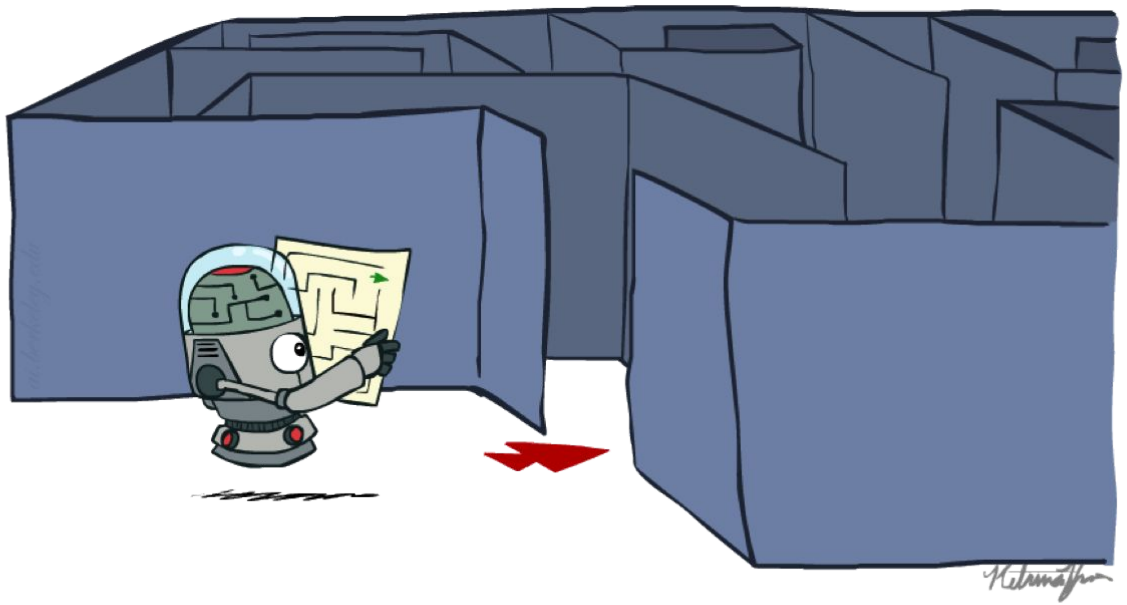
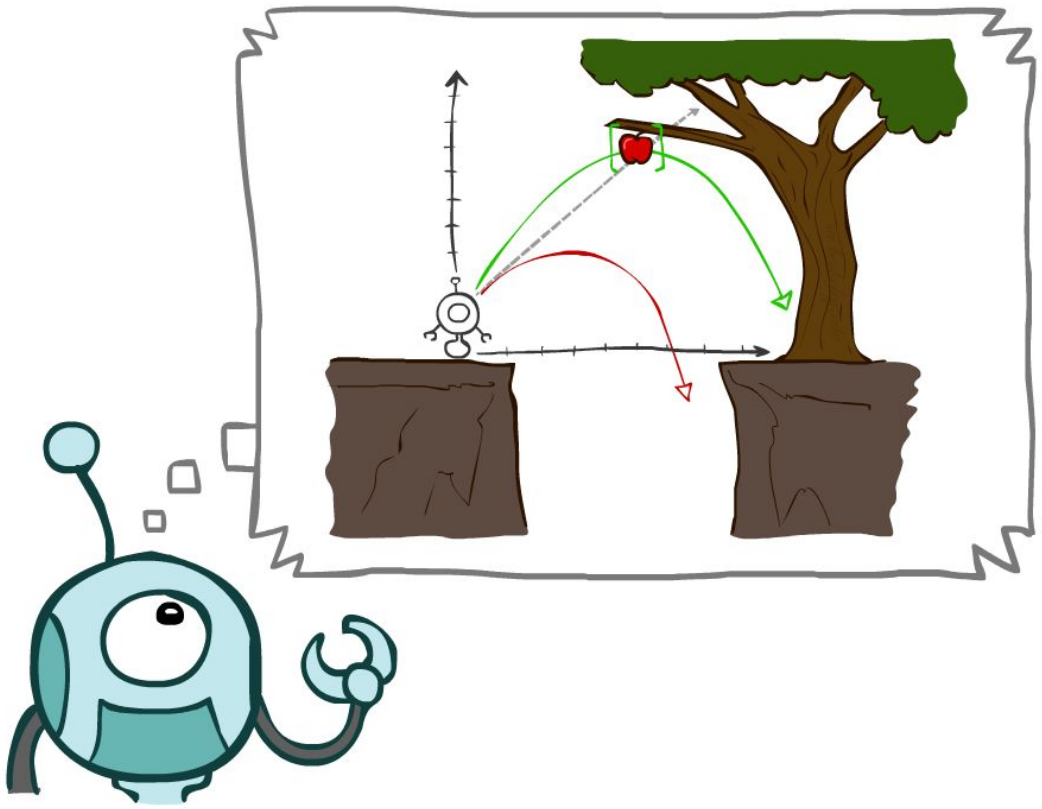
Search



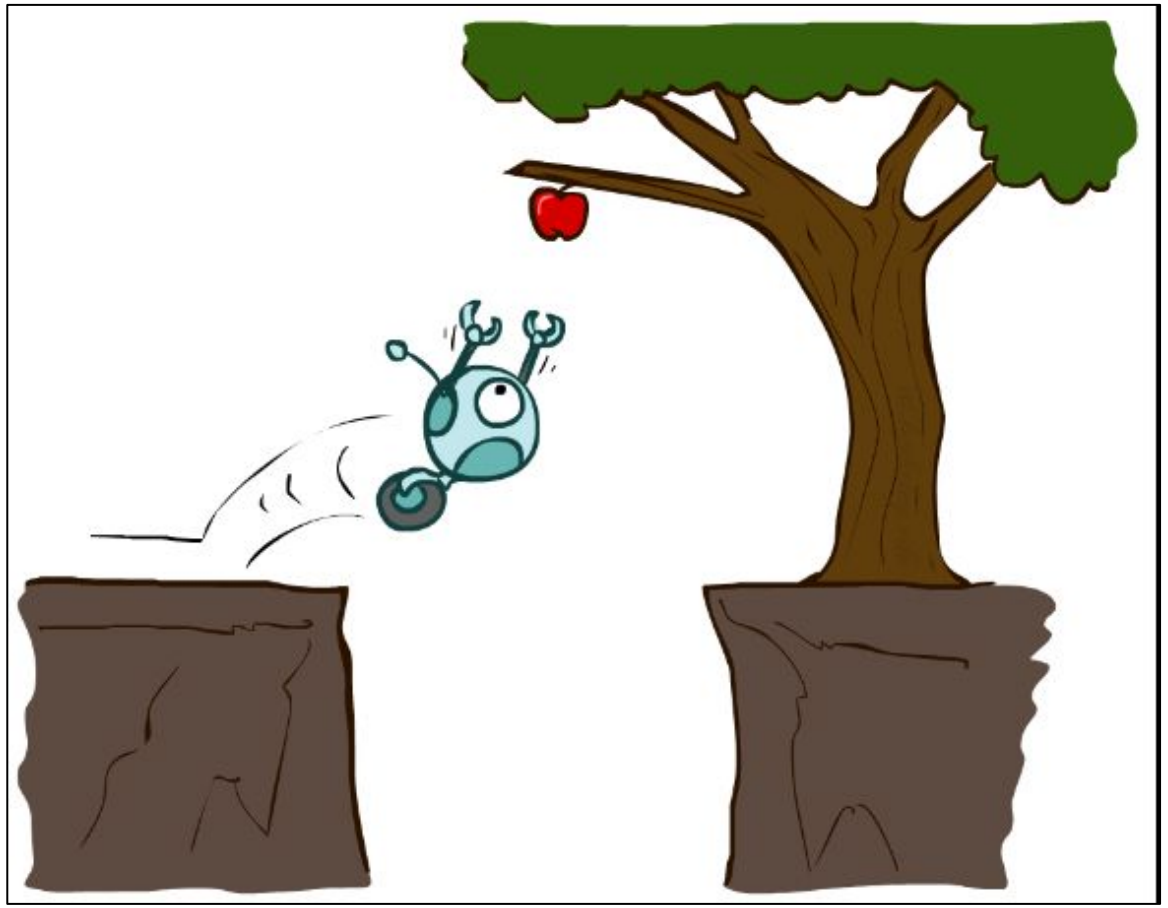
These slides are primarily based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.
The artwork is by Ketrina Yim.

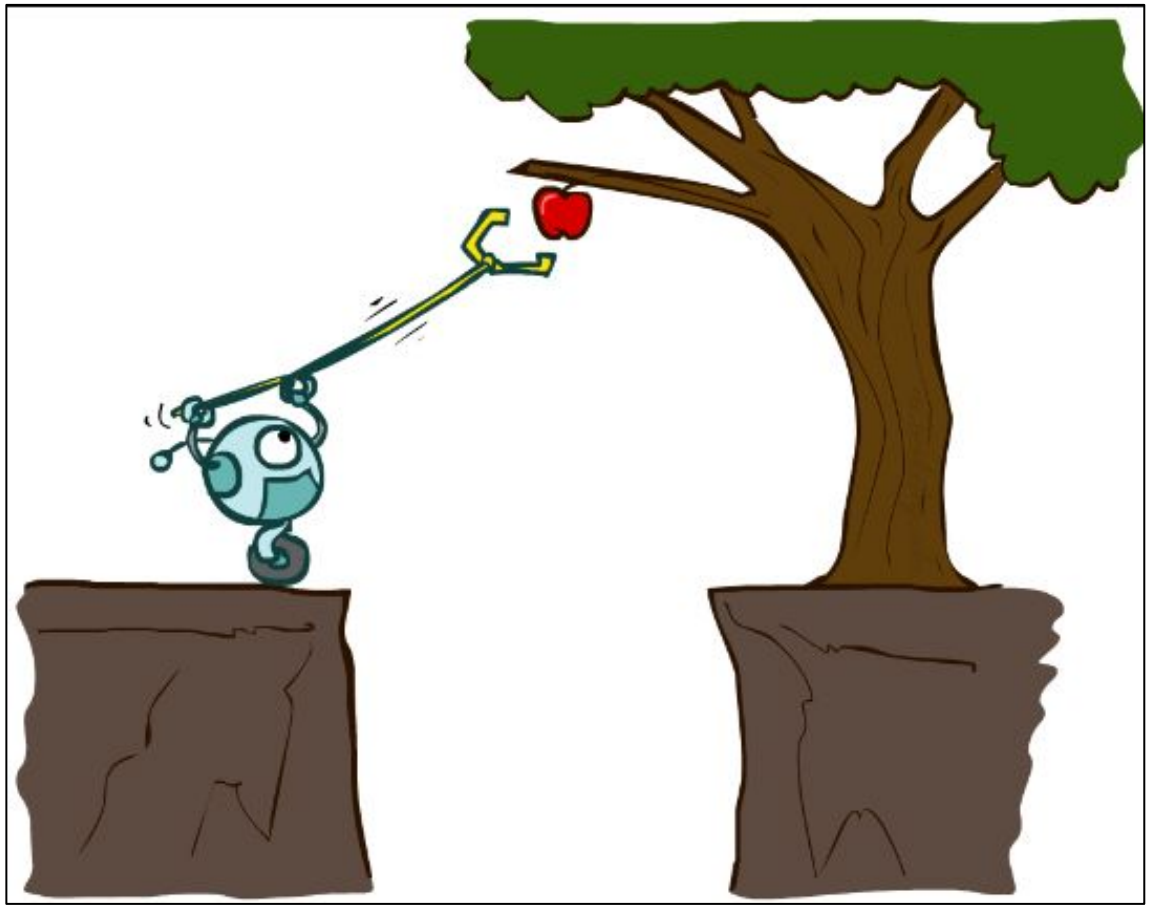
Today

- Problem Solving Agents
- Search Problems
- Uninformed Search Methods



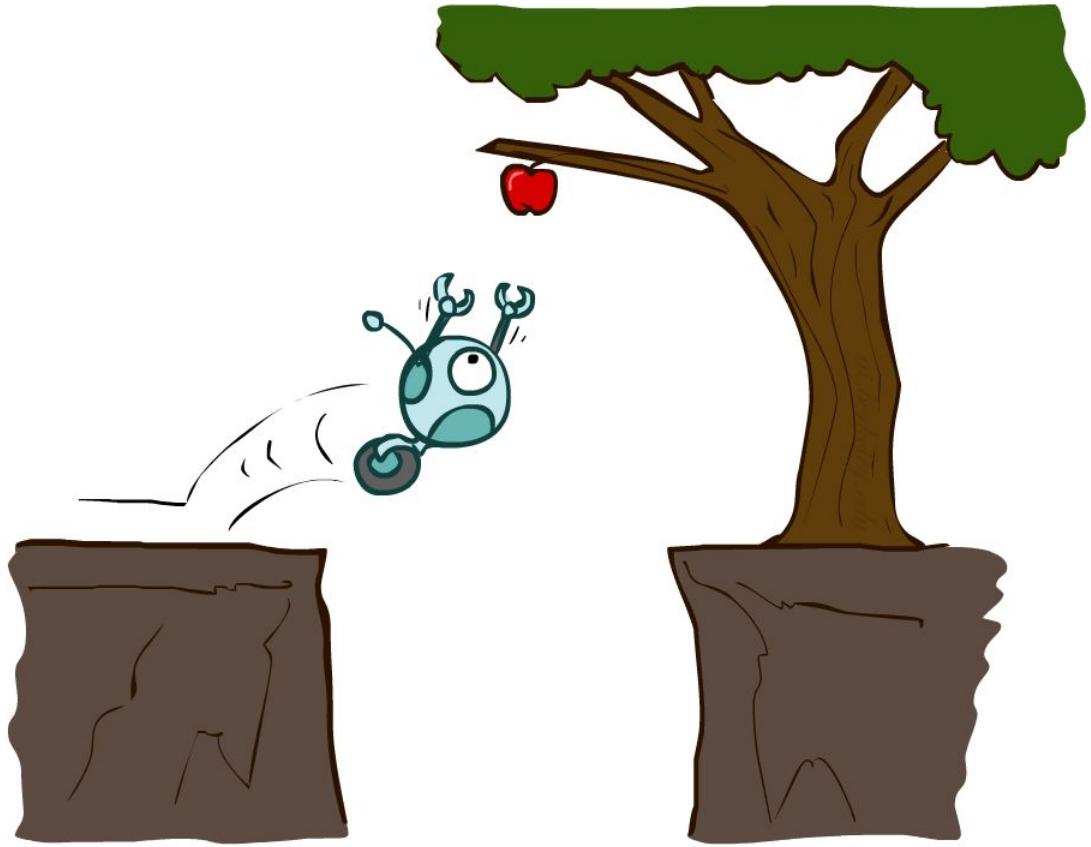
Agents



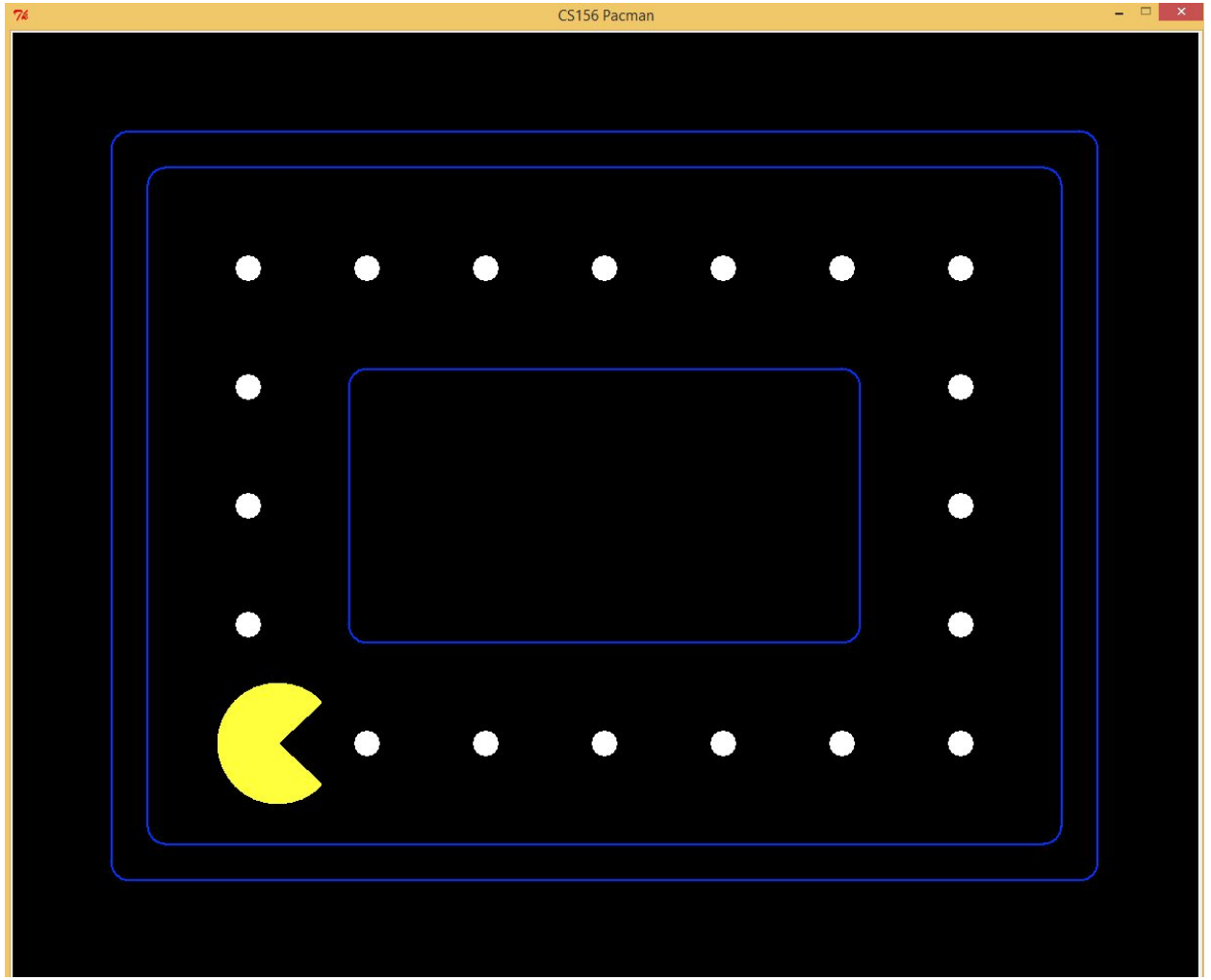


Reflex Agents

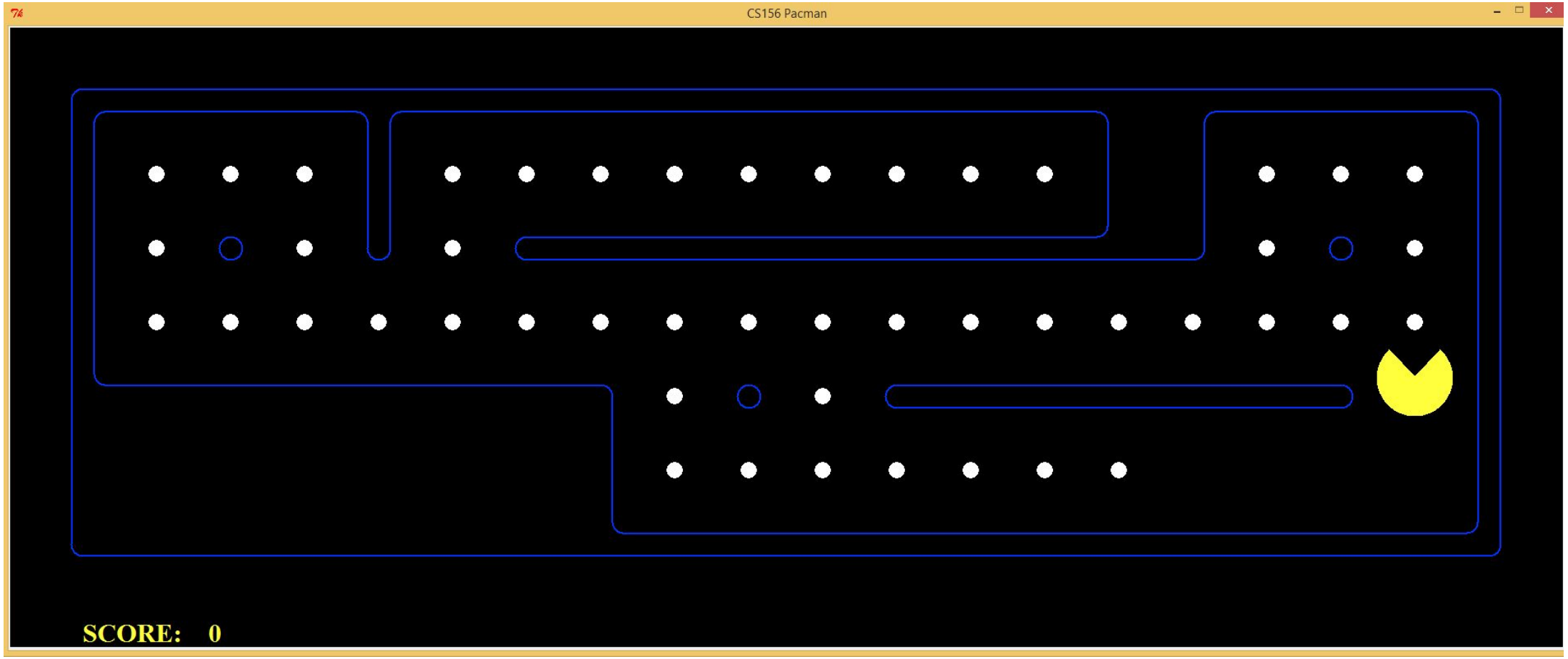
- Choose action based on current percept (and maybe memory)
- May have memory or a model of the world's current state
- Do not consider the future consequences of their actions
- Consider how the world IS



Demo Reflex Agent Optimal



Demo Reflex Agent Odd

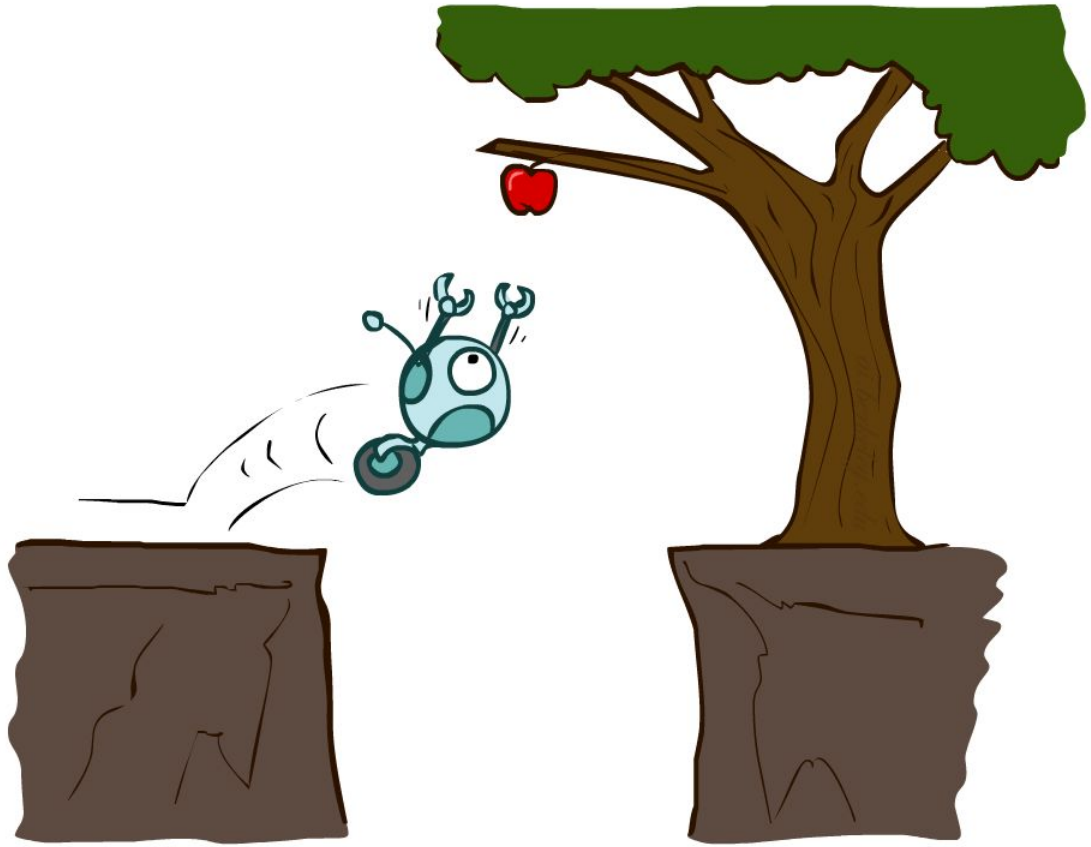


Reflex Agents

Can a reflex agent be rational?

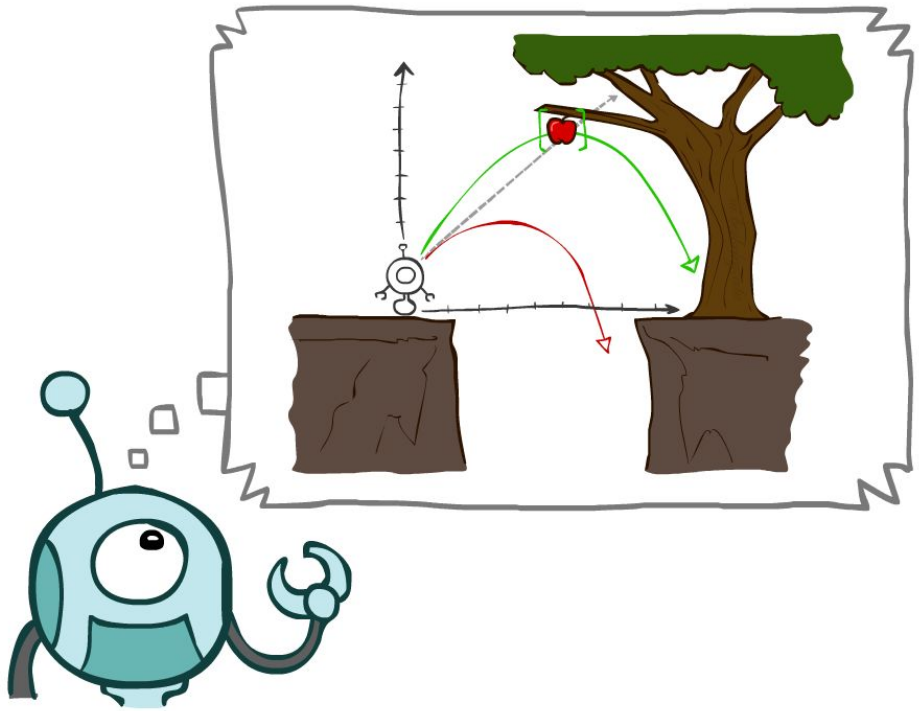
A. Yes

Rationality is a function of the actions taken, not of the thought process (or computation) that lead to them.



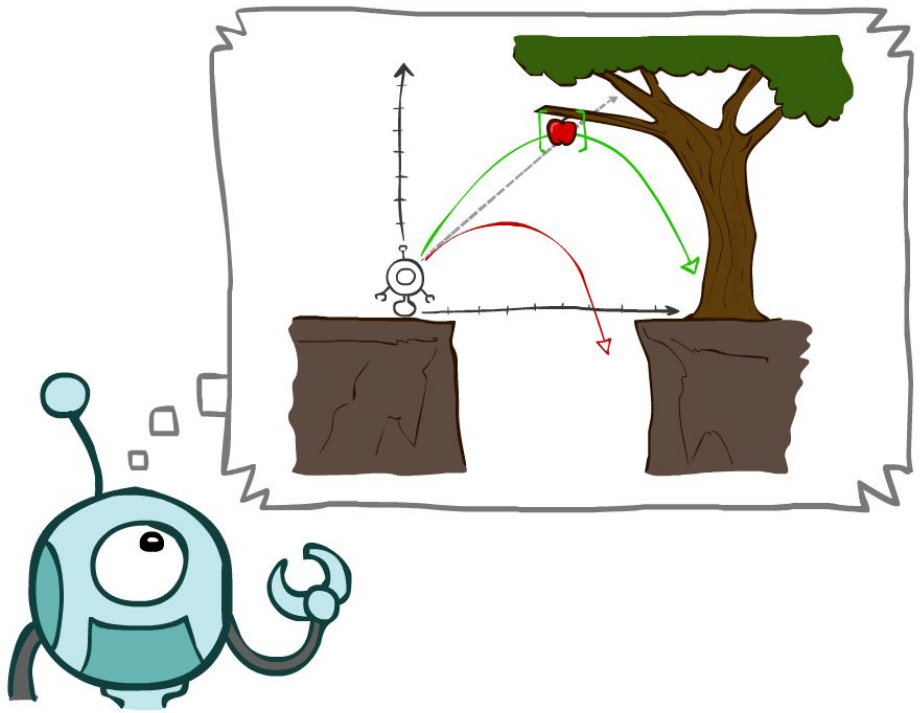
Problem Solving Agents

- 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



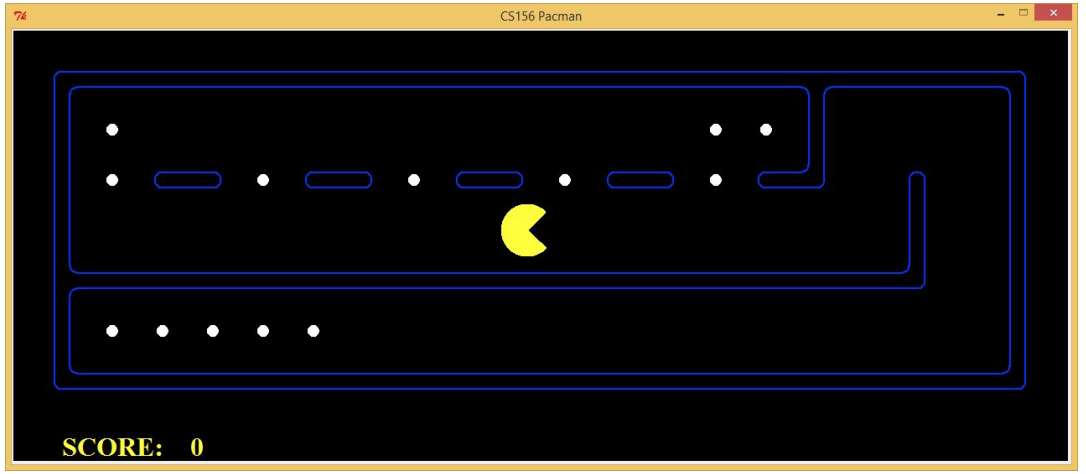
Problem Solving Agents

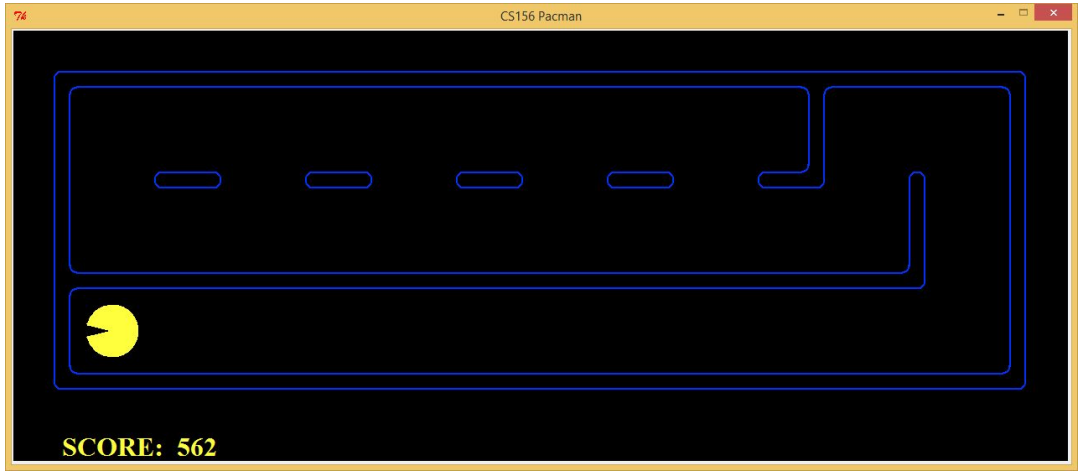
- Optimal vs. complete
  - Complete: the agent is guaranteed to find a solution
  - Optimal: the agent finds the best solution



Replanning Demo

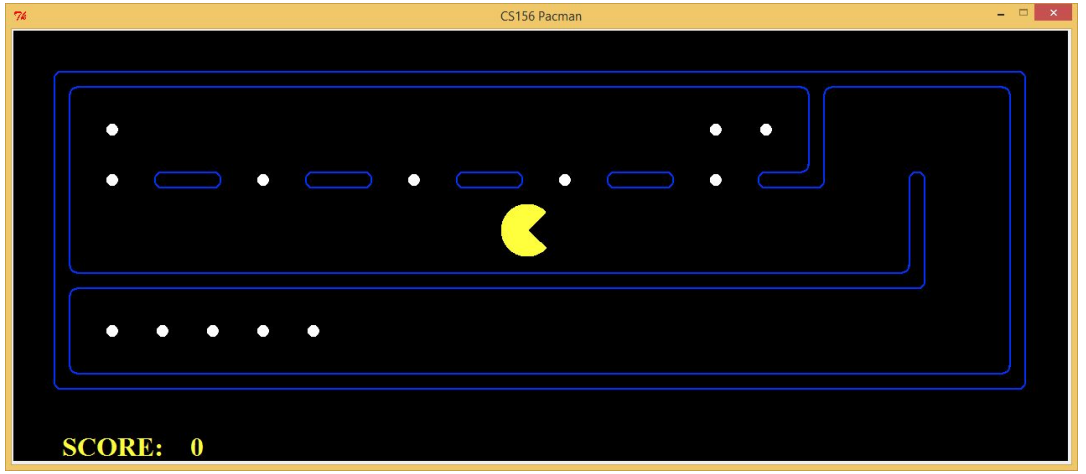
- It comes up with mini-plans and executes them one after the other
- Very fast
- It is complete: it found a way to achieve its goal of clearing the dots
- Cost 68 - Score 562
- May not be the most efficient way of clearing the dots

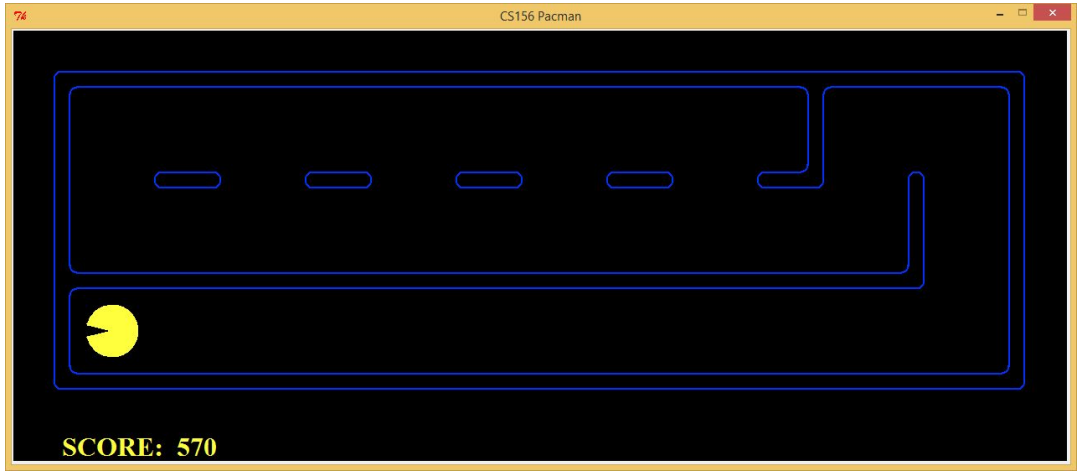




Planning Ahead Demo

- Spends ~ 3 seconds searching for the best way to clear all the dots
- Not a single wasted step
- It is complete: it found a way to achieve its goal of clearing the dots
- Cost 60 - Score 570
- It is optimal: it found the 'best' way to achieve its goal of clearing the dots





Problem Solving Agent Design

1. Formulate the problem
2. Search for a solution
3. Execute the solution

Search Problems - Goal

- Eat the closest dot
- Eat a dot
- Eat all the dots
- Eat as many dots as possible within a given time

Search Problems

Assumptions about the environment:

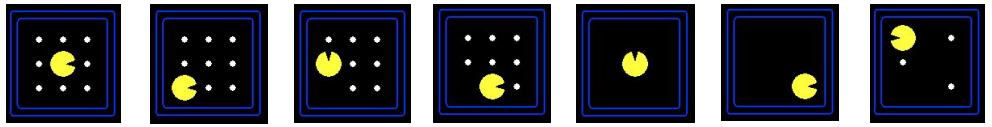
- a single agent
- deterministic
- fully observable
- discrete

Search Problems

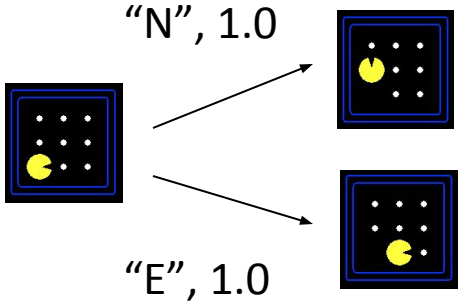
- A search problem consists of:
  - A state space: an abstraction that encodes how the world is at a given time.
  - A successor function: an abstraction that models how we think the world works, how it evolves in response to given actions. It includes actions and associated costs.
  - A start state
  - A goal test: all the dots are eaten, Pacman position?

Search Problems

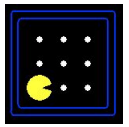
- A search problem consists of:
  - A state space



  - A successor function
    (with actions, costs)



  - A start state



  - A goal test: all the dots are eaten, Pacman position?

Search Problems

- A solution is a sequence of actions which transforms the start state to a state that satisfies the goal test.

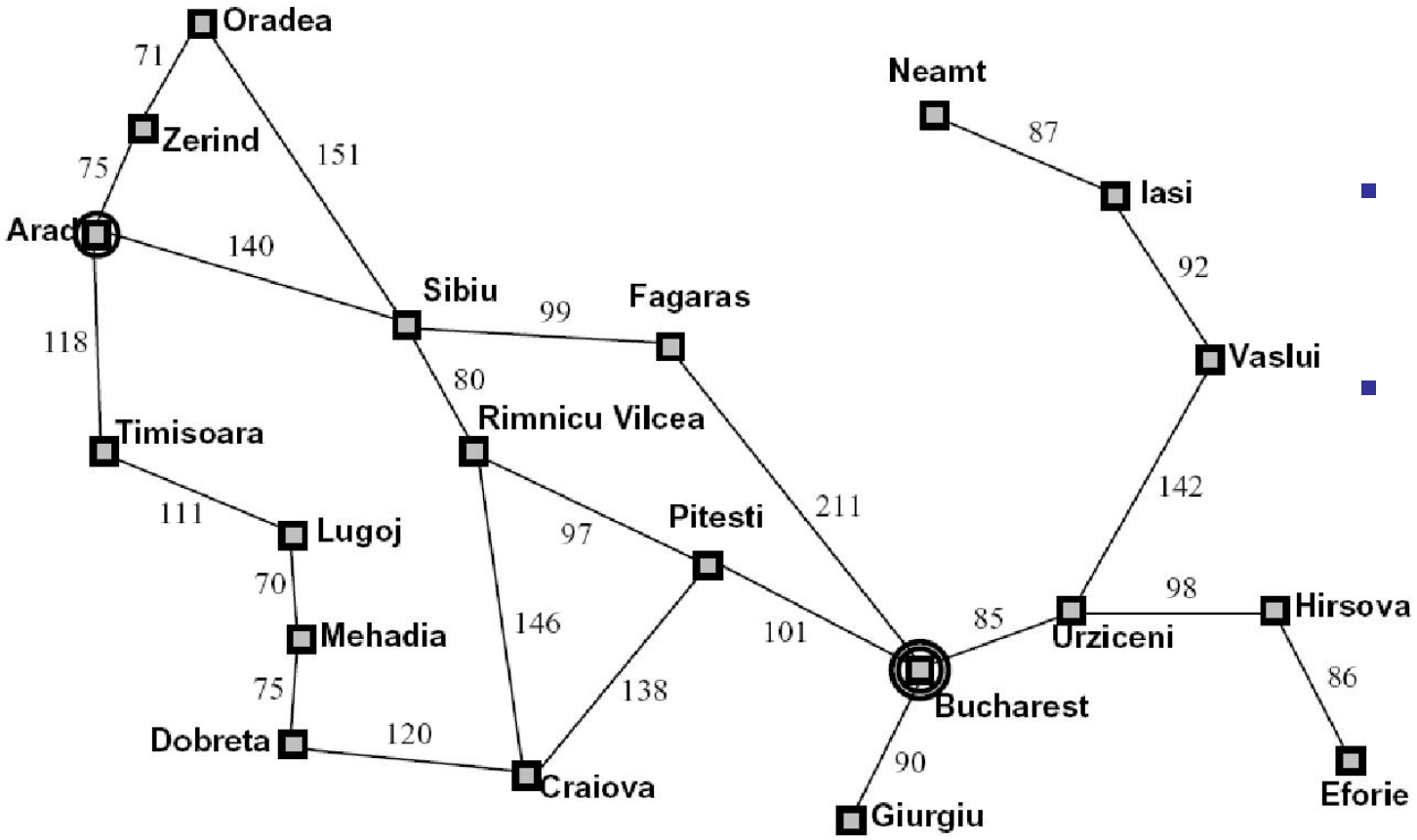
Search Problems Are Models

The real world is absurdly complex.

The state space must be abstracted for problem solving.

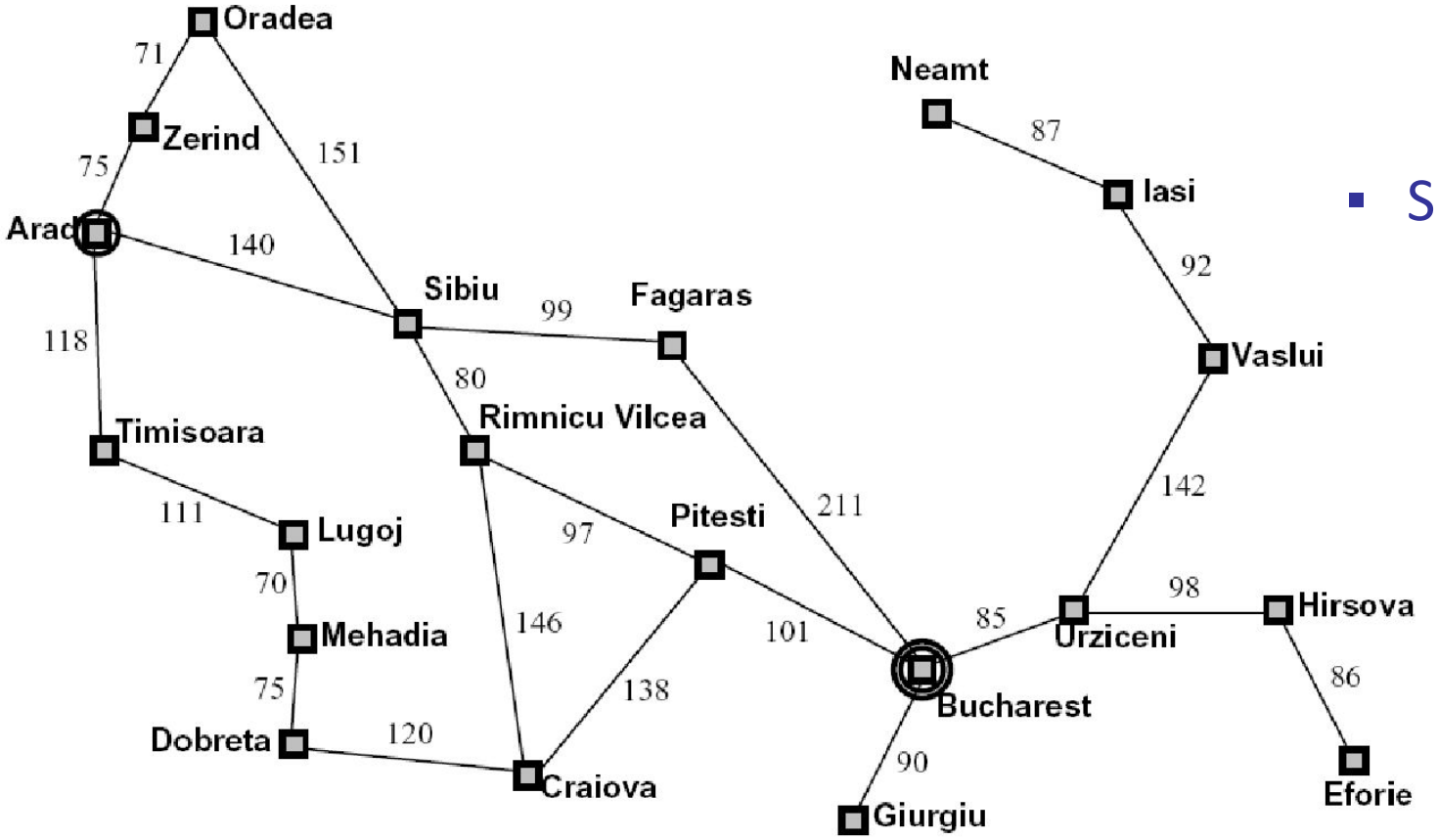
We only include relevant details.

Example: Traveling in Romania



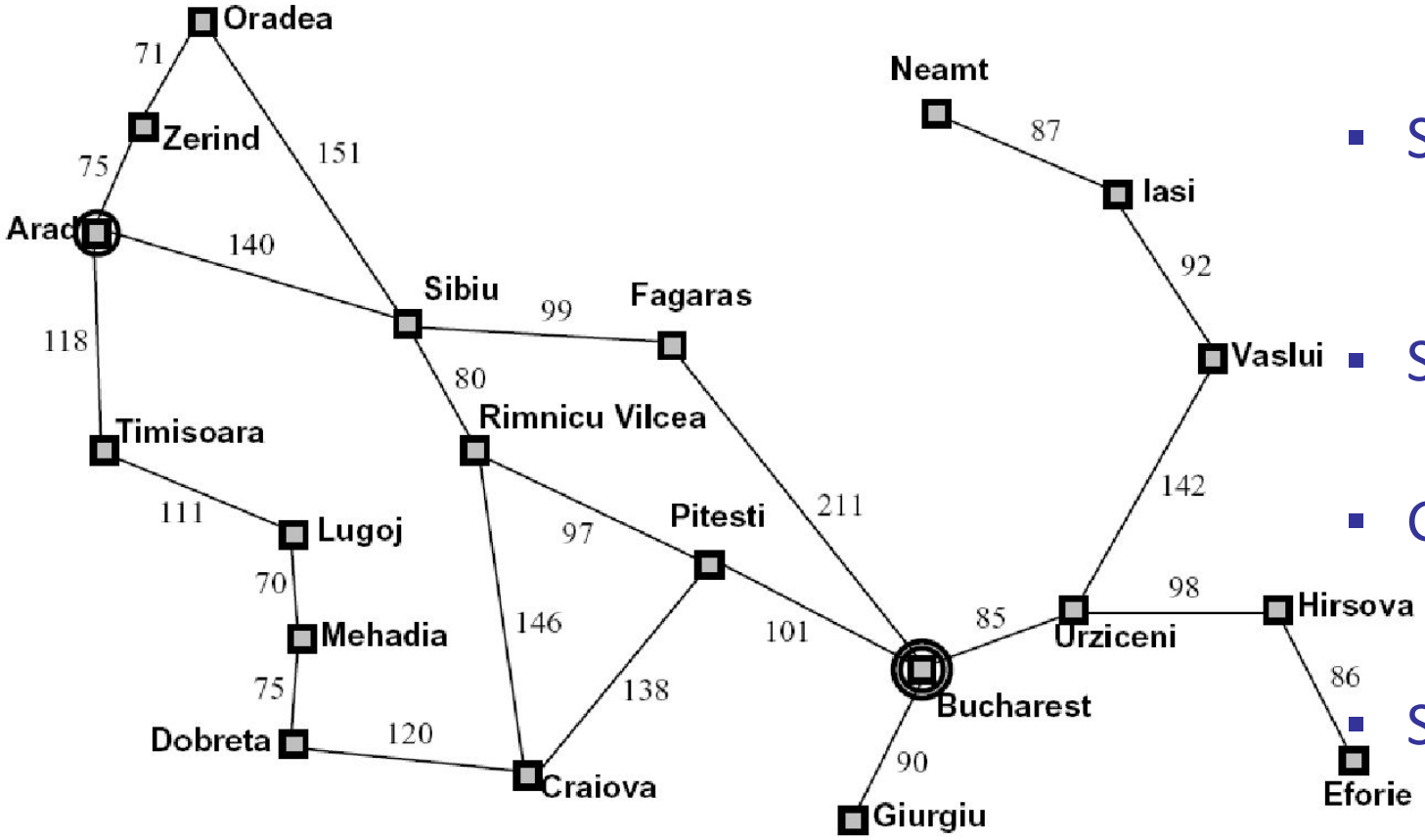
- We're in Arad and need to get to Bucharest.
- Our model is a map. It's an abstraction of the world.
- Our solutions will be only as good as our model.

Example: Traveling in Romania



- State space:
  - must be able to support the goal test: are we in Bucharest?
- Successor function:
  - How do we get from one state to another based on our model of how the world works

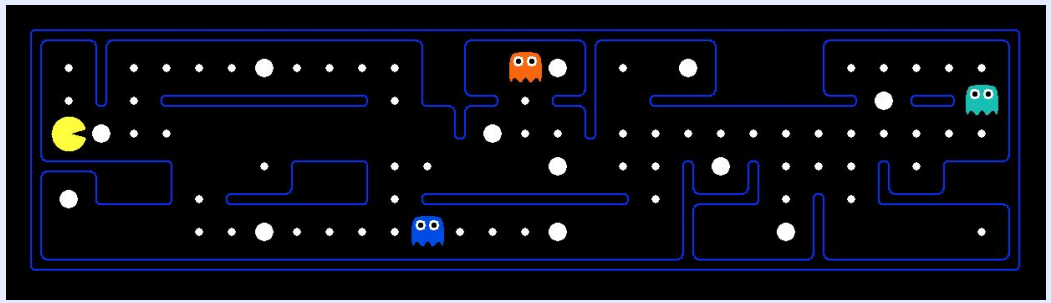
Example: Traveling in Romania



- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?
  - Sequence of actions that take me from Arad to Bucharest

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 the search (abstraction)

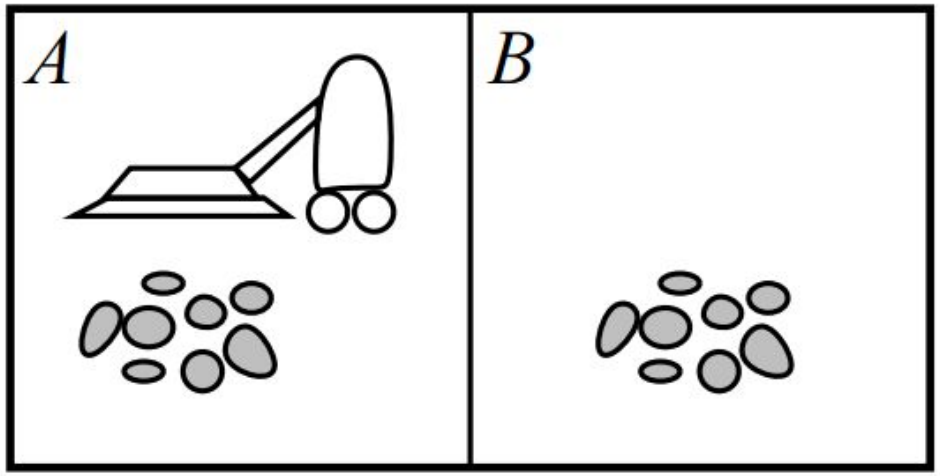
- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- 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:

- Vacuum positions: 2
- Dirt Locations: 2



How many world states?

$2x(2^2) = 8$

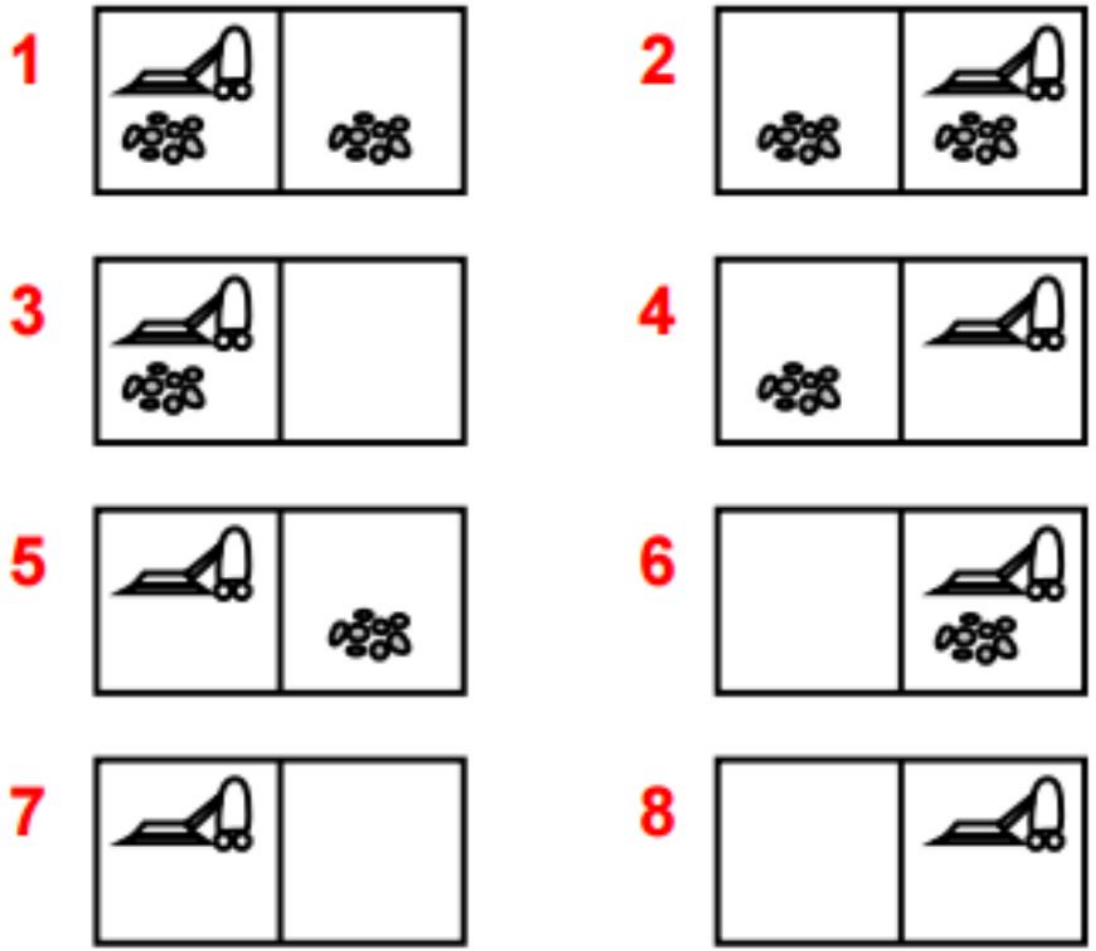
State Space Sizes?

World state:

- Vacuum positions: 2
- Dirt Locations: 2

How many world states?

$2x(2^2) = 8$



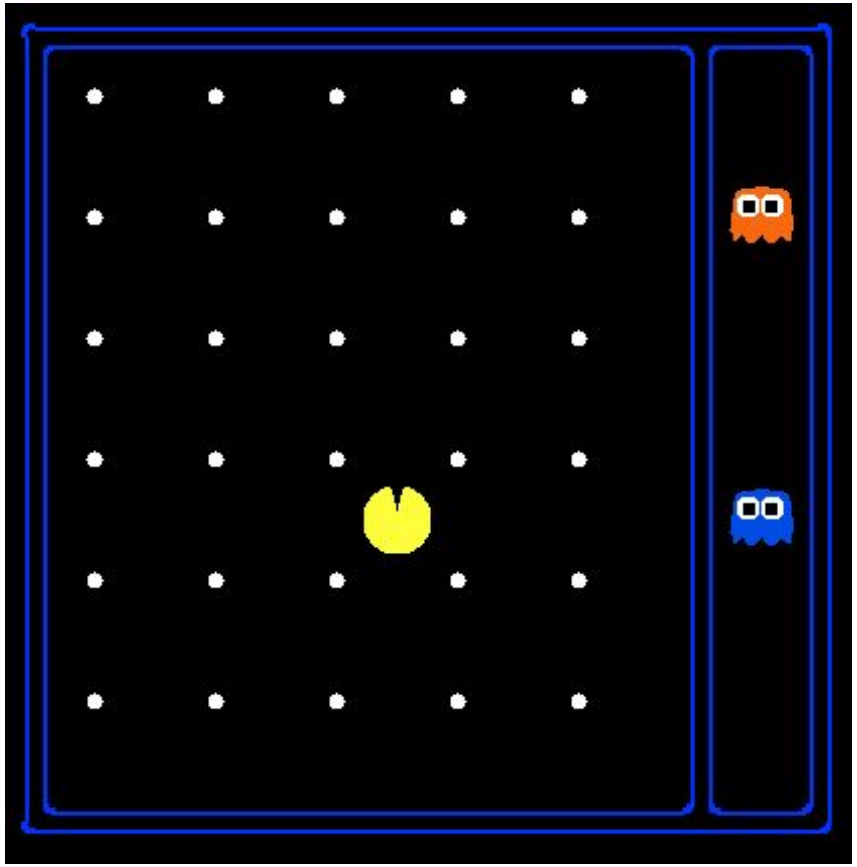
State Space Sizes?

World state:

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

How many world states?

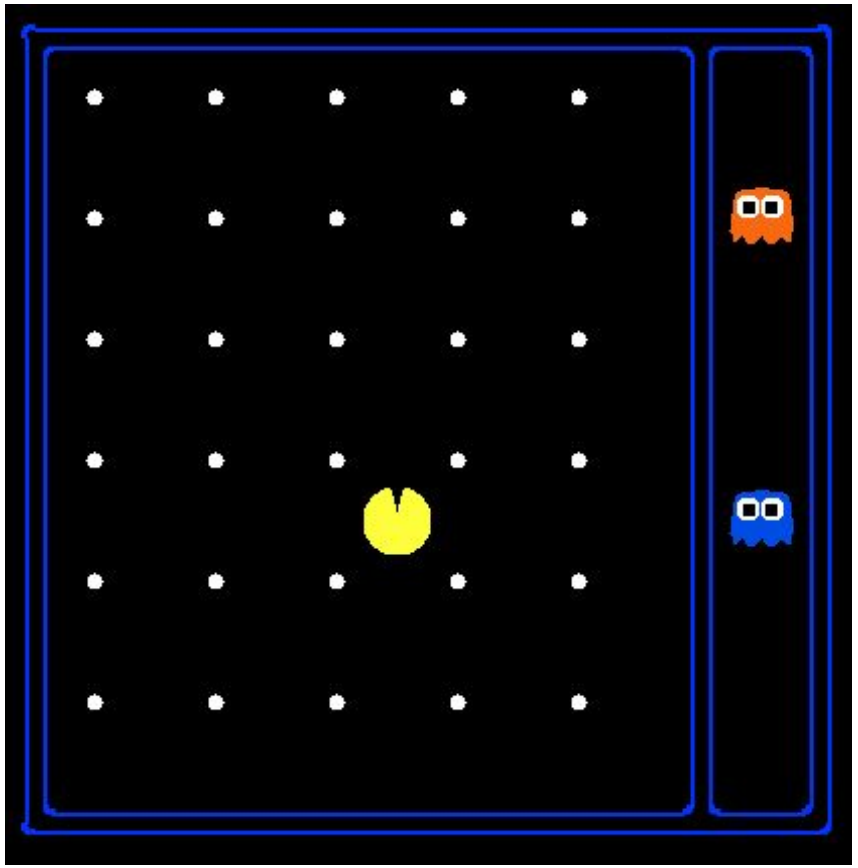
$120x(2^{30})x(12^2)x4$



Pathing

Problem:

- Get Pacman to the upper left corner



What does the state space have to specify?

- pacman position

How many states?

- 120

Eat All the Dots

Problem:

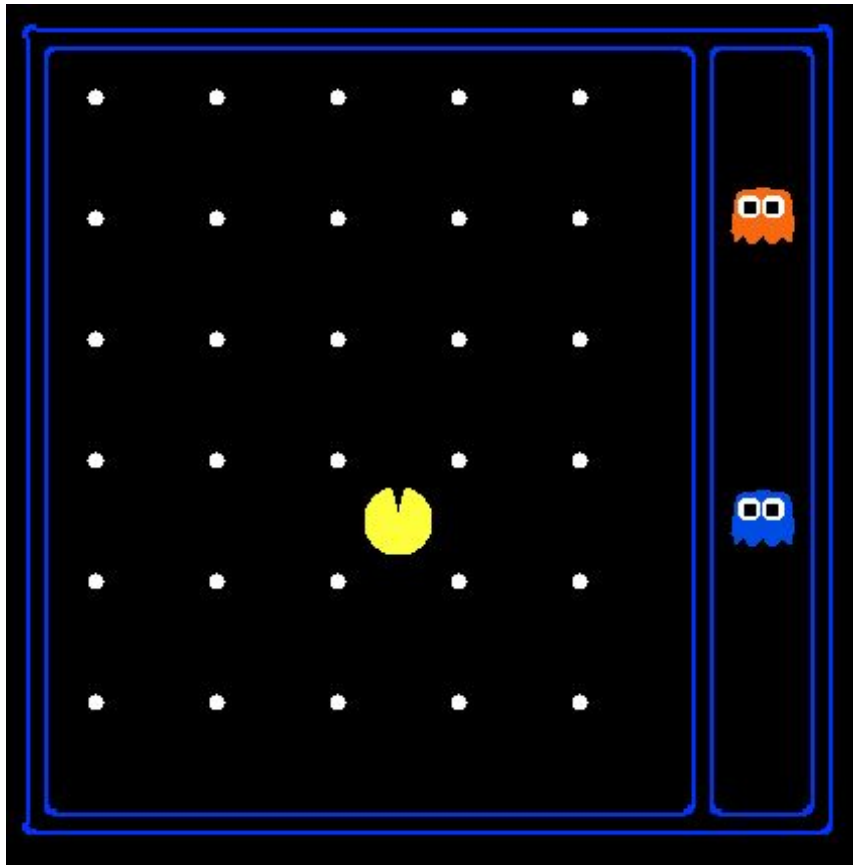
- Eat all the dots

What does the state space have to specify?

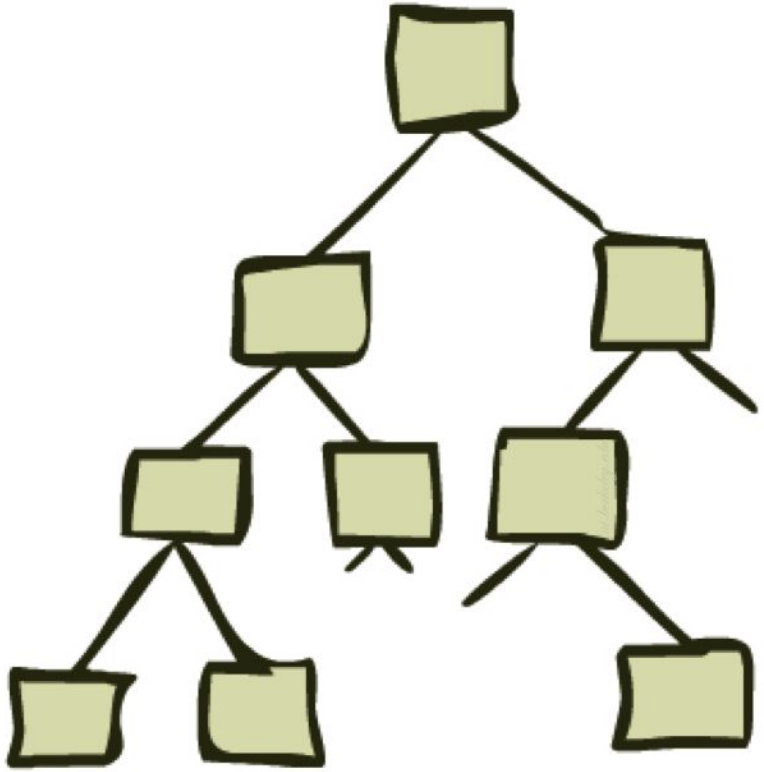
- pacman position, dot booleans

How many states?

- 120×2^{30}

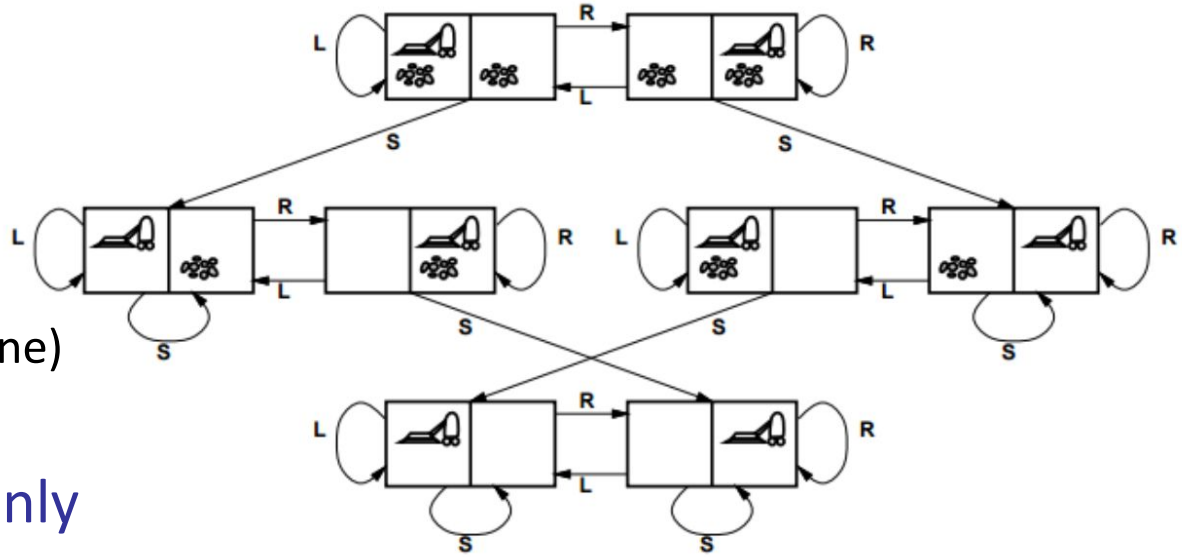


State Space Graphs and Search Trees



State Space Graphs

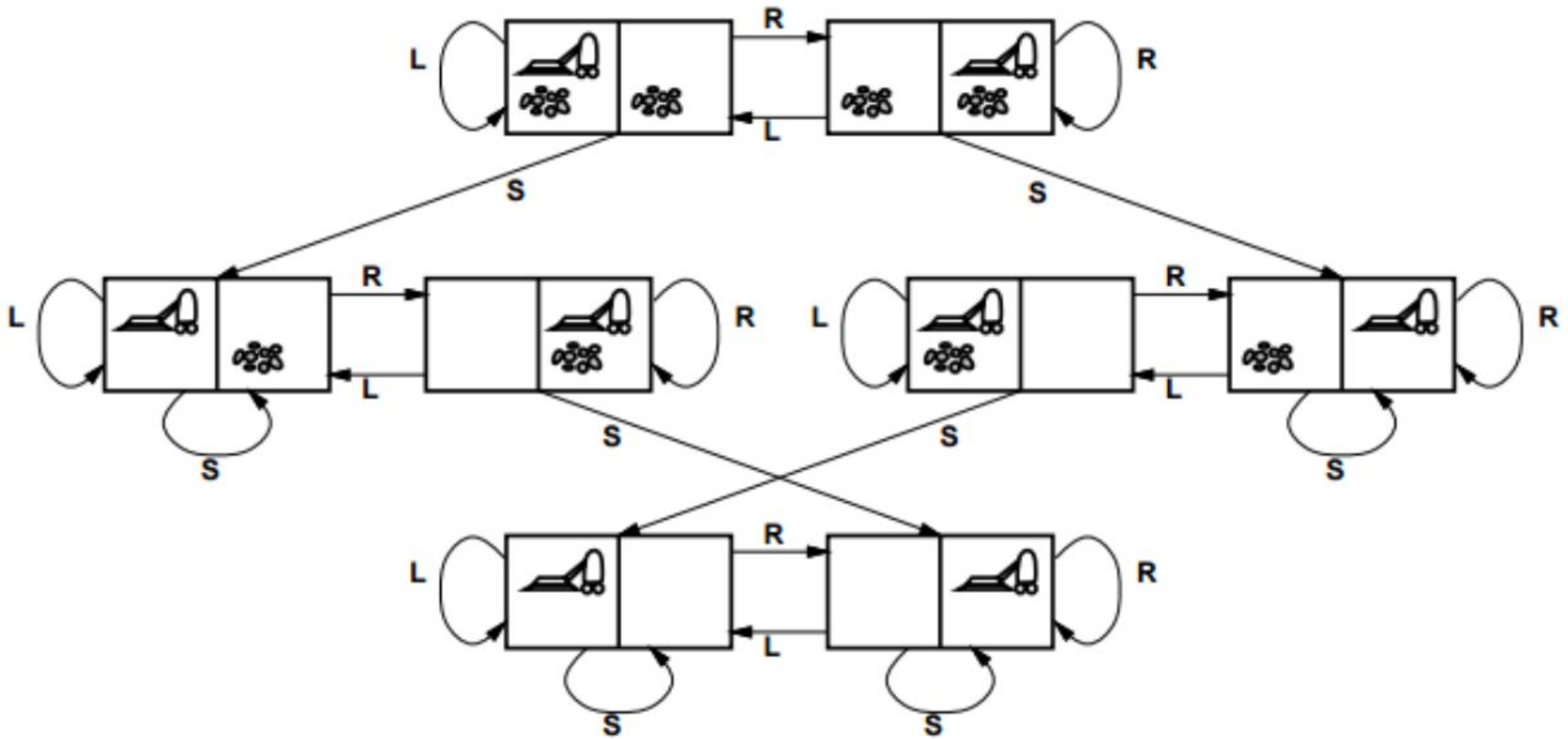
- State space graph: A mathematical representation of a search problem
  - States are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal states (maybe only one)



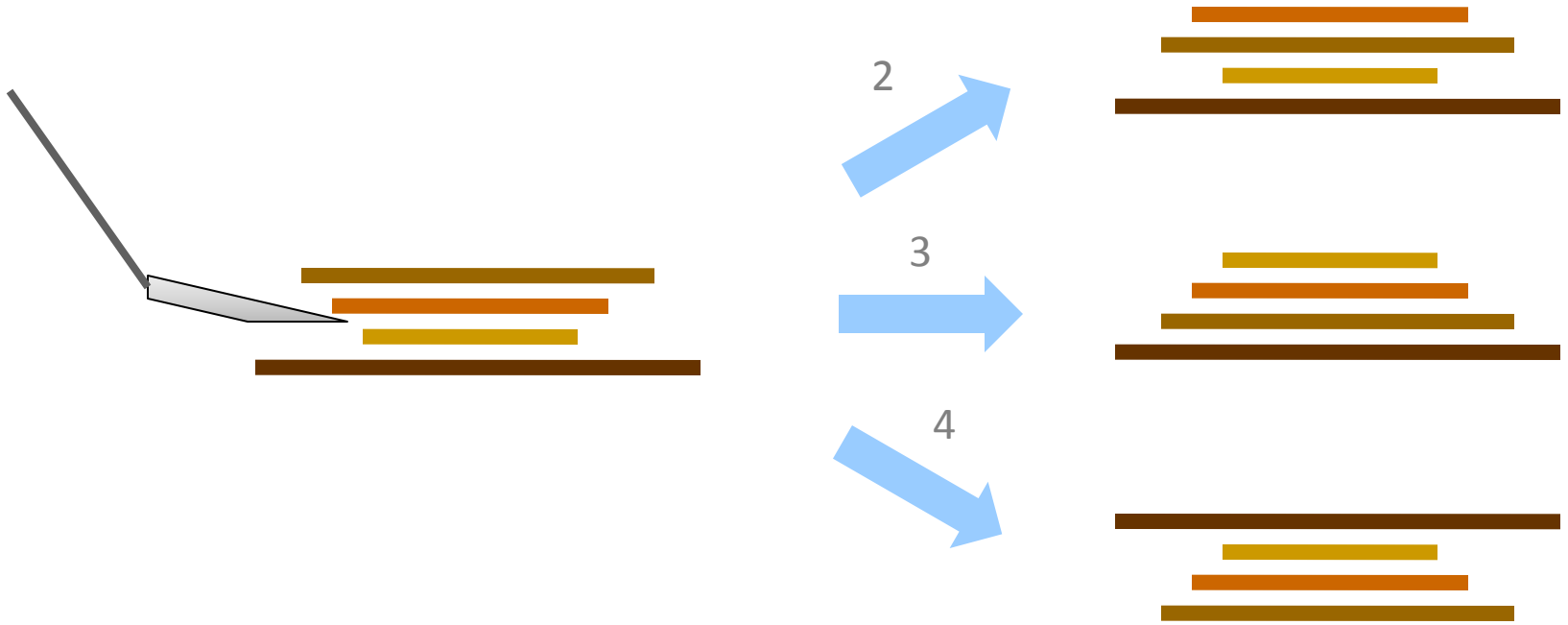
- In a state space graph, each state occurs only once!

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

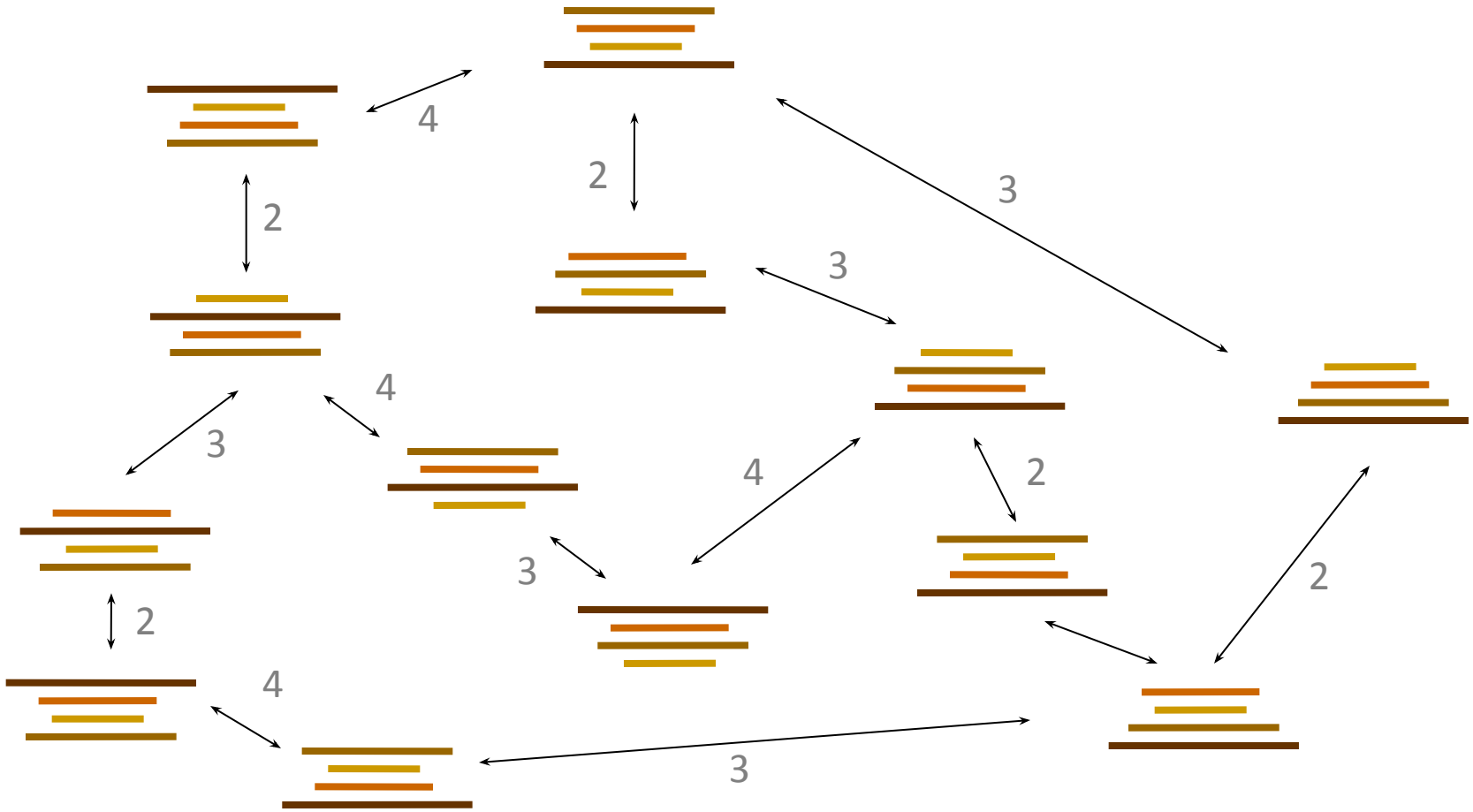
State Space Graphs



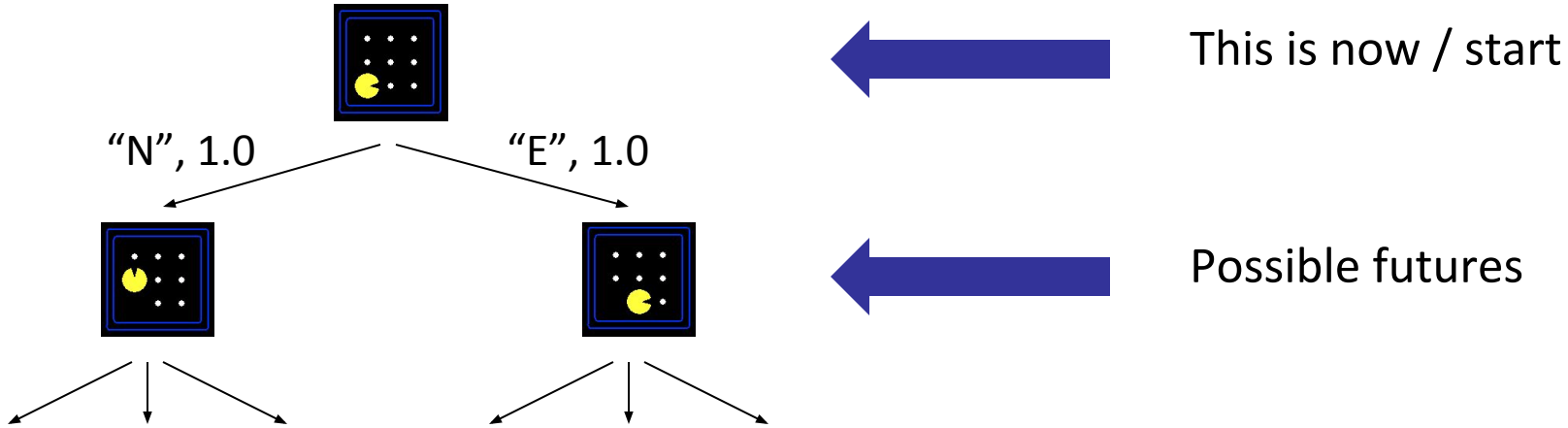
Example: Pancake Problem



Example: Pancake Problem

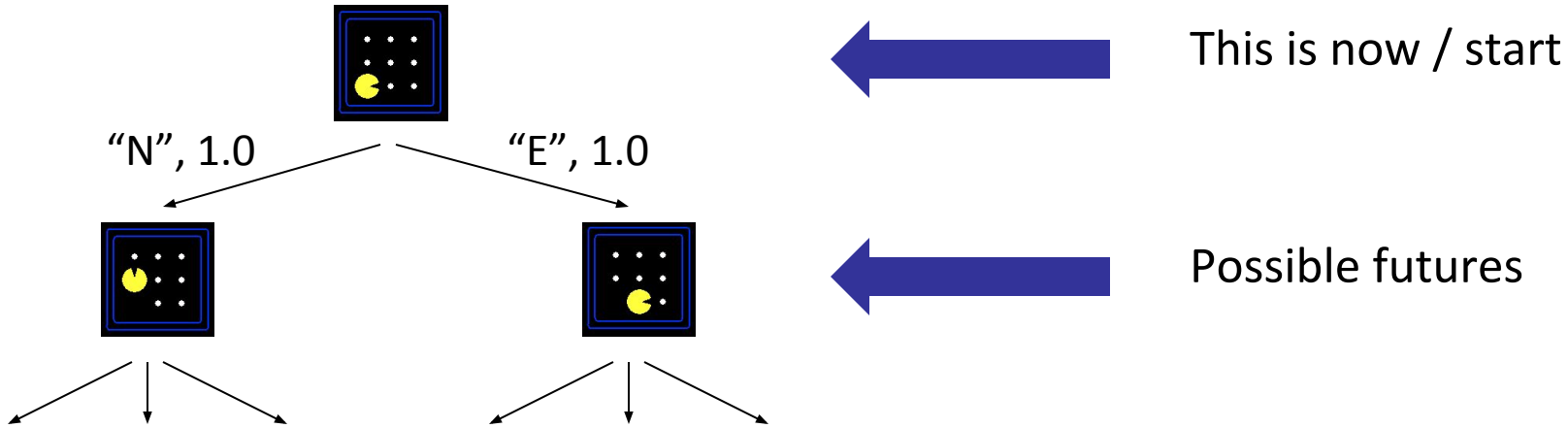


Search Trees



- 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

Search Trees: Nodes vs States



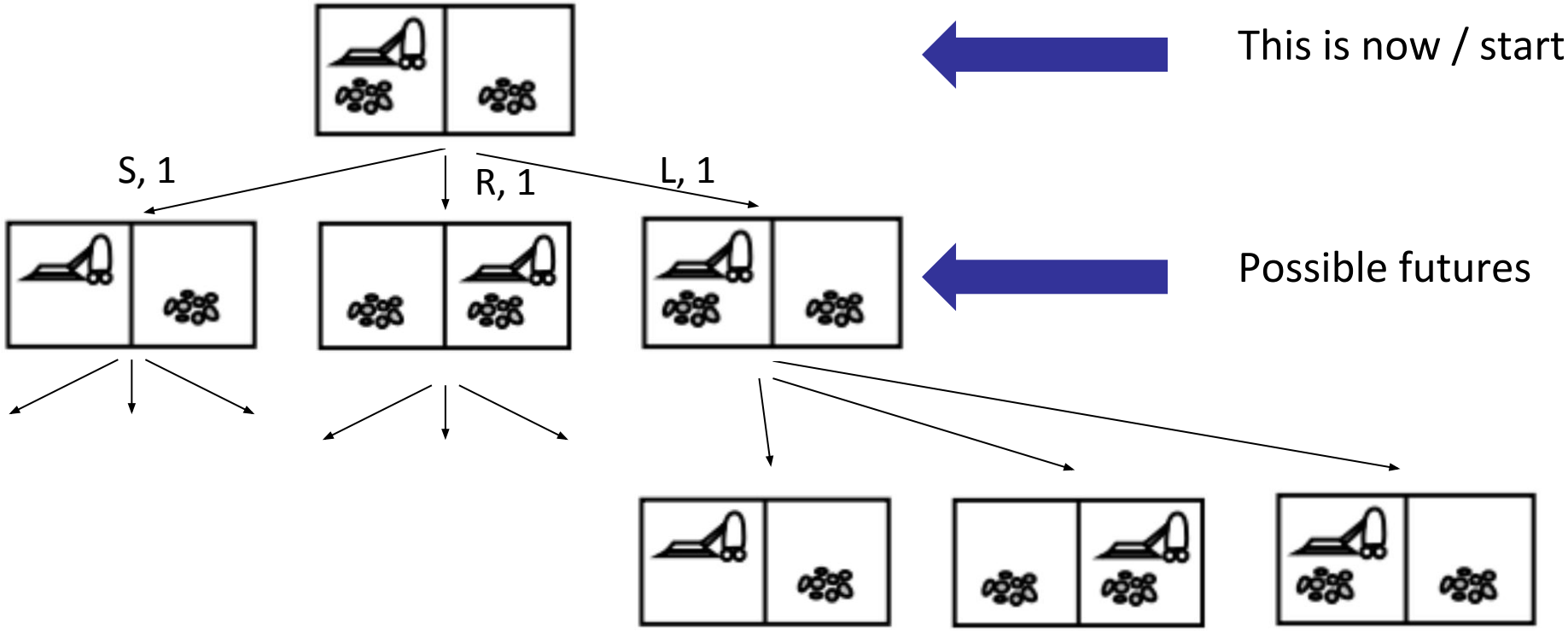
A state is a (representation of) a physical configuration

A node is a data structure that is part of a search tree

A node includes information about the state but also about the parent node, the action and the path cost

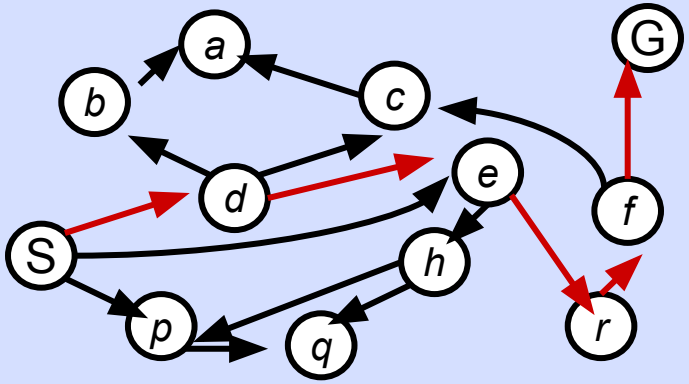
States do not have parents, actions or path cost.

Search Trees



State Space Graphs vs. Search Trees

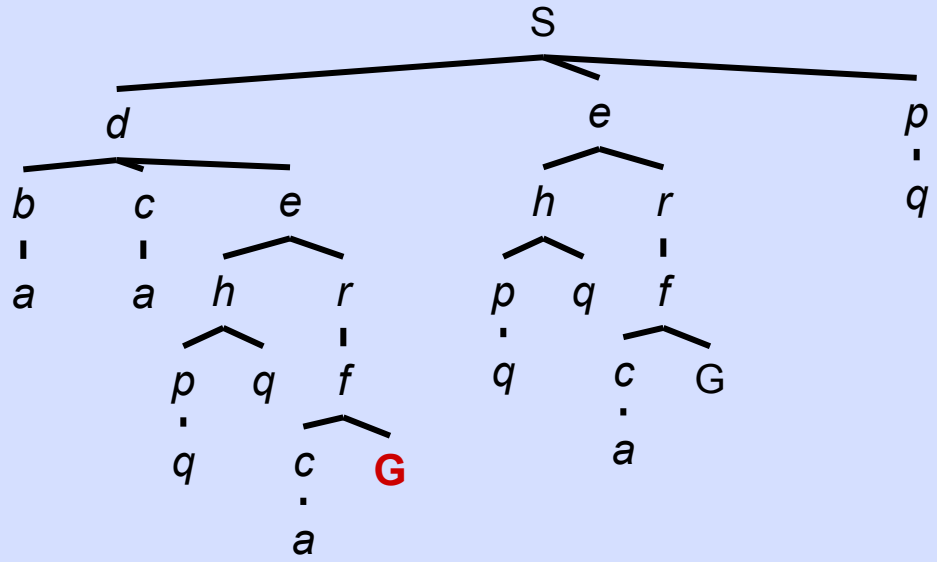
State Space Graph



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

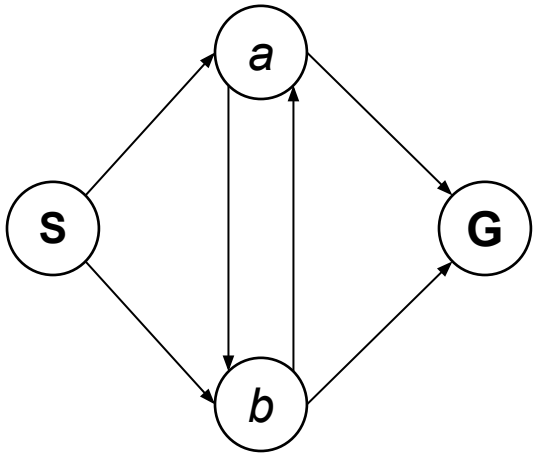
We construct both on demand – and we construct as little as possible.

Search Tree

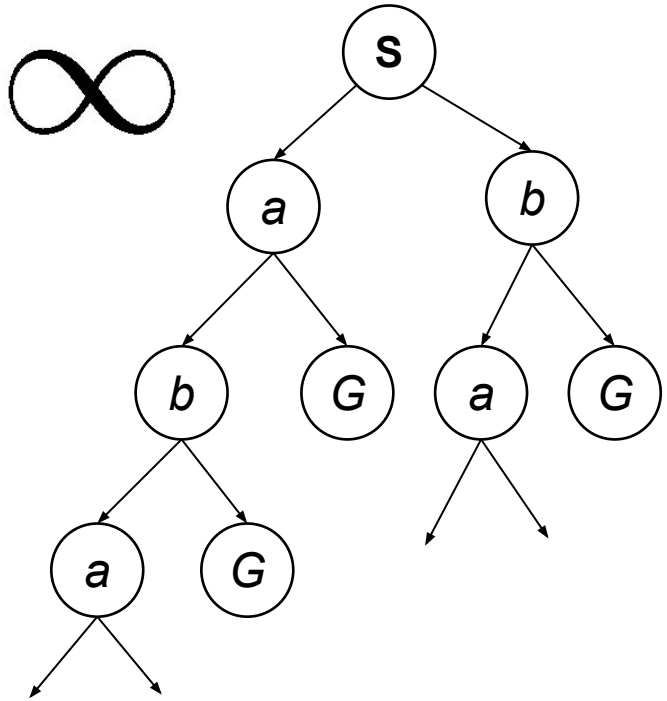


State Space Graphs vs. Search Trees

Consider this 4-state graph:

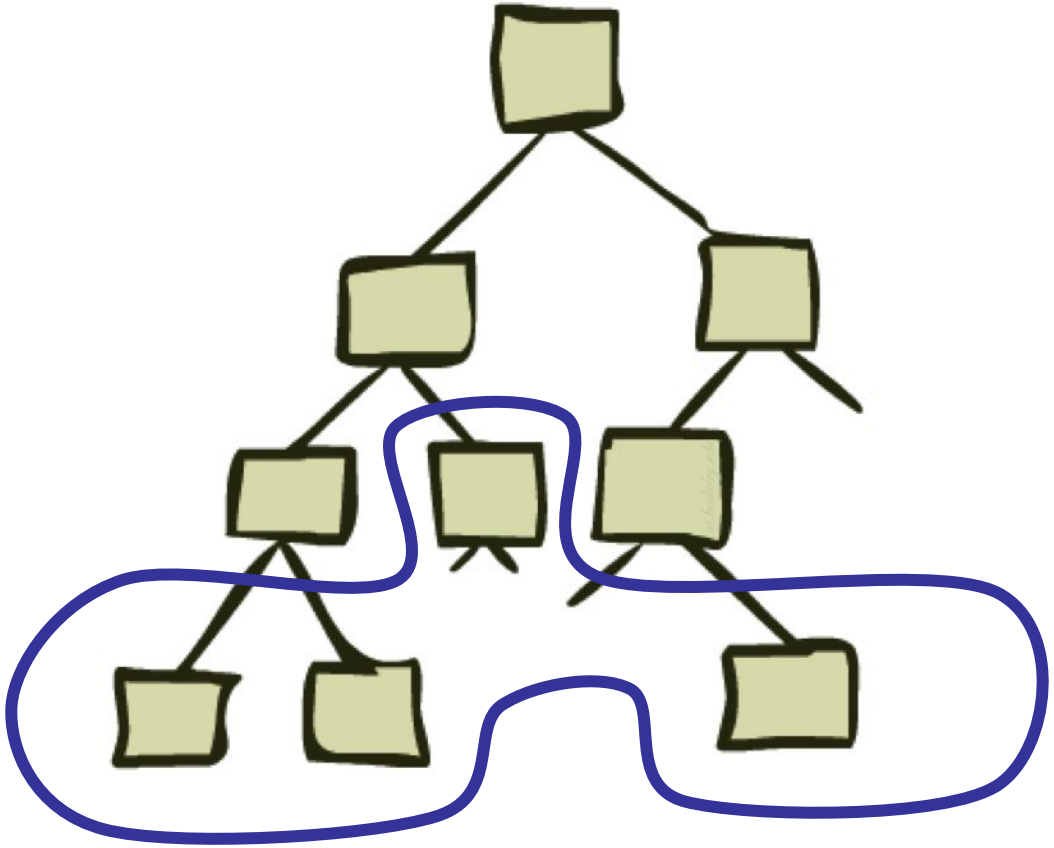


How big is its search tree (from S)?

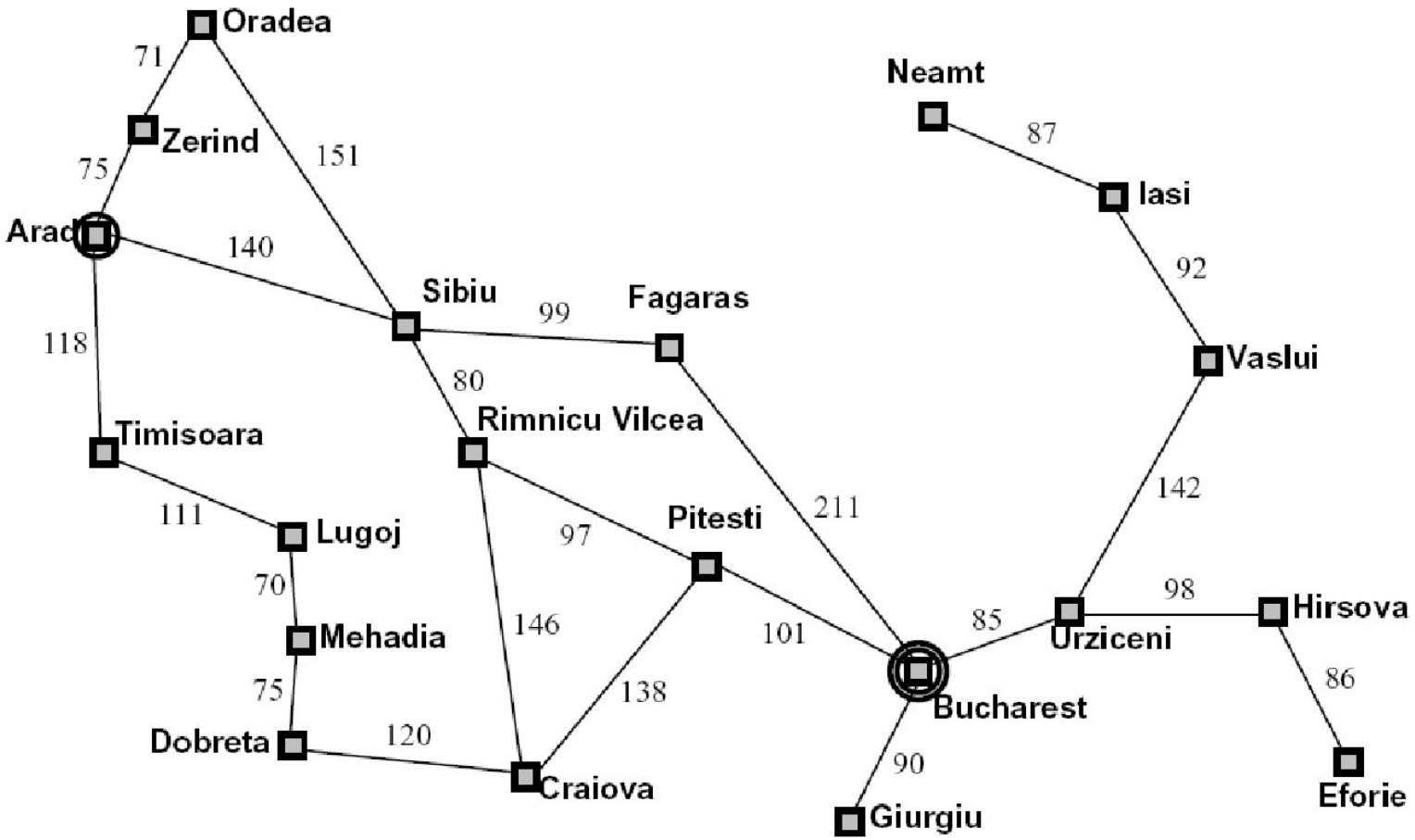


Important: Lots of repeated structure in the search tree!

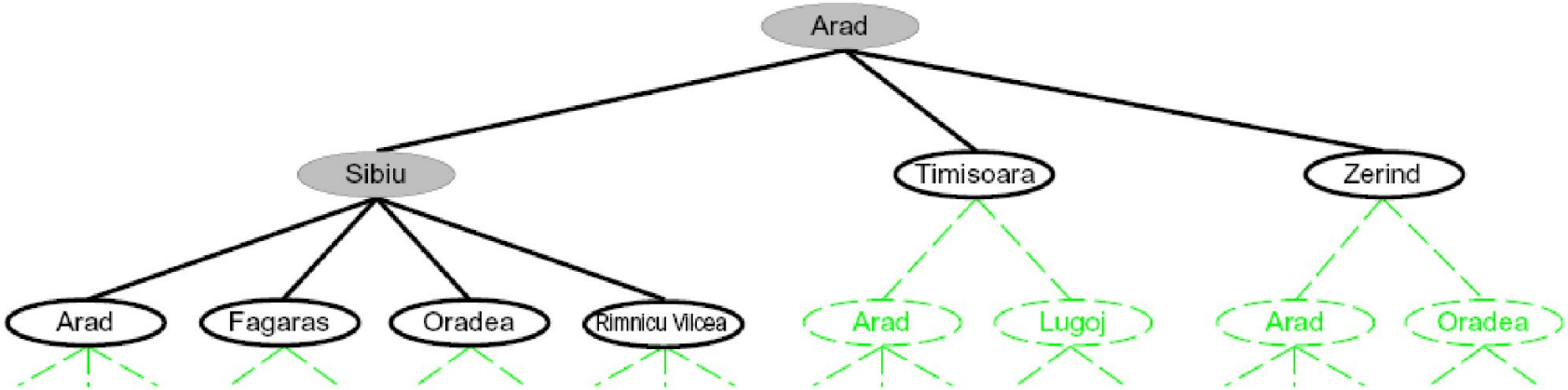
Tree Search



Search Example: Romania



Searching with a Search Tree



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe/ frontier 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/Frontier
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?