GROUP 2 MDP - Algorithm

Algorithm component places an important role in this Multi-disciplinary Design Project. The algorithm component acts like the brain that determines the route taken by the robot.

The overall software architectural style for the algorithm component uses event-driven architecture. The architectural diagram for the algorithm module is shown in Figure 1.

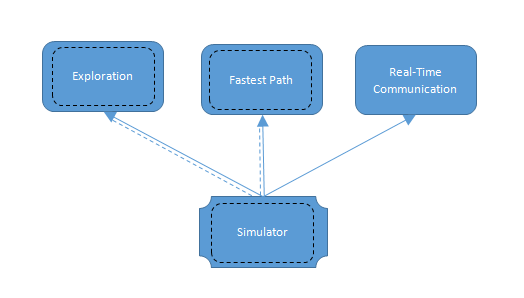


Figure : Architectural diagram-Event driven

The simulator is the user interface that triggers the other components such as exploration, fastest path and real-time communication. The simulator is implemented using Java Swing API which is based on event driven architecture, where the interface will calls the function when there is an action made by the user. The other components are implemented using Java.

The sections below will further the elaborate the implementation of the algorithm module including the features, logic, problems faced and solutions to overcome.

Simulator

Maze Map

Load Map

Map Descriptor

Solve Path

Terminate

Exploration

Real Timer

Right Side panel

Legend

Exploration & fastest path statistics

Set exploration value

**Simulator**

The implemented simulator interface is presented in Figure 2.

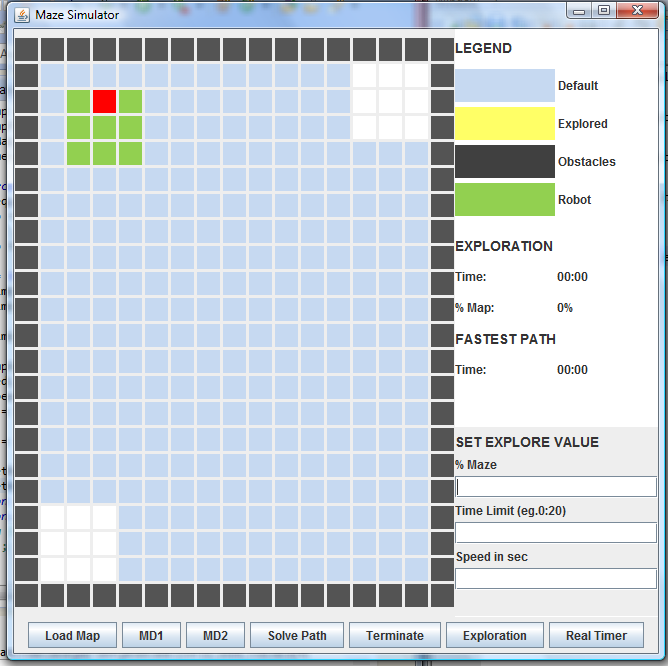


Figure : Simulator Layout

The simulator consists of the maze map represented by the 17 x 22 grids - including the walls, buttons to trigger the other components, and a right panel that display the robot statistic and user limitation for exploration. The robot in this simulator is represented by the green grid with a red grid representing the robot orientation. The actual robot is 20 cm by 20 cm. But in this simulator, the robot is represented 3 by 3 on the maze map, considering each grid represent 10 cm. We decided on 3 by 3 to allow 5 cm spaces around the robot to minimize the chances of the robot hitting obstacles. The start and goal are represented by the white grids. The walls and obstacles are represented by black grids. For simulation, obstacles can be set by two methods: Clicking the blue grids or Load Map button at the bottom left. The first method to plot obstacles is simply clicking the blue grid on the map shown in Figure. Click the obstacles to unselect the obstacles.

--CLICK GIF---

The latter method is the load map button shall allow user to load a txt file with the following format and the loaded result shown in Figure.

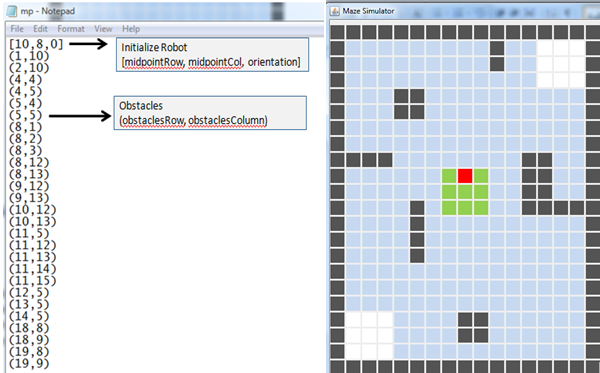


Figure : Load Map

The load map is implemented with the aid of Java Swing library, JFileChooser for the selection of file interface. The load map interface is presented in Figure.

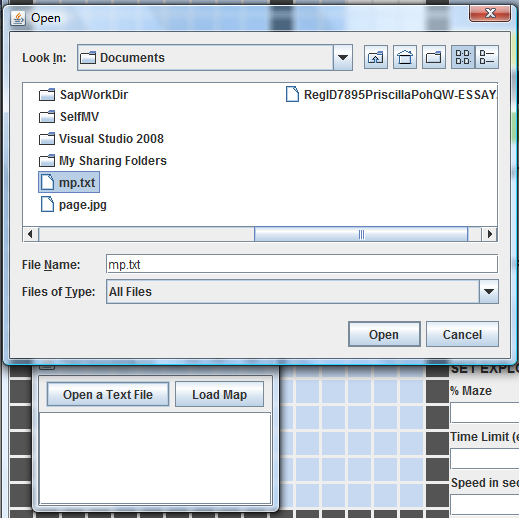
****

Figure : Load Map interface

The MD1 & MD2 buttons are the map descriptor 1 and 2 respectively. The map descriptor is a map of the environment where cells displayed a value that represents certain meaning for different map descriptor. The format for MD1 is: 0 represent unexplored grid, 1 represent grid have been visited. The format for MD2 is: 1 represent visited obstacles, 0 represent visited safe grid, unvisited grid should not display any number. The numbers on the grid are translated into hexadecimal that is displayed on the console. The MD1 & MD2 are designed to be dynamic if the robot is exploring the maze. This is shown in the video below.

The solve path button triggers the simulator’s fastest path. The algorithm is further explained in the Fastest Path section below.

Terminate button is to terminate all the actions happening in the maze map and reset the maze map and all the statistics on the right panel.

The exploration button triggers the simulator’s exploration. The algorithm is further explained in the Exploration section below.

The real-time button is to allow communication with the rpi to send and receives message from arduino and android. The implementation is further explained in the last section.

The right side of the panel is divided into three sections. The top section is the legend of the maze map. The middle section is the statistics of the simulator robot such as the time taken for the simulated robot for exploration and fastest path and the percentage of the map covered. The values are dynamically updated as the simulator’s robot moves. The bottom section is to control the variables of the exploration. This includes the percentage of maze covered, the maximum time for the robot to explore and the speed of the simulator’s robot. The user shall set the value before clicking on the exploration button, which will takes all the input as limitation for the robot exploration.

**Exploration**

Exploration module is to allow the robot to explore the maze as much as possible within 6 minutes. The robot shall start in the middle of the designated position and traverse through goal and start. During the traversal, the robot shall be able to avoid obstacles and plot the simulator’s map. In the simulator exploration, Java Swing timer is used to represent the delay of the robot.

## Previous Strategy

Initially, the exploration algorithm is to allow the robot to visit the whole maze and tries to avoid obstacles and visited grid. For this algorithm, we only uses short sensor at the front and side, the robot is only able to sense one grid away. This is shown in Figure, the yellow grid are the explored range.

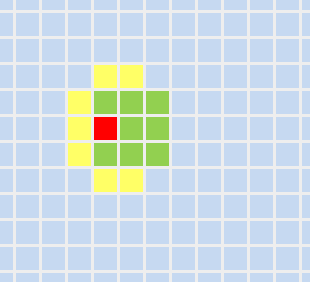


Figure : Sensor range for exploration

Figure shows the demonstration of the robot movement without obstacles.

The robot has to traverse every single grid on the maze except obstacles. The robot shall traverse according to psedocode as below:

Check if front no wall & not visited{

move forward

push position into backtrack stack

}

else Check if left no wall & not visited

spin left

else Check if right no wall & not visited

spin right

else

backtrack using stack

The visited check will ensure that the robot will not keep turning in circle. This method works well for all maze map since backtrack is also included, thus the robot will not be stuck in a loop. However, this method takes up a lot of time if the maze is too complicated.

## Final Strategy

The final strategy for our team exploration algorithm is to keep the right side of the robot to the wall, only traversing around the wall. However, the exploration rule stated that the robot has to start in the middle of the maze, therefore allowing the robot to reach the wall is a challenge. To increase the effectiveness of the wall algorithm, our group has increase the sensor range as shown in the figure below.

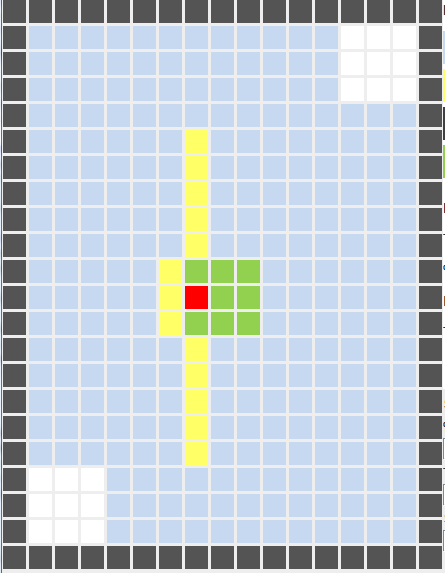


Figure : Sensor range for exploration algorithm

We did a lot of testing with different version of code, different version suits different kind of map. Our group finally settled down with pledge algorithm to traverse from the starting point to any of the four wall.

## Conclusion

Below shows the same map (taken from the first leaderboard), the difference in timing. As a demonstration, we assume the robot take 2 seconds per move, and that is the difference in timing for the robot to traverse the whole maze.