We created a vacuum which navigates using BFS instead of rule-based, as we did not manage to create a rule-based vacuum without having major problems.

The vacuum works by trying to follow a route, and when it doesn't have a route. It requests a new route from the BFS function. If the BFS function returns no more routes. We know that it's done. And we return home.

If there is a route, the vacuum takes the next tile it has to move to. And then rotate right until it points at that tile. It then moves forward and deletes the node from its path. This continues until the BFS can't find any more tiles and returns a path to home. The vacuum then turns off when it's home.

The interesting bit in the code, which handles all the navigation, is the BFS function. Its the following code:

```
//Search for new paths with BFS
public ArrayList BFS(Pos startPos) {
    System.out.println("start_node:" + startPos);

    //Creates a arraylist of arraylists of pos
    //Then add a arraylist with only the start position
    ArrayList<ArrayList<Pos>> queued = new ArrayList<ArrayList<Pos>>();
    ArrayList temp = new ArrayList<Pos>();
    temp.add(new Pos(startPos.x,startPos.y));
    queued.add(temp);

    //A map of visited nodes
    boolean[][] visited = new boolean[state.world.length][state.world[0].length];

    //Cont as long as there is more to be explored
    while(!queued.isEmpty()){
        //Take first element
        ArrayList<Pos> path = queued.remove(0);
        Pos pos = path.get(path.size()-1);
        System.out.println("SEEEE MEEE!!!!!" + pos);
        //If we find a pos which is unknown when cleaning
        //Or home when done cleaning. Return the path as we wanna explore that place
        if (state.world[pos.x][pos.y] == state.UNKNOWN || (state.finished == true && state.world[pos.x][pos.y] == state.HOME)){
            path.remove(0);
            System.out.println("path in bfs" + path);
            return path;
        }
        //Go through all neighbors and add all neighbors that is not visited/wall/invalid position, to a new queue.
        Pos[] neighbors = getNeighborsAbs(pos);
        for(Pos neighbor: neighbors) {
            if (visited[neighbor.x][neighbor.y] == false && state.world[neighbor.x][neighbor.y] != state.WALL && isValidPosition(neighbor)) {
                ArrayList<Pos> new_path = new ArrayList<>(path);
                new_path.add(neighbor);
                queued.add(new_path);
                visited[neighbor.x][neighbor.y] = true;
            }

        }
    }
    //Return empty list
    return new ArrayList<>();
}
```

The code is described with comments, so it should be easy to see how it works.