Ladi Bamgbose and Jaden Armstrong

Professor Forouraghi

Artificial Intelligence

03 May 2024

AI Final Project Report

Introduction

In this project, we focus on utilizing two prominent pathfinding algorithms, A* and Dijkstra's, to enhance the efficiency and effectiveness of a robot nurse medication delivery system. The goal is to send the robot many delivery locations and a specified delivery algorithm and then have the robot make all the deliveries successfully. The hospital environment consists of 12 wards, each with its own priority level based on the criticality of patient care. Priorities range from ICU and Emergency Room (Priority 5) to Admissions and Isolation Ward (Priority 1). The robot nurse operates according to these priorities, ensuring that urgent tasks are completed promptly while considering the overall workflow efficiency. The program, implemented in Python as combined.py, takes input asynchronously from a text file containing delivery requests. Each request specifies the algorithm to be used (A* or Dijkstra's), the start location of the robot nurse, and the delivery locations (goals). There is also an option to enter obstacles the robot must avoid. These requests are processed in a priority queue, with higher priority tasks taking precedence over lower priority ones. Additionally, the robot nurse intelligently handles situations where multiple requests exist within the same ward to optimize its route.The termination conditions of the program are clearly defined to ensure proper handling of various scenarios. The system strives for successful completion of all tasks, providing administrators with clear visualizations of the optimal paths taken. In case of obstacles or blocked paths, appropriate warnings are issued to notify administrators of any failures in task completion.

Project Description

The program starts off by making a Graph class. The purpose of this graph will be shown later, as it is used to transform our matrix into a graph to work with our a* and dijkstra algorithms. We then go on to define our heuristic which is the Manhattan distance, and define our a_star and dijkstra functions. We make a read_input_file function that handles the parsing of the .txt file containing the start location, delivery algorithm, delivery locations and obstacles, if any. This function handles the actual names of the ward instead of specified coordinates. It then returns 4 variables, start location, delivery algorithm, delivery locations and obstacles. We go on to define a function that serves to arrange the delivery locations that were received in the input file by their priority. This way the robot delivers the delivery locations in order based on their priority. If the wards specified have the same priority then order does not matter.

Moving into our main function, we get the 4 most important variables, start location, delivery algorithm, delivery locations and obstacles and print them to the screen. This way the user can see where it’s starting, where it's going and if any obstacles are specified. We then go on to define the matrix. We use a 25 x 25 matrix and the numbers inside the maze each belong to a ward. Our valid values array specifies the numbers in the matrix that the robot can travel to. We then turn our matrix into a graph so this graph can be used in our a* and dijkstra functions. Our ward coordinate map is a dictionary that sets every coordinate to a ward. This is how we are able to handle ward names instead of specified coordinates for our delivery locations. We will go into detail about this later when talking about our visit delivery locations function. We define a function called find optimum path. This function takes a graph, a start, a goal, and an algorithm. Based on these parameters it calculates the optimum path from one location to the next. This function is repeatedly called to find the best path from one location to the next. We then define a sorted delivery locations function that takes the original delivery locations specified and sorts them based on their priorities. Our visit delivery locations function takes 4 parameters. Our sorted delivery locations, a ward coordinate map, a current location (coordinate) and an array to print out to the user all the coordinates that were traveled. The function loops through every coordinate that belongs to a ward in our ward coordinate map, and sees if there is a path from our start location. If there is, it will continue to the next delivery and follow the same process. If no delivery locations were visited, it will print to the user that none were found. If some locations were visited then it will print that some delivery locations were found, and if none were found it will print that none were found.

We will now describe how we designed our maze. The "color_mapping" dictionary associates integers that represent different wall orientations associated with different wards to a hexadecimal code value that represents the color of the ward. There can be as many as eight integers that are all colored the same, representing the same ward, all relating to different orientations of walls and no walls along its sides. After this, four functions are defined "def draw_wall_(right|left|up|down)(x,y):". Since each of these processes in each of the functions need to be repeated so many times, it is easier to make these functions to each be called dozens of times. The methodology used to draw a wall is it takes the x and y coordinates, scaled to the cell size, and finds the start and end locations of where the wall would need to be and uses the TKinter Canvas object to create a line.

Next in the GUI process is the draw_maze function, which iterates through each x and y coordinate combination in the maze and a series of if clauses are made to catch each version of wall orientation and ward associated needed. First, every cell has its outlines removed to improve visual appeal and make walls more visible. If the integer value at the given cell is not -1 or -2 it is made white by default. Next, each integer of the matrix that includes a wall on any of its sides needs to be included in this code. An if clause is made individually for each integer and its necessary draw wall function will be called within this block.

Our draw path function handles printing the path to our GUI. It marks "S" as the start location and then when the robot has completed a delivery it marks a number at the location to show the order or deliveries. If there is an obstacle it will be marked by a "X".

Functionality and features

Below we will list all the functionality and features of our program:

1. *Reading the input file*- Our read input file method is able to handle actual ward names instead of just specific coordinates. It maps everything in the command line input file to variables that are used later in the program.

2. *Obstacles*- Our input file has an option to add obstacles, if specified the robot will avoid these coordinates and there will be an "X" in the graphical representation of the hospital.
3. *Printing individual paths and final path*- We show the individual paths from one location to the next and as well as the final path to the user in the terminal.
4. *Successful completion of deliveries or unsuccessful*- If some of the delivery locations can be visited the program is able to handle that and will tell the user that some of the locations were visited. If none were delivered it will tell the user that, and if all were visited it will tell the user as well.
5. *Ward coordinate dictionary*- This dictionary holds all the coordinates that are assigned to a specific ward. This allows the user to enter specific ward names instead of coordinates.
6. *GUI*- Our graphical user interface displays a representation of the hospital and it shows the path the robot took to make its deliveries. The start position is marked by a "S" and the order the robot makes its successful deliveries are numbered on the interface. It also marks obstacles by a "X".

Statement of ranking

Member 1: Ladi Bamgbose

My teammate and I agree that I handled **55%** of the overall project. My specific tasks included:

Task 1: I designed and implemented the function that handles arranging the delivery locations in order based on their priorities

Task 2: I worked on 50% of the matrix to represent the hospital

Task 3: I designed the function that helps to translate our matrix into a graph

Task 4: I designed the ward coordinate map function that assigned every coordinate to a ward

Task 5: I designed the find optimum path function that finds the optimum path from one location to the next

Task 6: I designed the visit delivery locations that handles printing the paths to the terminal and looping through the coordinates of the ward coordinate map to see if there is path from one location to the next

Task 7: I designed the draw path function that is responsible for drawing the path the robot takes to make its deliveries. It is also responsible for marking the start locations, delivery locations in order, and any obstacles in the graphical user interface

Member 1: Jaden Armstrong

My teammate and I agree that I handled **45%** of the overall project. My specific tasks included:

Task 1: I worked on 50% of the matrix to represent the hospital

Task 2: I designed the read input file function that handles reading and extracting all the information from the input file passed in the command line

Task 3: I designed the color mapping that associates an integer value from the matrix to a color signifying the ward it belongs to.

Task 4: I designed the draw maze function that draws the full representation of the hospital and all its walls

Task 5: I designed the draw wall functions that are responsible for drawing all the walls in our graphical user interface.

Task 6: I made the “Path_pictures.ipynb” that shows all the paths we showed in our presentation.

Examples

We have listed examples in a google colab called “Path_pictures.ipynb”. It is easy to see different paths and a graphical representation the robot took. Please feel free to look through the colab to see how our GUI looks!