# Module 1 Project: Path Planning Analysis

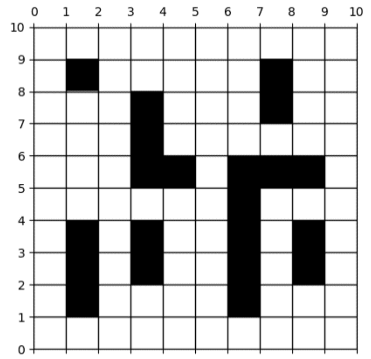
## Project Overview

You will use the concepts learned in this course to create a functioning path planner and evaluate its performance. You will implement path planning algorithms, test them in various scenarios, and analyze their performance.

You will be given several text files representing a binary occupancy grid where a `.` represents a free or open grid cell and an X is an occupied cell. For example, the text below in Table 1 on the left should be converted into a map like the image on the right.

*Table 1: Binary occupancy example*

```
. . . . . . . . . .
. X . . . . . X . .
. X . X . . . X . .
. . . X . . . . . .
. . . X X . X X X .
. . . . . . X . . .
. X . X . . X . X .
. X . X . . X . X .
. X . . . . X . . .
. . . . . . . . . .
```



For these maps, assume that the grid cells are 1 meter square. For each map, you will be given a set of starting locations and goal locations to analyze.

Your task is to implement and analyze various path-planning algorithms to find a path from each starting location to each goal location for a robot with a 1-meter diameter ensuring that the robot can traverse the path safely and avoid collisions.

## Project Requirements

1. **Collision Detection and Safety:**
   - Implement collision detection to ensure the robot does not intersect with obstacles during its path planning by converting the occupancy grid into an appropriate network graph
   - The robot is a 1-meter diameter circle, and the grid cells are 1-meter-wide squares (1m by 1m).
   - You may implement either a 4-connected or 8-connected grid movement
     - A 4-connected movement in a 2D grid will ensure no collisions, but a longer, sub-optimal path
     - An 8-connected movement will enable shorter paths, but the diagonal movements may require additional collision checking logic such as obstacle inflation.
2. **Algorithm Implementation:**
   - Implement at least two of the classical path planning algorithms: Breadth-first search, Depth-first search, A*, or Dijkstra; Note: you *must* implement either A* *or* Dijkstra.

    - Implement at least one of the following probabilistic path planning algorithms: RRT, or PRM.
    - Ensure your implementations handle various obstacle configurations and different start and goal locations.
    - Bonus points will be given for successfully implementing three or four algorithms.

3. **Performance Evaluation:**
    - Evaluate the performance of each algorithm in terms of:
        - Relative computation time.
        - Path length (physical distance).
        - Number of nodes explored.
        - Success rate in finding a path.
    - Compare the results and discuss the strengths and weaknesses of each algorithm.
4. **Visualization:**
    - Provide a visual representation of the environment, the obstacles, and the paths found by your algorithms.
    - Use plots to illustrate how the robot navigates the environment.
5. **Report:**
    - Write a detailed report documenting your project. The report should include but is not limited to:
        - An introduction to the problem and the selected algorithms.
        - A description of your implementation and the simulation environment.
        - An analysis of the performance results.
        - Figures showing the paths found by the algorithms.
        - Discussion of any challenges faced and how you overcame them.
        - Conclusion summarizing your findings and possible future improvements.

## Submission Requirements

- Submit your code in a well-documented format
- Submit your report in PDF format.
- Both the code and the report should be submitted through Canvas.
- Please do not zip or archive your files; upload them individually so that they can be viewed easily in Canvas.

## Notes and Tips

- You are encouraged to use existing libraries for visualization (e.g., Matplotlib for Python).
- You may use any programming language you are comfortable with but ensure that your submission includes clear instructions on how to run your code if necessary (particularly if you need feedback).
- Collaboration with peers is encouraged for brainstorming or working through problems, but each student must submit their individual work.
- Successful implementation of more than the required two algorithms will be considered for extra credit.

- Each map file has a few different scenarios for starting and goal points. Successfully finding a path and analyzing the performance of the algorithms for these point pairs (time to complete, length of path, nodes explored, etc.) will be considered acceptable for the project.
- To get a higher grade, consider creating additional scenarios to analyze edge cases, bulk processing many start and goal points, or other such creative ideas to demonstrate a thoughtful and thorough analysis, and analyze potential applications.

## Sample Code Snippets:

**Reading Grids from Files:**

```python
def read_grid_from_file(file_path):
    grid = []
    with open(file_path, 'r') as file:
        for line in file:
            # Strip the newline character and split by spaces
            grid.append(line.strip().split())
    return grid

# Example usage
file_path = 'path/to/your/grid.txt'
grid = read_grid_from_file(file_path)
```

**Tracking Time Complexity:**

```python
import time

start_time = time.time()
# Run the path planning algorithm
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")
```

**Tracking Memory Usage:**

```python
import tracemalloc

tracemalloc.start()
# Run the path planning algorithm
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage: {current / 10**6}MB; Peak: {peak / 10**6}MB")
tracemalloc.stop()
```

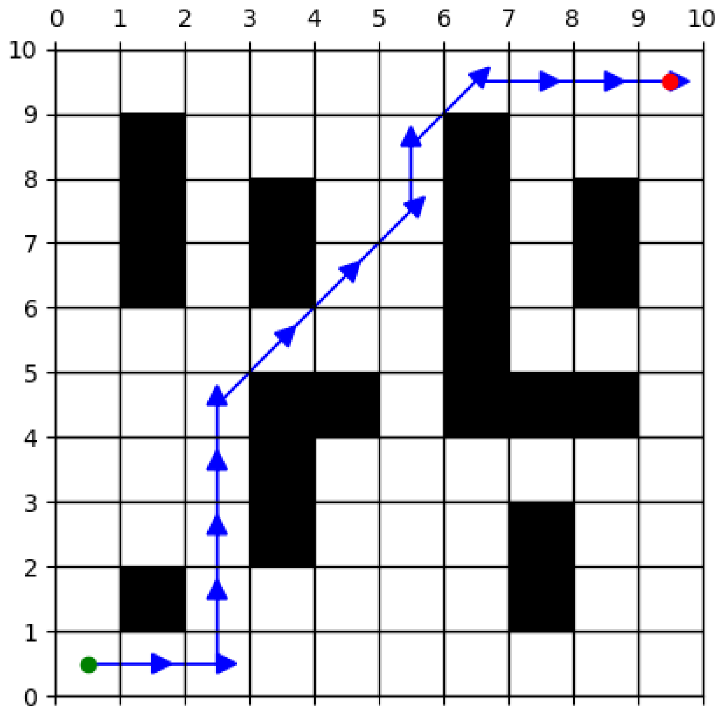
**Measuring Path Length:**

```python
def calculate_path_length(path):
    length = 0
    for i in range(1, len(path)):
        length += math.sqrt(sum((path[i][j] - path[i-1][j])**2 for j in range(len(path[i]
))))
    return length

path_length = calculate_path_length(path)
print(f"Path Length: {path_length}")
```

**Path Plotting**

```python
import matplotlib.pyplot as plt
import numpy as np

def plot_grid(grid, path=None, start=None, goal=None):
    # Create a color map for the grid
    cmap = plt.cm.get_cmap('Greys')
    cmap.set_under(color='white')  # Free space color
    cmap.set_over(color='black')   # Obstacle color

    grid_array = np.asarray(grid)
    fig, ax = plt.subplots()

    # Plot the grid with respect to the upper left-hand corner
    ax.matshow(grid_array, cmap=cmap, vmin=0.1, vmax=1.0, origin='lower')
    ax.grid(which='major', axis='both', linestyle='-', color='k', linewidth=1)
    ax.set_xticks(np.arange(-0.5, len(grid[0]), 1))
    ax.set_yticks(np.arange(-0.5, len(grid), 1))
    ax.set_xticklabels(range(0, len(grid[0])+1))
    ax.set_yticklabels(range(0, len(grid)+1))

    # Plot the path with direction arrows
    if path:
        for i in range(len(path) - 1):
            start_x, start_y = path[i]
            end_x, end_y = path[i + 1]
            ax.arrow(start_x, start_y, end_x - start_x, end_y - start_y,
                     head_width=0.3, head_length=0.3, fc='blue', ec='blue')
        # Plot the last point in the path
        ax.plot(path[-1][0], path[-1][1], 'b.')

    # Plot the start and goal points
    if start:
        ax.plot(start[0], start[1], 'go')  # Start point in green
    if goal:
        ax.plot(goal[0], goal[1], 'ro')  # Goal point in red
    return fig

# Example usage
grid = [
    ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.'],
    ['.', 'X', '.', '.', '.', '.', '.', 'X', '.', '.'],
    ['.', '.', '.', 'X', '.', '.', '.', 'X', '.', '.'],
    ['.', '.', '.', 'X', '.', '.', '.', '.', '.', '.'],
    ['.', '.', '.', 'X', 'X', '.', 'X', 'X', 'X', '.'],
    ['.', '.', '.', '.', '.', '.', 'X', '.', '.', '.'],
    ['.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'],
    ['.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'],
    ['.', 'X', '.', '.', '.', '.', 'X', '.', '.', '.'],
    ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.']
]

# Convert grid to numerical values for plotting
# Free space = 0, Obstacle = 1
grid_numerical = [[1 if cell == 'X' else 0 for cell in row] for row in grid]
grid_numerical = np.flipud(grid_numerical)
```

```
# Define start and goal positions
start = (0, 0)
goal = (9, 9)

# Example path
path = [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (5, 3), (6, 4), (7, 5), (
8, 5), (9, 6), (9, 7), (9, 8), (9, 9)]

# Plot the grid and path
f = plot_grid(grid_numerical, path=path, start=start, goal=goal)
```



Note that the plotting in the above example uses image coordinates where the origin is then forced to be in the lower left-hand corner.

## Grading rubric

| Category | Unacceptable | Marginal | Acceptable | Excellent |
|---|---|---|---|---|
| **Algorithm Implementation(s)**<br><br>Each algorithm implemented worth up to 10 points. Additional algorithms over 4 are worth half credit. The highest | No algorithm implemented, or code does not run.<br>***0 points*** | The algorithm is implemented with significant errors or omissions. Code can find a path for *at least one* start/goal pair.<br>***7 points*** | The algorithm is implemented correctly. Code runs with occasional minor errors. Algorithms successfully find a path between *all* start/goal pairs.<br>***8 points*** | The algorithm is implemented correctly and successfully finds a path to *all* start/goal pairs using an *8-connected* grid.<br>***10 points*** |

| | | | | |
|---|---|---|---|---|
| scoring implementations will be scored first. | | | | |
| **Performance Evaluation** | No performance evaluation is conducted.<br>***0 points*** | Minimal performance evaluation with limited metrics. Results are incomplete or poorly documented.<br>***14 points*** | Comprehensive performance evaluation with key metrics: computation time, path length, number of nodes expanded, and success rate. Results are well-documented and analyzed.<br>***16 points*** | In-depth performance evaluation with additional metrics and comparative analysis. Results include a variety of scenarios and edge cases with clear and detailed analysis and insights.<br>***20 points*** |
| **Visualization** | No visualizations are provided.<br>***0 points*** | Basic visualizations with limited clarity. Visualizations do not effectively convey the results.<br>***14 points*** | Clear and informative visualizations of the grid, obstacles, and paths.<br>***16 points*** | Highly detailed and professional visualizations. Visualizations include additional details indicative of understanding performance (ex: obstacle inflation, node exploration)<br>***20 points*** |
| **Code** | The student does not submit their code, or their code does not run.<br>***0 Points*** | The code is poorly organized, documented, or lacking architecture and would require significant explanation to a collaborator but is still sufficient to provide results.<br>***7 Points*** | The code is detailed with a clear structure. Code is organized into logical and intuitive segments with clear functionality.<br>***8 points*** | The code is comprehensive and well written. Details are put into the architecture such that it is readily apparent how to use the code. When not, additional comments and documentation are provided such that a collaborator can easily understand it.<br>***10 points*** |
| **Report** | The student fails to submit a report that | The report is incomplete or | The report is detailed with a | Comprehensive and well-written |

| | | | | |
|---|---|---|---|---|
| | adequately describes their work or addresses the questions and deliverables of the assignment, is poorly organized, unprofessional, or is otherwise unacceptable.<br>***0 points*** | poorly written. Lacks clear organization and sufficient detail in answering the questions and deliverables posed by the assignment.<br>***14 points*** | clear structure presenting an introduction, methodology, results, and conclusion. Thorough documentation of algorithms, simulation setup, and performance evaluation. Includes figures to support the analysis.<br>***16 points*** | report with exceptional clarity. In-depth analysis and discussion of results and challenges. Includes additional sections on potential improvements and future work. Professional presentation with high-quality figures and visuals.<br>***20 points*** |