MAI 5100: Fundamentals of Artificial Intelligence

Instructor: Dr. Christopher Clarke

Overview & Agenda (Part 1)

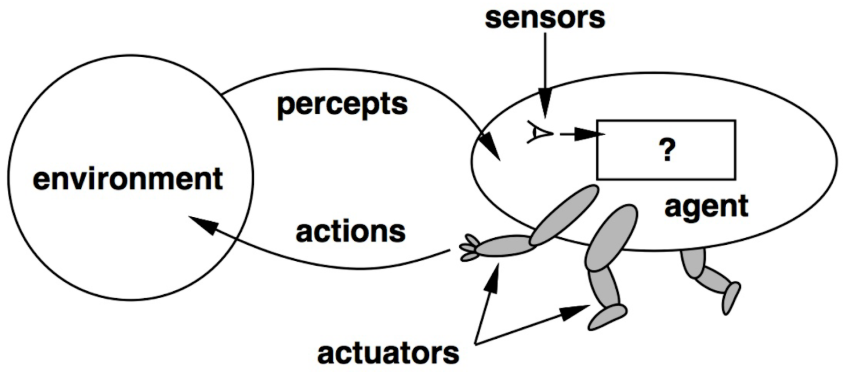
- Join the Slack Channel!
  - https://join.slack.com/t/uogmscai/shared_invite/zt-3118v5eqb-vpWNFUVtFYjKN_JUuAP~Ug
- Intelligent Agents
  - Agents and Environments
  - Good Behavior: The Concept of Rationality
  - The Nature of Environments
  - The Structure of Agents

Overview & Agenda (Part 2)

- Solving Problems by Searching
  - Problem-Solving Agents
  - Searching for Solutions
  - Uniformed Search Strategies
- Homework 0 (Released on GitHub)
  - **Due Date: Saturday, 25-Mar-2025 11:59 PM**
  - https://github.com/ChrisIsKing/MAI5100/tree/main/hw/hw0
- Group Formation
  - **Due Date: Today** Email your group members to christopher.clarke@uog.edu.gy

Intelligent Agents

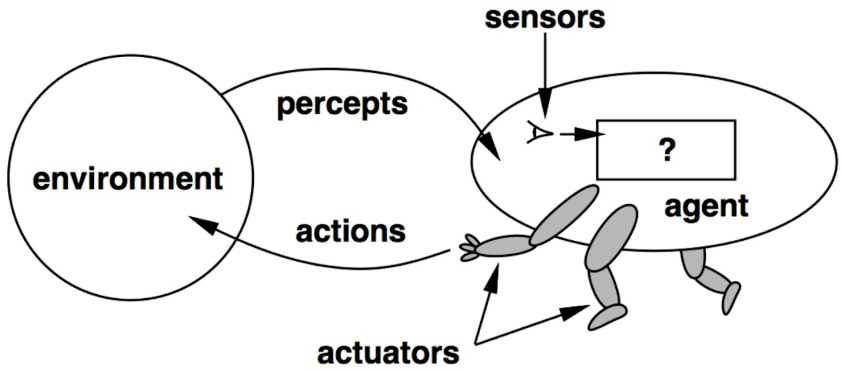
- **Intelligent Agents** are systems that perceive their environment and take actions to achieve specific goals.
- They can be physical entities (like robots) or virtual entities (like software agents).



- Agents can be simple reflex agents, model-based reflex agents, goal-based agents, or utility-based agents.
- The design of an intelligent agent involves understanding the environment, the tasks it needs to perform, and the actions it can take.

PEAS Principle

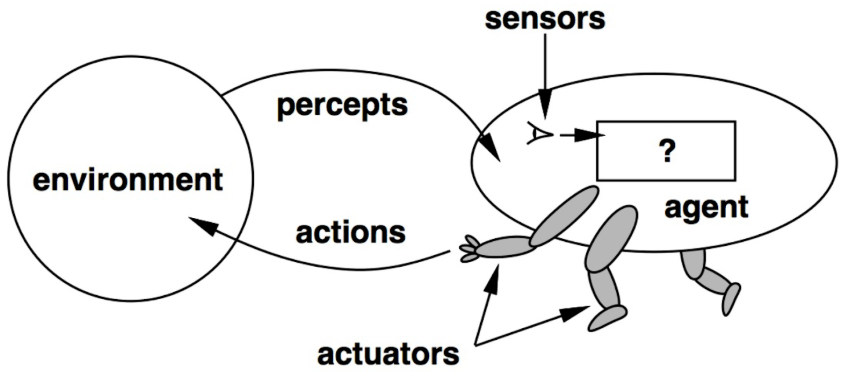
- **Performance Measure**: The criteria that determine how successful an agent is.
    - Example: For a self-driving car, the performance measure could include safety, speed, and passenger comfort.



- **Environment**: The external context in which the agent operates.
    - Example: For a self-driving car, the environment includes roads, traffic, pedestrians, and weather conditions.

PEAS Principle (Cont'd)

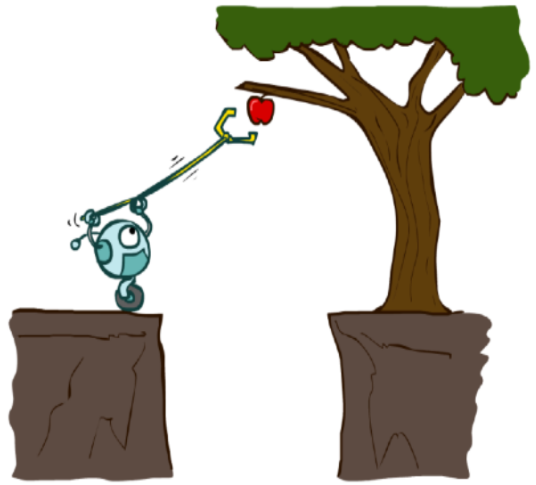
- **Actuators**: The mechanisms through which the agent acts on the environment.
    - Example: For a self-driving car, actuators include the steering wheel, accelerator, and brakes.

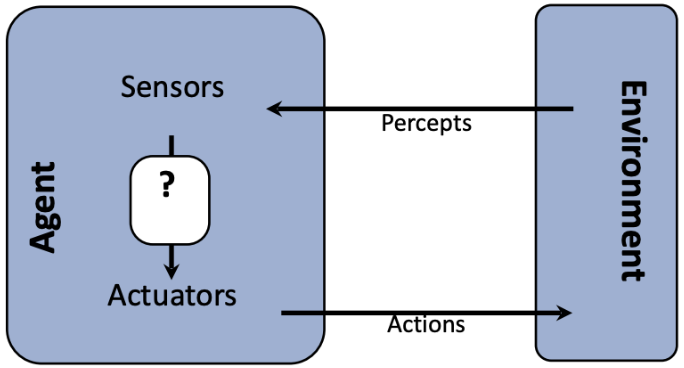


- **Sensors**: The mechanisms through which the agent perceives the environment.
    - Example: For a self-driving car, sensors include cameras, LIDAR, and GPS.

Good Behavior: The Concept of Rationality

- **Rationality**: An agent is rational if it acts to maximize its expected performance measure based on its knowledge.
- **Rational Agent**: An agent that selects actions that are expected to maximize its performance measure.





Environments

- **Fully Observable vs. Partially Observable**: Whether the agent has complete information about the environment.
    - Example: A chess game is fully observable, while a poker game is partially observable.
- **Deterministic vs. Stochastic**: Whether the environment's state is determined by the agent's actions or involves randomness.
    - Example: A chess game is deterministic, while a weather prediction is stochastic.

Environments (Cont'd)

- **Static vs. Dynamic**: Whether the environment changes while the agent is deliberating.
    - Example: A chess game is static, while a stock market is dynamic.
- **Discrete vs. Continuous**: Whether the environment has a finite number of states or an infinite number of states.
    - Example: A chess game has discrete states, while a robot navigating a room has continuous states.
- **Single-Agent vs. Multi-Agent**: Whether the environment involves multiple agents competing or cooperating.
    - Example: A chess game is single-agent, while a soccer game is multi-agent.

Environments (Cont'd)

- **Episodic vs. Sequential**: Whether the agent's current decision only depends on the current percept or also on past actions.
    - Example: A chess game is sequential, while a spam filter is episodic.

Task Environments

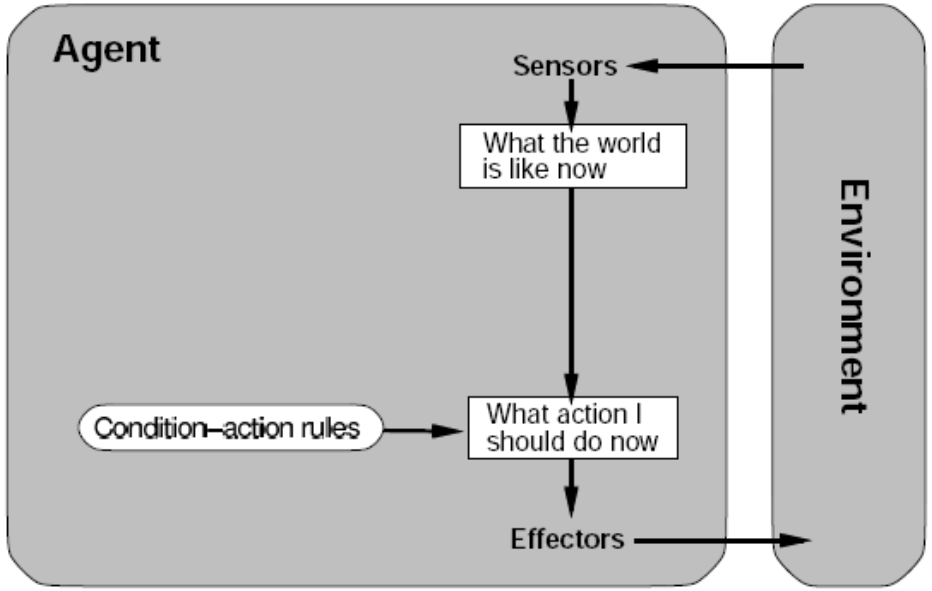
Task	Fully / Partially	Deterministic / Stochastic	Static / Dynamic	Discrete / Continuous	Single / Multi	Episodic / Sequential
Chess						
Poker						
Stock Market						
Robot Nav						
Email Filter						
Weather						

Task Environments (Filled)

Task	Fully / Partially	Deterministic / Stochastic	Static / Dynamic	Discrete / Continuous	Single / Multi	Episodic / Sequential
Chess	Fully	Deterministic	Static	Discrete	Single	Sequential
Poker	Partially	Stochastic	Static	Discrete	Multi	Sequential
Stock Market	Partially	Stochastic	Dynamic	Continuous	Multi	Sequential
Robot Nav	Partially	Deterministic	Dynamic	Continuous	Single	Sequential
Email Filter	Partially	Stochastic	Dynamic	Discrete	Single	Episodic
Weather	Partially	Stochastic	Dynamic	Continuous	Single	Episodic

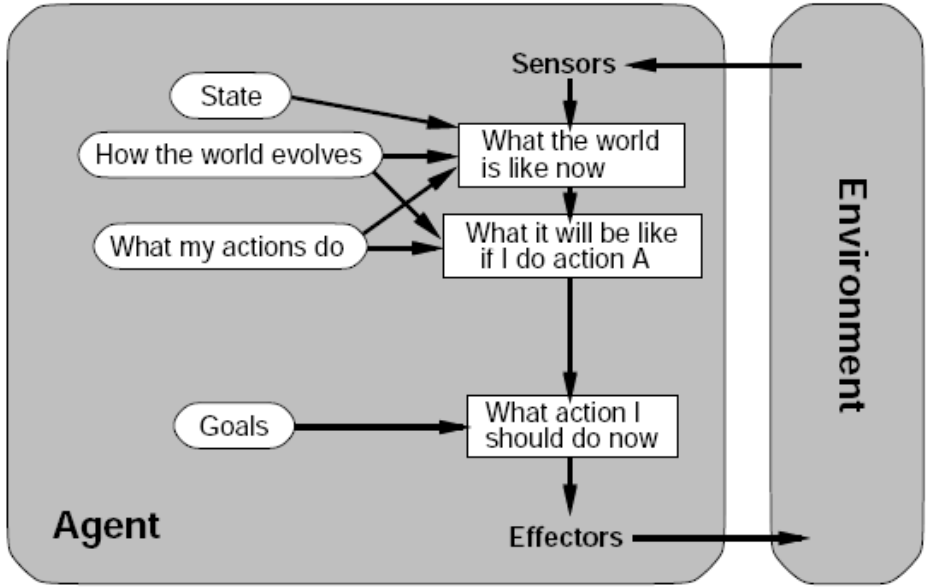
Types of Agents

- **Simple Reflex Agents**: Act based on current percepts and a set of condition-action rules.
    - Example: A thermostat that turns on the heater when the temperature drops below a certain threshold.
- **Model-Based Reflex Agents**: Maintain an internal state to keep track of the world and use it to make decisions.
    - Example: A robot that uses sensors to detect obstacles and adjust its path accordingly.



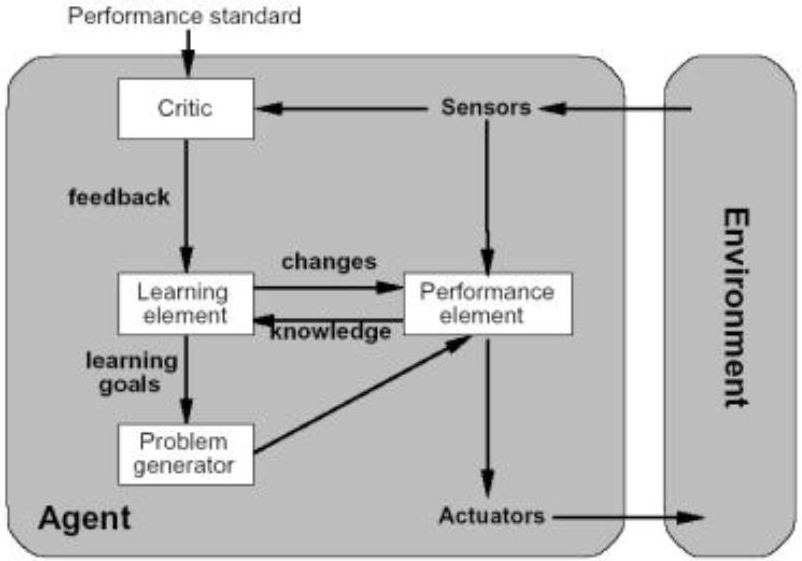
Types of Agents (Cont'd)

- **Goal-Based Agents**: Use a goal to determine the best action to take.
    - Example: A navigation system that calculates the best route to a destination.
- **Utility-Based Agents**: Consider multiple goals and their relative importance to choose the best action.
    - Example: A self-driving car that prioritizes safety, speed, and comfort when making driving decisions.



Types of Agents (Cont'd)

- **Learning Agents**: Improve their performance over time by learning from their experiences.
    - Example: A recommendation system that learns user preferences based on past interactions.
- **Multi-Agent Systems**: Involve multiple agents that can cooperate or compete to achieve their goals.
    - Example: A swarm of drones working together to map an area.



Problem-Solving Agents

- When the correct action to take is not immediately obvious, an agent may need to to plan ahead: to consider a sequence of actions that form a path to a goal state. Such an agent is called a **problem-solving agent**, and the computational process it undertakes is called **search**.
- **Problem-Solving Agent**: An agent that can solve problems by searching through a space of possible actions and states.

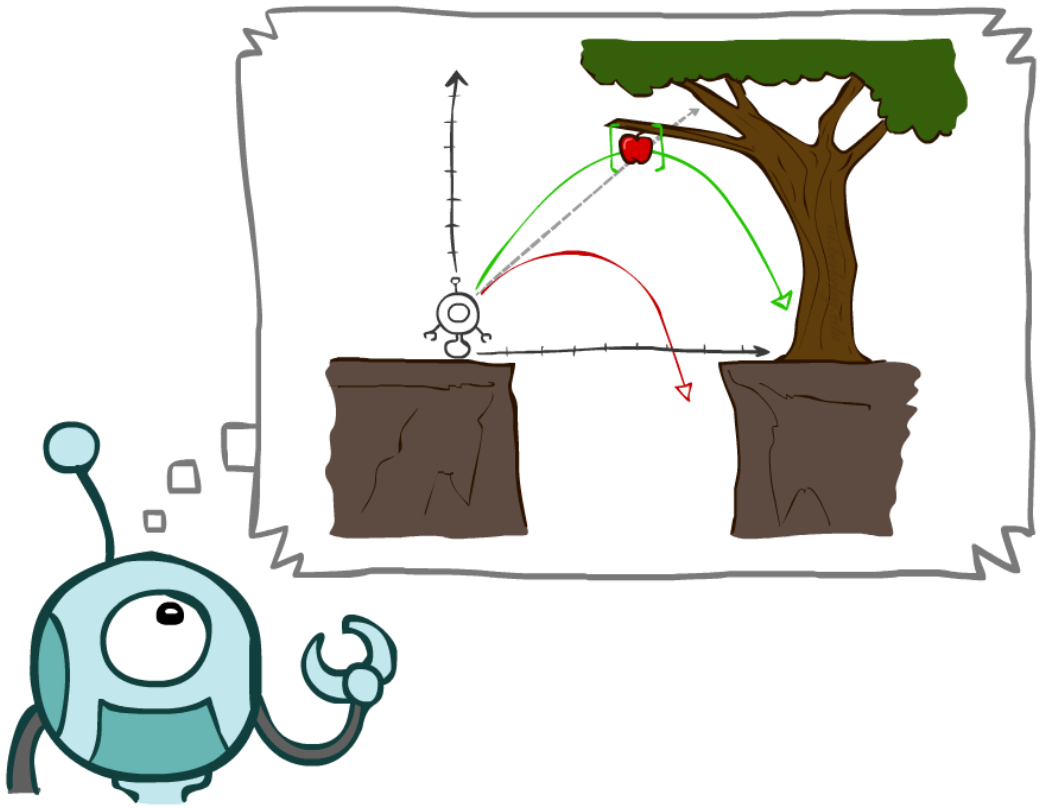
Problem-Solving Agents (Cont'd)

- **Problem**: A problem is defined by a **problem formulation**, which includes:
  - **Initial State**: The state of the world at the beginning of the problem.
  - **Actions**: The set of actions that can be taken to change the state of the world.
  - **Transition Model**: A description of how actions change the state of the world.
  - **Goal State**: The desired state of the world that the agent is trying to achieve.
  - **Path Cost**: A function that assigns a cost to each path from the initial state to the goal state.
  - **Solution**: A sequence of actions that leads from the initial state to the goal state.

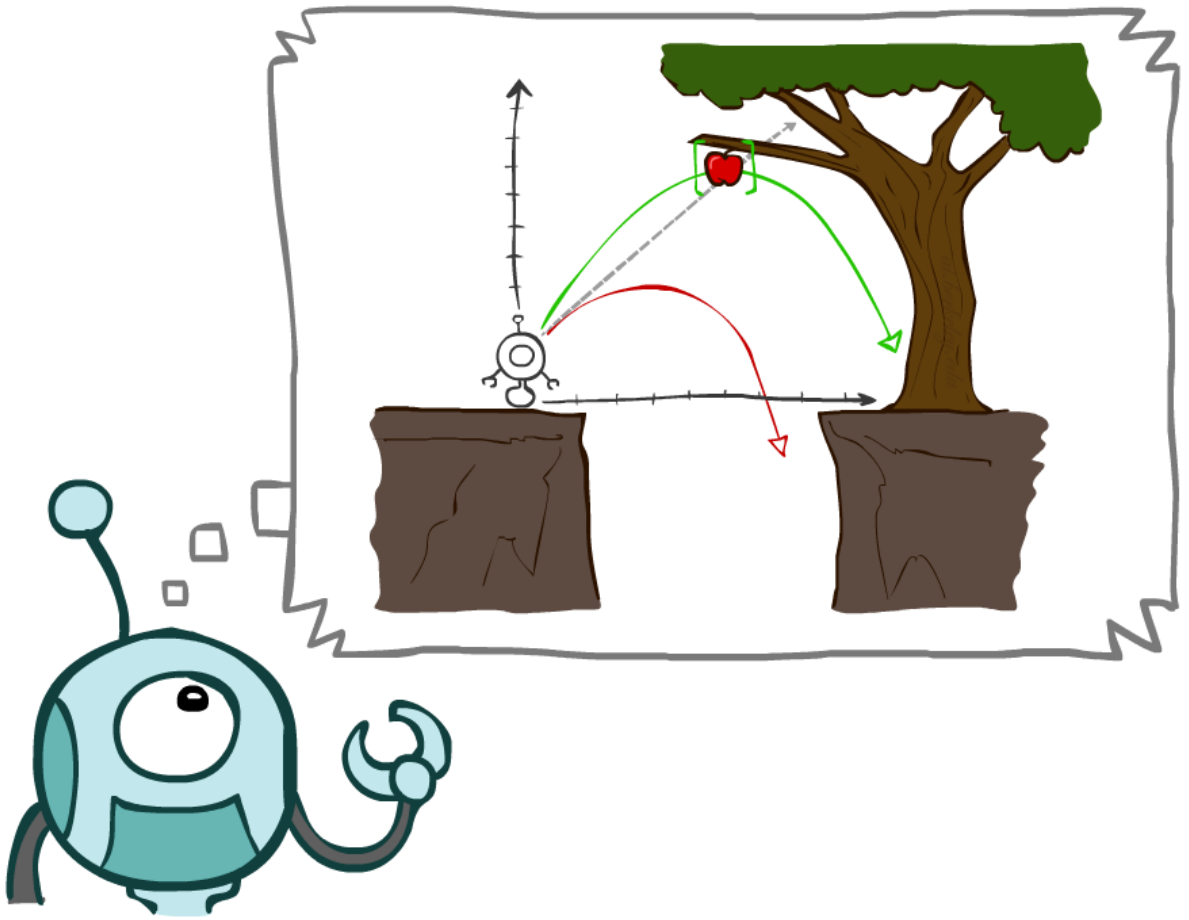
Adapted Slides from: CS 188: Introduction to Artificial Intelligence

Today

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

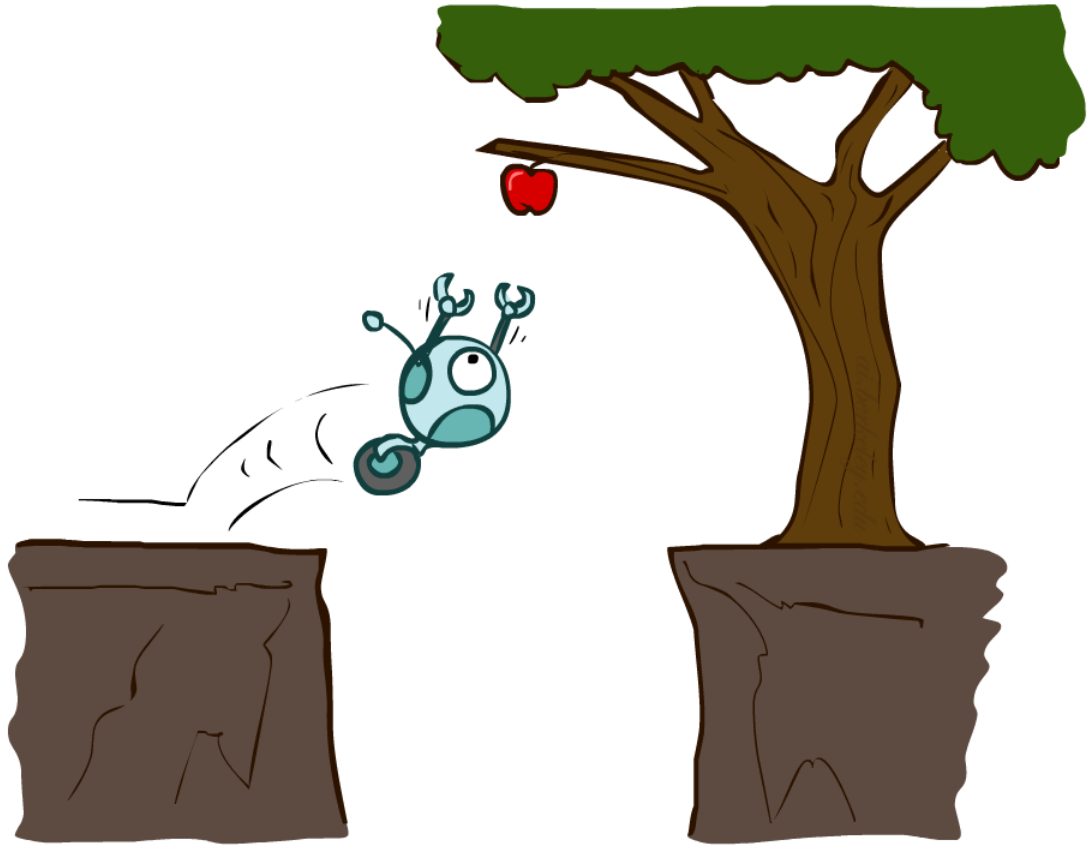


Agents that Plan



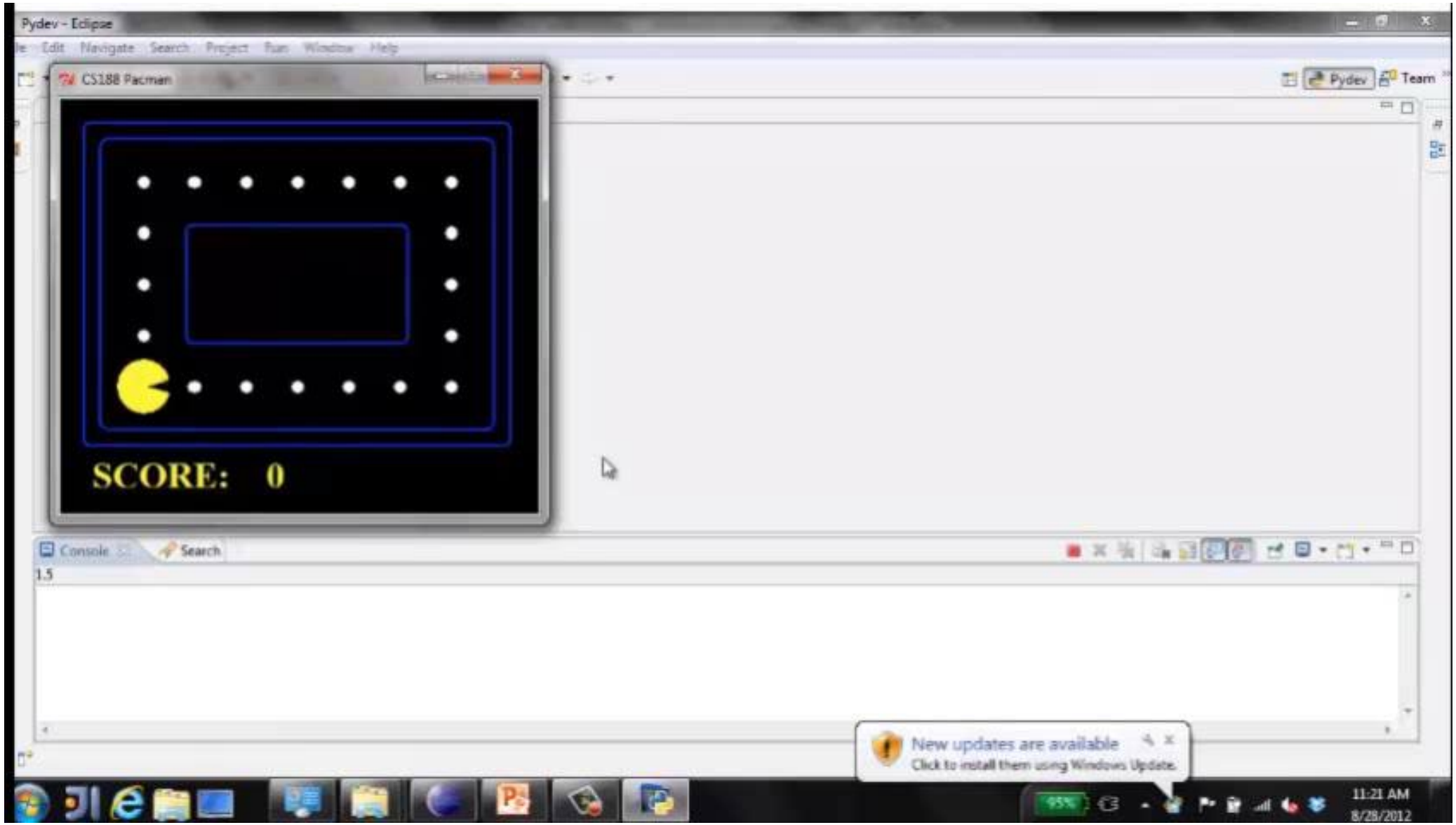
Reflex Agents

- 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?

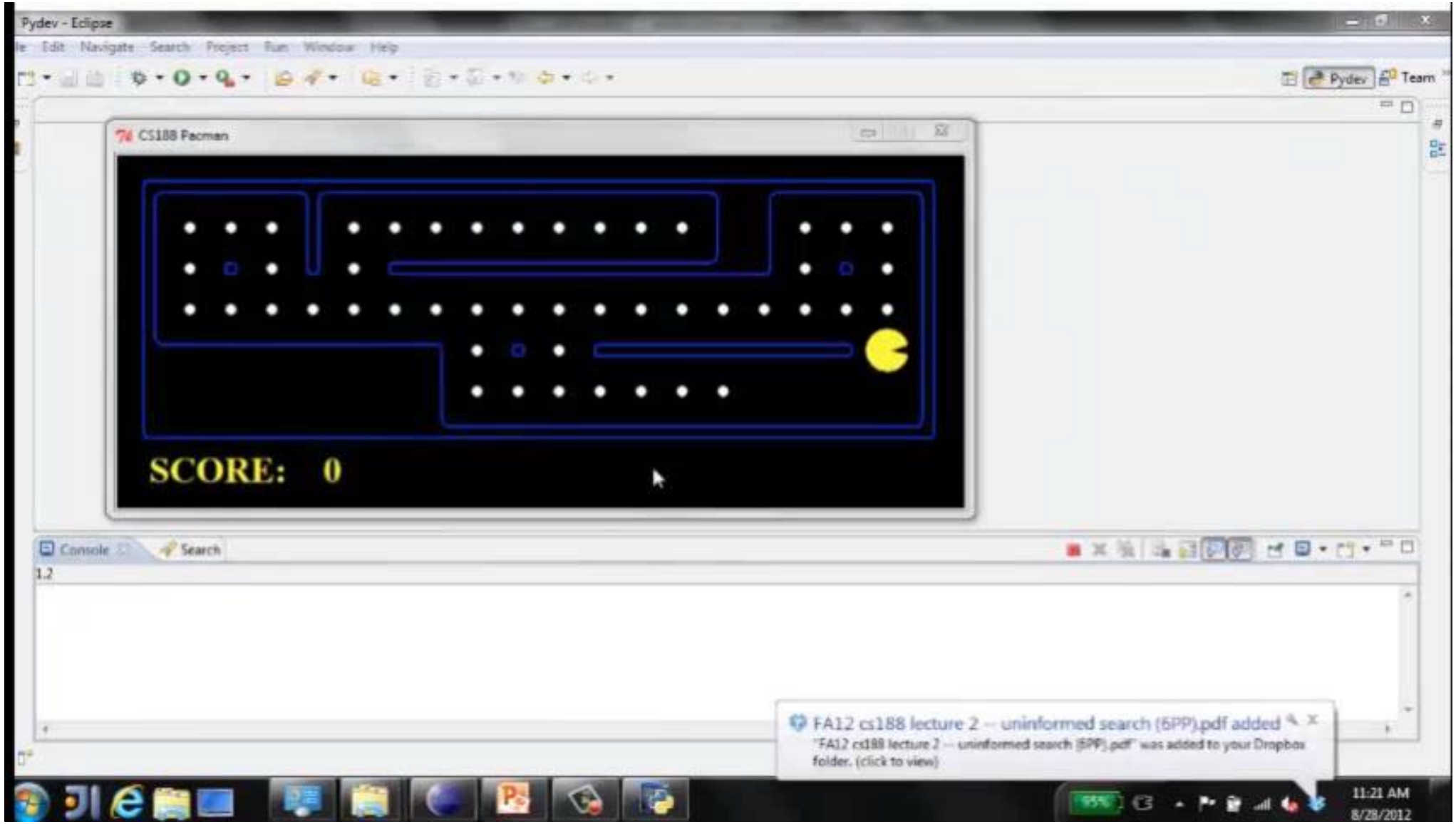


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Video of Demo Reflex Optimal

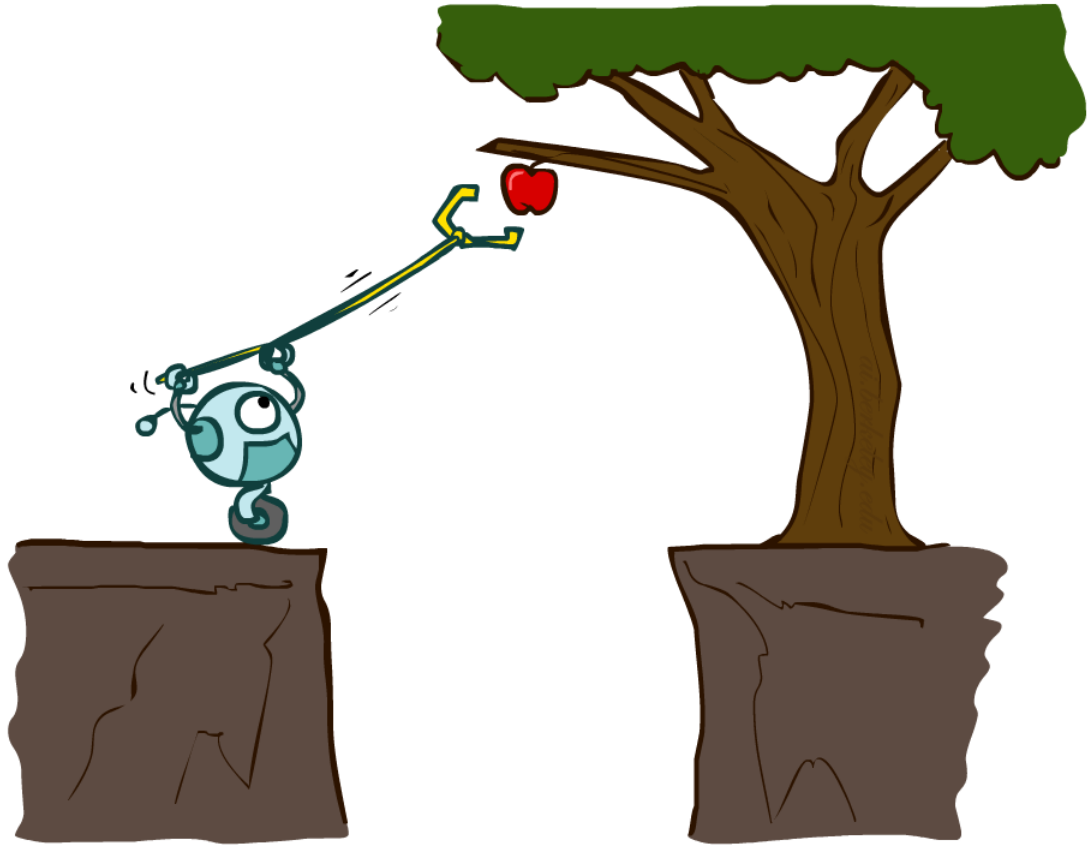


Video of Demo Reflex Odd



Planning Agents

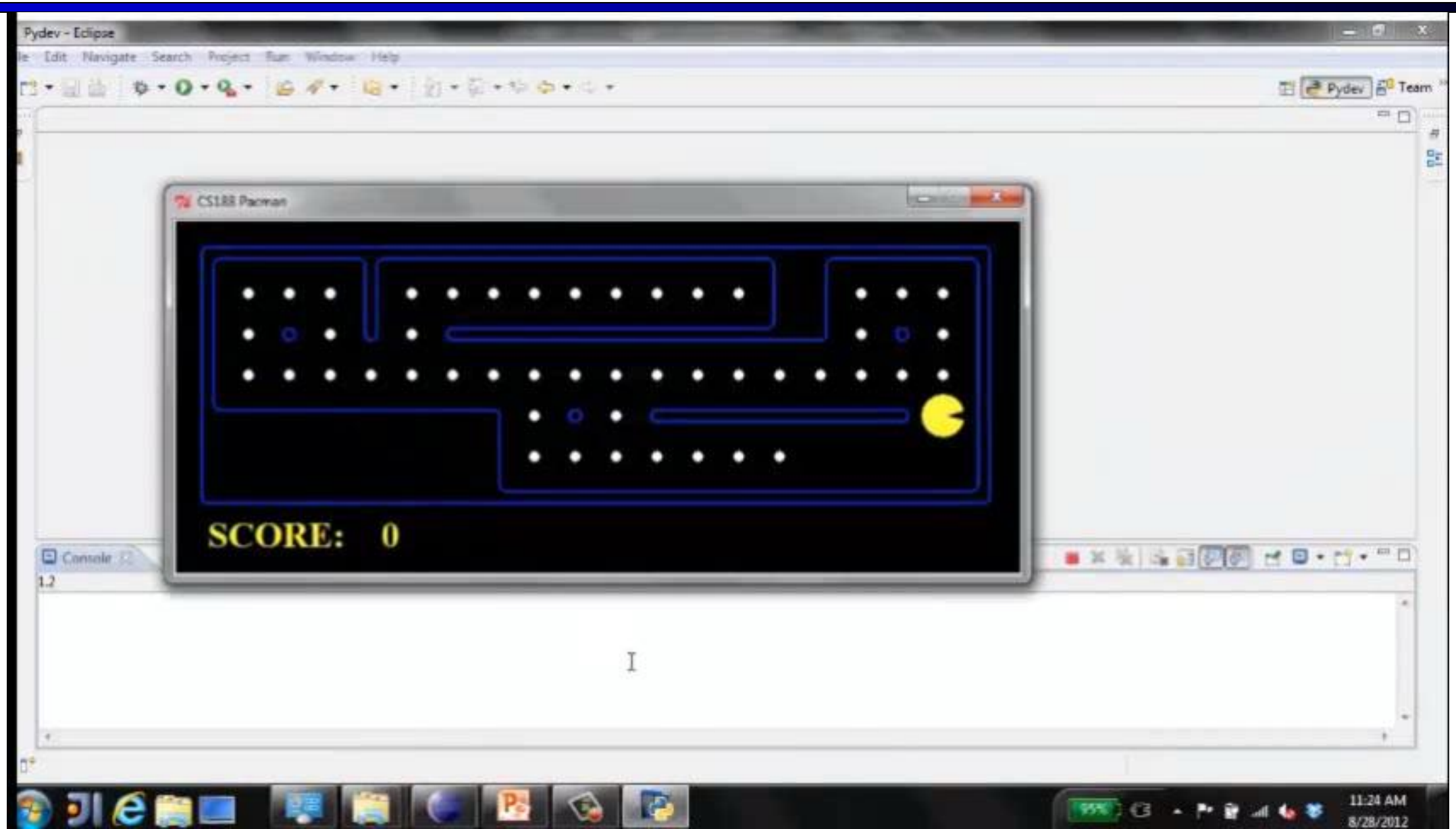
- Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Optimal vs. complete planning
- Planning vs. replanning



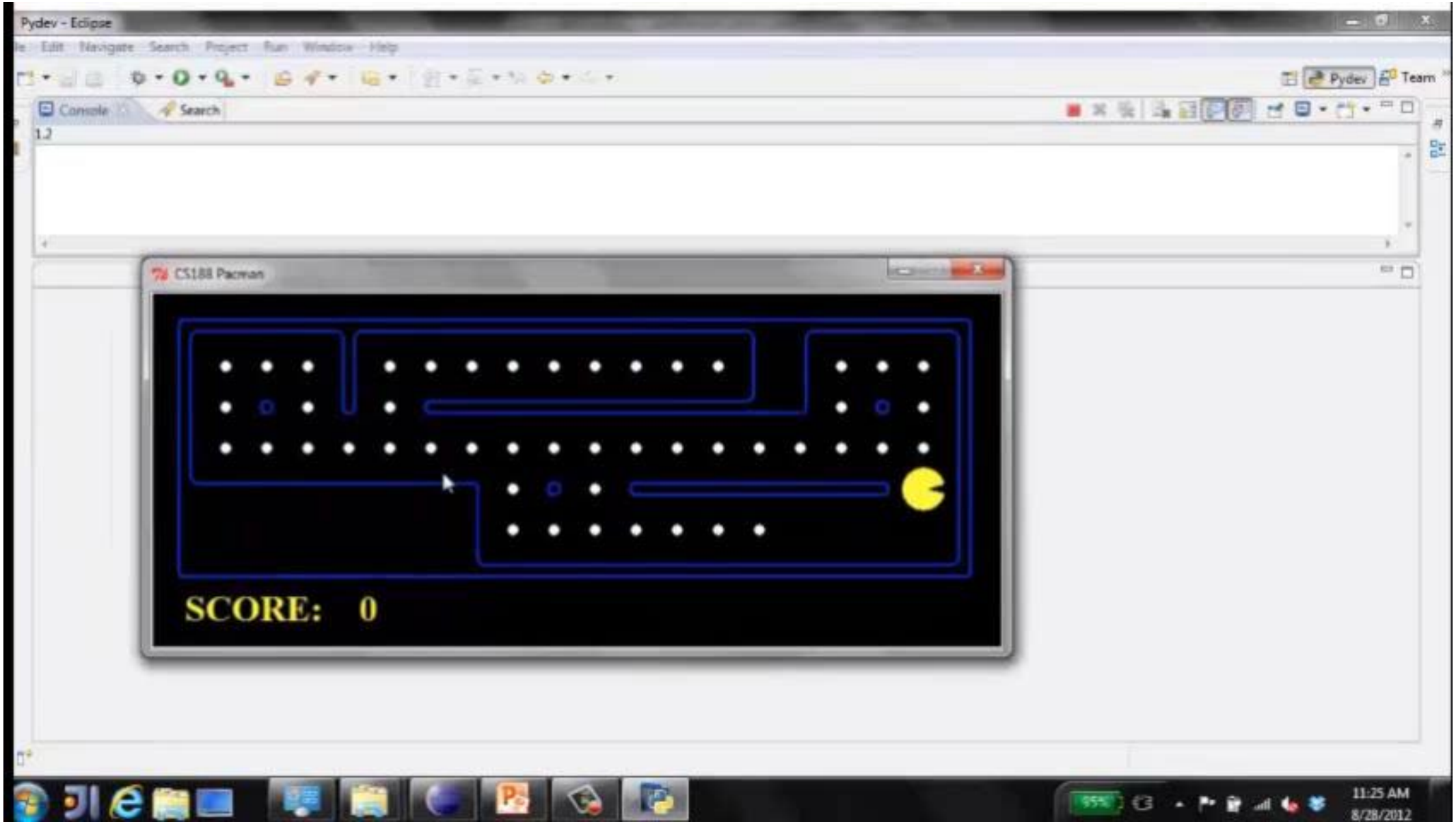
[Demo: replanning (L2D3)]

[Demo: mastermind (L2D4)]

Video of Demo Replanning



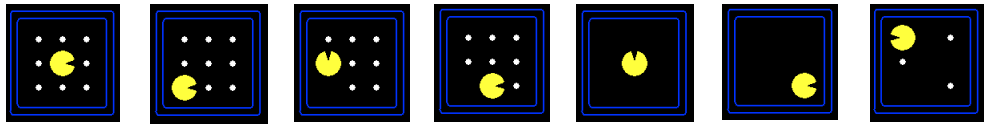
Video of Demo Mastermind



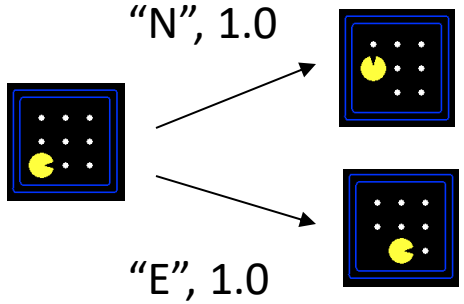
Search Problems

Search Problems

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)

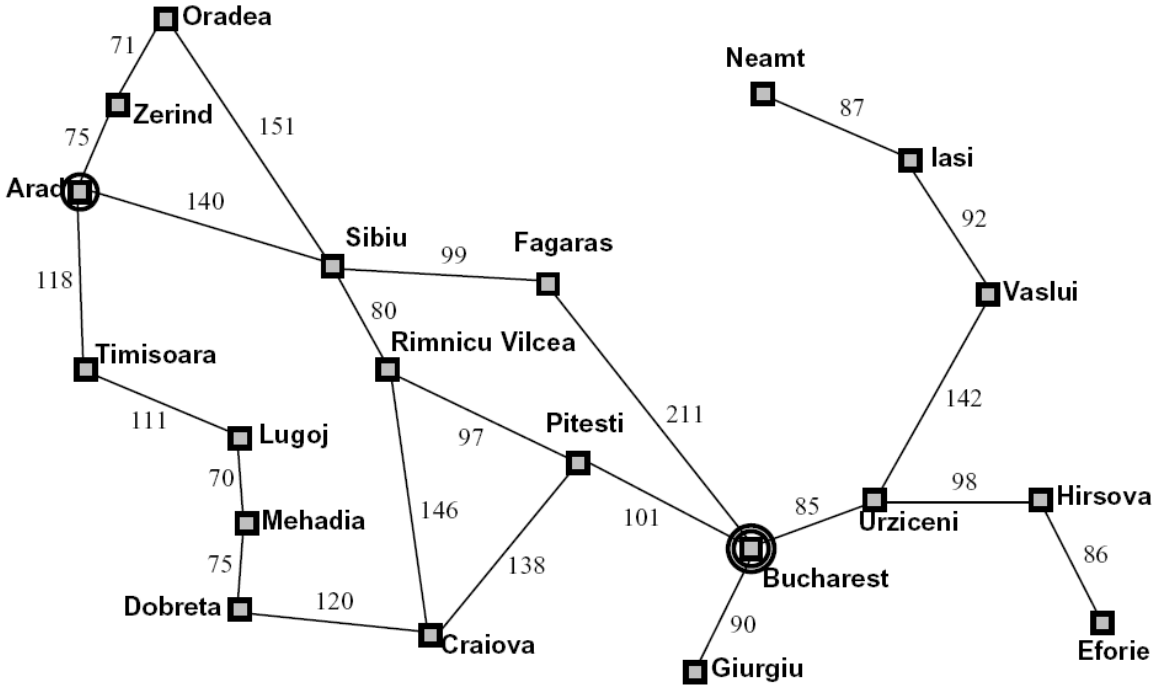


  - A start state and a goal test

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

Search Problems Are Models

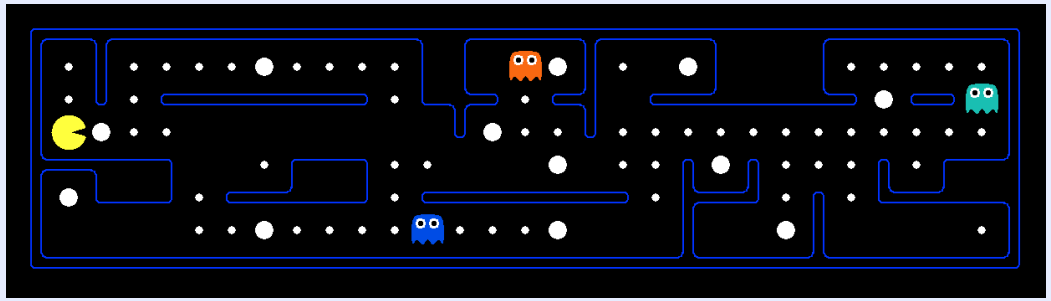
Example: Traveling in Romania



- 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?

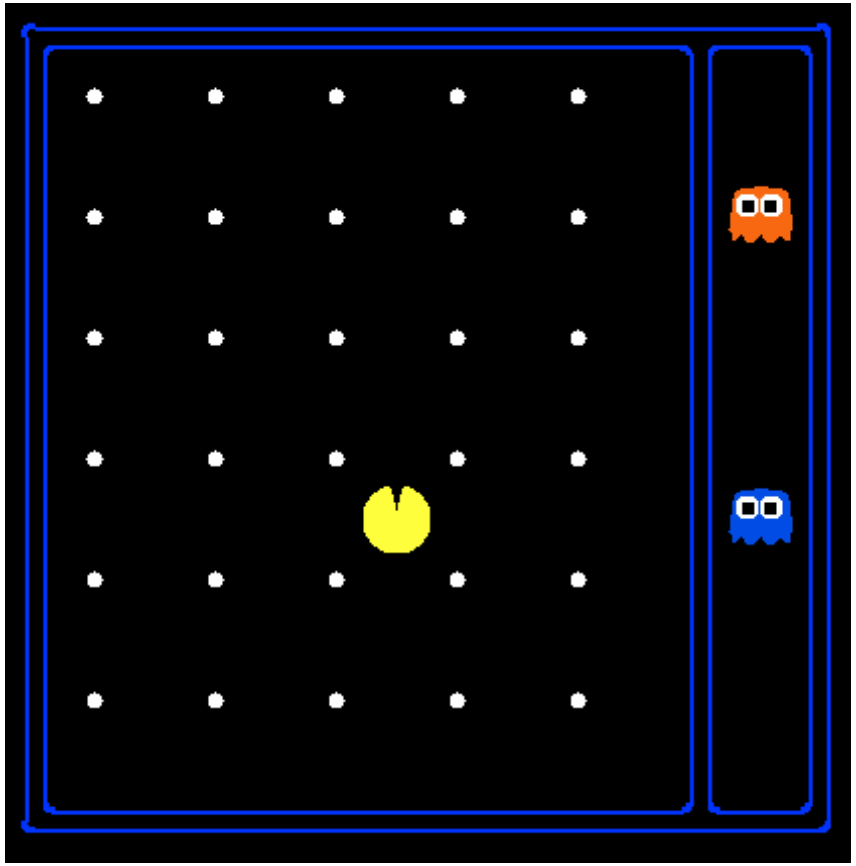
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

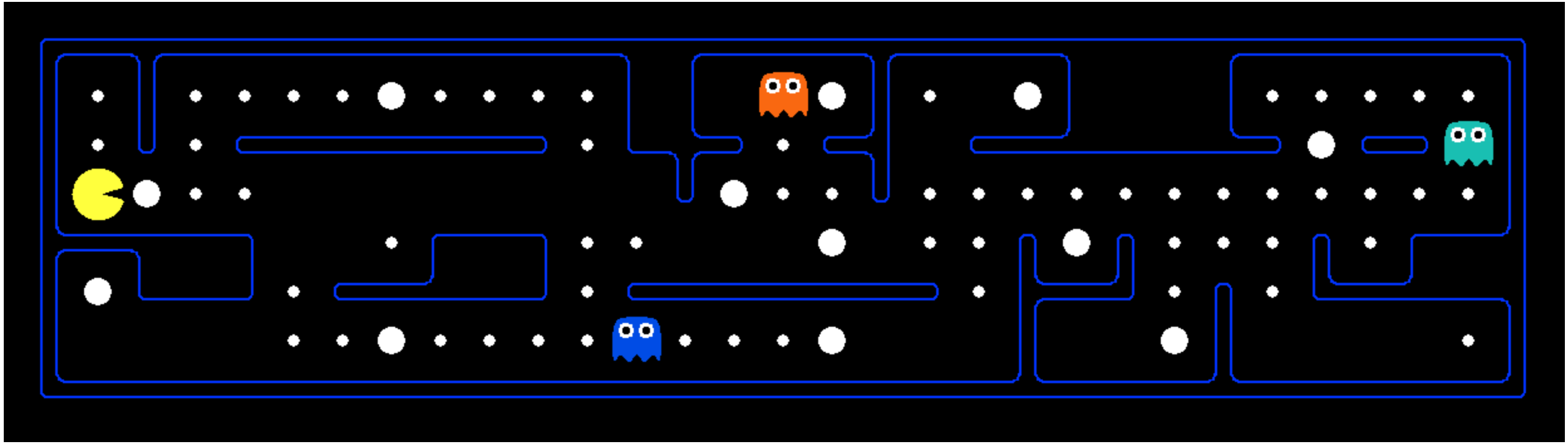
    $120x(2^{30})x(12^2)x4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120x(2^{30})$

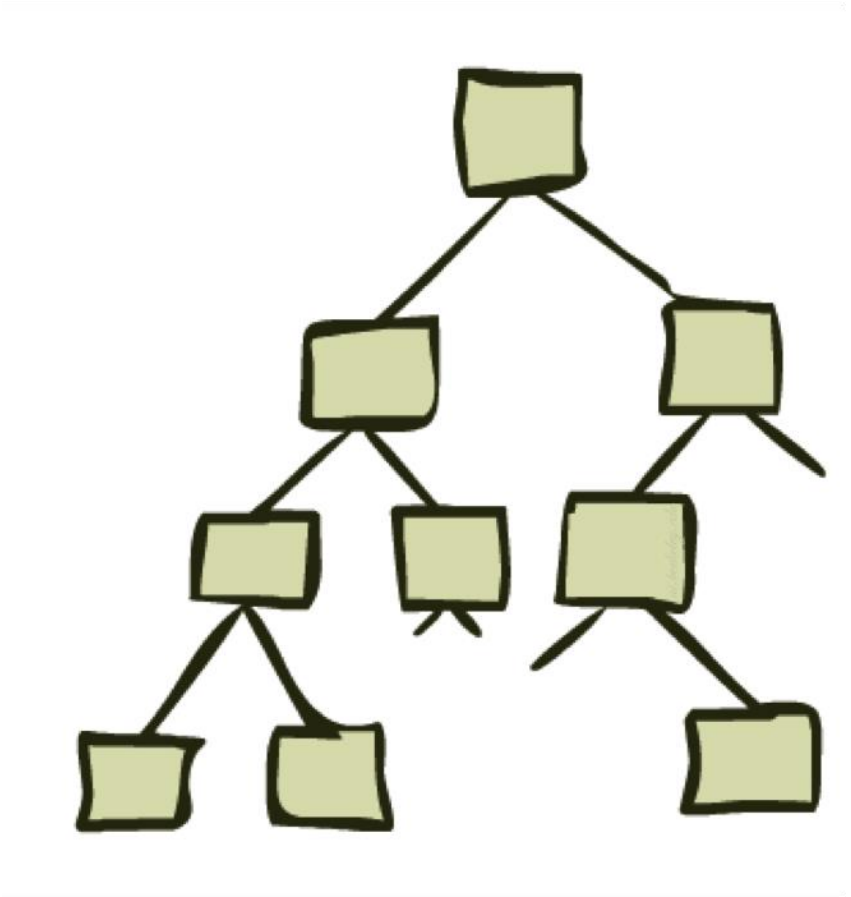


Quiz: Safe Passage



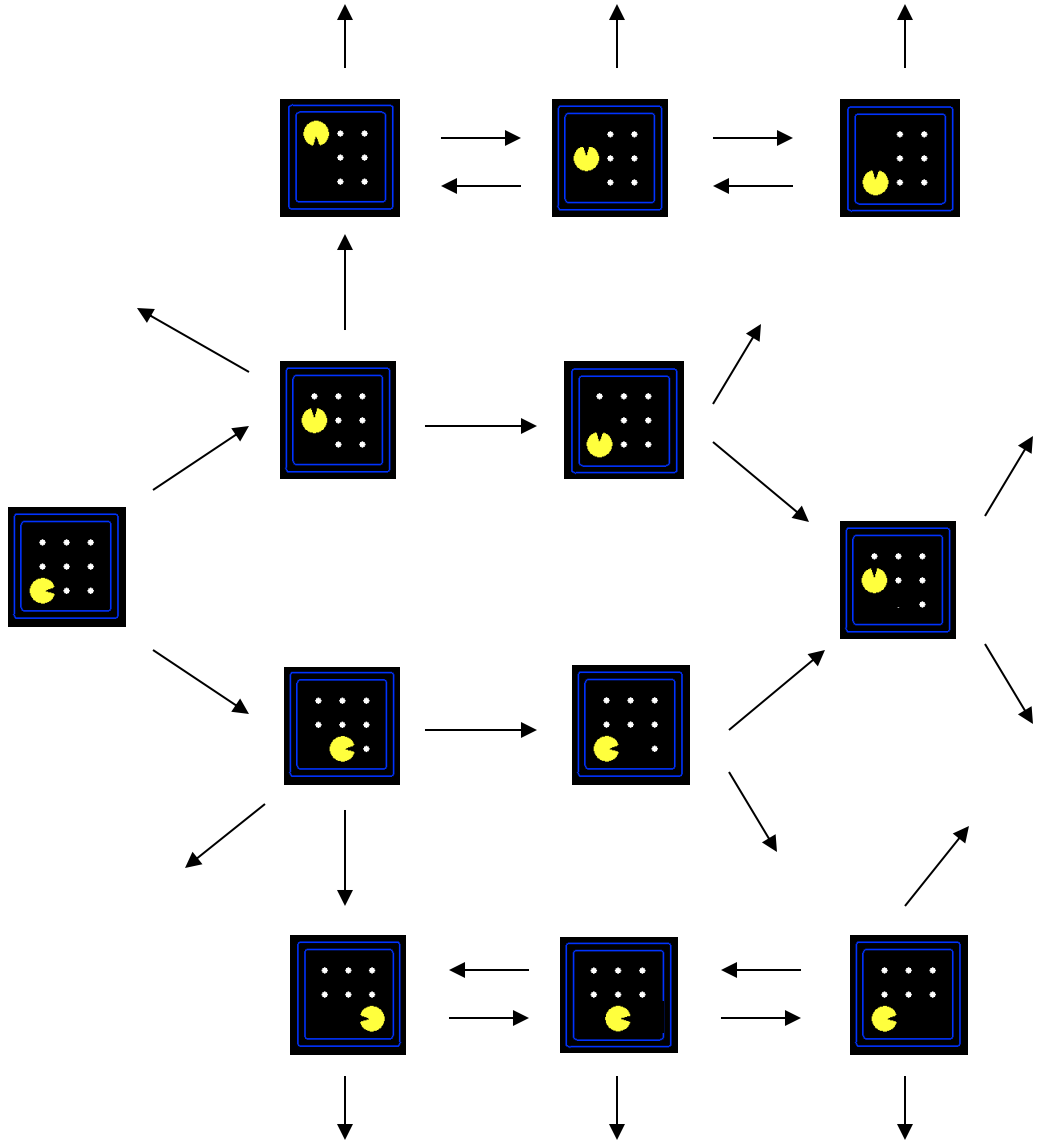
- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

State Space Graphs and Search Trees



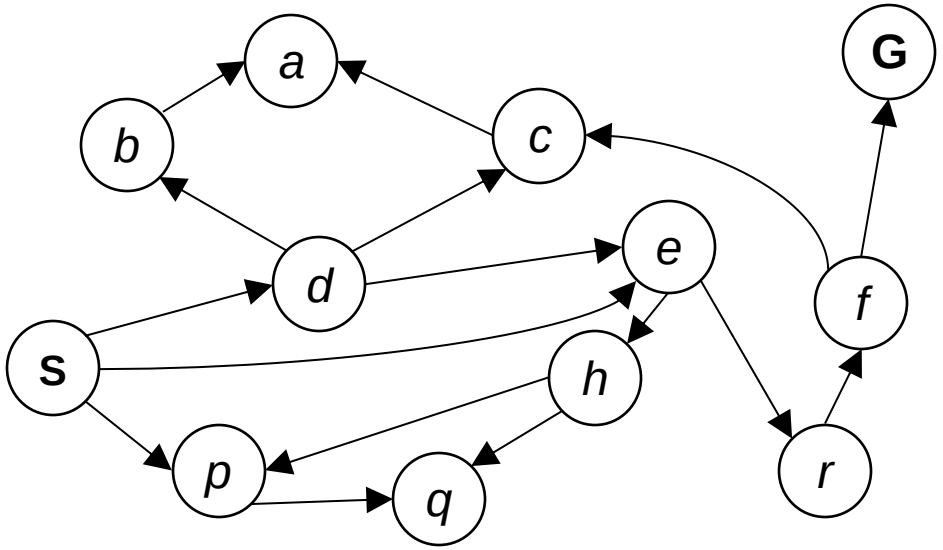
State Space Graphs

- 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



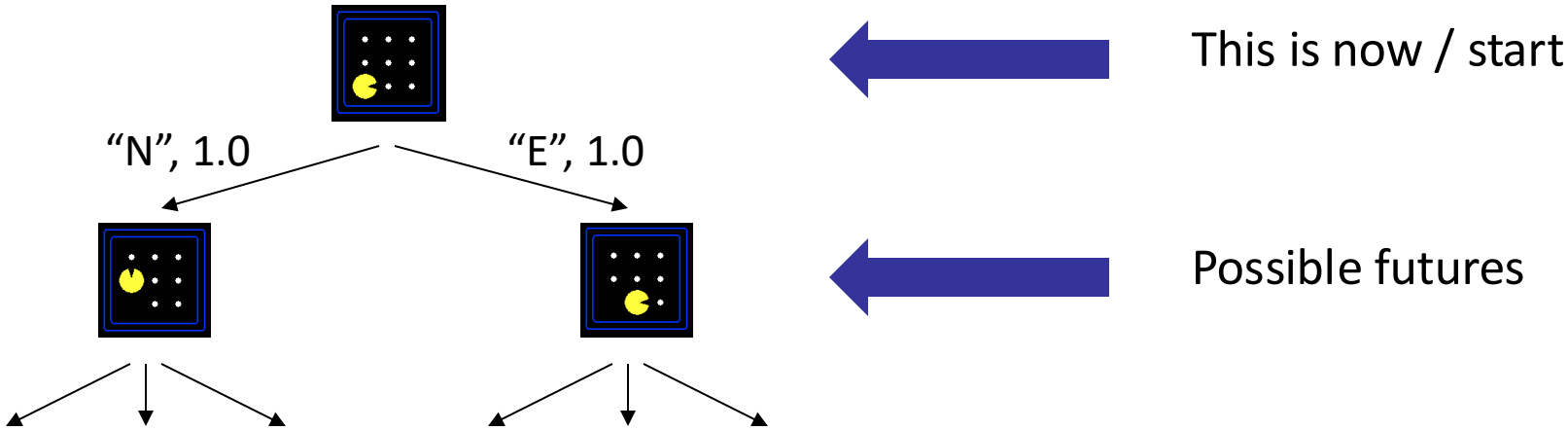
State Space Graphs

- 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

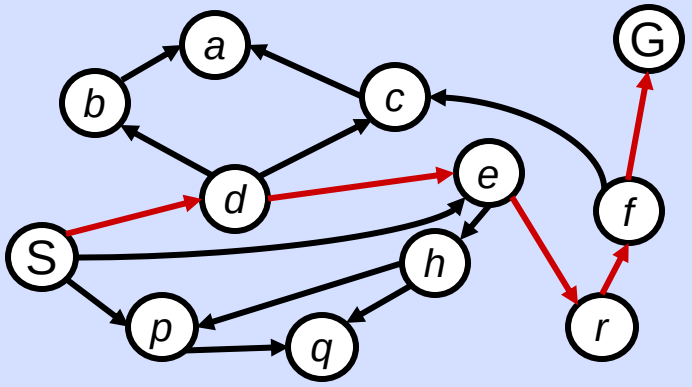
Search Trees



- 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

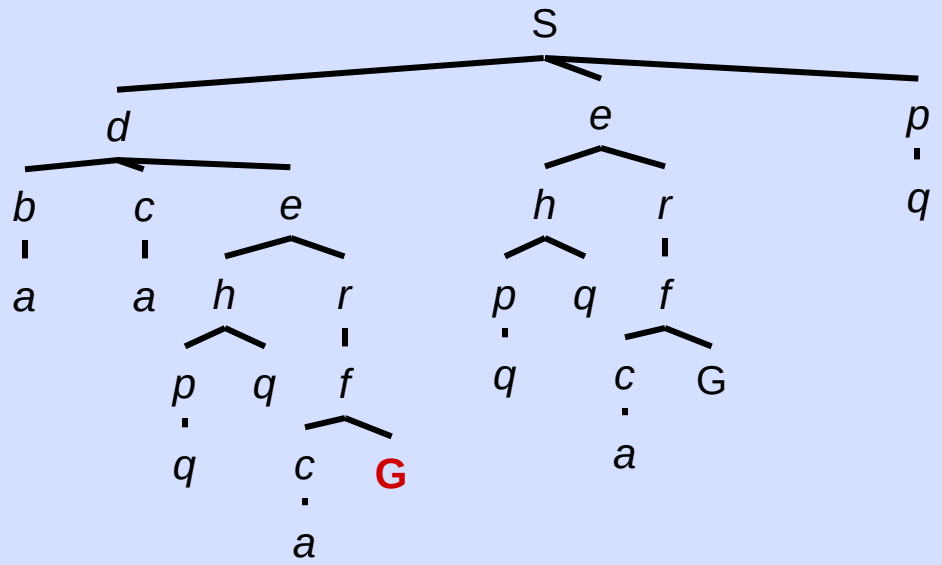
State Space Graph



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

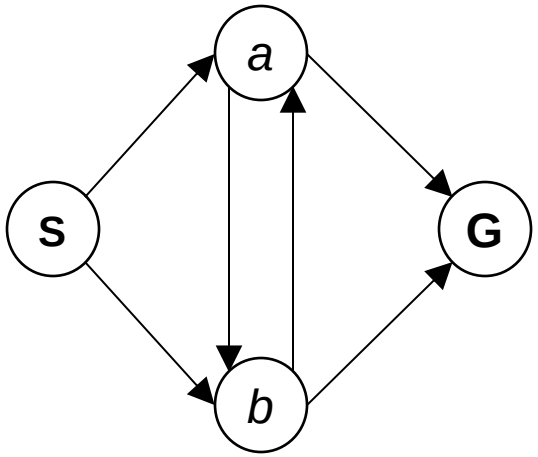
We construct both on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

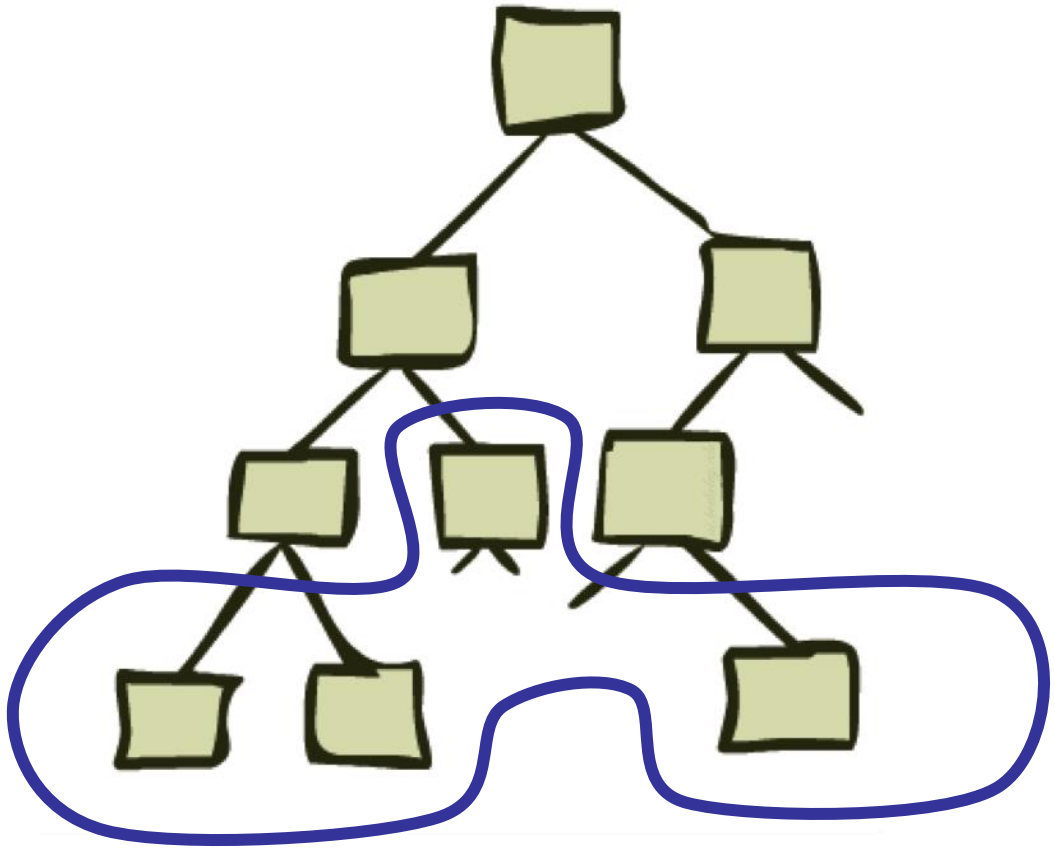
Consider this 4-state graph:



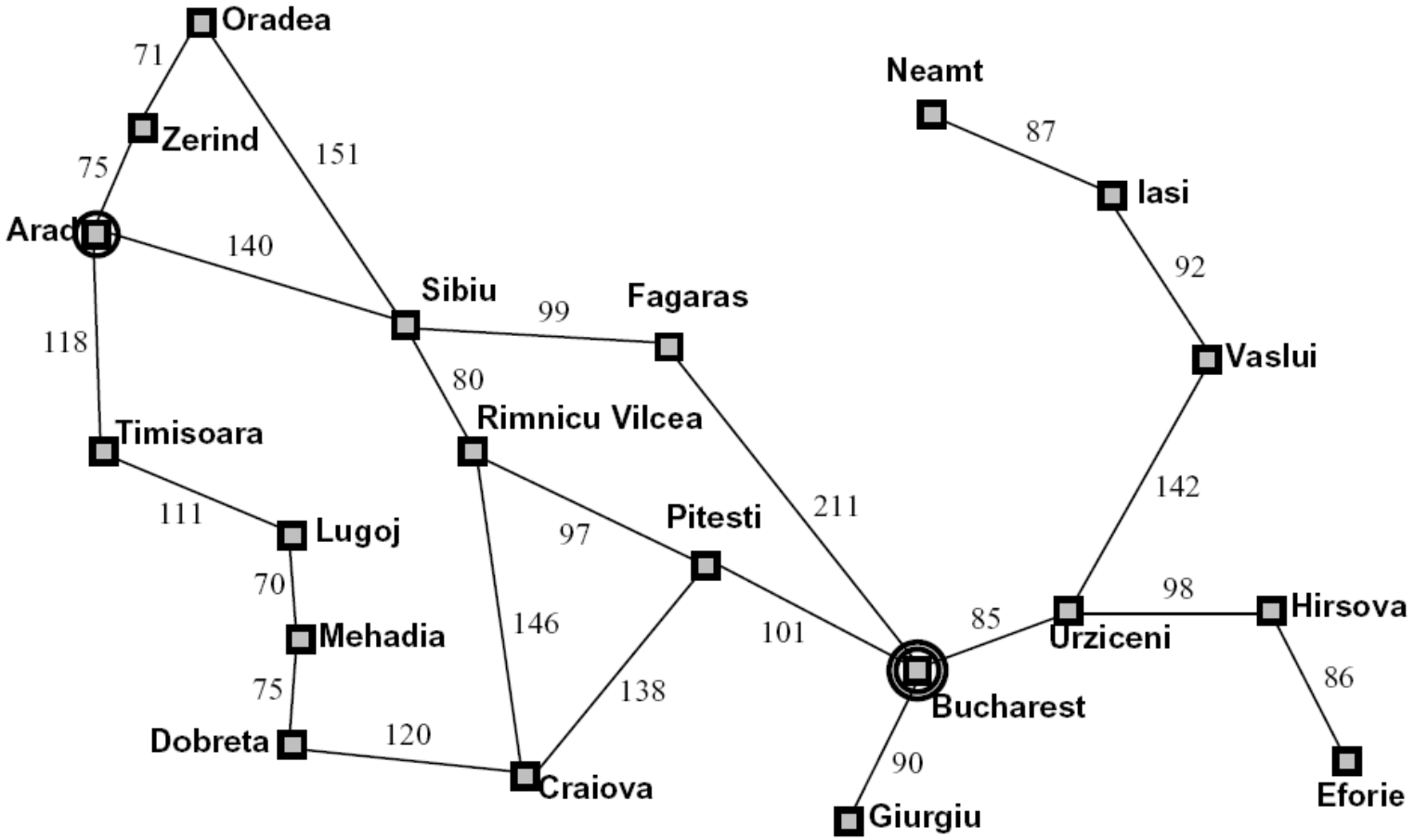
How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

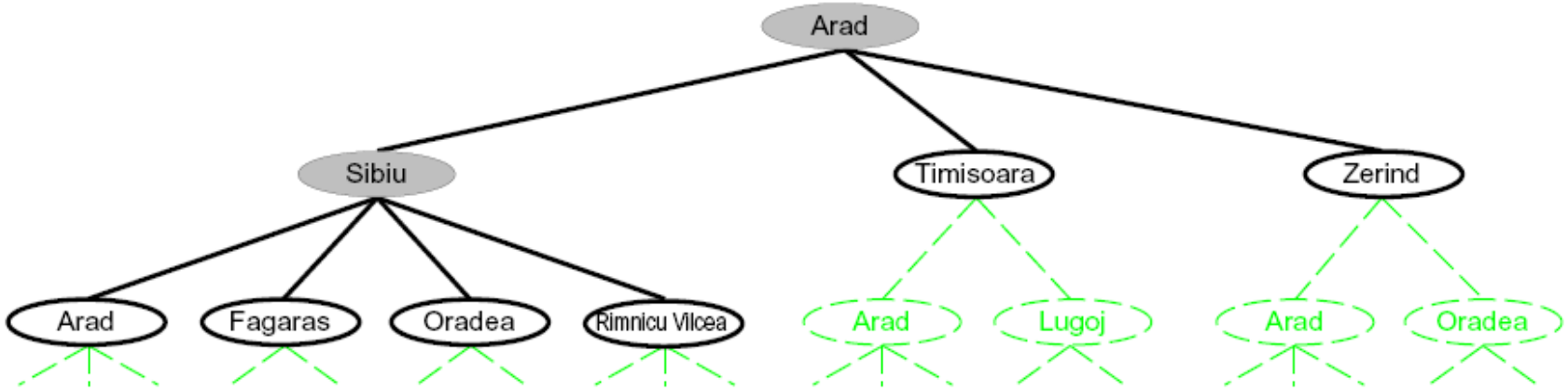
Tree Search



Search Example: Romania



Searching with a Search Tree



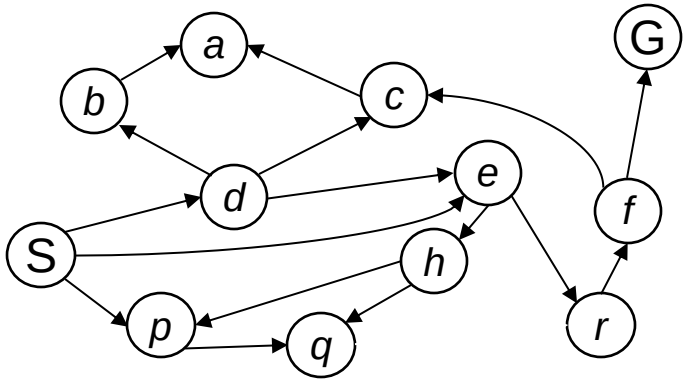
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

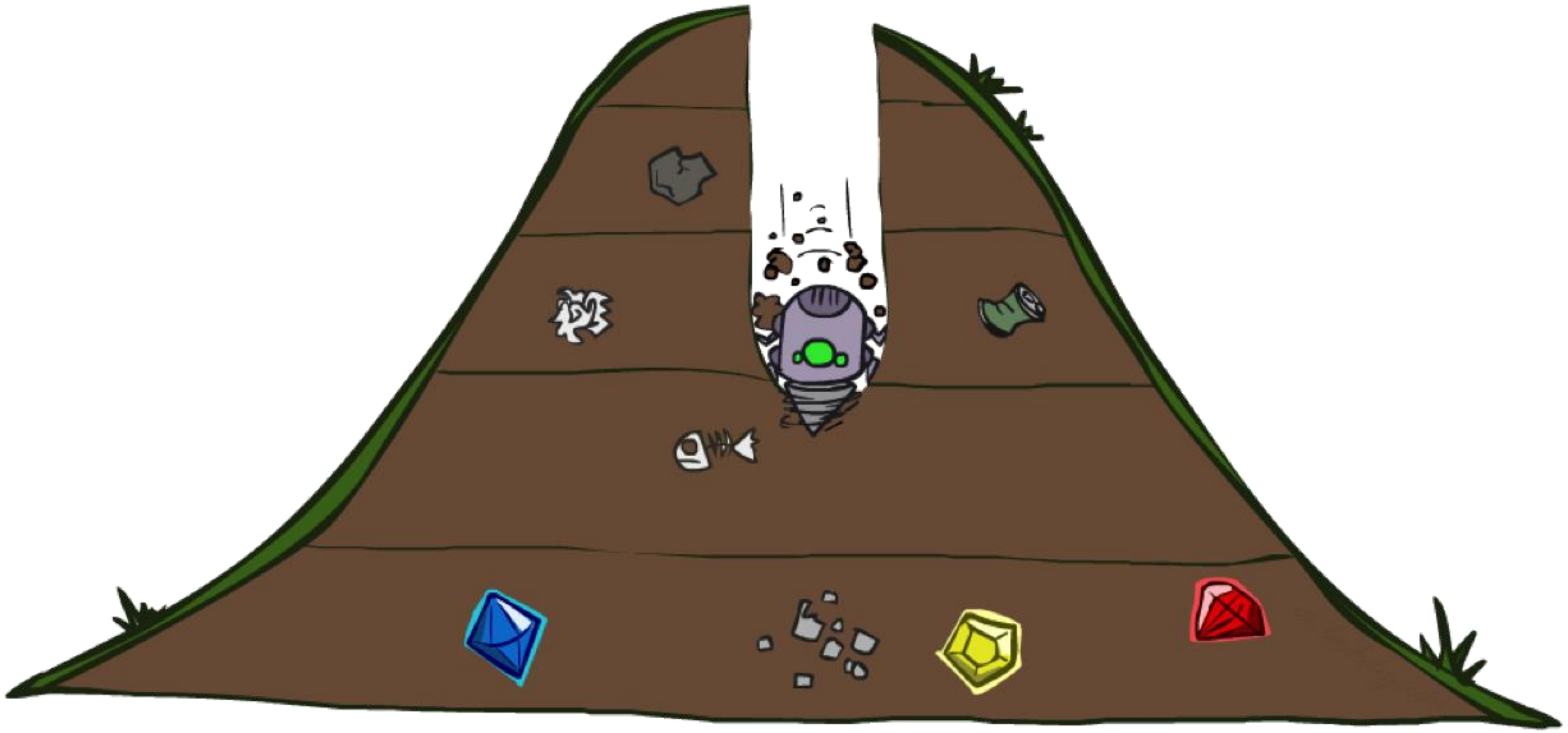
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the 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



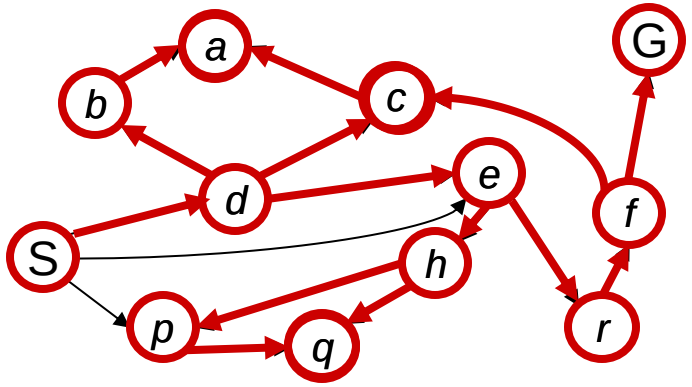
Depth-First Search

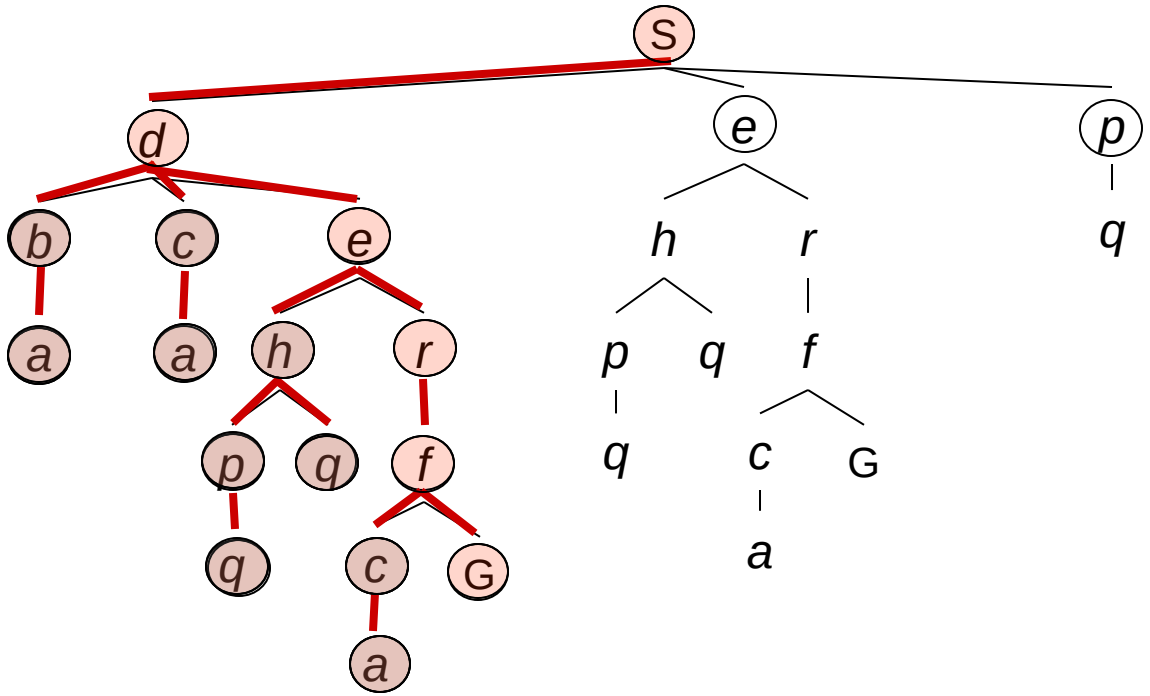


Depth-First Search

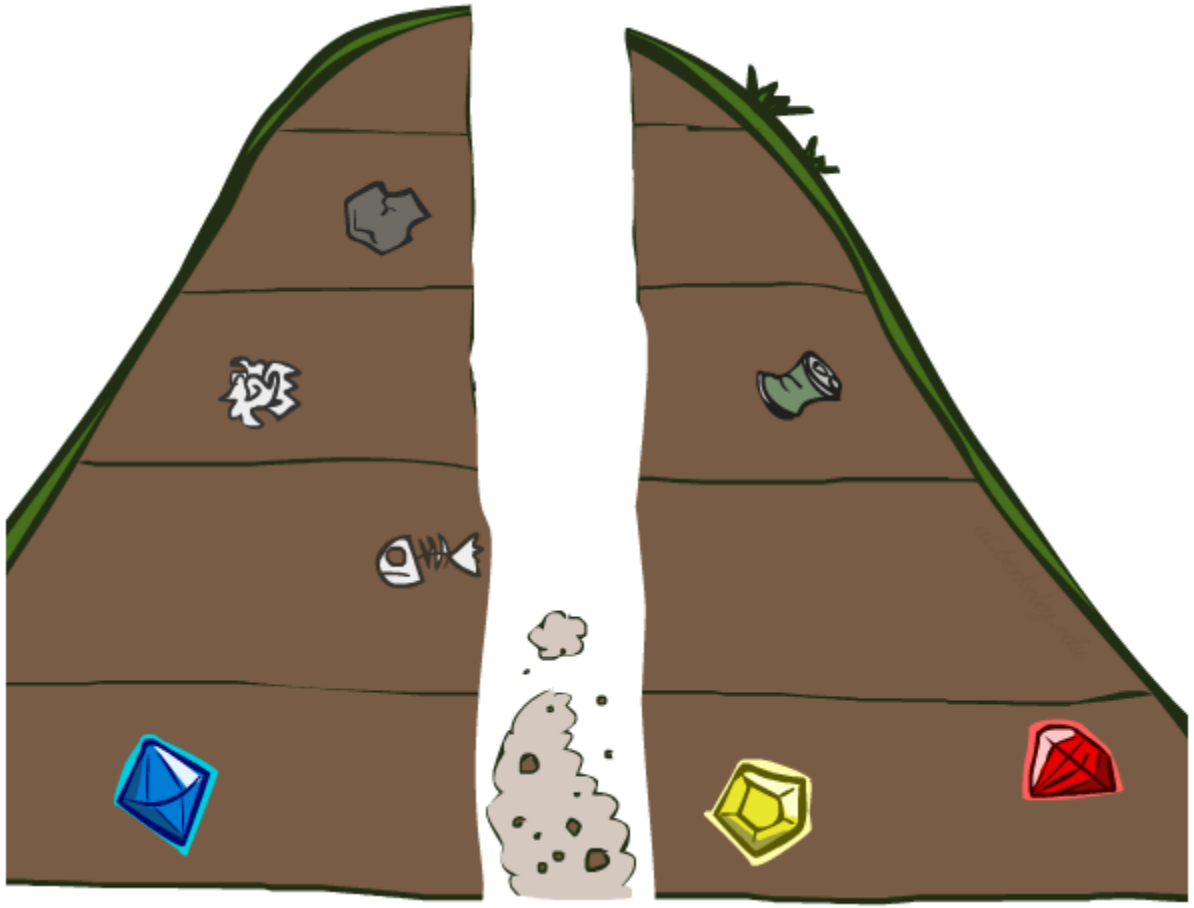
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





Search Algorithm Properties

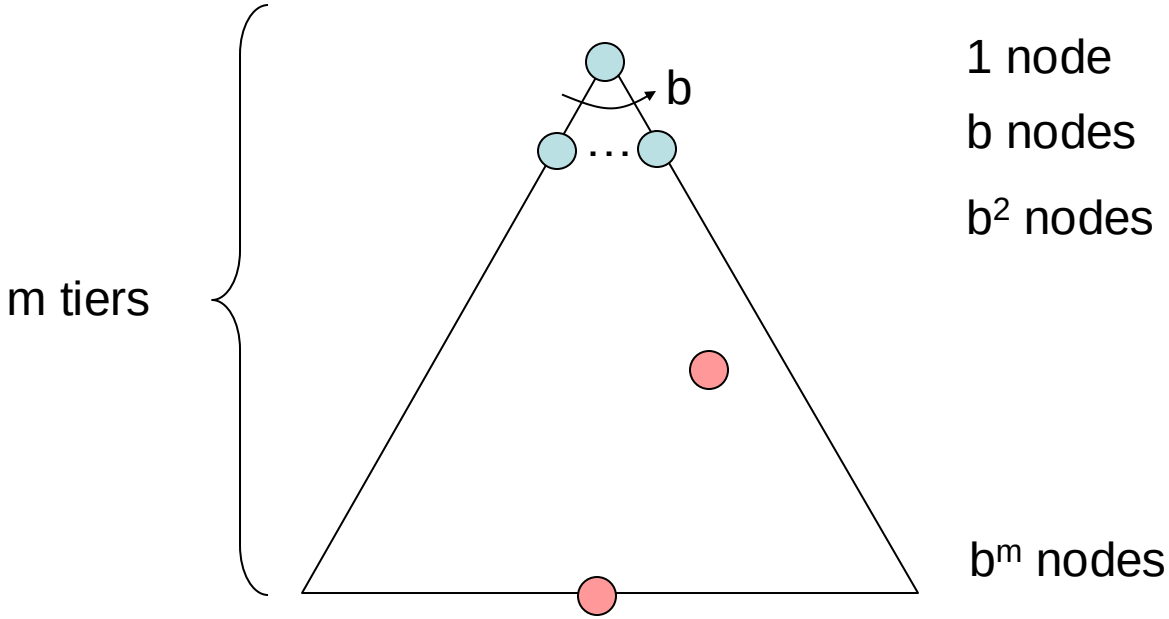


Search Algorithm Properties

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

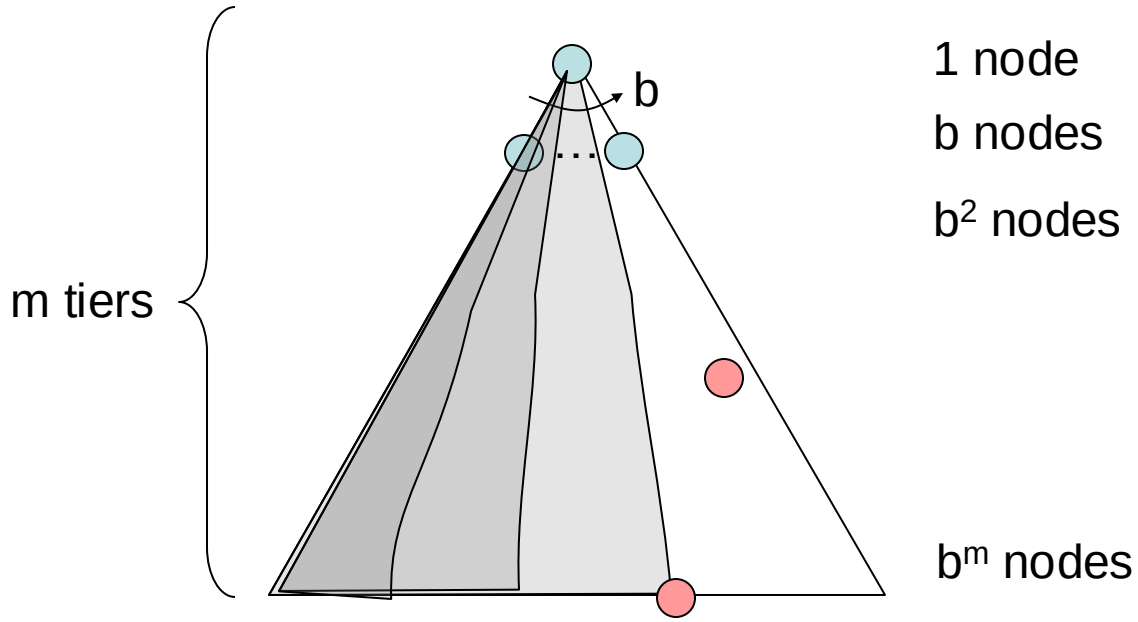
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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

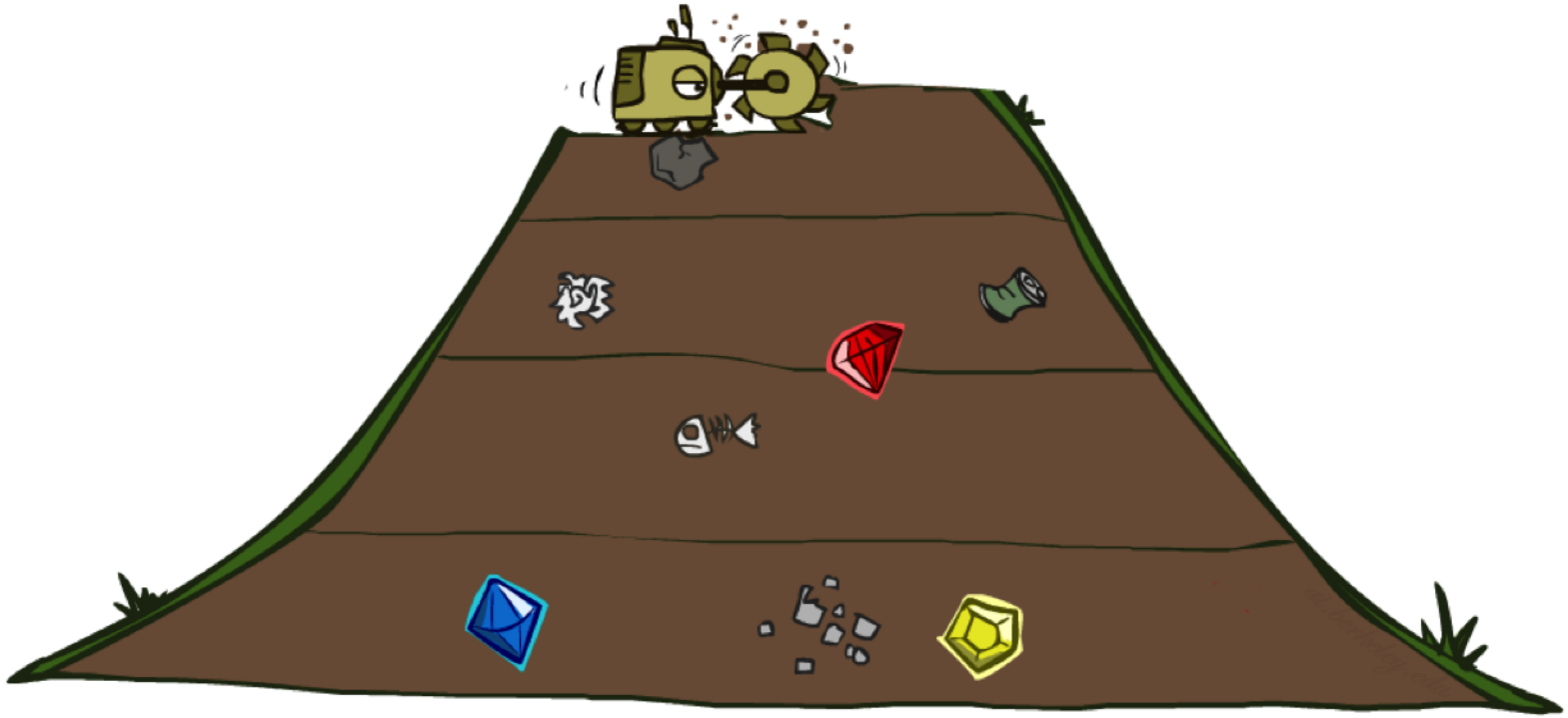


Depth-First Search (DFS) Properties

- 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



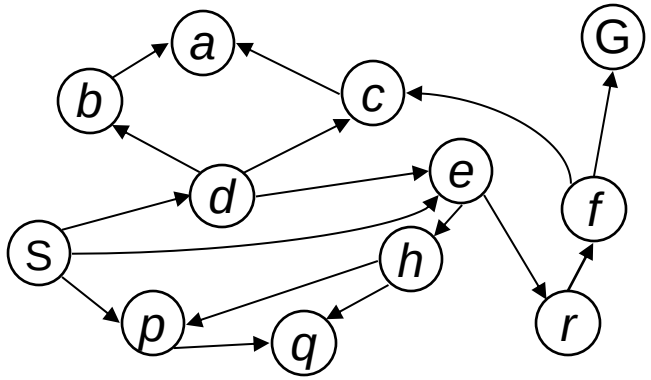
Breadth-First Search

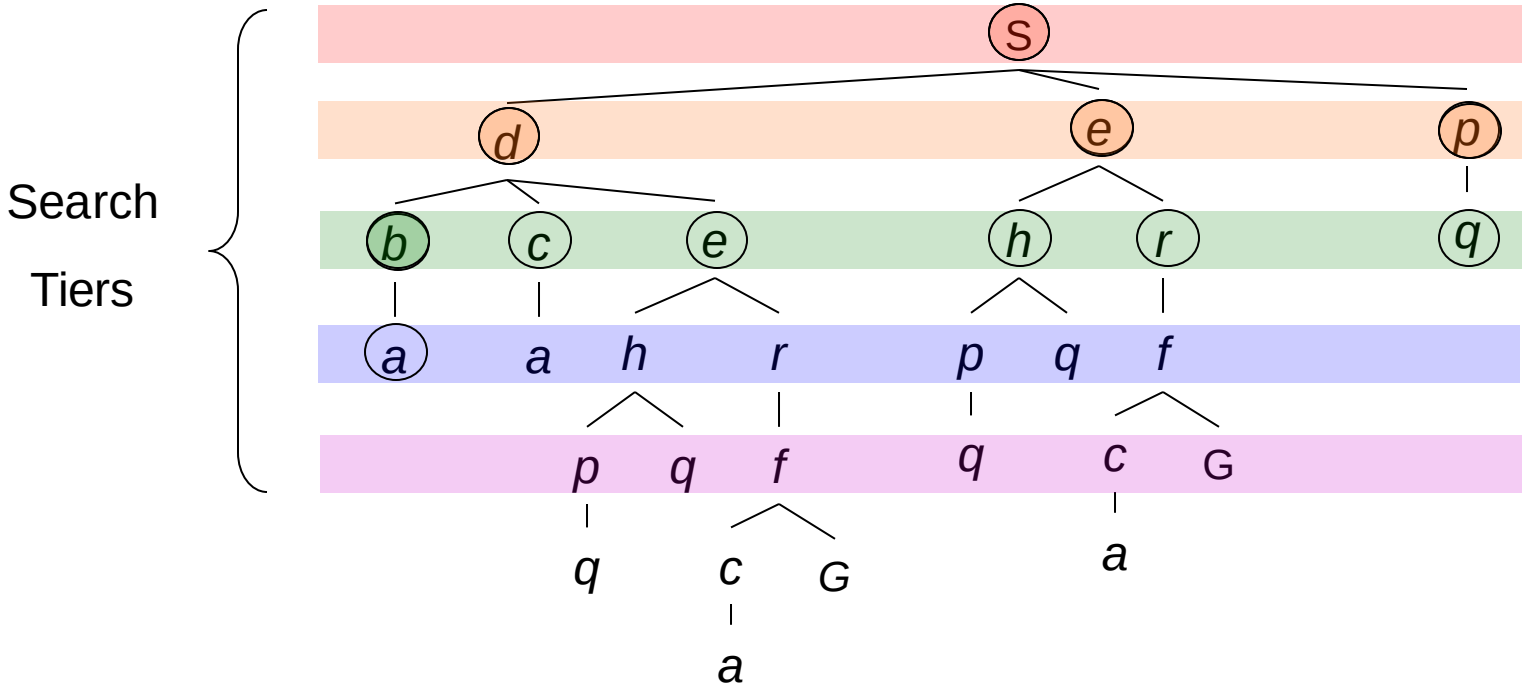


Breadth-First Search

Strategy: expand a shallowest node first

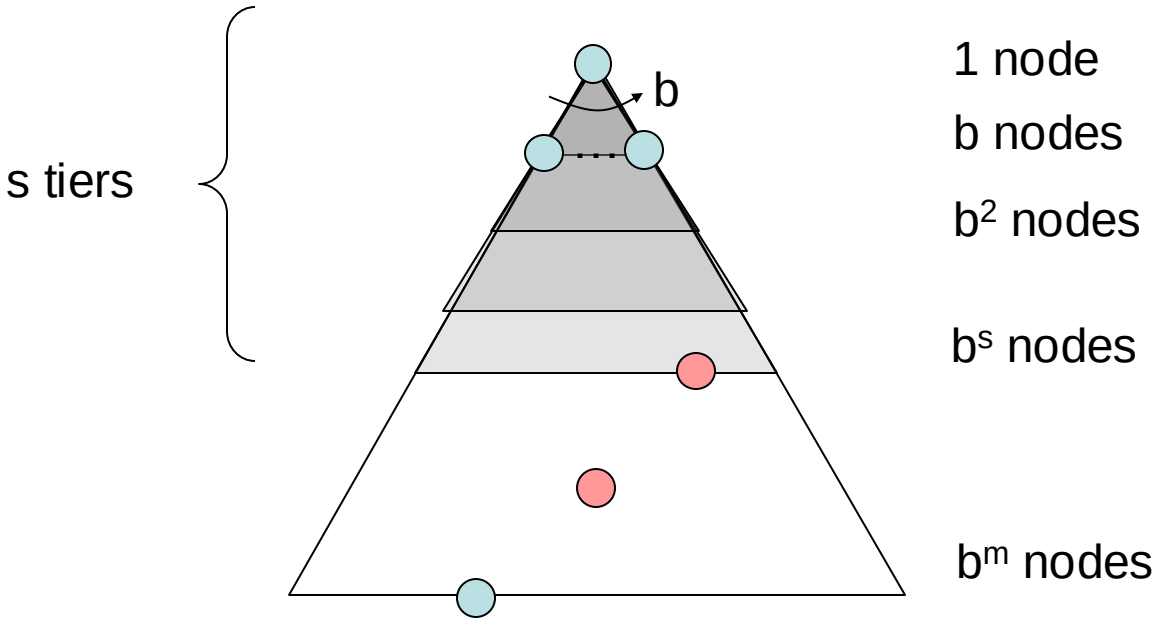
Implementation: Fringe is a FIFO queue



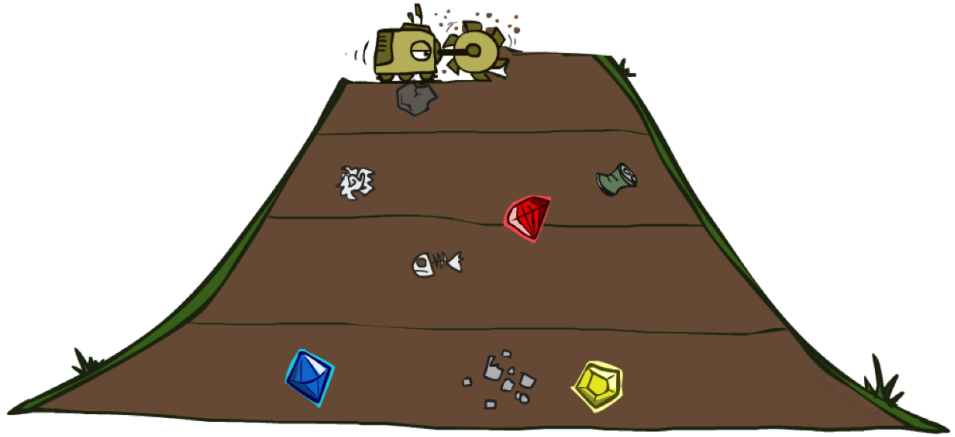


Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Quiz: DFS vs BFS



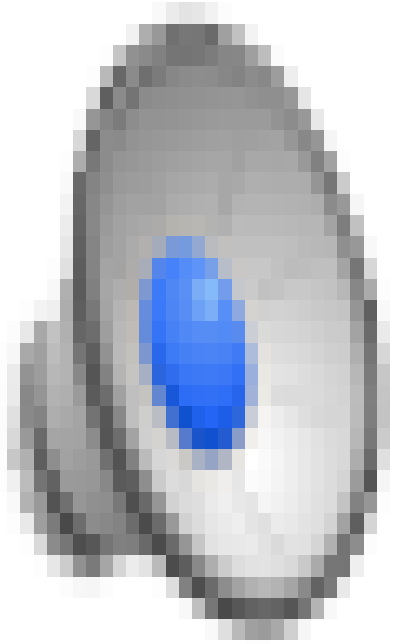
Quiz: DFS vs BFS

- When will BFS outperform DFS?

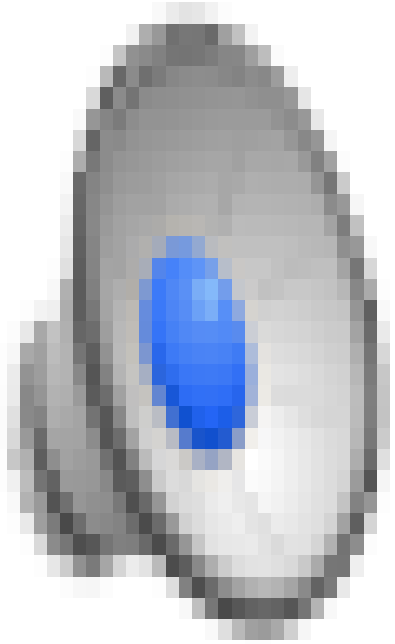
- When will DFS outperform BFS?

[Demo: dfs/bfs maze water (L2D6)]

Video of Demo Maze Water DFS/BFS (part 1)

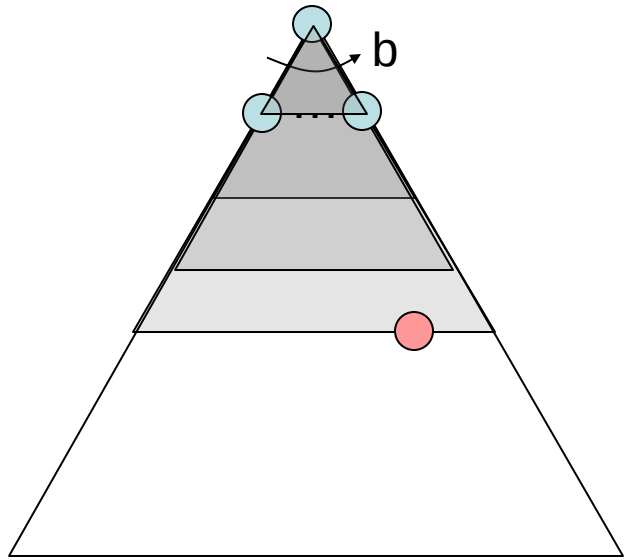


Video of Demo Maze Water DFS/BFS (part 2)

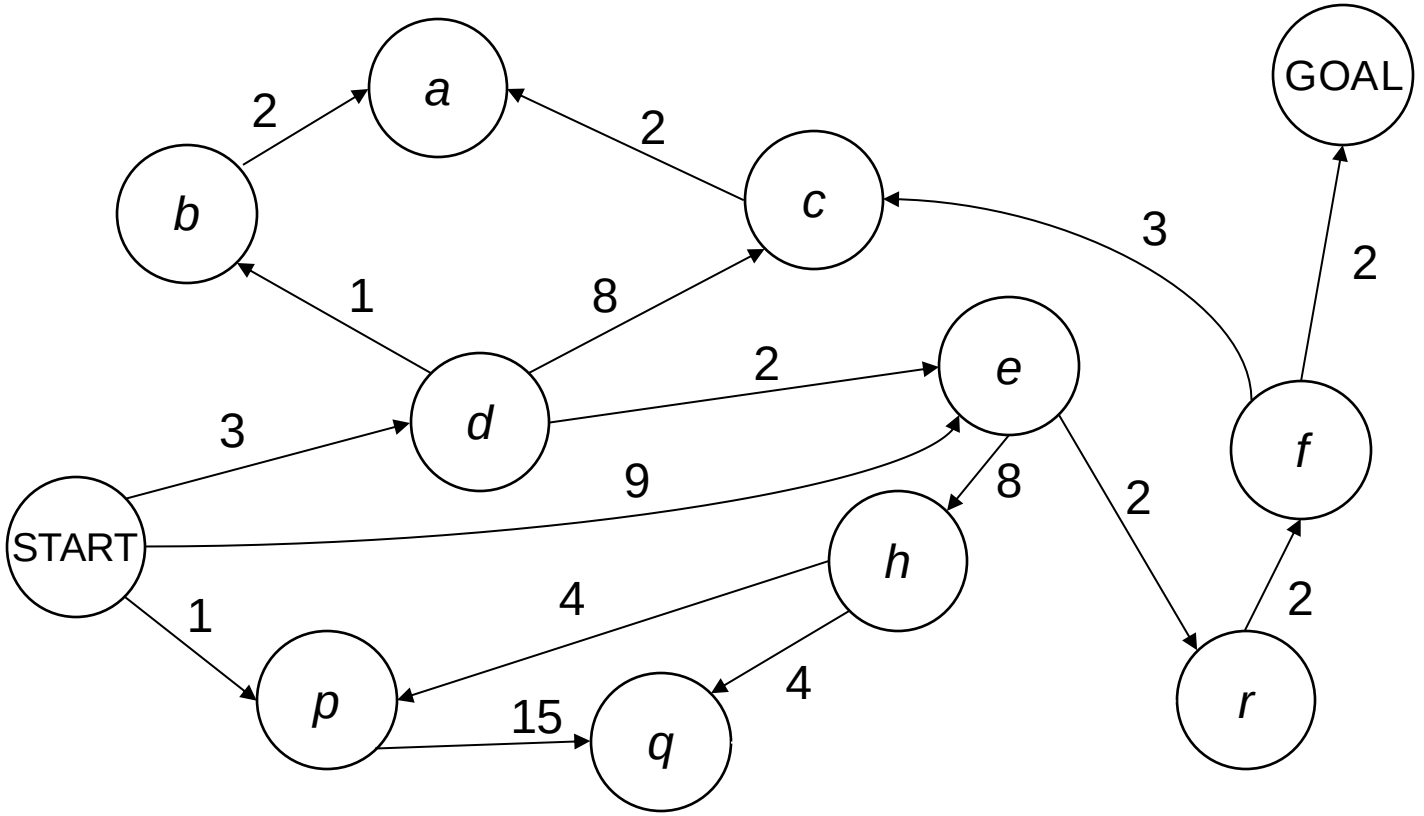


Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

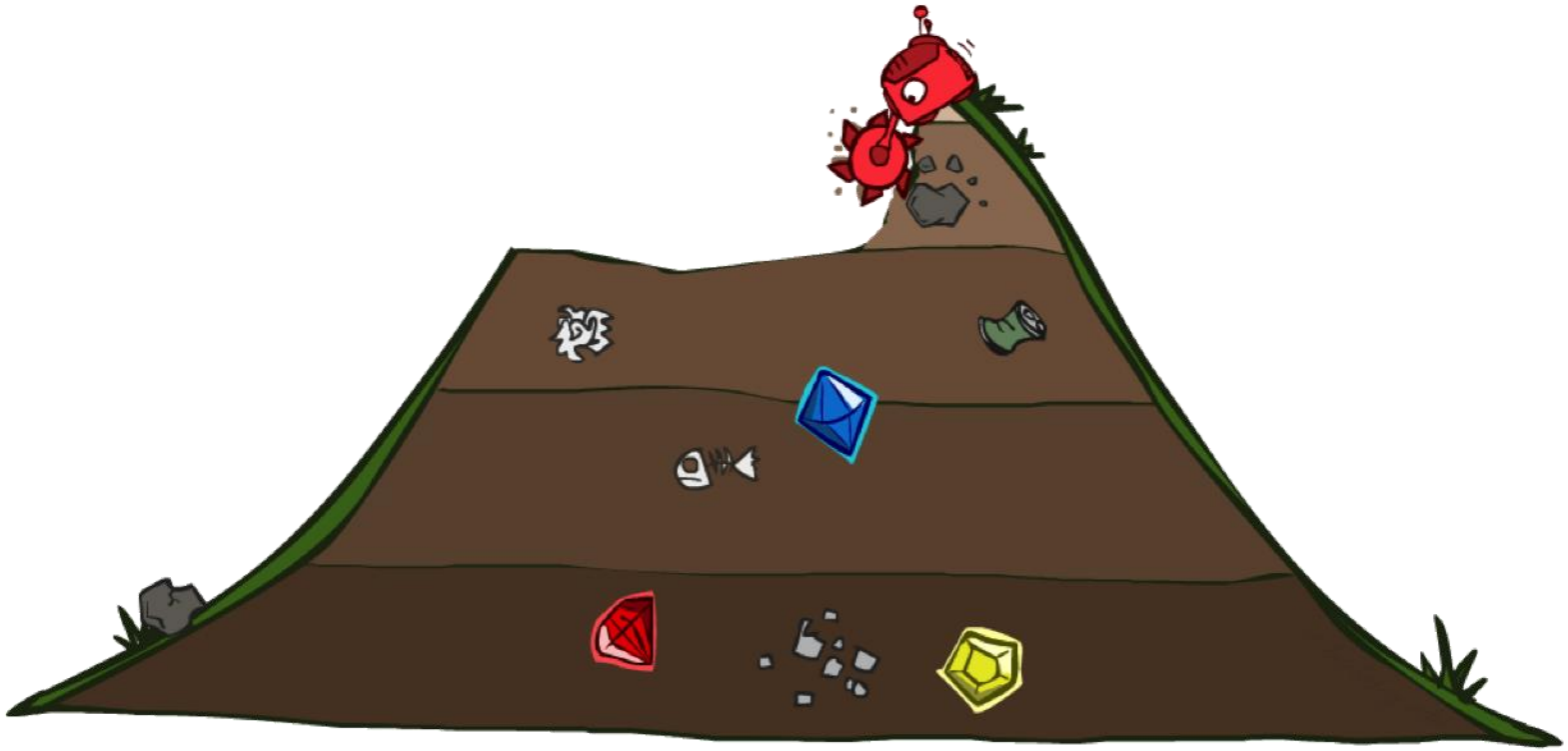


Cost-Sensitive Search



BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

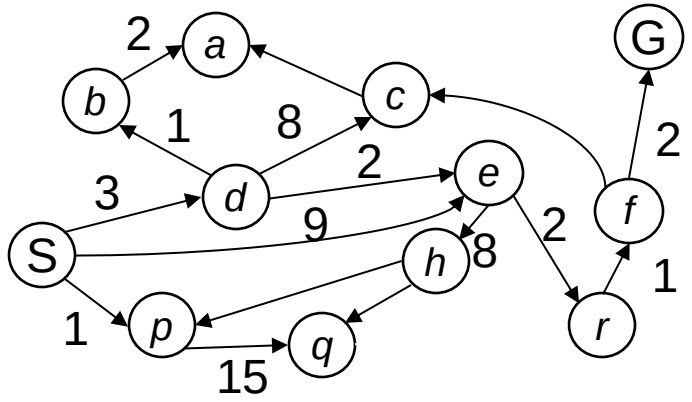
Uniform Cost Search



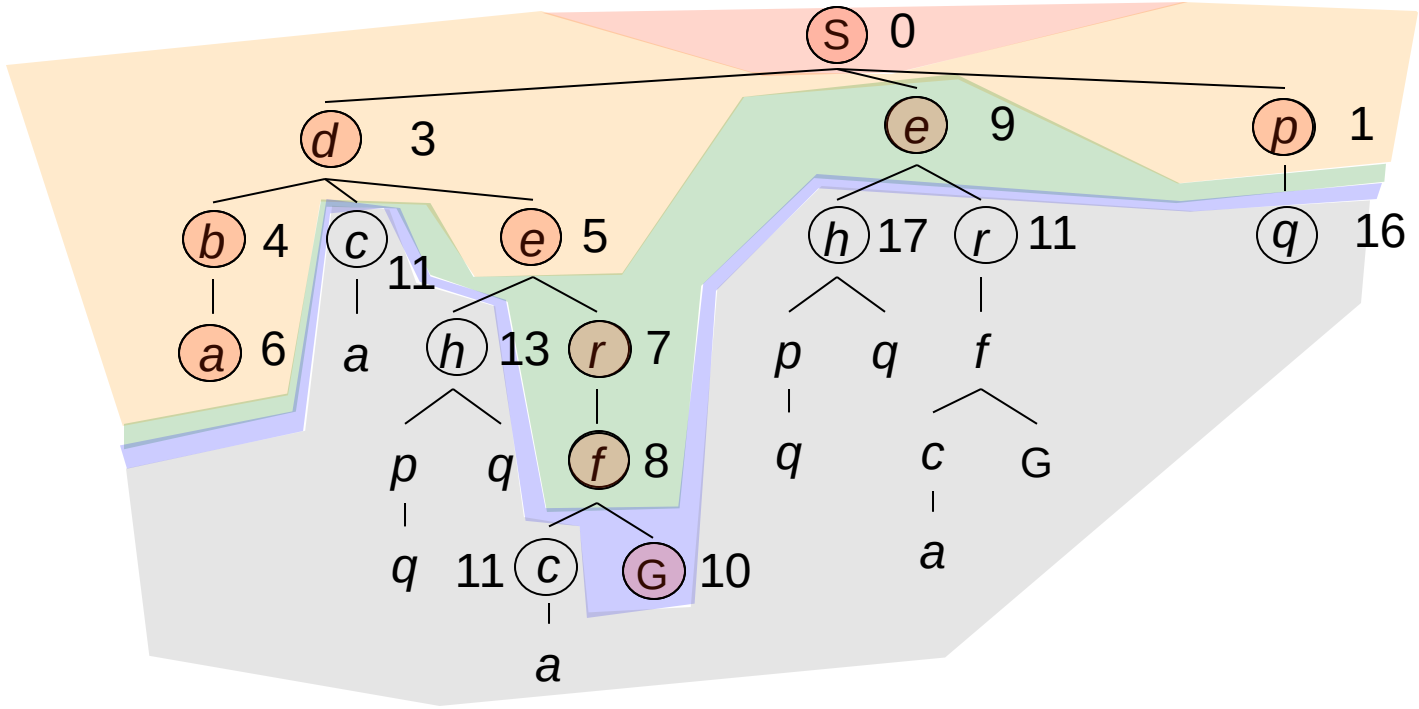
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

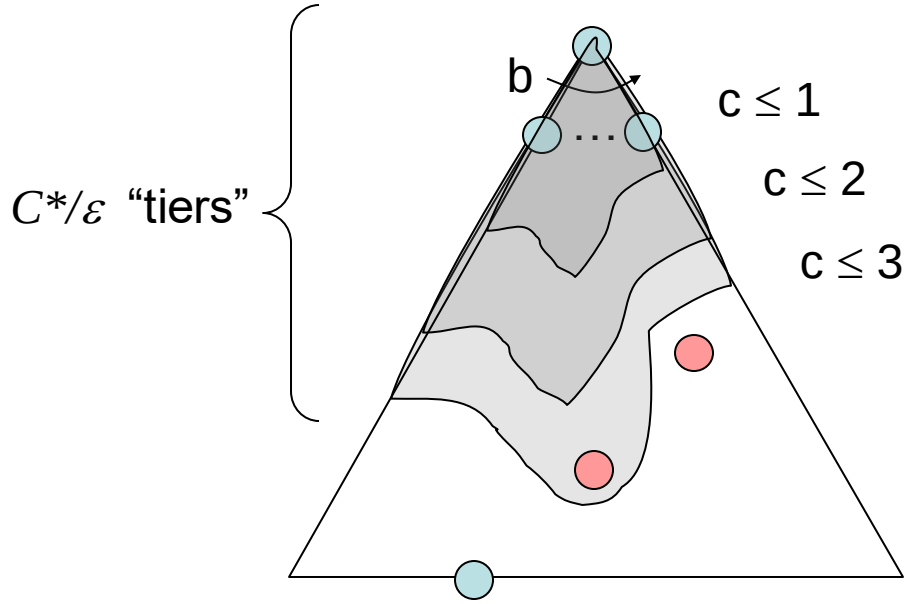


Cost contours



Uniform Cost Search (UCS) Properties

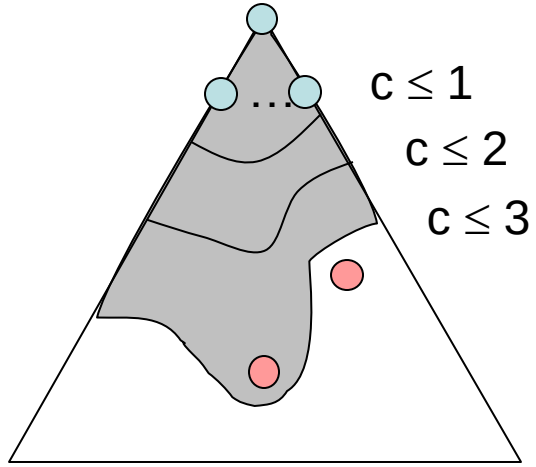
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)



- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

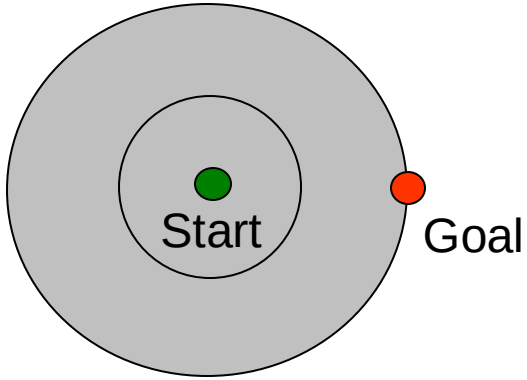
Uniform Cost Issues

- Remember: UCS explores increasing cost contours



- The good: UCS is complete and optimal!

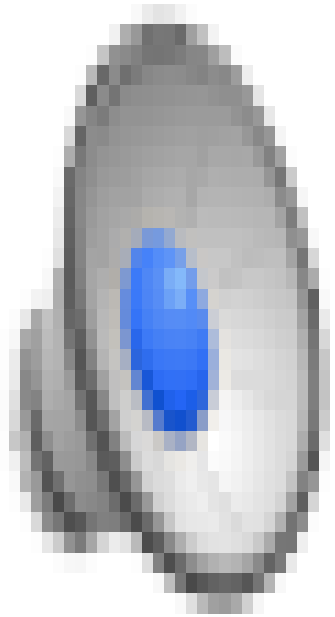
- The bad:
  - Explores options in every “direction”
  - No information about goal location



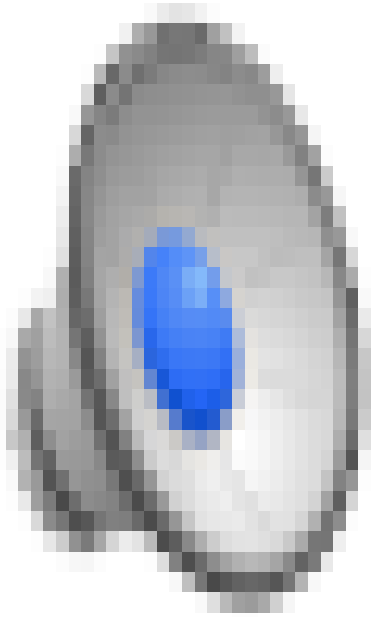
- We’ll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

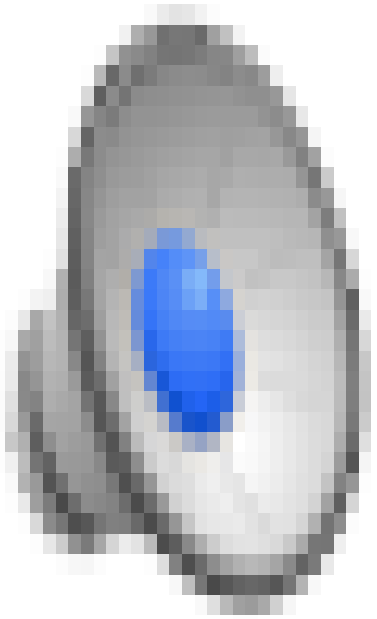
Video of Demo Empty UCS



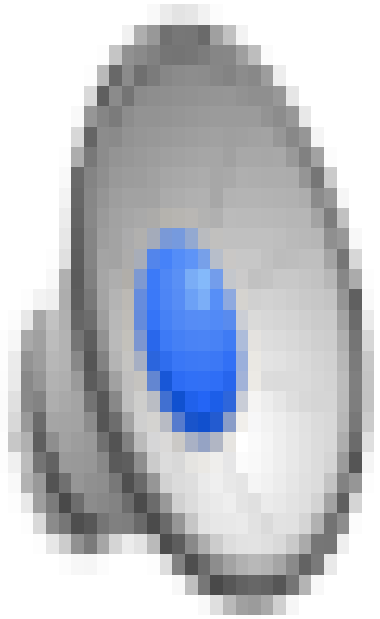
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 2)

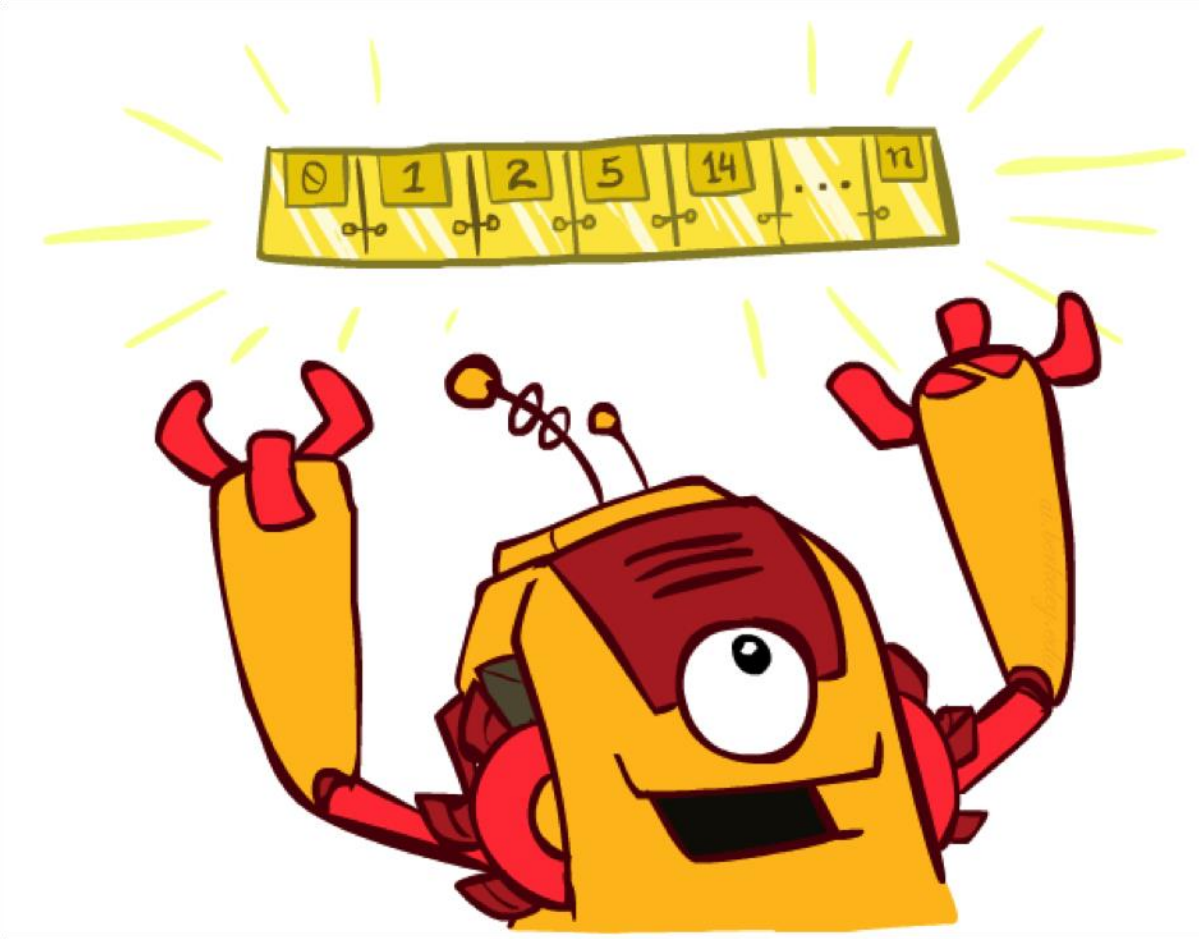


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)



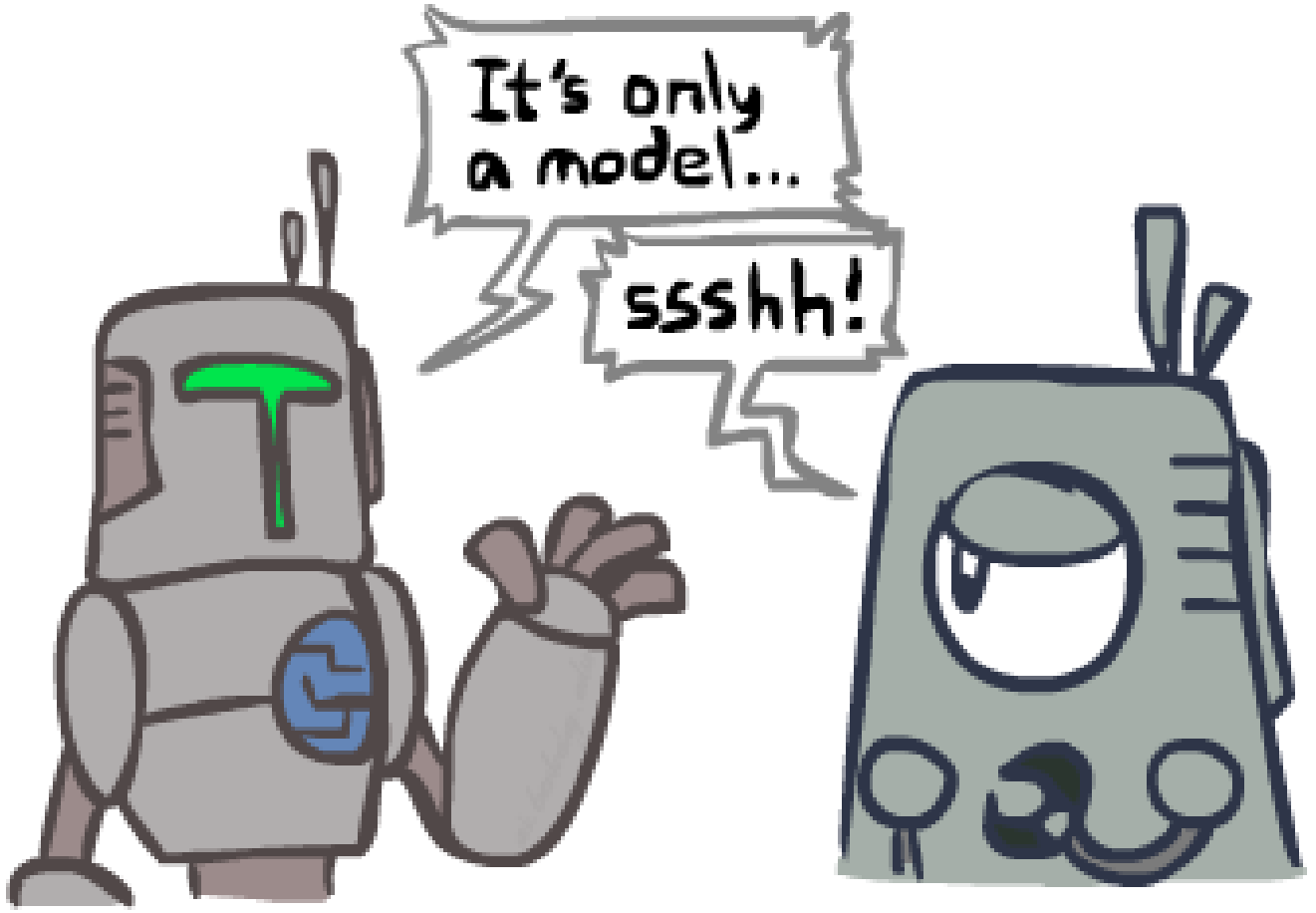
The One Queue

- 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

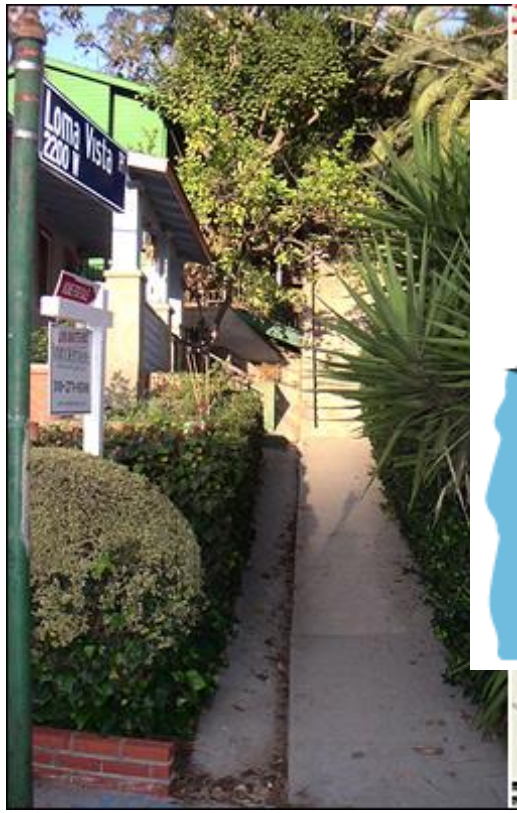


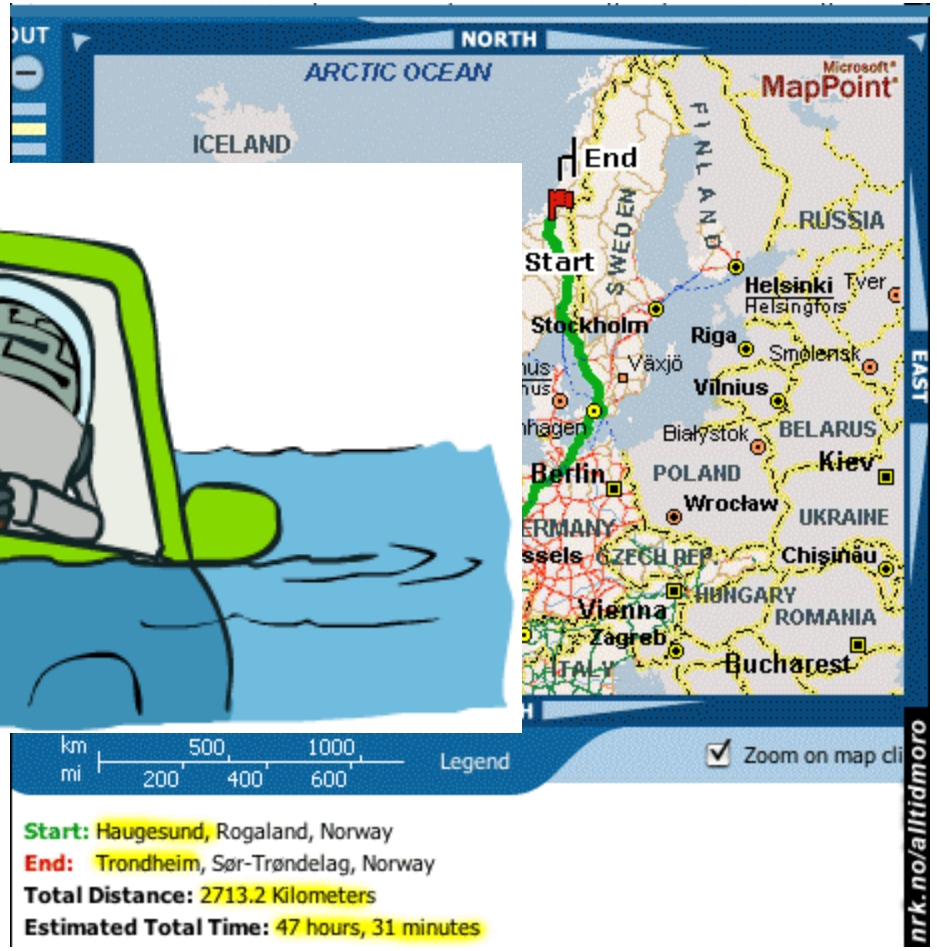
Search and Models

- 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…



Search Gone Wrong?





Homework 0: Search Algorithms

- **Due Date**: Saturday, 25-Mar-2025 11:59 PM (Guyana time)
- **Structure**:
    i. **Coding Portion**: Implement search algorithms in the PACMAN game.
    ii. **Written Problem-Solving**: Analyze search algorithms.
- **Please** start your homework early to avoid last-minute issues. You **must** show your work.

Coding Portion

- Implement and test the following search algorithms:
  - Depth-first search (DFS)
  - Breadth-first search (BFS)
  - Uniform-cost search (UCS)
  - A* search
  - Heuristics
  - Suboptimal search

Written Problem-Solving

- Solve problems related to search algorithms and state spaces.

Submission

- You **must** use Git for version control. We will be using your commits to track your progress.
- Create a pull request (PR) with your final submission.
- Submit the written portion to Moodle or email it to christopher.clarke@uog.edu.gy.

For more details, refer to the README.