CS110: Introduction to Computer Science with Python

Lecture 20
Maze Solving Algorithms.

Prof. O. Karpenko

Announcements

- Midterm 1 Average: 35.7 (close to B)
- Quiz today
  - Pop quizzes from now on
- Project 1 Updated (more details on Part 2)
- Project 1 is due this Friday night
  - If your project does not run in IDLE3.4, you get a 0
  - Do not share your code / copy anybody's code or use code from the web
    - You will loose 5% of your grade

Announcements

- Updated Late Policy for Projects
  - Each person can get 2 late days TOTAL in the whole semester to apply to projects
    - Save them for harder projects
    - This project is worth 5%, the remaining three: 10%
  - Does not affect late policy for homework
    - no late homework is accepted

Project 1

Read the Maze

- Do the lab called labOct10th
  - under Modules, scroll down

Draw the Maze

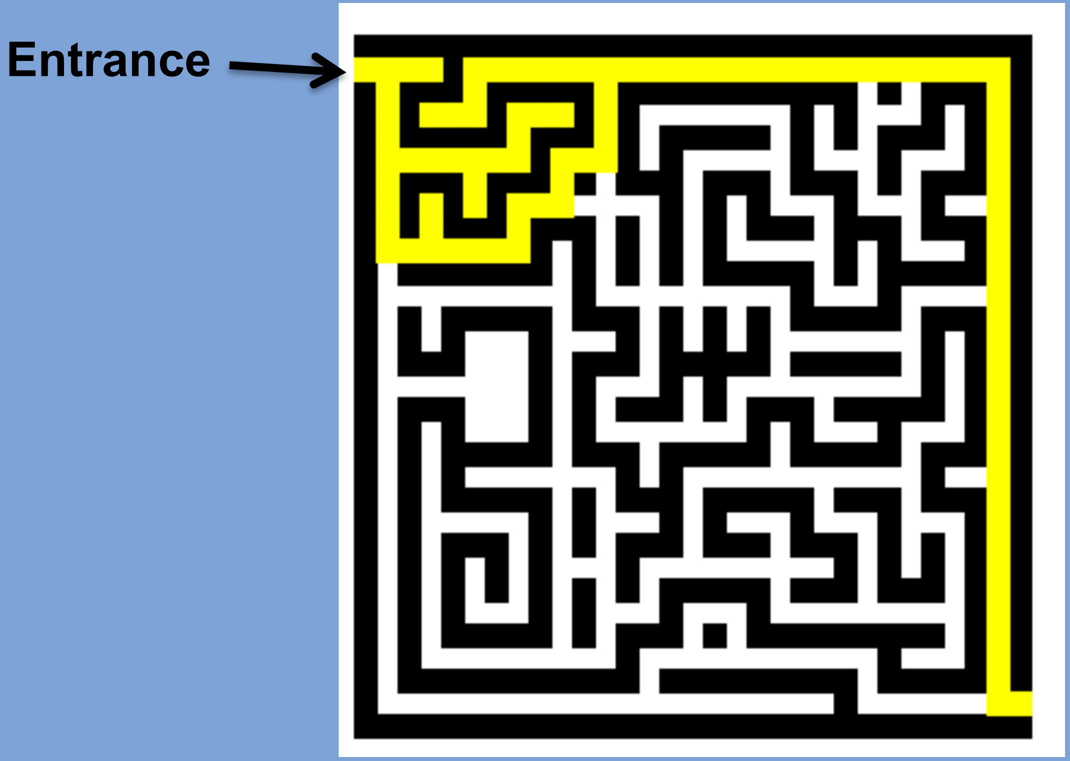
- Do the lab (LabOct6th) first
  - under Modules, scroll down

Solve the Maze

- Wall follower
  - Keep one hand (left, for instance) on the wall at all times
  - Whenever you reach a junction, always turn left (or always turn right)
- Extra credit:
  - Flood Fill
  - Random Mouse
  - Pledge algorithm

Where is the Entrance?

- For the mazes provided:
  - i = 1, j =0

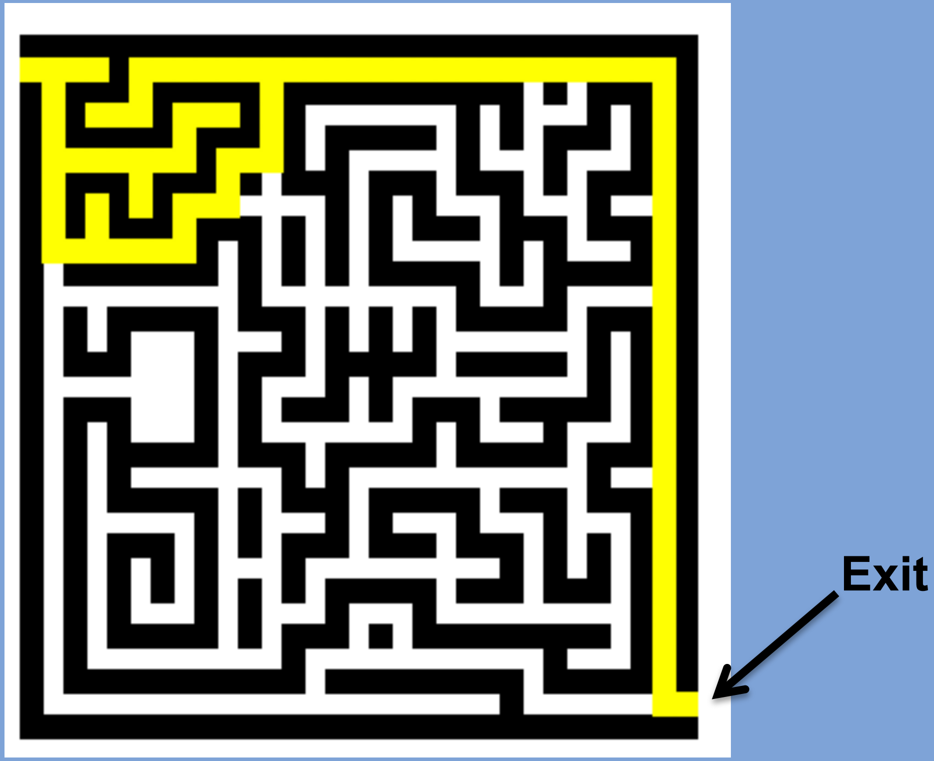


followLeftWall Function

- Start at i =1, j = 0 (Entrance to the maze)
- Initial direction is "east"
- Repeat until you find the exit:
  - call nextMove to determine where to go
  - Update i and j
    - For instance, if nextMove is "east": update to (i, j+1)
  - draw a yellow square at (i,j)

Where is the Exit?

- For the mazes provided:
  - last to last row, last column

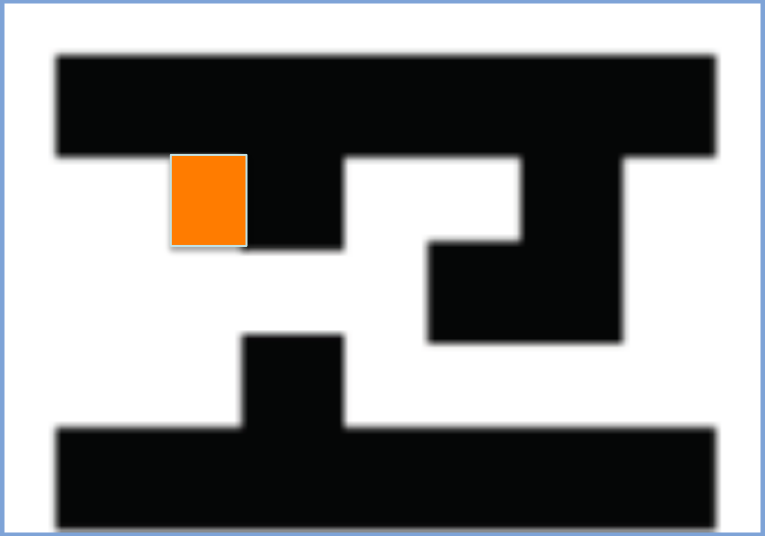


nextMove

- Takes the the current position (i,j),
- The current direction
  - one of : "east", "west", "north", "south"
- The maze

nextMove

- Needs to know where we can go:
  - Is there a wall to the east, to the south, to the north and to the west?

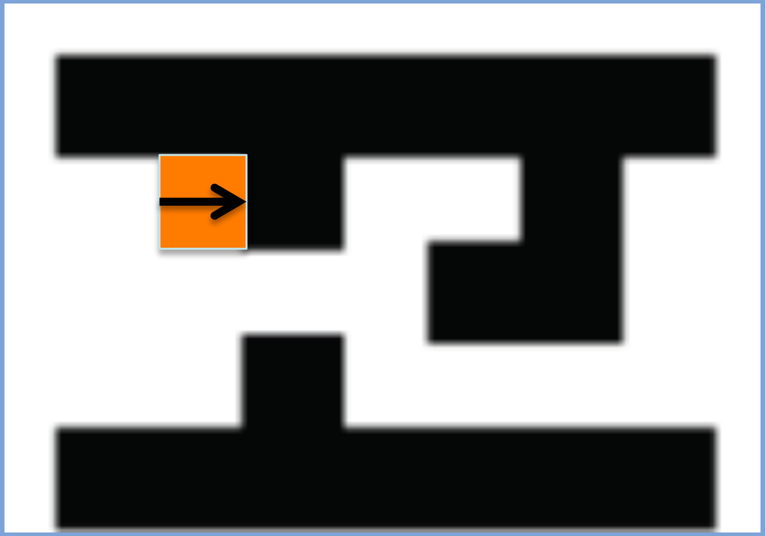


nextMove

- Tries to go left if possible (relative to your current direction)
- If can't go left, go straight,
- if can't go straight, go right
- If can't go right, go backwards

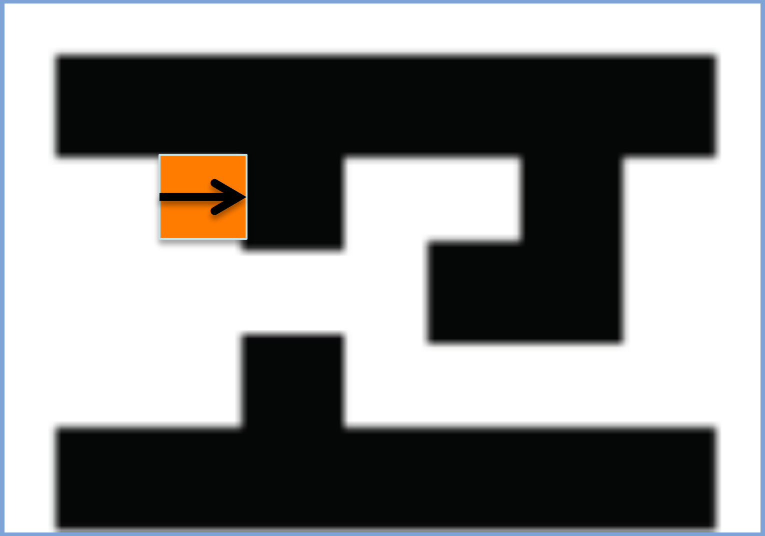
nextMove

- For this example, assuming we were going "east", where should we go?



nextMove

- For this example, assuming we were going "east", where should we go?
- New direction: "south"

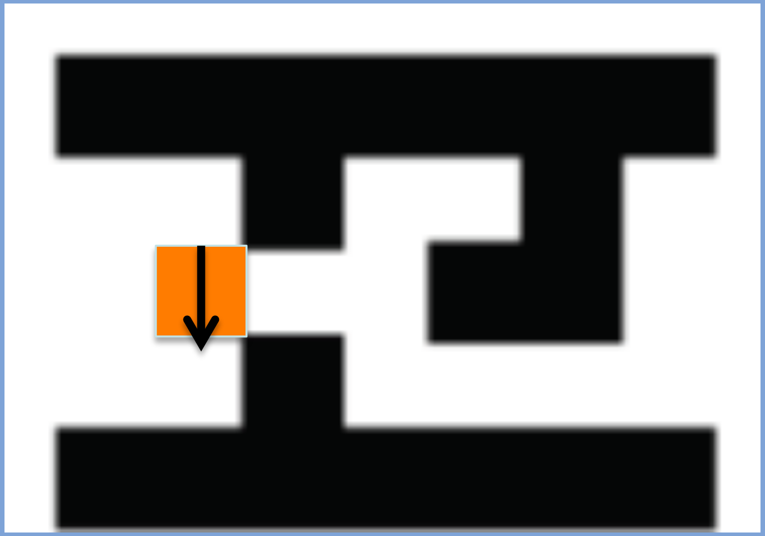


(because we can't

go left or straight)

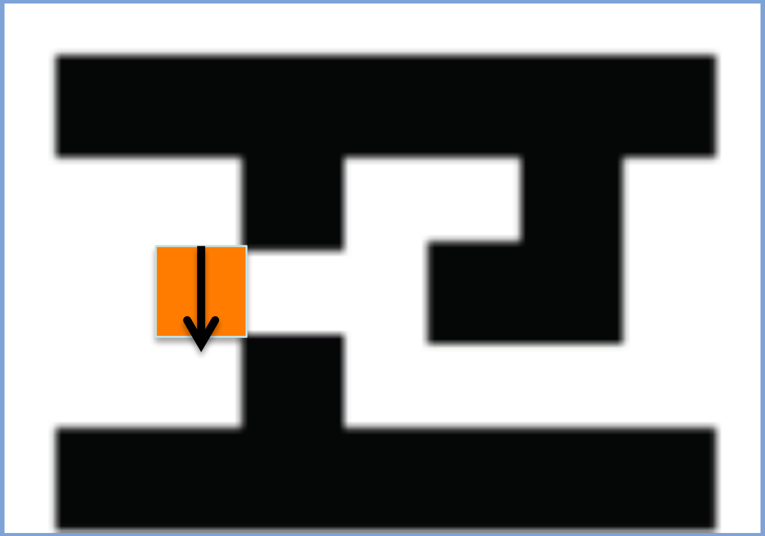
followLeftWall Function

- Change i, j to i+1, j
- New direction: "south"



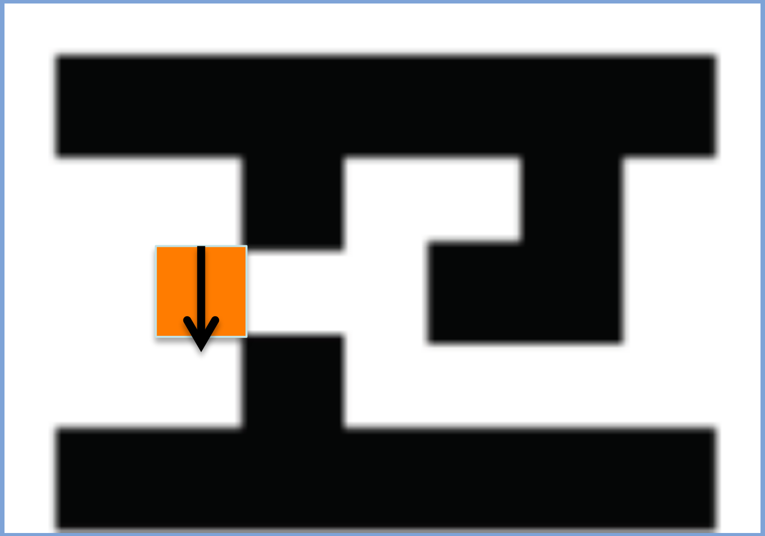
nextMove

- What will be the new direction?



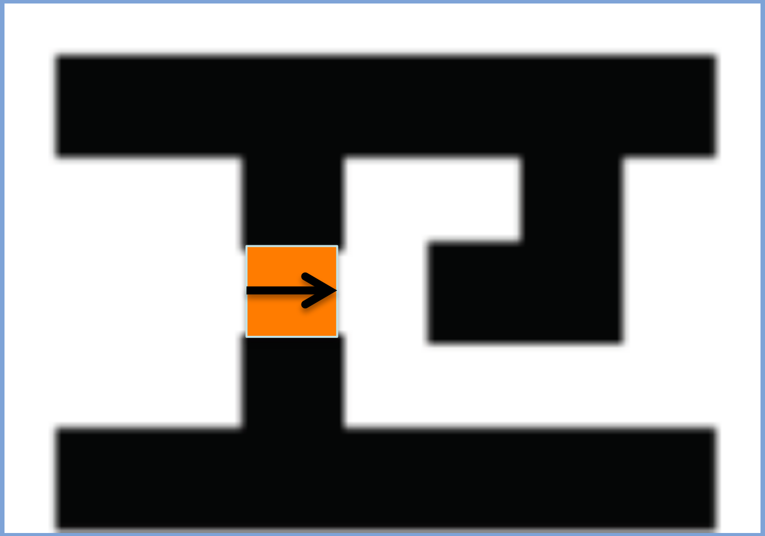
nextMove

- What will be the new direction?
  - East (because we prefer to go left)



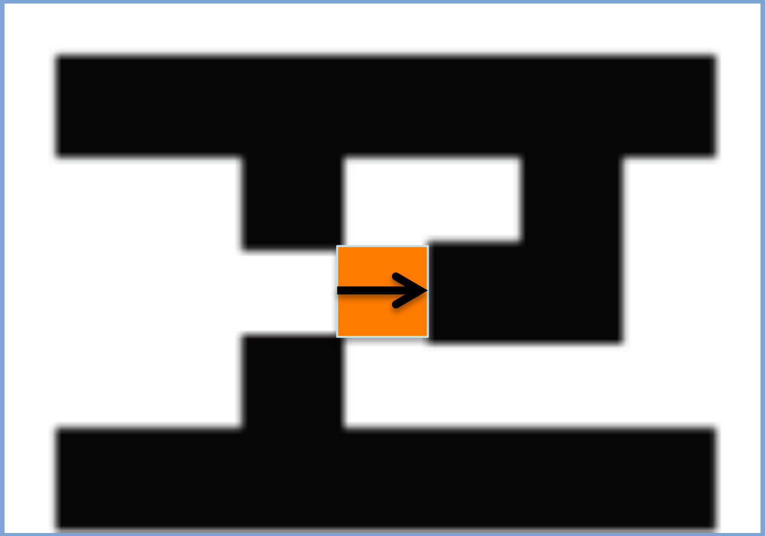
followLeftWall Function

- Update i, j (j = j + 1)
- Change the current direction to "east"



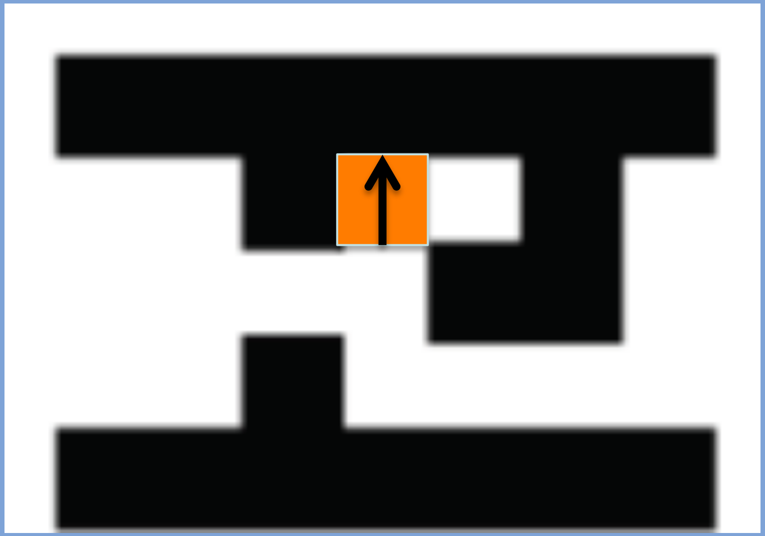
followLeftWall Function

- Update i, j (j = j + 1)
- Change the current direction to "east"



followLeftWall Function

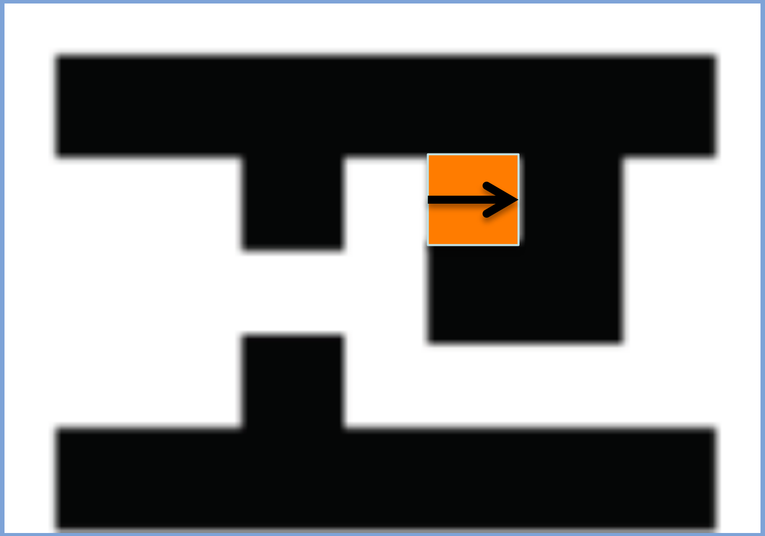
- Update i, j (i = i -1)
- Change the current direction to "north"



Becase nextMove returned "north"

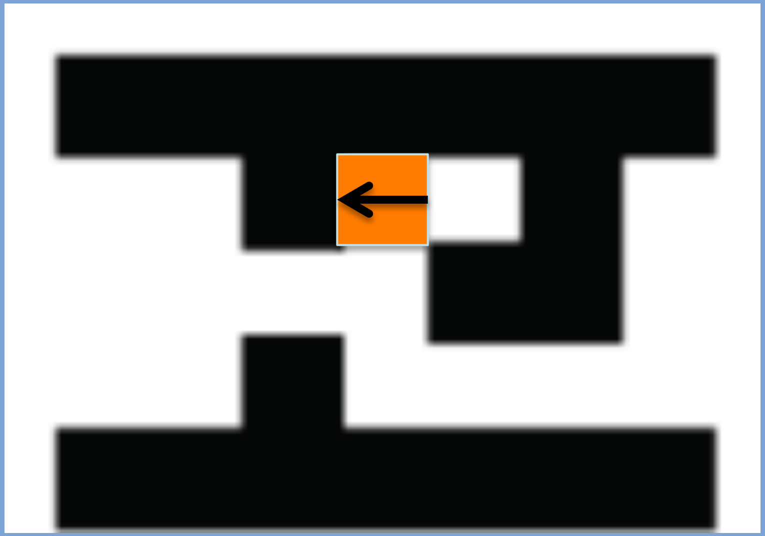
followLeftWall Function

- Update i, j (j = j +1)
- Change the current direction to "east"



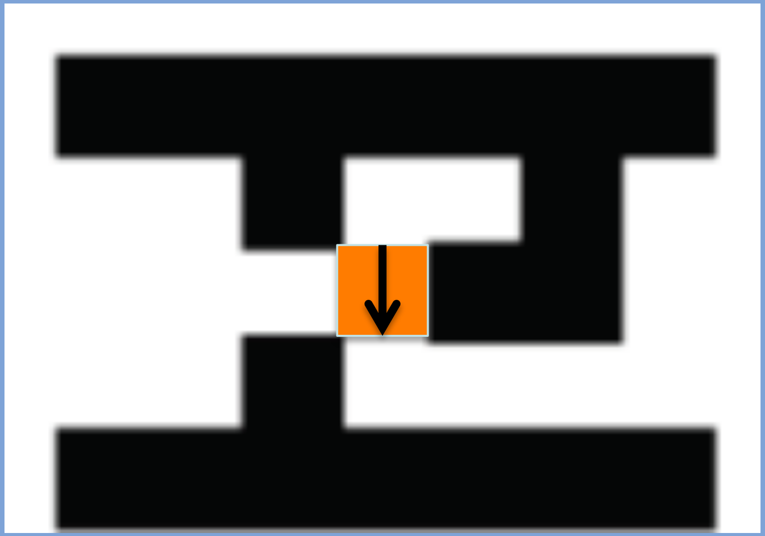
followLeftWall Function

- Update i, j (j = j -1)
- Change the current direction to "west"



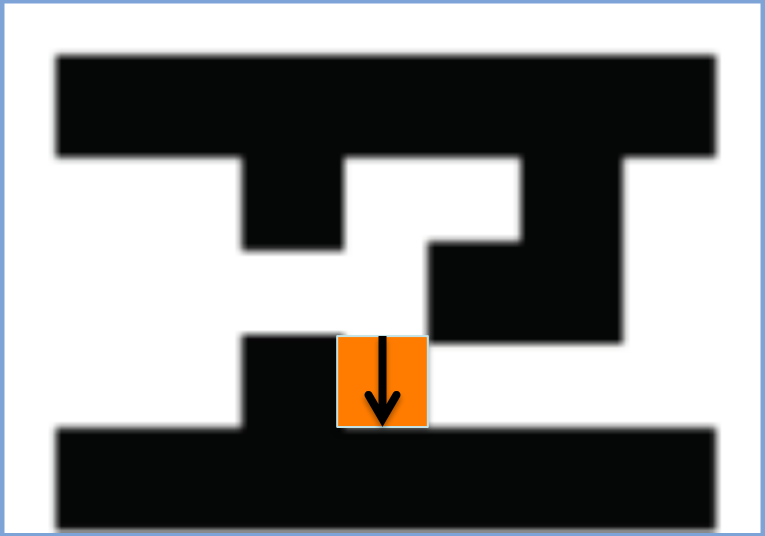
followLeftWall Function

- Update i, j (i = i +1)
- Change the current direction to "south"



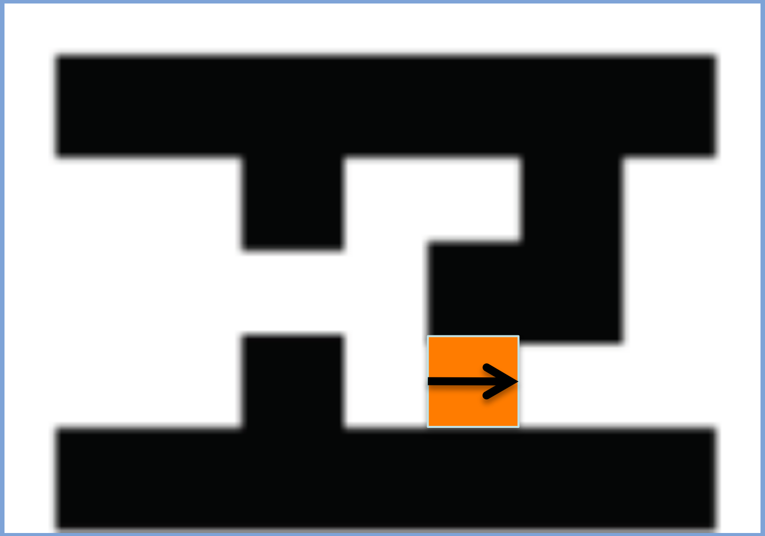
followLeftWall Function

- Update i, j (i = i +1)
- The current direction remains "south"



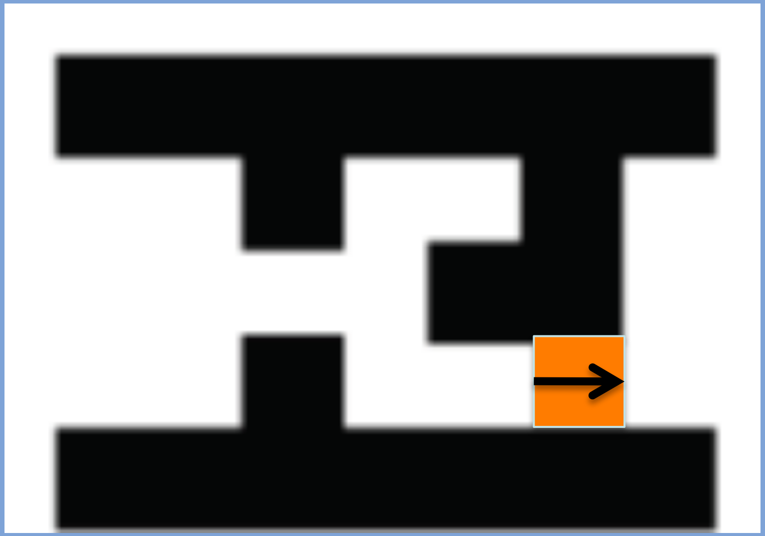
followLeftWall Function

- Update i, j (j = j +1)
- Change the current direction to "east"



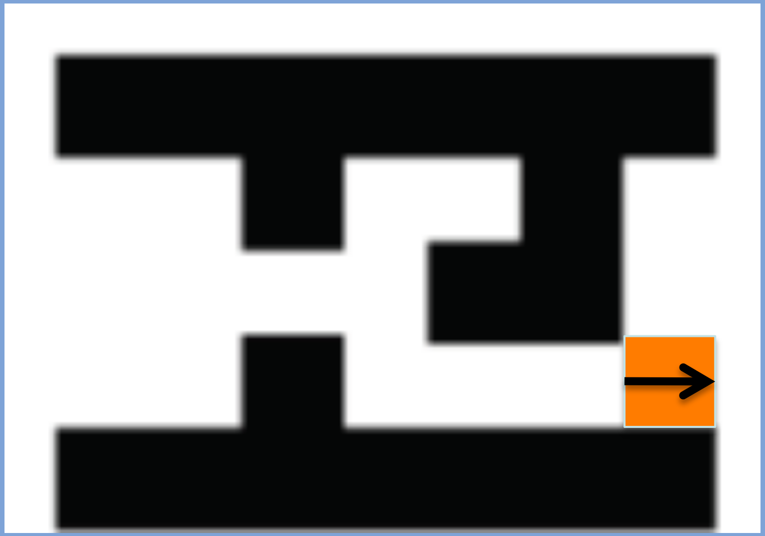
followLeftWall Function

- Update i, j (j = j +1)
- Keep going "east"



Wall Follower is Done

- Found the exit



Extra Credit Algorithms

Random Mouse Algorithm

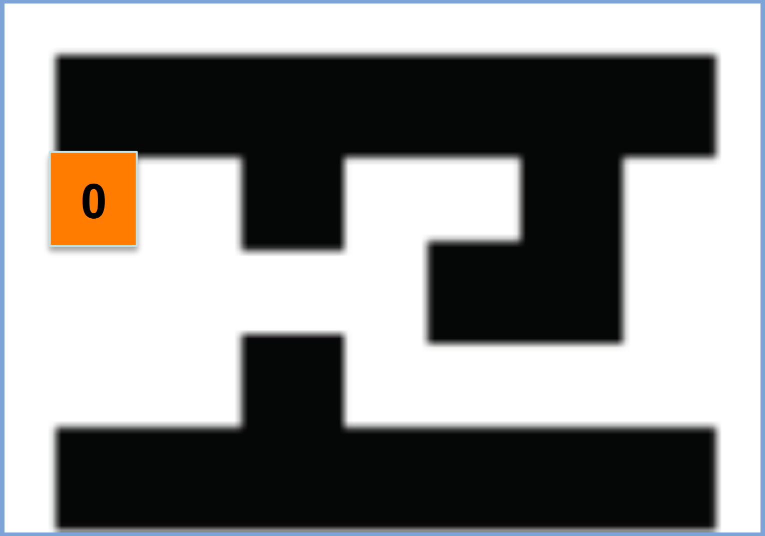
- Start going in one direction, follow this passage through any turnings until you get to a junction.
- Choose randomly between straight, right and left
- If you can't go straight, left or right, then go backwards till the next intersection

Flood Fill

- Stand at the entrance with the hose
- Turn the hose on, the water will start filling the maze
  - Assume that the water can not go through the walls

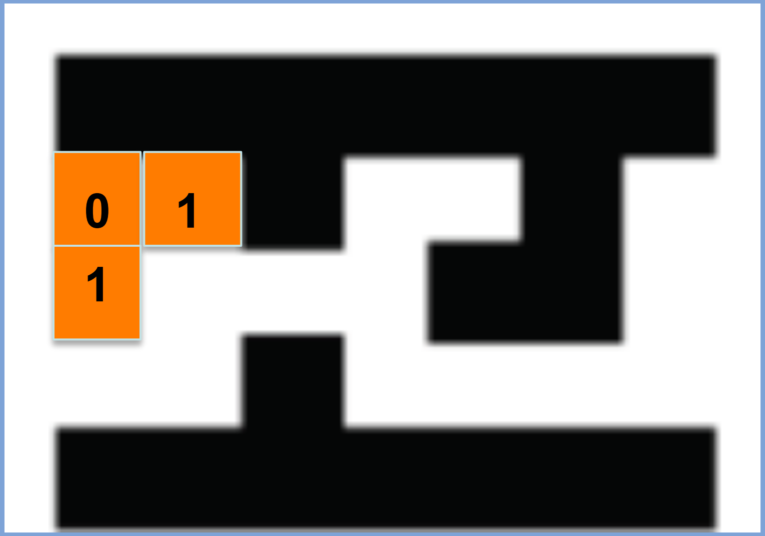
Flood Fill

- Start at the entrance
- 0: the distance the water traveled



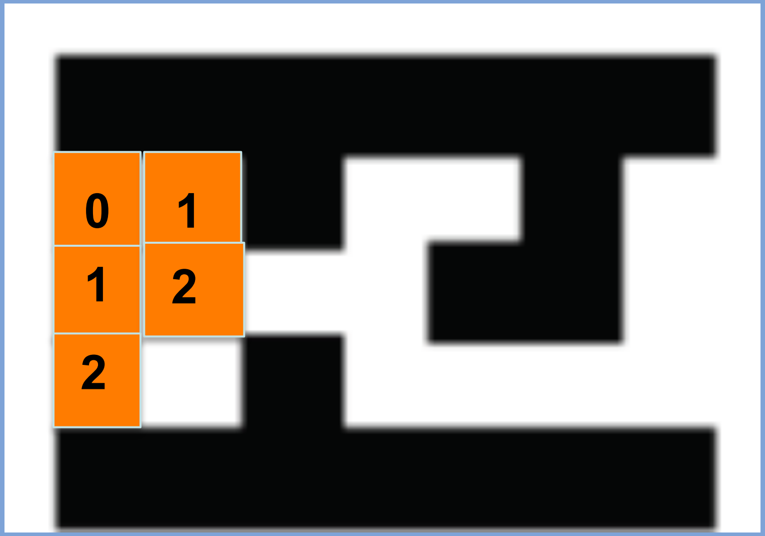
Flood Fill

- Time step 1: the water "flows" to the immediate neighbors



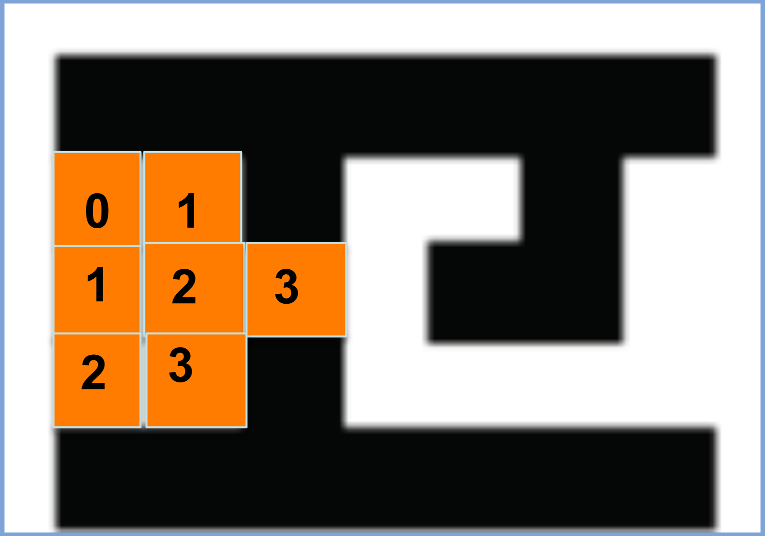
Flood Fill

- Time step 2: the water flows to the immediate neighbors of neighbors



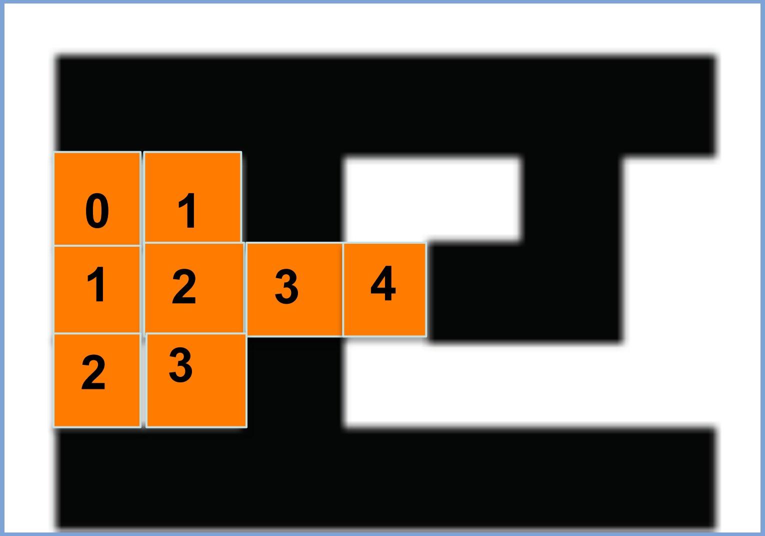
Flood Fill

- Time step 3



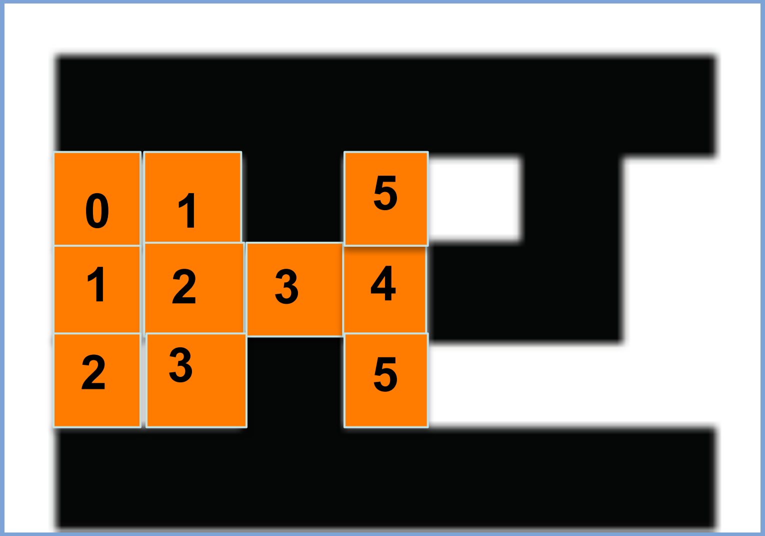
Flood Fill

- Time step 4



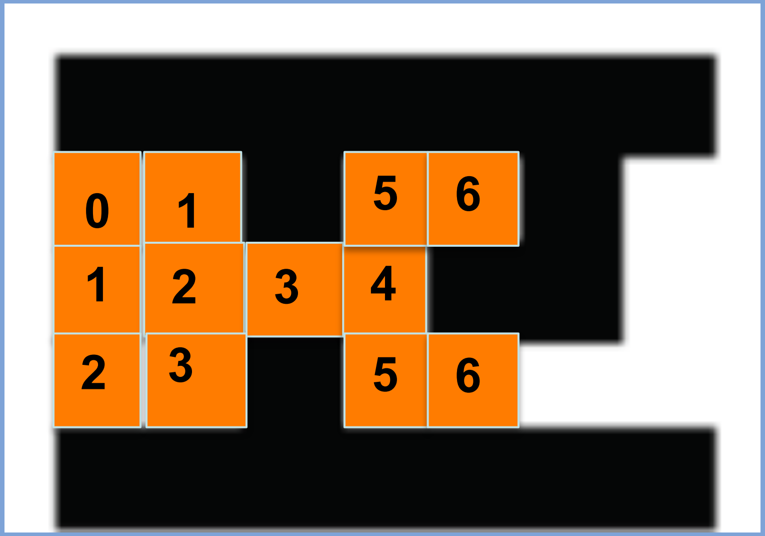
Flood Fill

- Time step 5



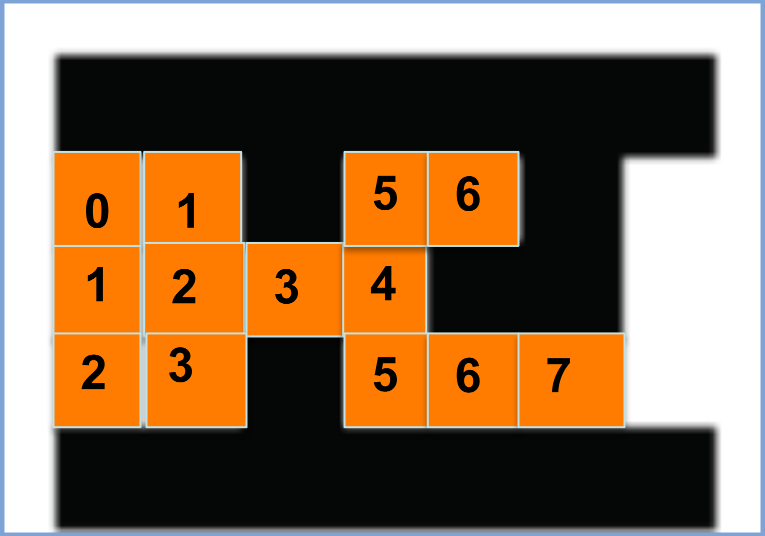
Flood Fill

- Time step 6



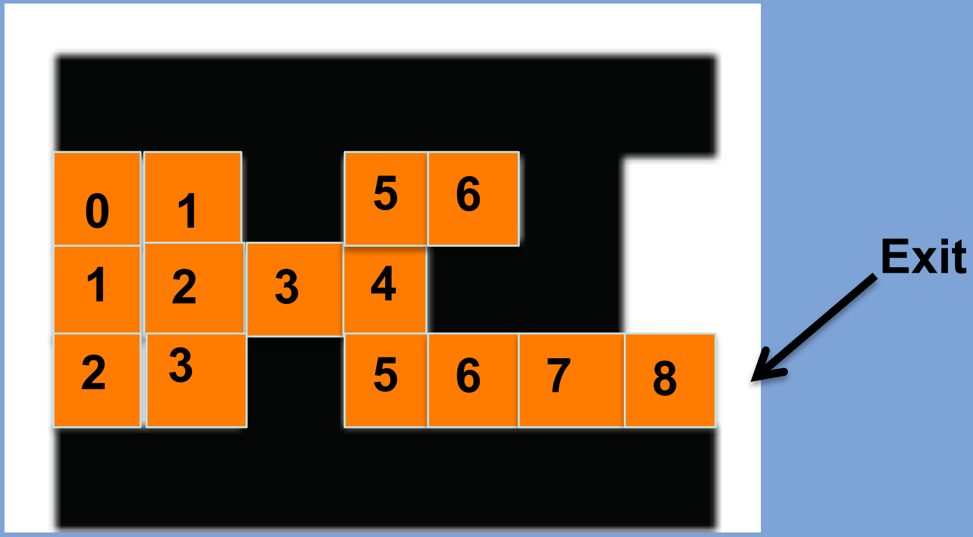
Flood Fill

- Time step 7



Flood Fill

- Time step 8



Implementation

```
REACHED = 2
While we did not "reach" the exit:
   Scan the whole maze (in i and j)
      •Consider the cell (i, j)
      •If it's a wall, or already set to REACHED, skip it
      •Set  (i,j) to REACHED if one of the immediate neighbors is REACHED
```

More on Maze Solving Algorithms

- More about maze solving algorithms:

http://en.wikipedia.org/wiki/Maze_solving_algorithm

Testing

- You need to make sure your project works for different inputs
- Need to test it

Testing

- Can you run it on the simplest example and conclude it "works"?

Testing

- Can you run it on the simplest example and conclude it "works"? No!!!
- The right approach:
  - What do I need to do to break this program?
    - Bang on it till you find bugs
- Once you found the case where it breaks, you start debugging it

Debugging

- Play Computer
  - print intermediate values
  - You can also use Python Online Tutor to execute your code line by line
- Think Backward

https://courses.cs.washington.edu/courses/cse140/13wi/debugging.html