Café Robot

Delivering Coffee to Teaching Staff

ID	Student Name	E-mail
196280	Ashraf Adel	ashraf196280@bue.edu.eg

Table of Contents

Introduction 3

Code Overview 3

- main.py 4
- globals.py 4
- Gui.py 5
- Spot.py 6
- Presets.py 7
- Algorithms.py 8

Program Demonstration 10

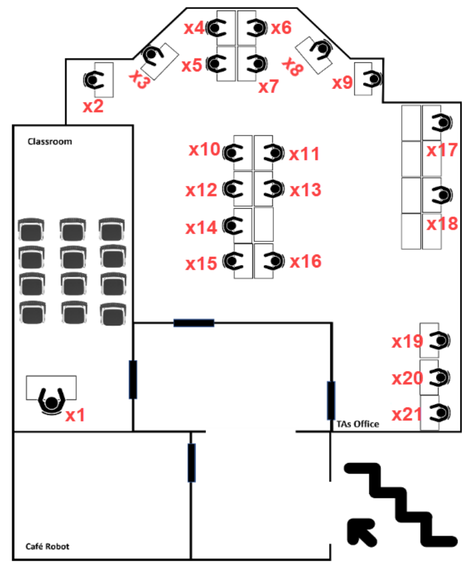
- Success Scenarios 10
  - Staff floor scenario 10
  - Random walls 11
  - Crooked smiley face 12
- Failure Scenarios 13
  - No solution scenario 13
  - Invalid member scenario 13
- Program Scope 14

References 15

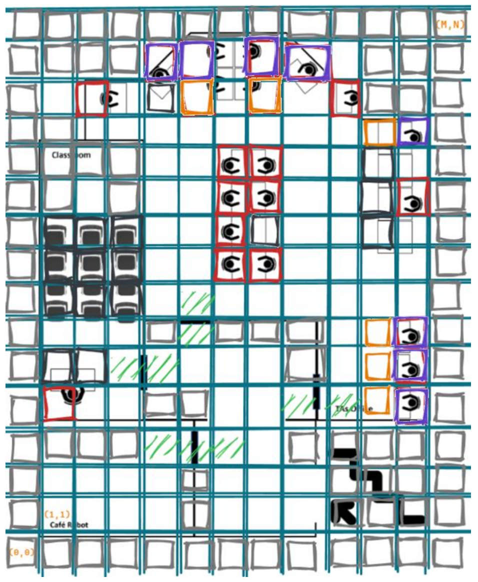
Introduction

As previously mentioned in the design document, we want to design an environment which our robot R can understand and from which it can execute its orders, where and “order” consisted of bringing coffee to a certain staff member (based on FCFS policy) or to return to its starting position.

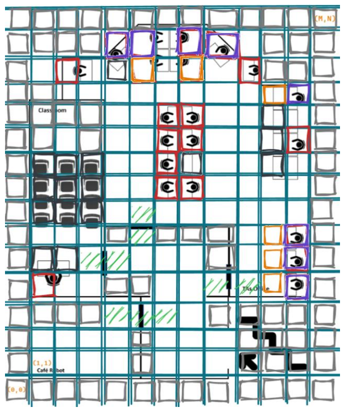
As also mentioned, it was decided that the suitable representation for this problem was the grid representation:



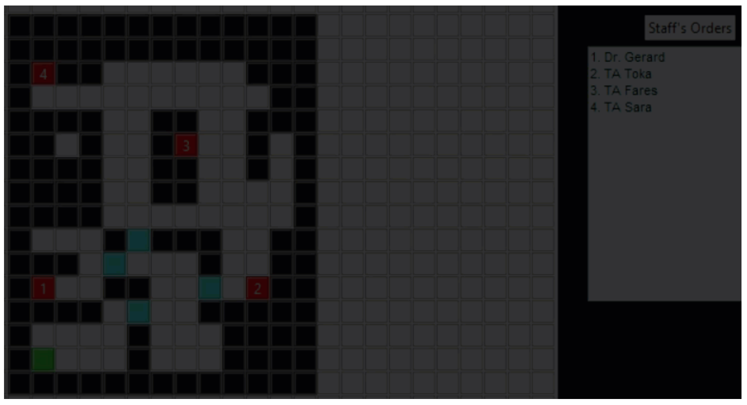
→



Furthermore, we chose the A* algorithm for R to use when executing its order (noting that the code also supports BFS for visualization purposes :)).
Now let’s see the code that made the following mapping possible:



→

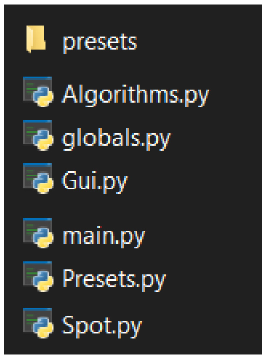


Staff's Orders

1. Dr. Gerard
2. TA Toka
3. TA Fares
4. TA Sara

Code Overview

The project has the code distributed logically as follows:


presets
Algorithms.py
globals.py
Gui.py
main.py
Presets.py
Spot.py

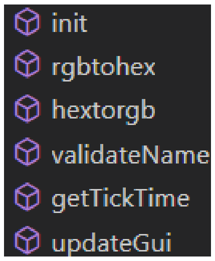
Let's discuss each of them!

main.py

- If you want to start the program, then run this file
- It initializes the tKinter GUI object [1] using globals.init() then sets up the initial UI of the tKinter object using GUI.ui() and runs the application from the initialized tKinter object using root.mainloop()

globals.py

It has the following functions:

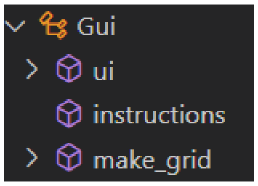

init
rgbtohex
hextorgb
validateName
getTickTime
updateGui

Such that:

- init(): initializes the global variables used in multiple .py files
- rgbtohex() and hextorgb(): helper functions made to facilitate the colouring process of the Spot instances' buttons (defined in Spot.py)
- validateName(): used to make sure the names entered in the "Staff's Orders" section in the UI's right frame are related to staff names. However, if the names weren't written when the execution of the orders start, then this function won't run, as the names will be initialized randomly (to staff names) instead.
- getTickTime(): returns the speed of running the A* or BFS algorithm.
- updateGui(): makes the Gui render new updates.

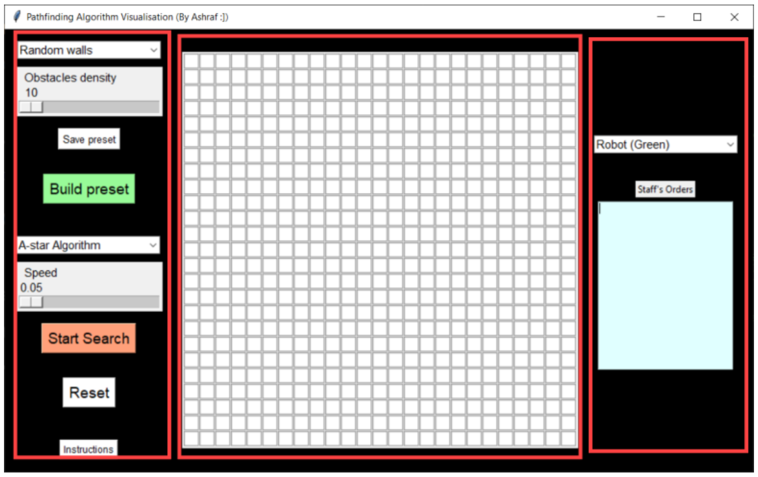
Gui.py

It has the following methods (of class Gui):

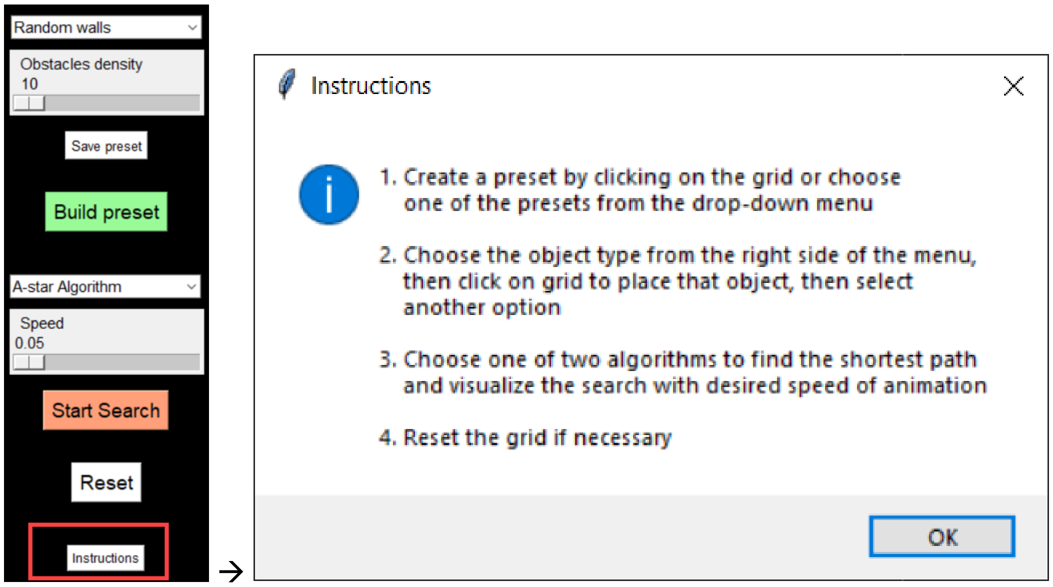


Such that:

- ui(): it forms the initial interface of the tKinter object [2]–[5], mainly consisting of left_frame, grid_canvas, and right_frame for the left, middle, and right columns of the program respectively:



- instructions(): displays program instructions upon clicking “instructions” button at the bottom left corner of the program:

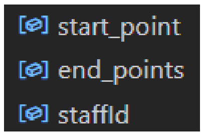


1. Create a preset by clicking on the grid or choose one of the presets from the drop-down menu
2. Choose the object type from the right side of the menu, then click on grid to place that object, then select another option
3. Choose one of two algorithms to find the shortest path and visualize the search with desired speed of animation
4. Reset the grid if necessary

- make_grid(): creates a ROWxROW number of Spot instances (from Spot.py) and returns a list of list of dimensions ROWxROW in order to be added to grid_canvas (where ROW is a variable in globals.py set to 25).

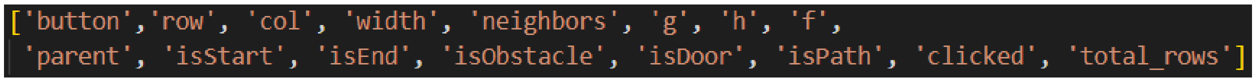
Spot.py

It has the following class attributes (accessible by just accessing the Spot class, instead of a specific instance):



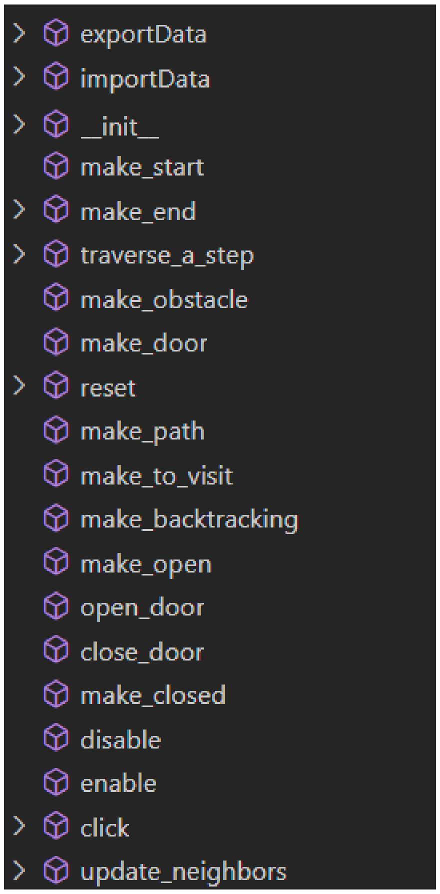
start_point
end_points
staffId

It also has the following instance attributes:



```
['button','row', 'col', 'width', 'neighbors', 'g', 'h', 'f',
 'parent', 'isStart', 'isEnd', 'isObstacle', 'isDoor', 'isPath', 'clicked', 'total_rows']
```

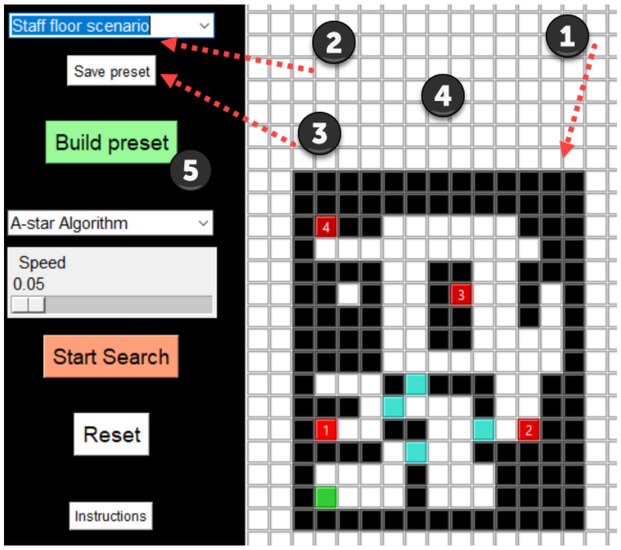
And the following instance methods:



exportData
importData
__init__
make_start
make_end
traverse_a_step
make_obstacle
make_door
reset
make_path
make_to_visit
make_backtracking
make_open
open_door
close_door
make_closed
disable
enable
click
update_neighbors

Such that:

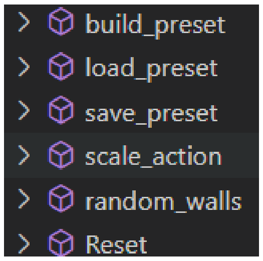
- export and import data functions are related to storing user-made environments in pickle files stored in “presets” folder and later retrieving them using the GUI:



  - (1) design the environment
  - (2) choose name for the custom-made environment
  - (3) click “Save Preset”
  - (4) close the program for it to load the preset into the combo-box
  - (5) open the program, choose newly-made preset, then click “Build Preset”
- The functions that start with “make_” are all to adjust the colors of the spots (i.e., cells on the grid) to reflect the current object type or the state of the robot/path.
- reset(): resets the spot (i.e., cell) to its initial values (e.g., white background, etc)
- enable and disable functions are to adjust state of the spot when an algorithm is running. However, making them disabled will make them have a darker shade of colour which is not good for visualising the algorithm. Therefore, they are not used, but are kept in case of future updates where they are needed.
- click(): specifies what happens when the user clicks on a spot on the grid canvas (e.g., make object type as “start” if the “Robot” option is selected from the right frame of the GUI, etc)
- update_neighbours(): adds spots that are near the current spot to the “neighbors” attribute (“near” hear means the orthogonal directions, not octilinear directions).

Presets.py

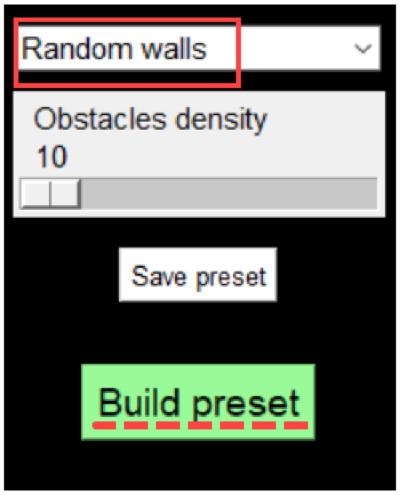
It has the following methods (in the class Presets):



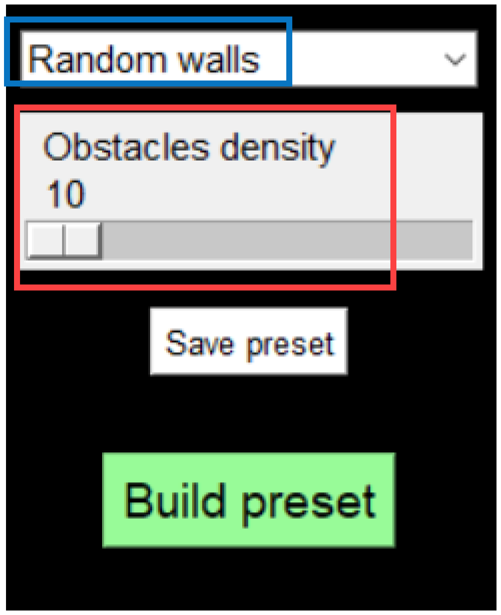
build_preset
load_preset
save_preset
scale_action
random_walls
Reset

Such that:

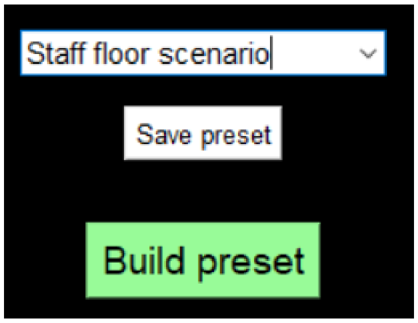
- build_preset(): will run either random_walls() or load_preset() based on chosen preset here:



- save_preset(): saves the current configuration of the spots on the grid_canvas along with the names in "Staff Orders" textbox into a pickle file located in "presets" folder.
- load_preset(): loads the spots (i.e., cells) in the stored pickle file to the grid_canvas's spots.
- random_walls(): creates obstacles in random locations on the grid.
- scale_action(): decides if the scaler (for obstacles density) is present on the GUI or not:



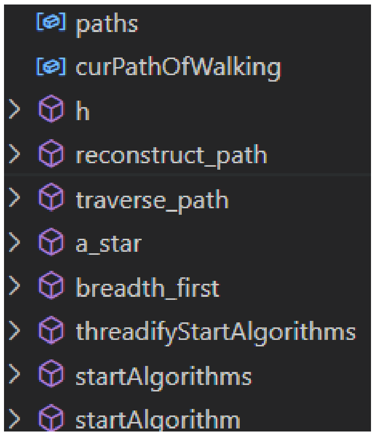
or



- Reset(): resets the spots in the grid_canvas to their original states.

Algorithms.py

It contains the following class attributes and methods (in the Algorithm class):



```
paths
curPathOfWalking
> h
> reconstruct_path
> traverse_path
> a_star
> breadth_first
> threadifyStartAlgorithms
> startAlgorithms
> startAlgorithm
```

Such that:

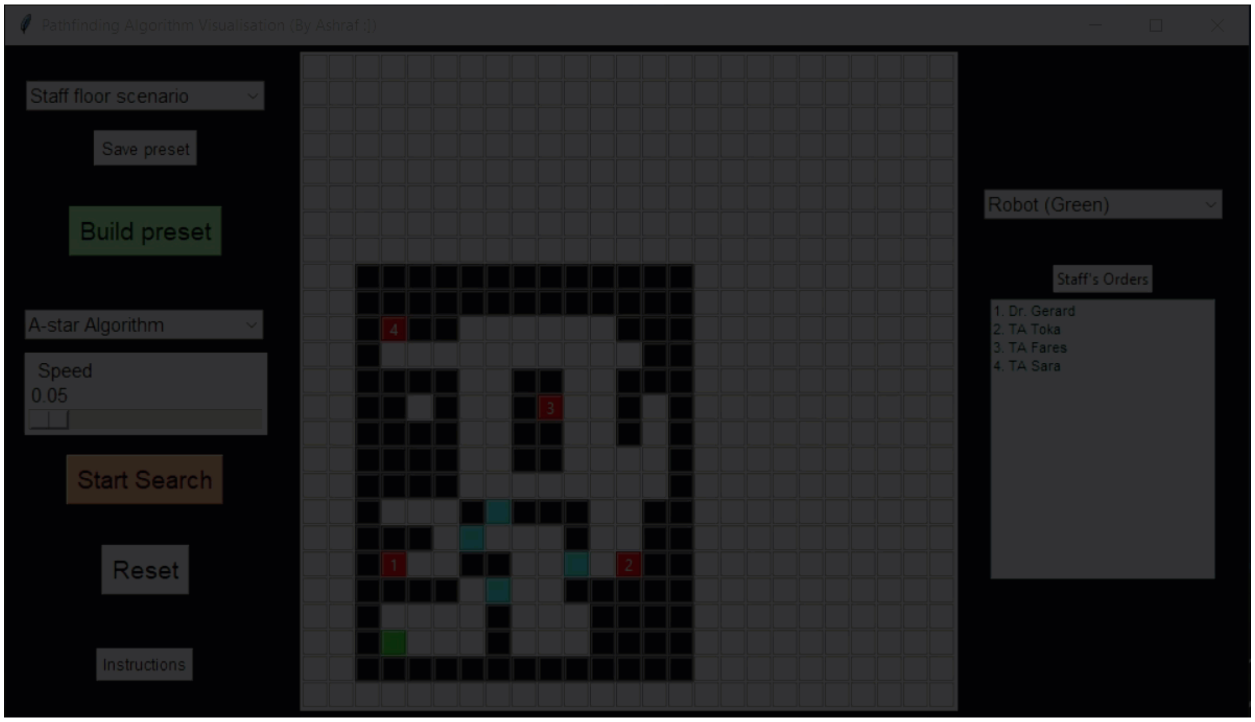
- paths: a list of lists specifying all the paths that the robot has traversed (in all its orders), including the path to get back to its starting position.
- curPathOfWalking: an integer (index) indicating the list in “paths” in which the robot should now traverse.
- h(): the heuristic function using Manhattan distance.
- reconstruct_path(): draws the yellow line from current ending point to the robot’s current location
- traverse_path(): makes the robot traverse through reconstruct_path().
- a_star(): search algorithm from current to goal point using A* algorithm.
- breadth_first(): search algorithm from current to goal point using BFS algorithm.
- thredifyStartAlgorithms(): wrapsstartAlgorithms() in a thread in order for the GUI not to freeze while running the algorithm.
- startAlgorithms(): runs startAlgorithm() for each staff member that the robot should deliver coffee to, in addition to the final traversal to the robot’s initial starting position.
- startAlgorithm(): starts A* or BFS algorithm (based on choice made in GUI) on the current start and end positions (current order that the robot should execute).

Program Demonstration

The following are demonstrations for the program using different presets.

Success Scenarios

Staff floor scenario



Staff floor scenario

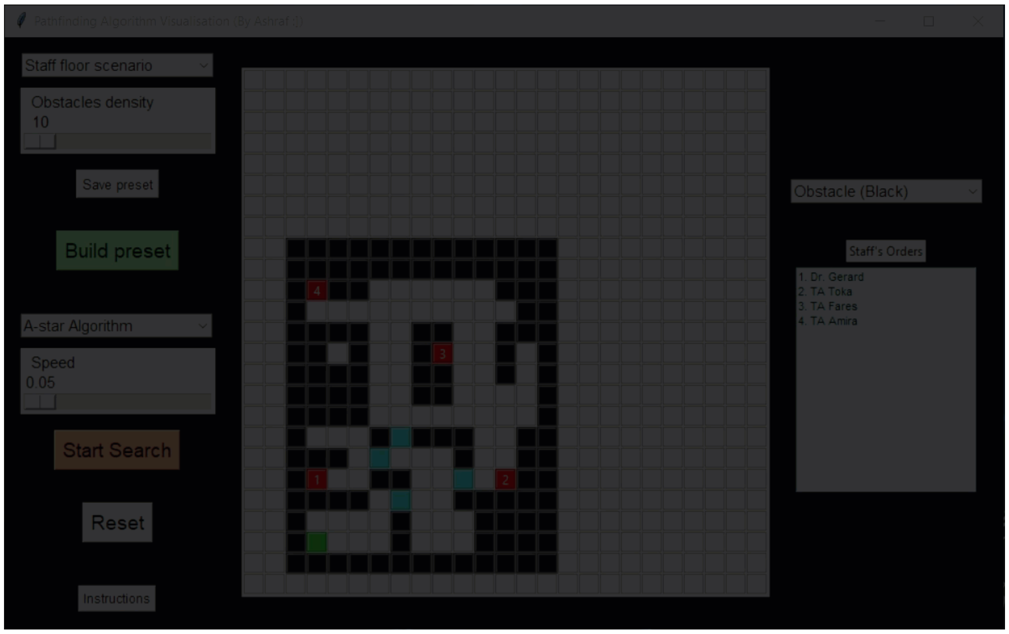
A-star Algorithm

Speed
0.05

Robot (Green)

1. Dr. Gerard
2. TA Toka
3. TA Fares
4. TA Sara

Slower version:



Staff floor scenario

Obstacles density
10

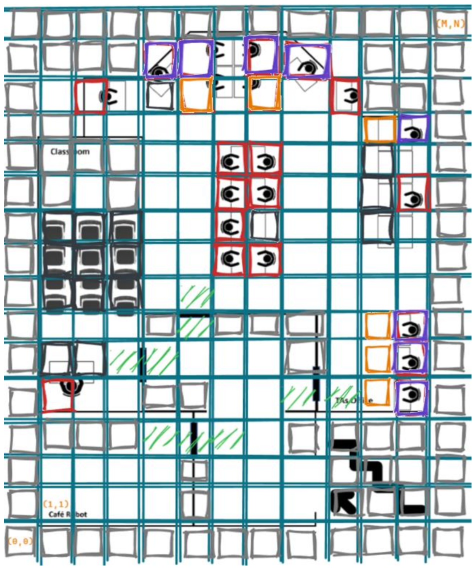
A-star Algorithm

Speed
0.05

Obstacle (Black)

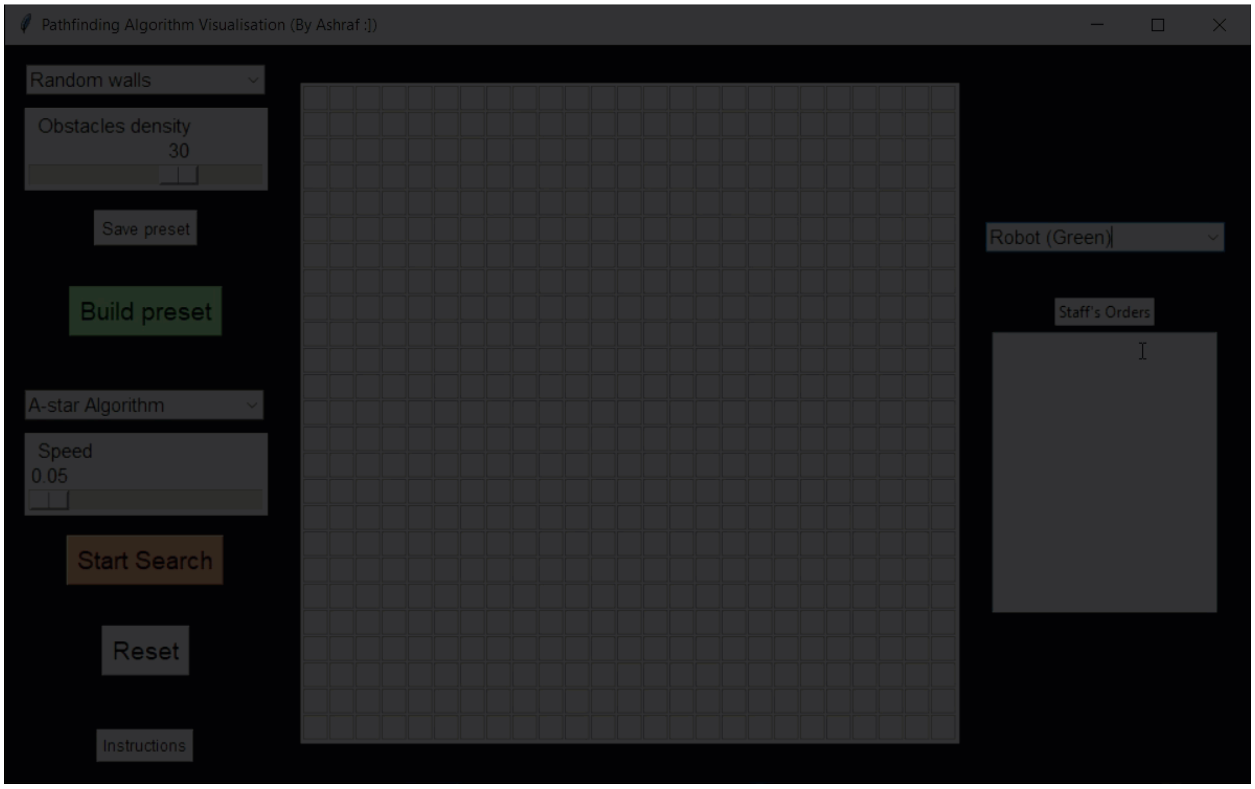
1. Dr. Gerard
2. TA Toka
3. TA Fares
4. TA Amira

As shown at the start of the report, this is a direct mapping to the initial Building H, floor 2 map:

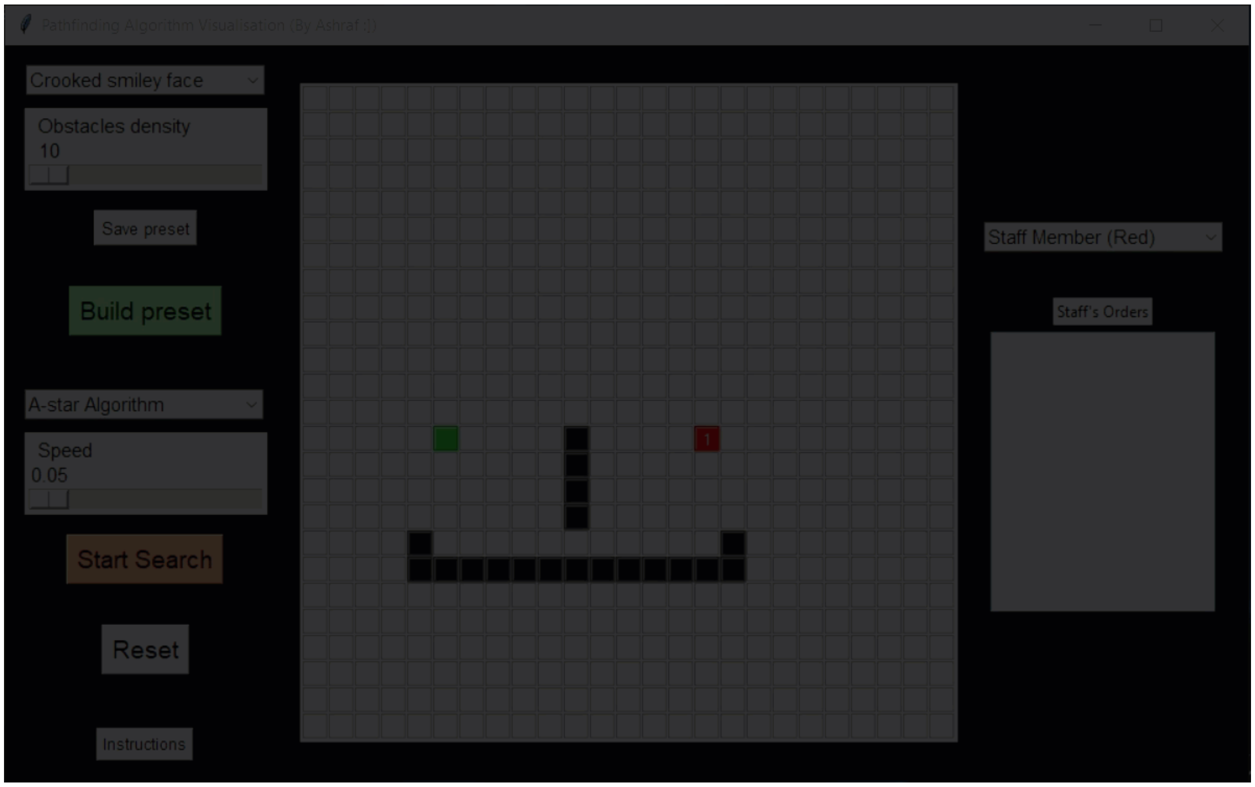


Random walls

(Random names are assigned)

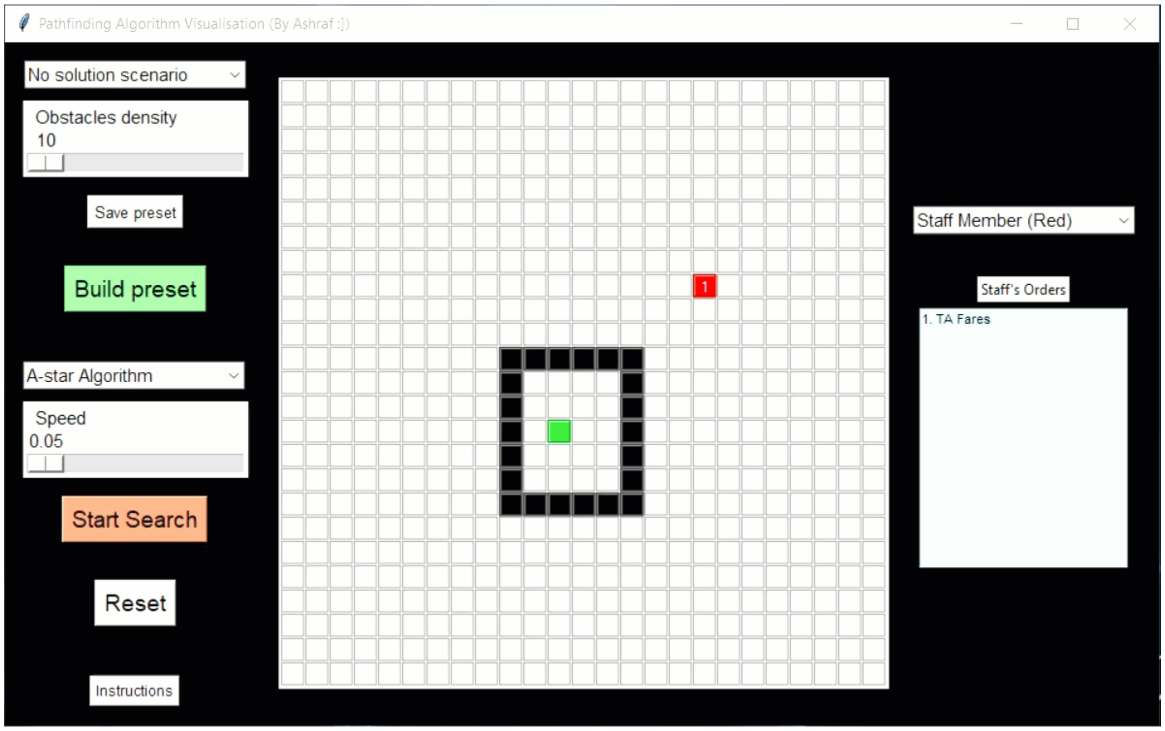


Crooked smiley face



Failure Scenarios

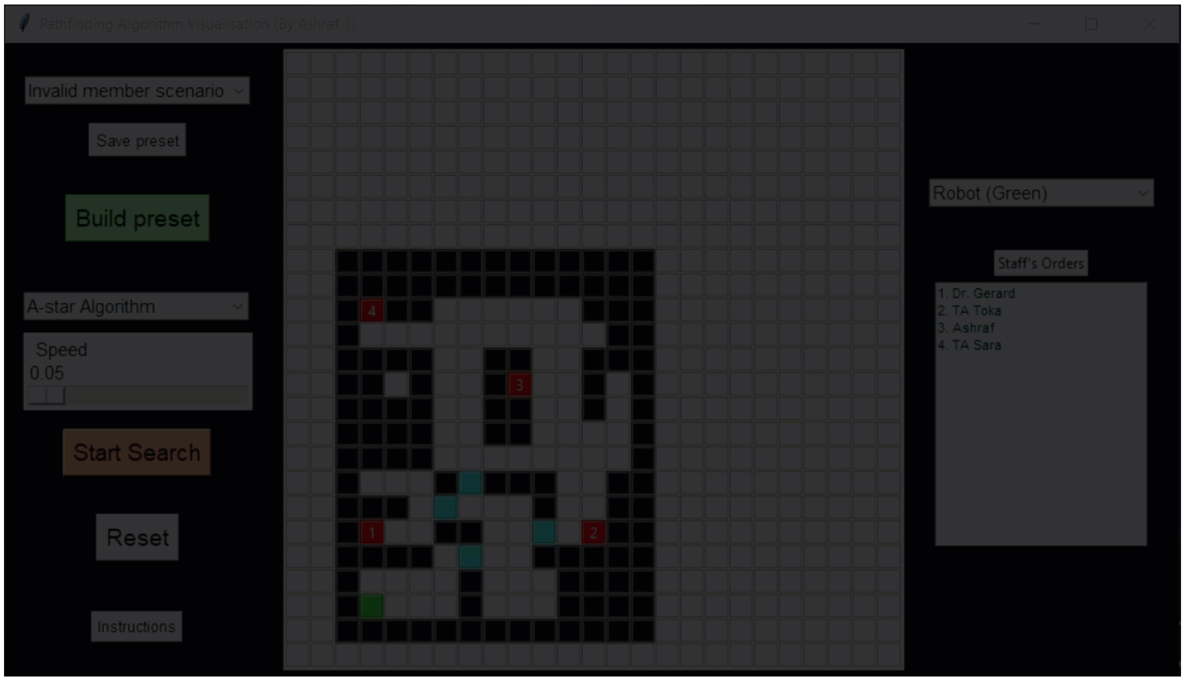
No solution scenario



1. TA Fares

Invalid member scenario

Notice how there's a non-staff member mentioned at (3.)



1. Dr. Gerard
2. TA Toka
3. Ashraf
4. TA Sara

Program Scope

- As can be seen from the demonstrations above, the program is able to generalize to any custom-made environment, given that it is logically suitable to represent this environment in a 25x25 grid.
- It also generalizes for any number of staff members. In real-life though, there maybe other constraints that prevent the robot from executing a non-limited number of orders (for example, limited memory size, etc). Therefore, it should be noted that the program's scope does not cover a restriction rule for the number of orders executed (i.e., coffees delivered).
- Regarding the algorithms, the scope of this program encompasses the A* and BFS algorithms only.
- Finally, regarding the presets, the program can save any new presets and later retrieve them (in a separate run of the program) via the combo-box in the upper left corner of the program.

References

[1] “Graphical User Interfaces with Tk — Python 3.11.0 documentation.” https://docs.python.org/3/library/tk.html (accessed Dec. 05, 2022).

[2] *A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial*, (Jul. 16, 2020). Accessed: Dec. 05, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=JtiK0DOeI4A

[3] *Python Pathfinding Vizualisation*, (Feb. 19, 2022). Accessed: Dec. 05, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=QNpUN8gBeLY

[4] *Breadth First Search Algorithm Visualization - Python Pygame Path Finding Visualization*, (Sep. 12, 2020). Accessed: Dec. 05, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=LF1h-8bEjP0

[5] “Breadth First Search Algorithm Visualization - Python Pygame Path Finding Visualization - YouTube.” https://www.youtube.com/watch?v=LF1h-8bEjP0 (accessed Dec. 05, 2022).