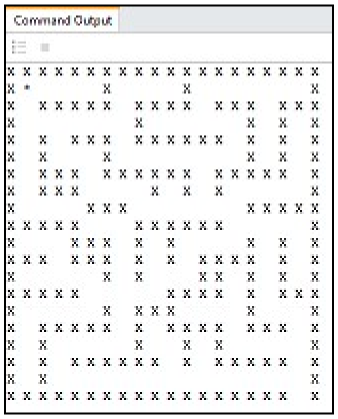
Breadth-first search in a maze



```
x x x x x x x x x x x x x x x x x x x x
x *         x         x               x
x   x x x x x   x x x x   x x x   x x x
x               x             x   x   x
x   x   x x x   x x x x x x   x   x   x
x   x       x                 x   x   x
x   x x x   x x x x x x   x x x x x   x
x   x x x         x   x   x           x
x         x x x               x x x x x
x x x x x       x x x x x x           x
x       x x x   x   x         x   x   x
x x x   x x x   x   x   x x x x   x   x
x           x   x       x x   x   x   x
x x x x x           x x x x   x   x x x
x           x   x x x         x       x
x   x x x x x   x   x x x x   x x x   x
x   x           x     x   x           x
x   x   x x x x x x   x   x x x x x   x
x   x                                 x
x x x x x x x x x x x x x x x x x x   x
```

The goal of this assignment is to find your way out of a maze using breadth-first search.

Here is a basic algorithm for BFS in an unweighted graph:

```
bfs(start vertex, goal vertex)
make frontier an empty queue
enqueue start onto frontier
until frontier is empty
    dequeue parent off frontier
    for each undiscovered child of parent
        enqueue child onto frontier
        stop if child is the goal
```

To apply this algorithm to a maze, think of grid locations as vertices. The "children" of a grid location are the open cells adjacent to it in the four cardinal directions.

A Maze class and an Agent class have been started for you below. Once you complete the indicated methods, you will be able to watch the agent navigate the maze. Adjust the height of your console window to match the maze height, so that each display appears in the same place.

Algorithm notes:

- You will need to keep track of the sequence of moves by which you discovered each location. The pseudocode above doesn't address this, so the way you do it is up to you.
- You will also need to decide how you want to represent moves.

Python notes:

- If you want to put Maze objects into sets or use them as dictionary keys, remember to overload the `__hash__` and `__eq__` and `__ne__` methods of the Maze class.

```
import time

class Maze(object):
    """A pathfinding problem."""

    def __init__(self, grid, location):
        """Instances differ by their current agent locations."""
        self.grid = grid
        self.location = location

    def display(self):
        """Print the maze, marking the current agent location."""
        for r in range(len(self.grid)):
            for c in range(len(self.grid[r])):
                if (r, c) == self.location:
                    print '*',
                else:
                    print self.grid[r][c],
            print
        print

    def moves(self):
        """Return a list of possible moves given the current agent location."""
        # YOU FILL THIS IN

    def neighbor(self, move):
        """Return another Maze instance with a move made."""
        # YOU FILL THIS IN

class Agent(object):
    """Knows how to find the exit to a maze with BFS."""

    def bfs(self, maze, goal):
        """Return an ordered list of moves to get the maze to match the goal."""
        # YOU FILL THIS IN

def main():
    """Create a maze, solve it with BFS, and console-animate."""

    grid = ["XXXXXXXXXXXXXXXXXXXX",
            "X     X    X       X",
            "X XXXXX XXXX XXX XXX",
            "X       X      X X X",
            "X X XXX XXXXXX X X X",
            "X X   X        X X X",
            "X XXX XXXXXX XXXXX X",
            "X XXX    X X X     X",
            "X    XXX       XXXXX",
            "XXXXX   XXXXXX     X",
            "X   XXX X X    X X X",
            "XXX XXX X X XXXX X X",
            "X     X X   XX X X X",
            "XXXXX     XXXX X XXX",
            "X     X XXX    X   X",
            "X XXXXX X XXXX XXX X",
            "X X     X  X X     X",
            "X X XXXXXX X XXXXX X",
            "X X                X",
            "XXXXXXXXXXXXXXXXXX X"]

    maze = Maze(grid, (1,1))
    maze.display()

    agent = Agent()
    goal = Maze(grid, (19,18))
    path = agent.bfs(maze, goal)

    while path:
        move = path.pop(0)
        maze = maze.neighbor(move)
        time.sleep(0.25)
        maze.display()

if __name__ == '__main__':
    main()
```