Multi-agent Adversarial Vacuum Cleaner Agent

Hasan Basbous

Abed Dandan, Ali Saad

Abstract—This report focuses on extending the first project environment to hold more than one agent. Added to the vacuum cleaner agent is the "dirt producer" agent whose job is to dump dirt on the floor. This will push the agents into a competitive environment where each move should be calculated for each agent to reach its goal. In this project, three classes of agents were developed; the first holds two agents whose moves are optimal, each following TSP[2], in the sense of number of moves, and performance. The second class also holds two agents but with the use of memory, such each agent holds a memory of the prior visited nodes in case there were previous runs; this helps in reducing the complexity of calculations needed to make a move, still not as optimal as the first class, but saves on the processing power needed to run the agents. The third class is a holding environment of four agents, two vacuum cleaning agents and two dirt producers, where each agent performs optimal move based on the state of the environment. In the three classes, a similar approach to the minimax algorithm is implemented, where the vacuum cleaner agent is the maximizer and the dirt producer is the minimizer, noting that the agents run simultaneously.

Contents

INTRODUCTION

In recent years, intelligent agents have taken a major role and attention due to their effectiveness in assisting and helping humans.

This paper presents the design of Intelligent Agents, namely Vacuum Cleaner and Dirt Producer Agents, which respective objects are:

- To clean tile-based environment anonymously
- To dump dirt in a tile-based environment anonymously

Both consider maximizing efficiency in terms of steps and cost of traversal. The dirt producer agent is introduced to create a competitive environment for the vacuum cleaner agent. The dirt producer could be seen as dynamic dirt environment change in real life, where besides dust build up, it could be dirty footsteps, splashed food on the ground, broken glass, which might happen at any time.

BACKGROUND

Previous work has been done regarding cleaning agents using different approaches and solutions, one of them being the Coverage Path Planning (CPP) approach [1].
The goal of CPP is to cover the entire environment, known or unknown, by creating a specific path. A basic CPP procedure can be described as follows:
Agent attempts to discover and identify the environment by following the outermost wall.

Once it returned to its initial position, a map representing the discovered environment is created.

The first part of the project paved the way for extensional part which is represented in this report. The use of the search state algorithms helped in building the choice of moves the agents will have. The complexity of computations that increases with the increase of the number of tiles is backed up using memory to save on processing cost. Similar projects on multiagent environments had the upper hand in the progress idea of the project. [1]

PROPOSAL

1.1 Overview

The grid shown in figure 1 is what the user sees implemented using JavaFX, with the backend side, algorithms implemented in Java.

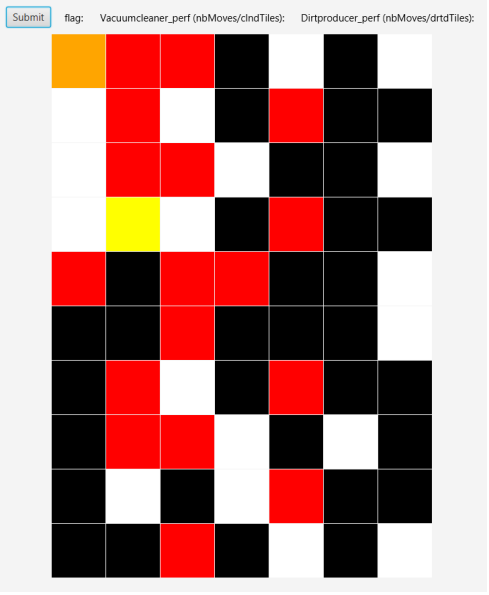


Figure 1: Two-agent environment

The guide through this grid is as follows:

- Vacuum Cleaner agent represented in yellow
- Dirt Producer agent represented in orange
- Blocks are represented in red
- Dirty tiles are represented in black
- Clean tiles are represented in white

1.2 Implementation

The report will distinguish three classes of agents depending on the visibility of each agent:

- **Environment with two agents**
- **Environment with agents that use memory**
- **Environment with four agents**

In the next sections, these three classes will be further explained:

1.2.1Two-agent Environment

These approaches were implemented for an agent with the following Performance, Environment, Actuators, and Sensors (PEAS):

- **Performance Measure**: Minimize energy consumption, maximize dirt pickup or dirt dump: -0.1 per step and +5 per dirt sucked, or dirt dumped.
- **Environment**: Tile-based room with walls as obstacles. Dirt distribution can be determined by the user or loaded from a saved distribution map.
- **Actuators**: Left, Right, Up and Down, NoOp, such (for vacuum cleaner), dump (for dirt producer).
- **Sensors**: The sensors of the agent capture the complete state of the environment (room) and know which tiles are dirty and which are clean. It also knows the location of walls, as well as its own location.

The agents use an algorithm to make optimal moves that follows a similar approach to TSP algorithm, where it computes the distance between the agents' position and each of the targeted tile type, and accordingly moves to the closest tile.

A similar approach to the minimax algorithm was implemented, that includes having a maximizer and a minimizer. The maximizer is the vacuum cleaner which tries to maximize the value called 'flag', with 'dirt producer', the minimizer trying to minimize it. Each no action-move (not reaching a dirty tile) of the vacuum cleaner will be an advantage to the dirt producer agent and vice versa. The move will cost 0.1, where it's +0.1 in case of dirt agent no action move (not reaching a clean tile) and -0.1 in case of vacuum cleaner not reaching a dirty tile. However, a vacuum cleaner agent cleaning a dirty tile will add to the value trying to maximize +5, while for the dirt producer will minimize it by 5.

Each agent has a performance measure, that equals the average number of moves needed to clean a tile or dirty which is dependent on which agent.
The two agents will be running in a timed environment specified by the user, where at the pass of this time their work is interrupted, and the winner is decided looking into the final flag value. If flag > 0, the vacuum agent won else it is the dirt producer. Note that the initial distribution of the environment has its say in deciding the outcome.

Another algorithm that was to be implemented yet we faced an issue while integrating was the Alpha Beta Pruning approach. A brief description of this algorithm is the following: [3]

It is usually referred to a modified version of the Mini Max algorithm yet following a certain optimization technique which would eventually produce better results. In the minimax algorithm the number of states examined by such an algorithm is relatively exponential. The elimination of this exponential component is relatively hard yet following the technique of pruning (this technique can compute the correct decision performed by minimax algorithm without checking the tree nodes) this exponential component could be cut into half. Alpha Beta pruning could be applied at any depth as making the pruning approach to both the main node and the sub node.

Alpha refers to the best choice in terms of highest-value computed by the algorithm along the path of the maximizer, while Beta refers to the best choice in terms of lowest-value computed by the algorithm along the path of the minimizer. (Condition to always keep in mind Alpha > = Beta)

The maximizer in our case which is the vacuum cleaner will update the value of alpha, while the minimizer which is the dirt producer will update the value of Beta.

1.2.2Two-agent environment with use of memory

This class adds the notion of memory to the first one, where the agents resort to their memory to check the tiles that were visited in previous runs. The first run will be as in the first class, where the closest action tile to the agent is visited. However, with the termination of the first run, a recorded list of tiles is saved in memory, which will be retrieved at each run of the agents. The agent will choose to move in the tile order of the memory if their action tiles (remember an action tile is a dirty tile for the vacuum cleaner and clean tile for the dirt producer). If all the tiles stored in memory were checked and moved according to, then if the time is not exhausted, the initial used algorithm will kick in to decide the path of the agent. The memory only holds the tile order, and not the whole path, so the agent will use BFS to find the shortest path to this tile, still more efficient that running distance computation between current agent's position and possible action tiles.

1.2.3 Four-agent environment

This class is built on the top of the two-agent environment class, where the environment is made to hold more than two agents, in this case four agents. The environment will have two teams, the vacuum cleaner agents' team and the dirt producer agents' team. Both teams will work under the umbrella of the minimax algorithm but will be moving simultaneously. What is challenging in this implementation, is the additional constraints to avoid collisions between agents. Each agent initially looks for a path taking into consideration the current position of the other agents. Once the path has been decided upon to, it can't go blinded in traversing the path, because at the time of the traversal some agents might be in the path, so this dynamic change of the environment is taken into consideration, with always sensing the status of the next tile in the path. The status of the target pile is changed before the agent reaches it, to avoid the same type of agent from visiting it, and at the same time is not made available for the other agents until the intended agent reaches it.

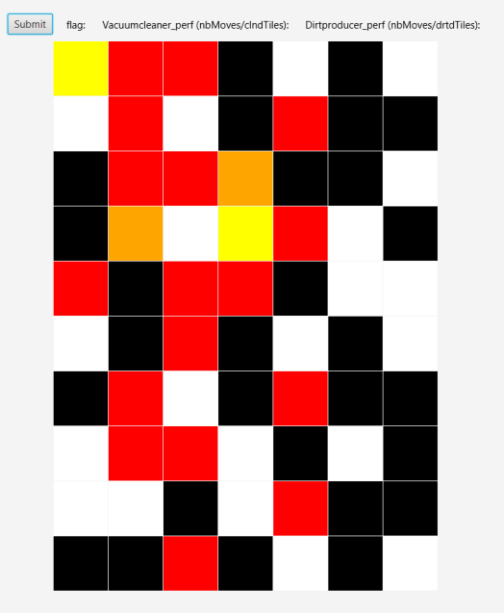
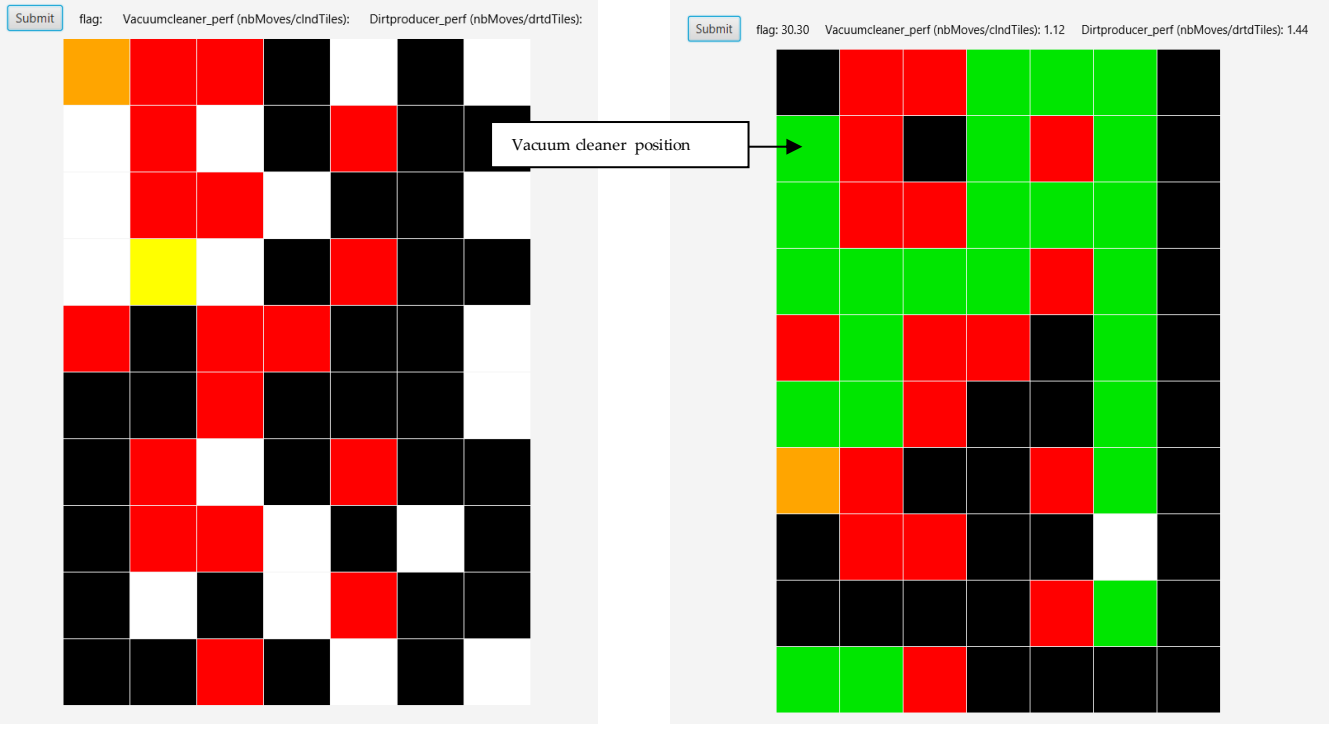
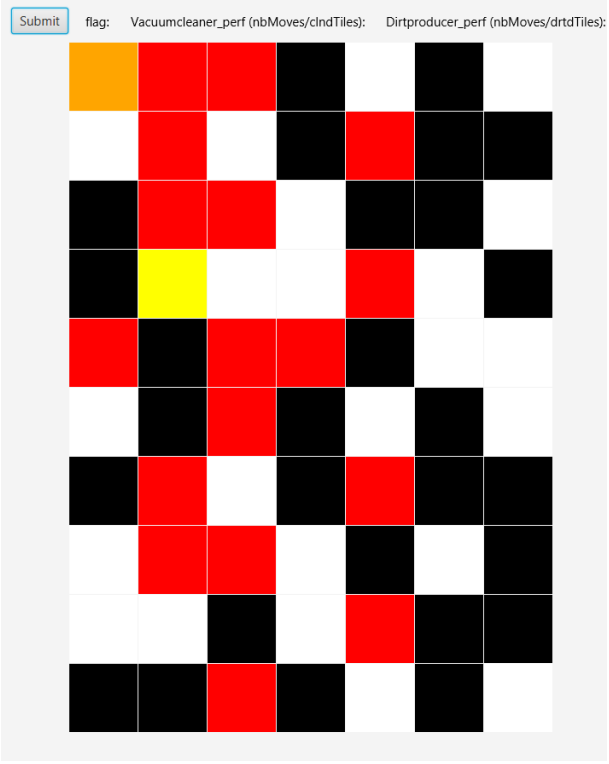


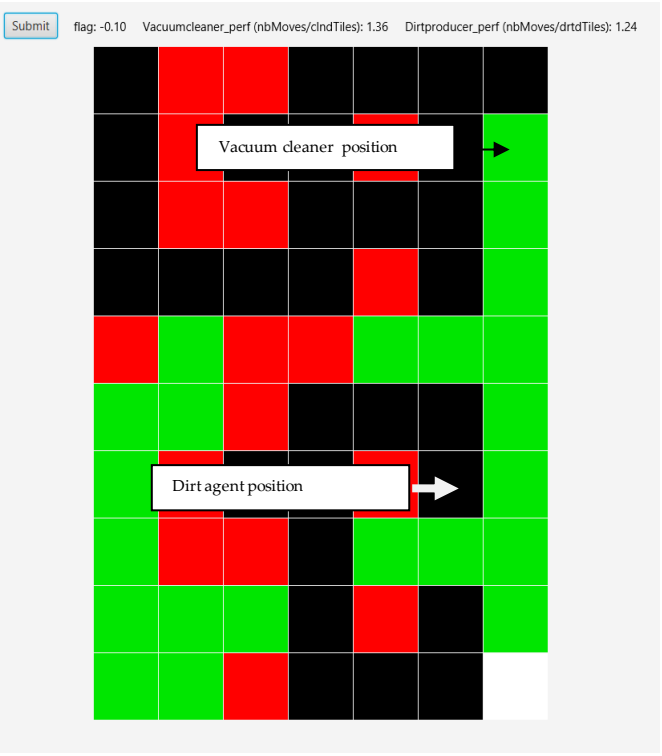
Figure 2: Four agent environment

The agents of the same group are in the same color, this is made to make the user more comfortable looking at the changes on the grid instead of having each agent with a color.

EXPERIMENTAL EVALUATION







The difference between the two runs is the initial configuration of the grid. With the first-row example, the initial grid made the vacuum cleaner the favor to win, since it has more dense surrounding of dirty tiles. On the contrary, a different distribution of dirt in the example in the second row, made the dirt agent the winner.
We can notice in the first simulation, that the vacuum cleaner agent had better performance with an average 1.12 moves to clean a tile over the dirt producer agent which had lower average of 1.44 moves to dirt a clean tile. This contributed to the final flag value to be in the favor of the vacuum cleaner agent, and vice versa in the second run.

Yet this is might not be a rule, that the better performance agent is the one to win with the given time, since at the time of exhaustion one of the agents might have a sequence of far tiles that made it lag behind the progress of the other agent, yet its performance measure was not affected because of previous better moves. So, the initial distribution along with the time allocated to the agents dictate who is the winner considering that the two agents are running optimally.
Note that, the agents might not appear in the simulation, since just before the rectangle of the agent position is to be shown, the time slice is exhausted. Still, you can now through the code from the status of the grid array where each agent is positioned at the end of the simulation time.

CONCLUSION

This project focused on implementing a multi-agent environment with dynamicity regarding the distribution of clean and dirty tiles. While trying different implementations, I found it hard to link between available algorithms and real implementation, specially if there are certain constraints such as simultaneousity. This project covers the appealing way to tackle the idea of multiple agents, yet it needs a lot of debugging and testing to make it work all together.

REFERENCES

[1] https://blockgeni.com/mini-max-algorithm-in-ai/
[2] https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/
[3] https://www.javatpoint.com/ai-alpha-beta-pruning