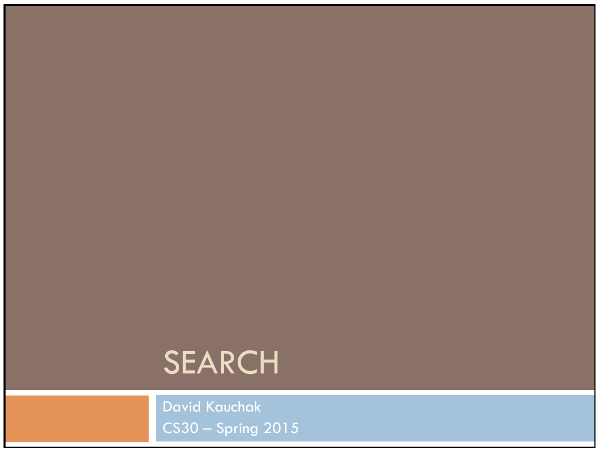
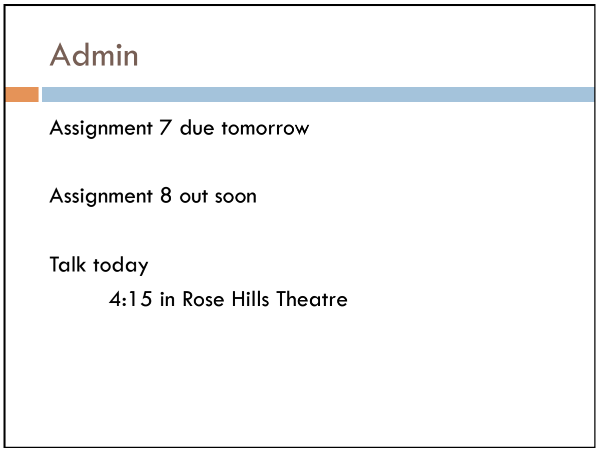


SEARCH

David Kauchak
CS30 – Spring 2015



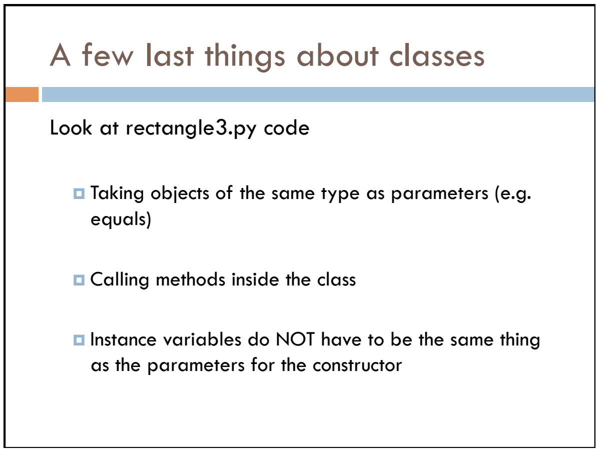
Admin

Assignment 7 due tomorrow

Assignment 8 out soon

Talk today

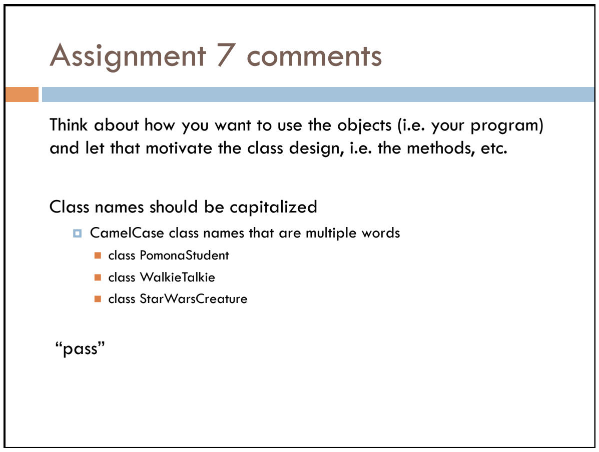
4:15 in Rose Hills Theatre



A few last things about classes

Look at rectangle3.py code

- Taking objects of the same type as parameters (e.g. equals)
- Calling methods inside the class
- Instance variables do NOT have to be the same thing as the parameters for the constructor



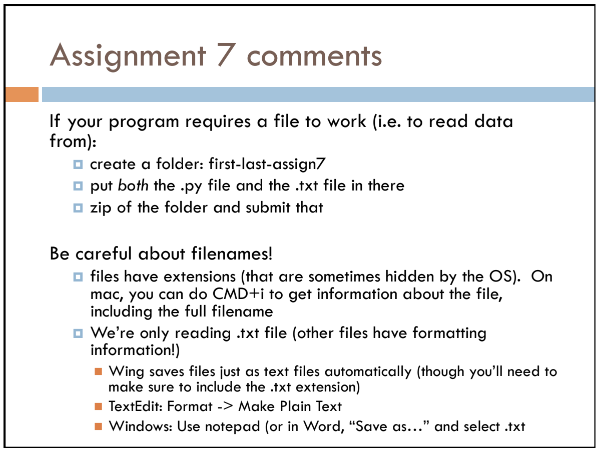
Assignment 7 comments

Think about how you want to use the objects (i.e. your program) and let that motivate the class design, i.e. the methods, etc.

Class names should be capitalized

- CamelCase class names that are multiple words
  - class PomonaStudent
  - class WalkieTalkie
  - class StarWarsCreature

"pass"



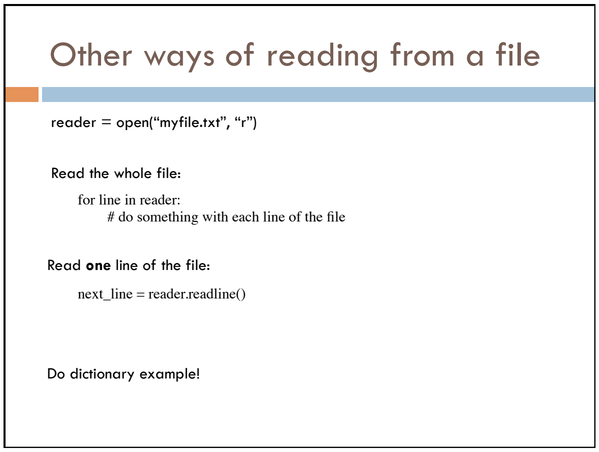
Assignment 7 comments

If your program requires a file to work (i.e. to read data from):

- create a folder: first-last-assign7
- put *both* the .py file and the .txt file in there
- zip of the folder and submit that

Be careful about filenames!

- files have extensions (that are sometimes hidden by the OS). On mac, you can do CMD+i to get information about the file, including the full filename
- We're only reading .txt file (other files have formatting information!)
  - Wing saves files just as text files automatically (though you'll need to make sure to include the .txt extension)
  - TextEdit: Format -> Make Plain Text
  - Windows: Use notepad (or in Word, "Save as…" and select .txt



Other ways of reading from a file

```
reader = open("myfile.txt", "r")
```

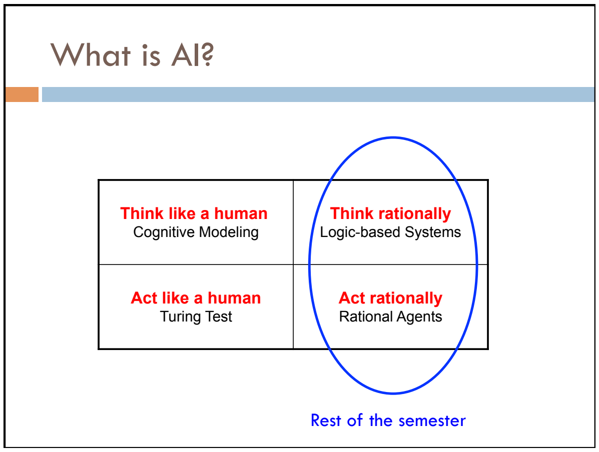
Read the whole file:

```
for line in reader:
    # do something with each line of the file
```

Read **one** line of the file:

```
next_line = reader.readline()
```

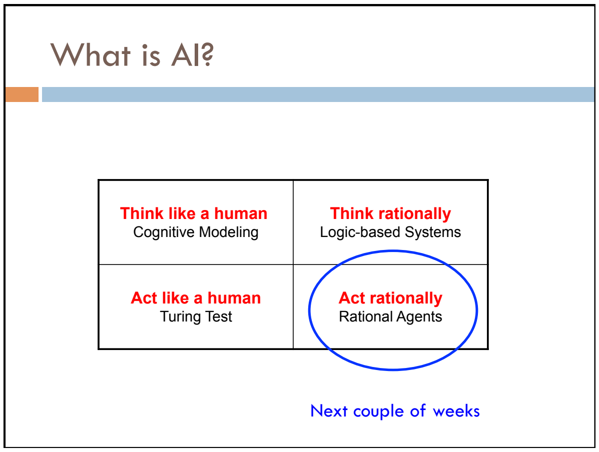
Do dictionary example!



What is AI?

Think like a human Cognitive Modeling	**Think rationally** Logic-based Systems
Act like a human Turing Test	**Act rationally** Rational Agents

Rest of the semester

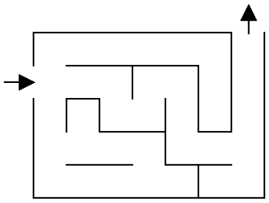


What is AI?

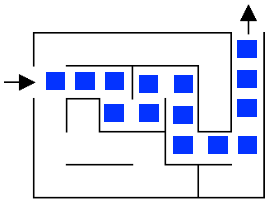
Think like a human Cognitive Modeling	**Think rationally** Logic-based Systems
Act like a human Turing Test	**Act rationally** Rational Agents

Next couple of weeks

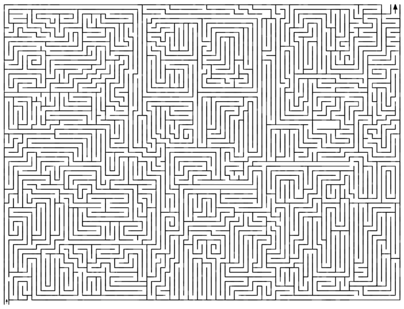
Solve the maze!



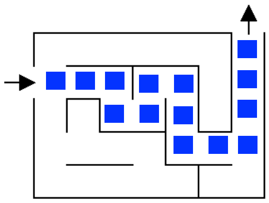
Solve the maze!



Solve the maze!

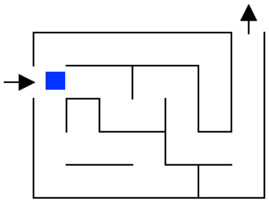


Solve the maze!



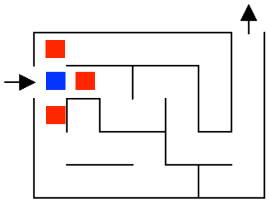
How did you figure it out?

One approach



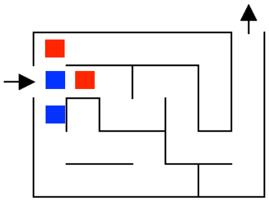
What now?

One approach



Three choices

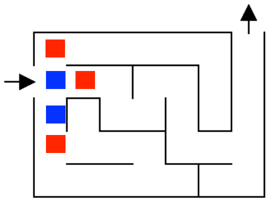
One approach



Pick one!

What now?

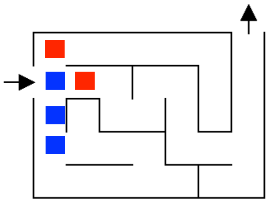
One approach



Still three options!

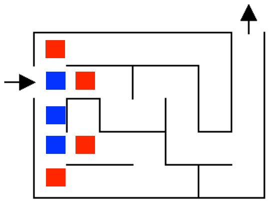
Which would you explore/pick?

One approach



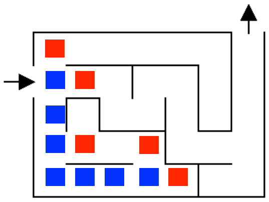
Most people go down a single path until they realize that it’s wrong

One approach



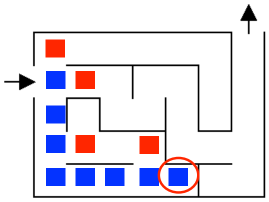
Keep exploring

One approach

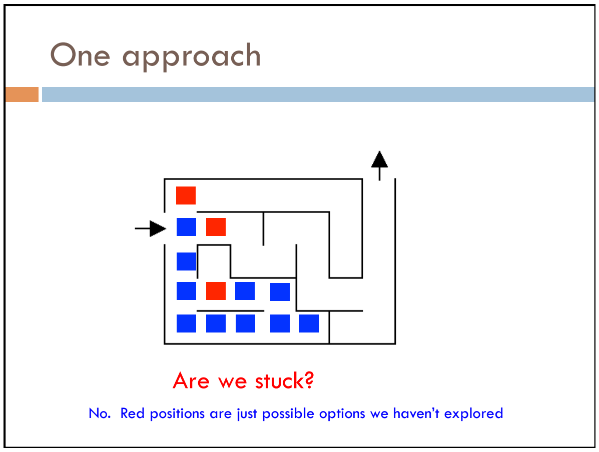


Keep exploring

One approach



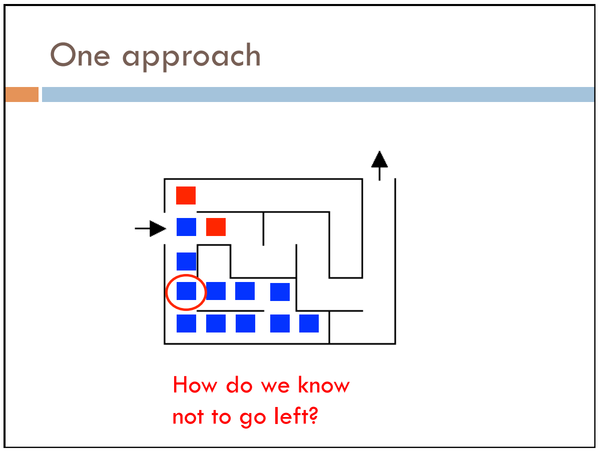
What now?



One approach

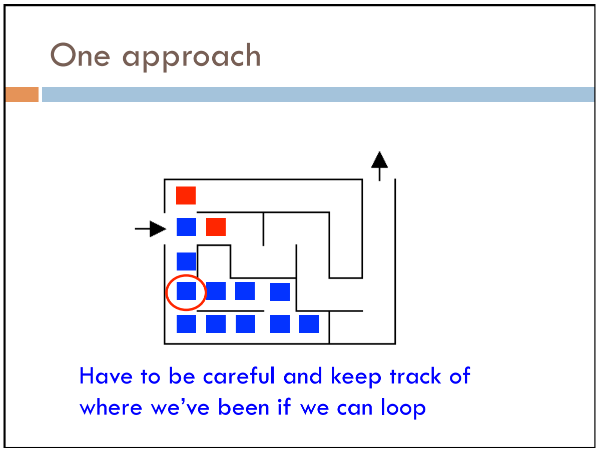
Are we stuck?

No. Red positions are just possible options we haven't explored



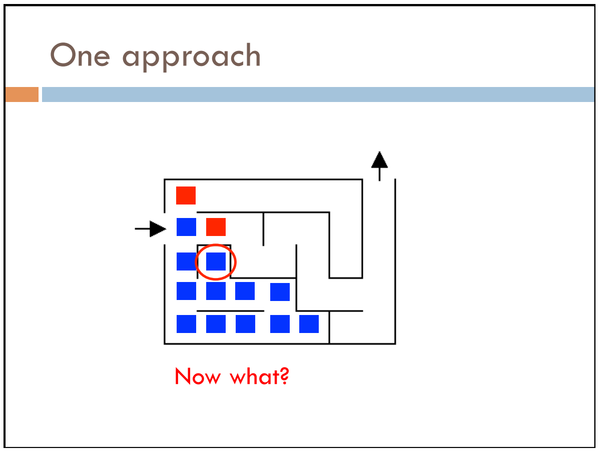
One approach

How do we know not to go left?



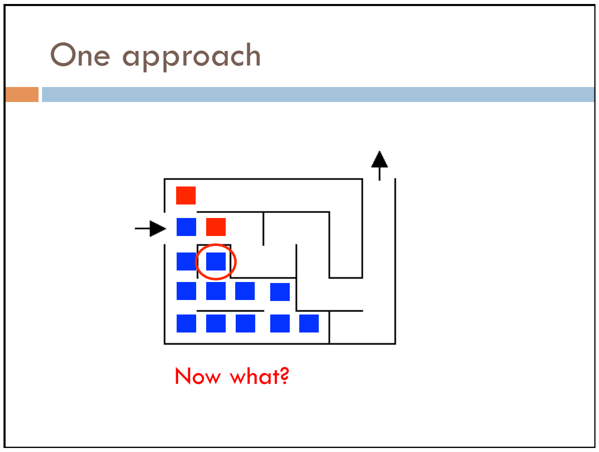
One approach

Have to be careful and keep track of where we've been if we can loop



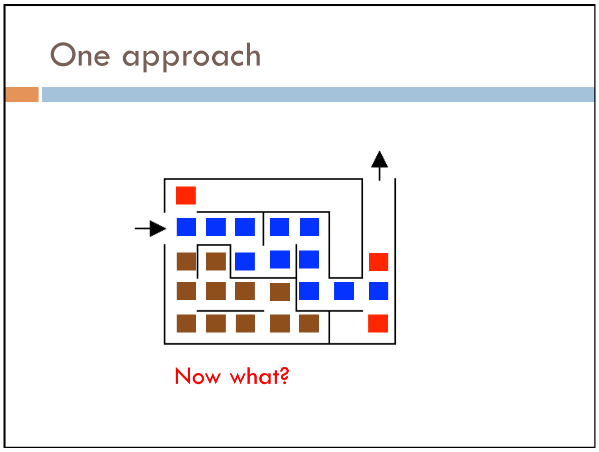
One approach

Now what?



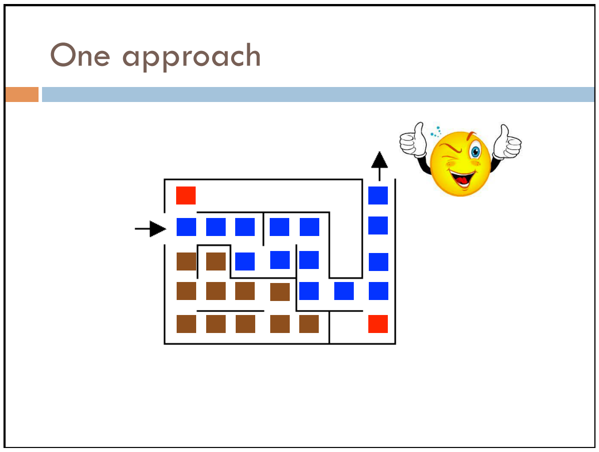
One approach

Now what?

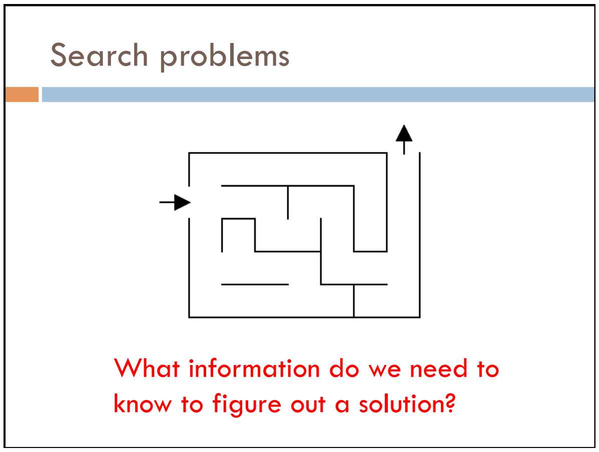


One approach

Now what?

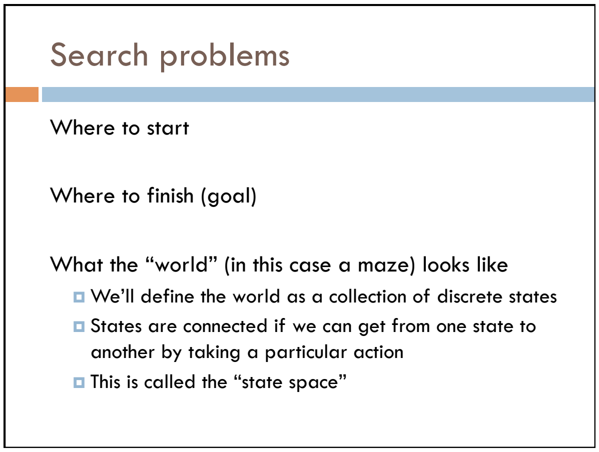


One approach



Search problems

What information do we need to know to figure out a solution?



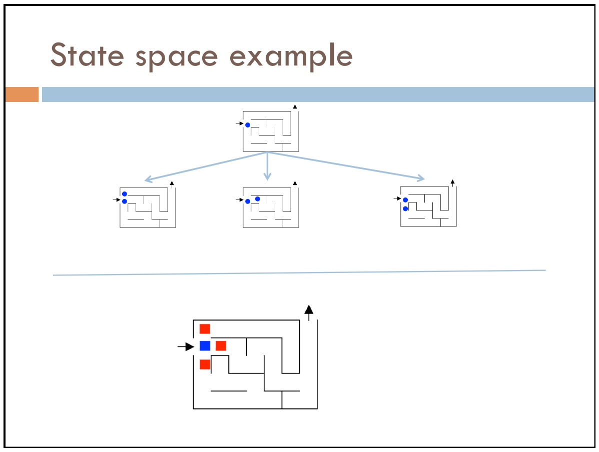
Search problems

Where to start

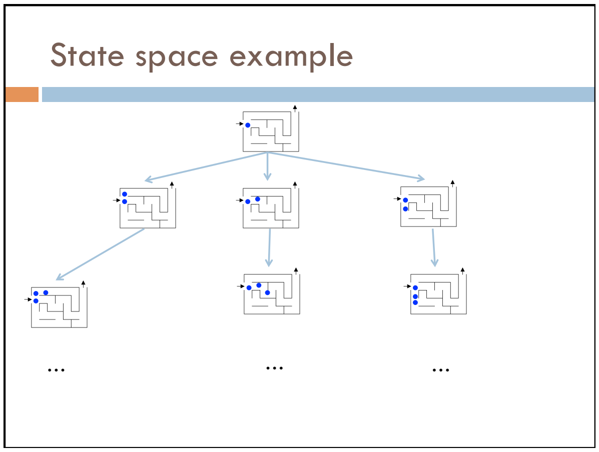
Where to finish (goal)

What the "world" (in this case a maze) looks like

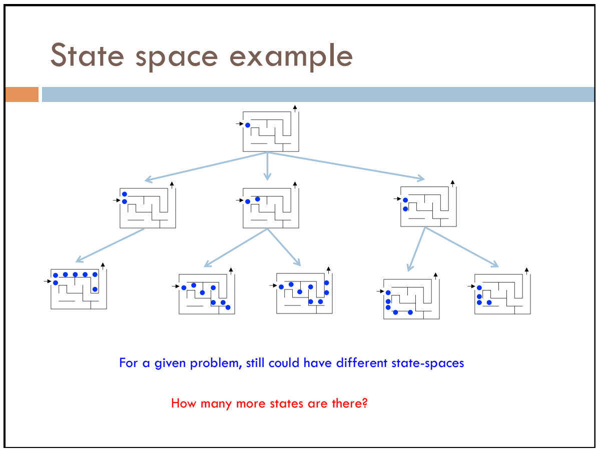
- We'll define the world as a collection of discrete states
- States are connected if we can get from one state to another by taking a particular action
- This is called the "state space"



State space example



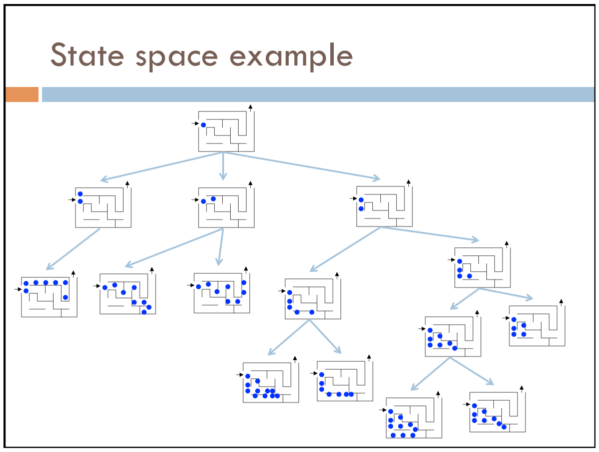
State space example



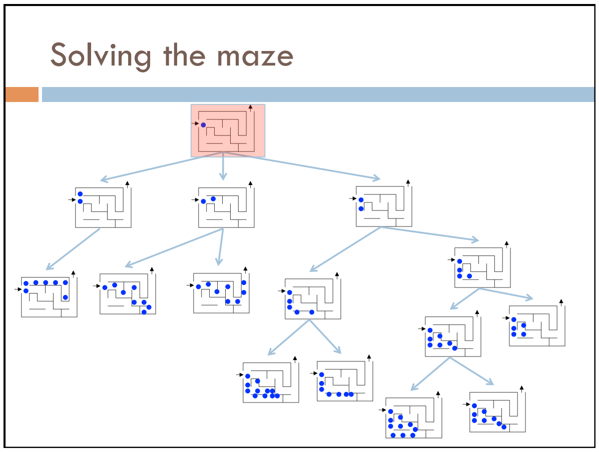
State space example

For a given problem, still could have different state-spaces

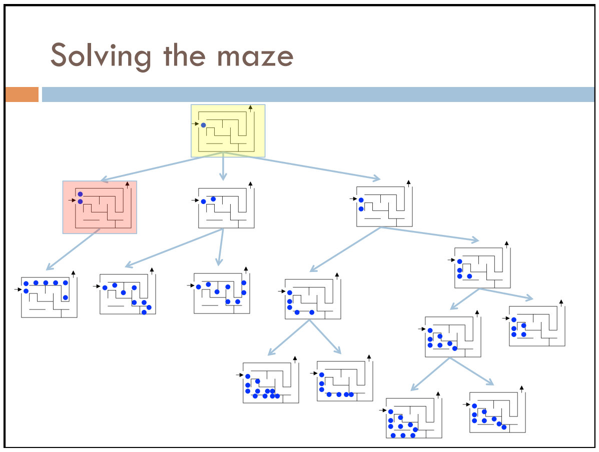
How many more states are there?



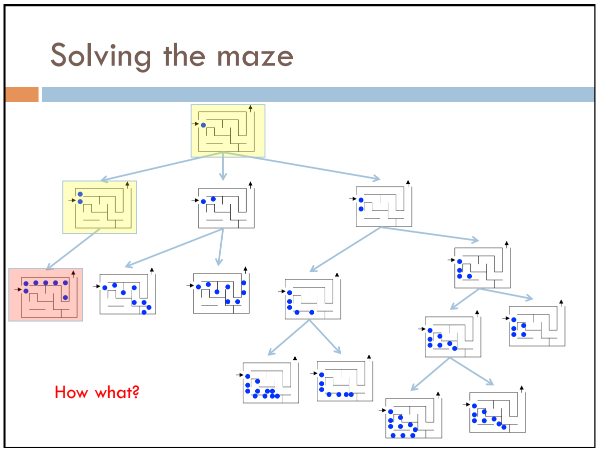
State space example



Solving the maze

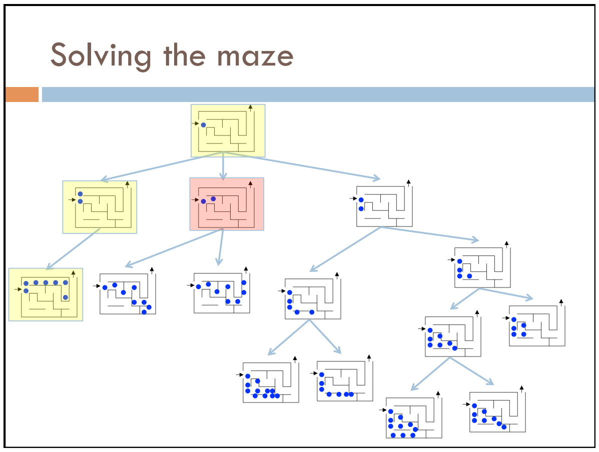


Solving the maze

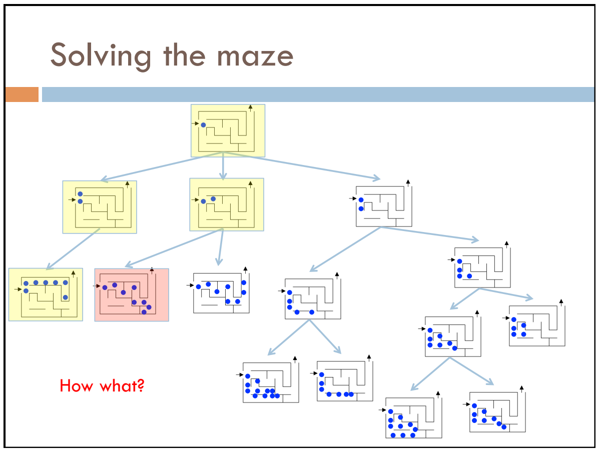


Solving the maze

How what?

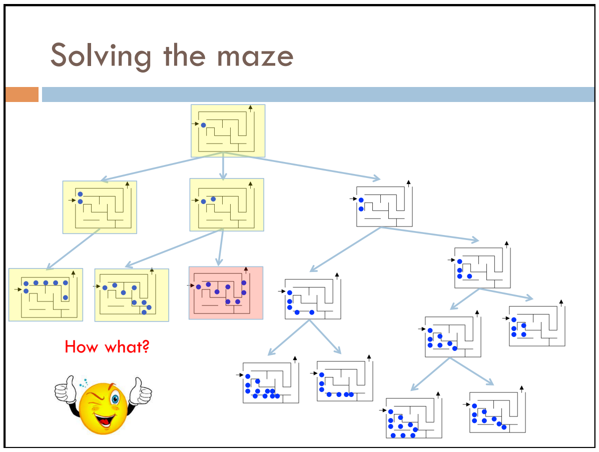


Solving the maze



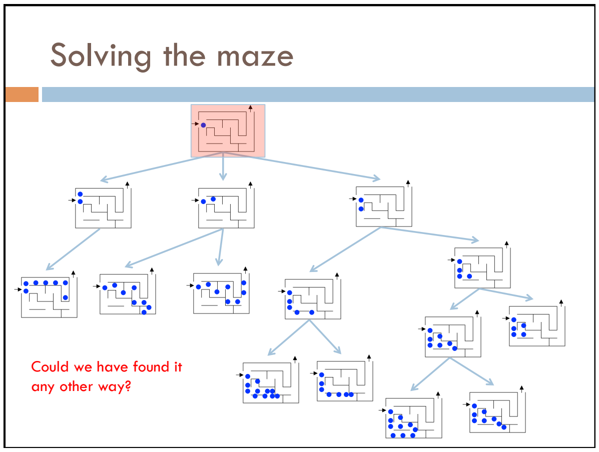
Solving the maze

How what?



Solving the maze

How what?



Solving the maze

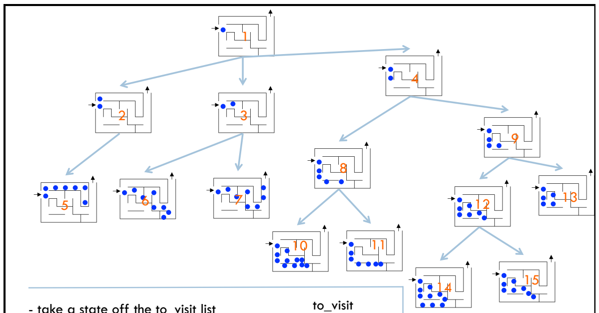
Could we have found it any other way?

Search algorithm

Keep track of a list of states that we *could* visit, we'll call it "to_visit"

General idea:

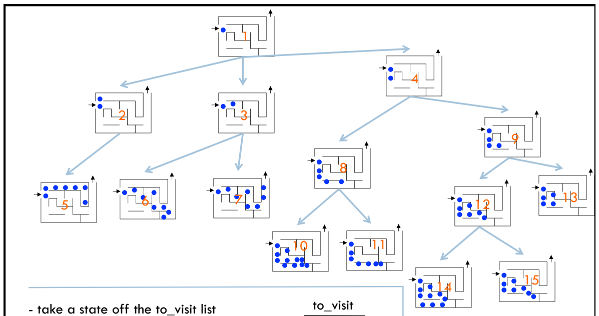
- take a state off the to_visit list
- if it's the goal state
  - we're done!
- if it's not the goal state
  - Add all of the successive states to the to_visit list
  - repeat



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

How do we start?

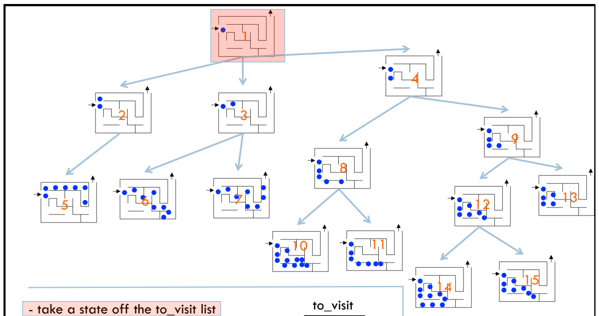


- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

1

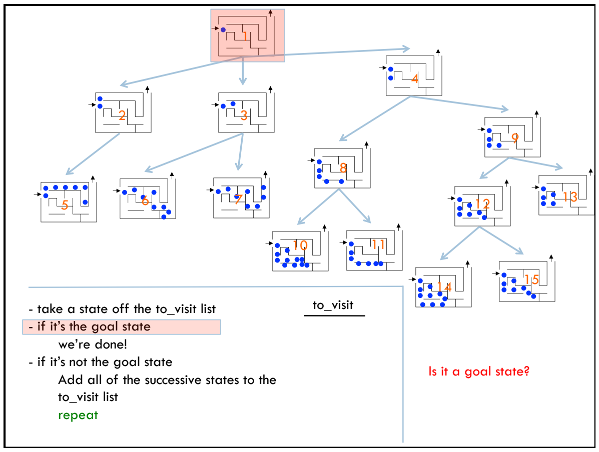
Add start not to to_visit



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

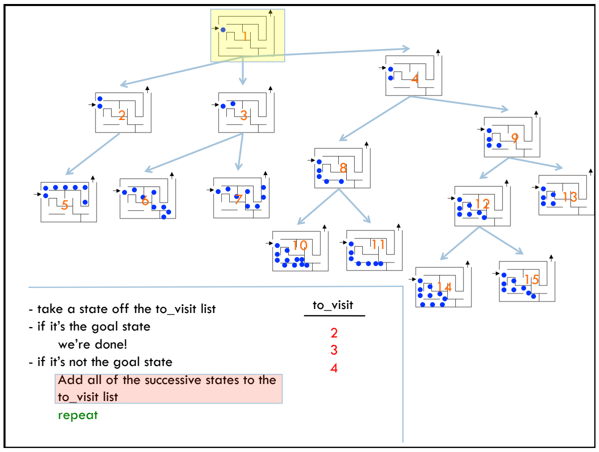
Add start not to to_visit



- take a state off the to_visit list
- if it's the goal state
    we're done!
- if it's not the goal state
    Add all of the successive states to the to_visit list
    repeat

to_visit

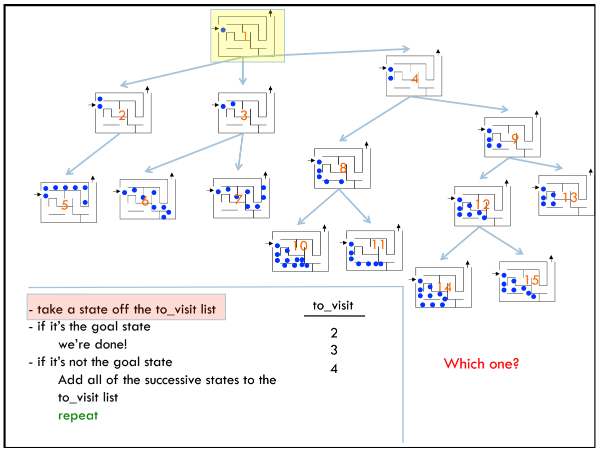
Is it a goal state?



- take a state off the to_visit list
- if it's the goal state
    we're done!
- if it's not the goal state
    Add all of the successive states to the to_visit list
    repeat

to_visit

2
3
4

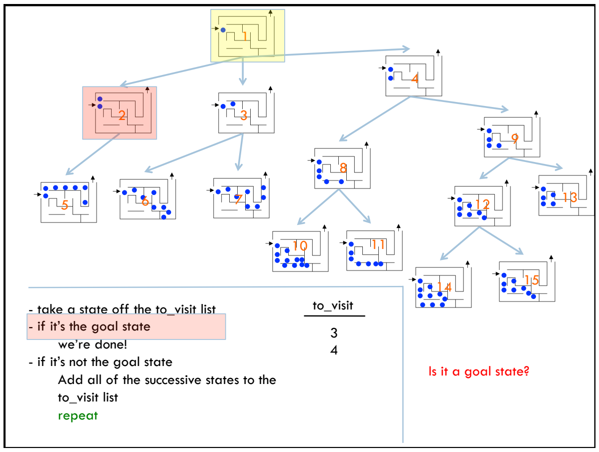


- take a state off the to_visit list
- if it's the goal state
    we're done!
- if it's not the goal state
    Add all of the successive states to the to_visit list
    repeat

to_visit

2
3
4

Which one?

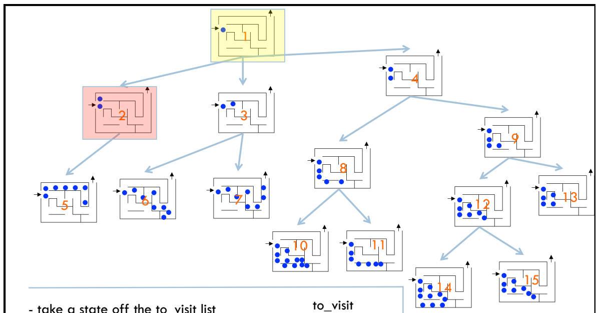


- take a state off the to_visit list
- if it's the goal state
    we're done!
- if it's not the goal state
    Add all of the successive states to the to_visit list
    repeat

to_visit

3
4

Is it a goal state?

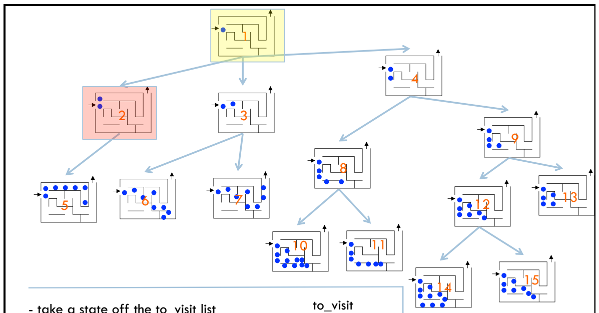


- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

3
4

Where should we add them in the list?

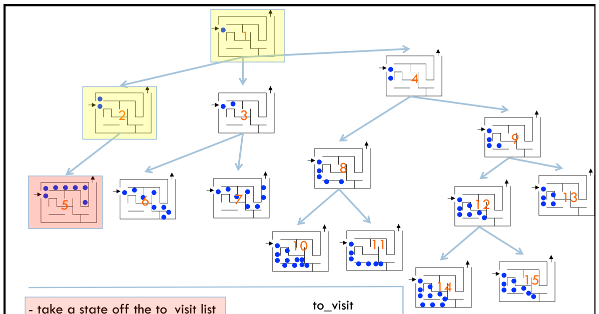


- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

5
3
4

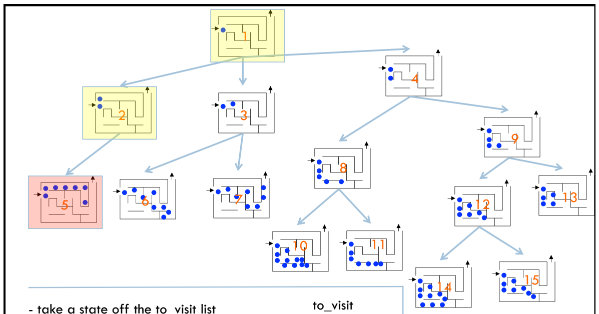
Let's add them to the front



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

3
4

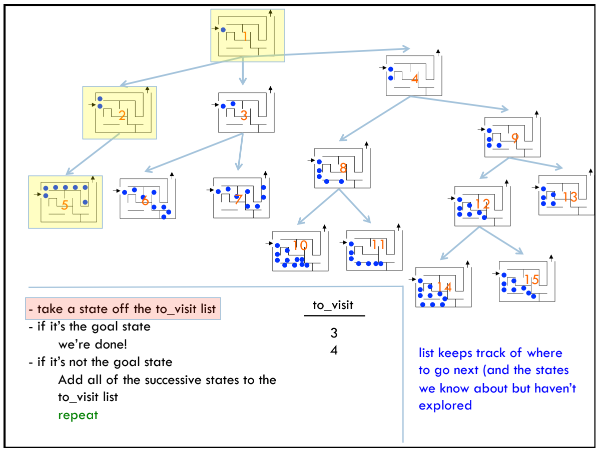


- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

3
4

What do we do here?

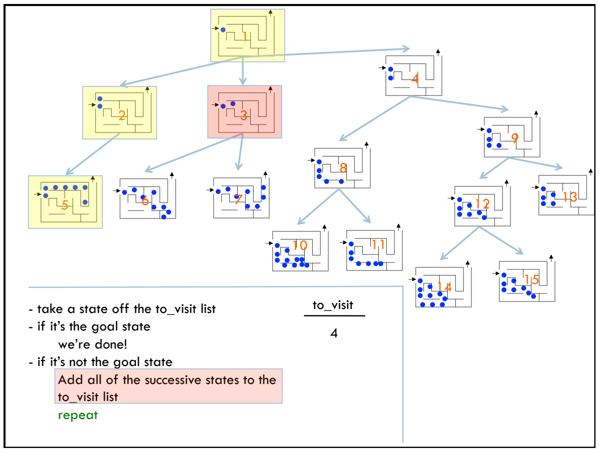


- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

3
4

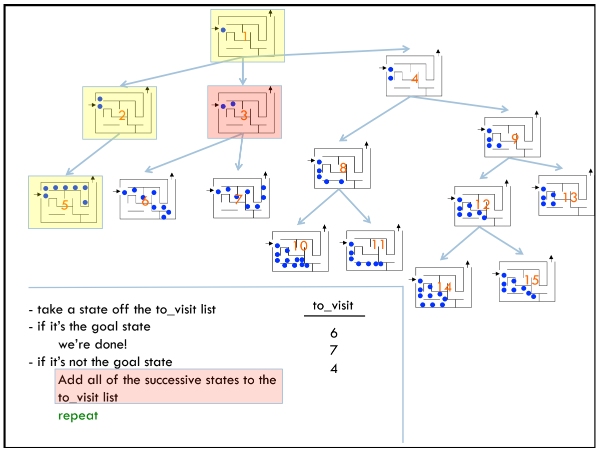
list keeps track of where to go next (and the states we know about but haven't explored)



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

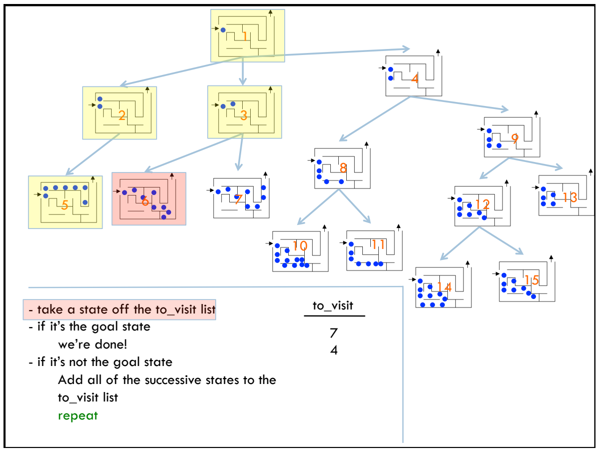
4



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

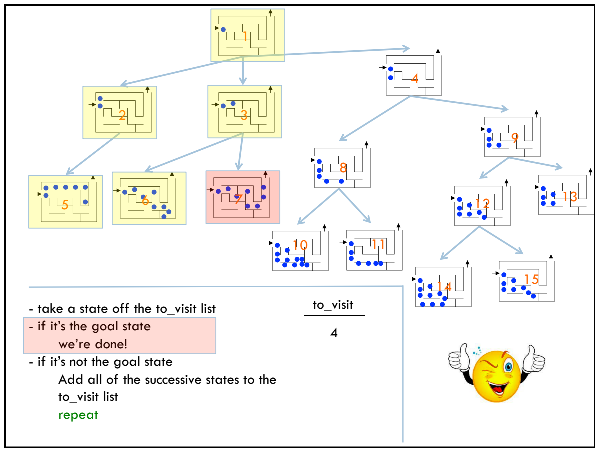
6
7
4



- take a state off the to_visit list
- if it's the goal state
  we're done!
- if it's not the goal state
  Add all of the successive states to the to_visit list
  repeat

to_visit

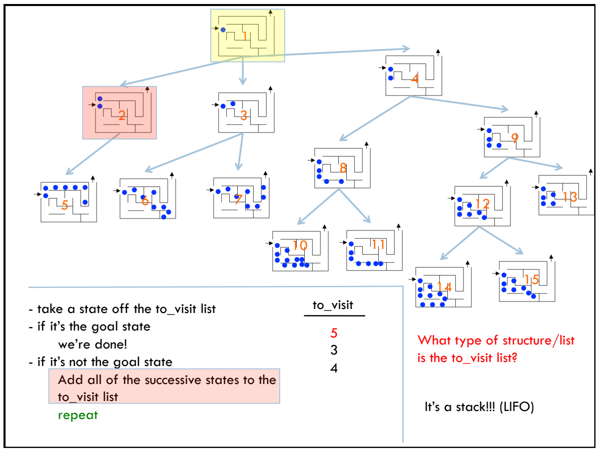
7
4



```
- take a state off the to_visit list
- if it's the goal state
      we're done!
- if it's not the goal state
      Add all of the successive states to the
      to_visit list
      repeat
```

to_visit

4



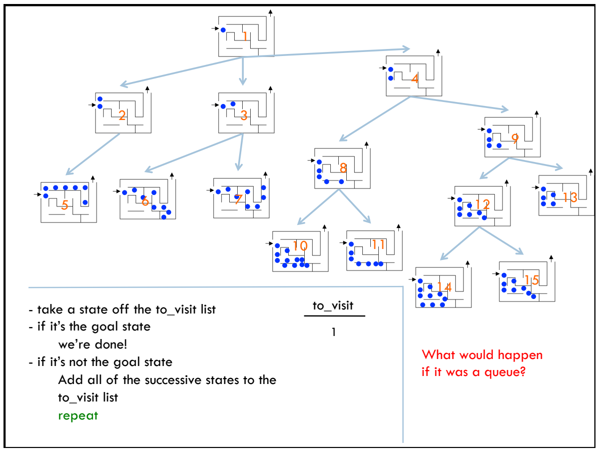
```
- take a state off the to_visit list
- if it's the goal state
      we're done!
- if it's not the goal state
      Add all of the successive states to the
      to_visit list
      repeat
```

to_visit

5
3
4

What type of structure/list is the to_visit list?

It's a stack!!! (LIFO)

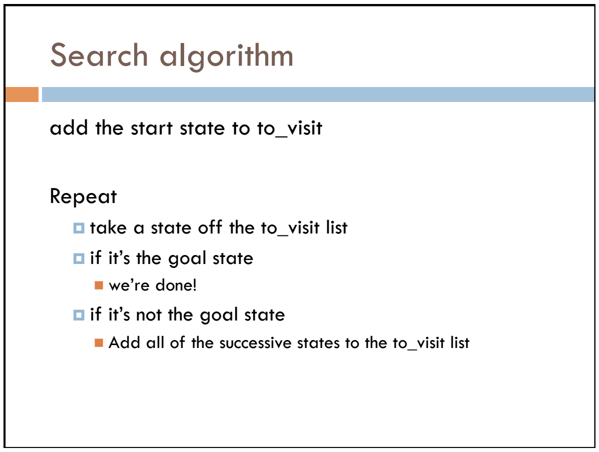


```
- take a state off the to_visit list
- if it's the goal state
      we're done!
- if it's not the goal state
      Add all of the successive states to the
      to_visit list
      repeat
```

to_visit

1

What would happen if it was a queue?

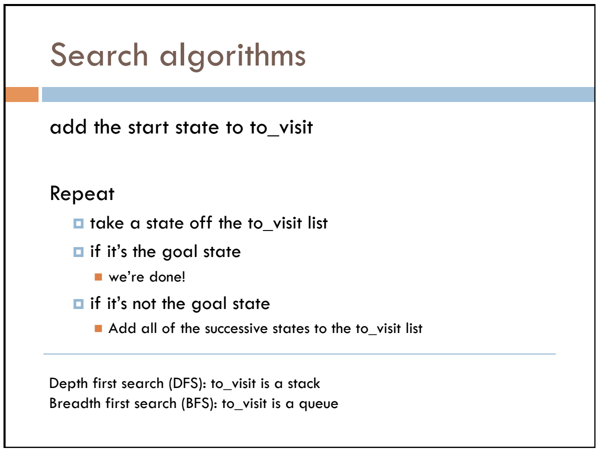


Search algorithm

add the start state to to_visit

Repeat

- take a state off the to_visit list
- if it's the goal state
  - we're done!
- if it's not the goal state
  - Add all of the successive states to the to_visit list



Search algorithms

add the start state to to_visit

Repeat
- take a state off the to_visit list
- if it's the goal state
  - we're done!
- if it's not the goal state
  - Add all of the successive states to the to_visit list

Depth first search (DFS): to_visit is a stack
Breadth first search (BFS): to_visit is a queue