Project: Solve a Maze
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/maze/maze.pdf

---

The Greek hero Theseus searched a maze to find and kill the Minotaur.

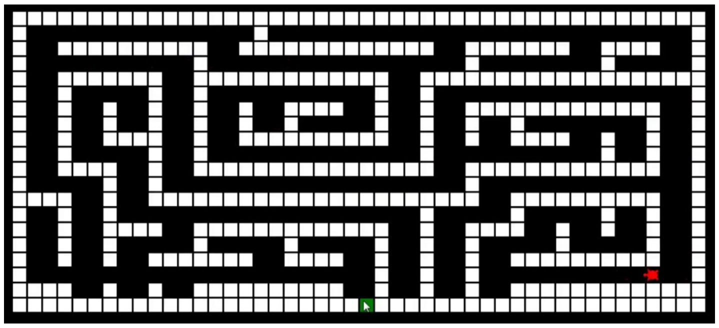
A *maze* is collection of rooms that are connected in a complicated way. A typical maze puzzle includes start and end rooms. The solver is to imagine beginning at the start, and trying to find a path through the connecting rooms that arrives at the end room.

Some questions that arise immediately:

- What pattern or format should we use for mazes?
- How can we create random mazes for testing?
- Can we tell if a solution exists?
- Is there a simple algorithm that usually works?
- Is there an algorithm that always works?
- What algorithm would we use if we could see a map of the whole maze before we start?
- How can we find the shortest path from start to finish?

1 Representing a maze

To simplify our computer approach, we will assume a simple description of a maze. We begin with an $m \times n$ rectangular array of cells. Each cell is either open or closed. For convenience, we might assume that all cells along the boundary are closed. Then every open cell will always have four immediate neighbors, to the north, south, east and west. If a cell and its neighbor are both open, then it is possible to move from one to the other.



The start and end cells are suggested by green and red colors.

If we are given the locations of the start and end cells, then a solution is a path from one to the other which passes successively through neighboring open cells.

To identify a particular cell, we use the typical (i, j) indexing used for matrices. The (closed) cell in the extreme upper left will be designated as (0,0), while the (closed) cell in the extreme lower right is (m-1,n-1).

2 The random mouse

Start with a simple-minded solution algorithm, called the random mouse. The idea is simple to program:

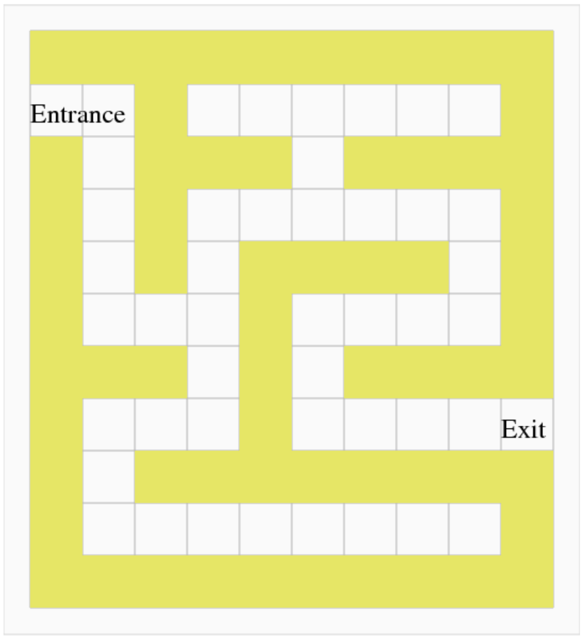
```
start at the designated beginning cell;
while ( True ):
  count your open neighbors.
  select one at random and move there;
  if you have reached the end cell:
    stop with success
```

This is a very stupid algorithm, but it is guaranteed to find the solution (if there is one!), and it helps you get used to calculating in a maze.

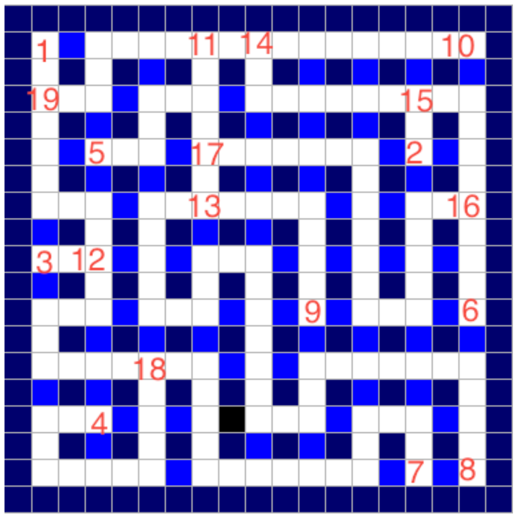
However, even on the simplest maze this can take a disastrously long time to finish. To show this, consider a maze which is simply an $1 \times n$ strip of cells; start the mouse at one end and wait for it to reach the other. As the value of n increases, what happens to the typical number of steps required to solve the maze?

3 The right hand rule

A famous teachnique for dealing with a maze is known as the “right hand rule”. It suggests that, to get from the start to the end, you simply walk forward with your right hand always touching the wall. Whenever you come to a junction, turn to the right, so that you keep your hand on the wall. You can see that in the yellow maze below, starting at the entrance, we will reliably reach the exit.



Unfortunately, this rule only works for relatively simple mazes. Consider the blue maze, for which we are to start in room 1, and are asked to reach room 2. Using the right hand rule, we will pass near room 2, but not enter it, and will simply return to room 1. This is because room 2 is part of an interior wall that is not connected to the main wall, so the right hand rule simply cannot reach it. This shows that a maze-solving rule will need to be more sophisticated.



4 Other maze solvers

The Wikipedia article *Maze-solving algorithm* also describes several other methods for solving a maze, including the Pledge algorithm, Tremaux's algorithm, dead-end filling, and recursion. You could also consider regarding the maze as a sort of mathematical graph, in which case there are graph-theory methods from finding a path from one position to another.

5 Possible projects

The Wikipdedia gives a general idea of how these methods works, but it would take some thought and experiement for you to write a Python implementation of any one of these methods. So if you wish to do something in this project, don't try to set up every method!

Try a few methods that make sense to you, generate some test mazes, and observe the results, and make a report.