INTRODUCTION

Running a restaurant smoothly is not just about serving tasty food. Efficient restaurant operations are paramount for ensuring customer satisfaction and profitability. One critical aspect of restaurant management is the optimization of the flow of orders from the kitchen to the customers' tables.

In today's fast paced world, where diners expect prompt service and seamless experiences, the role of technology in streamlining restaurant operations has become increasingly important. Restaurants might consider replacing the waiters with autonomous agents, which are responsible for delivering the orders to the customers in minimum time in order to improve the customer experience.

The core idea of this project is to simulate an environment to analyze the impact of the number of bots and different pathfinding algorithms on customer waiting times in a restaurant setting. Once a customer places an order, the bots are responsible for collecting the food from the kitchen and delivering it to the customer. Different path finding algorithms such as A*, Dijkstra's algorithm and the Depth First Search (DFS) have been implemented to find the shortest path for the bot, while avoiding collision with obstacles and other agents. The analysis is carried out for different number of bots for each of the mentioned algorithms in order to find out the optimal combination.

RESEARCH QUESTION

The question under consideration is: How do the number of bots and the choice of path finding algorithm impact the average customer waiting time in a restaurant environment.

RELATED WORK

The paper "Food Delivery Automation in Restaurants Using Collaborative Robotics" by Albin and Antony and P.Siviraj [1] proposes a centralized system that coordinates a team of robots in a restaurant to streamline food serving processes. The system employs a selection algorithm to optimally assign tasks to different robots based on factors like distance, time and energy. Additionally, a modified a-star path finding algorithm ensures the efficient robot navigation while avoiding obstacles. The emphasis on task optimization through selection algorithms aligns with our study's focus on efficient resource allocation. Moreover, the use of A* path finding algorithm for navigation resonates with our exploration of path finding algorithm's impact on customer waiting times.

The paper titled "Items-mapping and route optimization in a grocery store using Dijkstra's, Bellman-Ford, and Floyd-Warshall Algorithms" [2] explores the application of different algorithms to find the optimal shopping routes in grocery stores. The study assists customers in efficiently locating items using the computed shortest paths displayed on an android application. The algorithms are evaluated based on the path length and computational time. The paper

concludes with some recommendations for enhancements such as database expansion and real-time user tracking. While this study differs in setting, the core objective of optimizing customer experiences by minimizing traversal times aligns with our investigation.

The paper “Optimal and Efficient Path Planning for Partially-Known Environments” [3] addresses the challenge of trajectory planning for mobile robots in partially known environments, where the robot lacks complete information about its surroundings. Unlike some other methods that sacrifice the optimality or computational efficiency, this paper introduces an algorithm D* which is capable of planning paths in unknown or changing environments efficiently and optimally. Although our study focuses on a restaurant setting, the principles of adaptive path planning resonate with our exploration of pathfinding algorithm’s suitability in dynamic environments.

TECHNOLOGIES USED AND CHALLENGES

The restaurant layout has been implemented from scratch using the python library Tkinter. The matplotlib library was used to plot the graphs and charts for the analysis of the customer waiting time.

The initial worksheets about the intelligent vacuum cleaner were used as a guide to write the code for the bots in the restaurant. The basic functions for the movement of the robot have been adapted from the worksheets.

The main challenges were:

1. Optimizing the robot movement in the restaurant environment.
2. Handling the interaction of bots with customers and other bots.
3. Handling the slow movements in the Tkinter window due to numerous updates and interactions overwhelming the event loop, hindering smooth GUI responsiveness.

ENVIRONMENT AND AGENT DESIGN

The simulation environment consists of passive objects like tables, kitchen, doors and charging station. The agents used are customers, robots and the manager.

Restaurant Layout

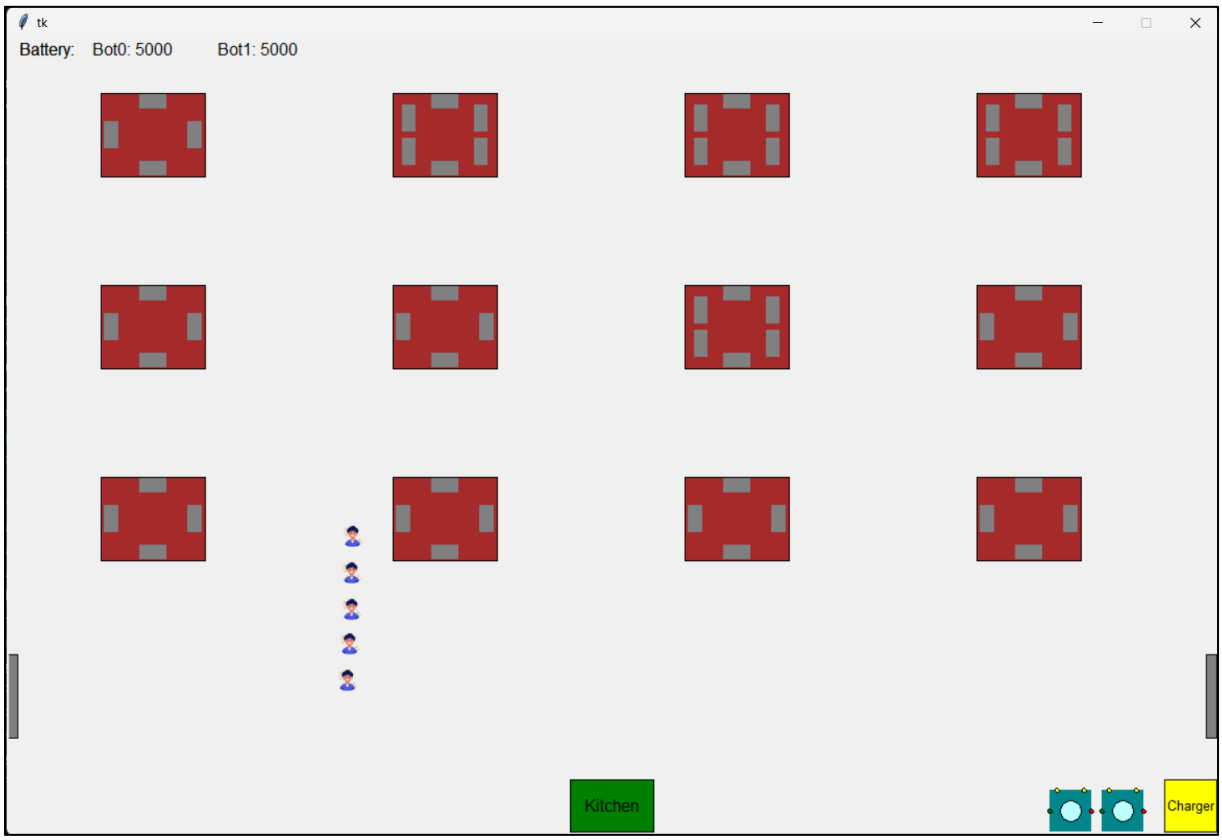
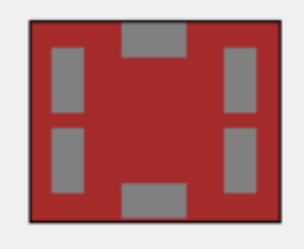
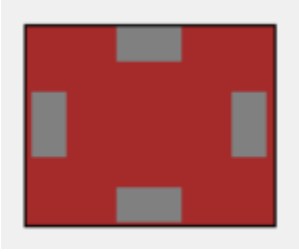


Figure 1. Restaurant Layout

The design of the restaurant environment can be seen in figure 1. The restaurant is represented by a 1160*760 pixels window. The kitchen (80*50), charger (50*50) and the doors (10*80) are represented by green, yellow and gray areas respectively. Tables are represented by the brown rectangles of size 100*80. The number of table mats shows the capacity of each table as shown in figure 2.



(a) Table capacity = 6



(b) Table capacity = 4

Figure 2. Table Capacity

Figure 3 shows different states of the table with and without customers. The number of customers sitting at a table is displayed on the table top.

(a) Table without customers (b) Table with customers

Figure 3. Table State

Agents:

The different agents present in the environment are:

1. Robots:
   Each robot is represented by a polygon of size 40*40 and has two kinds of movements. It can either rotate at its position or move forward. The robots are responsible for collecting the food from the kitchen and delivering it to the customer table. The battery capacity of each robot is 5000 as shown at the top left of the figure 1. As soon as the battery drops below 1000, the robot top changes color to red and goes towards the charger in order to recharge the battery. Different states of the robot can be seen in figure 4.

(a) Original (b) Low Battery (c) Carrying Food

Figure 4. Robot State

2. Customers  :
   Customers arrive at the restaurant through any of the two doors at random intervals. The customers arrive in the form of groups and select a random table depending on the capacity of the table. Each customer group can have 1 to 6 customers. Once the customers sit at the table, they place an order (suppose there is an online ordering facility at the table). After the order is delivered to them by the robot, they leave the restaurant through the door they entered. The size of each customer is 20*20.

3. Manager:

The manager receives all the customer orders and assigns responsibilities to the robots. Whenever an order is received, the manager assigns the order to the robot that has the least number of pending orders. Incase two robots have equal number of pending orders, then the robot that will be closest to the kitchen after finishing those orders will be assigned the new task.

SYSTEM DESIGN

Obstacle avoidance and path navigation were implemented in the system to optimize the food delivery process.

Obstacle Avoidance

1. Obstacle avoidance in customers:
   Whenever, a customer collides with some object or agent, it turns by an angle between 25 and 45 degrees towards the target. The customer keeps on turning until it avoids the collision.

2. Obstacle avoidance in robots:
   The robot uses different path finding algorithms to find the path towards the target. These algorithms make sure to find a path that avoids all the passive objects in the restaurant layout.
   Incase, the robot collides with another bot, the path is recalculated to avoid that collision. For collision with customers, the bot stops until the customer himself moves out of the way.

Path Navigation

Four different path finding algorithms are used to find the optimal path for the robots. Whenever a robot is assigned a task, it calculates the path to the target object (table, kitchen or charger) while avoiding all the obstacles. Incase the robot collides with another bot while following that path, it recalculates the path to avoid that robot.

The window is divided into a 29 by 19 grid with each square of size 40*40. The grid cells to avoid are calculated using the position and dimensions of the obstacles. Then the start and target cells along with the cells to avoid are provided as parameters to the path finding algorithms. Figure 5 marks the area around the table being avoided by the bot.

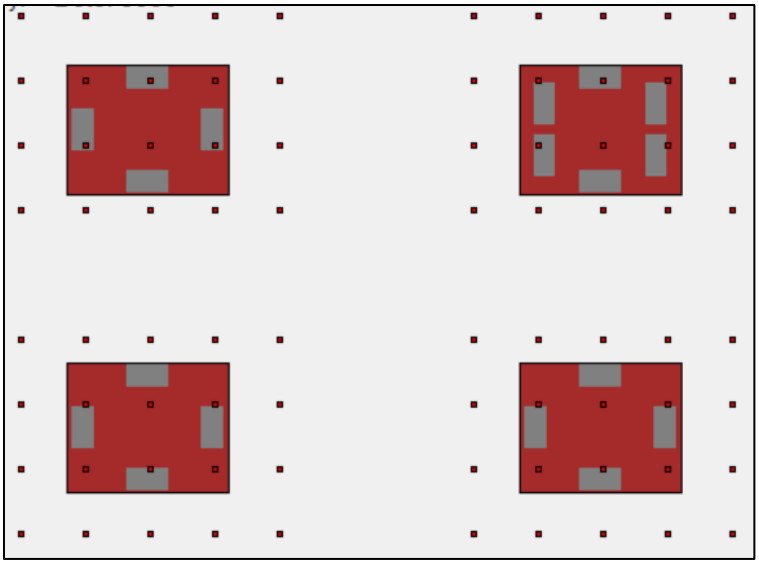


Figure 5. Grid cells to avoid

The algorithms used to find the path are as follows:

1. Depth First Search (DFS)
    a. Select the starting point.
    b. Get all the neighboring cells (including diagonals) for the starting point.
    c. For each neighbor:
        i. If the neighbor is not in the cells to avoid and is not the target:
            1. Recursively apply DFS to the selected neighbor.
    d. If the selected cell is the target cell, return success.
    e. If all neighbors have been explored and the target cell is not found, backtrack.
2. Breadth First Search (BFS)
    a. Enqueue the starting point into a queue.
    b. While the queue is not empty:
        i. Dequeue a node from the queue.
        ii. If the dequeued node is the target node, return success
        iii. If the dequeued node has been visited, go to step i, else mark it as visited.
        iv. For each neighbor (including diagonals) of the dequeued node:
            1. If the neighbor is a valid move and is not in the cells to avoid, enqueue it into the queue.
    c. If the target node is not found after exploring all reachable nodes, return failure.
3. Dijkstra's Algorithm
    a. Initialize a priority queue to store vertices along with their tentative Manhattan distances from the start vertex. Set the tentative distance of the start vertex to 0 and all other vertices to infinity.
    b. While the priority queue is not empty:
        i. Dequeue the vertex with the smallest tentative distance from the priority queue.

ii. If the dequeued vertex is the target, return the shortest path by following the vertex it came from.
iii. For each neighbor (including diagonals) of the dequeued vertex:
1. If the neighbor is not in the cells to avoid:
a. Calculate the tentative distance from the start vertex to the neighbor via the dequeued vertex.
b. If this tentative distance is smaller than the current tentative distance of the neighbor, update the neighbor's tentative distance.
c. Enqueue the neighbor with its updated tentative distance into the priority queue.

c. If the distance vertex is reached or all vertices have been explored, terminate the algorithm.

4. A* Algorithm
a. Initialize an empty priority queue to store vertices along with their f-scores (combination of the cost to reach the vertex from the start plus the estimated cost to reach the target). The heuristic used to calculate the cost to the target is the Manhattan distance.
b. Initialize dictionaries to store the current shortest distances from the start vertex to each vertex (g_score), and the estimated f-scores (f_score), with the start vertex having a distance of 0 and f-score calculated using a heuristic function(Manhattan distance).
c. Initialize an empty set to store visited vertices (closed_set).
d. Enqueue the start vertex into the priority queue with its estimated f-score.
e. While the priority queue is not empty:
i. Dequeue a vertex with the smallest f-score from the priority queue.
ii. If the dequeued vertex is the target vertex, reconstruct and return the shortest path using the came_from dictionary.
iii. For each valid neighbor of the dequeued vertex: If the neighbor is not in the cells to avoid:
1. Calculate the tentative distance from the start vertex to the neighbor via the dequeued vertex.
2. If the neighbor is in the closed set and the tentative distance is not better than the existing distance, skip to the next neighbor.
3. If the tentative distance is better than the current distance, update the neighbor's distances and enqueue it with its updated f-score into the priority queue.

f. If the target vertex is not found after exploring all reachable vertices, terminate the algorithm with no path found.

EXPERIMENT SETUP

Experiments were conducted by varying the number of bots and comparing different path finding algorithms while monitoring the customer waiting times. The waiting time includes the time after the arrival of the customer group until a table is assigned and the time taken to deliver the food to the customer after the order is placed.

In order to carry out the experiments, a simulation function was used to run the restaurant environment without displaying the GUI. This helped to conduct the experiments in an efficient manner.

1. The restaurant simulation was run for 300 seconds each time and the average customer waiting time was returned.
2. The simulation was run 10 times for each parameter combination and the average waiting times for each iteration were recorded to get a more accurate result.
3. Each of the above mentioned path finding algorithms was used one by one.
4. The number of bots were varied from 1 to 5 for each path finding algorithm.
5. The results were recorded in an excel sheet and box plots were created to compare the results.

The pseudocode for the experiment is as follows:

```
def runExperiments(duration, numberOfRuns, numberOfBots, pathFindingAlgo):
     waitingTimes = []
     for in range(numberOfRuns):
          waitingTimes.append(simulation(numberOfBots, pathFindingAlgo))
               return waitingTimes



def runExperimentWithDifferentParameters(duration,maxNoOfBots,numberOfRuns,pathFindingAlgo):

     resultsTable = {}
     for numberOfBots in range(1, maxNoOfBots + 1):
          waitingTimeList=runExperiments(duration,numberOfRuns,numberOfBots,pathFindingAlgo)
          resultsTable.add(waitingtimeList)

     resultsTable.toExcel()
     boxplot(resultsTable)



 for algo in ['astar', 'dijkstra', 'bfs', 'dfs']:
     runExperiments(300, 5, 10, algo)
```

EXPERIMENT RESULTS AND DISCUSSION

The following customer waiting times were obtained for each of the path finding algorithms. These show how the number of robots affects the waiting time of the customers for each algorithm.

	robots: 1	robots: 2	robots: 3	robots: 4	robots: 5
0	6260.6	6955.526	5205.731	5014.789	4909.808
1	7147.6	6576.143	5205.731	4961.292	4909.808
2	6708.8	6419.455	5205.731	4961.292	4869.338
3	6568.667	6121.49	5205.731	4961.292	4817.6
4	6408.263	5996.608	5205.731	4961.292	4817.6
5	6674.696	5831.944	5151.118	4961.292	4817.6
6	6832.179	5567.517	5061.386	4961.292	4770.224
7	7078.75	5373.081	5061.386	4909.808	4770.224
8	7182	5276.063	5014.789	4909.808	4770.224
9	7102.649	5205.731	5014.789	4909.808	4723.987

(a) Depth First Search

	robots: 1	robots: 2	robots: 3	robots: 4	robots: 5
0	3143.043	2629.466	2220.342	2091.511	2043.976
1	2682.947	2585.843	2211.132	2087.886	2031.684
2	2576.167	2545.088	2186	2083.671	2023.092
3	2810.518	2502.816	2170.662	2078.962	2018.825
4	2841.738	2439.221	2170.662	2074.403	2014.651
5	2738.591	2411.084	2165.28	2074.403	2003.225
6	2599.496	2341.293	2146.319	2065.14	1995.005
7	2532.177	2318.248	2141.718	2056.816	1987.639
8	2907.349	2311.66	2100.239	2048.431	1983.418
9	2821.821	2226.442	2095.871	2048.431	1983.418

(b) Breadth First Search

	robots: 1	robots: 2	robots: 3	robots: 4	robots: 5
0	3401.583	3097.311	2135.204	1837.348	1682.468
1	3685.948	3041.942	2085.589	1811.787	1673.008
2	3433.747	2971.832	2038.612	1784.536	1660.32
3	3395.7	2915.02	2003.521	1774.319	1651.052
4	3494.92	2756.065	1983.505	1763.559	1636.357
5	3443.444	2597.566	1968.468	1753.385	1630.036
6	3313	2516.475	1938.738	1748.515	1620.362
7	3150.355	2315.981	1896.954	1721.707	1613.864
8	3094.925	2271.705	1882.96	1695.642	1607.418
9	3135.004	2244.913	1851.398	1689.485	1596.126

(c) Dijkstra's Algorithm

	robots: 1	robots: 2	robots: 3	robots: 4	robots: 5
0	919.75	2382.386	1742.434	1499.821	1385.795
1	2313.343	2329.105	1705.27	1469.372	1378.686
2	2013.082	2207.677	1679.367	1459.283	1363.209
3	2411.395	2126.209	1644.022	1438.269	1355.717
4	2283.905	2067.509	1591.059	1430.854	1347.912
5	2235.722	1993.472	1581.434	1420.142	1340.075
6	2215.448	1957.419	1561.168	1412.029	1327.947
7	2516.848	1850.673	1540.27	1407.36	1322.674
8	2561.157	1810.023	1533.446	1400.047	1311.971
9	2489.419	1756.345	1514.555	1390.771	1291.287

(d) A* Algorithm

Figure 6. Customer Waiting Times for each algorithm

EFFECT OF THE NUMBER OF ROBOTS

In order to observe the effect of the number of robots on the customer waiting time, the following boxplots were plotted.

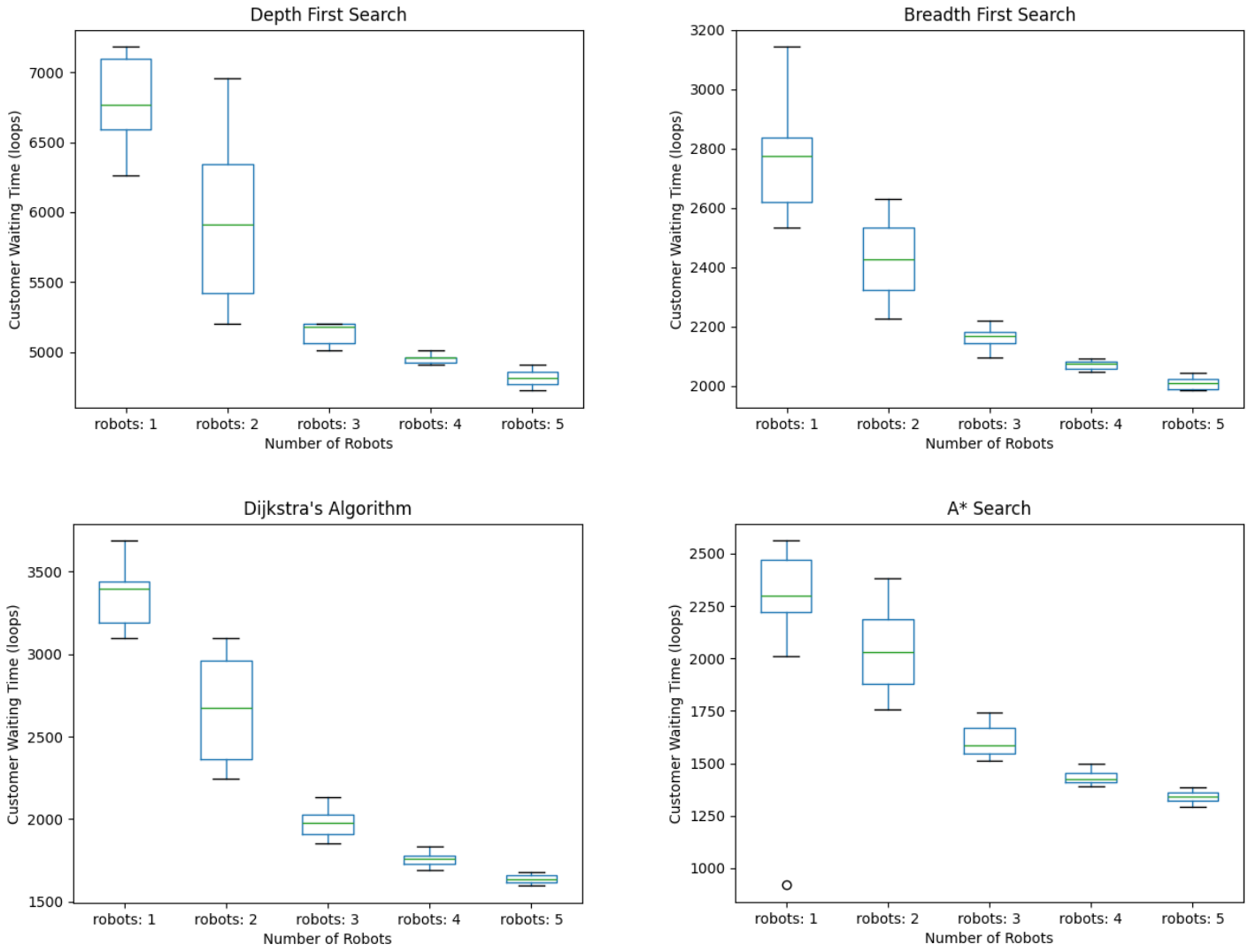


Figure 7. Boxplots for waiting time vs number of robots

From the above boxplots, we can see that the waiting time is decreasing as the number of robots increases. This shows that as the number of robots increases, the work is divided between the robots. Therefore, the customer waiting time is reduced as they receive their order earlier. Also, since the previous customers are served in less time, the new customers don't have to wait a lot for the tables to get free.

We also observe that the effect of the number of robots decreases as we continue increasing the bots. For example, we can see that the waiting time is greatly decreased as we increase the number of bots from 1 to 2. However, the difference in the waiting time is not as big as we move from 4 to 5 bots. This shows that increasing the number of robots will not always help in reducing the waiting time. After a certain number, increasing robots will not help since there might not be as many customers in the restaurant to engage all those bots.

Another observation is that the variance in the waiting time decreases with the increase in number of robots. This shows that for fewer robots, the customer waiting time varies a lot. The customers arriving first might not have to wait as much as the customers arriving late. This is because the robots are free when the first few customers arrive. However, after that they get busy and take more time to attend the customers arriving late. For larger number of robots, the previous customers are served faster therefore the new customers don't have to wait a lot for the previous ones to leave. Thus, the waiting time does not keep on increasing and most of the customers have a similar waiting time. This can also be observed in the following line graph.

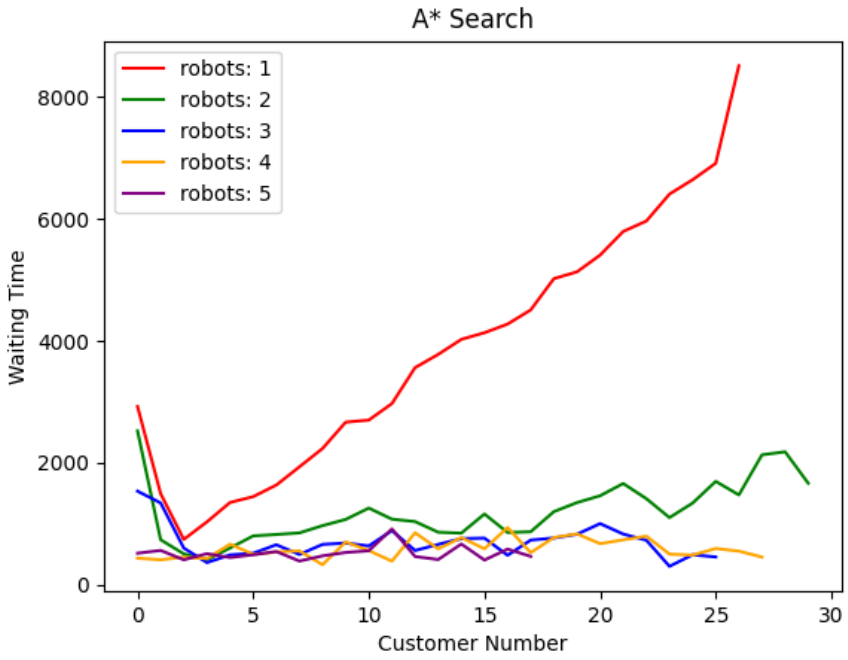
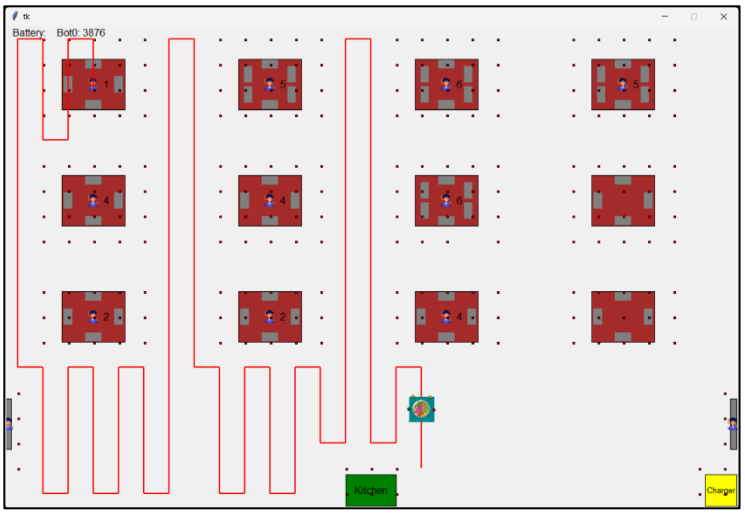


Figure 8. Plot for Customer waiting time vs Customer number (A* Search)

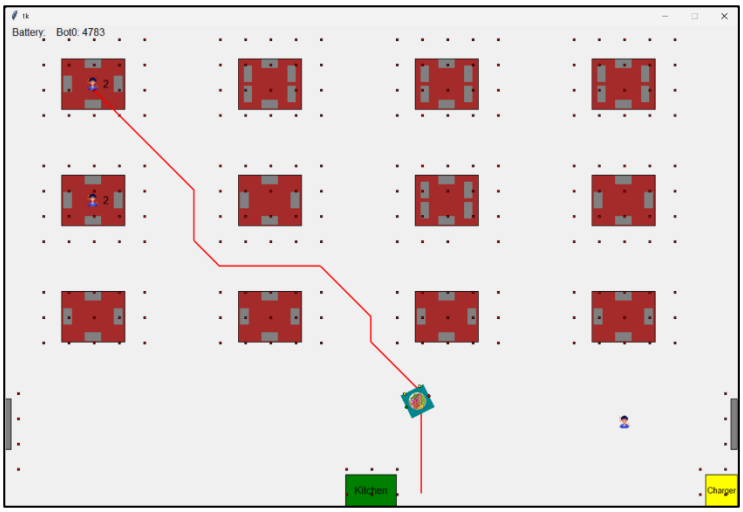
The above plot shows the affect on customer waiting time with each successive customer served. There is a rapid increase in waiting time in case of 1 robot. Incase of 2 robots, the increase in the waiting time with each successive customer is very small. However, for 3 or more robots, the customer waiting time stabilizes and there is no notable increase. This shows that increasing the number of robots helps in reducing the customer waiting time. However, after a certain number of robots, this does not help and increases the robot cost only. Thus, we need to decide the optimal number of robots required for a restaurant setup which is 3 in this case.

EFFECT OF THE PATH FINDING ALGORITHM

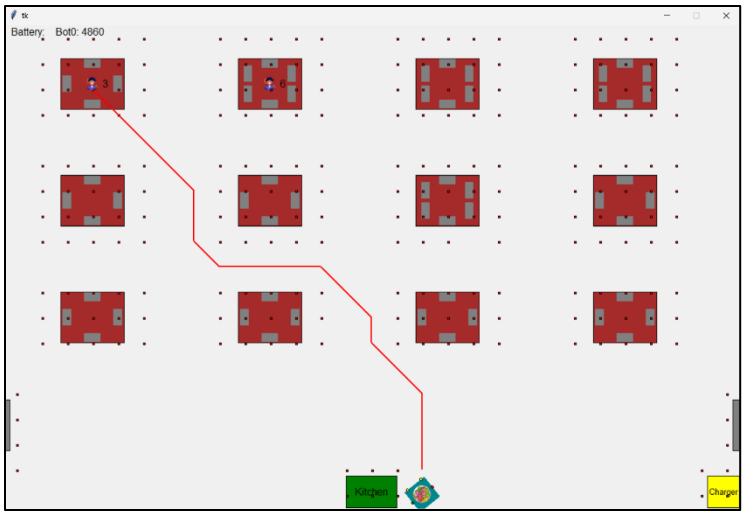
The following paths from the kitchen to a certain table were obtained for each of the algorithms used.



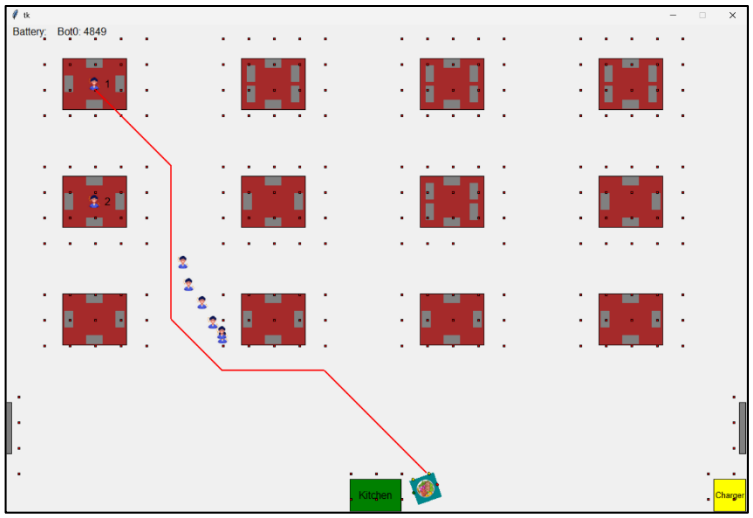
(a) Depth First Search



(b) Breadth First Search



(c) Dijkstra's Algorithm



(d) A* Algorithm

Figure 9. Robot path using different algorithms

In figure 9a, we can see that the path using Depth First Search is very long. It can easily be assumed that this is not the shortest path and will surely waste a lot of time. Therefore, the customer waiting time will be very high compared to other algorithms. This can be confirmed by the waiting times shown in figure 6 and figure 7. We can see in the figure 7, that the waiting time for DFS goes as high as 7000, while the time for other algorithms never goes this high.

The paths using BFS and Dijkstra's algorithm are very similar. Both are looking for the shortest path to the table. However, there appears to be more turns in the path for BFS and Dijkstra's algorithm. From figure 6 and 7, we can see that both these algorithms have a similar range of

customer waiting times. Although, the boxplot shows a higher maximum for dijkstra's algorithm, we can assume that is because of the randomness of the customer arrival and not because dijkstra's algorithm causes more waiting time. We can make this assumption because although Dijkstra's algorithm has higher waiting time for 1 robot, the waiting time for 5 robots is less than that of BFS. Thus, the higher maximum waiting time for dijkstra's algorithm is purely because of randomness of the arrival time.

The A* algorithm has a different path to the other 3 algorithms. It finds the shortest path based on cost (Euclidean distance) and a heuristic measure that is the Manhattan distance. Figure 6 and figure 7 show that A* algorithm causes the least waiting time for the customers. For any number of robots, the customers have the least waiting time when using A* algorithm to find the shortest path. Thus, it is the most optimal path finding algorithm for the restaurant environment.

CONCLUSION

The aim of this project was to analyze the impact of the number of bots and different pathfinding algorithms on customer waiting times in a restaurant setting. Through a number of experiments in a simulation environment, several key findings have emerged.

The experiments revealed a clear correlation between the number of robots and customer waiting times. As the number of robots increased, the waiting time decreased significantly. However, it was observed that beyond a certain threshold, increasing the number of robots had diminishing returns, indicating the importance of finding an optimal balance. Another observations was that as the number of robots increases, the increase in the waiting time for each successive customer decreases.

Moreover, the customer waiting time was also affected by the choice of the path finding algorithm. The Depth First Algorithm had the worst results as it did not produce the shortest path. The A* algorithm, Breadth First Search and Dijkstra's Algorithm produced similar results by generating the shortest paths. A* algorithm was found out to be the most optimal solution. This finding emphasizes the importance of selecting an appropriate algorithm for optimizing the restaurant operations.

The above mentioned results suggest that a combination of optimal number of robots and an efficient path finding algorithm is necessary to enhance the restaurant operations by minimizing the customer waiting times and improving overall efficiency. However, further research could explore additional factors such as restaurant layout optimization and dynamic adaption of the algorithms to real-time conditions.

FUTUURE WORK

The further work on this project will focus on managing more tasks for the robots like cleaning, guiding the customers etc. The future work will also include further improvement in the path finding algorithms. Currently this report covers four path finding algorithms. Some other algorithms can also be explored in the future.

REFERENCES

[1] Antony, A., & Sivraj, P. (2018, July 11-12). Food Delivery Automation in Restaurants Using Collaborative Robotics. 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India. https://ieeexplore.ieee.org/document/8597280

[2] Dela Cruz, J. C., Magwili, G. V., Mundo, J. P. E., Gregorio, G. P. B., Lamoca, M. L. L., & Villasenor, J. A. (2016, November 22-25). Items-mapping and route optimization in a grocery store using Dijkstra's, Bellman-Ford and Floyd-Warshall Algorithms. 2016 IEEE Region 10 Conference (TENCON). Singapore. https://doi.org/10.1109/tencon.2016.784799

[3] Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation (pp. 3310-3317). Volume 4. doi: 10.1109/ROBOT.1994.351061. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=351061.