Practical - 7

Implementing a Maze Solver using AI Search Algorithms (BFS & DFS).

Objective: Solve AI search problems using Graph Search Algorithms..

Explanation:

Maze Representation:

- The maze is represented as a list of lists (a 2D grid).
- Each inner list represents a row.
- '#' indicates a wall.
- ' ' indicates an open path.
- 'S' is the starting point.
- 'E' is the ending point.

Example: Tree Expansion for Maze Search

Assume a simple maze grid (S = Start, E = End):

```
#####
#S  #
```

```
# #E#
#####
```

BFS/DFS Exploration Tree (from 'S' at (1,1))

```
(1,1) S
|
+-- (1,2)
|   |
|   +-- (1,3)
|       |
|       +-- (2,3) E
|
+-- (2,1)
```

Explained:

- The root node `(1,1)` is the start position 'S'.
- Possible moves:
    - Right to `(1,2)`, then right to `(1,3)`, then down to `(2,3)` (which is 'E').
    - Down to `(2,1)`, but from there only wall or already visited positions are possible.

BFS Path:

- Explores all neighbors at each "layer"—would reach `(2,3)` through `(1,2)`, `(1,3)` as shortest.

DFS Path:

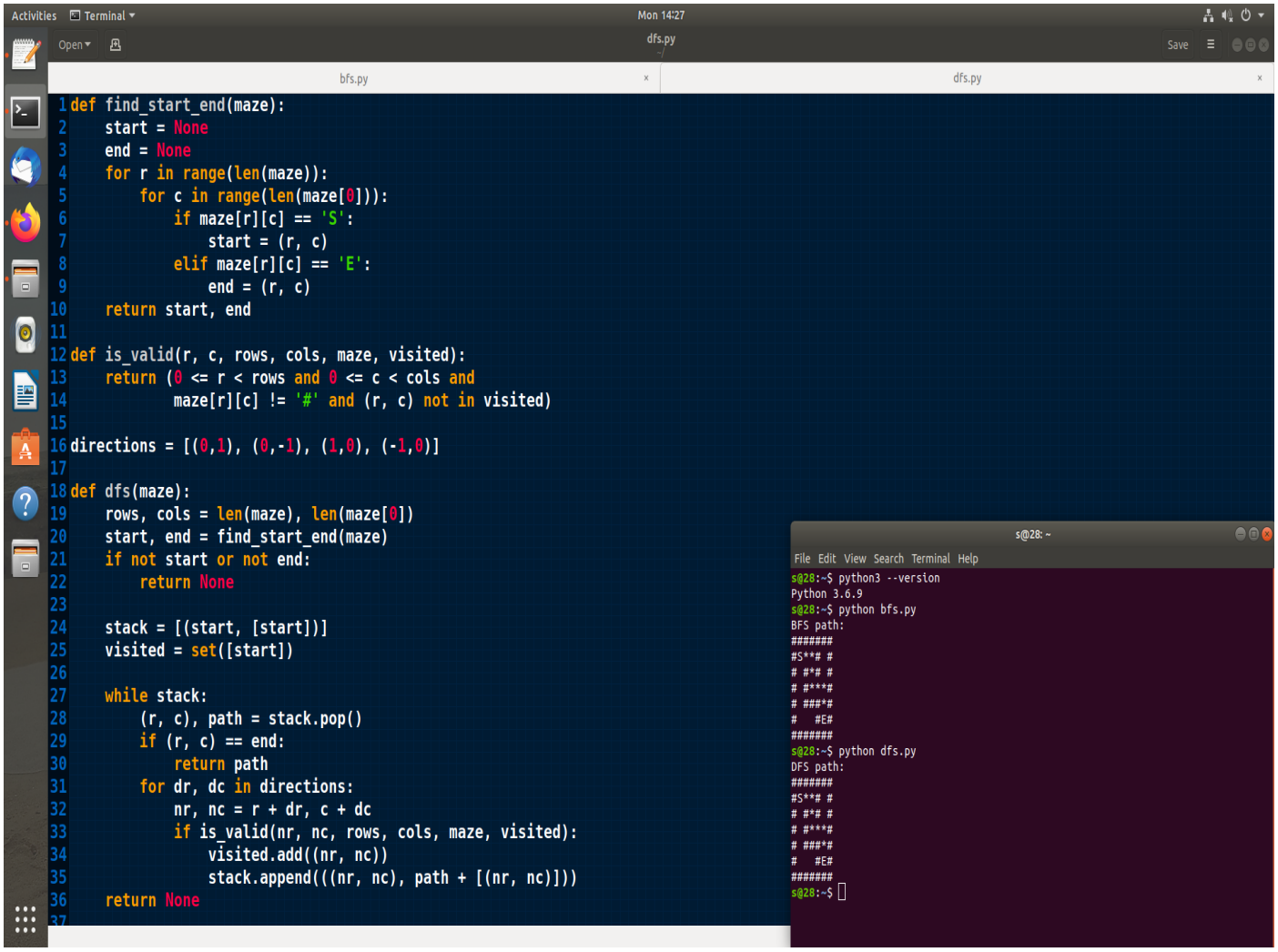
- Could go deep along one path: e.g., from `(1,1)` to `(1,2)`, then to `(1,3)`, and finally to `(2,3)`.

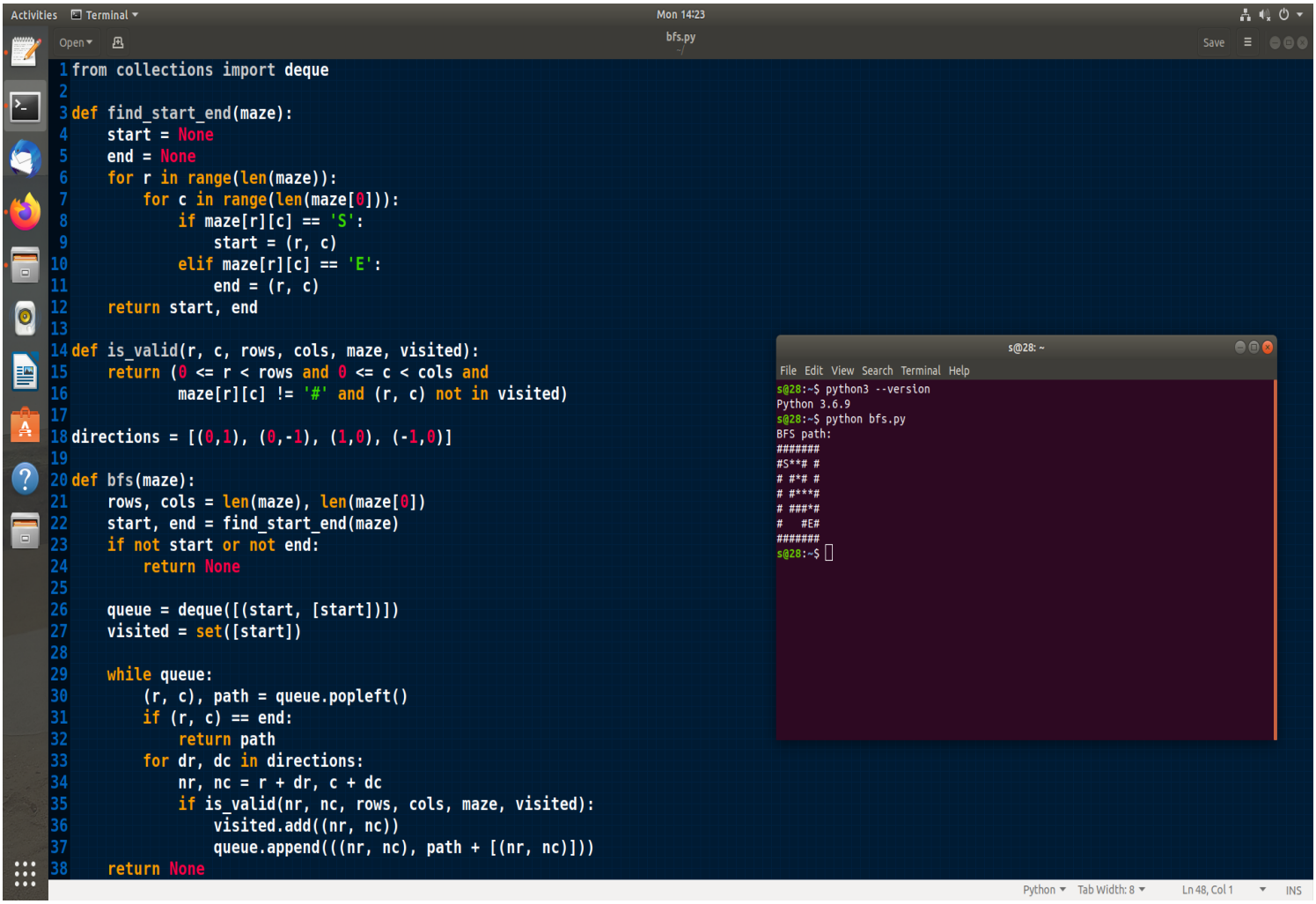
Visual Outline (Text Tree)

```
S (1,1)
|
+-- Right -> (1,2)
|    |
|    +-- Right -> (1,3)
|         |
|         +-- Down -> (2,3) [E]
|
+-- Down -> (2,1)
```

- BFS/DFS both expand nodes like a tree rooted at start, each branch showing a possible move.
- For BFS, all nodes on each level are explored before going deeper.
- For DFS, a full branch is explored as far as possible before backtracking.

This tree structure helps visualize how AI search algorithms traverse decision points in a maze.




Activities
Terminal
Mon 14:23
Open
bfs.py
Save
from collections import deque
def find_start_end(maze):
start = None
end = None
for r in range(len(maze)):
for c in range(len(maze[0])):
if maze[r][c] == 'S':
start = (r, c)
elif maze[r][c] == 'E':
end = (r, c)
return start, end
def is_valid(r, c, rows, cols, maze, visited):
return (0 <= r < rows and 0 <= c < cols and
maze[r][c] != '#' and (r, c) not in visited)
directions = [(0,1), (0,-1), (1,0), (-1,0)]
def bfs(maze):
rows, cols = len(maze), len(maze[0])
start, end = find_start_end(maze)
if not start or not end:
return None
queue = deque([(start, [start])])
visited = set([start])
while queue:
(r, c), path = queue.popleft()
if (r, c) == end:
return path
for dr, dc in directions:
nr, nc = r + dr, c + dc
if is_valid(nr, nc, rows, cols, maze, visited):
visited.add((nr, nc))
queue.append(((nr, nc), path + [(nr, nc)]))
return None
s@28: ~
File Edit View Search Terminal Help
s@28:~$ python3 --version
Python 3.6.9
s@28:~$ python bfs.py
BFS path:
#######
#S**# #
#*#
#***#
###*#
#E#
#######
s@28:~$
Python
Tab Width: 8
Ln 48, Col 1
INS