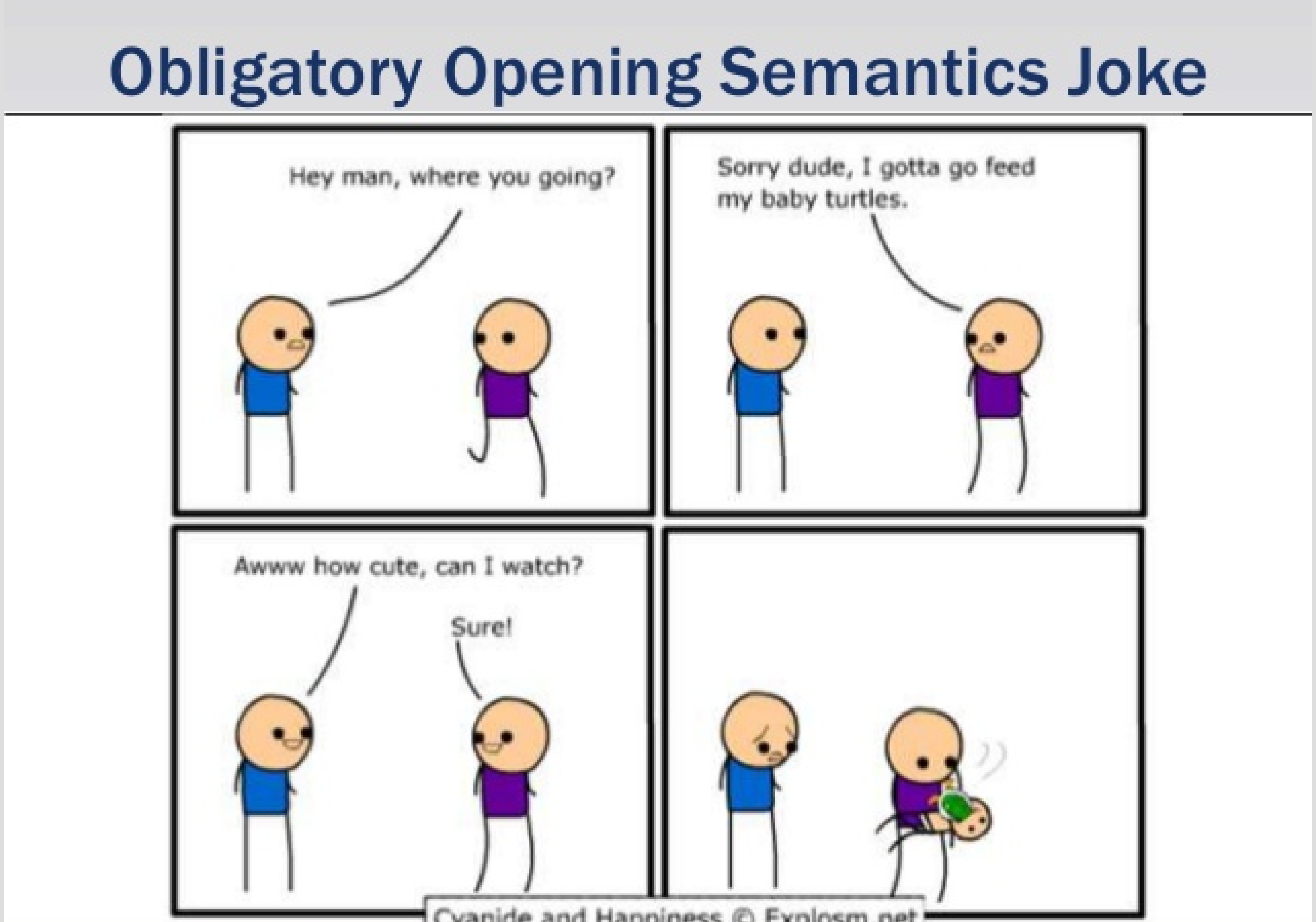
Rational Agents (Ch. 2)



Agent models

Can also classify agents into four categories:

1. Simple reflex
2. Model-based reflex
3. Goal based
4. Utility based

Top is typically simpler and harder to adapt to similar problems, while bottom is more general representations (generalization)

Agent models

A <u>simple reflex</u> agents acts only on the most recent part of the percept and not the whole history

Our vacuum agent is of this type, as it only looks at the current state and not any previous

These can be generalized as:
“if state = _____ then do action _____”
(often can fail or loop infinitely)

Agent models

A model-based reflex agent needs to have a representation of the environment in memory (called internal state)

This internal state is updated with each observation and then dictates actions

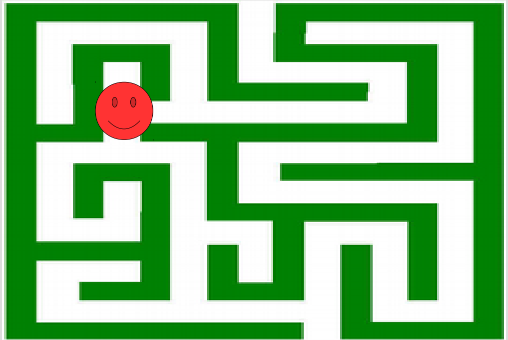
The degree that the environment is modeled is up to the agent/designer (a single bit vs. a full representation)

Agent models

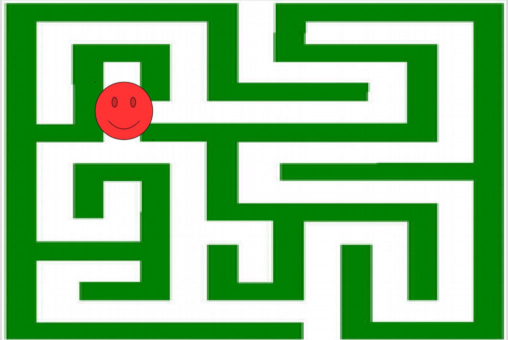
This internal state should be from the agent's perspective, not a global perspective
(as same global state might have different actions)

Consider these pictures of a maze:
Which way to go?

Pic 1



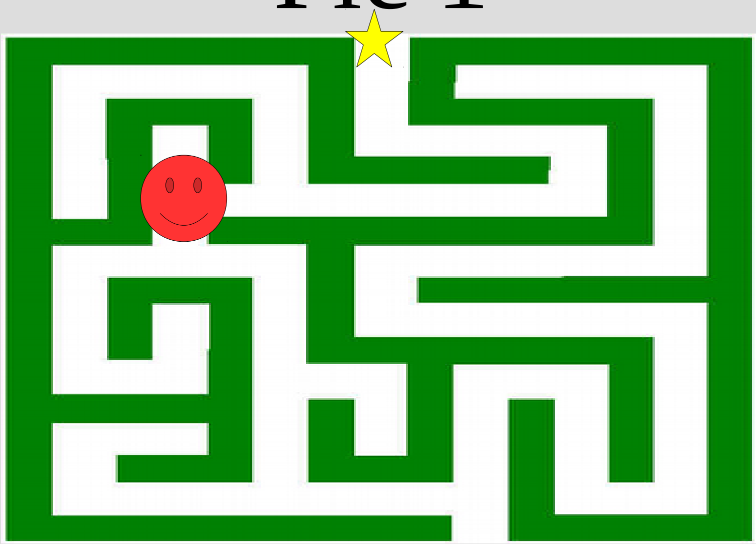
Pic 2



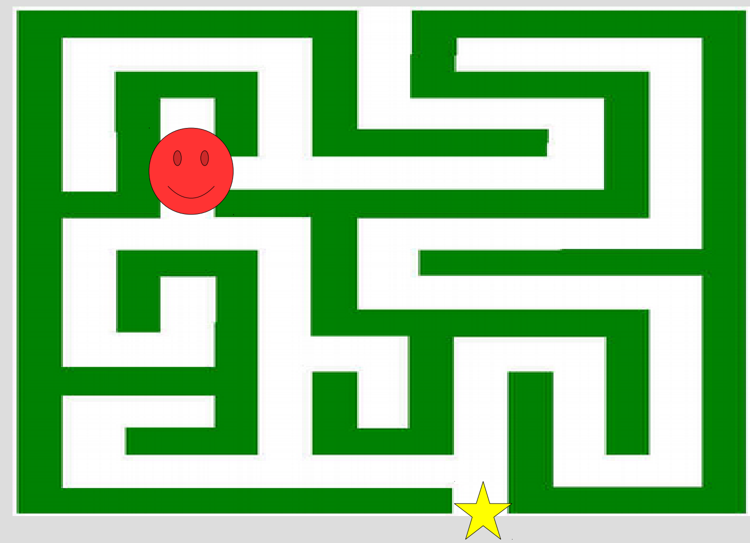
Agent models

The global perspective is the same, but the agents could have different goals (stars)

Pic 1



Pic 2



Goals are not global information

Agent models

We also saw this when we were talking about agent functions (also from agent's perspective, not global)

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
$\vdots$	$\vdots$

Agent models

For the vacuum agent if the dirt does not reappear, then we do not want to keep moving

The simple reflex agent program cannot do this, so we would have to have some memory (or model)

This could be as simple as a flag indicating whether or not we have checked the other state

Agent models

The goal based agent is more general than the model-based agent

In addition to the environment model, it has a goal indicating a desired configuration

Abstracting to a goals generalizes your method to different (similar) problems
(for example, a model-based agent a specific tree/graph, goal-based can solve any)

Agent models

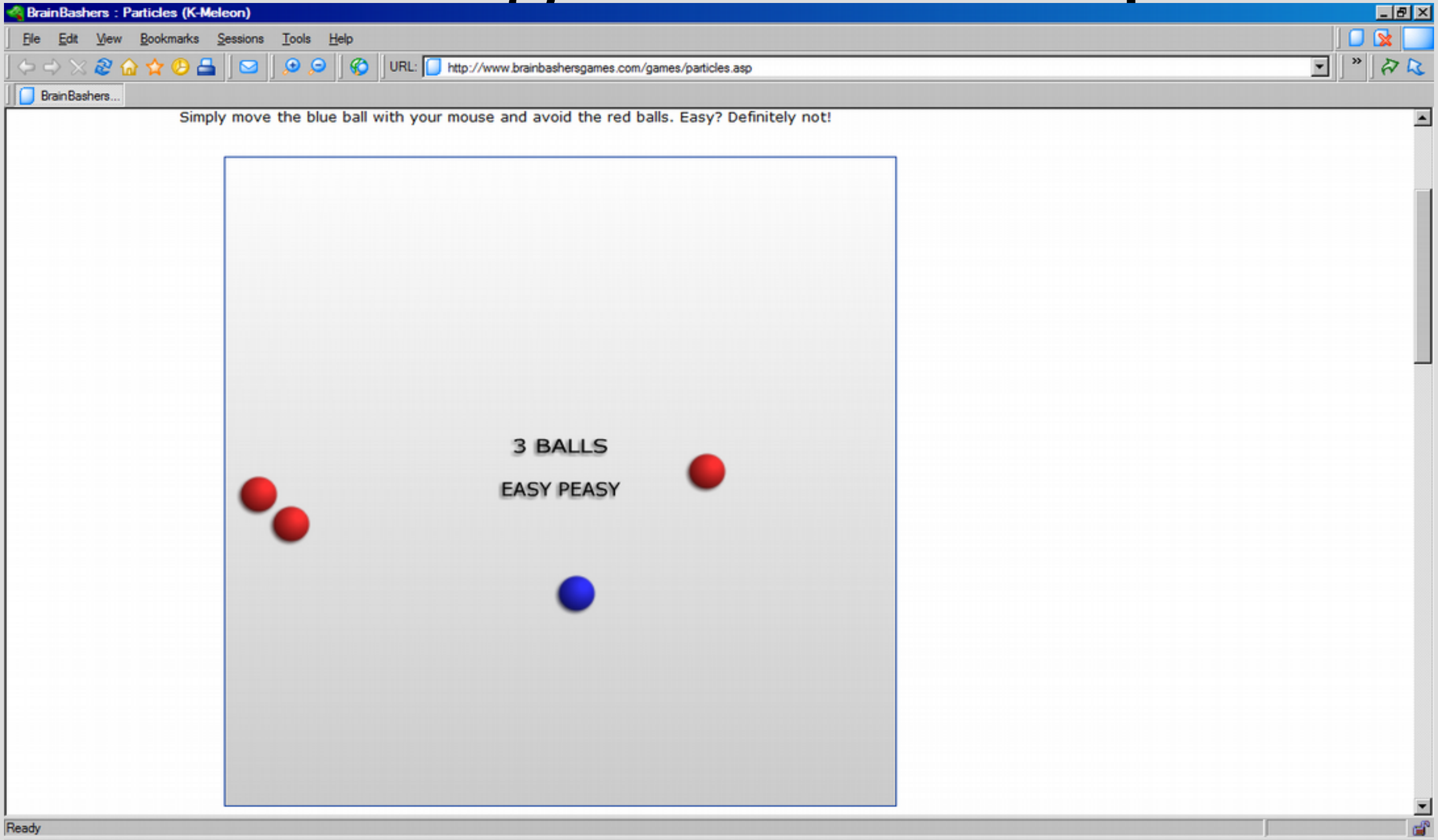
A utility based agent maps the sequence of states (or actions) to a real value

Goals can describe general terms as “success” or “failure”, but there is no degree of success

If you want to go upstairs, a goal based agent could find the closest way up...
A utility based agent could accommodate your preferences between stairs vs. elevator

Agent models

What is the agent model of particles?



Think of a way to improve the agent and describe what model it is now

Environment classification

Environments can be further classified on the following characteristics:(right side harder)

1. Fully vs. partially observable
2. Single vs. multi-agent
3. Deterministic vs. stochastic
4. Episodic vs. sequential
5. Static vs. dynamic
6. Discrete vs. continuous
7. Known vs. unknown

Environment classification

In a fully observable environment, agents can see every part.

Agents can only see part of the environment if it is partially observable

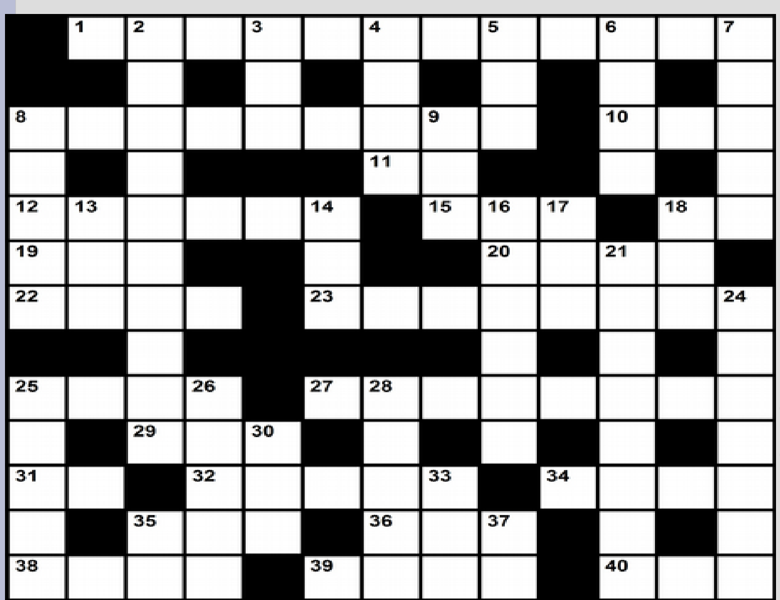
Full

Partial

Environment classification

If your agent is the only one, the environment is a single agent environment

More than one is a multi-agent environment (possibly cooperative or competitive)



← single

multi →

Environment classification

If your state+action has a single known outcome in the environment, it is <u>deterministic</u>

If actions have a distribution (probability) of possible effects, it is <u>stochastic</u>

deterministic

stochastic

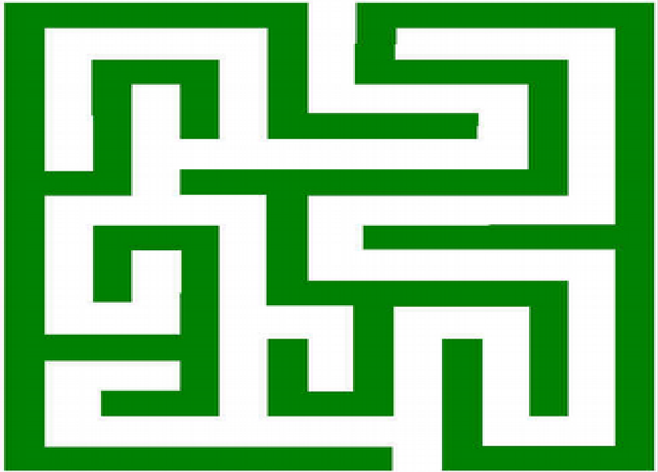
Environment classification

An <u>episodic</u> environment is where the previous action does not effect the next observation (i.e. can be broken into independent events)

If there is the next action depends on the previous, the environment is <u>sequential</u>

episodic

sequential



Environment classification

If the environment only changes when you make an action, it is <u>static</u>

a <u>dynamic</u> environment can change while your agent is thinking or observing

static

dynamic

Environment classification

Discrete = separate/distinct (events)
Continuous = fluid transition (between events)

This classification can applies: agent's percept and actions, environment's time and states

discrete (state)

continuous (state)

Environment classification

<u>Known</u> = agent's actions have known effects on the environment

<u>Unknown</u> = the actions have an initially unknown effect on the environment (can learn)

know how to stop

do not know how to stop

Environment classification

1. Fully vs. partially observable = how much can you see?
2. Single vs. multi-agent
   = do you need to worry about others interacting?
3. Deterministic vs. stochastic
   = do you know (exactly) the outcomes of actions?
4. Episodic vs. sequential
   = do your past choices effect the future?
5. Static vs. dynamic = do you have time to think?
6. Discrete vs. continuous
   = are you restricted on where you can be?
7. Known vs. unknown
   = do you know the rules of the game?

Environment classification

Some of these classifications are associated with the state, while others with the actions

State: Actions:

1. Fully vs. partially observable
2. Single vs. multi-agent
3. Deterministic vs. stochastic
4. Episodic vs. sequential
5. Static vs. dynamic
6. Discrete vs. continuous
7. Known vs. unknown

Environment classification

Pick a game/hobby/sport/pastime/whatever and describe both the PEAS and whether the environment/agent is:

1. Fully vs. partially observable
2. Single vs. multi-agent
3. Deterministic vs. stochastic
4. Episodic vs. sequential
5. Static vs. dynamic
6. Discrete vs. continuous
7. Known vs. unknown

Environment classification

What?	Perfor mance	Environ ment	Actuator s	Sensors
Ring fit	level score	multiple tracks	wheel move	wheel, leg pos

Partially observable, single agent, deterministic, sequential, dynamic (sorta), continuous, known (tells you what to do if stuck)

State structure

An atomic state has no sub-parts and acts as a simple unique identifier

An example is an elevator:
Elevator = agent (actions = up/down)
Floor = state

In this example, when someone requests the elevator on floor 7, the only information the agent has is what floor it currently is on

State structure

A factored state has a fixed number of variables/attributes associated with it

You can then reason on how these associated values change between states to solve problem

Can always "un-factor" and enumerate all possibilities to go back to atomic states,
but might be too exponential or lose efficiency

State structure

Structured states simply describe objects and their relationship to others

Suppose we have 3 blocks: A, B and C
We could describe: A on top of B, C next to B

A factored representation would have to enumerate all possible configurations of A, B and C to be as representative

State structure

We will start using <u>structured</u> approaches when we deal with logic:

Summer implies Warm
Warm implies T-Shirt

The current state might be:
!Summer (¬Summer)
but the states have intrinsic relations between each other (not just actions)

Search

Goal based agents need to search to find a path from their start to the goal (a path is a sequence of actions, not states)

For now we consider problem solving agents who search on atomically structured spaces

We will focus on uninformed searches for now, which only know cost between states but no other extra information

Search

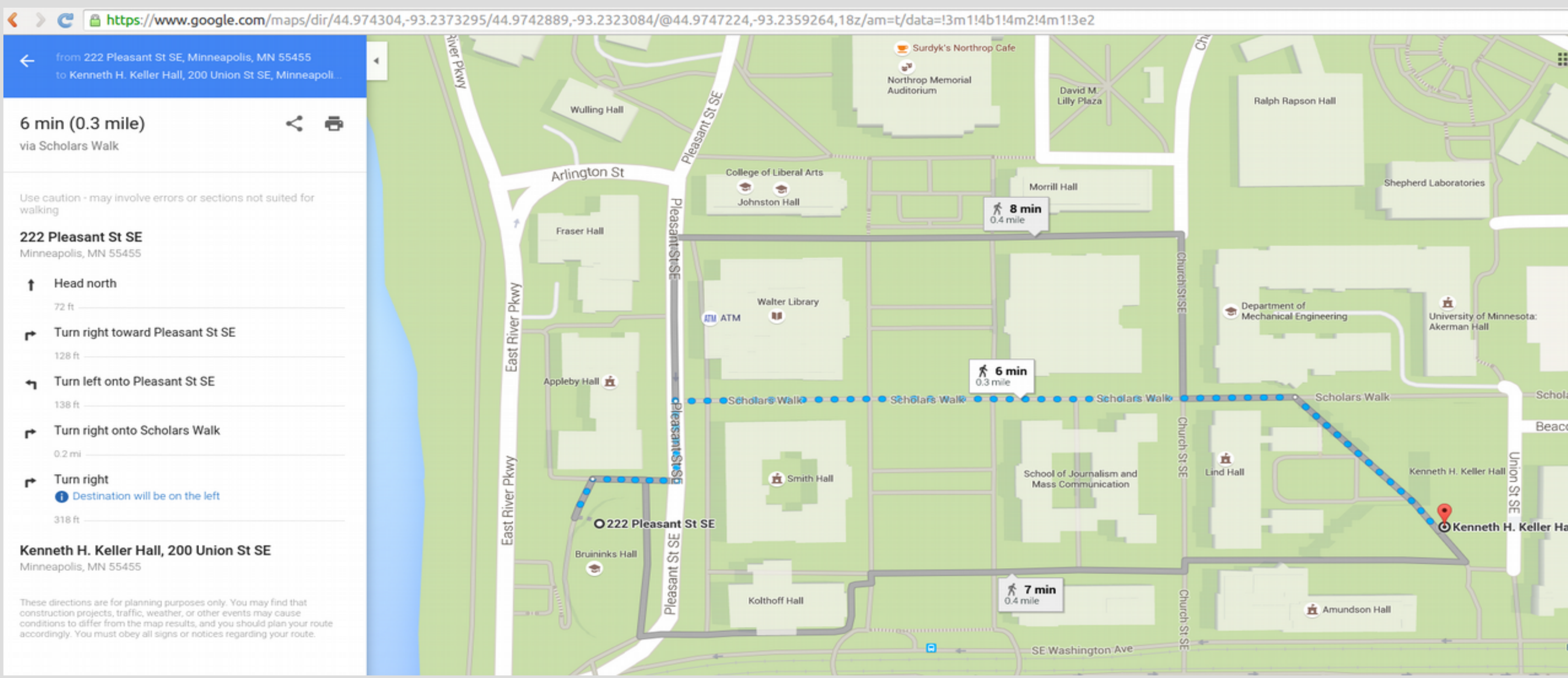
In the vacuum example, the **states** and **actions** I gave upfront (so only one option)

In more complex environments, we have a choice of how to abstract the problem into simple (yet expressive) states and actions

The solution to the abstracted problem should be able to serve as the basis of a more detailed problem (i.e. fit the detailed solution inside)

Search

Example: Google maps gives direction by telling you a sequence of roads and does not dictate speed, stop signs/lights, road lane



Search

In deterministic environments the search solution is a single sequence (list of actions)

Stochastic environments need multiple sequences to account for all possible outcomes of actions

It can be costly to keep track of all of these and might be better to keep the most likely and search again when off the main sequences

Search

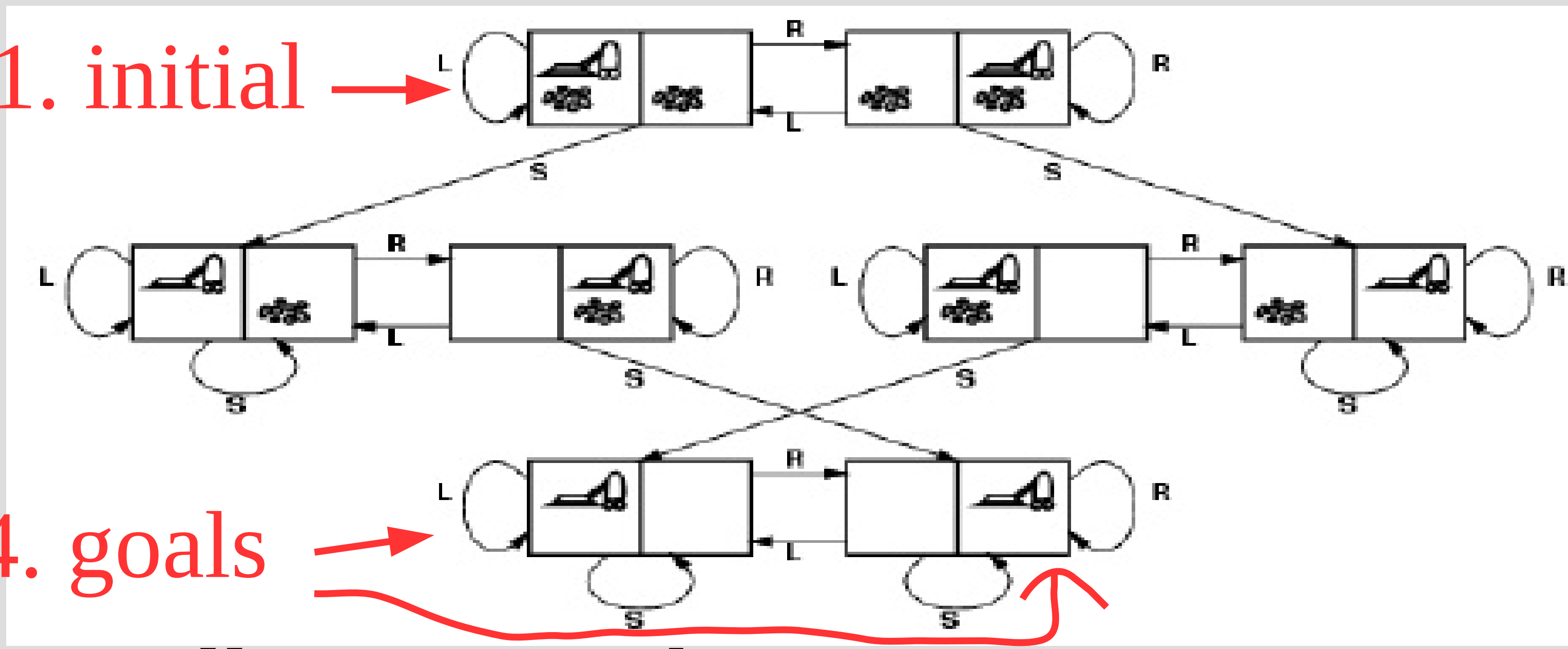
There are 5 parts to search:

1. Initial state states are nodes in tree/graph
2. Actions possible at each state actions are edges
3. Transition model (result of each action)
4. Goal test (are we there yet?)
5. Path costs/weights (not stored in states) (related to performance measure)

In search we normally fully see the problem and the initial state and compute all actions

Small examples

Here is our vacuum world again:



2. For all states, we have actions: L, R or S

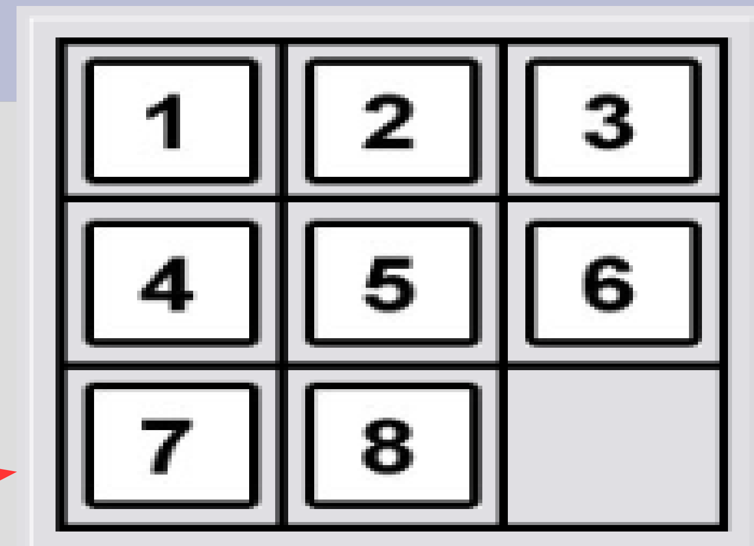
3. Transition model = black arrows

5. Path cost = ??? (from performance measure)

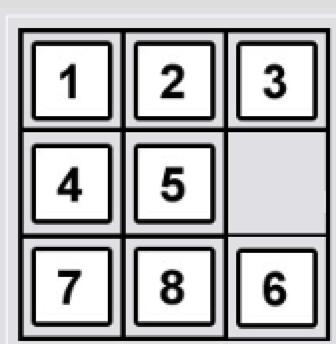
Small examples

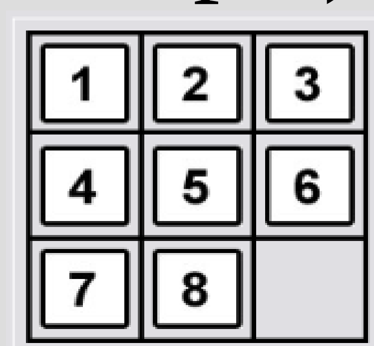
8-Puzzle

1. (semi) Random
2. All states: U,D,L,R
4. As shown here



5. Path cost = 1 (move count)
3. Transition model (example):

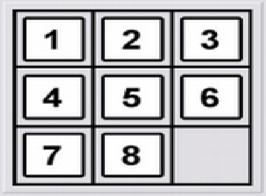
Result(

,D) = 

(see: https://www.youtube.com/watch?v=DfVjTkzk2Ig)

Small examples

8-Puzzle is NP complete so to find the best solution, we must brute force

3x3 board = 

= 181,440 states

4x4 board = 1.3 trillion states

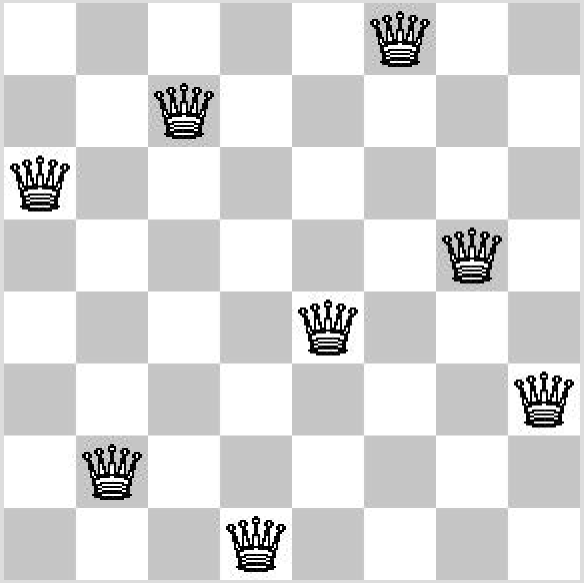
Solution time: milliseconds

5x5 board = 10^{25} states

Solution time: hours

Small examples

8-Queens: how to fit 8 queens on a 8x8 board so no 2 queens can capture each other



Two ways to model this:

<u>Incremental</u> = each action is to add a queen to the board (1.8×10^{14} states)

<u>Complete state formulation</u> = all 8 queens start on board, action = move a queen (2057 states)

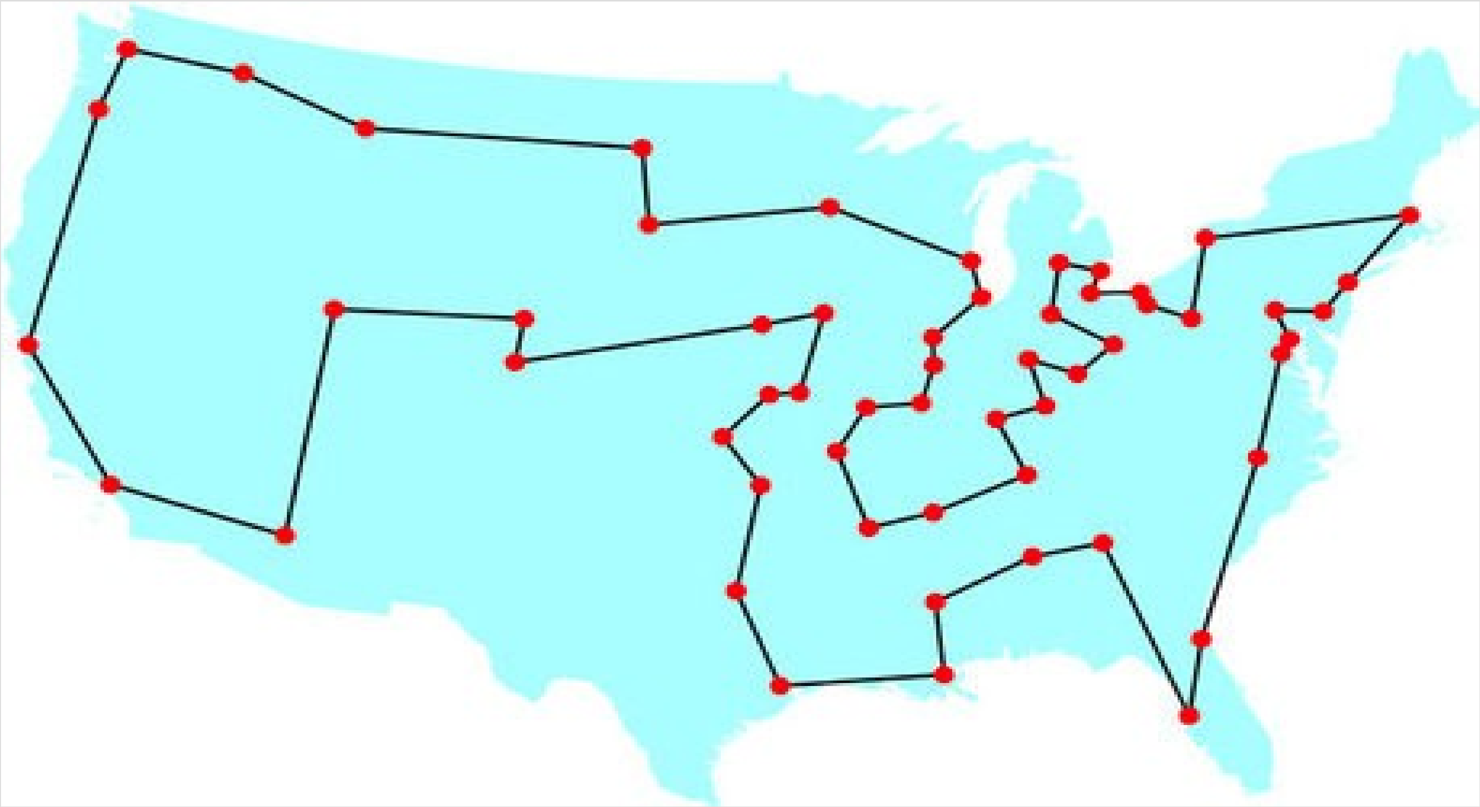
Real world examples

Directions/traveling (land or air)

Model choices: only have interstates?
Add smaller roads, with increased cost?
(pointless if they are never taken)

Real world examples

Traveling salesperson problem (TSP): Visit each location exactly once and return to start



Goal: Minimize distance traveled

Search algorithm

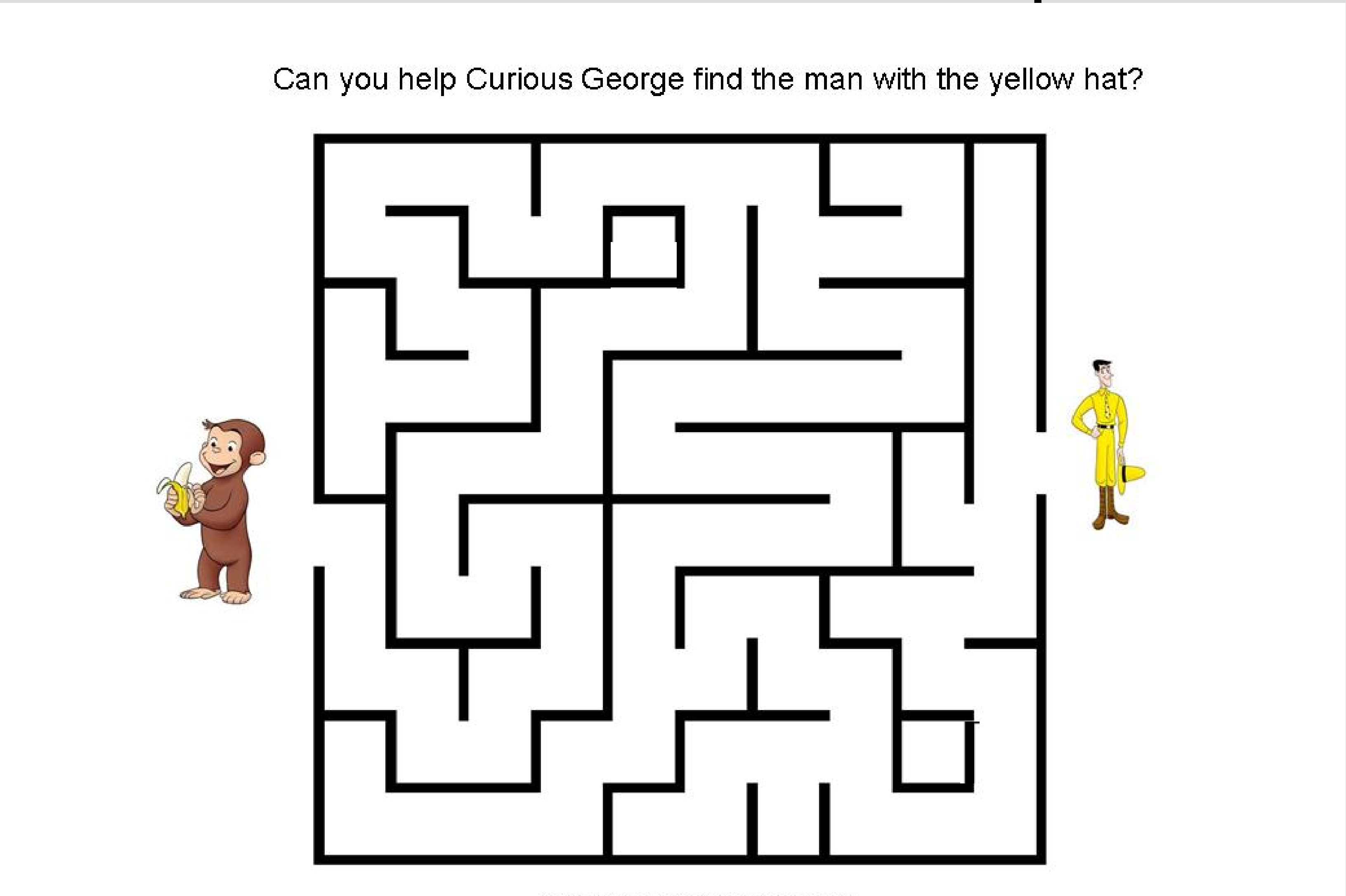
To search, we will build a tree with the root as the initial state

```
function tree-search(root-node)
   fringe ← successors(root-node)
   while ( notempty(fringe) )
         {node ← remove-first(fringe)
          state ← state(node)
          if goal-test(state) return solution(node)
          fringe ← insert-all(successors(node),fringe) }
   return failure
end tree-search
```

(Use same procedure for multiple algorithms)

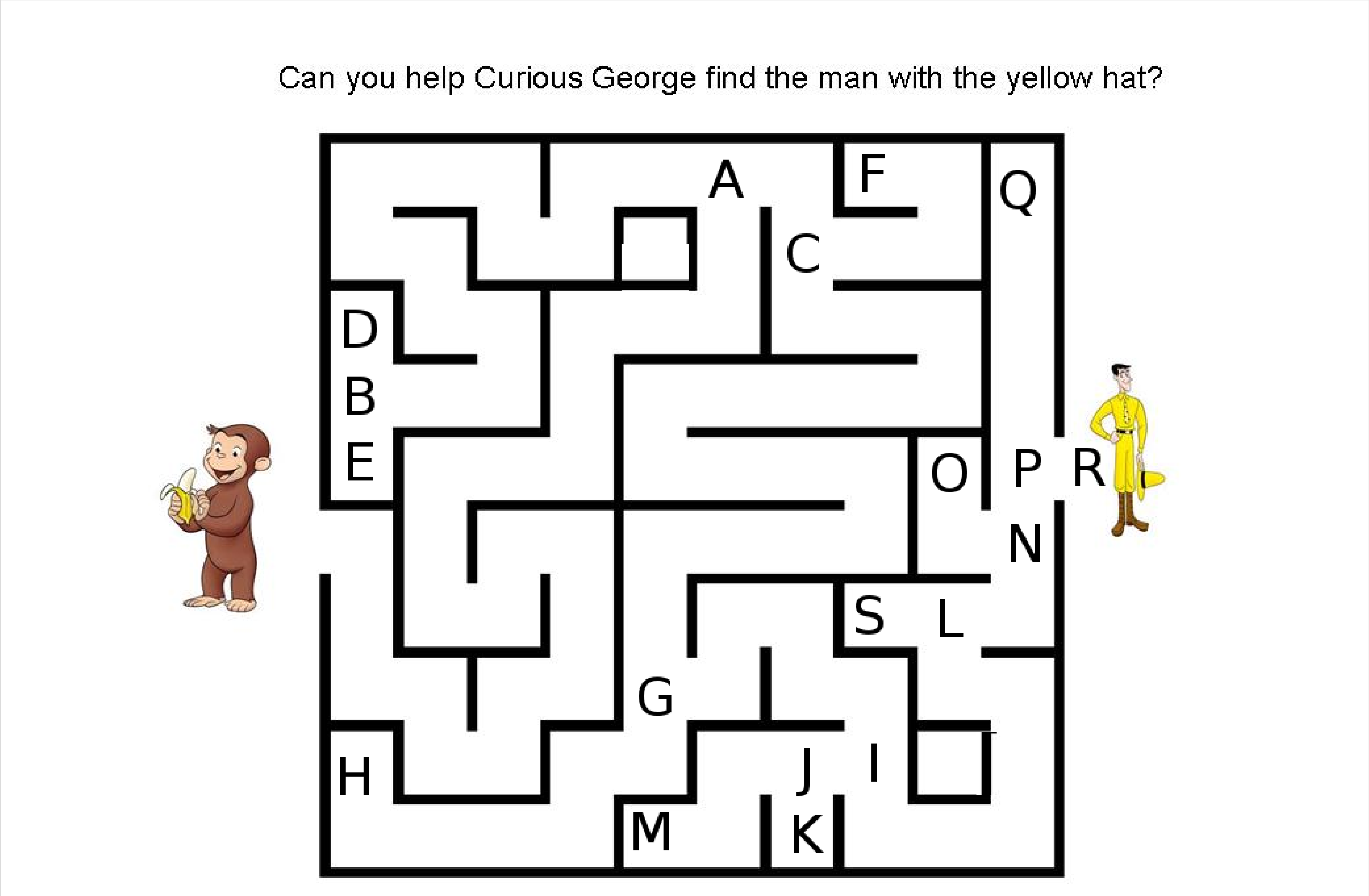
Search algorithm

What are states/actions for this problem?



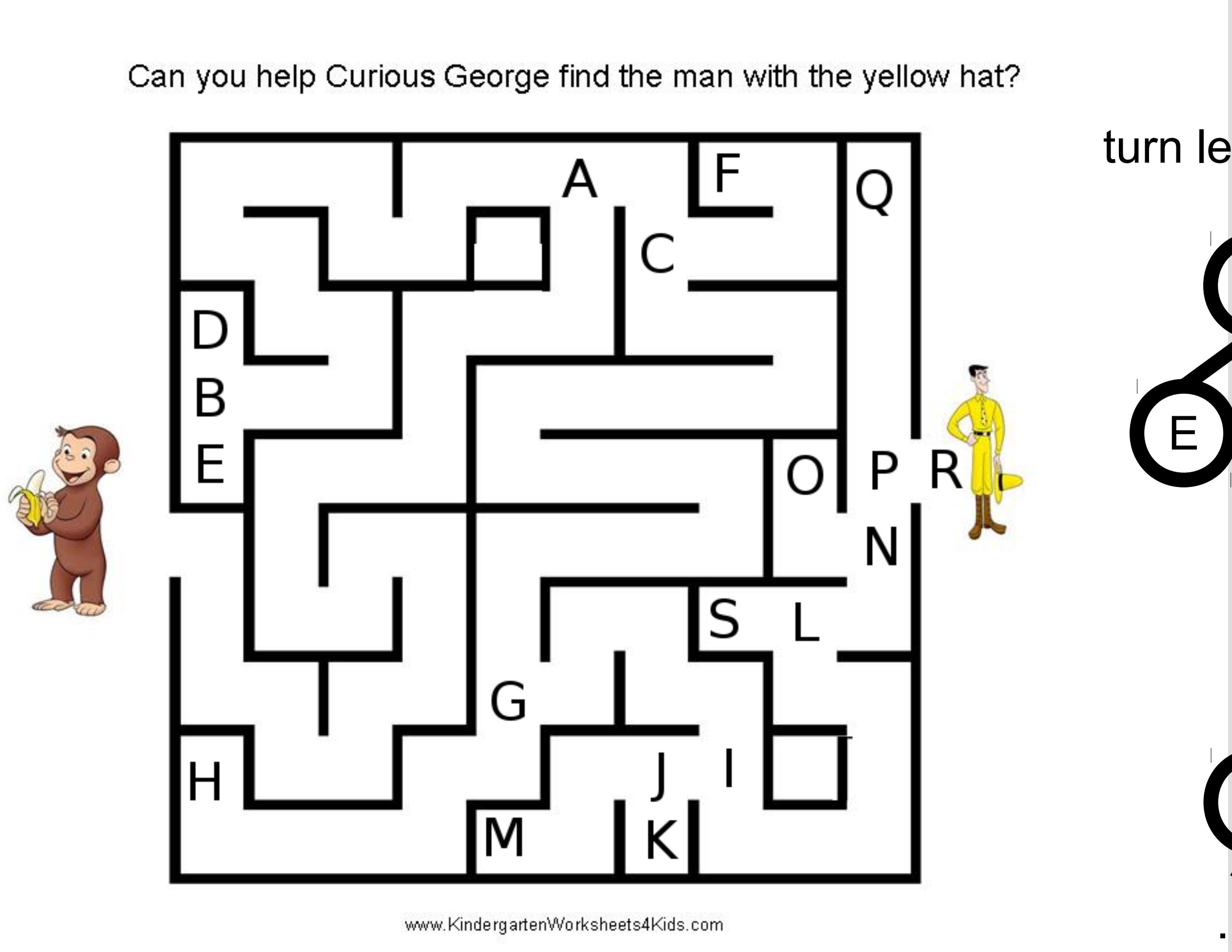
Search algorithm

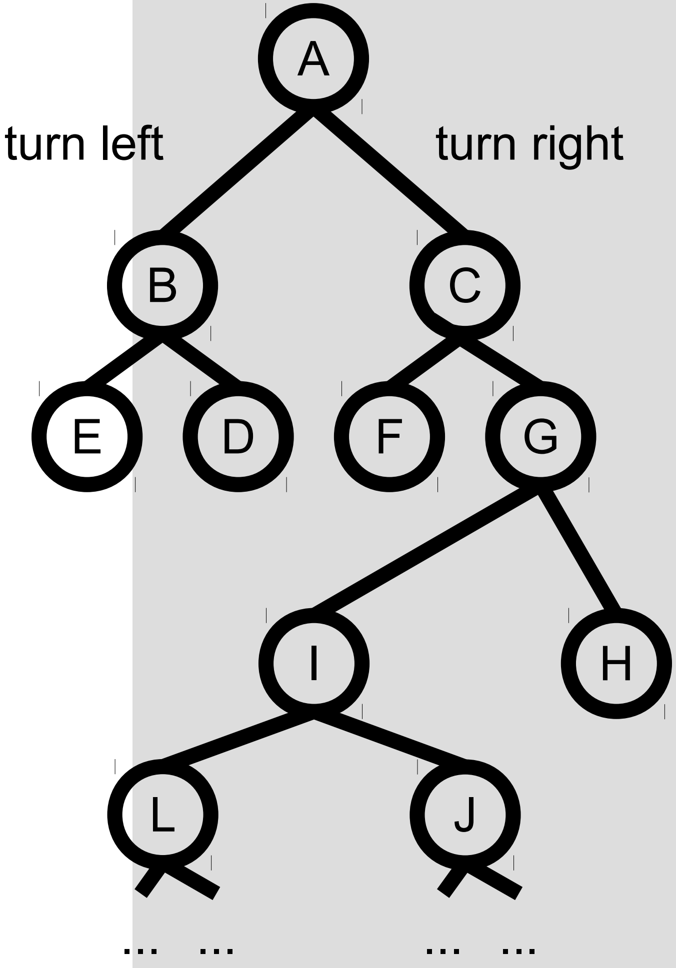
Multiple options, but this is a good choice



Search algorithm

Multiple options, but this is a good choice



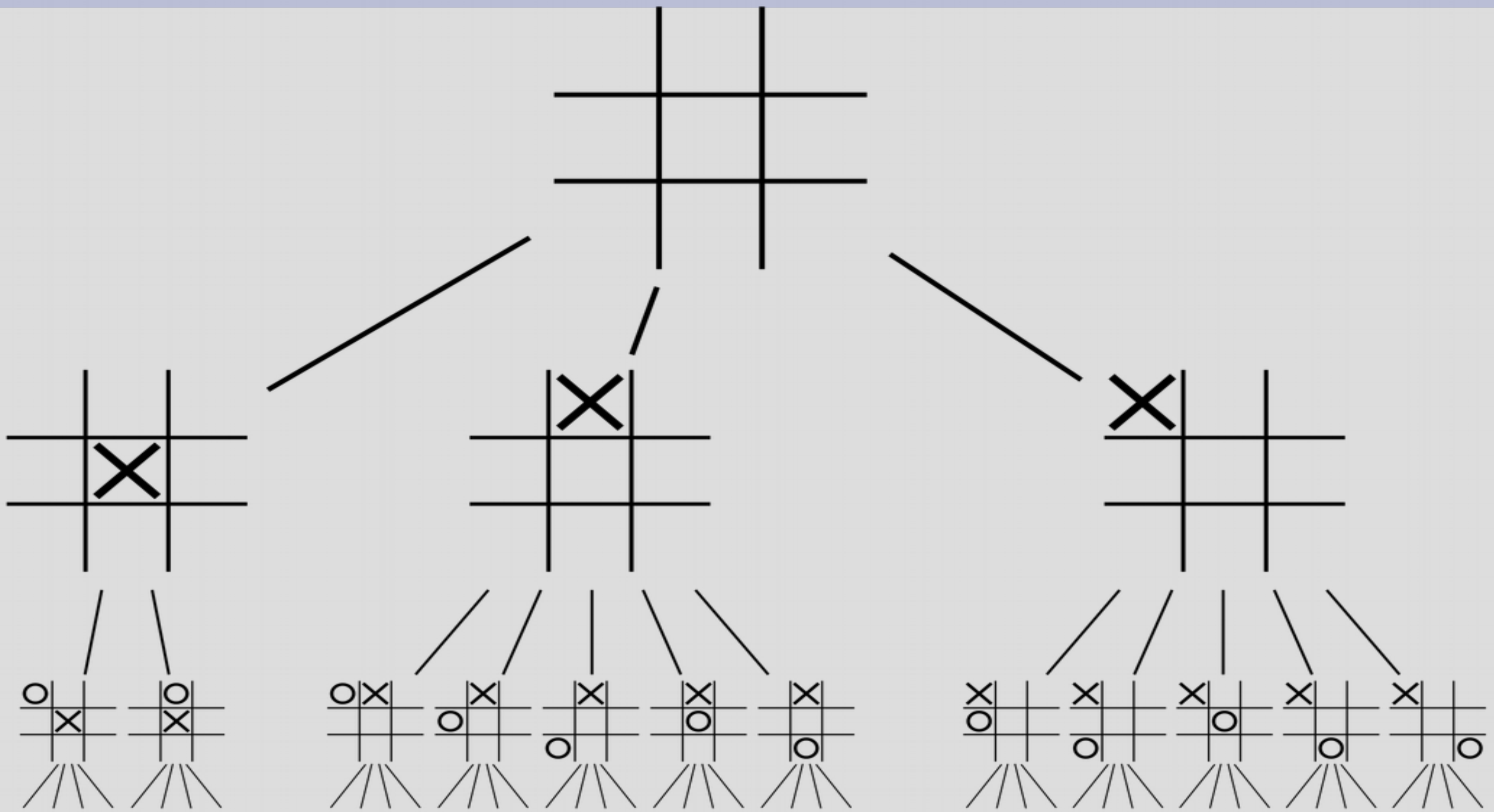


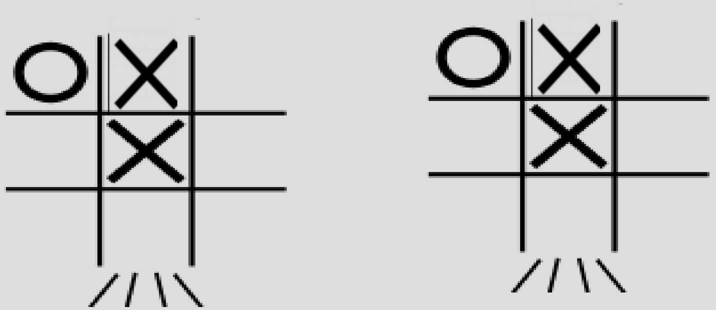
Search algorithm

What are the problems with this?

```
function tree-search(root-node)
  fringe ← successors(root-node)
  while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node),fringe) }
  return failure
end tree-search
```

Search algorithm





Search algorithm

We can remove visiting states multiple times by doing this:

```
function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
      {node ← remove-first(fringe)
        state ← state(node)
        if goal-test(state) return solution(node)
        explored ← insert(node,explored)
        fringe ← insert-all(successors(node),fringe,  if node not in explored)
      }
  return failure
end tree-search
```

But this is still not necessarily all that great...

Search algorithm

When we find a goal state, we can back track via the parent to get the sequence

To keep track of the unexplored nodes, we will use a queue (of various types)

The explored set is probably best as a hash table for quick lookup (have to ensure similar states reached via alternative paths are the same in the has, can be done by sorting)