Week 9, Class 1

Stacks of Fun

CS 42: Principles & Practice of Computer Science

Melissa O'Neill

Fall, 2011

Recap

What are objects/classes for?

Recap (contd.)

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def __eq__(self, rhs):
        return self.x == rhs.x and self.y == rhs.y
```

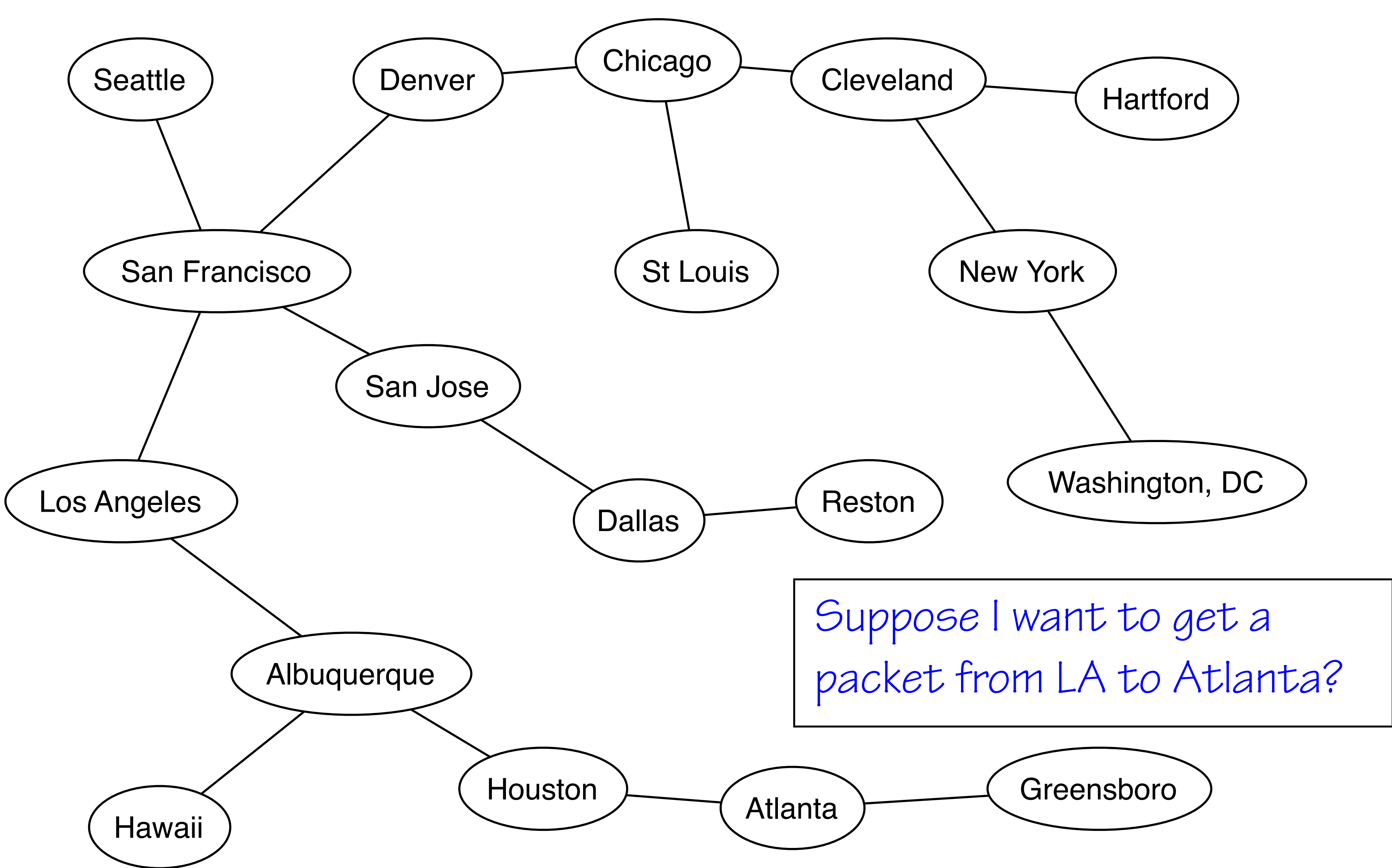
Recap (contd.)

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def __eq__(self, rhs):
        return type(self) == type(rhs) and
                    self.x == rhs.x and self.y == rhs.y
```

Recap (contd.)

```
def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == (0.0, 0.0):
       print "Equally valid points!"
    else:
       print "They're not equal!"
```

Teh Interwebz (Remember Trees?)



Suppose I want to get a packet from LA to Atlanta?

Recursion to the Rescue

```
def traverse(node):
     print node.name()
     for neighbor in node.connections()
          traverse(neighbor)
```

Seattle
Denver
Chicago
Cleveland
Hartford
San Francisco
St Louis
New York
San Jose
Washington, DC
Los Angeles
Dallas
Reston
Albuquerque
Houston
Atlanta
Greensboro
Hawaii

But I want the *path* from LA to Atlanta!

How much space?

No recursion allowed?

Stacks!

Stacks

What are the fundamental operations?

Does it matter *how* the stack works?

Rocks stack, and stacks rock!

Stack Interface

The key operations

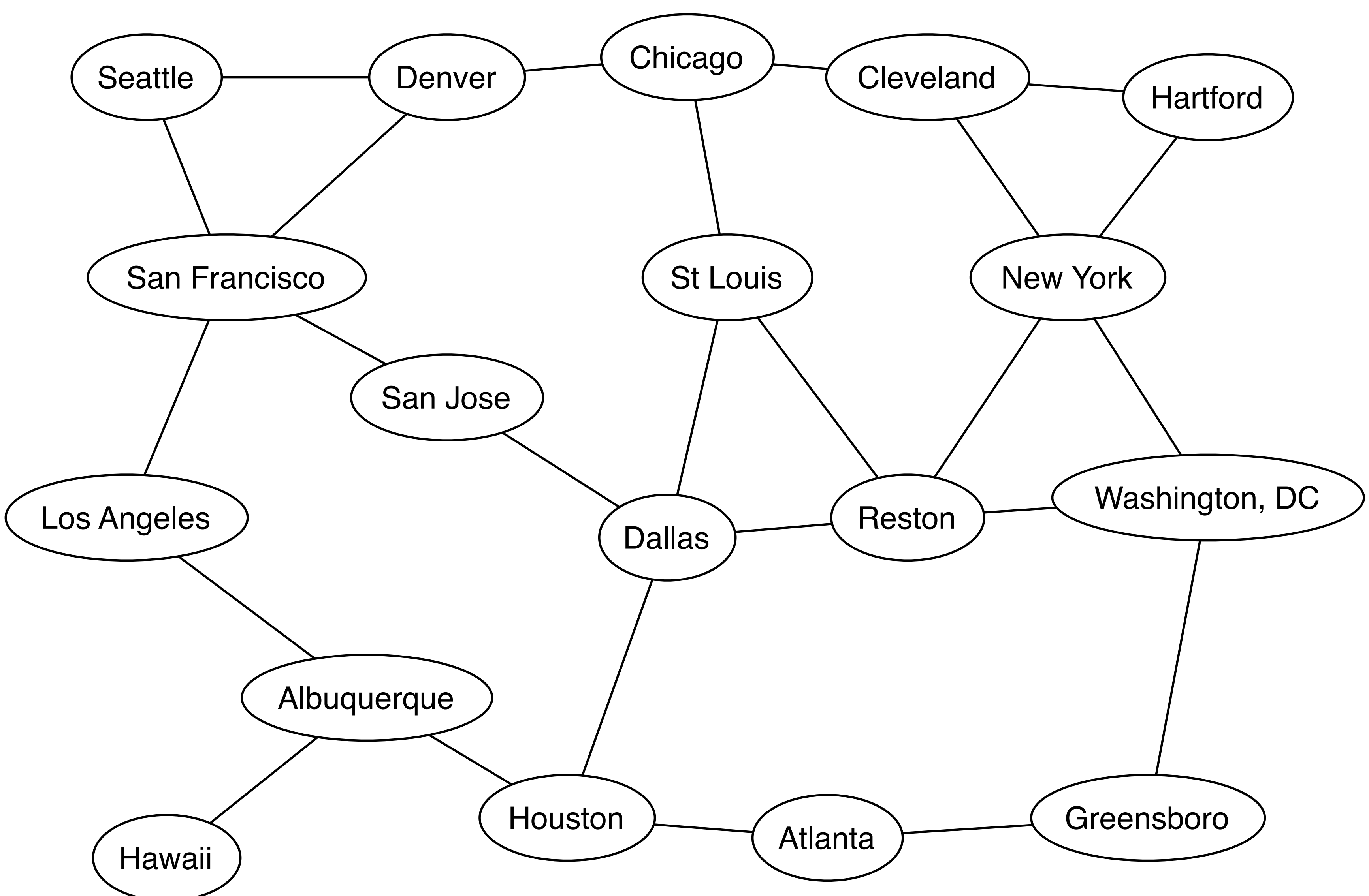
- push
- pop
- empty

How can we use the stack interface to eliminate recursion in our Internet example?

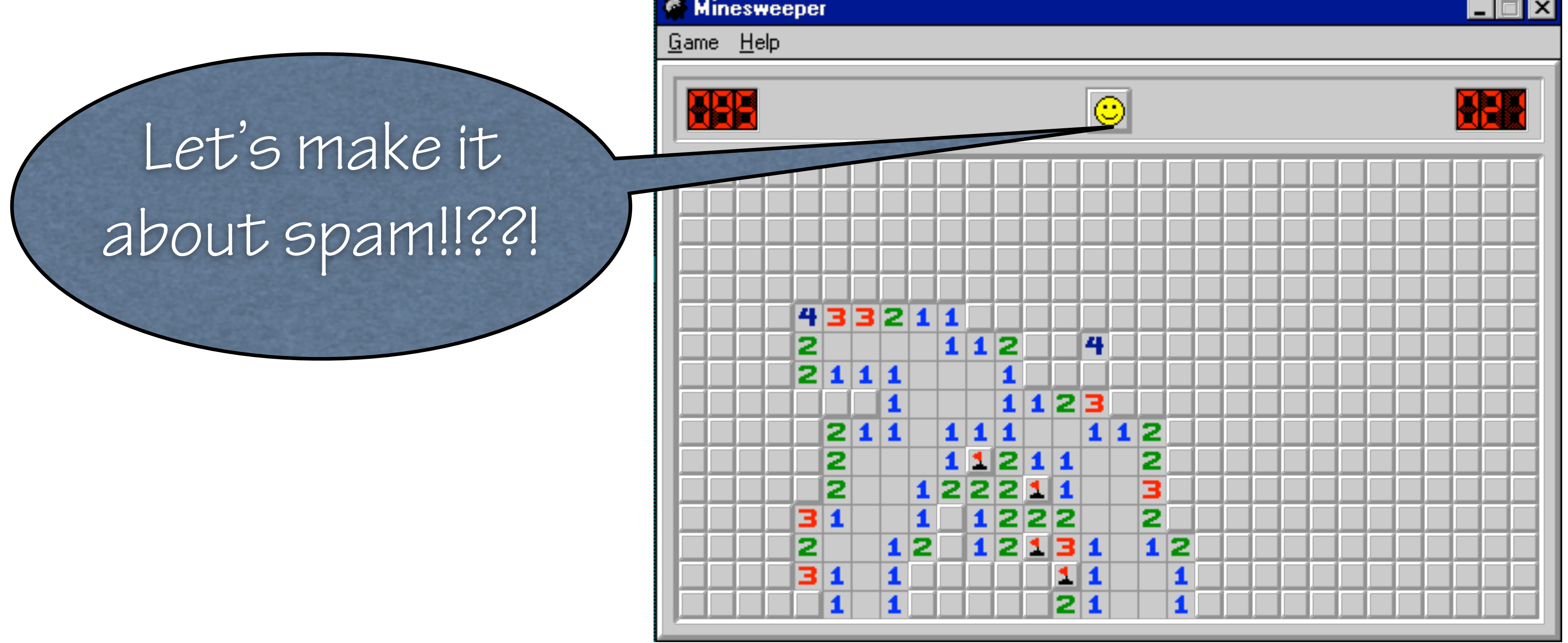
Interface + Behavior (but not saying *how*) =

- **ABSTRACT DATA TYPE**

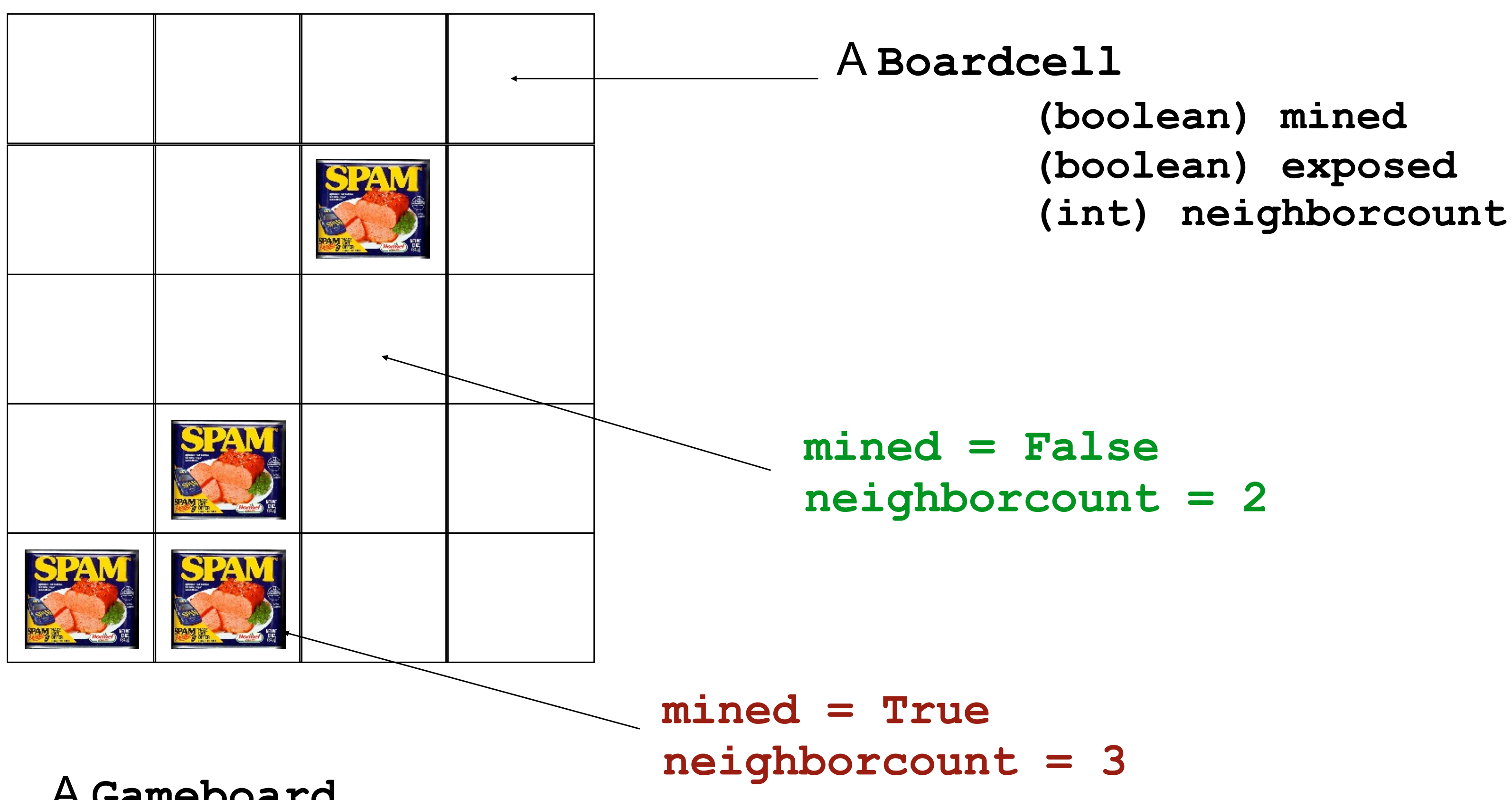
More Realisitic



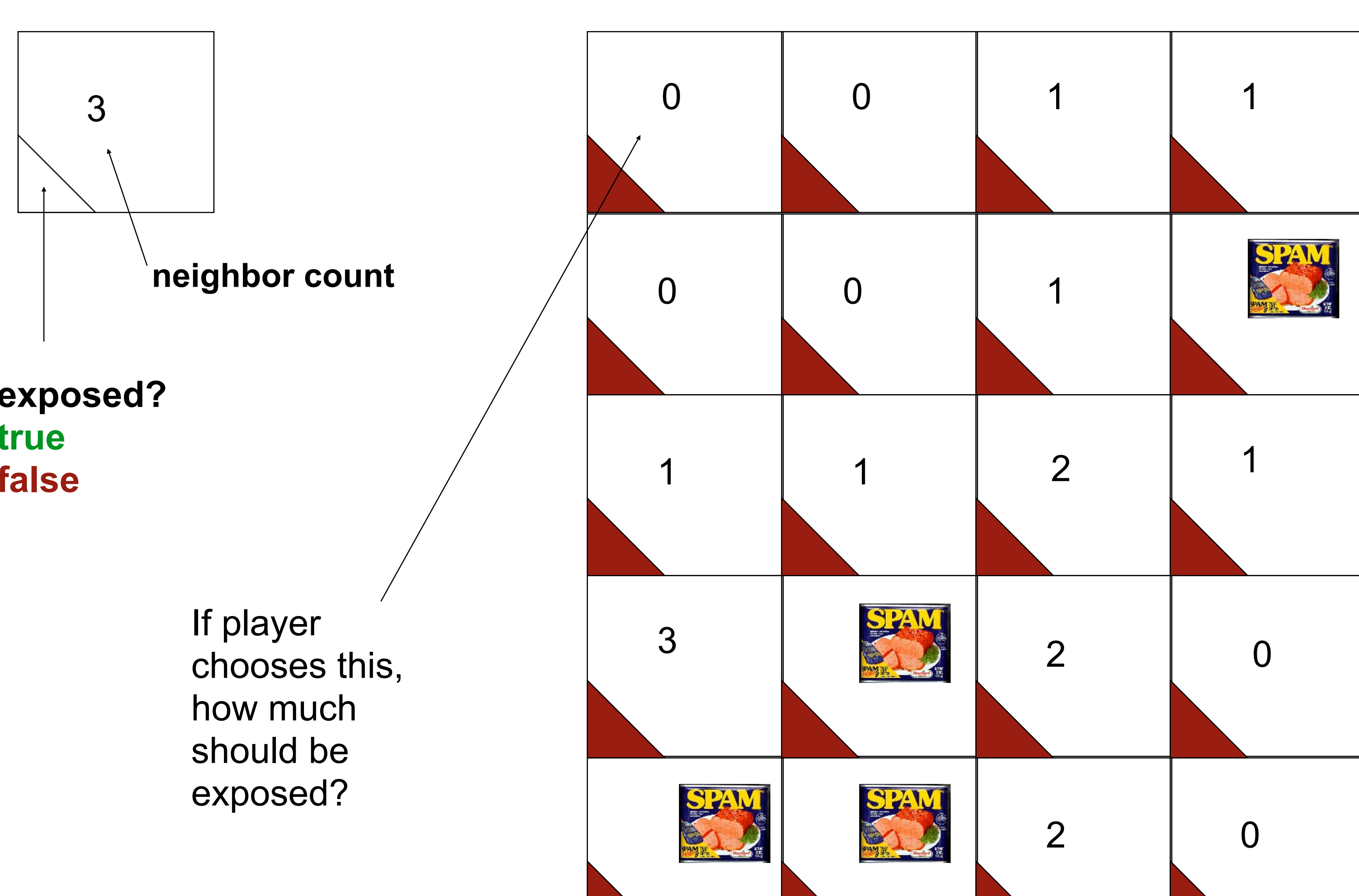
The Homework — A Demo



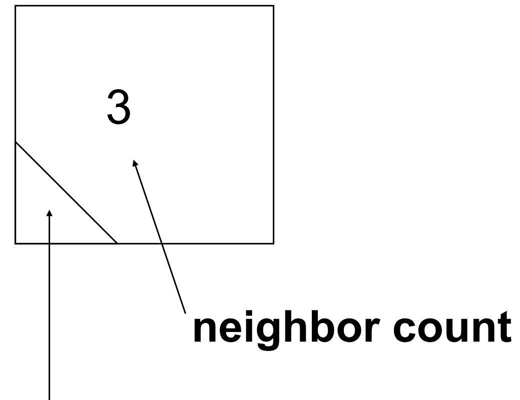
Spamsweeper!



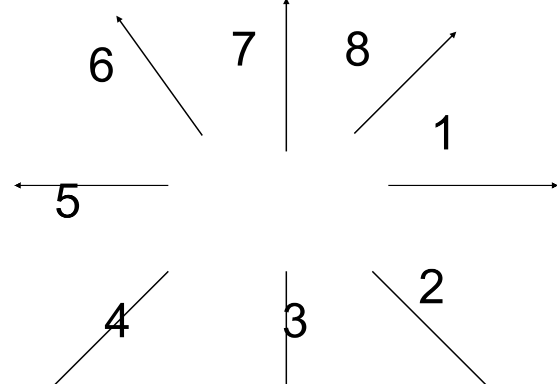
Automatic Exposure



Automatic Exposure



exposed?
true
false



0	0	1	1
0	0	1	SPAM
1	1	2	1
3	SPAM	2	0
SPAM	SPAM	2	0

Stacks, Implementation

Ideas?

Stacks, in pictures with lists

ADT = Many possible implementations

```
S = Stack()
S.push("Sleep")
S.push("IWP")
S.push("CS42")
S.push("Dinner!")
S.pop()
```

after these 6 lines of code...

name: S
class: Stack

name: data

"Sleep"	"IWP"	"CS42"	"Dinner!"
0	1	2	3

topIndex 2

name: topIndex
type: int

Stacks, in pictures with linked lists

ADT = Many possible implementations

```
S = Stack()
S.push("Sleep")
S.push("IWP")
S.push("CS42")
S.push("Dinner!")
S.pop()
```

after these 6 lines of code...

name: S
class: Stack

name: top
type: StackCell

n: data | n: next
"CS42" |

name: ?
type: StackCell

n: data | n: next
"IWP" |

name: ?
type: StackCell

n: data | n: next
"Sleep" | None

name: <none>
class: StackCell

n: data | n: next
"Dinner!" |

Stacks,
in code

```
class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
```

Stacks, in code

```
class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
```

"Quiz"

Complete the linked-list implementation of a **`Stack`**.

```
class Stack(object):
    def pop(self):
```

write pop()

```
    def push(self, data):
```

write push(Object data)

```
s
```

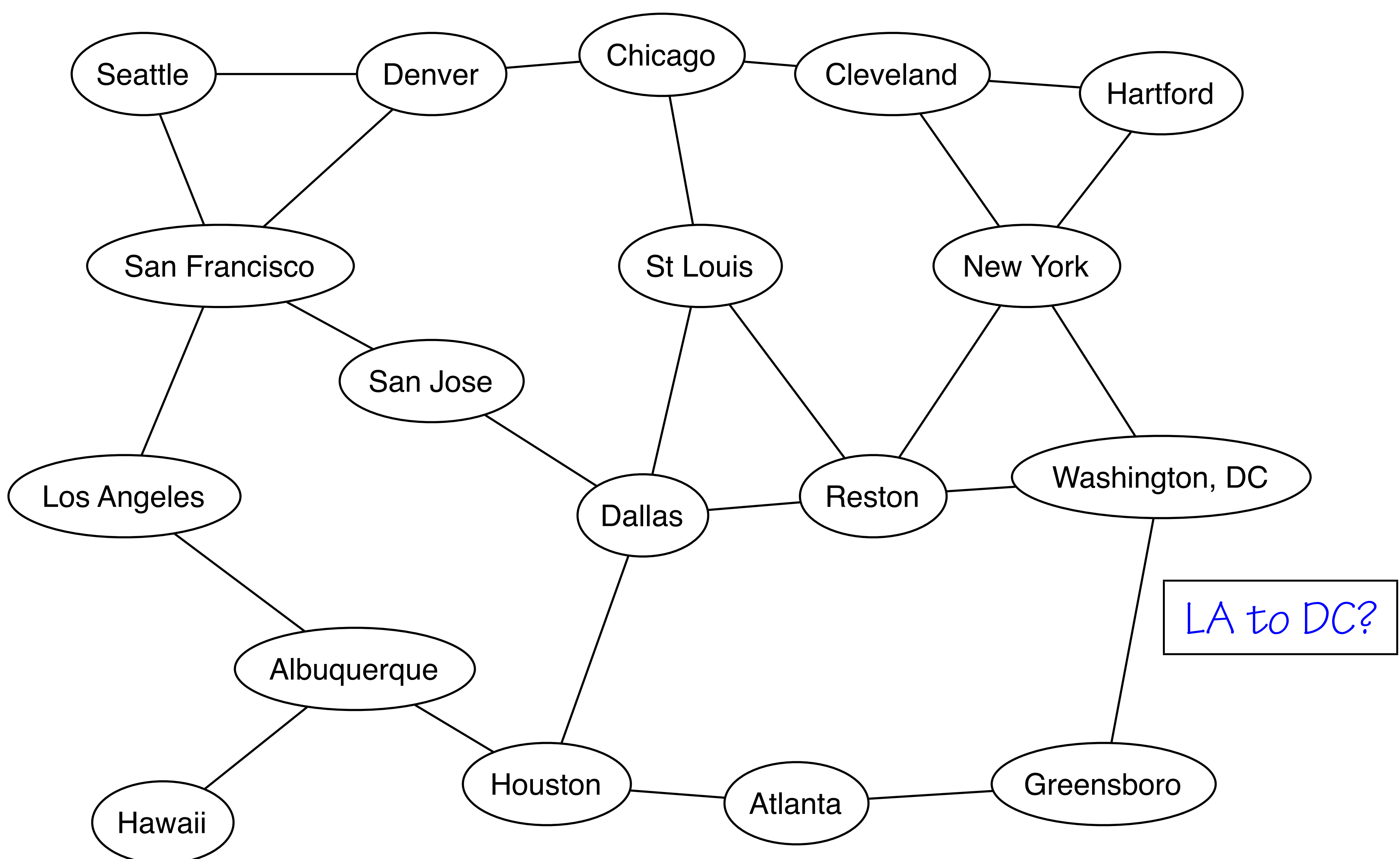
extra: str

```
def __str__(self):

}
```

```
    def empty(self):
```

write empty()

Interwebs, Revisited



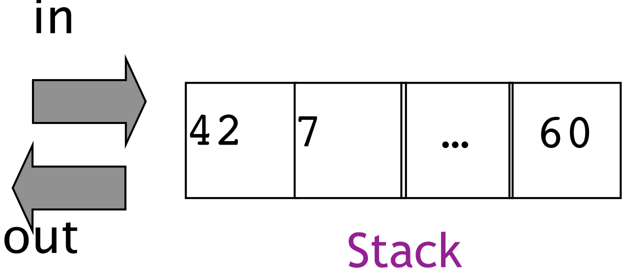
Stack vs. Queue

Stack Contract

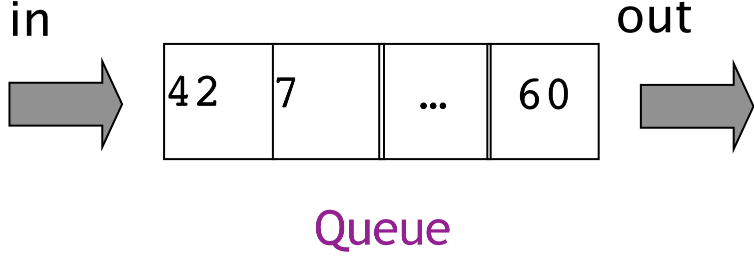
```
(object) pop()
push(data)
(bool) empty()
```

Queue Contract

```
(object) dequeue()
enqueue(data)
(bool) empty()
```



Stack



Queue

How will switching to a Queue change our search procedure?

Queues, Implementation

Queues

Three ways in the assignment

- Using Python lists
- Using Python's deque class (provided by us!)
- Hand-implemented with a `QCell` class.

Queue vs Stack

Can we just reuse our design for the stack class?

Stacks, in code

```
class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
```

Queues, in code

```
class QCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Queue(object):
    def __init__(self):
        self.front = None
```

Queues, in code

```
class QCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Queue(object):
    def __init__(self):
        self.front = None
        self.back  = None
```

"Q"uiz

Draw the results of each of these lines of code…

```
Bert = Queue()
Bert.enqueue("U")   (1)
Bert.enqueue("C")   (2)
Bert.dequeue()      (3)
Bert.dequeue()      (4)
```

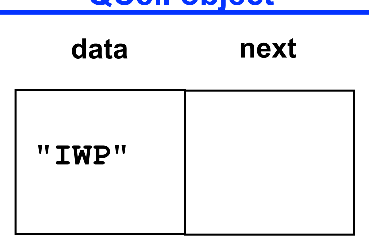
back (QCell) front (QCell)

back (QCell) front (QCell)

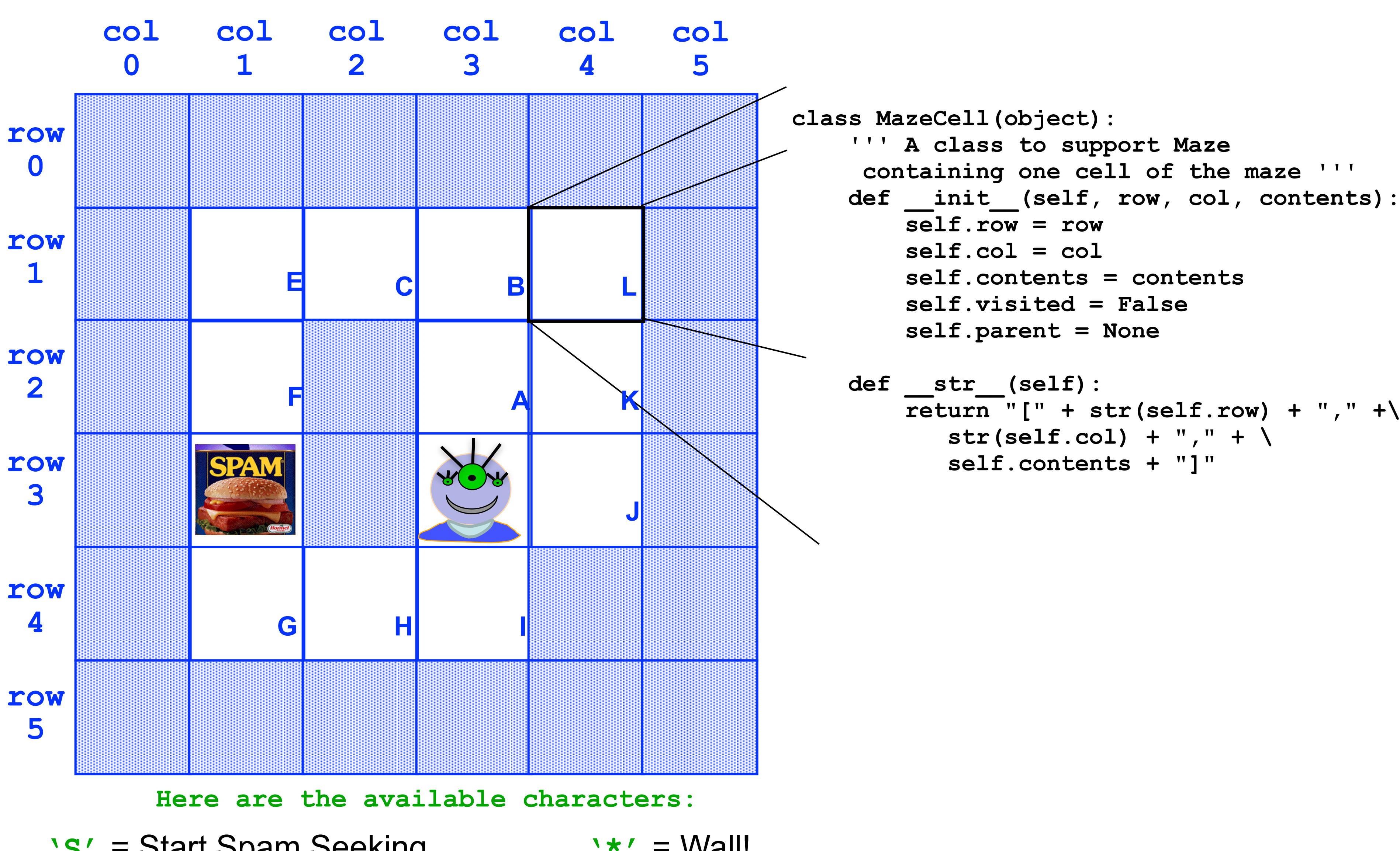
back (QCell) front (QCell)

back (QCell) front (QCell)

Here is a picture of a QCell…



Search for Spam!




```
class MazeCell(object):
    ''' A class to support Maze
     containing one cell of the maze '''
    def __init__(self, row, col, contents):
        self.row = row
        self.col = col
        self.contents = contents
        self.visited = False
        self.parent = None

    def __str__(self):
        return "[" + str(self.row) + "," +\
           str(self.col) + "," + \
           self.contents + "]"
```

Here are the available characters:

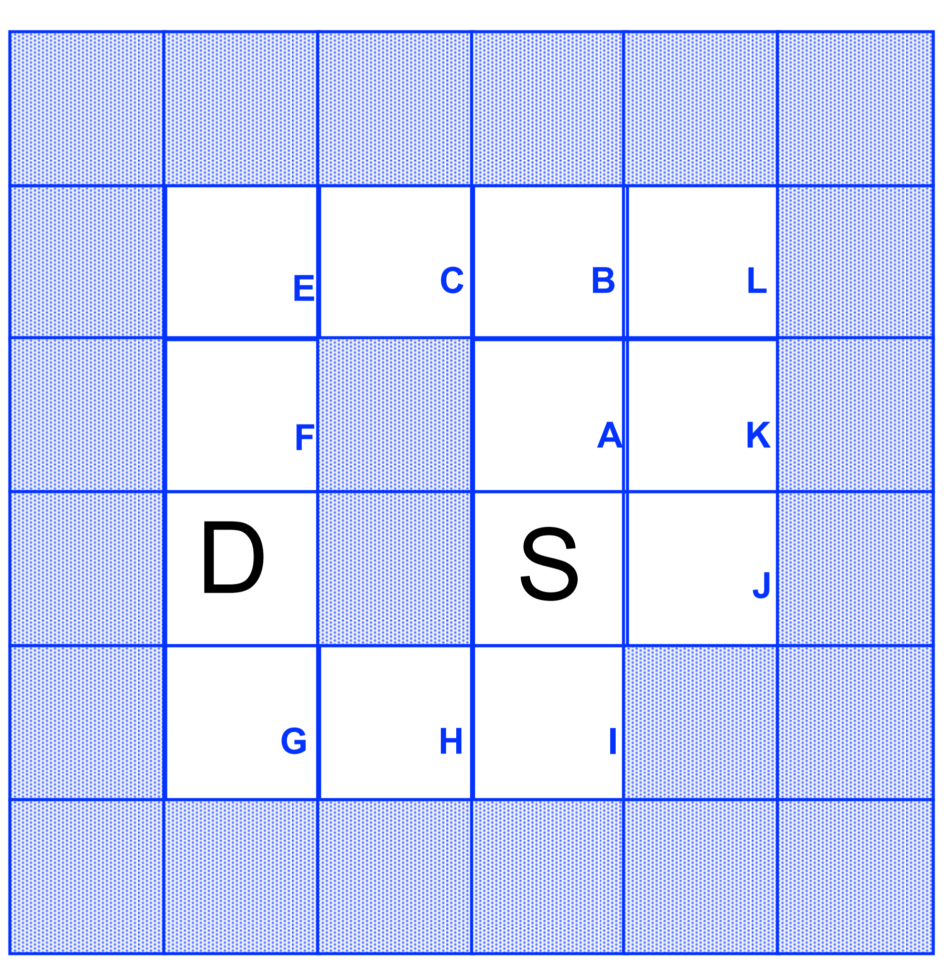
'S' = Start Spam Seeking

'D' = Delectible Dinner Destination

'*' = Wall!

' ' = Open *Space*

Depth-first search (DFS)



Pseudocode

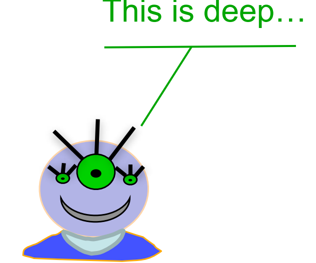
Create an empty Stack
mark starting MazeCell as visited
push starting MazeCell onto our Stack

while (the stack's not empty)
    current = pop the stack
    for each of current's neighbors
        if (it's not visited or wall)
            mark it (neighbor) as visited
            set its parent to current MazeCell
            push it onto the stack

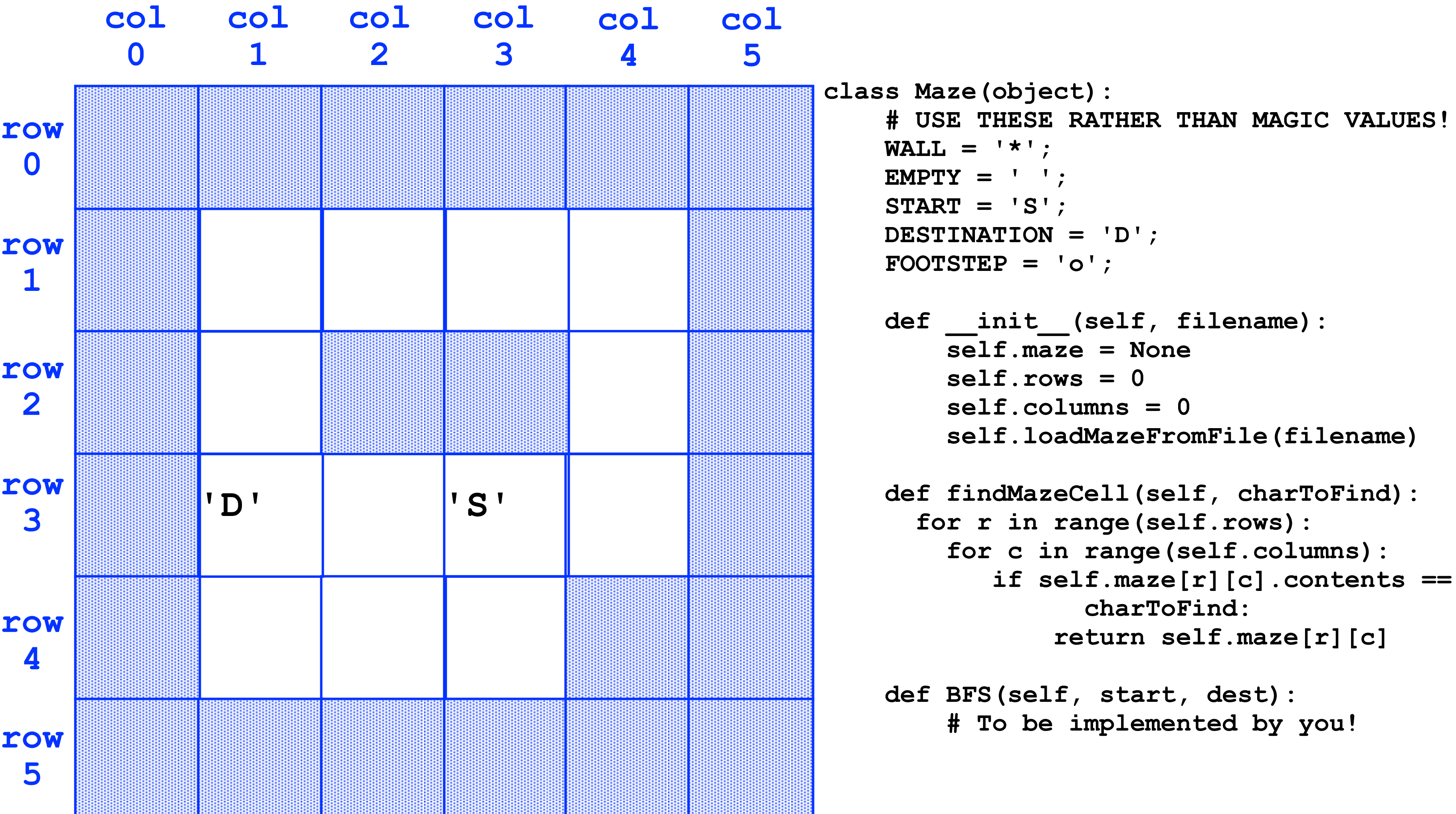
4
3
2
1

'S' = Start Spam Seeking
'D' = Delectible Dinner Destination

This is deep…



Amazed by details?



Hw8, part 3: implement ***breadth-first*** search