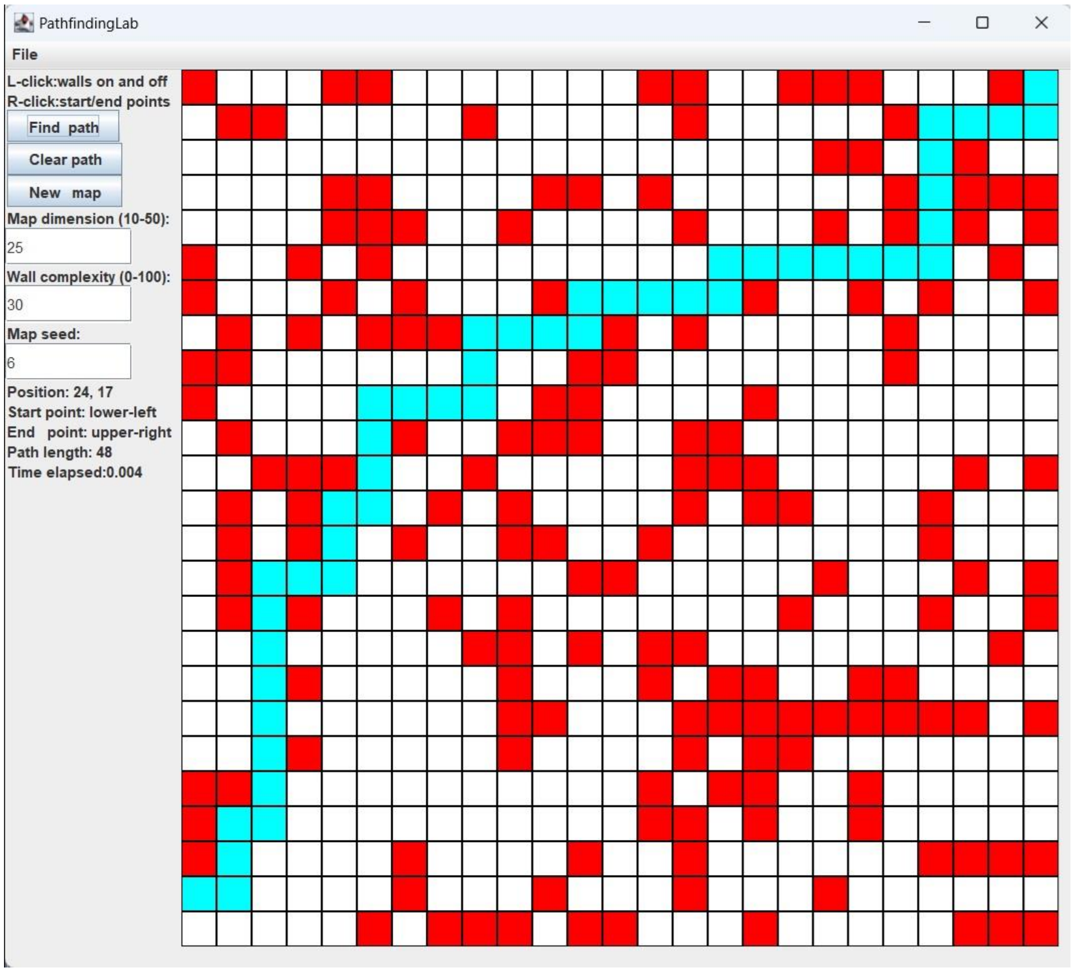
Pathfinding Lab

The aim of this lab is to help you understand how algorithms can be used to find optimized paths. Your task is to write the method `shortestPath1` in `PathfindingDriver.java`. There are numerous algorithms which can be used, and it is your task to compose one that is best suited for the job. You can invent your own algorithm, or research ideas so long that you write your own code. If you implement an idea from research, make sure to cite your source as comments in your program.



Map dimension (10-50): 25
Wall complexity (0-100): 30
Map seed: 6
Position: 24, 17
Start point: lower-left
End point: upper-right
Path length: 48
Time elapsed:0.004

- White tiles are traversable and red tiles are obstacles.
- The **Find path** button starts your pathfinding algorithm and will highlight all steps. The number of moves and the time it took will be shown at the bottom labeled **Path length** and **Time elapsed**.
- The **Clear path** button clears all highlighted tiles.
- The **New map** button creates a new map with a new randomly generated seed.
- The first text box labeled **Map dimension** will set how big the map is (25 means 25 x 25).
- The second text box called **Wall complexity** will set the density of walls placed down. A value of zero will be an empty map (no walls) and 100 will be entirely filled with walls.
- The third text box labeled **Map seed** allows you to change the random seed value.
- The **Position** text indicates what grid position your cursor is currently located on the grid. Note that the position 0,0 is in the lower-left corner where x increases to the right and y increases up.
- The **Start point** and **End point** can be selected with the right-mouse button.

Custom Maps:

- To change a tile from red to white or vice versa simply left-mouse-click on the tile.
- The start and end cells that your pathfinding algorithm is going to attempt to connect default to the lower-left and upper-right corners. If you want to set new start and end points, right-mouse-click on a cell to select the start point, then again on a different cell for the end point. They will be marked as cyan and green respectively.
- Any map can be saved to a file by clicking the File tab in the upper-left toolbar.
- If you wish to load a map, click File and then click load. A file prompter will open allowing you to choose a dot-plmap file in the maps subfolder.

Default Values:

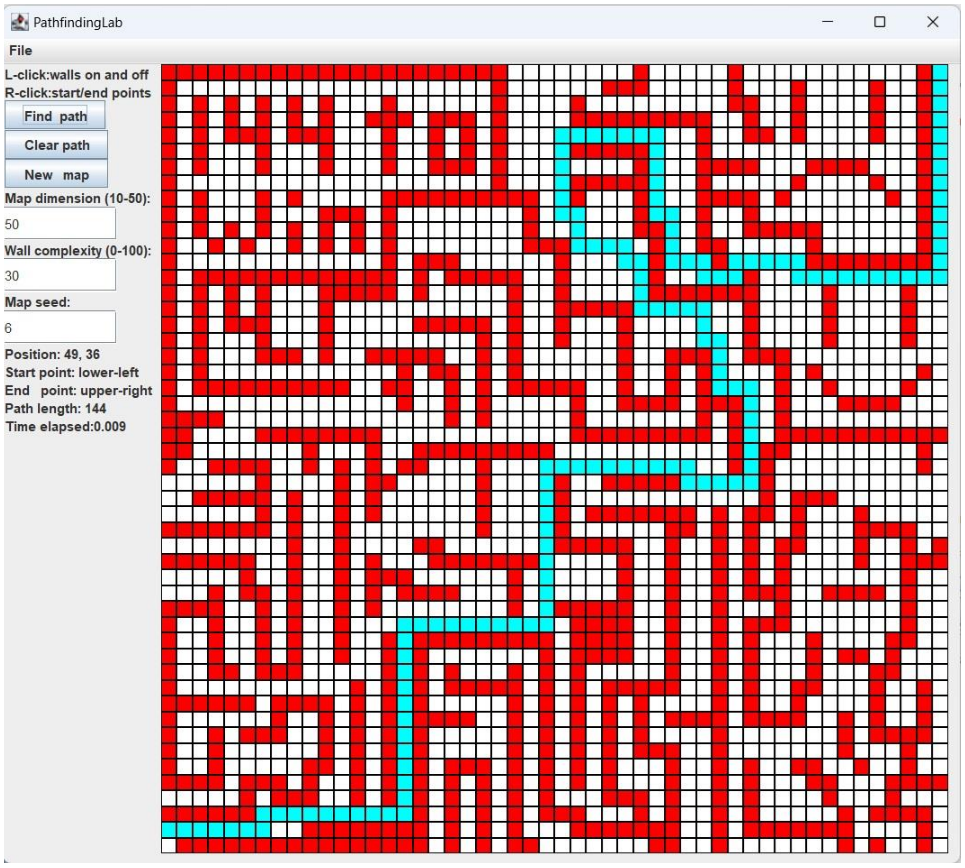
- If you want to change the default seed or size the program opens with, change the *`DEFAULT_MAP_SIZE`* or *`DEFAULT_MAP_SEED`* variable in `Map.java.`
- If you want to change how traversable the randomly generated maps are, change the *`DEFAULT_DIFFICULTY`* variable in `Map.java`: the value should be between 0 and 100, with 0 being the easiest (empty with no walls) and 100 being the hardest (unsolvable, being all walls).

Helper Methods:

Several methods exist to help you write and debug your code:

- Note: the **`java.awt.Point`** class is used to represent grid locations.
- **`boolean map.onTheMap(Point p)`**
  - Returns whether or not a grid location is in bounds of the defined map.
- **`boolean map.isTraversable(Point p)`**
  - Returns whether a grid location is traversable.
- **`void map.highlightTile(Point p, Color c)`**
  - Sets the tile at position P to the color C, useful for debugging.
- **`void map.highlightTiles(List<Point> points, Color c)`**
  - Same as above but for multiple points, useful for debugging.
- **`void map.loadFile(String filePath)`**
  - If you want to load a file programmatically.

```java
//pre:   map, initial and destination are not null.
//       initial and destination have (x,y) values that represent valid
//       (row,col) indexes of the 2D-array in map.
//post:  returns a List of Point objects that mark the shorted path in the map
//       from initial to destination, avoiding obstacles.
//defined in PathfindingDriver.java, called in UI.java
public static List<Point> shortestPath1(Map map, Point initial, Point destination)
{
     //this is the method you are to complete
}
```



L-click:walls on and off
R-click:start/end points
Map dimension (10-50): 50
Wall complexity (0-100): 30
Map seed: 6
Position: 49, 36
Start point: lower-left
End point: upper-right
Path length: 144
Time elapsed:0.009

While there are many algorithms and ideas to consider, your goal is the same: find your way to a specific location in the most efficient way, meaning the least number of moves. You should consider the following questions:

1) Will my algorithm always find a path from start to end if one is possible? (A complete algorithm.)
2) How much work is my algorithm doing? Can it be made more work or memory efficient?

Feel free to research and have some conversations: experiment with ideas and code, test and refine. There is room to put in up to four different algorithms in the Driver program if you want to experiment, refine, compare, and contrast different ideas.

Urav Tanna, Douglas Oberle 2024