Trevor Lacoste, Joe Popp, Deston Willis

Artificial Intelligence

5/7/2024

Final Project

## Project Description

The goal of this project is use the A* and Dijkstra's algorithms to help a robot nurse deliver medication, collect test samples, and function as a telemedicine interface allowing nurses to provide remote patient care. The hospital has a total of 12 wards: Admissions, General Ward, Emergency, Maternity Ward, Surgical Ward, Oncology, ICU, Isolation Ward, Pediatric Ward, Burn Ward, Hematology, and Medical Ward. The robot may initially start in any given location on the map and can move to any ward in the hospital as long as there is a clear path.

## Functionality

The find_path_color.py and the find_path_grad.py files both function exactly the same with a slight difference in the path coloring. Both of these files take input from text files in following format:

testinput.txt

```
1  Delivery algorithm: A*
2  Start location: 1,6
3  Delivery locations: 47,56 13,59 26,21 1,18 46,20 11,50 7,27 11,1 6,21
```

Once an input text file is given via the console, the parse input function will check if the algorithm and locations are correctly formatted. If the user does not follow the format, an error message will be displayed in the console with information on the issue.

```
PS C:\Users\trev6\Downloads\AIFinalProject\AIFinalProject> python find_path_color.py testinput2.txt
--------------------------------------------------
Invalid Algorithm, enter A* or Dijkstra's
--------------------------------------------------
```

In this case, the delivery algorithm entered was “greedy” so the program outputs the message “Invalid Algorithm, enter A* or Dijkstra’s.” Once a correctly formatted text file is inputted, the Maze window will open and the console will display the algorithm, starting location, and delivery locations as such:

```
==========================================================
Algorithm: A*
Start Location: (1, 6)
Delivery Locations: [(47, 56), (13, 59), (26, 21), (1, 18), (46, 20), (11, 50), (7, 27), (11, 1), (6, 21)]
----------------------------------------------------------
```

If there are locations that are either inside a wall or not inside a ward, they are removed from the queue and a message is printed for the specific reason:

```
Location (11, 1) is not inside a ward.
Location (6, 21) is inside a wall.
```

The remaining valid locations are put into a priority queue based on the priority of the ward it is in, producing a reordered queue that the find path function will iterate through with the given algorithm:

```
Reordered Queue: [(1, 18), (11, 50), (26, 21), (47, 56), (46, 20), (7, 27), (13, 59)]
----------------------------------------------------------
```

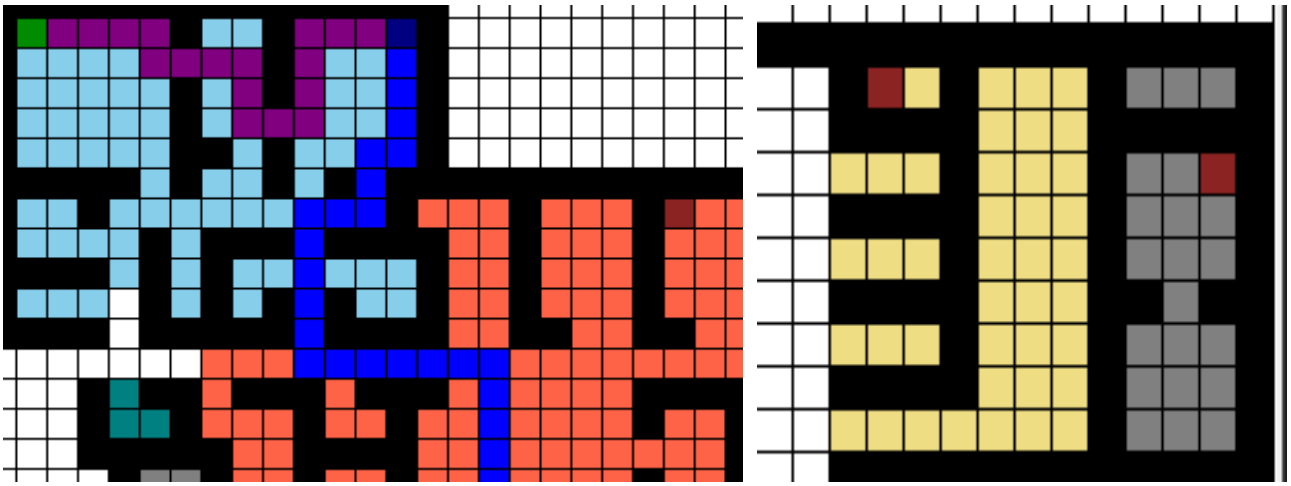
While the find path function is going through each location, the check ward function looks to see if there is another location within the same ward as the starting location, if there is, that location goes to the front of the queue. The find path function also handles any locations that are unreachable due to being encased by walls.

```
Unable to access location (11, 50)
Unable to access location (13, 59)
```

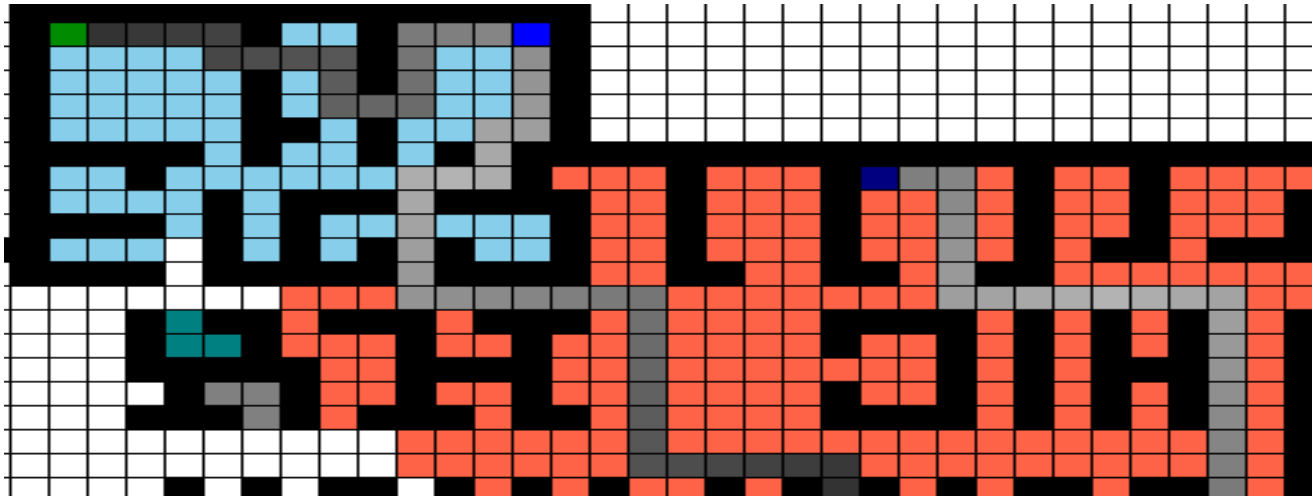
After all locations from the delivery locations list have been handled, the deliveries that were successful are printed in order:

```
Successfully delivered to [(1, 18), (26, 21), (47, 56), (46, 20), (7, 27)]
============================================================
```

In the maze window, the process is easy to see as the start location is highlighted green, the blue square indicates that a delivery location has been visited and the red square indicates a delivery location that has not yet been visited or cannot be reached. The other colors shown indicate the different wards. Each ward is represented by a unique color. The path is programmed to be displayed one cell at a time so that the user can easily follow where the algorithm finds the best route to each location. The path is also colored in a way to make it easy to see which locations have been visited and in what order:



In this case, you can see that after the first delivery location is hit, the path color changes from purple to blue. This is one visualization of the pathing. The other method has the same indicators of location but uses a gradient path.



**Path Examples**

Input:

testinput.txt

```
1   Delivery algorithm: A*
2   Start location: 1,6
3   Delivery locations: 47,56 13,59 26,21 1,18 46,20 11,50 7,27 11,1 6,21
```

Output for find_path_color.py:


Output for find_path_grad.py:

```
============================================================
Algorithm: A*
Start Location: (1, 6)
Delivery Locations: [(47, 56), (13, 59), (26, 21), (1, 18), (46, 20), (11, 50), (7, 27), (11, 1), (6, 21)]
------------------------------------------------------------
Location (11, 1) is not inside a ward.
Location (6, 21) is inside a wall.
Reordered Queue: [(1, 18), (11, 50), (26, 21), (47, 56), (46, 20), (7, 27), (13, 59)]
------------------------------------------------------------
Unable to access location (11, 50)
Unable to access location (13, 59)
Successfully delivered to [(1, 18), (26, 21), (47, 56), (46, 20), (7, 27)]
```

## Statement of Ranking

### Member 1: Deston Willis

My teammates and I agree that I handled 30% of the overall project. My specific tasks included:

Task 1: I created the create_goal_queue function which initially took in delivery locations as a parameter and returned a priority queue with the ward number with the correct priority.

Task 2: I edited the create_goal_queue function so that it  takes in a maze as well as delivery locations as parameters and returns a priority queue with the delivery location coordinates with the correct priority instead of just the ward number.

Task 3: I created the check_ward function which takes in a maze, start location, and delivery queue as parameters and checks if any locations in the queue are in the same ward as the starting location and moves them to the front of the queue if that is the case.

Task 4: I remade the maze so that the values in the maze correspond to the correct ward.

Task 5: I edited the displayed maze so that it shows the wards in distinct colors.

**Member 2: Joe Popp**

My teammates and I agree that I handled 35% of the overall project. My specific tasks included:

Task 1: I created the matrix to represent the map, determining where the walls were placed and numbered the wards initially in groups based off priority

Task 2: Starting with the A* algorithm in the class github, I altered the code to be able to accept multiple goal states, displaying an optimal path that went to each location in order

Task 3: I changed the colors of the output window to initially display a difference between priority of location

Task 4: I coded the path to change its color gradually in a grayscale gradient so as to be able to easily follow it from start to finish

Task 5: I smoothly implemented Trevor's file input algorithm to be able to accept the input text file in the run command, allowing the algorithm to use the information in the file rather than hard coding a starting location and set of goals.

Task 6: I coded a new heuristic function to account for both A* and Dijkstra's algorithm, and implemented a way for it to change based off the inputted algorithm

Task 7: I coded a way for the algorithm to skip unreachable goals without interrupting the trial and continue to ones in the list that are reachable.

Task 8: I added Deston's priority queue functions to the search algorithm so that the queue of locations were searched in order of a particular priority rather than in order of appearance on the input file.

Task 9: I implemented an output in the main function and a print results function so that everything that happens in the search can be communicated to the user in the terminal in a clear and understandable way. Information on the input, the reordered queue, unreachable locations, and the success of the trial was communicated as the algorithm ran.

Task 10: I included error checks to check for goals in walls and not in wards, starting locations in walls, and starting locations listed as a goal.

Task 11: I recolored the start and goal locations that updated as the algorithm ran. The start cell is green and all goal cells start out as red. Cells that are found turn blue, cells that are unreachable remain red.

Task 12: I edited and made the final touches to the final report

**Member 3: Trevor Lacoste**

My teammates and I agree that I handled 35% of the overall project. My specific tasks included:

Task 1: I researched the argaparser in python and created a function to read input from a text file.

Task 2: I updated this function to check the input specified by the user, so that only valid input will be accepted. Valid input is then saved into a string or tuples to be used by the path finder.

Task 3: I created functions that specify the input error if one occurs. So if a number is out of range or the algorithm is spelled incorrectly, an error message will be displayed to inform the user of a solution.

Task 4: I reorganized code from the find path and reconstruct path functions into other functions to simplify the code and allow for changes to be made to alter the pathing animation.

Task 5: I added a function in the maze game class that processes one goal at a time, so that our path would be sequential instead of all paths being drawn simultaneously.

Task 6: I implemented the draw path function to show our path one being drawn one step at a time instead of all at once.

Task 7: I updated the path drawing function to change to a new color after each destination location, making it easier to distinguish between deliveries.

Task 8: I wrote the final report and organized the document