ENPM-809Y
Introduction to Robot Programming

Final Project - Group 10

Objective

- Given a starting location and orientation (fixed in this case), reach the goal (centers of the maze).
- Couple of constraints;
    - We do not know the locations of the obstacles (walls)
    - Mouse cannot move in arbitrary direction (only forward, but can turn-holonomic behaviour)
    - MMS simulator interface

What to learn from this Project?

- OOP based code development
- Algorithm development using Dynamic Programming
- Use Data structures such as queue, stack

Approach

- OOP based project design
- Interface with the simulator - API class
- Implement Dynamic Programming algorithms like Breadth-First-Search, Depth-First-Search to find the path.
    - **NOTE**: Need not be an optimal path!
- Compare and analyse the performance of two algorithms

Breadth-First Search (BFS)

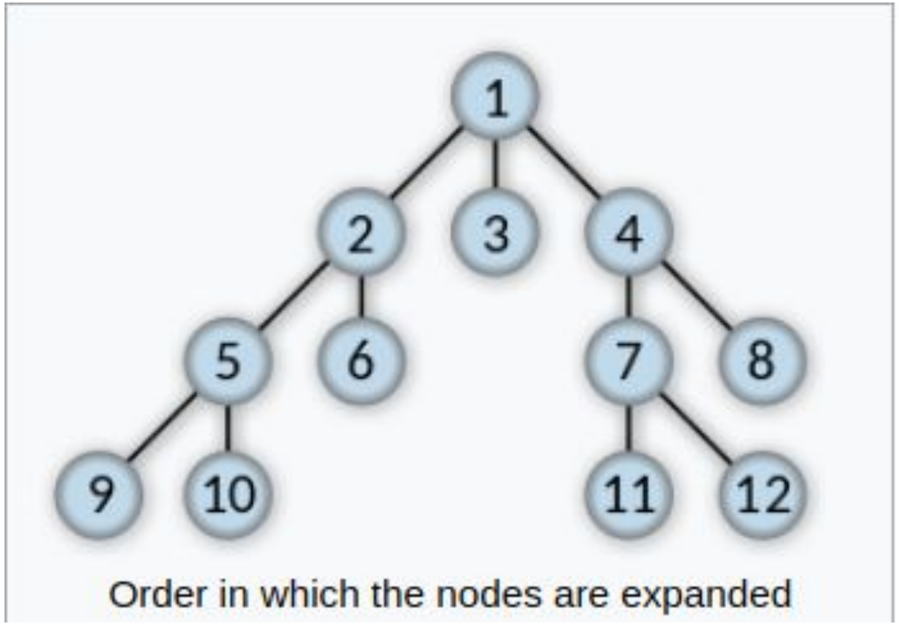


Figure: Breadth-First Search spanning tree

Credits: https://en.wikipedia.org/wiki/Breadth-first_search

Depth-First-Search (DFS)

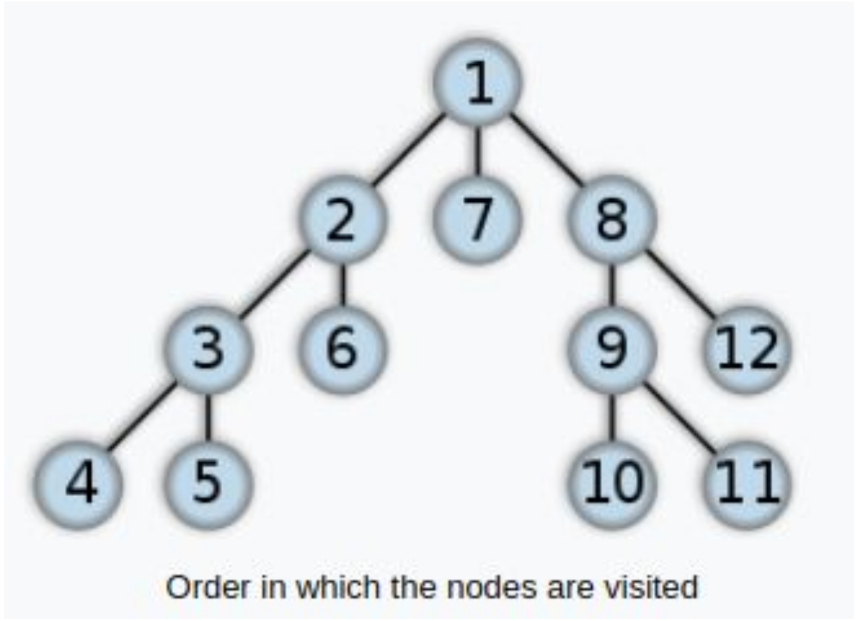
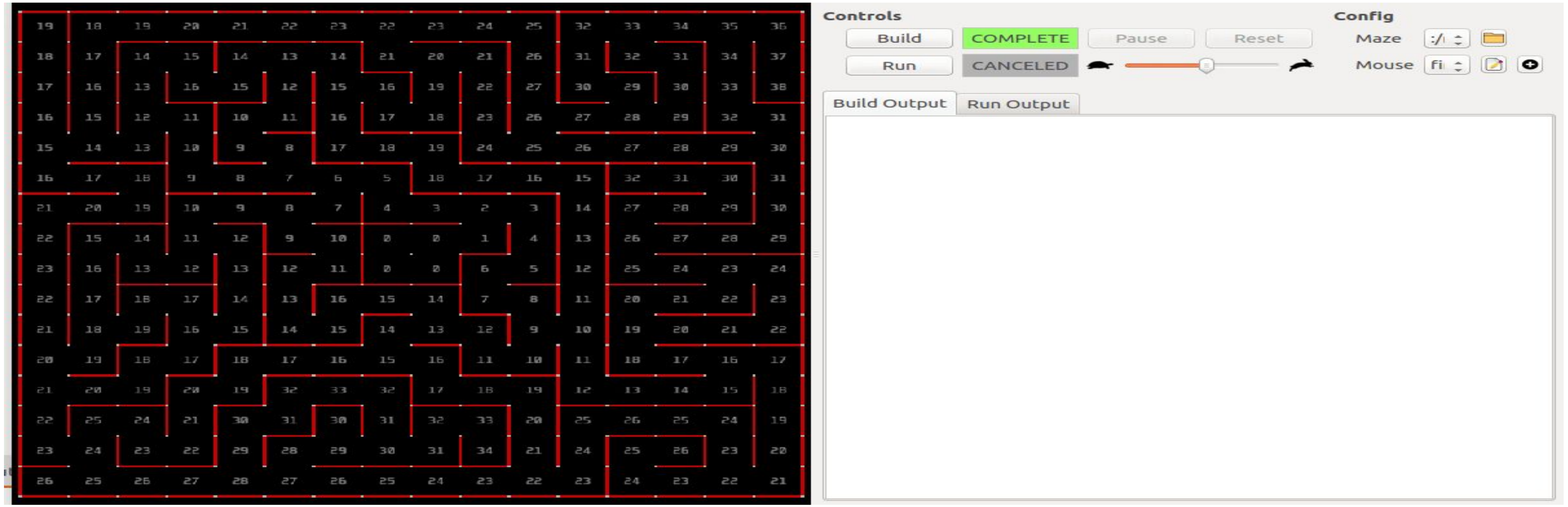


Figure: Depth-First Search spanning tree

Credits: https://en.wikipedia.org/wiki/Depth-first_search

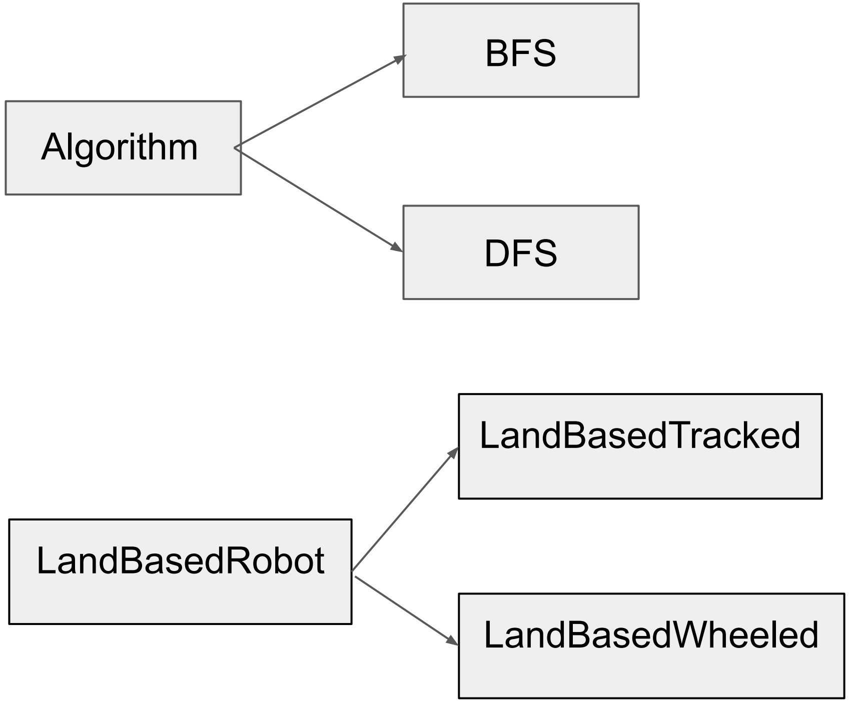
Micromouse simulator



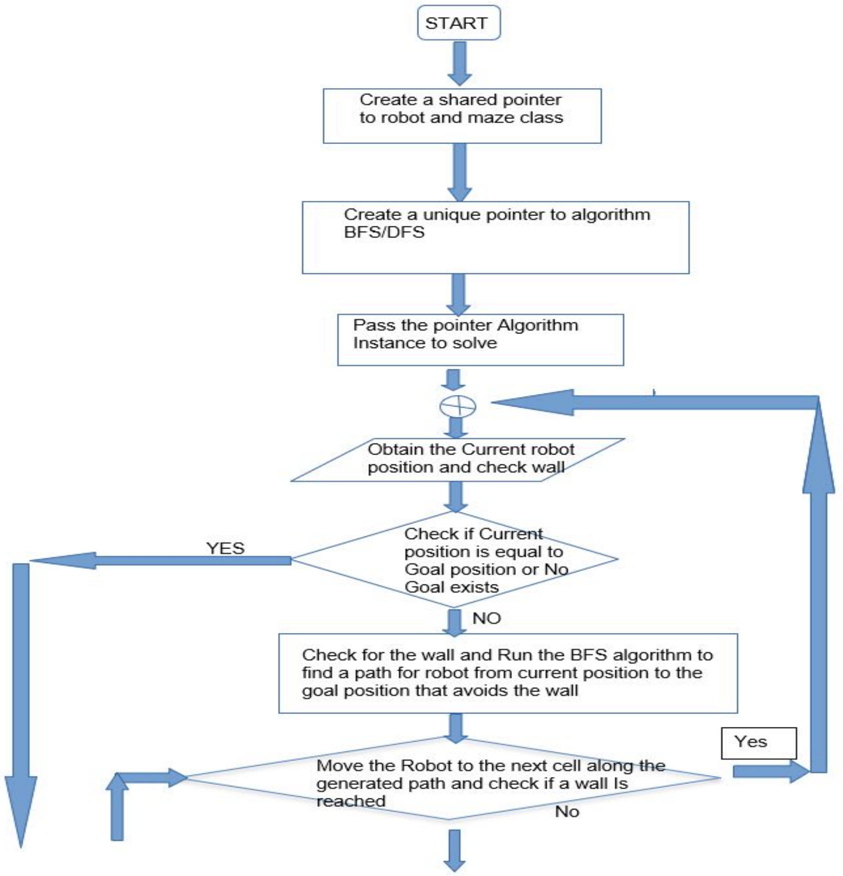
Mms simulator is used to interact with the maze.

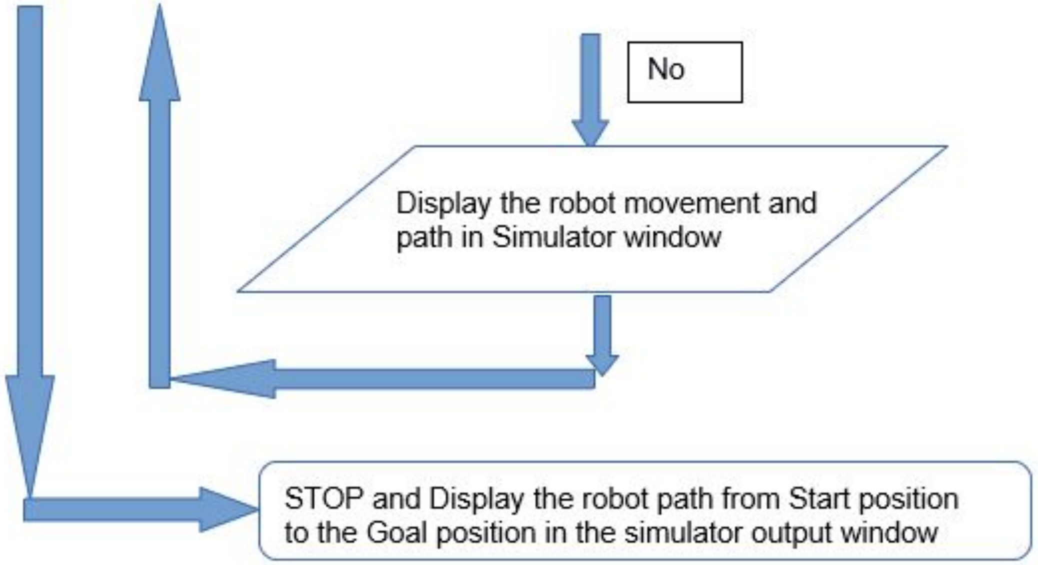
Class structure

- Algorithm
  - BFS
  - DFS
- API
- Direction
- Maze
- LandBasedRobot
  - LandBasedTracked
  - LandBasedWheeled



Process Flow


START
Create a shared pointer to robot and maze class
Create a unique pointer to algorithm BFS/DFS
Pass the pointer Algorithm Instance to solve
Obtain the Current robot position and check wall
Check if Current position is equal to Goal position or No Goal exists
YES
NO
Check for the wall and Run the BFS algorithm to find a path for robot from current position to the goal position that avoids the wall
Move the Robot to the next cell along the generated path and check if a wall Is reached
Yes
No


No
Display the robot movement and path in Simulator window
STOP and Display the robot path from Start position to the Goal position in the simulator output window

Technical Details

- Used unsigned char (8 bits) to store values in the maze
- Byte represented _ _ _ V/NV W S E N
- Used std::map to store the 2-D maze information
- For BFS, queue was used to store the visited nodes
- For DFS, stack was used to store the visited nodes

Conclusion

- Successfully implemented BFS and DFS algorithm to solve the path problem.
- OOP based code development
- BFS is better than DFS!!
    - Why?
        - Goal is closer to start - Fast search convergence with BFS
        - BFS does not depend on heuristic chosen (Down, Right, Up, South) whereas DFS does.
        - This was also verified empirically with all the 5 mazes. DFS solved the problem slow (atleast 5 times) than BFS.

Future Improvements

- We could implement more path planning algorithms so as to find the optimal path.
- Algorithms like
    - Dijkstra
    - A*
- Change the mms simulator to add dynamic obstacles on already explored locations.

References

1. BFS algorithm - https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
2. DFS algorithm - https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
3. Mms simulator - https://github.com/mackorone/mms
4. C++ Programming book - https://books.goalkicker.com/CPlusPlusBook/
5. Object Oriented Programming concepts - https://beginnersbook.com/2017/08/cpp-oops-concepts/

Thank You

<u>Group 10</u>

Rachith Prakash

Prasanna Balasubramanian

Alexandre Filie

Govind Ajith Kumar

Dinesh Kadirimangalam

Abhiram Dapke