### Report TDDC17-Lab1

### Task 1:

### Using the template implement of MyVacuumAgent.java (or myvacuumagent.py) we implemented an agent that follows the basic idea of a horizontal zigzag pattern which starts at the top left corner. The vacuum cleaner just moves in horizontal lines until he bumps against a vertical wall. Then he changes the direction by 180 degree by turning twice in the same direction. With this simple algorithm the vacuum agent is able to search systematically through the whole world and update the world model based on the perception.

### This approach is limited to the assumption that our world is a rectangle where we can simply move to the left top corner by moving north and west until we bump in each direction. Moreover, this algorithm is not able to handle obstacles within the world. In this case it would assume that the obstacle is a wall at the end of the world, and we would lose everything that is on the same vertical latitude but behind the obstacle.

### All in all, is the zigzag approach for the stated assumptions a solid method that is efficient in exploring the whole world and thereby sucking in all the dirt. Although the assumption that there are no obstacles in the world and that the world is a rectangle are strong and will be addressed further in Task 2.

### Task 2:

### In Task 2 we are additionally considering obstacles in the world. Therefore, an effective algorithm should be able explore the whole world even if going straight lines is not possible. This requirement increases the complexity significantly. This problem is a classic search problem and can be solved by applying a search algorithm that extensively seeks all fields or paths. Since we had not requirement concerning efficiency our goal was it to find a search method that is complete i.e. that it guarantees to find a goal state if one exists. Breadth-first is the most simple and popular representative algorithm for this problem.

### The Breadth First Search algorithm (BFS) starts at some arbitrary node of a graph, and visits all of the neighbour nodes at the present level prior to moving on to the next depth level. In our world model we start at one random node (after running the random actions in the beginning). Then we explore the start node which includes visiting all neighbour nodes. Each neighbour node that is visited (and no wall or bump) and not explored is added to the queue which we want to explore later. Thereby we collect all neighbours in the queue until this depth level is fully explored. Then we pick the next node in the queue (i.e. move one level deeper in the search) and explore this node by visiting its neighbours and adding them if necessary, to the queue. So, we have a backtracking and move explore complete graph levels before digging deeper in the world. This is the opposite to the depth search algorithm that moves one path as deep as possible before exploring another path.

### For our implementation we used a non-recursive implementation. We use a queue (First In First Out) and save our discovering progress in the state / world model to remember whether we have explored nodes and where walls are. Also we used a map that remembers for a node his previous node. This allows the agent to get back at his start position or to the goal.

### At the end our agent has a fully updated world model as output.

### The performance of our agent is limited due to a simplification in the implementation. To move when changing the currently node of interest which is getting explored it is sometimes necessary to move more than just one field. In this case we go remember the path from the global start point. Therefore, we then move back to the start and then to the goal node. This is effective because we use our progress in the state / world model but it is not efficient since we sometimes have short ways to move that get unnecessarily long when we go back to the reference node (start node). This could be improved easily by implementing a more advanced path memory or a heuristic approach. Apart from that our BFS implementation is much more efficient in the way it is searching than the reactive agent because it is systematically searching and more effective because it is a complete search algorithm.