AI_Assignment_2

August 22, 2025

0.1 AI Assignment 2 Solution

Part A — Best First Search (Greedy Search)

[4]:
```
## Python Implementation
import heapq
import math

# 8 possible moves
DIRS = [(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0), (1,1)]

def valid(n, x, y, grid):
    return 0 <= x < n and 0 <= y < n and grid[x][y] == 0

# Heuristic (Euclidean distance)
def heuristic(x, y, gx, gy):
    return math.sqrt((x-gx)**2 + (y-gy)**2)

# ---------------- Best First Search ----------------
def best_first_search(grid):
    n = len(grid)
    if grid[0][0] != 0 or grid[n-1][n-1] != 0:
        return -1, []
    goal = (n-1, n-1)

    pq = [(heuristic(0,0,*goal), (0,0), [(0,0)])]
    visited = set([(0,0)])

    while pq:
        _, (x,y), path = heapq.heappop(pq)
        if (x,y) == goal:
            return len(path), path
        for dx,dy in DIRS:
            nx, ny = x+dx, y+dy
            if valid(n, nx, ny, grid) and (nx,ny) not in visited:
                visited.add((nx,ny))
                heapq.
 ↪heappush(pq,(heuristic(nx,ny,*goal),(nx,ny),path+[(nx,ny)]))
    return -1, []
```

Part B — A* Search

[6]:
```
# ---------------- A* Search ----------------
def a_star_search(grid):
    n = len(grid)
    if grid[0][0] != 0 or grid[n-1][n-1] != 0:
        return -1, []
    goal = (n-1, n-1)

    pq = [(heuristic(0,0,*goal), 0, (0,0), [(0,0)])]
    g_cost = {(0,0):0}

    while pq:
        f,g,(x,y),path = heapq.heappop(pq)
        if (x,y) == goal:
            return len(path), path
        for dx,dy in DIRS:
            nx, ny = x+dx, y+dy
            if valid(n,nx,ny,grid):
                ng = g+1
                if (nx,ny) not in g_cost or ng < g_cost[(nx,ny)]:
                    g_cost[(nx,ny)] = ng
                    f = ng + heuristic(nx,ny,*goal)
                    heapq.heappush(pq,(f,ng,(nx,ny),path+[(nx,ny)]))
    return -1, []


def run_case(grid):
    blen, bpath = best_first_search(grid)
    if blen == -1:
        print("Best First Search → Path length: -1")
    else:
        print(f"Best First Search → Path length: {blen}, Path: {bpath}")
    alen, apath = a_star_search(grid)
    if alen == -1:
        print("A* Search → Path length: -1")
    else:
        print(f"A* Search → Path length: {alen}, Path: {apath}")


# Example 1
print("Example 1:")
grid1 = [[0,1],[1,0]]
run_case(grid1)

# Example 2
print("\nExample 2:")
grid2 = [[0,0,0],[1,1,0],[1,1,0]]
```

```
run_case(grid2)

# Example 3
print("\nExample 3:")
grid3 = [[1,0,0],[1,1,0],[1,1,0]]
run_case(grid3)
```

```
Example 1:
Best First Search → Path length: 2, Path: [(0, 0), (1, 1)]
A* Search → Path length: 2, Path: [(0, 0), (1, 1)]

Example 2:
Best First Search → Path length: 4, Path: [(0, 0), (0, 1), (1, 2), (2, 2)]
A* Search → Path length: 4, Path: [(0, 0), (0, 1), (1, 2), (2, 2)]

Example 3:
Best First Search → Path length: -1
A* Search → Path length: -1
```

Comparison Best First Search (Greedy): Chooses nodes only by heuristic (Euclidean distance). It is fast but not guaranteed to find the shortest path.

A* Search: Uses both path cost (g) and heuristic (h). Always returns the shortest path length (or -1 if none). More reliable but slightly more computationally expensive.