Phu Do, Katrina Cwiertniewicz

CSC362

5/5/2024

Final Project

Robot Nurse Functionality Documentation

General Requirements

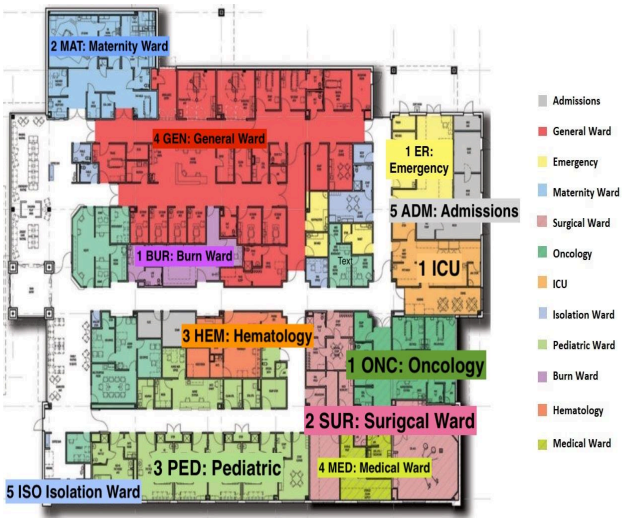
Our project includes two versions that output different map displays. They both use the same input file and algorithm to display their outputs. Version 1 displays each delivery path individually and continues to the next location when the window is closed. Version 2 displays all delivery path grids in one pop-up window in the order in which they were visited.

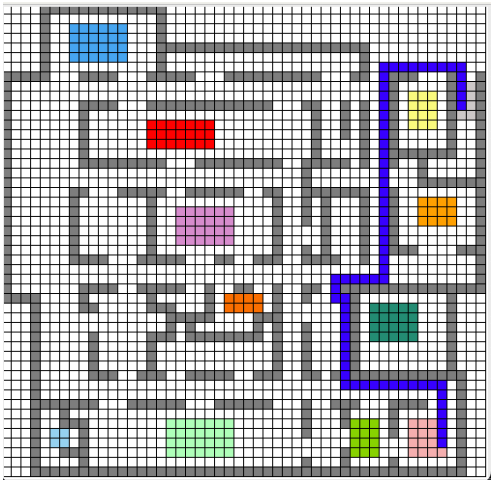
The program uses the A* algorithm for its delivery algorithm. A* incorporates a Manhattan Distance as its heuristic that makes the cost for moving an agent north, south, east, or west 1. This uses the algorithm $f(n) = g(n) + h(n)$ to determine the optimal path for the robot.

The priority queue is used to determine the order in which each ward is visited regardless of what order they were inputted by the user. Below is the code implemented to create the priority queue. 1 is the highest priority and will be a ward that is chosen first and 5 is the lowest priority and will be a ward that is chosen last in comparison to the other ward's priorities. If a ward that is chosen as a delivery location has the same priority as another one, the order will be chosen based on the ward that is closest to the current location.

```
def get_delivery_ward():
wards = "ICU": (1, (22, 44)), "ER": (1, (10, 44)), "ONC": (1, (32, 40)),
        "BUR": (1, (22, 20)), "SUR": (2, (45, 45)), "MAT": (2, (4, 10)),
        "HEM": (3, (31, 24)), "PED": (3, (44, 20)), "MED": (4, (44, 38)),
        "GEN": (4, (13, 19)), "ADM": (5, (10, 48)), "ISO": (5, (46, 5))}
```

Below on the left is a labeled hospital map and on the right is an output image of our matrix-created map. The path is shown in blue as it completes a delivery from Admissions (ADM) to the Surgical Ward (SUR). We created smaller color-coded ward blocks so the path could be displayed more prominently. Each block shows the location of where the robot nurse will arrive at the destination. The map on the left could show the user where the boundaries of the wards are and the map on the right would indicate where the robot would arrive at their destination in each ward. For our matrix map, Oncology (ONC) and Isolation Ward (ISO) are blocked with obstacles and are not accessible.



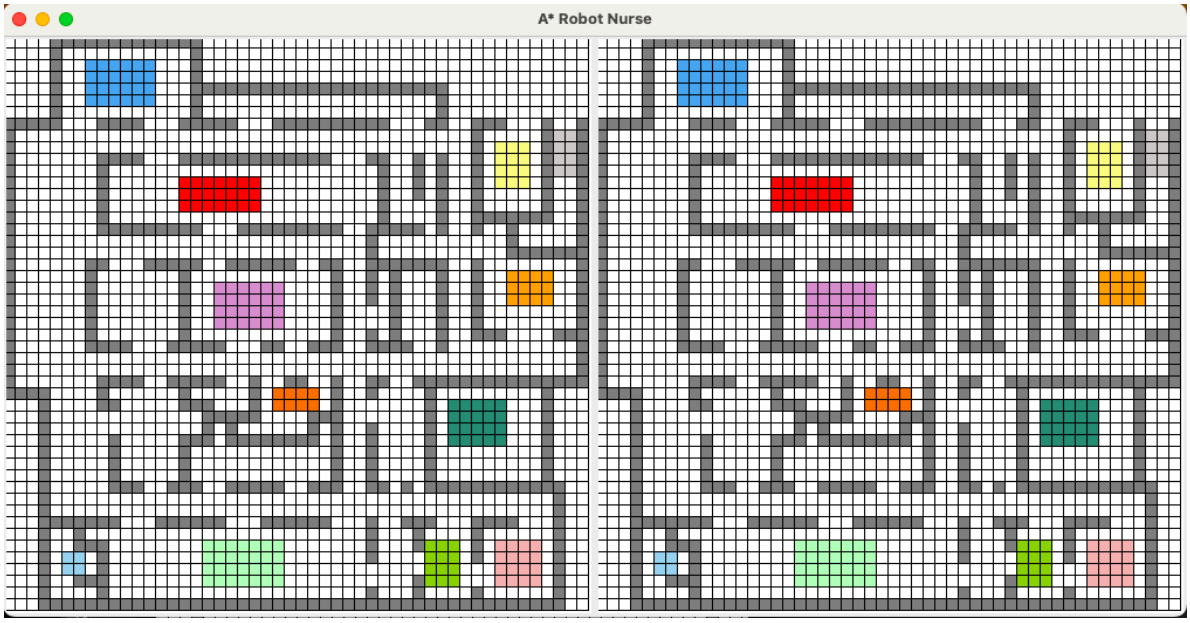


Program Termination Conditions

If all requests are completed, the following text will be displayed: "`No Failed delivery!`". This code was implemented as an else statement in the performance summary code when displaying the final notification output. (This code is mentioned below in the additional features section). Between each successful delivery, the following text will be displayed: "`Successfully delivered medication to` (ward delivered to)`!`". In version 1, the code "`sleep (10)`" delays each pop-up window for 10 seconds to simulate the robot moving to the next location. Below is the code that implemented these outputs.

```
#### When a goal is reached, reconstruct the path from start to goal
#### and update the delivery status
if current_pos == self.goal_pos:
   self.reconstruct_path()
   sleep(10)
   print("Successfully delivered medication to", self.ward_name(self.goal_pos),
"!\n")
   return self.goal_pos
```

If some of the requests are not completed due to obstacles, but the rest are completed, the following process will occur. When the failed request occurred, the pop-up window would appear but not display a path. Below is an example of a path not found for Admissions (ADM) to Oncology (ONC).



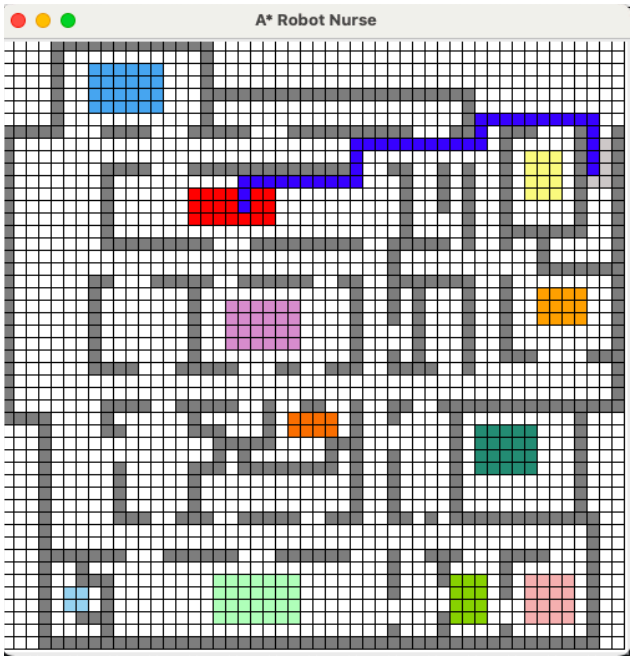
This would result in the following output:

```
The robot is moving from ADM to ONC ......
The robot is moving from ADM to ONC ......
Robot can not access ONC , need HELP!
```

The code below displayed this output for the inaccessible ward:

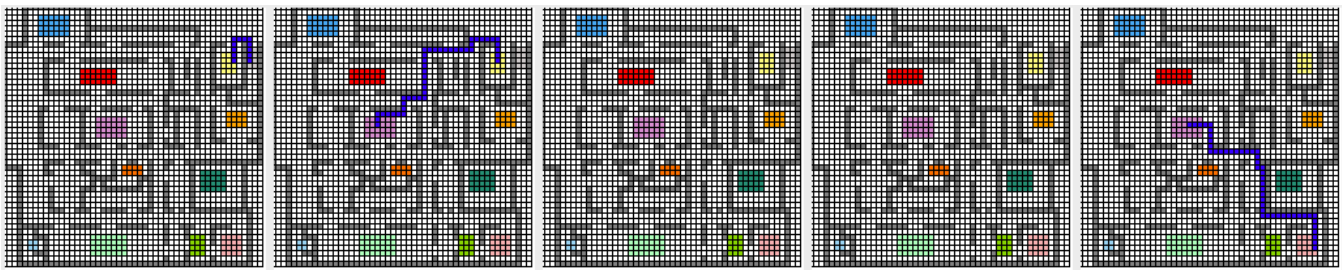
```
if goal_pos != floorSet(root).find_path():
   unaccessed_list.append(goal_pos)
   print("Robot can not access", floorSet(root).ward_name(goal_pos), ",need HELP!\n")
   continue
```

The program continues without having to exit out of the pop-up window and the robot traverses to the next location in the priority order, skipping the ward that was inaccessible. The current inaccessible ward pop-up window and subsequent accessible ward pop-up window will be displayed simultaneously. Below is the map that would display the successful path, Admissions (ADM) to General Ward (GEN) simultaneously with the map above from Admissions (ADM) to Oncology (ONC).



For version 1, the staff at this department will close the window for the robot to continue to its next location. For version 2, all grid displays will show up simultaneously to display the summary of all paths. Closing the window in version 2 will exit the window without the additional features of progressing the robot as all tasks have been processed. Below is the image summary of version 2 from the following input:

```
Enter the locations you want to deliver the medication:
Start location:ADM
Delivery locations:ER,BUR,ONC,SUR
```



If the robot is not able to complete any of its tasks, such as starting at an inaccessible ward and the robot attempts to traverse to an accessible ward, the following text will be displayed: “No Successful delivery!”. This code was implemented as an else statement in the performance summary code when displaying the final notification output. (This code is mentioned below in the additional features section).

Error Correction

Error inputs such as misspelled words or wards that did not exist were corrected through error detection. Any input that did not match the wards already identified would create a KeyError output. with the following text, "`KeyError: 'input' input is not a department name. Check your input!`" Below is the code that implemented this output.

```
if l0[0] not in wards.keys():
   print(l0[0], 'is not a department name. Check your input!\n')
   break
```

Additional Features

Following the final delivery location the robot processed, an output text displays:

```
----PERFORMANCE SUMMARY----
SUCCESSFULLY delivered:
ER SUR GEN
FAILED to access:
ONC
```

Under "`SUCCESSFULLY delivered:`" are the wards where the delivery was successful and in the order they were processed. Under "`FAILED to access:`" are wards where the delivery was unsuccessful.

```
#### Update the final notification
print('----PERFORMANCE SUMMARY-----')
if len(deliveried_list) != 0:
    print("\nSUCCESSFULLY delivered: ")
    for pos1 in deliveried_list:
        print(get_ward_name(pos1),end= ' ')
else:
    print("\n No Successful delivery!")
print("\n")
if len(unaccessed_list) != 0:
    print("\nFAILED to access:")
    for pos2 in unaccessed_list:
        print(get_ward_name(pos2), end= ' ')
else:
    print("\n No Failed delivery!")
```

Time stamps were used for our robot's output to demonstrate the time between each start and goal location. This was estimated by multiplying 0.2 by each grid block the robot traversed. We thought this could be adjusted depending on the speed and space of steps the robot takes. Below is the code that was implemented for the output, "`Estimated time: 2 minutes.`"

```
print("Estimated time:", int(self.count * 0.2), "minutes.")
```

Ward identification was used to easily identify ward locations based on their matrix coordinate. Each ward position was given its designated name that can be inputted by the user and displayed as output text. For example:

```
(22, 44): "ICU"
```

Text output was displayed where the robot was during each part of the process. Below is the code that was implemented for the output, "`The robot is moving from ADM to ER ......`".

```
print("The robot is moving from", self.ward_name(self.agent_pos), "to", self.ward_name(self.goal_pos), "......")
```

Statement of ranking

Member 1: Phu Do
My teammate and I agree that I handled **50%** of the overall project. My specific tasks included:

Task 1: I designed and implemented the program module that assigned the priority queue to each ward.
Task 2: I designed and implemented the program module that displayed the map matrix.
Task 3: I documented the Python code.
Task 4: I designed and implemented the program module that displayed the time estimates for the robot.
Task 5: I designed and implemented the program module that displayed time delays and the output summary for the robot.

Member 2: Katrina Cwiertniewicz
My teammate and I agree that I handled **50%** of the overall project. My specific tasks included:

Task 1: I designed and implemented the program module that displayed the color index for the matrix map.
Task 2: I tried integrating different ways to display and add text to the matrix map.
Task 3: I researched ways to try and solve errors with the priority queue functionality.
Task 4: I wrote up a summary of our project to present.
Task 5: I wrote the report and summarized the functionalities of our project.