

---

## Project 2: Path Planning COSC423/523, Fall 2021

Due: Sept 23, 11:59pm

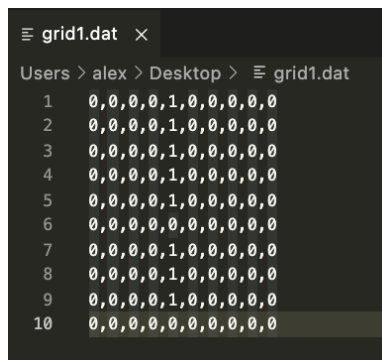
---

### 1 Overview and Instructions

Path planning is a ubiquitous task that invisibly pervades our every-day lives. This project requires that you implement a command utility that automatically finds the optimal path given a grid environment alongside a set of starting and ending nodes in the grid environment. Your implementation must be written in Python.

### 2 The Grid Environment

In this project, you will be required to implement path planning algorithms within a 2D grid environment. The environment includes obstacles that your algorithm should not consider as valid locations. The representation of the grid environment is characterized in text with 0's that indicate a valid grid node and 1's that indicate an invalid grid node, e.g. an obstacle:



```
grid1.dat x
Users > alex > Desktop > grid1.dat
1  0,0,0,0,1,0,0,0,0,0
2  0,0,0,0,1,0,0,0,0,0
3  0,0,0,0,1,0,0,0,0,0
4  0,0,0,0,1,0,0,0,0,0
5  0,0,0,0,1,0,0,0,0,0
6  0,0,0,0,0,0,0,0,0,0
7  0,0,0,0,1,0,0,0,0,0
8  0,0,0,0,1,0,0,0,0,0
9  0,0,0,0,1,0,0,0,0,0
10 0,0,0,0,0,0,0,0,0,0
```

Figure 1: An example grid environment for Project 2.

### 3 Implementation Requirements

This project requires that you write a command-line utility that implements several path planning algorithms. The implementation requirements for this project are as follows:

1. Your implementation must be in Python. You must use Python's basic data structures (e.g., lists, tuples) for storage. Python libraries for additional data structures (e.g., NumPy) or algorithm implementations (e.g., A\*) are not permitted.
2. Your implementation must be executable with the following command: `python main.py --input FILENAME --start START_NODE --goal GOAL_NODE --search SEARCH_TYPE`. If any of these arguments are missing, the program report an error to the user and should not execute. Each argument is described as follows:

- (a) **FILENAME**: A datafile that contains the grid environment. You are provided with three grid environment files as examples on Canvas. The grid can be any size  $M \times N$  where  $M$  is the number of rows and  $N$  is the number of columns. If an input file contains an invalid character (e.g., one that is not a comma, 0, or 1), the program should not run and provided an error message to the user.
  - (b) **START\_NODE**: The starting node coordinate in the grid, e.g. "0,0". If the provided argument is not in a coordinate format (i.e., number, comma, number) or outside the grid environment (i.e., beyond the grid walls), the program should not run and provide an error message to the user.
  - (c) **GOAL\_NODE**: The goal node coordinate in the grid, e.g. "7,6". If the provided argument is not in a coordinate format (i.e., number, comma, number) or outside the grid environment (i.e., beyond the grid walls), the program should not run and provide an error message to the user.
  - (d) **SEARCH\_TYPE**: The search algorithm to be used for the path planning procedure. There are four valid arguments are:
    - BFS to execute with Breadth-First Search
    - DFS to execute with Depth-First Search
    - A\* to execute with A\*
    - ALL to execute all approaches, and execute BFS, DFS, and A\* in this order.

If an invalid argument is provided, the program should not run and provide an error message to the user.
3. You must implement a `PathPlanner` class that implements three different path planning methods. The class may include other supporting methods that accompany these implementations. The required class methods each take the same set of arguments and are defined as follows:
- (a) **`depth_first_search(start, goal, grid)`**: Implements Depth First Search.
  - (b) **`breadth_first_search(start, goal, grid)`**: Implements Breadth First Search.
  - (c) **`a_star_search(start, goal, grid)`**: Implements A\* Search.

**Note:** When pathfinding, you will be required to check for valid movements. Valid moves include up, down, left, and right. Thus, you should be aware that:

- You cannot move diagonally, e.g. 0,0 to 1,1.
- You cannot move to a location outside the grid.
- You cannot move to a location that has an obstacle, i.e., with a 1

All path planning methods should return two items:

- (a) An ordered list of tuples that represents the shortest path beginning with the starting coordinate and ending with the goal coordinate. Each tuple in this list should contain a coordinate in the grid system.
- (b) The total number of points/nodes traversed.

```
(base) alex@MacBook-Pro-2:~/Desktop$ python main.py --input grid1.dat --start 0,0
--goal 7,6 --search A*
Path: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (5, 2), (5, 3), (5
, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Traversed: 21
(base) alex@MacBook-Pro-2:~/Desktop$
```

**Figure 2:** An example of appropriate command-line output for Project 2.

The output of your program should replicate the **exact** format shown in Figure 2. Your output will be automatically parsed by scripts that automatically grade your assignment. If there are mismatches in output (e.g., capitalization, missing characters, etc), your assignment will be more challenging to grade.

4. The PathPlanner class should be defined in `main.py`. Below the class definition, you should read in the appropriate input file, instantiate the class, and provide it with the appropriate information needed to execute correctly (e.g., ideally through a constructor). You are welcome to implement a class method that explicitly wraps the class's search behavior (e.g., a `run()` method).
5. Your code should be adequately documented. Each function should have a supporting docstring followed the PEP 257 convention. The top of `main.py` should include your name as an author and provide a brief description of your source code.
6. You should include a "README" file that provides an overview of your code. You should provide instructions for running your code locally.

## 4 Grading and Submission

The assignment is worth 100 points. Create a ZIP file that includes all of your Project 2 files. Upload the ZIP file to the Project 2 submission folder on Canvas by the due date. For questions regarding the late policy for assignment submission, please consult the syllabus.

### 4.1 Grading Scheme

The following grading scheme will be used for undergraduate students:

Requirement	Point Value
README exists and follows specification	10 points
Code is documented following specification	10 points
Command-Line Specification / Args / Errors	30 points
PathPlanner: DFS Implementation + Correctness	10 points
PathPlanner: BFS Implementation + Correctness	10 points
PathPlanner: A* Implementation + Correctness	20 points
TestCase Breadth: Correctness on Other Grids	10 points
<b>Total</b>	<b>100</b>