# Lab 1 : Intelligent Agents

## Task 1

### Behaviour mechanism description

The first solution was designed to be a finite state machine. Four different agent modes were introduced, which describe alternative chain of actions to perform in order to reach the goal. The mode is chosen depending on the current state of surrounding environment, which is stored in boolean variables (percepts of the agent), and last action performed.

Below short description of the modes are presented:

- FORWARD: moves forward until bump
- TURN: depending on the last turn, turns right or left in order to continue “snake” movement on the map
- GO_START_POSITION: after calculating needed number of steps in both x and y axis goes to the start position (position from where we launched after initial random moves)
- GO_HOME: performs the same actions as in mode above, but the start position coordinates are always the same (HOME position (1,1))

The agent goes forward until it reaches the wall, then it performs two turns in the row in order to get to the row below and continue its journey. It continues described “snake” strategy movement until it detects two bumps in a row, which implies reaching the end of the map. Then, the agent goes back to start position and executes the “snake” in the opposite direction to be sure that it covers the whole map. Once this task is completed it goes back to home position.

Regarding memory allocation, in this approach only simple variables were used in order to store initial information about the agent’s position and direction after initial random moves.

### Performance

In comparison to the reactive agent our solution assures that every position in presented world will be checked and cleaned if necessary, whereas the reactive agent simply follows only the border of the map missing the central part of it. On the basis of the performance score given at the end of running our simple simulator we can state that our solution is much more efficient. The difference in scores of these two agents varies between 700 and 1000.

### Why this implementation ?

We wanted to implement our agent in the simplest way possible, so we think that the finite state machine with multiples modes was the best way to do so. We didn’t want a large amount of data for our agent, it’s following a path and every time there is dirt, it sucks it, and then the agent is going back home.

# Task 2

## Behaviour mechanism description

Basically, the agent has a goal. Every step, the agent wants to reach its goal. When the goal is reached, the agent find a new goal and try to reach it again and so on. When no more goal is found, then the agent is going back to its home position.
Our agent is using search algorithm, more precisely, a Depth-First Search in a Tree structure.

## Data stored in the agent

- `tree` : data structure containing all informations about the current environment
    - A tree has a root. This root is a `Node` containing coordinates (x and y), four possibles sons and a parent.
- `visited` : set of visited nodes
- `goal` : the goal node
- `path` : list containing the path in order to reach the goal

At each step, the agent is extending its knowledge, so the tree is growing as the agent discover new possibilities.

## Performance

The algorithm works but it's not quite efficient. As a matter of fact, the agent does't have enough moves in order to discover every squares of the map due to the constraint of the `iterationCounter` set to [width x height x 2]. However, putting that aspect aside, the agent achieves all of its goals and go back to its home position.

## Why this implementation ?

We decided to use search strategy in our solution, because it doesn't follow one strict strategy every time, which was the case in Task 1. The agent's action depend on the current environment, it uses already discovered paths in order to avoid obstacles and reach the given goal.

## Improvement clue

We could have used a Graph instead of a Tree to search the goal. That would have been more efficient to find the shortest path between two differents points in the map. As said above, the Tree is not quite efficient to find the shortest path. Indeed, with Graph structure we could have avoided creating many nodes representing the same position on the map.