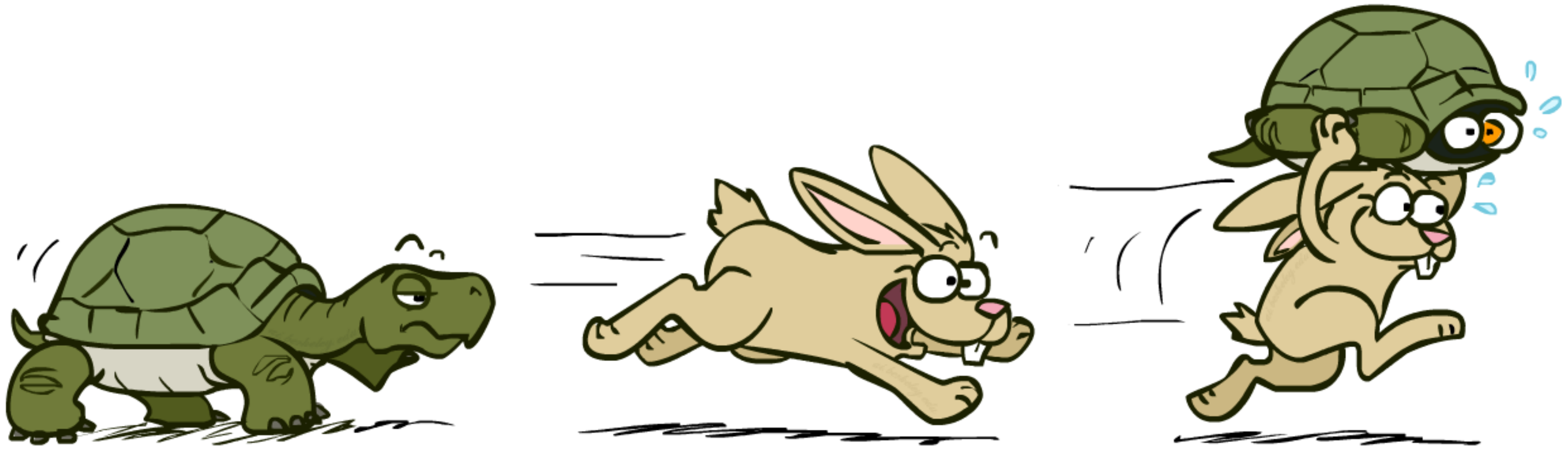
- Consistency implies admissibility

- 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
```

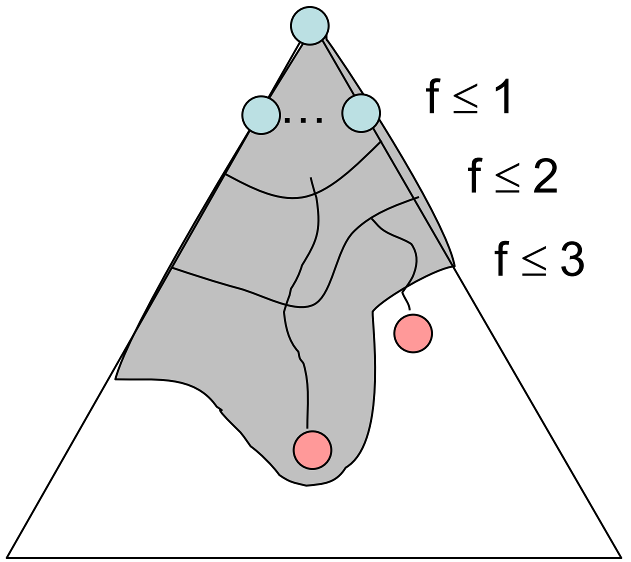
# Graph Search Pseudo-Code

```
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
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE[node]) then return node
      if STATE[node] is not in closed then
         add STATE[node] to closed
         for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
         end
   end
```

# Optimality of A* Graph Search

- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
    Reminder: f(n) = g(n) + h(n) = cost to n + heuristic
  - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first

There's a problem with this argument.  What are we assuming is true?



# Optimality of A* Graph Search

Proof:

- New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expanded first (disaster!)
- Take the highest such *n* in tree
- Let *p* be the ancestor of *n* that was on the queue when *n*' was popped
- *f(p) < f(n)* because of consistency
- *f(n) < f(n')* because *n'* is suboptimal
- *p* would have been expanded before *n*'
- Contradiction!