CS188 Announcements (1)

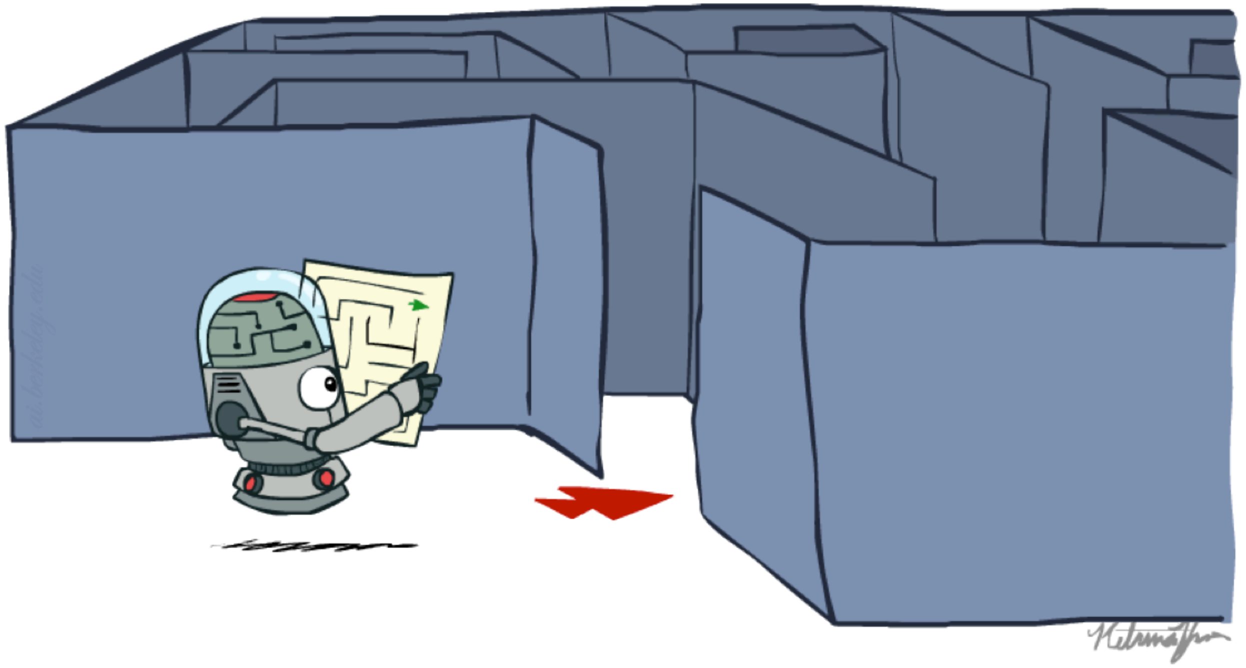
- Piazza used for class communication.
- Waitlist
  - We’re trying to make the class bigger. We’ll let you know.
- Project 0
  - Ungraded, but will be used to see if waitlisted people are still active.
  - Due tomorrow **5pm.**
- Math self test
  - Ungraded. If you’re unfamiliar with material, you may have trouble in 2nd half of the course.

CS188 Announcements (2)

- Homework 1
  - Out this week. See edge.edx.org "Course” tab.
  - Due Sept 6 midnight (local time).
- Sections *this week only*
  - Try to go to assigned section.
  - Exam practice sections will be “overflow” standard section.
- Exam schedule
  - Thu Oct 6 7-9pm, and Tue Nov 8 7-9pm
  - Makeup 8am the next morning. See Piazza post.

CS 188: Artificial Intelligence

Search



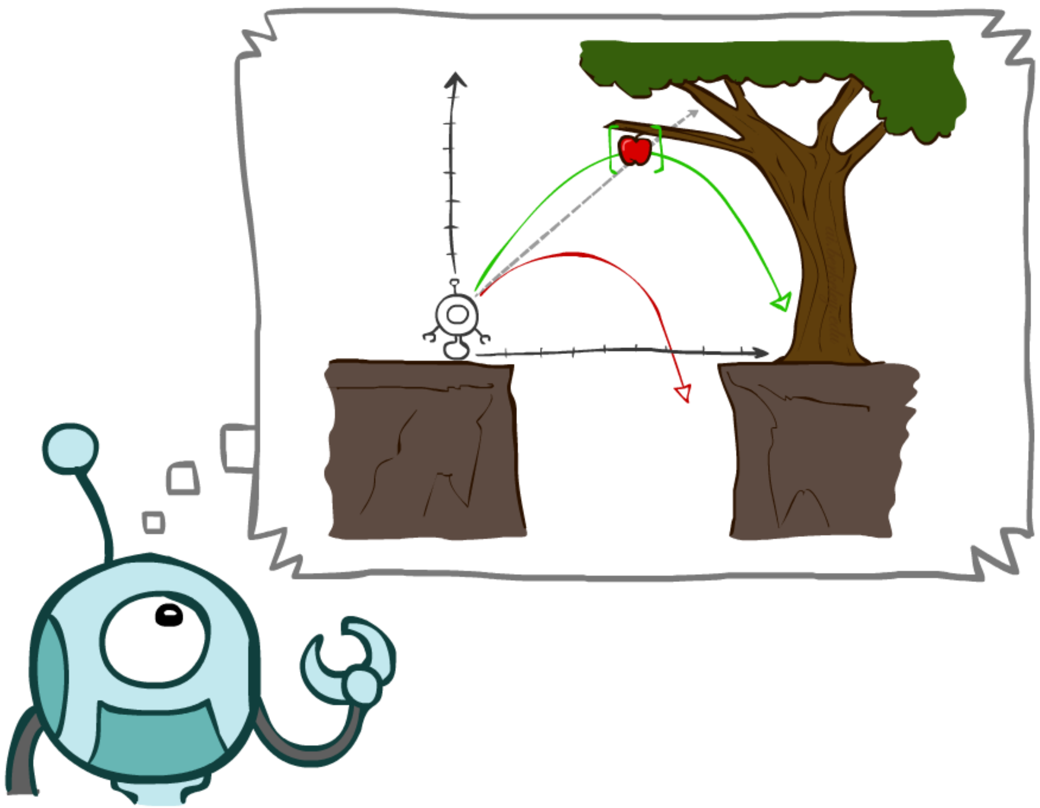
Instructors: Josh Hug and Adam Janin

University of California, Berkeley

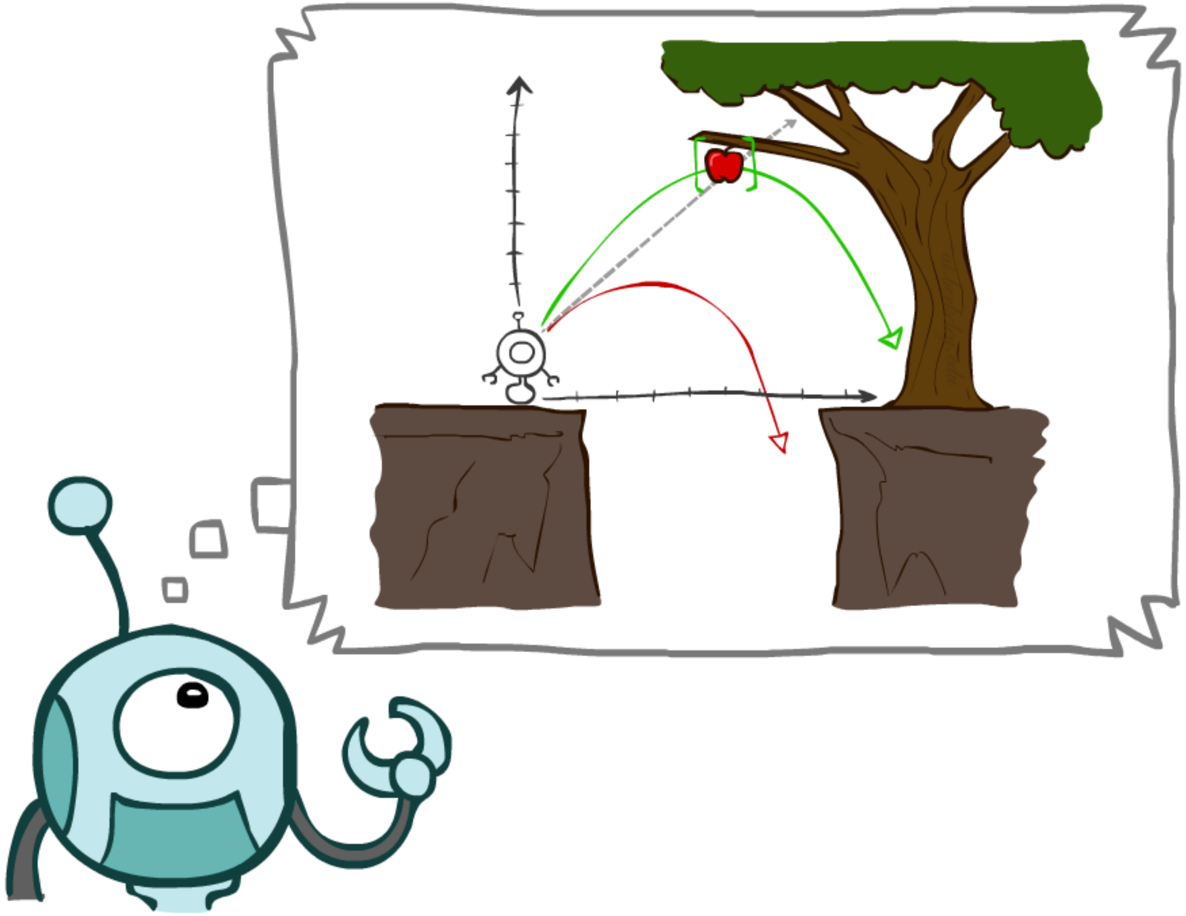
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



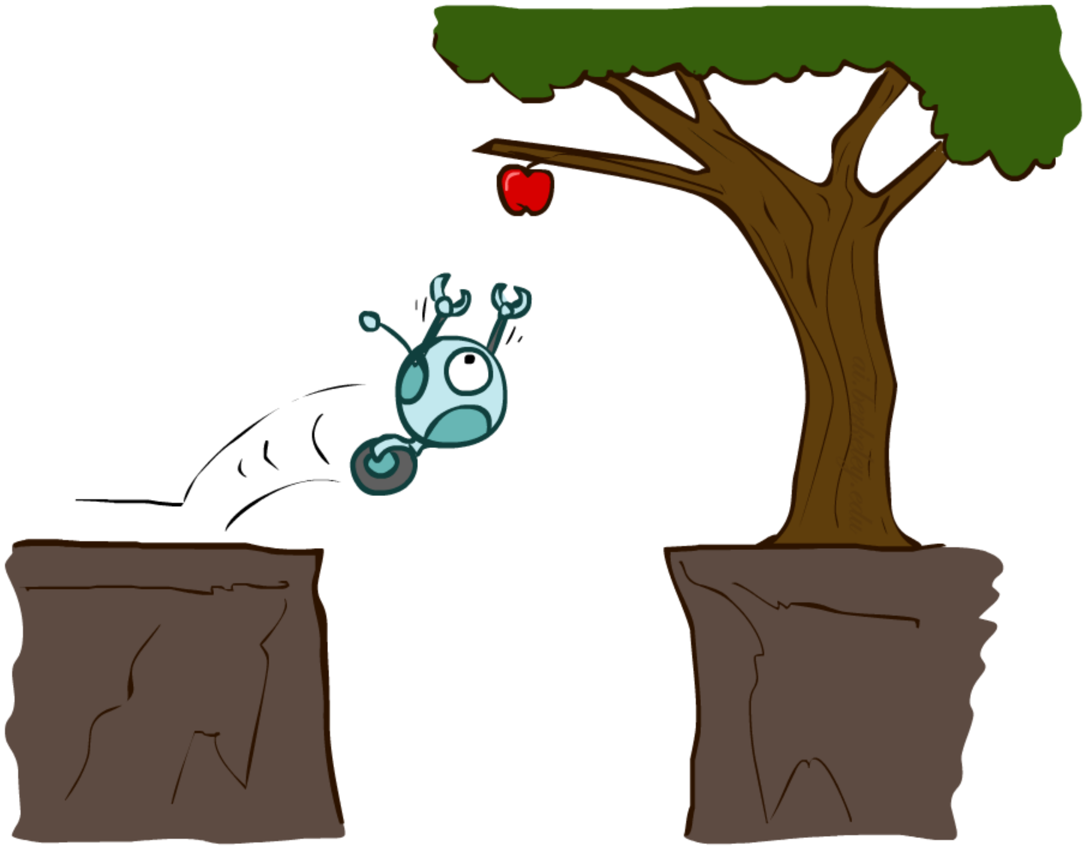
Agents that Plan



Reflex Agents

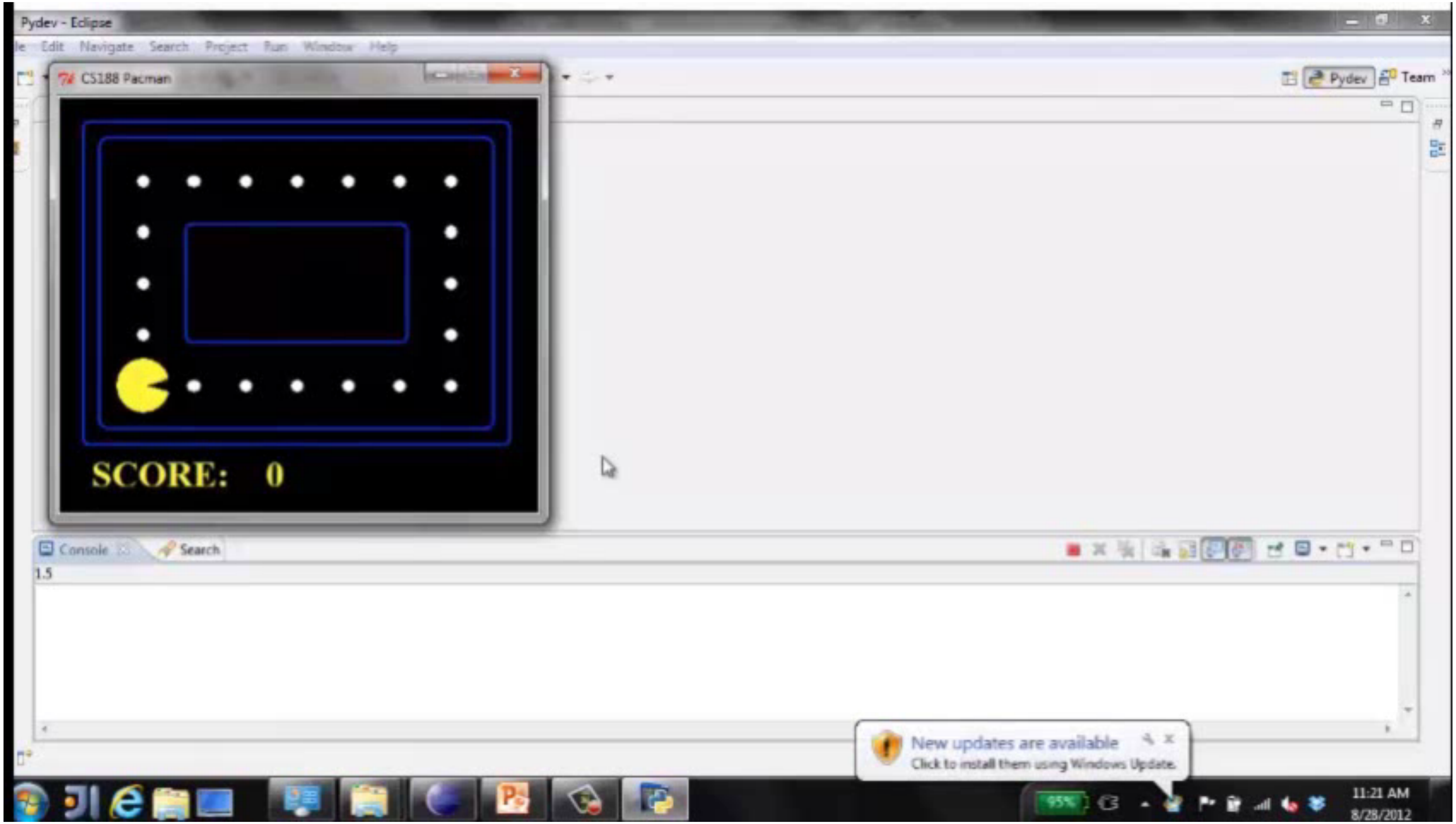
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS

- Can a reflex agent be rational?

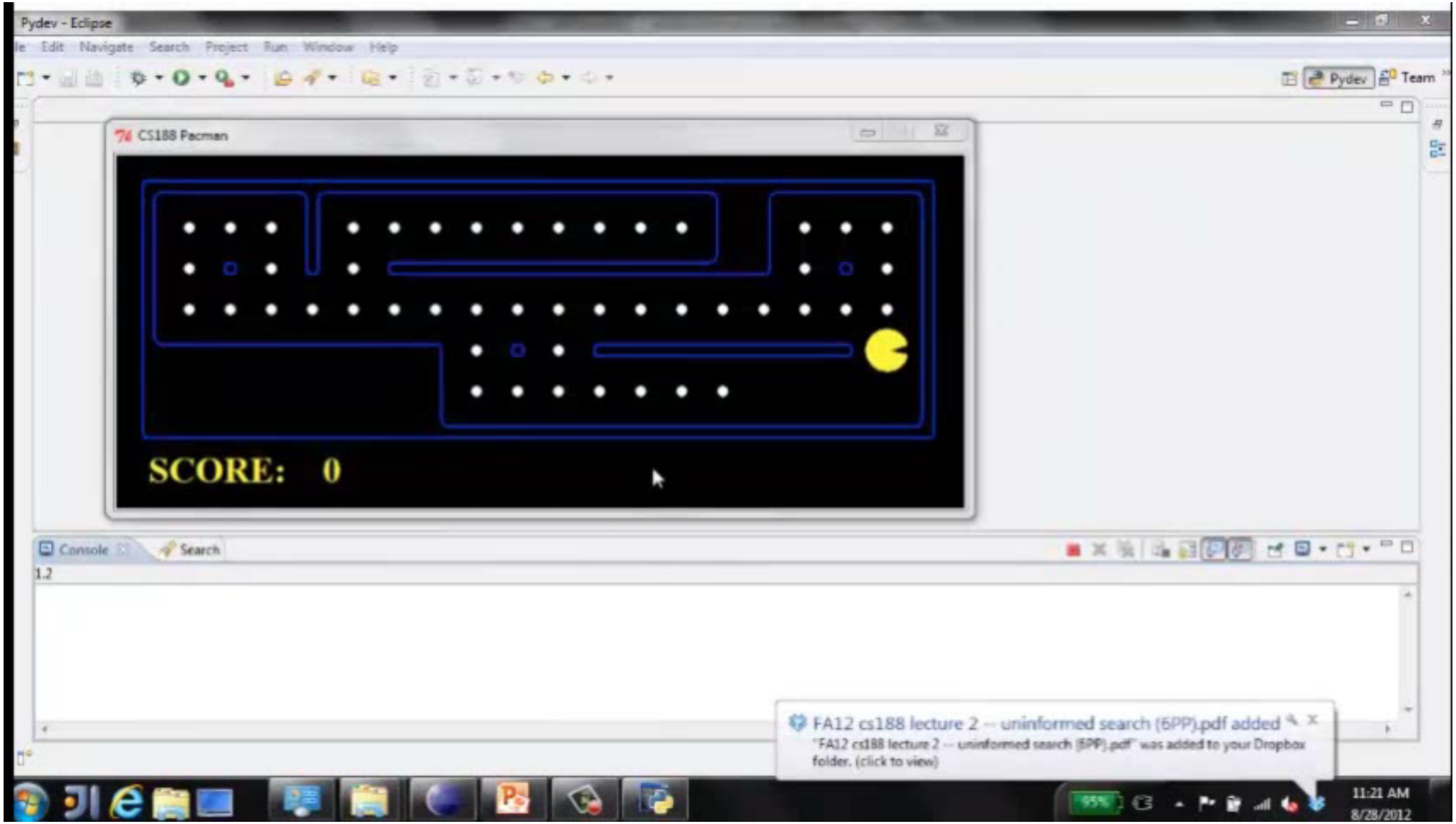


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Video of Demo Reflex Optimal

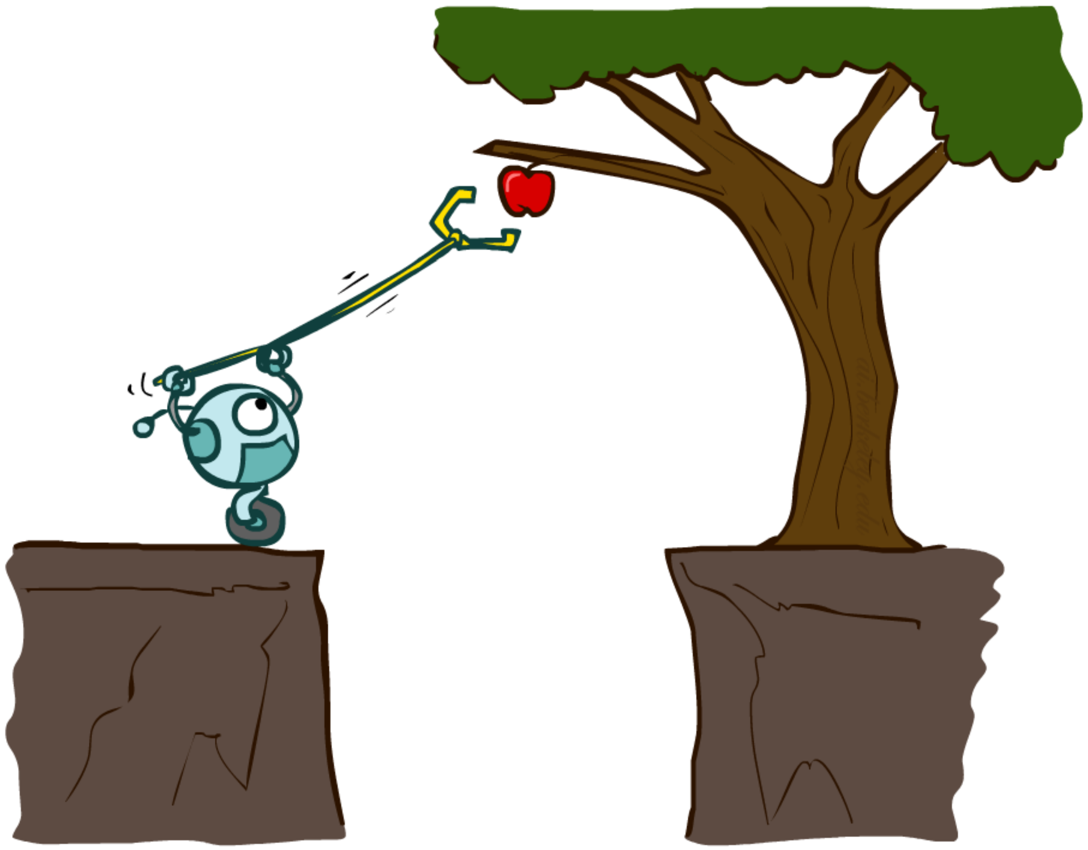


Video of Demo Reflex Odd



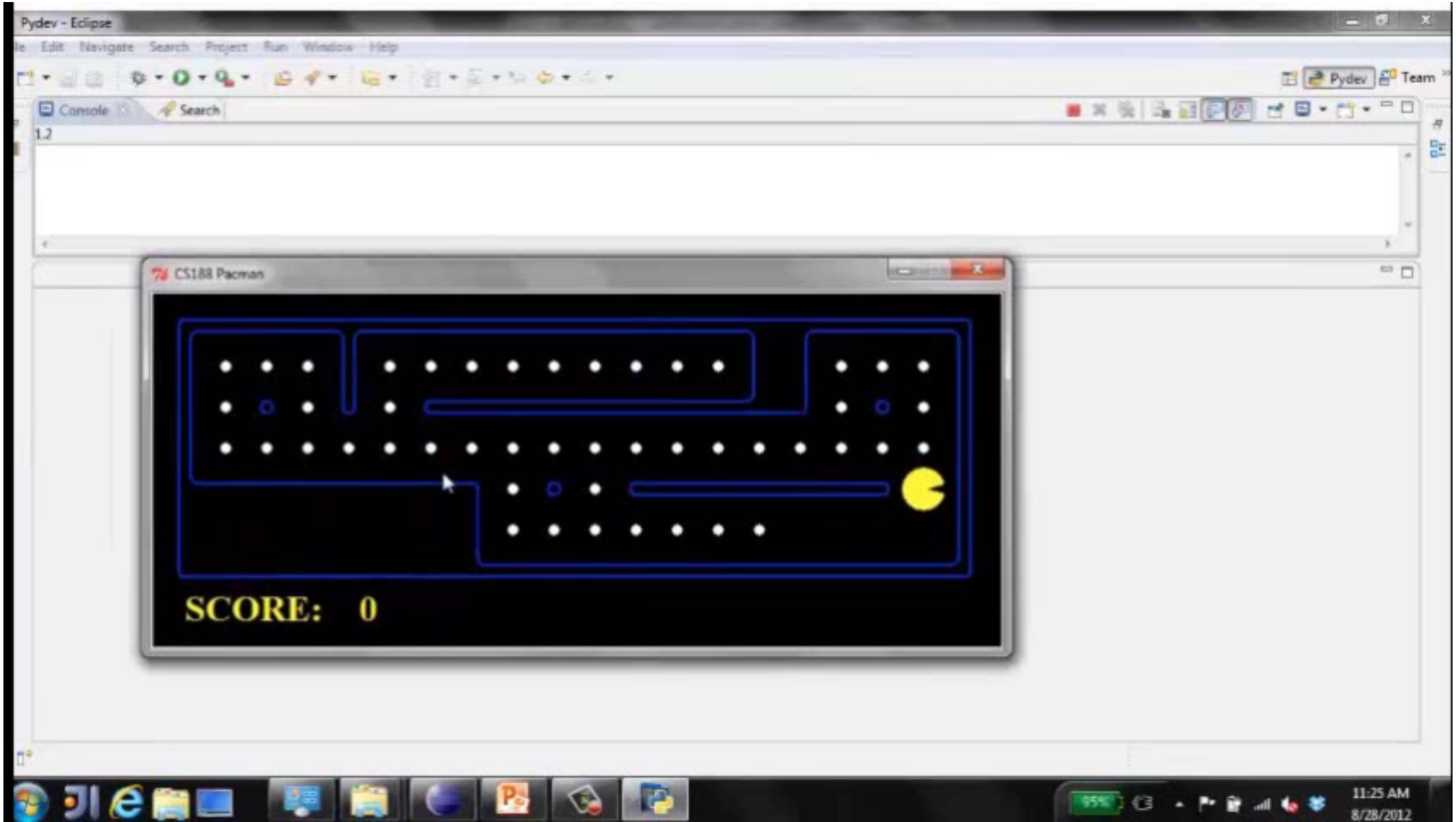
Planning Agents

- Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Optimal vs. complete planning
- Planning vs. replanning

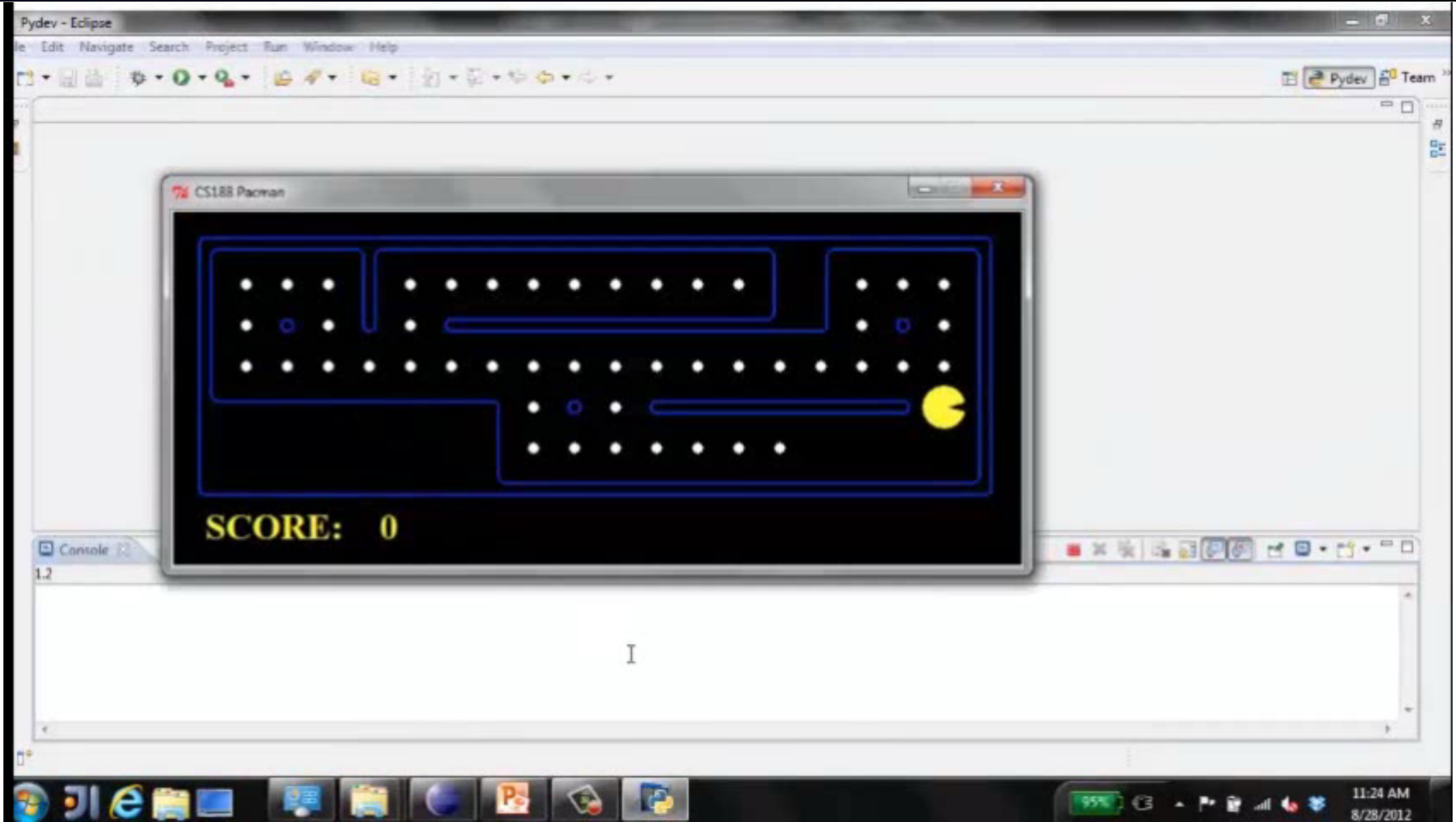


[Demo: re-planning (L2D3)]
[Demo: mastermind (L2D4)]

Video of Demo Replanning



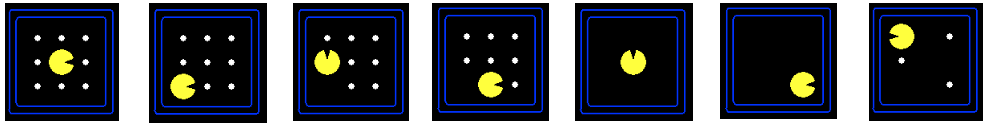
Video of Demo Mastermind



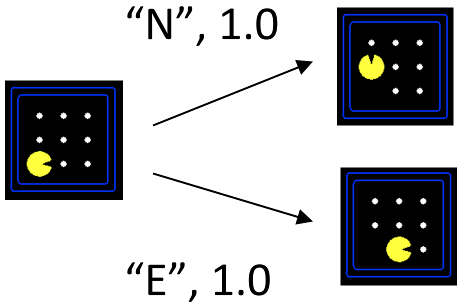
Search Problems

Search Problems

- A **search problem** consists of:
  - A state space



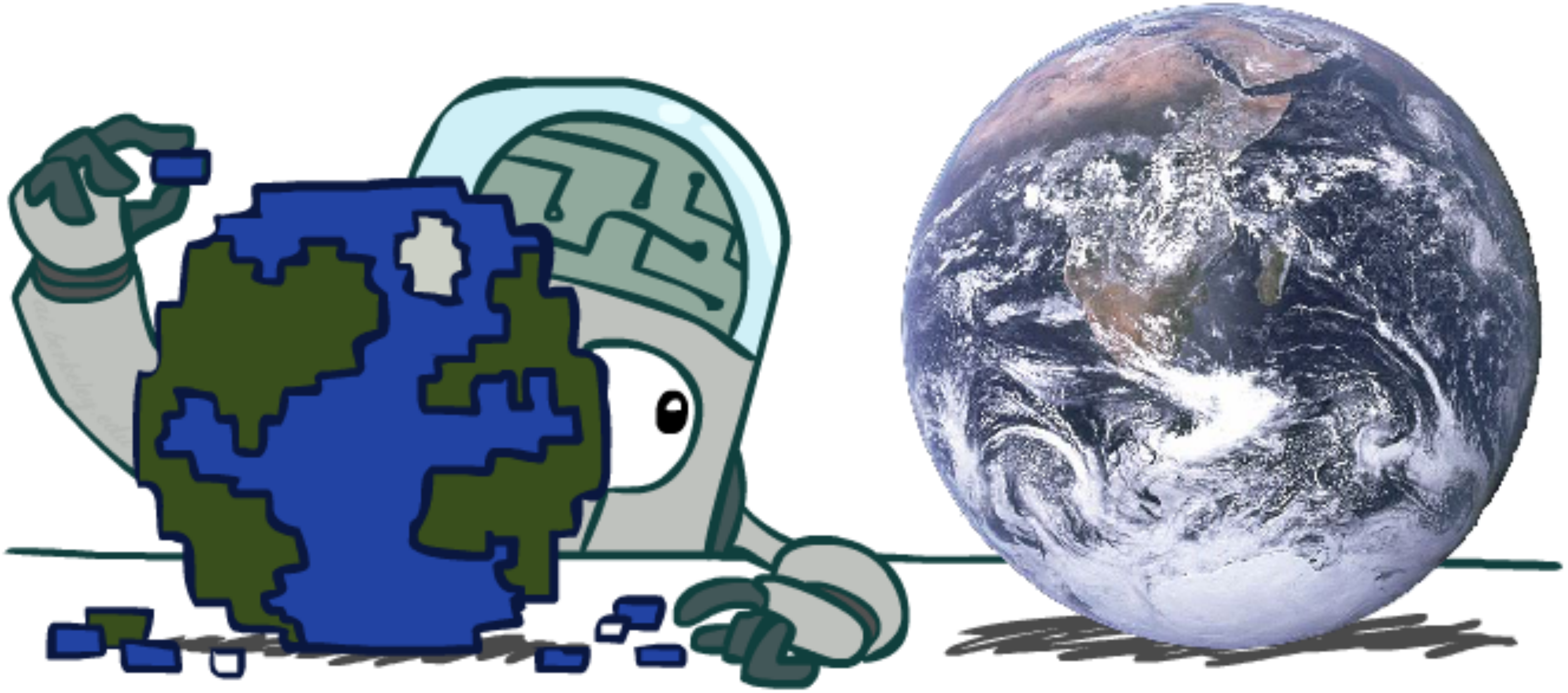
  - A successor function (with actions, costs)



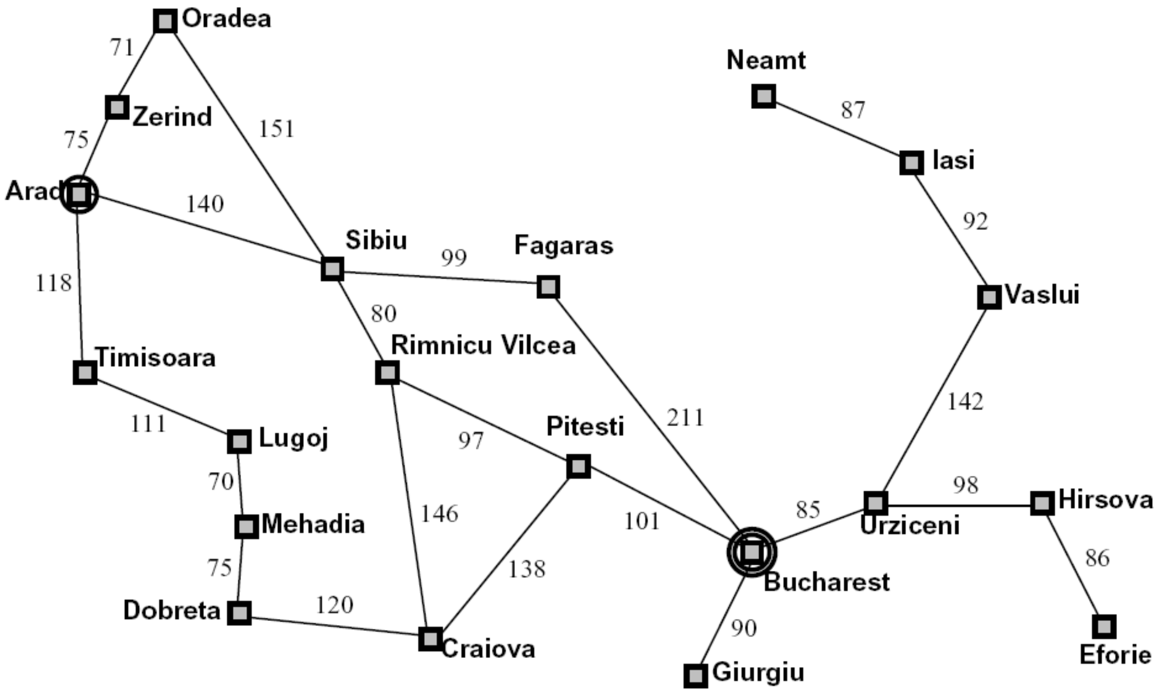
  - A start state and a goal test

- A **solution** is a sequence of actions (a plan) which transforms the start state to a goal state

Search Problems Are Models



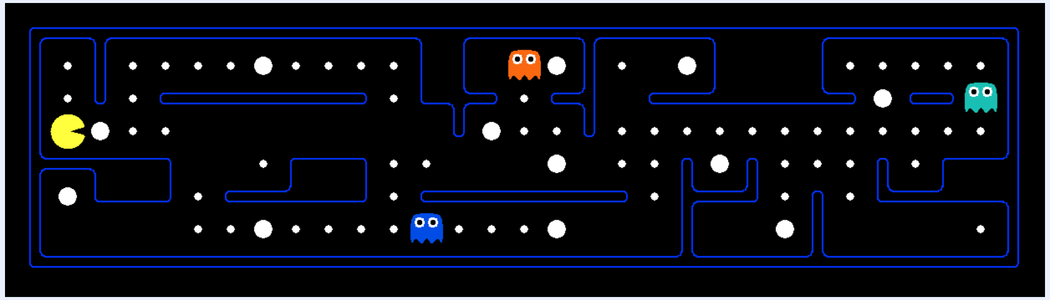
Example: Traveling in Romania



- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

What's in a State Space?

The world state includes every last detail of the environment



A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes?

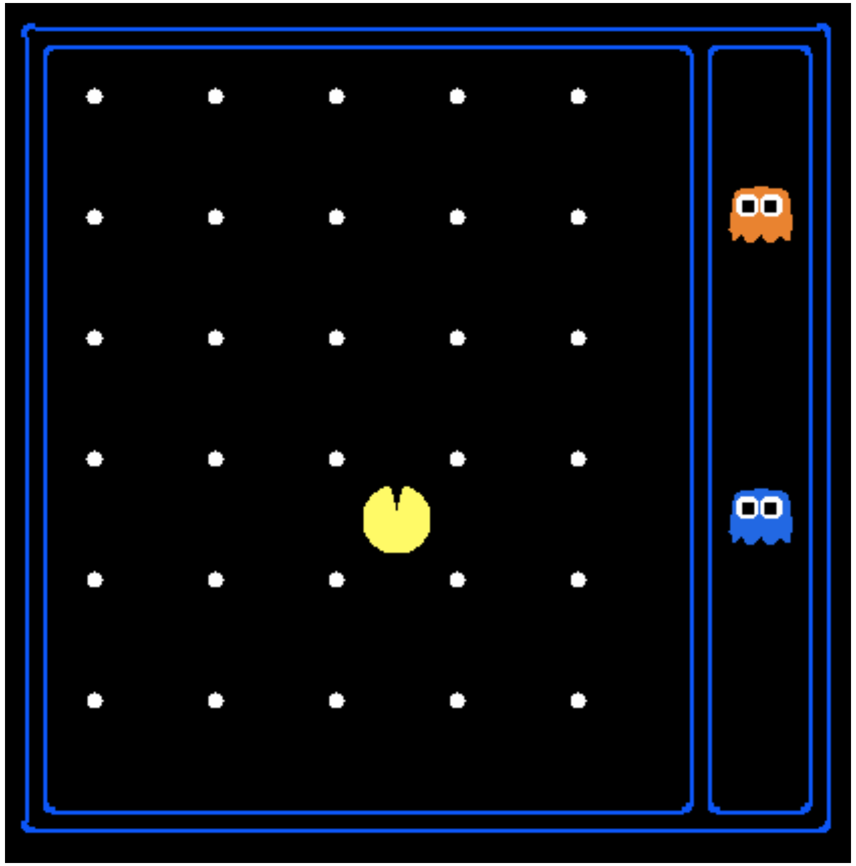
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

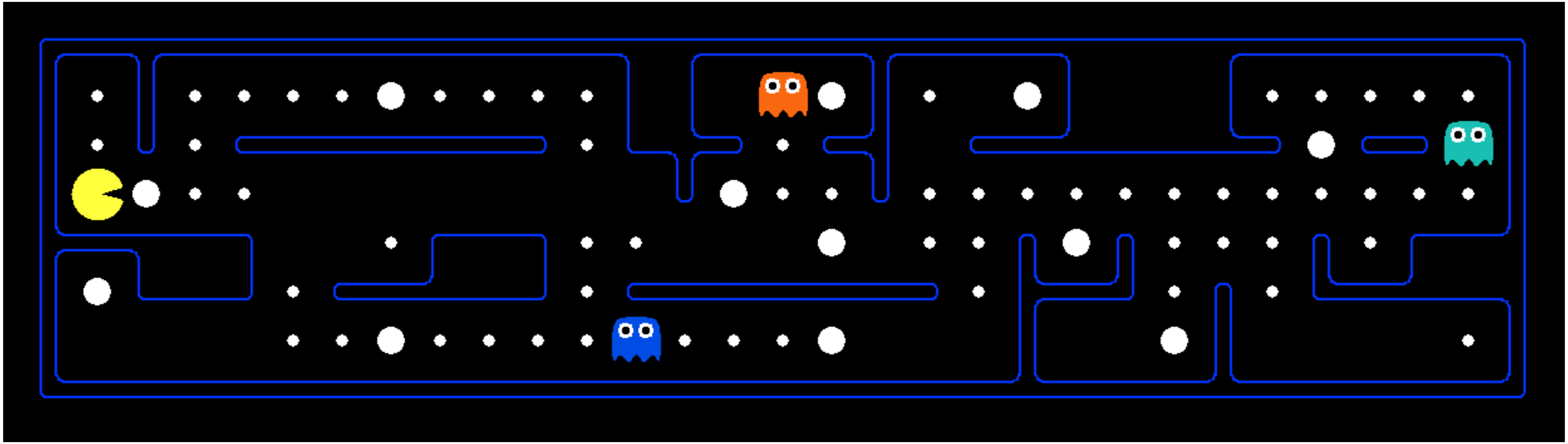
    $120x(2^{30})x(12^2)x4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120x(2^{30})$

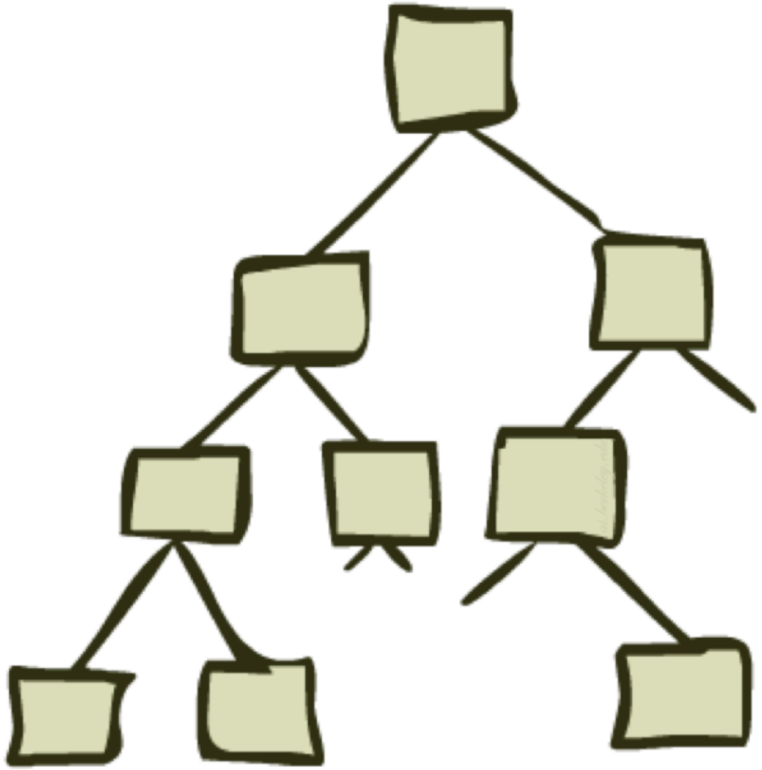


Quiz: Safe Passage



- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

State Space Graphs and Search Trees

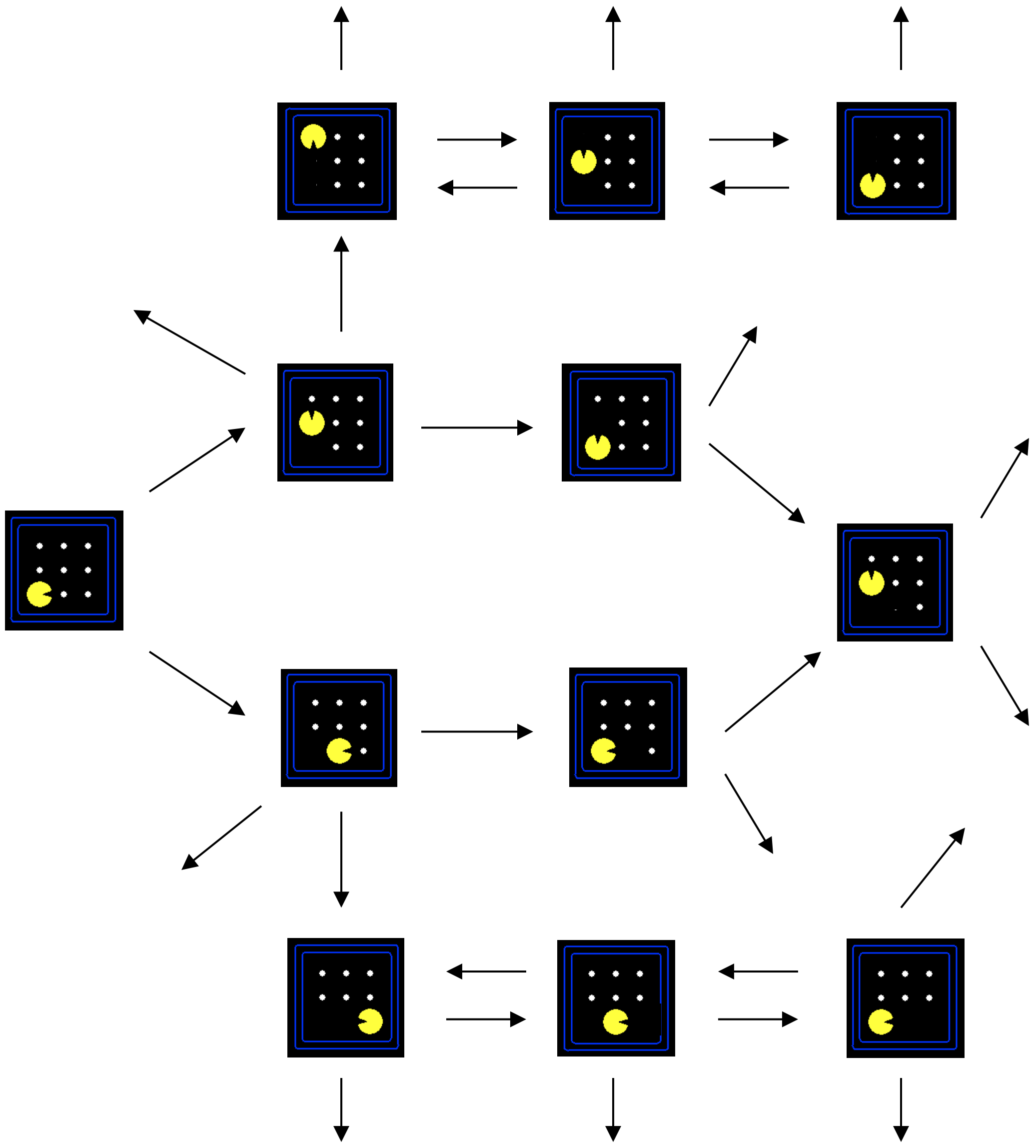


State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)

- In a state space graph, each state occurs only once!

- We can rarely build this full graph in memory (it's too big), but it's a useful idea

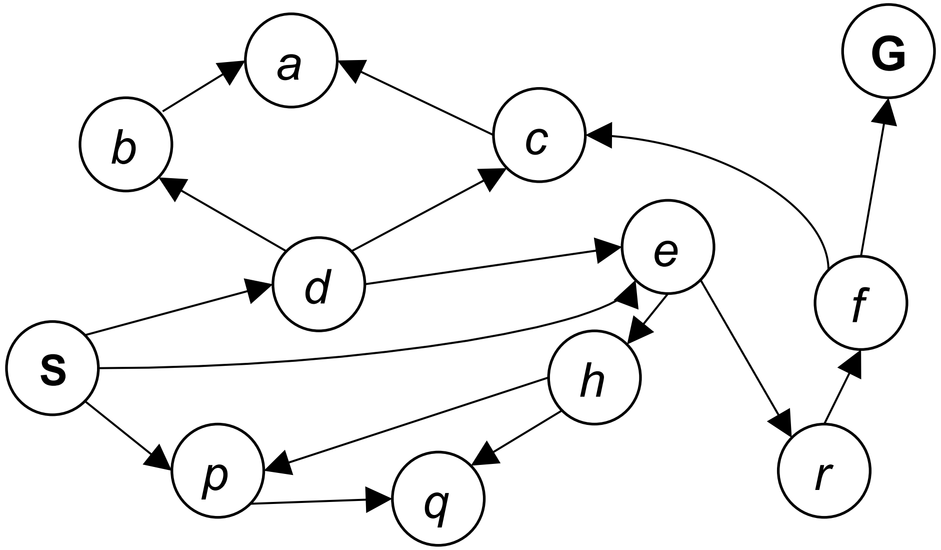


State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)

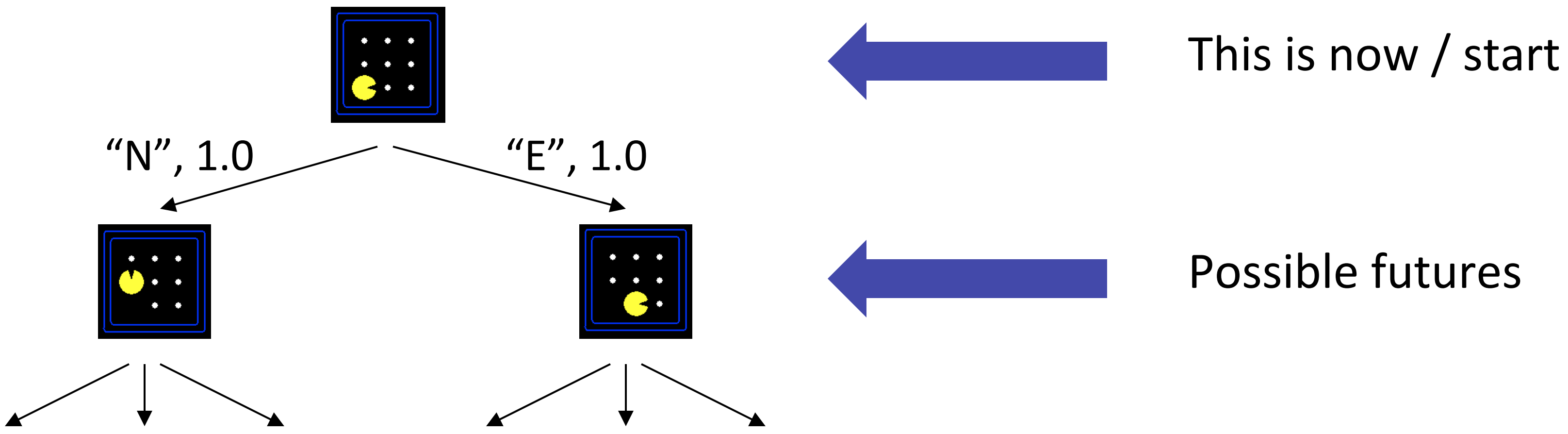
- In a search graph, each state occurs only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea



Tiny search graph for a tiny search problem

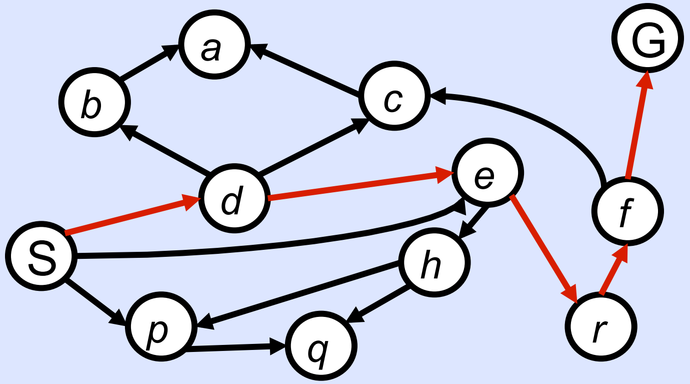
Search Trees



- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

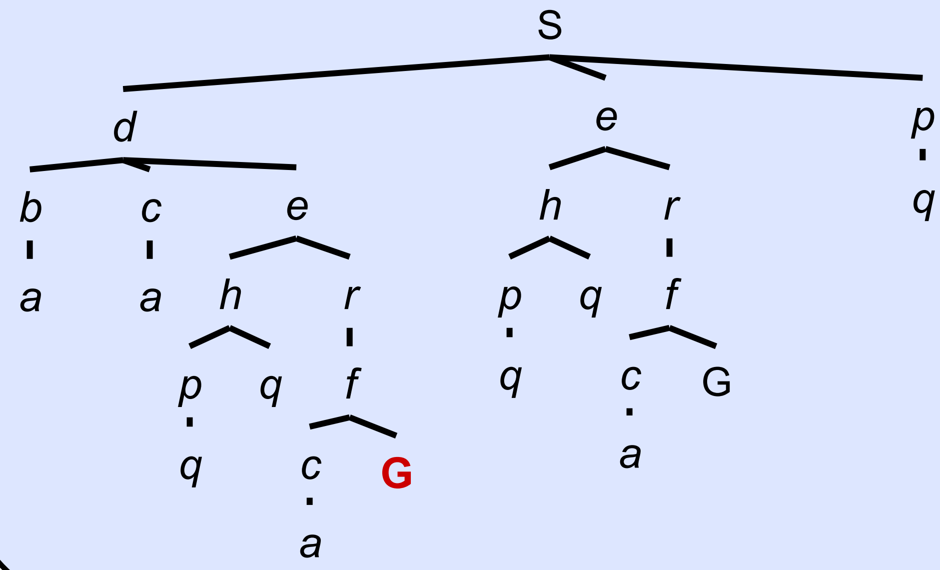
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

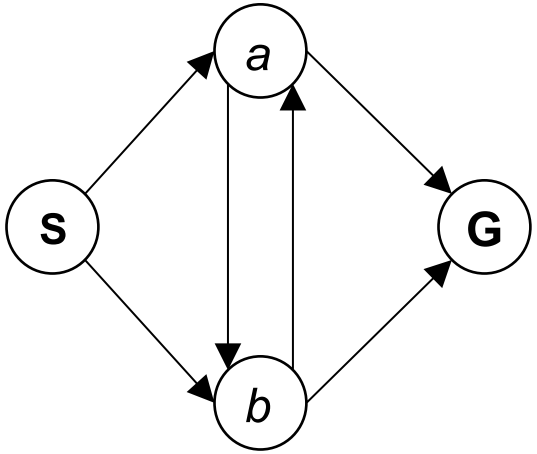
We construct both on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



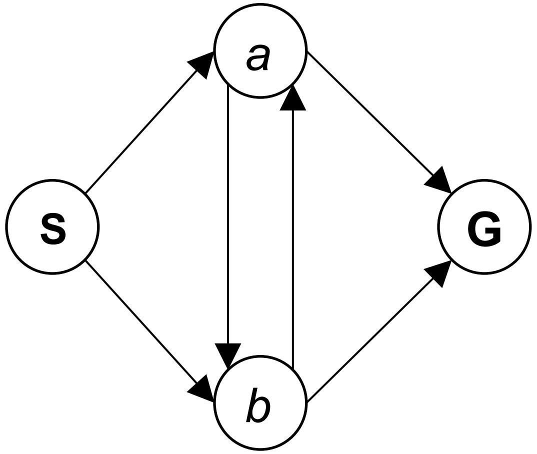
How big is its search tree (from S)?

$$\infty$$

Important: Lots of repeated structure in the search tree!

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



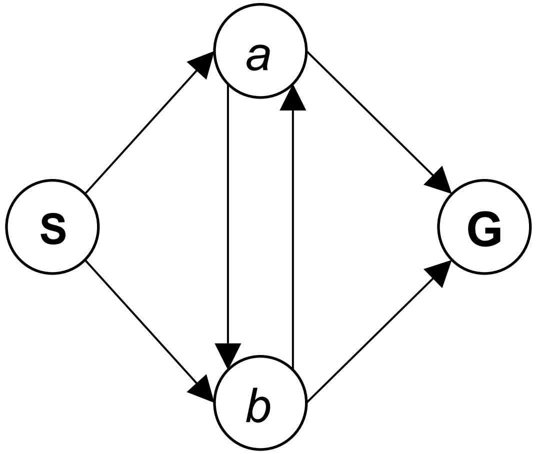
How big is its search tree (from S)?

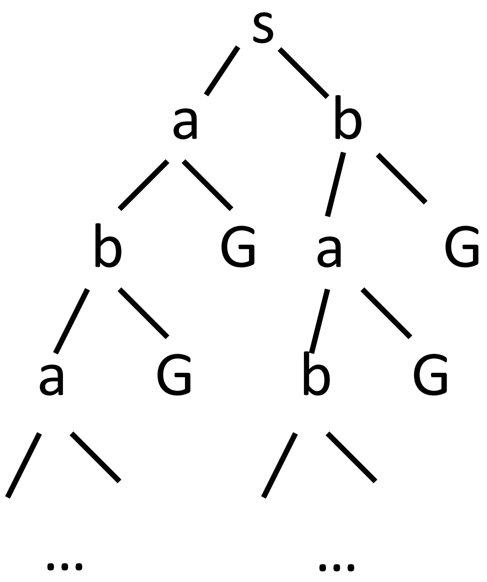
$$\infty$$

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

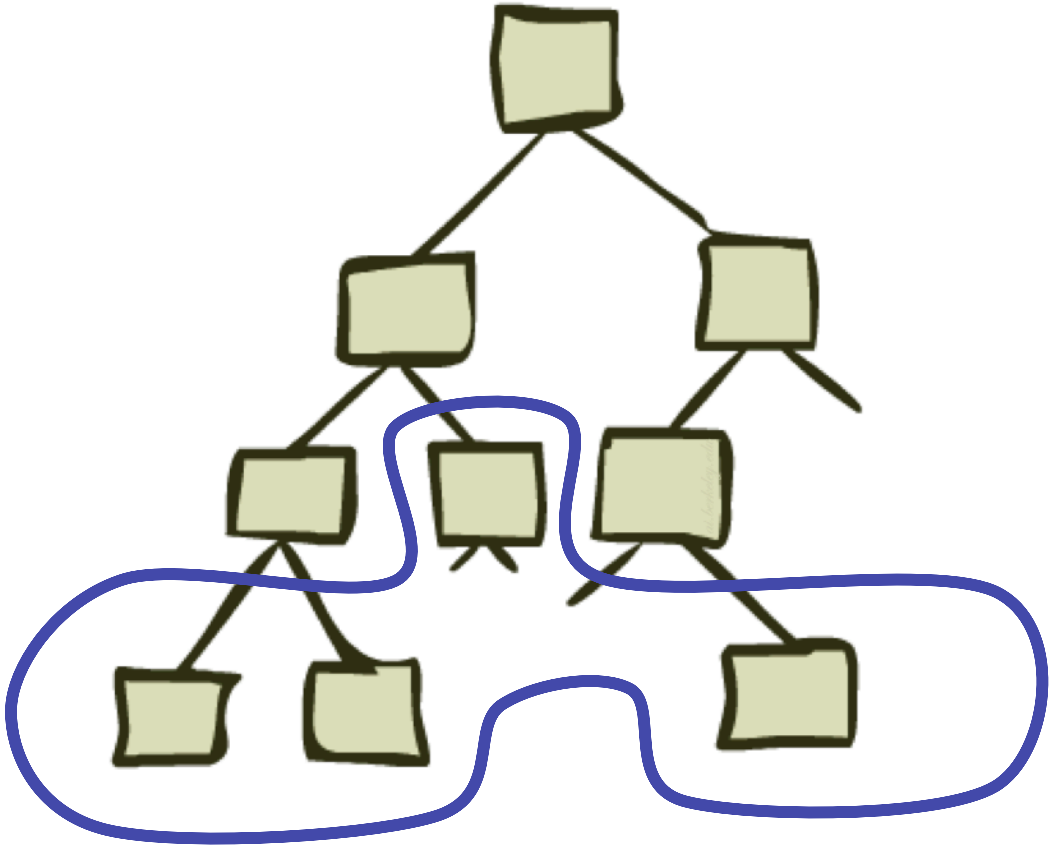
How big is its search tree (from S)?



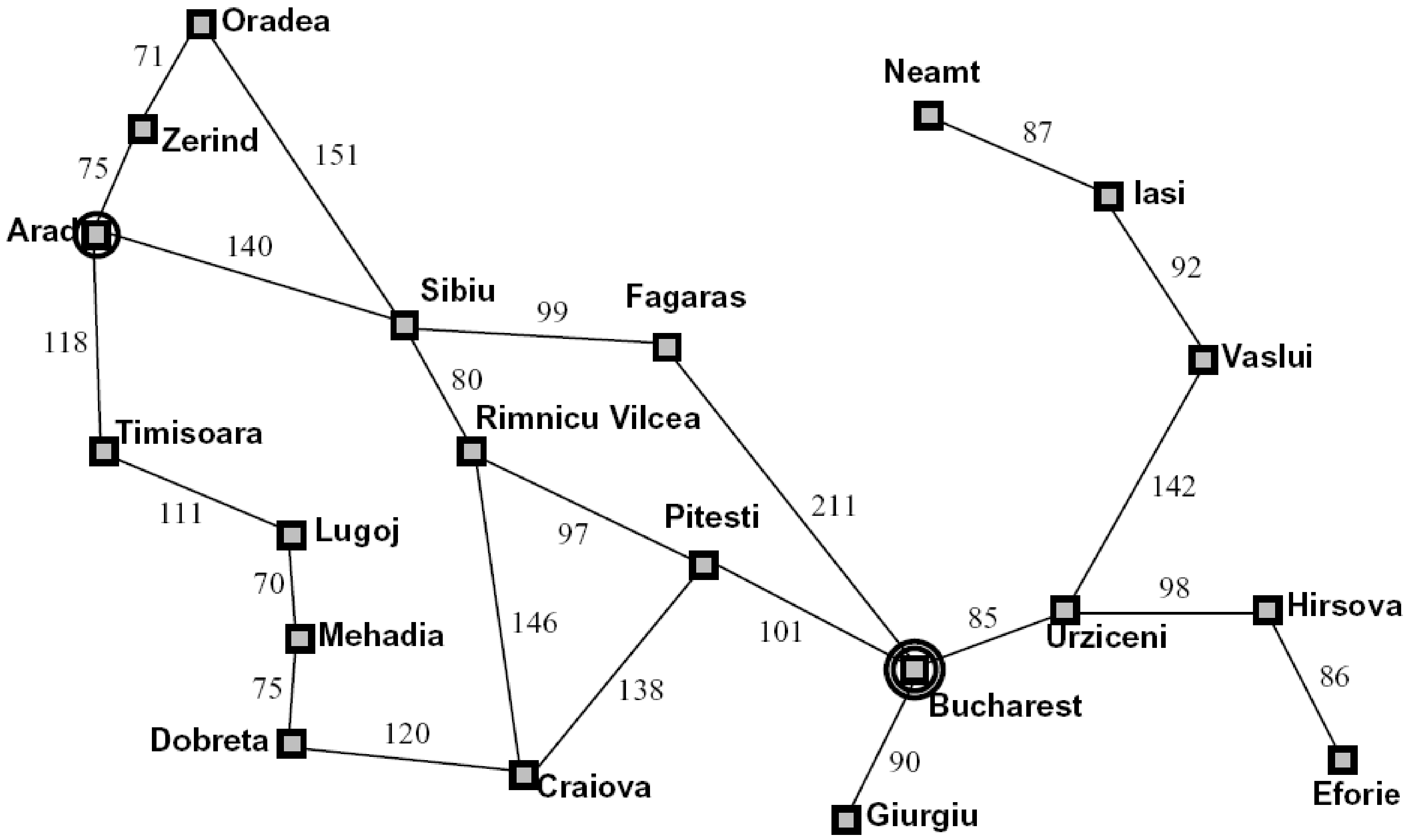


Important: Lots of repeated structure in the search tree!

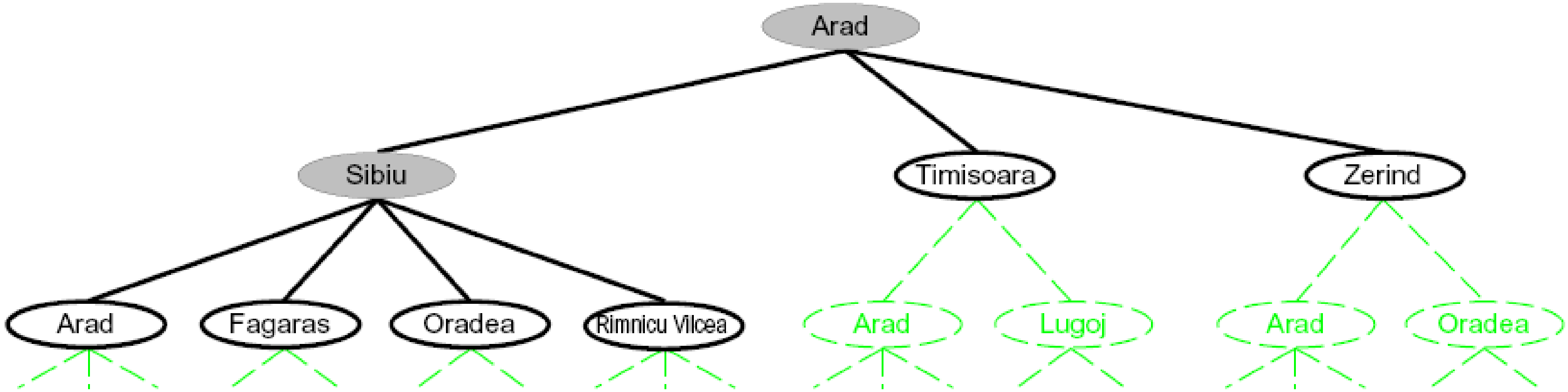
Tree Search



Search Example: Romania



Searching with a Search Tree



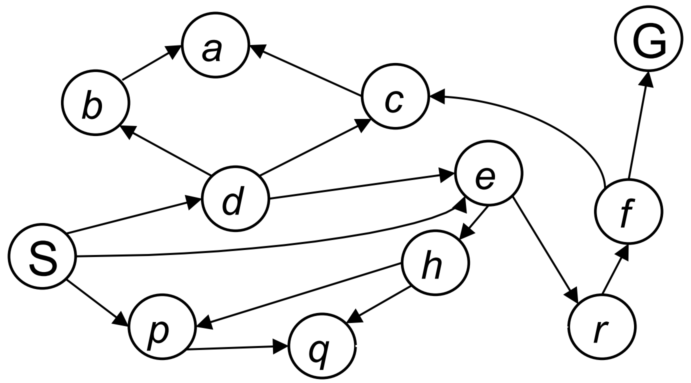
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

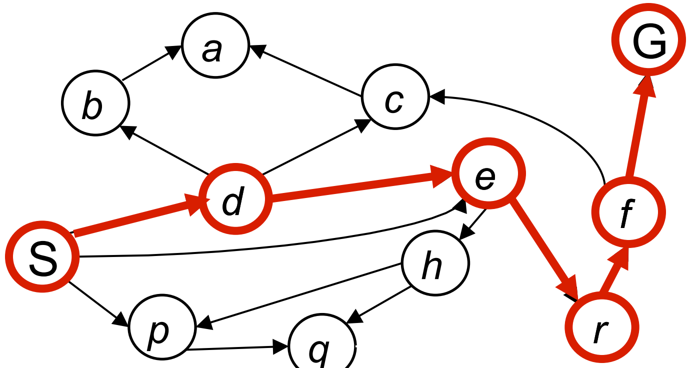
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

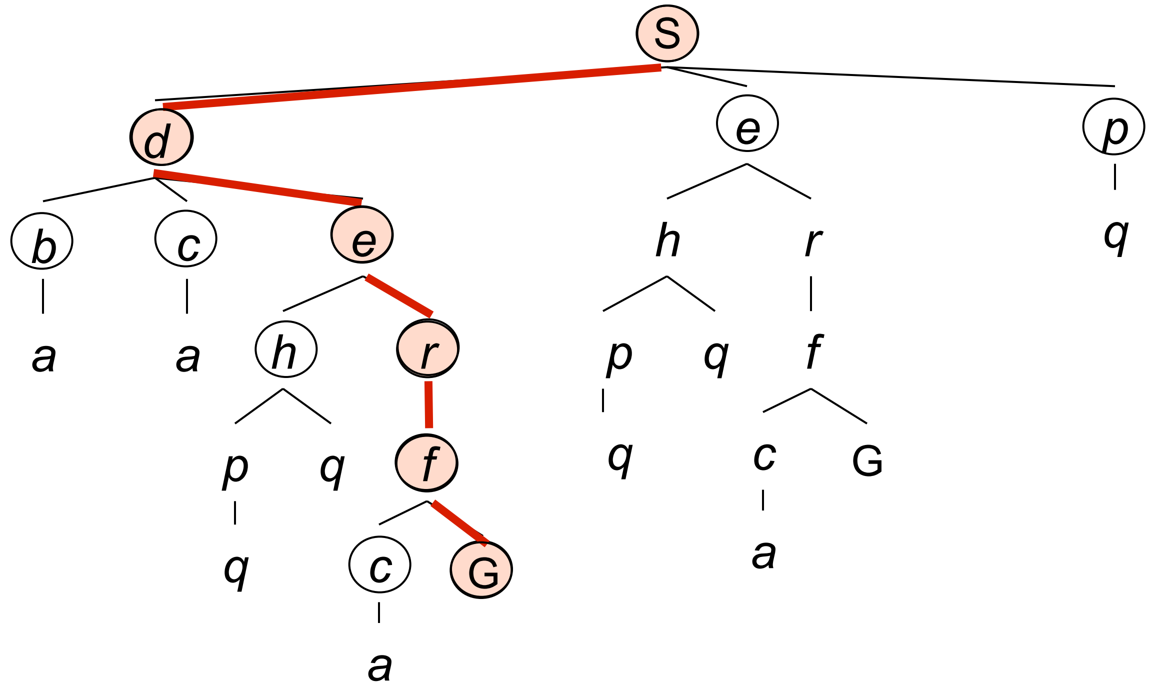
- Important ideas:
    - Fringe
    - Expansion
    - Exploration strategy
- Main question: which fringe nodes to explore?

Example: Tree Search



Example: Tree Search





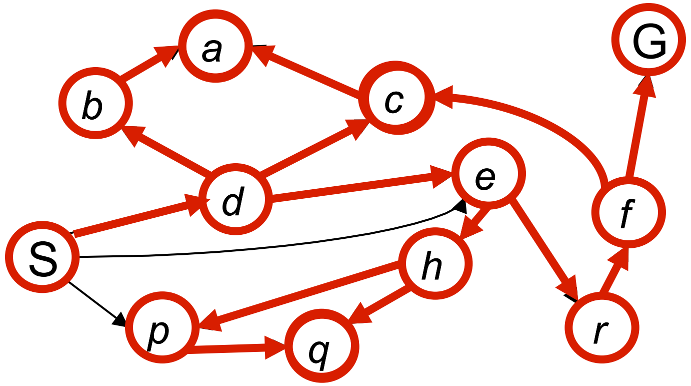
~~s~~
~~s → d~~
s → e
s → p
s → d → b
s → d → c
~~s → d → e~~
s → d → e → h
~~s → d → e → r~~
~~s → d → e → r → f~~
s → d → e → r → f → c
~~s → d → e → r → f → G~~

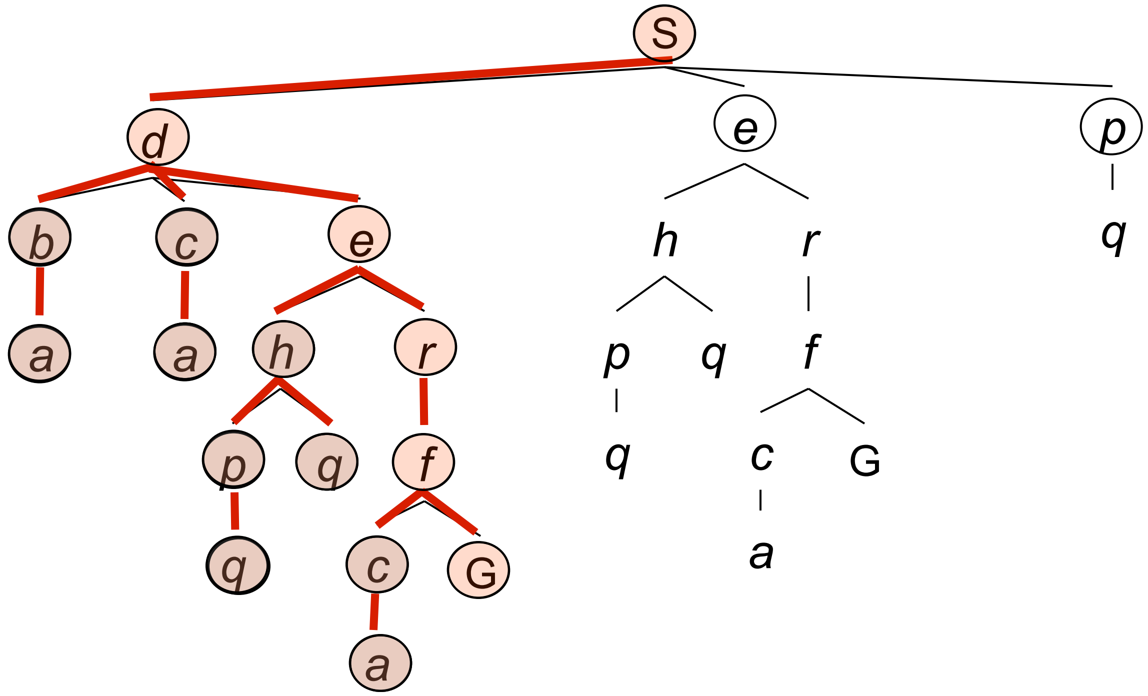
Depth-First Search

Depth-First Search

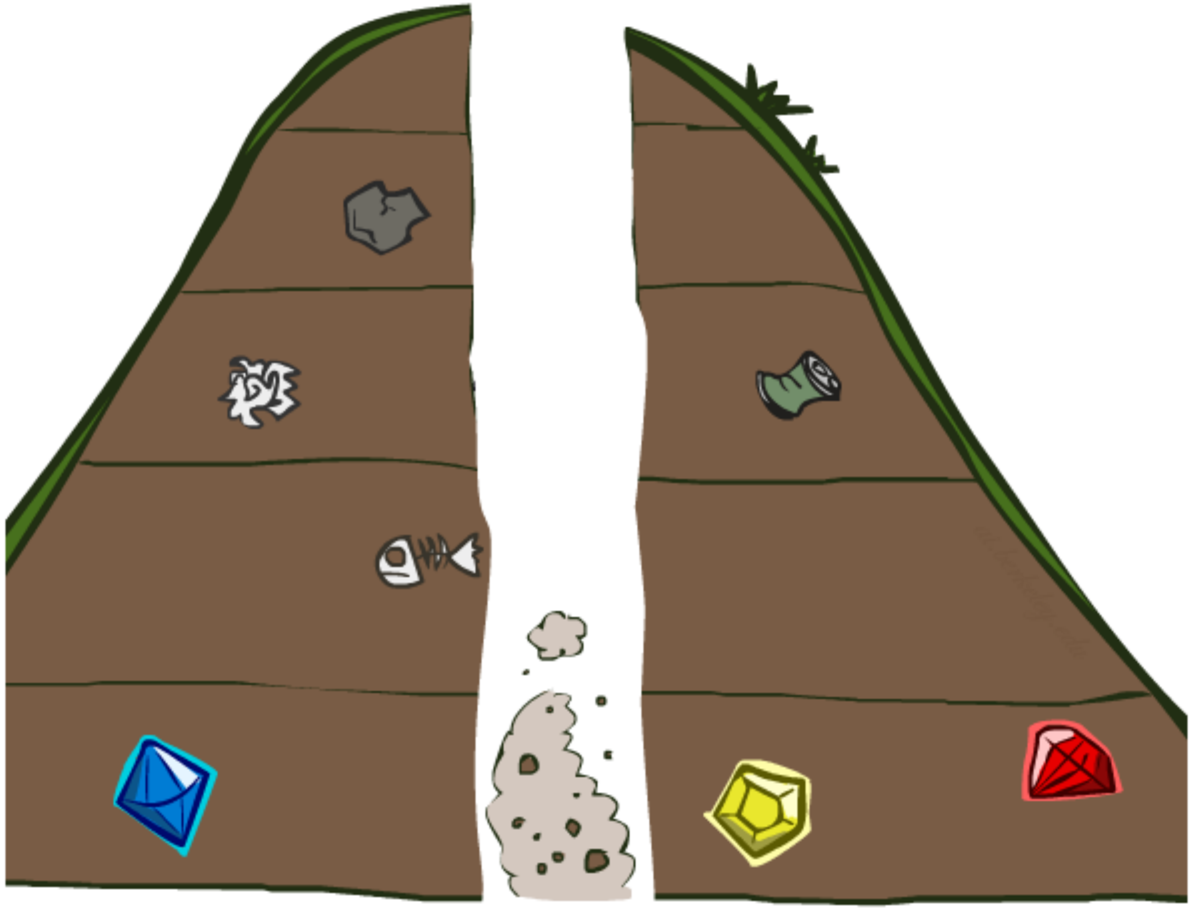
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





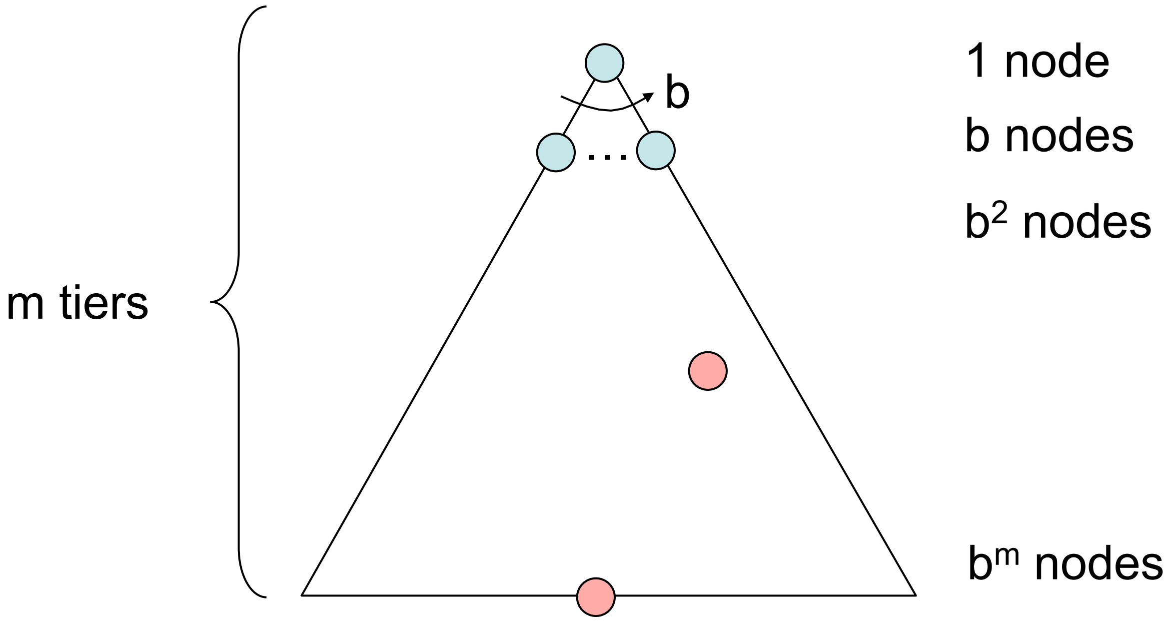
Search Algorithm Properties



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



- Number of nodes in entire tree?
  - $1 + b + b^2 + ... b^m = O(b^m)$

Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost