Maze Solver using BFS

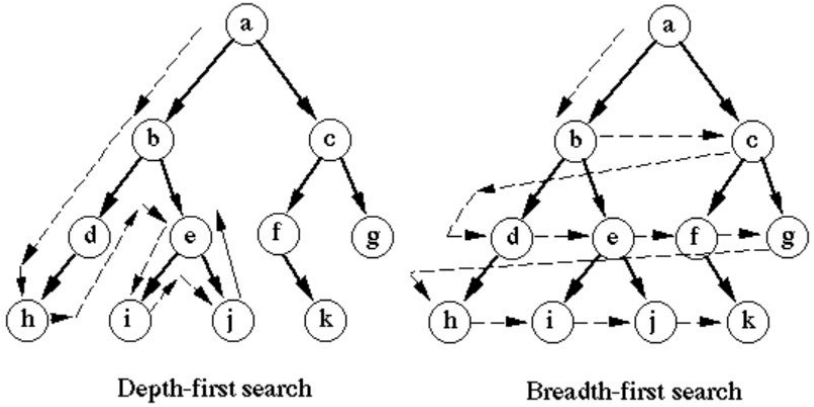
ENPM809Y - Introductory Robot Programming

Group 7

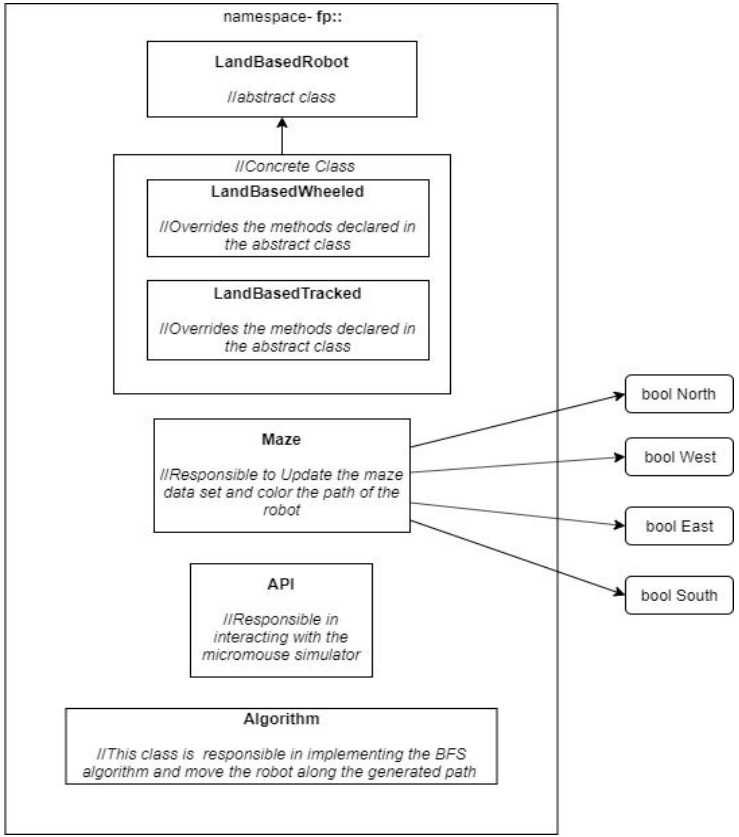
Aditya Khopkar , Raghav Nandwani , Nalin Das , Nidhi Bhojak , Balasankula Sai Chaitanya , Kushagra Agrawal

What is BFS?

- Breadth First Search is an algorithm for traversing or searching a tree or graph data structures.
- It starts at the base node, and explores all of the neighbour nodes at the present node level prior to moving on to the next node level.
- A child node is regarded as the next node which is or can be traversed from the parent node. The parent and the child node is updated through each iterations till the goal is reached.
- Once the goal is reached, the algorithm returns a path from the base node to the goal node.
- Unlike DFS, The BFS explores all potential the nodes at the same level. Whereas, DFS explores the nodes at one depth before moving on the next node.



C++ Class Structure



Class Methods

1. **API** : The API class contains all the methods which are implemented to interact with the micromouse simulator, such as:

a. wallFront() : a boolean function which returns true if a wall is detected in the front. Similarly, wallRight(), wallLeft() follows.

b. moveForward() : This method simulates a forward motion in the simulator.

c. turnRight() : This method simulates the robot to turn right in the simulator at the present node. Similarly, turnLeft() follows.

d. setWall(): This method sets the wall at the given position and orientation.

2. **Land Based Robot**: This is a base class it has two derived classes i.e. Land based tracked and land based wheeled.

Class Methods

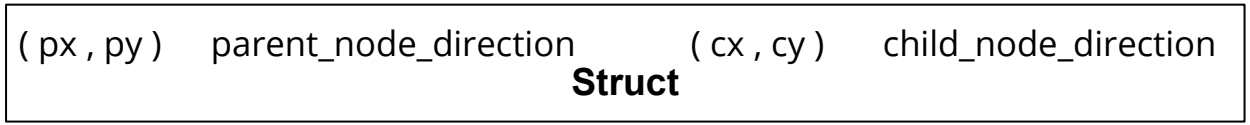
3. **Land based tracked and wheeled**: It is a derived class and it follows dynamic polymorphism because the methods are being virtually overridden. The methods such as MoveForward(), TurnLeft() and TurnRight() are defined in these classes.

4. **Maze**: The class contains information regarding the maze to be explored by the robot.

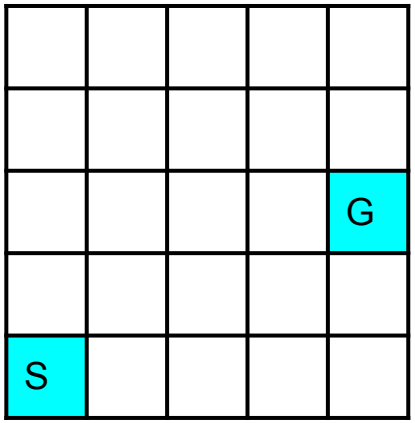
a. ReadMaze(): It updates maze matrices for current position and orientation.
b. colorPath():Sets color to the path to be followed by the robot by ‘g’

5. **Algorithm**: Breadth First Search Algorithm is used to navigate through the maze. The start node, goal node and current direction is known. The robot has to travel through the maze generating the walls simultaneously. This is achieved by methods such as CheckNodes(), ValidNodes(), GenerateSequence() and SolveMaze().

Data structure



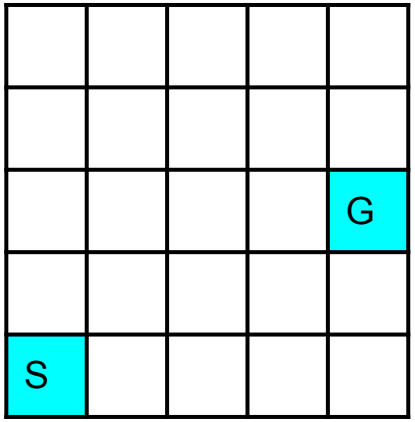
Data structure



(px , py)	parent_node_direction	(cx , cy)	child_node_direction
(0 , 0)	n	(0 , 1)	n
(0 , 0)	n	(1 , 0)	e
(0 , 1)	n	(0 , 2)	n
(0 , 1)	n	(1 , 1)	e
(1 , 0)	e	(2 , 0)	e
(0 , 2)	n	(0 , 3)	n
(0 , 2)	n	(1 , 2)	e
(1 , 1)	e	(2 , 1)	e
(2 , 0)	e	(3 , 0)	e
(0 , 3)	n	(0 , 4)	n
(0 , 3)	n	(1 , 3)	e
(1 , 2)	e	(2 , 2)	e
(2 , 1)	e	(3 , 1)	e
(3 , 0)	e	(4 , 0)	e
(0 , 4)	n	(1 , 4)	e
(1 , 3)	e	(2 , 3)	e
(2 , 2)	e	(3 , 2)	e
(3 , 1)	e	(4 , 1)	e
(1 , 4)	e	(2 , 4)	e
(2 , 3)	e	(3 , 3)	e
(3 , 2)	e	(4 , 2)	e

Vector<Node>

Back tracking

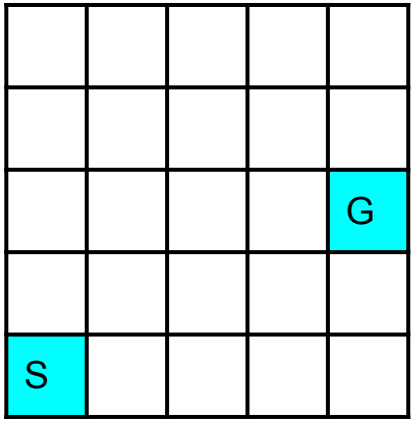


(px , py)	parent_node_direction	(cx , cy)	child_node_direction
(0 , 0)	n	(0 , 1)	n
(0 , 0)	n	(1 , 0)	e
(0 , 1)	n	(0 , 2)	n
(0 , 1)	n	(1 , 1)	e
(1 , 0)	e	(2 , 0)	e
(0 , 2)	n	(0 , 3)	n
(0 , 2)	n	(1 , 2)	e
(1 , 1)	e	(2 , 1)	e
(2 , 0)	e	(3 , 0)	e
(0 , 3)	n	(0 , 4)	n
(0 , 3)	n	(1 , 3)	e
(1 , 2)	e	(2 , 2)	e
(2 , 1)	e	(3 , 1)	e
(3 , 0)	e	(4 , 0)	e
(0 , 4)	n	(1 , 4)	e
(1 , 3)	e	(2 , 3)	e
(2 , 2)	e	(3 , 2)	e
(3 , 1)	e	(4 , 1)	e
(1 , 4)	e	(2 , 4)	e
(2 , 3)	e	(3 , 3)	e
(3 , 2)	e	(4 , 2)	e

Sequence Vector

(4 , 2)	e
(3 , 2)	e

Back tracking



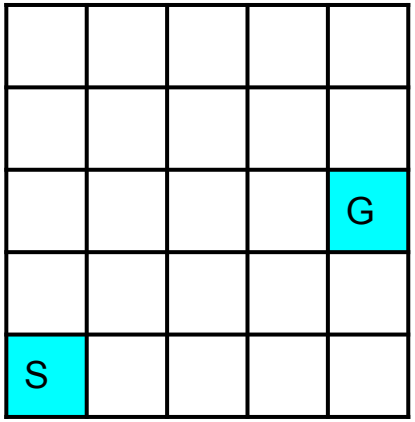
(px , py)	parent_node_direction
(0 , 0)	n
(0 , 0)	n
(0 , 1)	n
(0 , 1)	n
(1 , 0)	e
(0 , 2)	n
(0 , 2)	n
(1 , 1)	e
(2 , 0)	e
(0 , 3)	n
(0 , 3)	n
(1 , 2)	e
(2 , 1)	e
(3 , 0)	e
(0 , 4)	n
(1 , 3)	e
(2 , 2)	e
(3 , 1)	e
(1 , 4)	e
(2 , 3)	e
(3 , 2)	e

(cx , cy)	child_node_direction
(0 , 1)	n
(1 , 0)	e
(0 , 2)	n
(1 , 1)	e
(2 , 0)	e
(0 , 3)	n
(1 , 2)	e
(2 , 1)	e
(3 , 0)	e
(0 , 4)	n
(1 , 3)	e
(2 , 2)	e
(3 , 1)	e
(4 , 0)	e
(1 , 4)	e
(2 , 3)	e
(3 , 2)	e
(4 , 1)	e
(2 , 4)	e
(3 , 3)	e
(4 , 2)	e

Sequence Vector

(4 , 2)	e
(3 , 2)	e
(2 , 2)	e

Back tracking



(px , py)	parent_node_direction
(0 , 0)	n
(0 , 0)	n
(0 , 1)	n
(0 , 1)	n
(1 , 0)	e
(0 , 2)	n
(0 , 2)	n
(1 , 1)	e
(2 , 0)	e
(0 , 3)	n
(0 , 3)	n
(1 , 2)	e
(2 , 1)	e
(3 , 0)	e
(0 , 4)	n
(1 , 3)	e
(2 , 2)	e
(3 , 1)	e
(1 , 4)	e
(2 , 3)	e
(3 , 2)	e

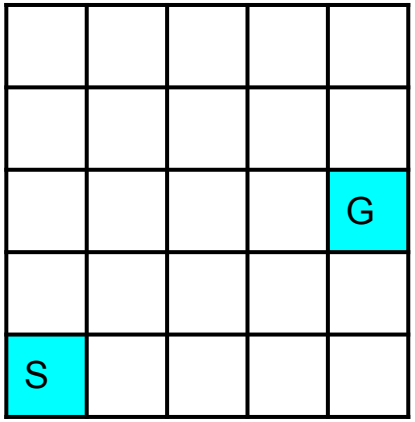
(cx , cy)	child_node_direction
(0 , 1)	n
(1 , 0)	e
(0 , 2)	n
(1 , 1)	e
(2 , 0)	e
(0 , 3)	n
(1 , 2)	e
(2 , 1)	e
(3 , 0)	e
(0 , 4)	n
(1 , 3)	e
(2 , 2)	e
(3 , 1)	e
(4 , 0)	e
(1 , 4)	e
(2 , 3)	e
(3 , 2)	e
(4 , 1)	e
(2 , 4)	e
(3 , 3)	e
(4 , 2)	e

Sequence Vector

(4 , 2)	e
(3 , 2)	e
(2 , 2)	e
(1 , 2)	e

& so on

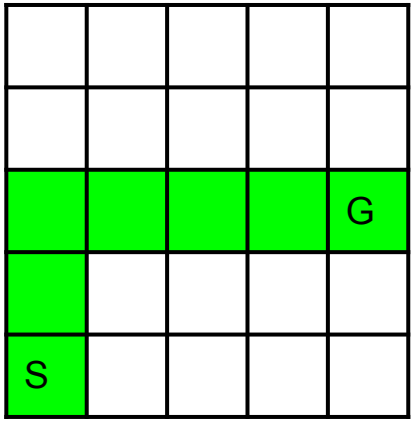
Back tracking



Sequence Vector	
(4,2)	e
(3,2)	e
(2,2)	e
(1,2)	e
(0,2)	n
(0,1)	n
(0,0)	n

(px , py)	parent_node_direction	(cx , cy)	child_node_direction
(0,0)	n	(0,1)	n
(0,0)	n	(1,0)	e
(0,1)	n	(0,2)	n
(0,1)	n	(1,1)	e
(1,0)	e	(2,0)	e
(0,2)	n	(0,3)	n
(0,2)	n	(1,2)	e
(1,1)	e	(2,1)	e
(2,0)	e	(3,0)	e
(0,3)	n	(0,4)	n
(0,3)	n	(1,3)	e
(1,2)	e	(2,2)	e
(2 ,1)	e	(3,1)	e
(3,0)	e	(4,0)	e
(0,4)	n	(1,4)	e
(1,3)	e	(2,3)	e
(2,2)	e	(3,2)	e
(3,1)	e	(4,1)	e
(1,4)	e	(2,4)	e
(2,3)	e	(3,3)	e
(3,2)	e	(4,2)	e

Back tracking



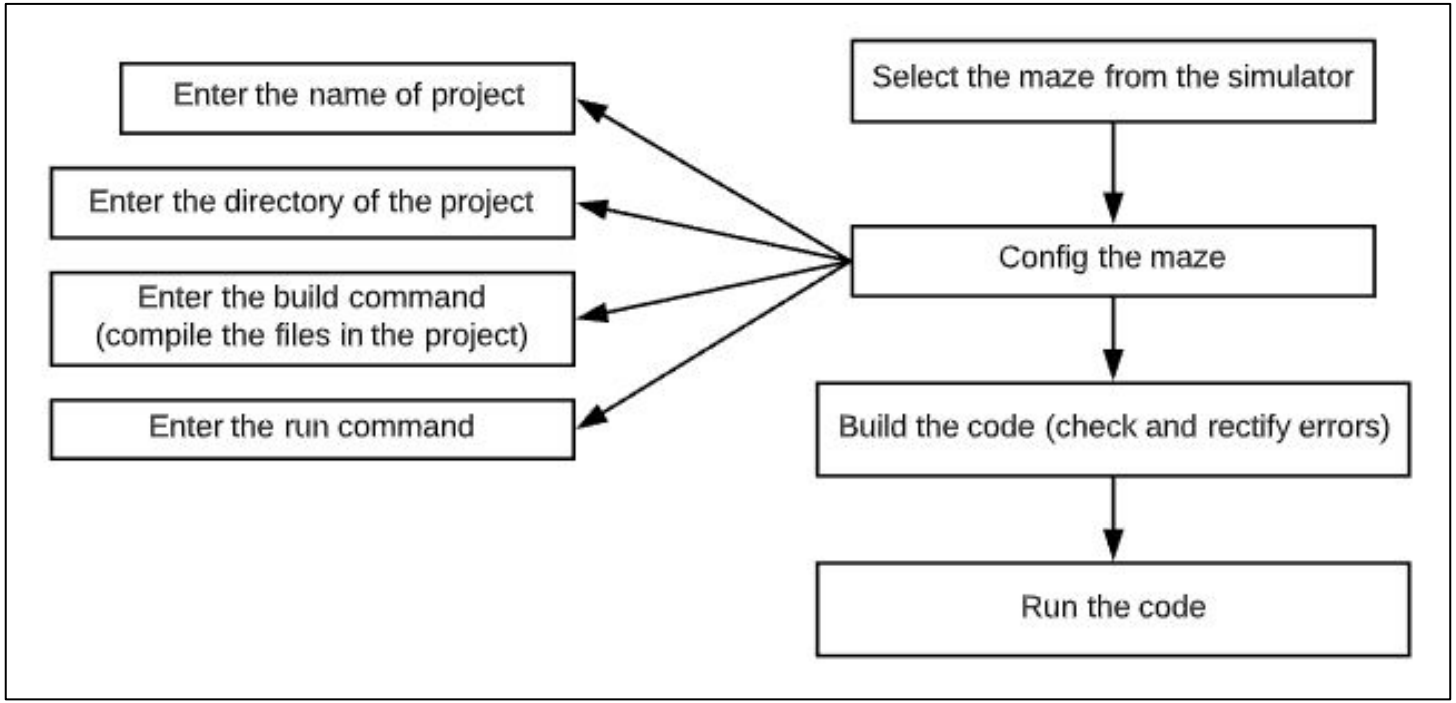
Sequence Vector		
	(0 , 0)	n
	(0 , 1)	n
	(0 , 2)	n
	(1 , 2)	e
	(2 , 2)	e
	(3 , 2)	e
	(4 , 2)	e

(px , py)	parent_node_direction	(cx , cy)	child_node_direction
(0 , 0)	n	(0 , 1)	n
(0 , 0)	n	(1 , 0)	e
(0 , 1)	n	(0 , 2)	n
(0 , 1)	n	(1 , 1)	e
(1 , 0)	e	(2 , 0)	e
(0 , 2)	n	(0 , 3)	n
(0 , 2)	n	(1 , 2)	e
(1 , 1)	e	(2 , 1)	e
(2 , 0)	e	(3 , 0)	e
(0 , 3)	n	(0 , 4)	n
(0 , 3)	n	(1 , 3)	e
(1 , 2)	e	(2 , 2)	e
(2 , 1)	e	(3 , 1)	e
(3 , 0)	e	(4 , 0)	e
(0 , 4)	n	(1 , 4)	e
(1 , 3)	e	(2 , 3)	e
(2 , 2)	e	(3 , 2)	e
(3 , 1)	e	(4 , 1)	e
(1 , 4)	e	(2 , 4)	e
(2 , 3)	e	(3 , 3)	e
(3 , 2)	e	(4 , 2)	e

Pseudo code

```
SolveMaze(Maze and Robot){
    while(There is a valid path and Goal not achieved){
        Generate Path using BFS{
            Clear all colors from the maze
            push the robot's current position in the NODES generated
            for(iterate through all the NODES generated){
                Generate all the valid nodes from a current node
                and push it in the NODES untill reaches goal
            }
            if goal found{
                Backtrack and push the nodes in sequence untill start node
            }
        }
        MoveRobotin the maze{
            for (iterate through the generated path){
                if(The path is valid i.e. no walls in the path) { Move robot }
                else if(There is a wall in the path) { break }
            }
        }
    }
}
```

Project Execution



Thank You !!