Project 1 - Report

Name : Hugo Duret
Student ID : 20555806
Due date : Sep 20, 2019

This report is for the project 1 given in the course 4350 - Artificial Intelligence.
It contains :

1. the list of files submitted, which are the source code of the program
2. instructions for running your program
3. screenshots of sample runs
4. additional information

I) List of files submitted

- this report
- source_code/
    - main.cpp
        - *just launch the GUI from mainwindow files*
    - agents.h / agents.cpp
        - *agent strategies*
    - mainwindow.h / mainwindow.cpp
        - *main functions*
        - *simulation management*
        - *benchmark management*
        - *GUI management*
    - world_map.h / world_map.cpp
        - *world data structure*
- release/ :
    - VacuumCleaner.exe
    - .o files
    - list of .dlls files required to run the application

I have made sure that all required files are in the folder I submitted. I have tested the application on the campus computers where Qt is not installed, and it works perfectly. If it doesn't work on your computer, please let me know as soon as possible. I know that you said "No programming assignment will be graded if the submitted program does not run.", but I consider that if it runs on the campus computers it should work with no problem on your computer.

II) Instructions for running your programming

To launch the application simply launch the VacuumCleaner.exe file. A Graphical User Interface has been implemented, it is shown in *Figure 1.*

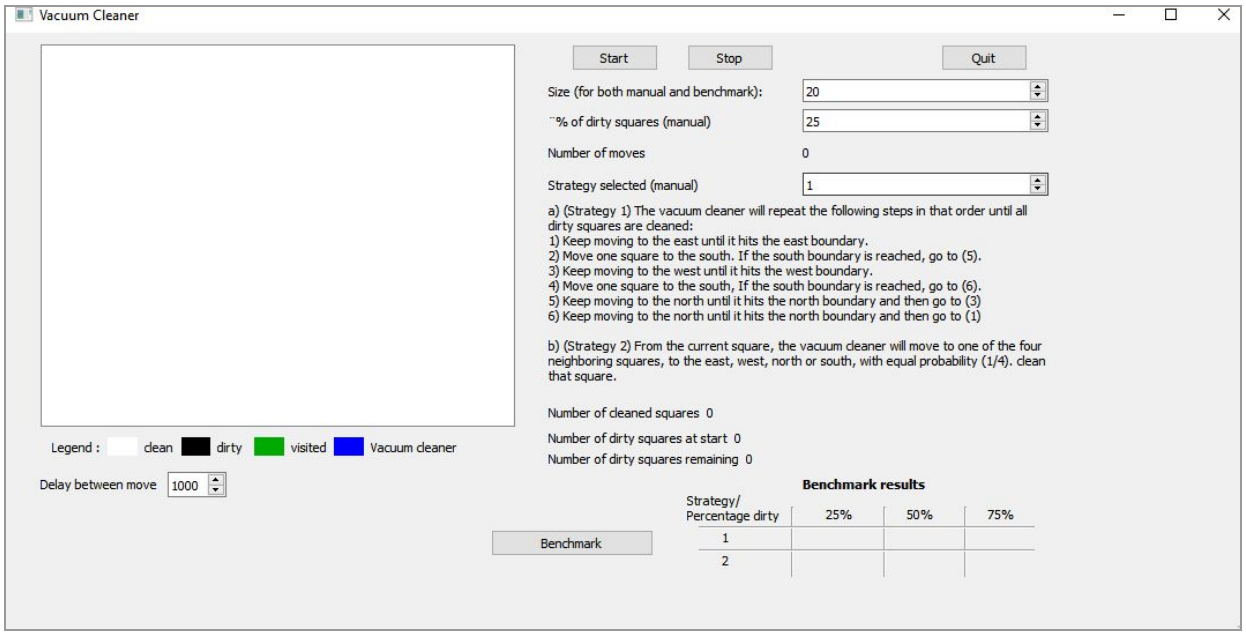


Figure 1 : GUI

The application can be used in two different ways : manual and benchmark.

In manual mode, the user can set the **size** of the squared world. The **size** is the number of squares on one side of the world, it goes from five to twenty. The user can also set the **percentage of dirty squares** at the beginning of the simulation, three values are possible : 25%, 50% and 75%. Finally, the user can choose one of the two strategies, as explained in the project description.

For changes in these settings to be taken into account, one must *Stop* the simulation if it is running. Otherwise the GUI will update, but not the variables.

Once the settings are set, click on *Start*. The simulation will start, showing the progress of the vacuum cleaner. It will stop when all dirty squares are cleaned, or after a long period of time (depending on the world's size) to avoid getting stuck in an infinite loop. The *Number of moves* is displayed, as well as information on the status of the squares (clean, dirty). To change the speed of the simulation, one can change the *Delay between move* value.

To launch a new simulation, wait for the current one to terminate, or click on *Stop*. Change the settings, and click on *Start*.

In benchmark mode, the user can only choose the size of the world. Then, click on *Benchmark*. The process is all automatic and proceeds as follows :

- For each strategy (1, 2) :
    - For each percentage of dirt (25, 50, 75) :
        - Launch the simulation and count the number of moves necessary to clean all dirty squares
        - Repeat the step above five times, and calculate the average number of moves
- Print the results on the GUI

The *Quit* button quits the application.

III) Screenshots of sample runs

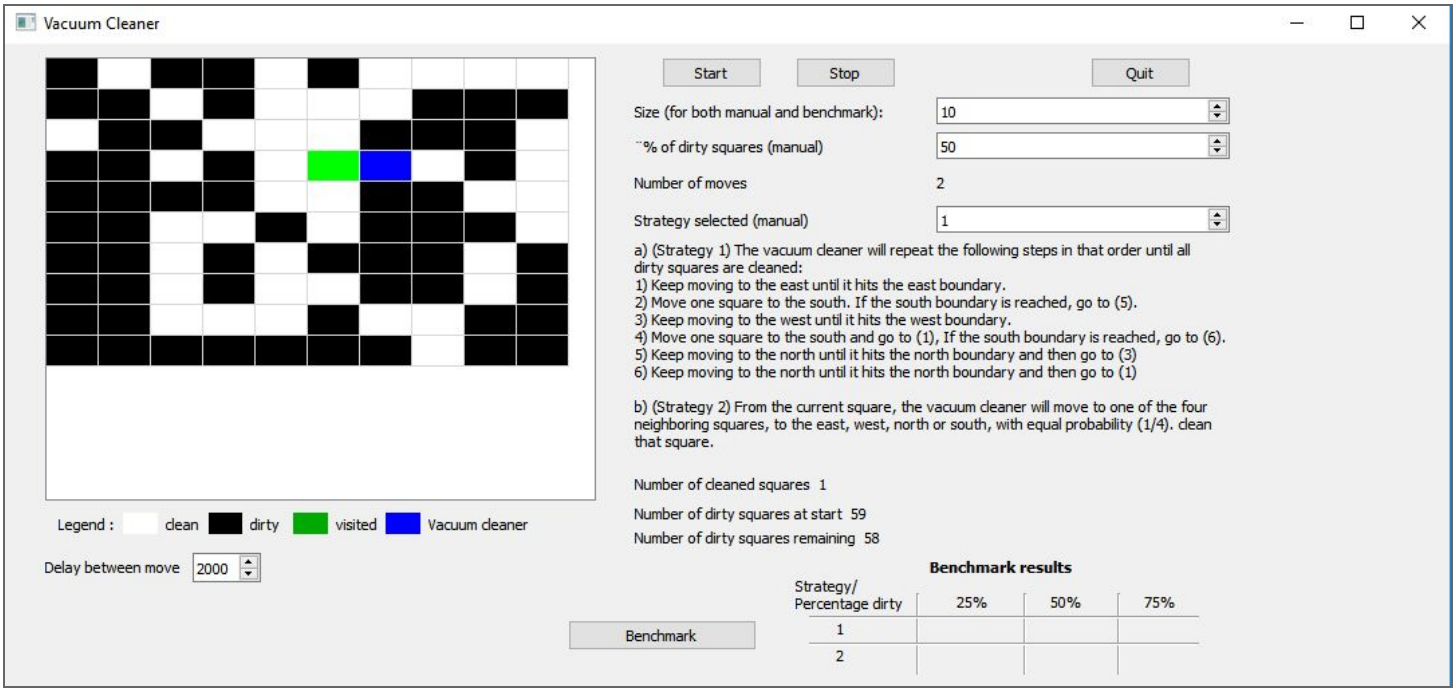
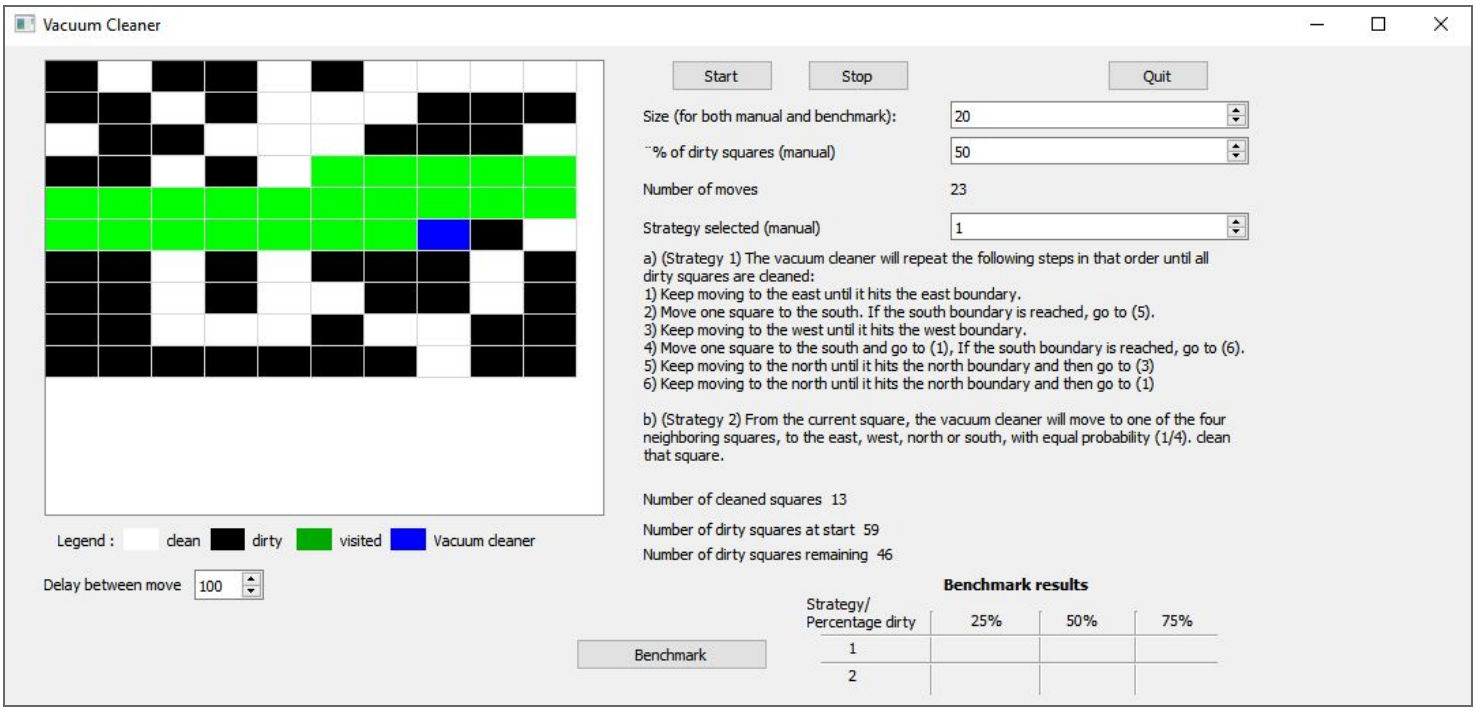


Figure 2 (above) : Simulation just after start for size 10, percentage of dirty squares 50%, and strategy one. Note that here the vacuum cleaner has started on a dirty square, and has cleaned it before moving to another square, so the number of moves is 2 (cleaning the first square counts as a move).

Figure 3 (below) : In the middle of the simulation. So far the vacuum cleaner has cleaned 13 squares out of 59, so 46 dirty squares are remaining.



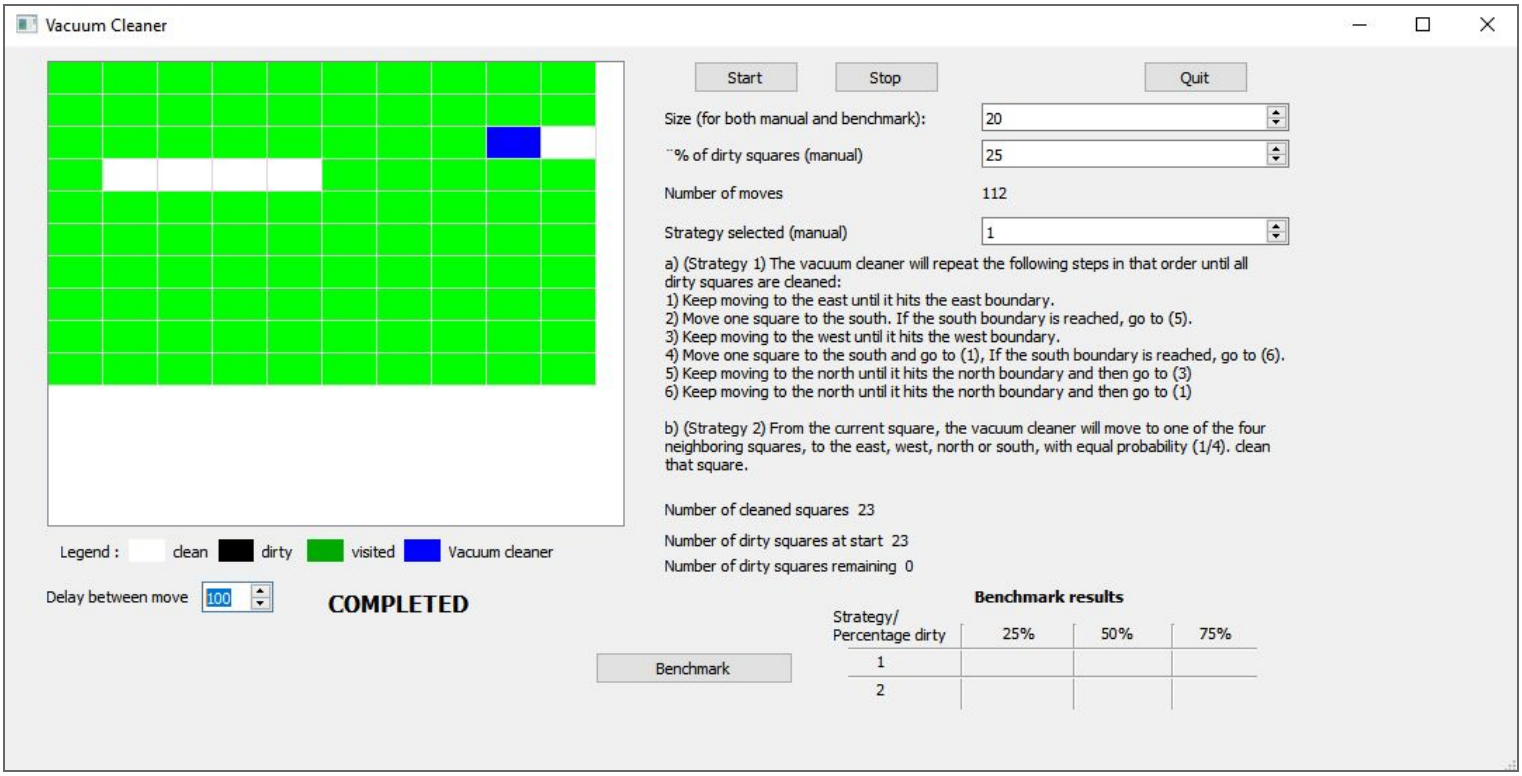


*Figure 4 (above) : End of the simulation. The vacuum cleaner stops when all dirty squares are cleaned. Here, the total number of moves necessary was 112 to clean the 23 dirty squares. The GUI shows a label **COMPLETED** when the simulation ends successfully.*

Benchmark results

Strategy/ Percentage dirty	25%	50%	75%
1	400.6	428.6	429.8
2	5286.2	6014	6442.4

Benchmark results

Strategy/ Percentage dirty	25%	50%	75%
1	110.4	114.4	114.2
2	846.8	833.2	975

Figure 5 (above) : Results of a benchmark for a world of size 20 (left) and size 10 (right).

IV) Additional information

1- Discussion on the benchmark results.

The benchmark provides some interesting information concerning the efficiency of each strategy (see results in *Figure 5*). One can also notice that the number of moves necessary to clean the world increases as the number of dirty squares increases, this result was quite predictable.

Concerning the strategies, there is a very clear difference of efficiency.

The first strategy tries to optimize its path to reduce the number of moves. The last should never be much greater than the size of the world. For example, for a percentage of dirty squares of 50% in a world of size 20, (that is, 400 squares), the vacuum cleaner will clean the world in about 427 moves. Some squares are visited twice, due to the strategy choices, but also from defects in the implementation (discussed later). Still, the point is that the total number of moves should always be more or less equal to the total number of squares in the world.

The second strategy is clearly less efficient. Due to a random choice of direction, the vacuum cleaner can visit a square many times, even if it had been cleaned after the first visit. As the number of squares in the world is **size²**, the number of useless moves greatly increases when the size of the world increases. With the results in *Figure 5*, the strategy one is about eight times more efficient than the strategy two, for a world of size 10. It also means that a square will be visited on average eight times. For a world of size 20, the vacuum cleaner spends more than 5,000 moves to clean the 400 squares, so on average a square is visited twelve times ! (Please note that as it is a random method, the last figure varies greatly between each benchmark.)

This benchmark demonstrates that the choice of strategy for an agent is very important. A random one has poor efficiency for small worlds, and becomes really inefficient for great instances. A strategy such as strategy one, which aims to minimize the number of moves by trying to visit each square once, proves to be quite efficient for such a problem, even if it can probably been improved.

2- Known issues

a) GUI refreshing

Launching a new simulation doesn't refresh completely the GUI grid. Consequently, some squares appear with the wrong color, and the simulation sometimes completes even if some black squares are still visible (but the squares in the data structure are clean, hence the vacuum cleaner stops). Also, at the end of the benchmark, the strategy choice is set to two, whatever the GUI shows. To resolve this issue, simply switch to two and then back to one. A solution to

avoid all these problems is to quit and restart the application between each simulation. Note that this doesn't affect the benchmark.

b) Hack for strategy one

In strategy one, when the east or west side is reached, we need to move south once before moving east or west again. I had a hard time trying to figure out how to do it. The current solution uses a variable *last_move* that remembers the last direction taken by the vacuum cleaner. It is used to tell the agent not to move south if it had moved south before. I am aware that our agent should not be aware of the world around it, and should act only with the percept (status, location). So this is a problem that I still need to fix in the future. Meanwhile, this hack only requires to remember one action, and seemed reasonable in terms of "illegal" solutions, as it can be easily implemented in a real situation.

c) Notes on implementation choices

Directions as integers instead of characters :

As it is much easier to use integers rather than characters or strings in C++, the cardinal directions are represented by integers as follows :

(Clockwise order) :

1	2	3	4
North	East	South	West

States as integers :

For the same reason, the squares' states are represented as follows :

0	1	2	3
Clean	Dirty	Visited	Vacuum cleaner

World data structure :

The world is represented as a matrix(size, size) thanks to the use of an array of arrays, where every element is an array of size two containing the values (state, location).

Simulating function :

The simulating function has the knowledge of how many squares are dirty in the area and will keep running the simulated agent function until all dirty squares are cleaned, or a certain number of moves has been made to avoid infinite loops.