

Shortest Path Search User Manual

by Martin Colucci

Table of Contents

| | |
|---|----------|
| Section 1: About | 1 |
| Section 2: Keywords | 2 |
| Section 3: How to Run | 3 |
| Section 4: Using the Program | 3 |
| Section 4.1: Option 0 - Generate a Random Map | 4 |
| Section 4.2: Option 1 - Generate a Seeded Map | 4 |
| Section 4.3: Option 2 - Generate a Map with an Input File | 4 |
| Section 4.4: Option 3 - Run a Search Benchmark | 5 |
| Section 5: After Map Generation | 6 |
| Section 6: Inspect Cell | 8 |
| Section 7: Outputting a Map | 8 |
| Section 8: Maps as Text Files | 9 |

Section 1: About

A program that finds the shortest path from one randomly selected cell to another in a 2D grid using various search algorithms. All UI is text based and is designed to be run in console.

The program selects two positions on a 2D map and attempts to find the shortest path between them using different search algorithms. The map itself is made of different cells that each have certain qualities about them that make it more difficult for an agent to find the shortest path. Cells can be of the following types:

- **Normal Terrain:** The most common cell type, normal terrain allows the agent to travel over it and has a distance (cost) of 1. Represented with an underscore character “_”.
- **Rough Terrain:** Generates randomly across the map, allows the agent to travel over it but has a distance of 1.5. Represented with a decimal point character “.”.
- **Road:** Generates starting on one edge of the map and ending on another end of the map. Allows the agent to travel over it with a distance of 0.5. Represented with an “A” character.
- **Boulder:** Generates in randomly laid out blocks. Agents cannot travel over this type of cell. Represented with a colon character “:” normally, and a “B” character if the boulder is placed on a road.
- **Start Point:** The cell that the agent starts on. Represented with a plus character “+”.
- **End Point:** The cell that the agent is trying to navigate to. Represented with an equals character “=”.
- **Traveled:** Cells that have been traveled on by the agent while moving to the end point. Represented with an at character “@”.

This program was written by Martin Colucci for Rutgers CS440 Fall 2020

Section 2: Keywords

This user manual will make reference to several terms that may not be completely obvious to all readers. These terms are defined in context here:

- **Agent:** The simulated object that tries to travel from one point to another on the grid map based on the desired search algorithm.
- **Cell:** One tile on the grid map that stores information about how difficult it is to travel over, whether it is a start or end position, and whether the agent has traveled over it.
- **Grid Map:** The 2D space consisting of a 160x120 grid of cells that the agent is attempting to travel through.
- **UI:** User Interface; The part of the program that the user sees and interacts with.
- **Weight:** A user defined value that determines certain ranges that some of the available search algorithms will use. The ideal value of a weight is unknown and testing different weights for efficiency is one use of this program. Typically a smaller weight (between 0 and 2) produces better results than larger weights.

Section 3: How to Run

After downloading the files of the program, navigate to the folder called “AI Project 1”. Once there, open the folder called “src” and locate the “Run.java” file. Run that file in your IDE; this should open up the text UI which looks like this:

```
Welcome to the pathfinding program!  
If you would like to generate a new random map in the program enter: 0  
If you would like to generate a new seeded map in the program enter: 1  
If you would like to generate a map based on a single input file enter: 2  
If you would like to run a benchmark on a search by testing 50 maps enter: 3
```

Section 4: Using the Program

From the start screen the program will wait for a user input. Acceptable inputs are the numbers 0, 1, 2, and 3 as shown in the UI. If an unacceptable input is detected, the program will notify the user with this message before returning to its previous state:

```
Command not recognized, please try again.
```

Section 4.1: Option 0 - Generate a Random Map

Entering the number 0 when prompted instructs the program to generate a new grid map randomly. The program will notify the user of the start coordinates and the end coordinate in the grid. These are the cells that the search algorithm will attempt to find a path between.

Section 4.2: Option 1 - Generate a Seeded Map

Entering the number 1 when prompted instructs the program to generate a new grid map using a specific seed that the user will be prompted for as so:

```
Please enter the desired seed using only numbers:
```

Only numbers are accepted as input for this prompt. Attempting to enter any non-numeric characters will result in the following error and return the user to the initial menu:

```
Invalid seed, please try again.
```

When the grid map is successfully generated, the program will notify the user of the start coordinates and the end coordinate in the grid. These are the cells that the search algorithm will attempt to find a path between.

Section 4.3: Option 2 - Generate a Map with an Input File

Entering the number 2 when prompted instructs the program to generate a map based on an input file. The program will prompt the user for a file path to desired text file:

```
Please enter the file path:
```

The file path should be entered the same as it would be copied from a file explorer using backslashes (\) and not wrapped in double quotes (“ ”) or single quotes (‘ ’):

```
C:\Users\... \eclipse-workspace\AI Project 1\testMaps\input1\input1.txt
```

When the grid map is successfully generated, the program will notify the user of the start coordinates and the end coordinate in the grid. These are the cells that the search algorithm will attempt to find a path between.

If needed, the program files should include 50 maps as readable text files. To find these files, open the folder “AI Project 1” and then navigate to the “testMaps” folder. Inside of the testMaps folder are five additional folders: input1 through input5. Each input folder contains ten maps as text files. For more information about how to create a map text file that the program can read, see **Section 8: Maps as Text Files** later in this manual.

Section 4.4: Option 3 - Run a Search Benchmark

Entering the number 3 when prompted instructs the program to run a benchmark on a search algorithm selected by the user. The program will prompt the user to select one of four search algorithms. Details about which algorithms these are and how a user should select one are explained in the section **After Map Generation**.

After a search algorithm is chosen, the program will run that search on fifty preselected maps. This can take some time depending on the strength of the computer used to run this feature. When all fifty searches are done, the program will display several statistics as shown in this example:

```
Average runtime: 28 ms  
Average memory usage: 1620848 bytes  
Average move count: 173.44  
Average cells explored: 6782.16  
Average total distance (cost): 75.40556274847714
```

To interpret this information, see **Section 5: After Map Generation**. The program automatically quits after this action is performed.

Section 5: After Map Generation

After displaying the start and end coordinates, the program will prompt the user to select which search algorithm should be used. The options display as follows:

```
Enter the type of search you would like to use: A* = 0, Uniform Cost = 1, Weighted A* = 2, Sequential = 3
```

If either option 0: A* (A-Star) or option 1: Uniform Cost, are entered the program will execute the search and display the grid immediately. If option 2: Weighted A* or option 3: Sequential are selected, the program will further prompt the user to enter the desired weight(s) formatted as double(s). To format an input as a double a user can enter either a plain number or a number with a single decimal point followed by another number as shown here:

```
Please enter the desired first weight as a double:
3
Please enter the desired second weight as a double:
3.14
```

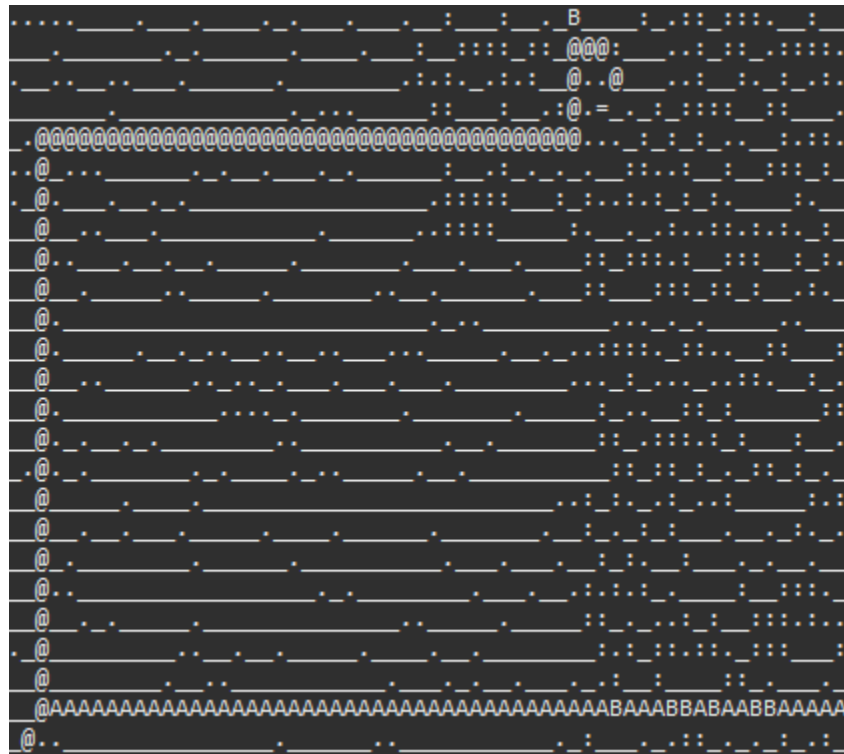
After a search algorithm and weights, if needed, have been entered, the program runs the algorithm on the map it has generated. The statistics of the search will be displayed to the user as in this example:

```
-----Search Statistics-----
Runtime: 99 ms
Memory Used: 8084712 bytes
Cells Explored: 13327
Cells in Path: 207
Total Distance (cost): 107.08084412271569
```

- **Runtime:** The amount of time, in milliseconds, it took the machine to run the search algorithm.
- **Memory Usage:** The amount of memory the algorithm used while running.
- **Move Count:** The number of cells the agent moved through to get from its start point to its end point.
- **Cells Explored:** The numbers of cells the algorithm had to investigate before finding the ideal shortest path.

- **Total Distance:** The measurable distance of the shortest paths found by the algorithm.

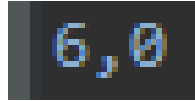
After displaying the statistics of the search, the program will display the grid map including a visual of the path taken by the agent. For details about what each character represents on the grid map, see **Section 1: About** at the beginning of the manual. Note that the grid map used in the example is smaller for the purposes of visualization, actual grid maps are much larger than this example:



Once the grid map has been printed the user will be prompted to either investigate a specific cell, output their grid map, or quit the program. To quit the program, the user must enter the character “Q”. See further in the manual for information about inspecting a cell and outputting a grid map.

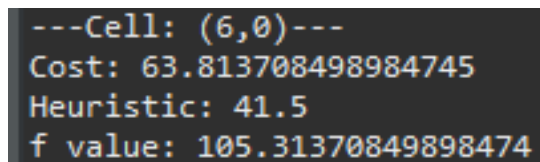
Section 6: Inspect Cell

When the grid map has displayed, a user can inspect an individual cell for information. To do this, the user must input the X and Y coordinate of the cell they wish to inspect separated by only a comma as shown in this example:



```
6,0
```

Note that the grid map considers the top-left corner of the map to be position 0,0. This means that the point 6,0 used in the above example would display information about the cell six places right of the top-left corner. The inspection displays the following data:



```
---Cell: (6,0)---  
Cost: 63.813708498984745  
Heuristic: 41.5  
f value: 105.31370849898474
```

- **Cost:** The distance (cost) the algorithm calculated it would take the agent to travel to this location, if this location was looked at by the algorithm.
- **Heuristic:** The heuristic value of the cell as calculated by the algorithm.
- **f value:** The f value of the cell as calculated by the algorithm.

The data shown by inspecting a cell is primarily used for algorithm investigation and bug testing.

Section 7: Outputting a Map

When the grid map has displayed, a user can output that grid map by entering the “O” character. This will instruct the program to construct a text file of the grid map and export it to the “AI Project 1” folder as “output.txt”. The text file does not contain the shortest path found by the search algorithm. For more information about the text file, see **Section 8: Maps as Text Files**.

Section 8: Maps as Text Files

In order for a text file to be successfully read by the program, it must follow a specific format. This format is also used when the program exports a map as a text file. In the following example image the map is reduced for visibility, an actual text file will contain a full 160x120 map of characters:

```
(18,104)
(142,30)
(22,34)
(144,20)
(59,68)
(133,112)
(34,11)
(49,10)
(22,18)
(156,50)
111110111111101111111B2010112222121200222020222222022221B100
1111110111011010111A11121101201121212102222002220122200A112
110010111111110101B11222112021011202121222220222211112B212
1110011210021122111B02220222221102222220201212002201212A221
101011112000221121B20212220122220222201221211021111111B121
1111111212112000121A02222022202022022211221200022220221B221
1011110101100122001A122022212022222221112222222111221A121
111111021121001020A22102010112202220022220221202121110B212
101011022221002221B1022122222012202220212112222212100A211
111011022112112211A02222221222220222221201212222221B201
1010111221122112111B20122212222210220200201222221212221A111
0111111011022122201B22220120222212222202220222012102202A112
1101110111212022101B2010212220222222220122112011211221A122
1001110012111112101B22212202222020120122212210021201012B211
1111101111122022200B12112221121222220121202222020210211A111
1011011021112111120B212200222222222202221222220200220B212
111111021201022120A21212122211012020002120121222222011A110
1111111121211022020A2222212211212122222222220012221212B212
1111011022121112111B2202002221222222022122220222221211A212
0111000221222212121B22102222202222220112220121012001011B201
```

- The first line of the text file contains the coordinates of the start point.
- The second line of the text file contains the coordinates of the end point.
- The third through tenth lines of the text file contain the coordinates of the eight points that chunks of boulders generate around.
- The remaining contents of the text file contains the actual map, with each cell represented as a specific character:
 - **Normal Terrain:** 1
 - **Rough Terrain:** 0

- **Road:** A
- **Boulder:** 2 if normal, B if over a road

With this information a grid map can be saved in a way that is both space efficient and speed efficient.