Project Presentation on

Maze Solving Software module (Using C++)

ENPM809Y Introductory Robot Programming

University of Maryland, College Park

GUIDED BY:
Prof. Zeid Kootbolly

PREPARED BY:
Group 4

Main Focus & Development Steps

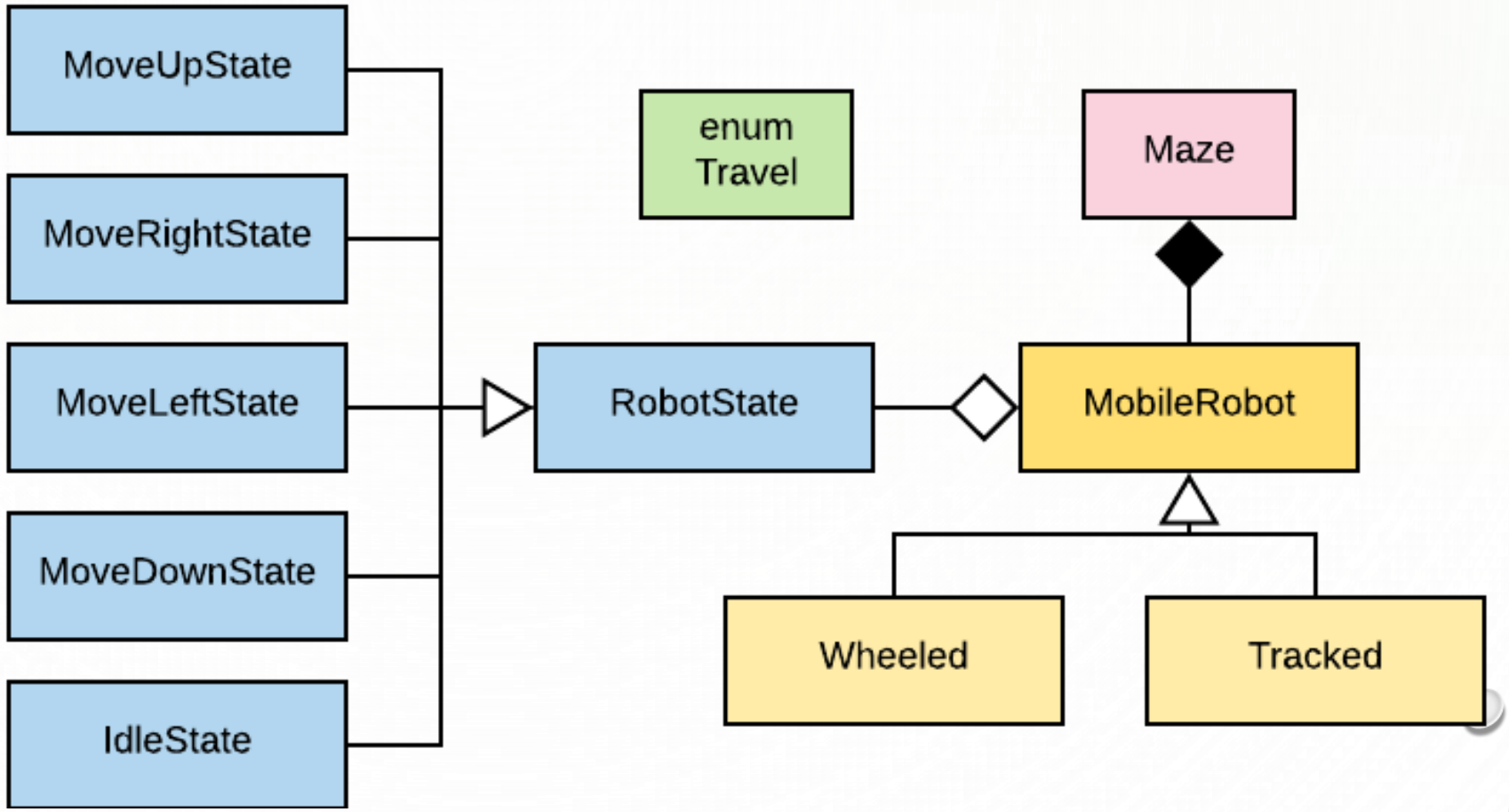
- Code Reusability
- Finding a Feasible Path (Not the shortest one)
- Focusing on Managing Class Objects
- Encapsulation & Abstraction
- Using C+11 features wherever we were able to use…

- Steps:
  - Started with development of UML Class diagram & Activity Diagram
  - Coded based on the diagram & requirements
  - Refactored UML diagrams and re-coded again

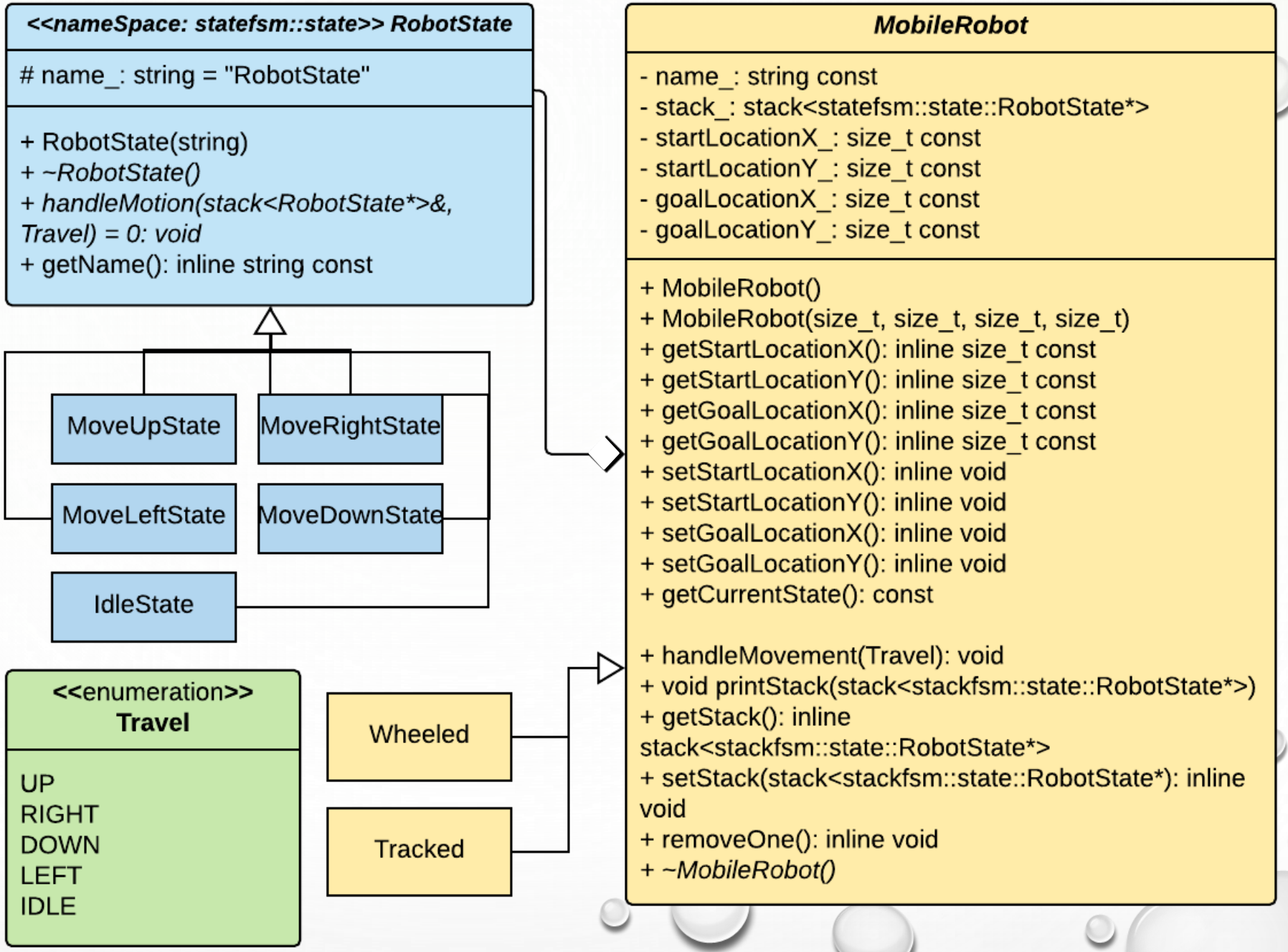
Requirements

- Start & Goal location coordinates shouldn't be same i.e. the user inputs must be different for a robot and its target.
  - Start coordinates for both the robots can be same, denoted by 'S'.
  - Goal coordinates for both the robots can be same, denoted by 'G'.
  - If all coordinates are different, the locations are denoted by the initials.
- If path doesn't exist for both the robots print the message and re-enter the coordinates.
- Solution path for wheeled robot is represented by '-'.
- Solution path for tracked robot is represented by '|'.
- If solution path overlaps, then it's represented by '+'.
- Map origin (Different from vector's starting location, problem with y-coordinates)
- Usage of Pushdown automata (FSM + STACK).
- Need for the use of dynamic dispatch to control the robots in the maze.

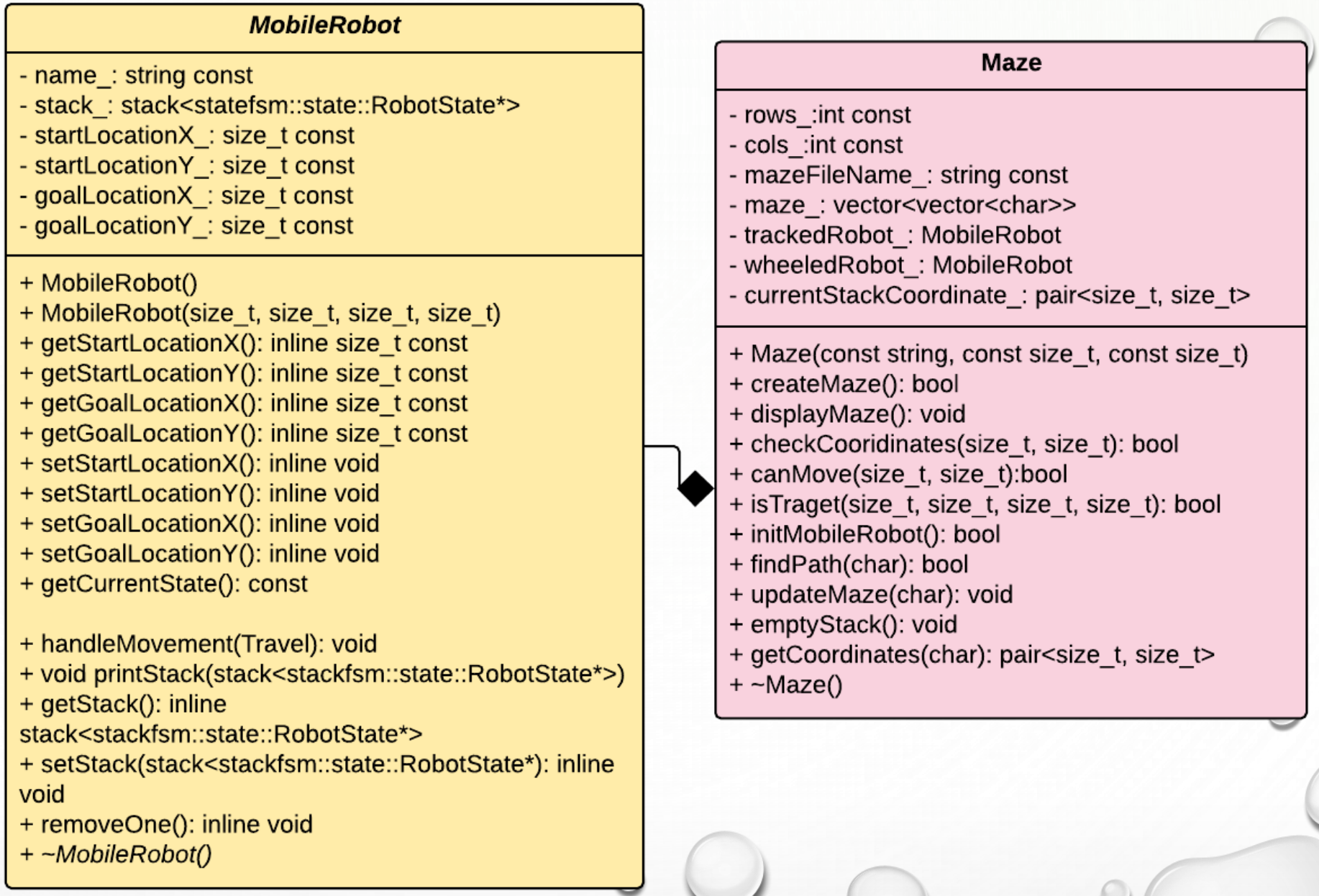
- Main Focus
- Development Steps
- Requirements
- UML Class Diagram
- UML Activity Diagram
- FSM Diagram
- Path Algorithm
- Maze and User inputs
- Class RobotState
- Methods
- Bug 1
- Approach
- Future Work

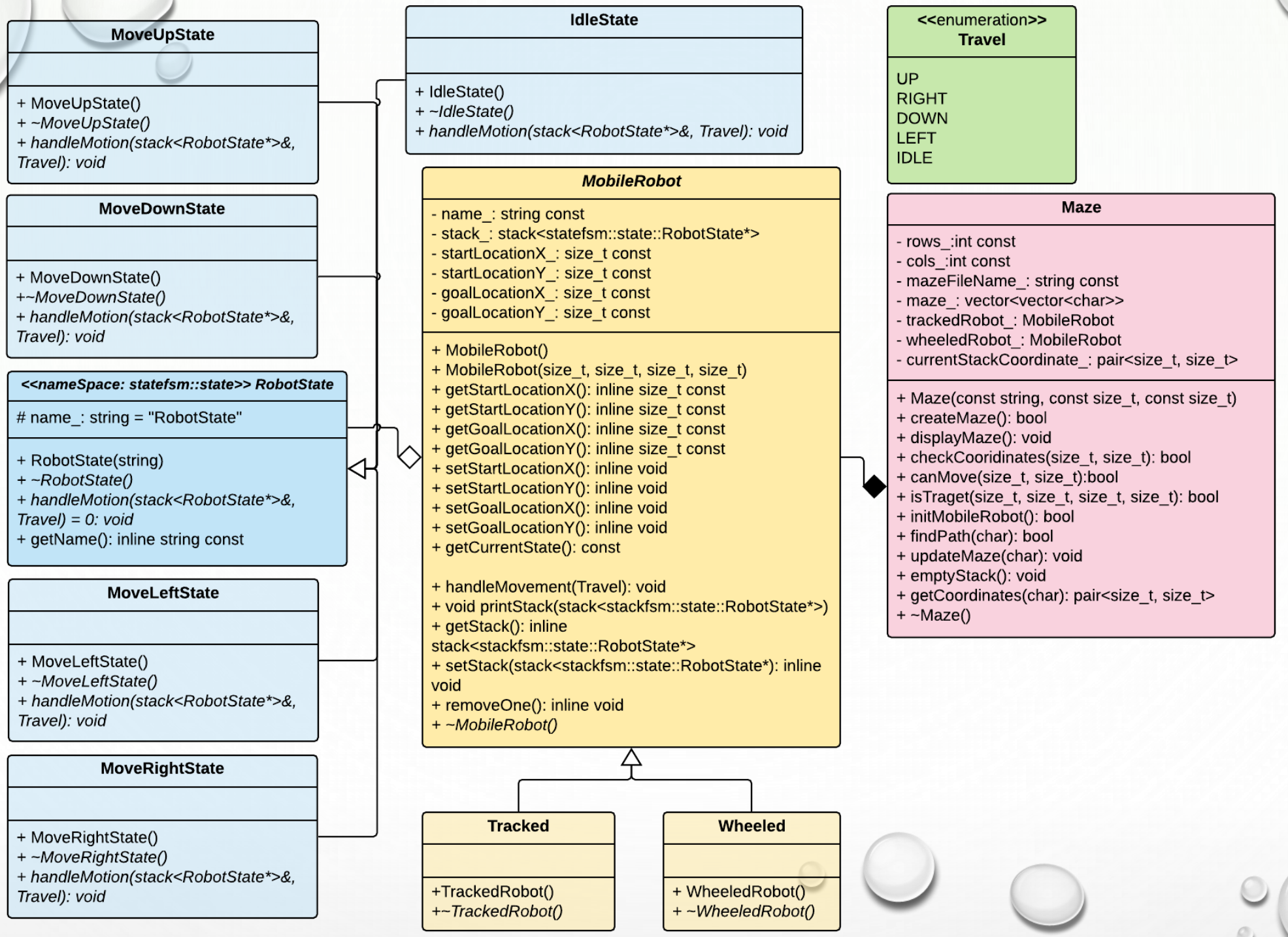


- Main Focus
- Development Steps
- Requirements
- UML Class Diagram
- UML Activity Diagram
- FSM Diagram
- Path Algorithm
- Maze and User inputs
- Class RobotState
- Methods
- Bug 1
- Approach
- Future Work

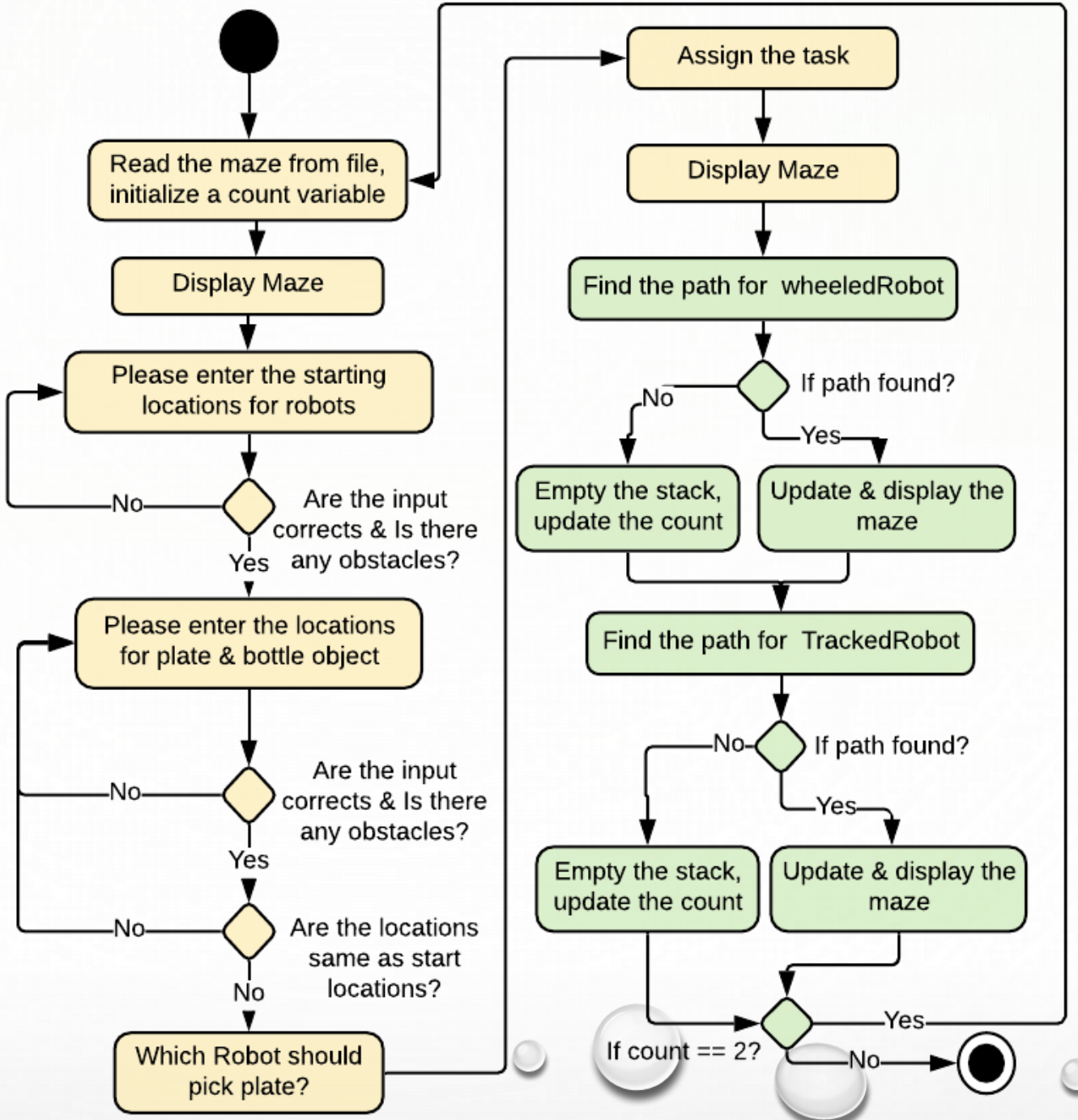


- Main Focus
- Development Steps
- Requirements
- UML Class Diagram
- UML Activity Diagram
- FSM Diagram
- Path Algorithm
- Maze and User inputs
- Class RobotState
- Methods
- Bug 1
- Approach
- Future Work

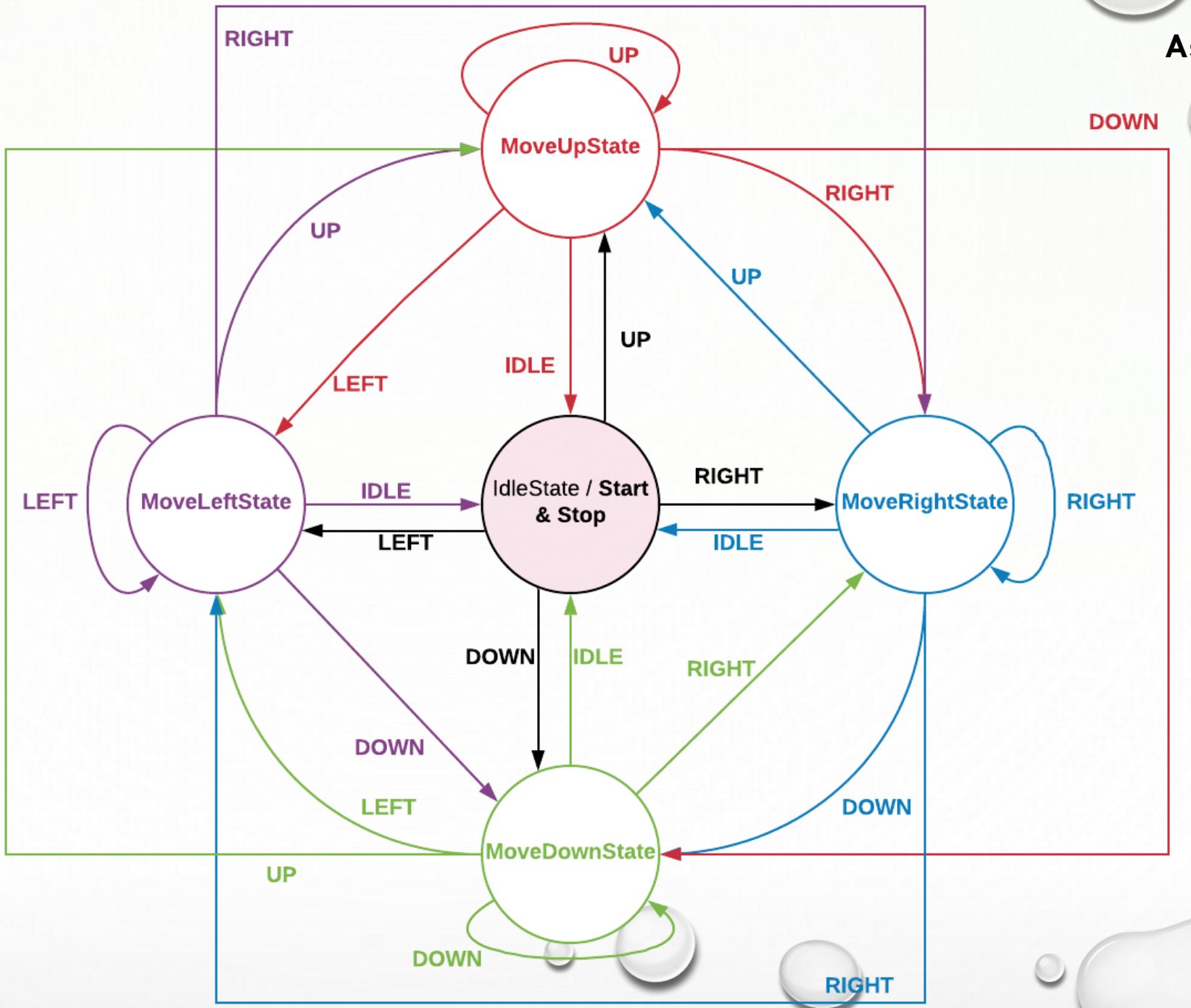



MoveUpState
+ MoveUpState()
+ ~MoveUpState()
+ handleMotion(stack<RobotState*>&, Travel): void
IdleState
+ IdleState()
+ ~IdleState()
+ handleMotion(stack<RobotState*>&, Travel): void
<<enumeration>>
Travel
UP
RIGHT
DOWN
LEFT
IDLE
MoveDownState
+ MoveDownState()
+~MoveDownState()
+ handleMotion(stack<RobotState*>&, Travel): void
MobileRobot
- name_: string const
- stack_: stack<statefsm::state::RobotState*>
- startLocationX_: size_t const
- startLocationY_: size_t const
- goalLocationX_: size_t const
- goalLocationY_: size_t const
+ MobileRobot()
+ MobileRobot(size_t, size_t, size_t, size_t)
+ getStartLocationX(): inline size_t const
+ getStartLocationY(): inline size_t const
+ getGoalLocationX(): inline size_t const
+ getGoalLocationY(): inline size_t const
+ setStartLocationX(): inline void
+ setStartLocationY(): inline void
+ setGoalLocationX(): inline void
+ setGoalLocationY(): inline void
+ getCurrentState(): const
+ handleMovement(Travel): void
+ void printStack(stack<stackfsm::state::RobotState*>)
+ getStack(): inline stack<stackfsm::state::RobotState*>
+ setStack(stack<stackfsm::state::RobotState*): inline void
+ removeOne(): inline void
+ ~MobileRobot()
Maze
- rows_:int const
- cols_:int const
- mazeFileName_: string const
- maze_: vector<vector<char>>
- trackedRobot_: MobileRobot
- wheeledRobot_: MobileRobot
- currentStackCoordinate_: pair<size_t, size_t>
+ Maze(const string, const size_t, const size_t)
+ createMaze(): bool
+ displayMaze(): void
+ checkCooridinates(size_t, size_t): bool
+ canMove(size_t, size_t):bool
+ isTraget(size_t, size_t, size_t, size_t): bool
+ initMobileRobot(): bool
+ findPath(char): bool
+ updateMaze(char): void
+ emptyStack(): void
+ getCoordinates(char): pair<size_t, size_t>
+ ~Maze()
<<nameSpace: statefsm::state>> RobotState
name_: string = "RobotState"
+ RobotState(string)
+ ~RobotState()
+ handleMotion(stack<RobotState*>&, Travel) = 0: void
+ getName(): inline string const
MoveLeftState
+ MoveLeftState()
+ ~MoveLeftState()
+ handleMotion(stack<RobotState*>&, Travel): void
MoveRightState
+ MoveRightState()
+ ~MoveRightState()
+ handleMotion(stack<RobotState*>&, Travel): void
Tracked
+TrackedRobot()
+~TrackedRobot()
Wheeled
+ WheeledRobot()
+ ~WheeledRobot()

- Main Focus
- Development Steps
- Requirements
- UML Class Diagram
- UML Activity Diagram
- FSM Diagram
- Path Algorithm
- Maze and User inputs
- Class RobotState
- Methods
- Bug 1
- Approach
- Future Work



- Main Focus
- Development Steps
- Requirements
- UML Class Diagram
- UML Activity Diagram
- FSM Diagram
- Path Algorithm
- Maze and User inputs
- Class RobotState
- Methods
- Bug 1
- Approach
- Future Work



Path Search Algorithm

- Use two stack, one trailStack & currentState stack_, which is basically the stack in MobileRobot class. (Stores actions)
- Two additional stack, one for trailStackCoordinate & other for currentStackCoordinate_, which is basically the stack in Maze class. (Stores Coordinates)
- Along with this stack and finite state machines (i.e. RobotState Class), we are implementing pushdown automata.

Note: Atleast three is needed to efficiently implement planning algorithm.

Path Search Algorithm

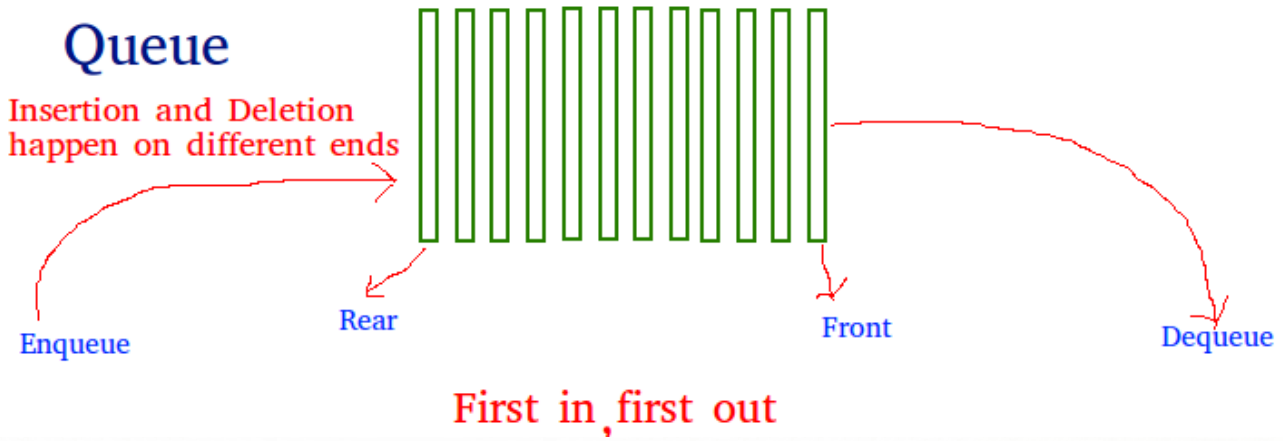
Ashish

- Empty the memory of the robot. (i.e. trailStack & CurrentStack)
- Push the start cell on the trailStack
- Enter this cell into the "visited" set
- While the trailStack is not empty
  - Initialize and declare the variable "stuck = true"
  - Pop a cell from trailStack into a variable
  - Push that variable to CurrentStack (handleMotion)
  - Explore the neighbors and if valid coordinates push it into trailStack
    - For each neighbor check whether goal is reached or not
      - If reached push IDLE state and print the stack (return back from function) && stuck = false
  - If stuck, backtrack the cell by comparing the topCurrent & topTrail cell. Remove a topCurrent cell till the topTrail and topCurrent cell are neighbours

Breadth First Search

Lih-Narn Wang

1. Queue



2. 2D Vector: Visited and Parent

True	True	True	True	False
True	True	True	True	False
True	True	True	False	False
True	True	True	False	False
False	False	False	False	False

Figure Visited

0	0	0101		
0	0	0102		
0101	0102	0103		

Figure Parent

Breadth First Search

Lih-Narn Wang

3. Hash Table

0	0	0101		
0	0	0102		
0101	0102	0103		

Figure Parent

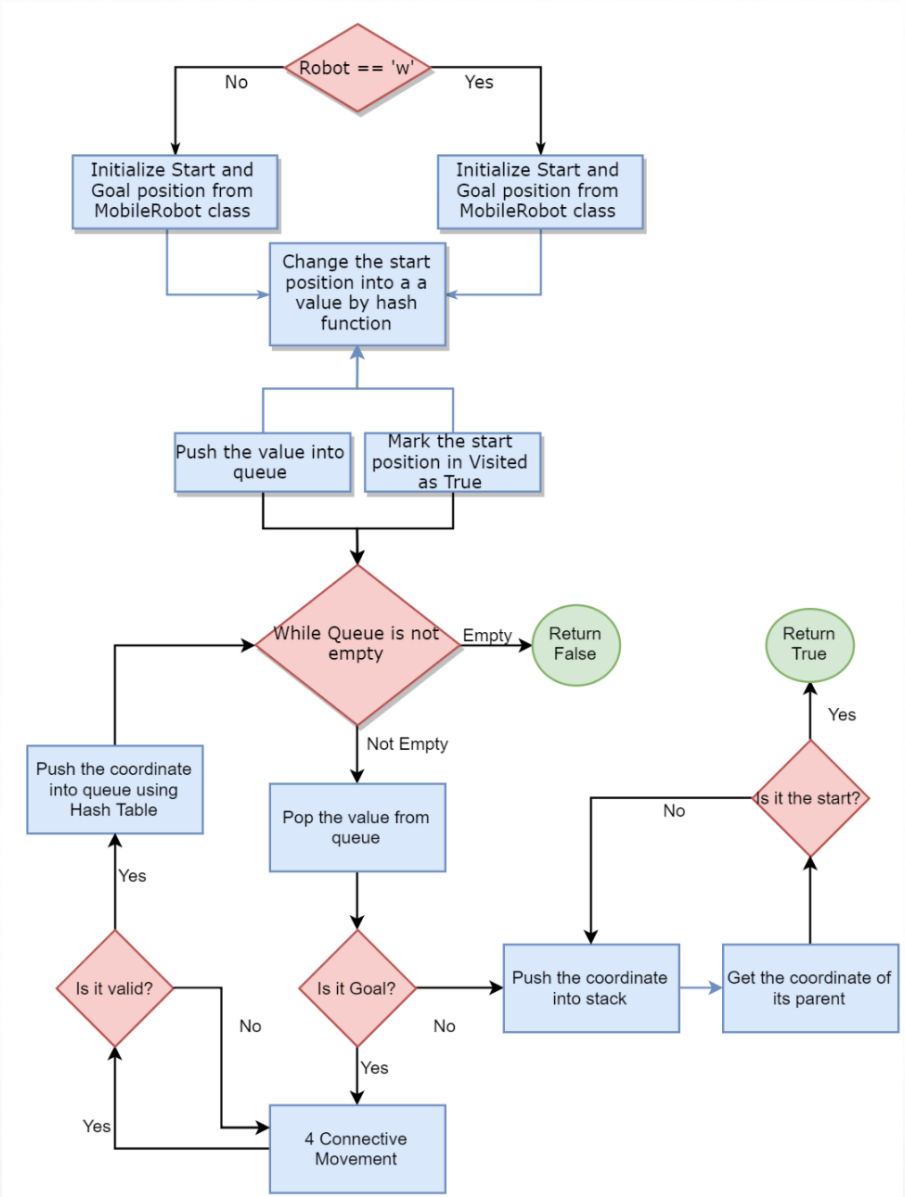
Using hash function to store the information about the index of the parent.

Hash function:

$$key = Row * 100 + Column$$

Breadth First Search

Lih-Narn Wang



Ending Condition:

1. Empty queue.
   Return False

2. Back Track to the start.
   Return True

Reading the Maze

Class Maze

In built functions used to read text file – **ifstream** and **getline.**

Maze
- rows_: int const - cols_: int const - mazeFileName_: string const - maze_vector<vector<char>> - MobileRobot tracked - MobileRobot wheeled
+ createMaze() + initMobilerRobot() + displayMaze() + checkCoordinates() + canMove()

Methods	Purpose
createMaze()	Creates the maze by reading the text file
displayMaze()	Prints the maze along with the coordinates on margin
checkCoordinates()	Checks if the coordinates lie in the bounds of the map and returns Boolean value
initMobilerRobot()	Assignment of coordinates
canMove()	Checks if the coordinates lie in the wall (obstacle) and returns Boolean value

User inputs

The following inputs need to be taken:

1) Start point of the Wheeled robot
2) Start point of the Tracked Robot
3) Position of the Target Plate
4) Position of the Target bottle

→ Data type – Pair

Checks performed:

- If the input is an integer using the method – getCoordinates()
- If the input is in the bounds of the map – checkCoordinates()
- If the input lies in obstacle space – checkCoordinates()

5) The robot with target as plate

→ Data type – char

Assigns:

The second robot the remaining target

Checks performed:

- If the start and goal coincide for either of the robots

Class Mobile Robot

- To assign the start and goal to each robot, a class called MobileRobot was defined.
- It contains the following members:
  1. **Methods**
     1. Setters for Start and Goal points.
     2. Getters for Start and Goal points
  2. **Attributes**

     StartLocationX, StartLocationY, goalLocationX, goalLocationY

- *Objects of MobileRobot namely Wheeled and Tracked were created as attributes in the maze class.*

Class RobotState

- Abstract (base) class
- It contains the following members:
  1. **Methods**
     1. Constructor and destructor.
     2. A virtual method, "HandleMotion", which will be overridden by the derived classes.
     3. A *getName* method, which can access the *name_* attribute.
  2. **Attributes**
     1. *name_* attribute, which is assigned a default value depending on the type of pointers formed.

Class RobotState (continued)

- The classes derived from class *RobotState* are as follows:
  1. *StandState*
  2. *MoveUpState*
  3. *MoveRightState*
  4. *MoveLeftState*
  5. *MoveDownState*
- All of the above contain a constructor and destructor, and a virtual method, *HandleMotion*.
- *travel.h* is included in the base and derived classes.

The HandleMotion Methods

- There are two variants of the *HandleMotion* method.
- The first is present in the class *MobileRobot* and the second is in the class *RobotState*.

MobileRobot::HandleMotion

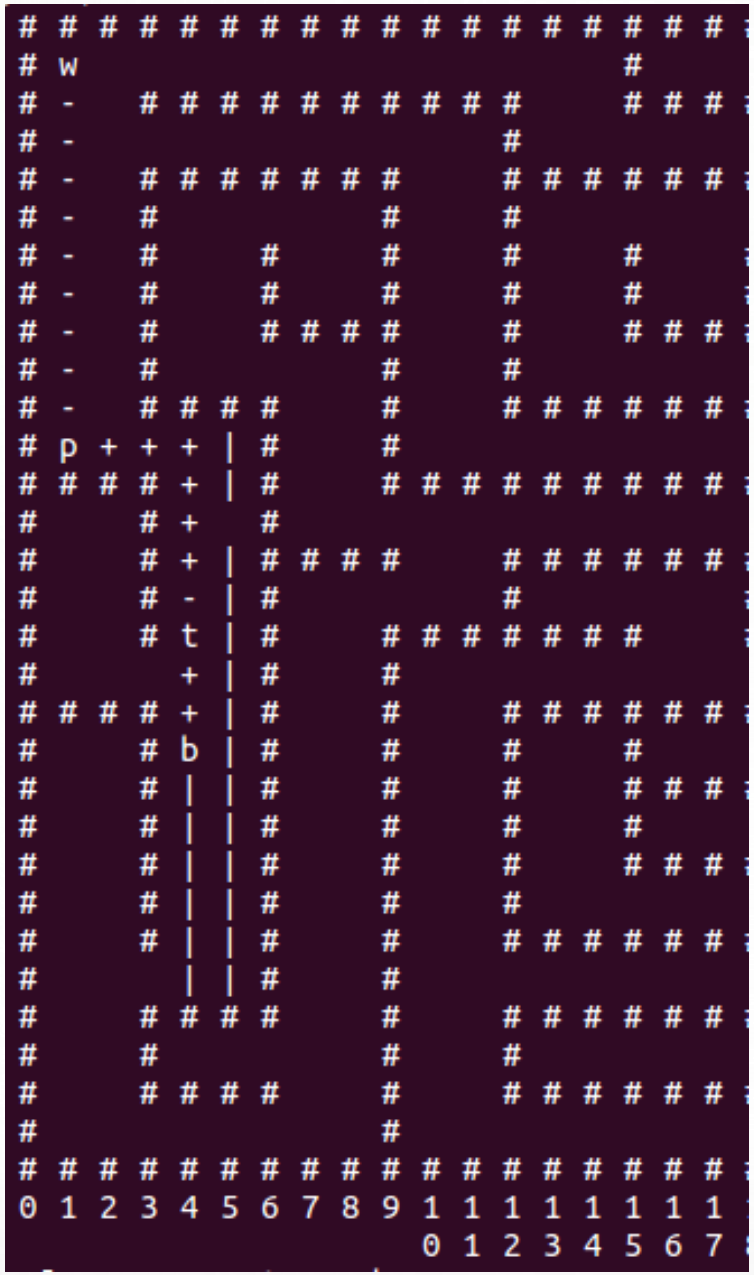
- **Prototype:** *void HandleMotion(Travel)*
- Among other members, class *Maze* has a method, *FindPath*, and an attribute, which is an object of class *MobileRobot*.
- *FindPath* is called in main.cpp (through the object of class *Maze*) and *HandleMotion* is called from within *FindPath* using the object attribute.

MobileRobot::HandleMotion (continued)

- **Prototype:** *void handleMotion(Travel)*
- Takes in an enum value and updates the *stack_* attribute of class *MobileRobot*.
- This *stack_* attribute is a stack of pointers of class *RobotState*.
- It updates *stack_* by calling the *RobotState::HandleMotion* method.
- This is where Dynamic Polymorphism takes place.
- It calls the version of the *HandleMotion* method based on the class of pointer stored at the top of the *stack_* attribute.

RobotState::HandleMotion

- **Prototype:** *virtual void handleMotion(std::stack<RobotState*>&, Travel)*
- Takes in a stack of pointers of type *RobotState* (base class), and an enum value.
- Note that the stack is passed by reference, and this stack is the attribute of class *MobileRobot* as mentioned earlier.
- The stack is updated by adding *new* pointers in the *stack_* attribute depending on the value of *Travel*.
- This is achieved using if/else statements.




```
[STACK]: StandState
[STACK]: MoveLeftState
[STACK]: MoveLeftState
[STACK]: MoveLeftState
[STACK]: MoveUpState
[STACK]: MoveUpState
[STACK]: MoveRightState
[STACK]: MoveUpState
[STACK]: MoveUpState
[STACK]: MoveLeftState
[STACK]: MoveUpState
[STACK]: MoveUpState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveRightState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: MoveDownState
[STACK]: StandState
```

Path from tracked Robot to Plate object (t to p)

1. Resolved the stack error by storing it afterwards using coordinates.
2. Add a condition to check the direction while retracing.

Future Work

- Incorporating a search algorithm that can find the shortest path.
- Modifying a Maze class into two separate classes, one for Path Search algorithm and other for Maze.
- Using Advanced C++ features