Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

# MEU44B12 Introduction to Autonomous Mobile Robotics

## Individual Robotics Project

**Name:** Shirish Senthil

**Student ID:** 21349979

**Date of Submission:** 21/04/2023

## Table of Contents

## Abstract

This project describes the use of A* path planning, an inertial measurement unit (IMU) sensor, and a GPS sensor to navigate through a maze from an initial position to the final goal. The robot is set up as a differential robot with two wheels on the software Webots. **Webots 2023a** is used to simulate and test out the robot controller that contains the robot code. The programming is done with Python 3.11 for its extended capabilities in the region of data wrangling, manipulation, and numerical calculations.  The robot is also set up with a keyboard to take in user input. With the addition of enumerations, the robot shows the path and directions that the bot had taken on the console. Furthermore, 2 maps are explored, one known and one unknown. These occupancy grids are fed as ".CSV" files. The starting point and final goal are predetermined. With the use of rotational motors, the robot moves forward and orients itself towards the goal. Further, the different parameters considered when coding the robot's logic are discussed in this report. The use of different helped functions that enable the A* path planning is explained in this report. Furthermore, the main program and the detailed flow of the code are highlighted. Future developments in this field and on the robot are suggested at the end of the paper with a conclusion on the individual project. PyCharm was used to code the Python as it is easy to work with and user-friendly.

## Introduction

The development of autonomous capabilities in robots and systems is moving quickly and is in great demand. The uses of this AI technology range from rovers and unmanned vehicles for exploration to autonomous marine robots that operate underwater. The "Roomba robot" is one example of a commercially available autonomous robot that uses a navigation beacon, cameras for localization, and room mapping to enable continuous cleaning of indoor spaces. [1]. There is a serious demand in this technical and Mazebots have proved to be advantageous in various real-world applications such as security and agricultural operations where it is hard for humans to traverse and reach.

There are numerous methods by which a bot can navigate a maze, utilizing a combination of different sensors and flowcharts to achieve the intended goal. Based on the sensors, a flowchart can be constructed to assist the robot in mapping and navigation. Some of the sensors that can aid in the navigation of a maze are Lidar, Odometric sensors (IMU), and GPS to name a few. Using the real-time data relayed by these sensors, the Mazebot can correct itself and reroute itself to the end goal.

In this report, the A* path planning is selected for numerous reasons. Developed at Stanford University, it finds the shortest path that can be found from the predetermined starting and endpoints, using less computational power. Since it implements a heuristic search approach, it is a major advantage of A star as it can find a solution in a solution space if one exists. This along with its balance of accuracy and efficiency makes it a preferable choice for autonomous navigation and maze-solving. [2]

Webots 2023a is chosen as the software to develop the robot. There are a multitude of factors for this selection. Primarily, it is less computational than existing supercomputers that are used, while delivering a rather accurate simulation and physics of what the robot may experience. It is an open-source software that is constantly being updated with the latest features. It is compatible with many operating systems and has a very user-friendly graphical user interface. Its simplicity in integrating with programs like MATLAB is also favorable.

The objectives of this study are to design and develop a simulation of a mobile robotic platform that is tasked with autonomously navigating a maze using a 2D navigation occupancy grid. The performance of the robot is determined by how far the robot can traverse the maze in 2 minutes.

There are numerous libraries, modules, functions, and methodologies used in this program, and each one of them has a separate usage and is vital to the success of the robot.

## Literature Review

There are numerous path planning approaches that are being used like A* Algorithms, D* Algorithm (Heuristic method), Dijkstra's Algorithm (Deterministic method), Cell Decomposition Technique, etc. [3] The principal concept behind this project is the use of conventional A* path planning. Path planning is referred to as the autonomous robot's capacity to analyze in a fast-paced dynamic environment or static which allows the robot to achieve its objectives and reach the goal.

The application of this path planning is recommended for obstacle avoidance. [4] Initially, Peter E. Hart et al. put forward the concept of A* planning. This was with a heuristic approach and these values increased the efficiency of calculating the shortest possible path between the start and goal. German A Vargas et al. demonstrated the usage of A* planning on Webots, the open-source software. [5] The process is mainly used in grids for easier calculation and visualization of the path.

A* path planning has three main metrics

1. F(n) = The minimum value of the cell entry estimated by all paths starting node (S) to the goal node (G)
2. G(n) = The function regarding the real cost of the starting node of the robot
3. H(n) = The indicative estimate of the minimum path costs from the current node to the target node

These three metrics can be represented as a simple formula shown below. [6]

$$F(n) = G(n) + H(n)$$

The algorithm of A* planning can be represented using a flowchart. Fig. 1 shows how the robot uses conventional algorithm and heuristic values to find the shortest path from start to goal. [7]

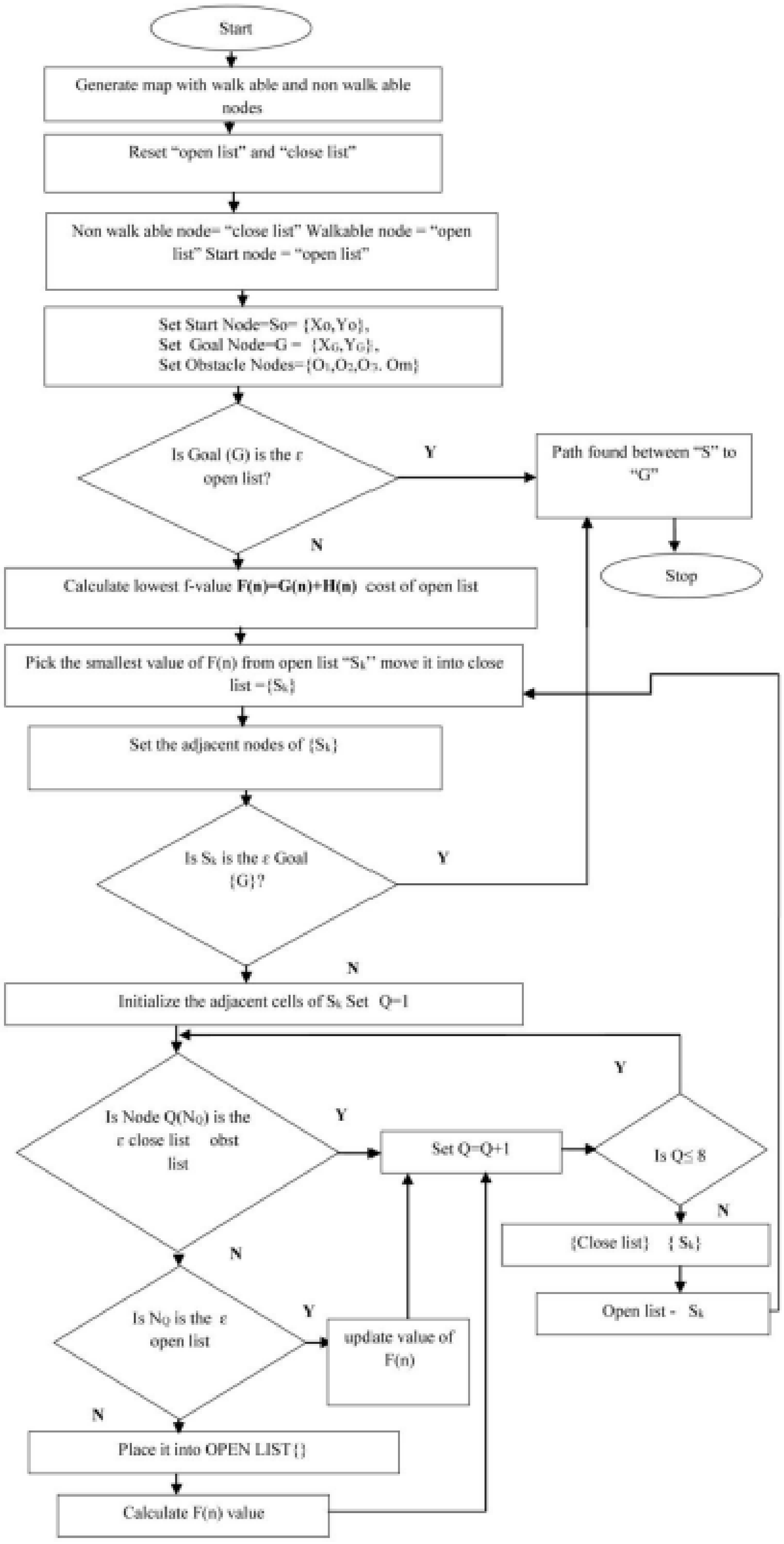

Start
Generate map with walk able and non walk able nodes
Reset "open list" and "close list"
Non walk able node= "close list" Walkable node = "open list" Start node = "open list"
Set Start Node=So= {Xo,Yo}, Set Goal Node=G = {XG,YG}, Set Obstacle Nodes={O1,O2,O3. Om}
Is Goal (G) is the ε open list?
Y
N
Path found between "S" to "G"
Stop
Calculate lowest f-value F(n)=G(n)+H(n) cost of open list
Pick the smallest value of F(n) from open list "Sk" move it into close list ={Sk}
Set the adjacent nodes of {Sk}
Is Sk is the ε Goal {G}?
Initialize the adjacent cells of Sk Set Q=1
Is Node Q(NQ) is the ε close list obst list
Set Q=Q+1
Is Q≤ 8
{Close list} {Sk}
Open list - Sk
Is NQ is the ε open list
update value of F(n)
Place it into OPEN LIST{}
Calculate F(n) value

Fig 1. Flowchart of A* Algorithm using H(n), F(n) and G(n). [7]

The use of A* Path planning results in a safer means of avoiding obstacles. As shown in Fig 2., the robot can navigate past the obstacles at point 1 and 2. [3] This is in fact the shortest path that the robot can take and is considered as part of the classical A* path planning. Further advancements have been done, to make the navigation safer, by implementing a virtual boundary around the obstacle. In that way, the robot will stay clear of the obstacle before it gets close to the object or wall.

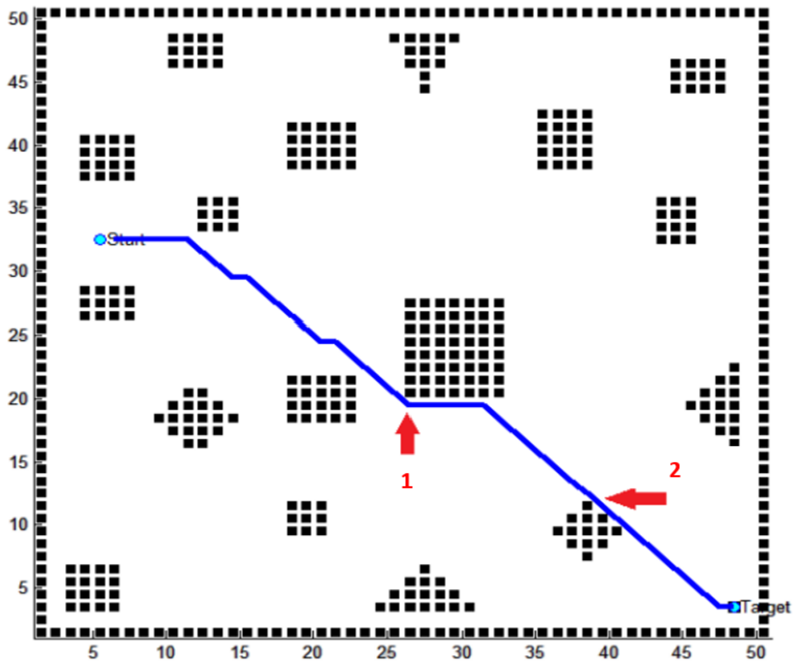


Fig 2. Safe Navigation of the bot using A* path planning [3]

## Technical Development

For the design of the robot, a  basic yet effective differential robot is taken as shown in Fig 3.

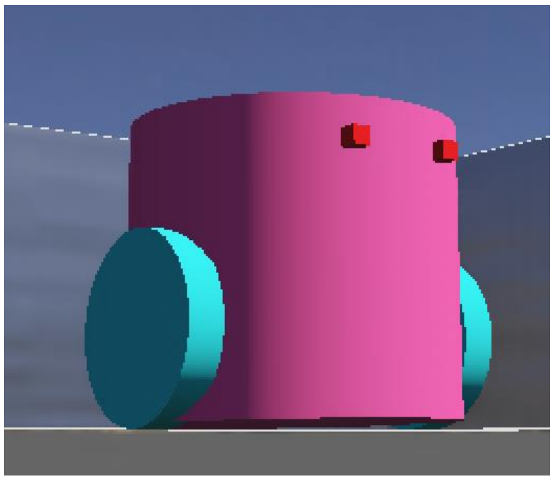


Fig 3. Robot used in the simulation tests

## Parts & Joints used in development

For the body of the robot, a cylinder of radius 0.045 was chosen which is 4.5 cm and the wheel radius was 0.025 which is 2.5 cm. The default unit of Webots is meters. Furthermore, the wheels are connected to the body of the robot by 2 hinge joints for 2 wheels. For propelling the wheels forwards, 2 rotational motors are attached at the wheels, and these are anchored to the body. Additionally, the function of body transform is used to move the entire robot as one single piece around the rectangle arena.

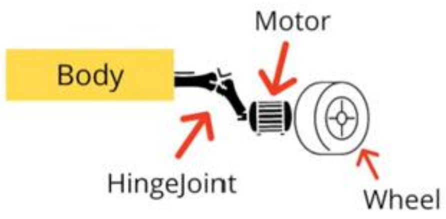


Fig 4. Attaching of the wheel to the body with hinge joint

In the robot, a GPS sensor and an odometric (IMU) sensor were used to full fill the A-star path planning rather than the two GPS sensors calculating the robot pose. The GPS sensor is placed at front and top and since it measures only the x and y coordinates, the height doesn't matter. The same goes with IMU, however, we have placed it dead center of the cylinder's body and it is facing front. This will be explained further in the robot controller section.

## The maze world and occupancy grid

The world used in this project was given by the supervisors and is shown in Fig 5. It is a 50*50 cell grid in the occupancy CSV file. The world is an exact representation of the CSV file. The red dot that can be seen is the starting point of the differential robot and it has to go to the green cross mark at (3, -4). The coordinate x and y axis are shown along with the origin in the top left corner. In the CSV file, the places where there is a wall or obstacle is represented by a "1" and "0" if it is free space.

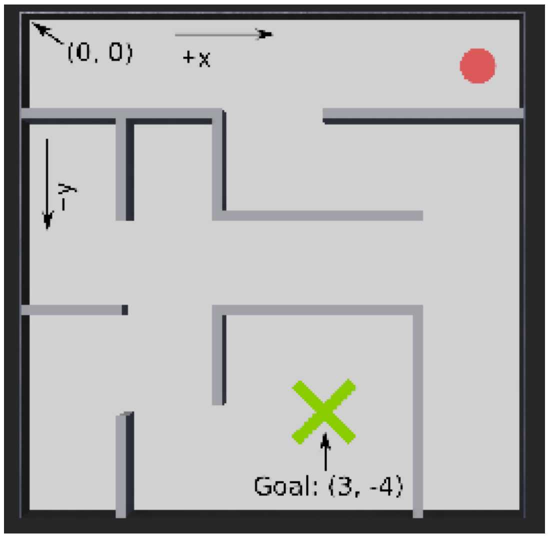


Fig 5. The Maze with starting (Red) and goal (green) locations

## Robot Controller

The robot controller is the code where the A* path planning is implemented. The controller can be broken down into three parts namely, A* path planning code, Enumerations, and robot controls. A* path planning section calculates the shortest path and passes it to the robot controls, which uses features like wheel velocity and position to move the robot according to the path. Enumeration aids in indicating to the user what path the robot has calculated and is going to take. The step-by-step flow of the code:

1. Import necessary libraries and classes.
2. Enumeration classes for direction, state, and orientation for user aid.
3. Define functions to get the next and previous orientation.
4. Read the occupancy grid CSV file of the robot's map.
5. Define a dictionary to store the graph data and create a basic function to set a map value.
6. Define a function for the A* search algorithm to find the shortest path from the source to the target location.
7. Create edges and set heuristics for the graph.
8. Define a function to get the route between the start and goal node.
9. Define a Webots robot instance and enable the GPS, keyboard, and inertial unit.
10. Define motor instances for the left and right motors and set their position and velocity.
11. Initialize values for position, orientation, and state.
12. Enter a loop to control the robot's movement:
    a. Read the keyboard input.
    b. Update the robot's orientation based on the keyboard input.
    c. Move the robot forward or turn it based on its state and orientation.
    d. Update the robot's position based on its movement.
    e. Check if the robot has reached its destination and update its state accordingly

As stated before instead of 2 GPS sensors, a single GPS sensor and an IMU were utilized. The GPS unit gives the x, y, and z coordinates and is used to update the robot's current position. The current position is used to calculate the shortest path to the target position using A*.

The IMU sensor is used instead of another GPS sensor and the orientation data is collected. Pitch, roll, and yaw angles are three ways that the IMU conveys orientation data. The yaw angle taken from the IMU is not used directly but is used to determine the robot's current location. An enumeration of the eight cardinal directions (north, northeast, east, southeast, south, southwest, west, and northwest) is used in this instance to indicate the robot's orientation and is updated using the yaw angle.

## Libraries and Modules

The necessary modules for this project are Webots robot, GPS, Keyboard, and Inertial measurement units. The libraries from the Python standard library are Pandas for reading the CSV, Enum for enumerations, defaultdict to create a new key with a default empty list (avoid 'KeyError'), and PriorityQueue classes to keep track of the nodes that are visited when implementing A* path planning.

## Functions used in the Robot Controller

There are several main and helper functions in this Python code and their individual functionality is mentioned below:

Helper Functions:

1. get_next_orientation(current_orientation): Function that takes the current orientation of the robot and returns the next orientation
.
2. get_prev_orientation(current_orientation): Function that takes the current orientation of the robot and returns the previous orientation.

3. set_map_value(x, y, value): Function that sets the value of a specific cell in the occupancy grid.

4. a_star_search(source, target): Function that performs the A* search algorithm to find the shortest path between two points in the occupancy grid.

5. addedge(x, y, cost): Function that creates the edges between nodes in the graph representation of the occupancy grid.

6. create_edge(): Function that creates the edges between the nodes in the graph representation of the occupancy grid

7. Set_heuristic(): Function that sets the heuristic value for each node in the graph representation of the occupancy grid.

Main Functions:

1. get_route(start, finish): Function that returns the shortest path between two points in the occupancy grid.

2. ‘__main__’ : Main function that defines the robot and its components, and controls its movement.

Finally, the code enters the while look that runs till the simulation ends. The code uses the GPS and IMU sensors to take the robot's current position and orientation during the loop. With the present state and the user's keyboard input, it updates the robot's status. When the robot is idle/stationary, it awaits a new goal node. The robot traverses by altering the left and right motors that have been anchored on the body. By adjusting the motors, the robot advances forward or in any direction till it finds the goal.

## Results

There are a few considerations for the project:

1. The start and goal nodes are taken from the rectangular arena coordinate system; however, the origin is slightly offset, which was accounted for when predetermining the goal and start nodes.
2. There is always a possibility of sensor inaccuracy.
3. The entered coordinates, whether for anchoring or fixing the wheel to the robot body has the possibility to fluctuate due to random vibration errors.

The robot controller was run successfully, and the robot reached its intended goal as shown in Fig 6. The path followed by the robot was tracked on the console. At each critical stage, it was checked if the controller is working as intended by printing the output. For example, when the CSV was read, the array that was appended by rows was outputted.

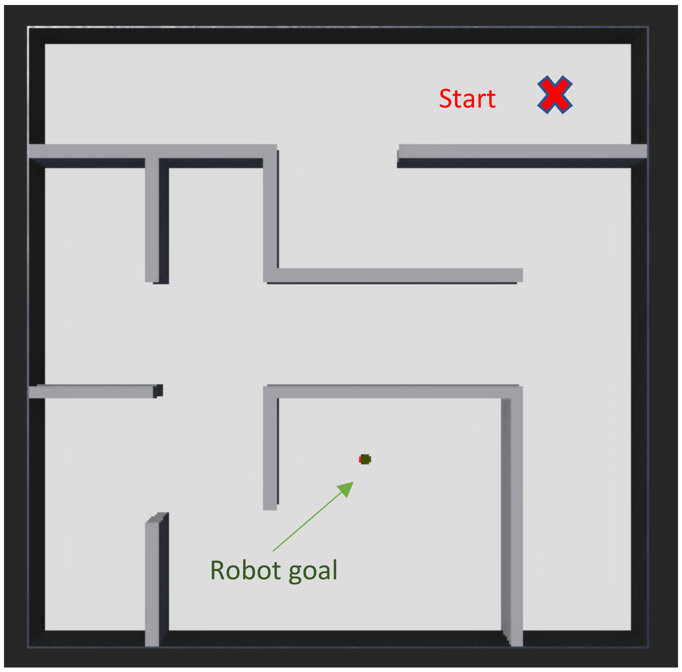


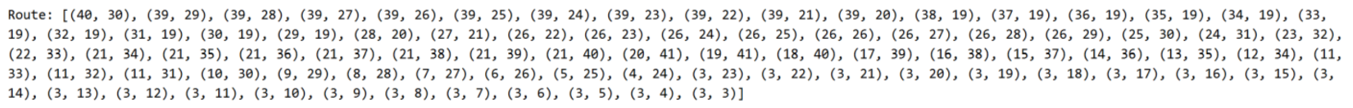
Fig 6. Robot achieving its objective and reached the goal node

The output of the code reading the occupancy grid CSV file can be found in Appendix A.1 of this report. This is just to confirm that the original map is being parsed the same and no discrepancies.

The code is set up to display the total cost after undergoing the A* Path planning and analyzing the occupancy grid. From the console, the total cost to reach the goal node is:

```
Total cost to reach the goal node:  157.2337661840735
```

The path taken by the robot can be seen in Fig 7, starting from (3,3) to (40,30) that is taken from the console.



```
Route: [(40, 30), (39, 29), (39, 28), (39, 27), (39, 26), (39, 25), (39, 24), (39, 23), (39, 22), (39, 21), (39, 20), (38, 19), (37, 19), (36, 19), (35, 19), (34, 19), (33,
19), (32, 19), (31, 19), (30, 19), (29, 19), (28, 20), (27, 21), (26, 22), (26, 23), (26, 24), (26, 25), (26, 26), (26, 27), (26, 28), (26, 29), (25, 30), (24, 31), (23, 32),
(22, 33), (21, 34), (21, 35), (21, 36), (21, 37), (21, 38), (21, 39), (21, 40), (20, 41), (19, 41), (18, 40), (17, 39), (16, 38), (15, 37), (14, 36), (13, 35), (12, 34), (11,
33), (11, 32), (11, 31), (10, 30), (9, 29), (8, 28), (7, 27), (6, 26), (5, 25), (4, 24), (3, 23), (3, 22), (3, 21), (3, 20), (3, 19), (3, 18), (3, 17), (3, 16), (3, 15), (3,
14), (3, 13), (3, 12), (3, 11), (3, 10), (3, 9), (3, 8), (3, 7), (3, 6), (3, 5), (3, 4), (3, 3)]
```

Fig 7. Route taken by robot from (3,3)

The enumerations are added in Appendix A.2 for better understanding of the directions the robot turned.

## Limitations and Future Work

Within the timeframe, a small mazebot has been developed and demonstrated however there are endless possibilities in integrating autonomous technology to systems and robots. The limitations do include less computational power, leading to lower timestep and less accurate models. The maze world can incorporate dynamic objects to further test the bot's capabilities. It can make use of better sensors like Lidar and sensory accuracy to improve the results obtained. For future work, with the establishment of the general maze-bot, a suggestion is that a particular problem can be targeted, and a solution can be brought about.

## Conclusion

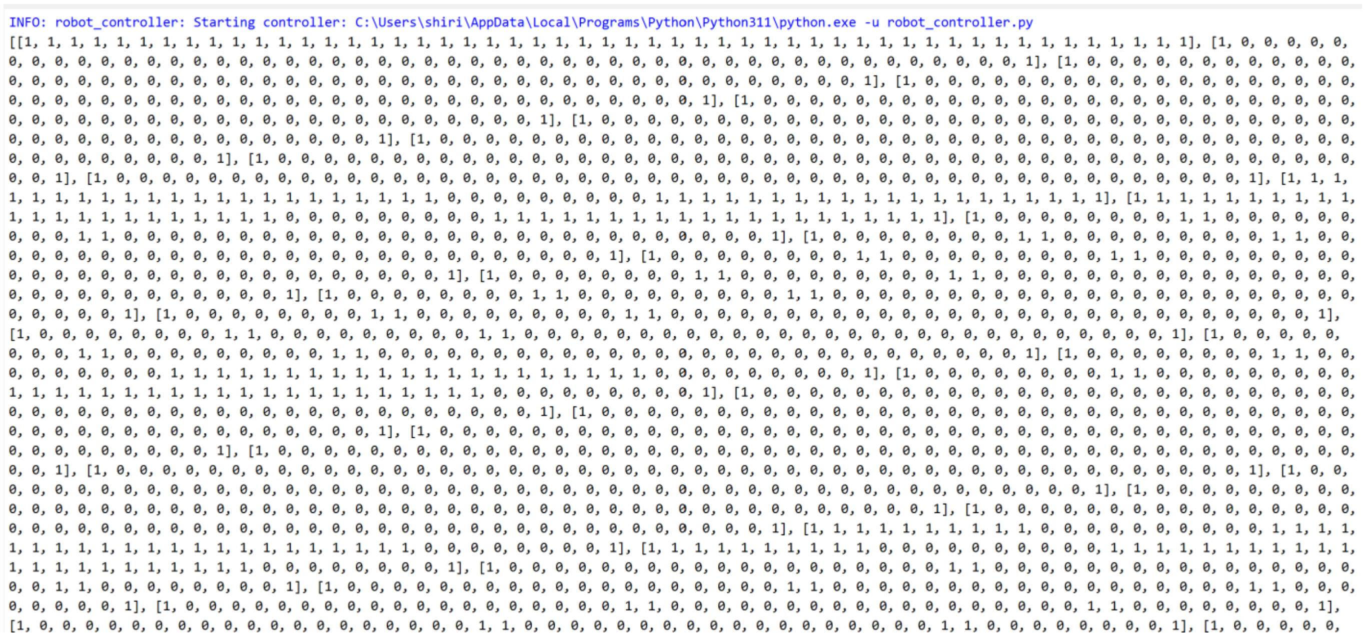
In conclusion, the robot has achieved basic maze navigation using conventional A* path planning. The total cost of moving from the start node to the goal is **158** upon testing the known map and occupational grid. Furthermore, the robot's performance was monitored continuously with the path it followed and it did take the shortest path available using A*. The hinge joints that were set and anchored with precision worked as intended for the rotational motors as well. All the joints and components worked as intended in the robot. The Python robot control exists successfully stating that there are no errors and the console which states the enumerations gives a better understanding of the robot's logic and process when moving forwards or taking a turn.

## References

1. Zhang, T., Li, Q., Zhang, Cs. et al. Current trends in the development of intelligent unmanned autonomous systems. Frontiers Inf Technol Electronic Eng 18, 68–85 (2017). https://doi.org/10.1631/FITEE.1601650
2. Dang, C.V. et al. (2022) "Improved analytic expansions in hybrid A-star path planning for non-holonomic robots," Applied Sciences, 12(12), p. 5999. Available at: https://doi.org/10.3390/app12125999.
3. Mujtaba, H. and Singh, G., 2017. Safe navigation of mobile robot using A* algorithm. International Journal of Applied Engineering Research, 12(16), pp.5532-5539.
4. Bornstein, J. and Y. Koren, Real-time obstacle avoidance for fast mobile robots. Systems, Man and Cybernetics, IEEE Transactions on,1989. 19(5): p.p 1179-1187.
5. German A. Vargas et.al "Simulation of e-puck path planning of Webot" International Journal of Applied Engineering Research 2016 vol-11 no.19 pp 9772-9775
6. Richardo acatillo et.al "Implementation and assembly of a robotic module for mecabot-3 reconfigurable system" International Journal of Applied Engineering Research 2016 vol-11 no.21 pp 10681- 10684
7. G.E. Jan, K. Y. Chang, and I. Parberry, "Optimal path planning for mobile robot navigation, " IEEE/ASME Transaction on Mechatronics vol.13, no.4, pp. 451-459, 2008

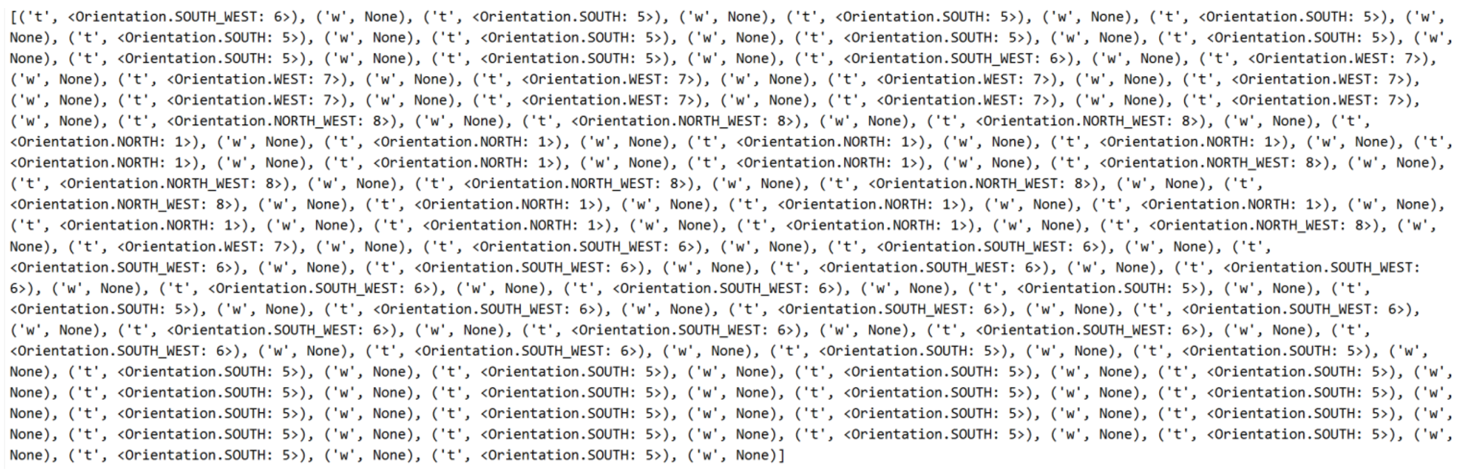
# Appendix

## A1. CSV File reading



```
INFO: robot_controller: Starting controller: C:\Users\shiri\AppData\Local\Programs\Python\Python311\python.exe -u robot_controller.py
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0,
```

## A2. Enumerators in Python Script



```
[('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.WEST: 7>),
('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>),
('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.WEST: 7>),
('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t',
<Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t',
<Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None),
('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w', None), ('t',
<Orientation.NORTH_WEST: 8>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None),
('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH: 1>), ('w', None), ('t', <Orientation.NORTH_WEST: 8>), ('w',
None), ('t', <Orientation.WEST: 7>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t',
<Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST:
6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t',
<Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>),
('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t',
<Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH_WEST: 6>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w',
None), ('t', <Orientation.SOUTH: 5>), ('w', None), ('t', <Orientation.SOUTH: 5>), ('w', None)]
```