# CS531 Programming Assignment 1: Vacuum-Cleaning Agents

Michael Lam, Xu Hu
EECS, Oregon State University

October 8, 2012

**Abstract**

In this assignment we design and implement 3 different vacuum-cleaning agents and ran experiments to compare and discuss their performance.

## 1 Introduction

Consider an environment where the world is a rectangular room consisting of $n \times m$ cells, with each cell having a probability of dirt. There are three vacuum cleaner agents in consideration: 1) memoryless deterministic reflex agent, 2) randomized reflex agent, and 3) deterministic model-based reflex agent with 3 bits of memory. The agent has three percepts (wall, dirt, home) and five actions (forward, right, left, suck, turn off). The performance measure is the total number of clean cells as a function of how many action steps were taken. We also would like the agent to return to its home cell and shut off after it cleans the entire room.

Having described the environment, agent and performance measure, the general goal of a successful vacuum cleaner agent is to clean the entire room in a minimal number of steps. While that is the ideal case, two of our agents are not able to satisfy this goal. Thus we have different goals for our agents due to their designs by definition. The first agent in consideration, the memoryless deterministic reflex agent, is limited by pure reflex actions with no memory. We discovered that we cannot clean every cell with this agent, so our goal for this agent is to clean as many cells as possible and to return home at the end. For the randomized reflex agent, its performance is improved and simultaneously hindered by its probabilistic actions. We discovered that while it can clean the entire room over time, it is difficult to maneuver the agent back to the home cell due to randomness. The goal for this agent then is to clean every cell without needing to return home. Finally, the deterministic model-based reflex agent has memory, which gives it advantage to remember which actions it has taken. Given this advantage, the goal of this agent is to clean the entire room and navigate home successfully. We determined that this was possible in less than 8 states, the maximum cap for the agent's memory.

# 2 Memoryless Deterministic Reflex Agent

For this task, we are trying to design the simplest agent with a limited number of if-then rules. Since there are 3 sensors for wall, dirty and home grid, the agent may face $2^3$ different cases for every percept. The idea behind the memoryless agent is based on the (situation, action) pairs. For simplicity, we assume that the agent will only take one step for each situation, and then forget what he has done. The designed rules are inserted into the main loop to control his actions. So the task here is to learn a mapping $f(wall, dirty, home)$ for the action space.

## 2.1 Is It Able To Finish?

Before coding these rules, we thought about a question: is it possible to finish the cleaning task only with the one-to-one mapping? To answer this question, we make a simple experiment on each type of cell to see if an agent can travel through it. We show a $4 \times 4$ map in figure 1, in which we can see that there are in fact three types of cells. The diamond cells (corner cells) or circle cells (border cells) can be easy to access: just let the agent go ahead when there is no wall and turn left or right when facing a wall. But how about the triangle cells? Can it step into such a cell? If so, we can make sure that the memoryless agent can finish the task properly. Consider the triangle cell. There are only four ways to step into the central cell (we show arrows in figure 1). It is not possible to pass other triangle cells to get there before we find a way to the triangle cell from the outside. The outside cells

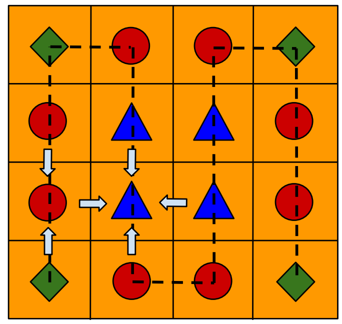


Figure 1: We use different shapes to represent different cell types. We can see that it is not possible to step into a triangle cell in the memoryless deterministic reflex agent case, because an agent cannot take 2 different actions in circle cells, but which is necessary to get into the center.

have the same type, i.e. circle cells. If we want the agent to get to the circle cell, a forward action has to be taken. And for getting into the triangle cell, a turning action should be taken. The problem is that once it turns right or left, it will do the turning again and again (also turning when facing the wall) because 1) the agent is perceiving the same percept again and again, and 2) the agent has no memory to distinguish steps before and after making a turn. In many situations it is typical to have 2 potential consecutive actions, but for this case it is only possible to assign it a single reflex rule per percept. Thus, the best strategy is to clean up dirty cells on the border as soon as possible.

## 2.2 Rules

We design the if-then rules as follows:

```
if DIRT then SUCK
if not DIRT and not WALL and HOME then FORWARD
if not DIRT and WALL and not HOME then RIGHT
if not DIRT and WALL and HOME then TURN OFF
```

```
if not DIRT and not WALL and not HOME then FORWARD
```

Therefore, the agent will go around the map along the boundary and return to home at the end.

# 3 Randomized Reflex Agent

In the randomized reflex agent, it is possible to take several different actions in terms of the action probability distribution. We use the multinomial distribution in this case. For designing the parameters in these multinomial distributions, we use heuristics in light of an optimal path, which is shown as the dashed line in figure 1. For a $n \times m$ map, there are $(n-1) \times m$ forward actions, and $\lfloor (m-2)/2 \rfloor$ left and right turns when no wall and dirt have been detected. When facing walls, the ratio of turning left and turning right is $2:1$. Based on these observations, we design the if-then rules and multinomial distributions as in table 1. The agent will choose an action according to the random number generated by the multinomial distribution.

| WALL | DIRT | HOME | FORWARD | RIGHT | LEFT | SUCK | OFF |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.0 | 0.65 | 0.33 | 0.0 | 0.02 |
| 1 | 1 | 0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 1 | 1 | 1 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 1 | 0 | 0 | 0.0 | 0.67 | 0.33 | 0.0 | 0.0 |
| 0 | 0 | 1 | 0.9 | 0.05 | 0.05 | 0.0 | 0.0 |
| 0 | 1 | 0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 0 | 1 | 1 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 0 | 0 | 0 | 0.8 | 0.1 | 0.1 | 0.0 | 0.0 |

Table 1: Parameters of multinomial distributions for each situation.

We prefer the agent to stop at home once it has cleaned as many cells as possible. Note that in practice, we didn't let the agent turn off before reaching the maxmum number of permitted actions[1]. Our design favors cleaning as many dirty cells as possible over getting home correctly since the actions are randomized and it's possible that the agent can never get home if

[1]We set a maximum cap on the number of actions an agent can take so that the randomized agent would not keep running forever potentially

it is unlucky. Thus for a dirty cell, we let the agent clean this cell first given our greedy strategy. When there is no wall and dirt detected, we designed the agent to be inclined to keep going forward, but the ratio is not as high as the case of the optimal path. We found that decreasing the probability of going forward a little bit will achieve better performance.

# 4 Deterministic Model-Based Agent

Since the deterministic model-based reflex agent has some memory to store state information, we exploit this property to design a better agent that can 1) suck up dirt at every square (which fails for the memoryless reflex agent) and 2) get home properly (which fails for the randomized reflex agent).

## 4.1 Design Considerations

The requirements of the deterministic model-based reflex agent other than its definition is that it can represent state with only 3 bits of memory, which implies up to 8 states[2]. We want the agent to suck every dirt as our first goal. Since the agent has memory, we are able to perform two "consecutive" actions by taking advantage of state information. This will allow the agent to steer into the middle of the world in contrast to the simple memoryless reflex agent, which could only suck dirt near the boundaries of the world.

In addition, our second goal is to incorporate a method for the agent to return home after it has cleaned every cell. We can use state information to represent the state "done cleaning." In fact, we will show that in our design, only one state in addition to the percept is sufficient to steer home.

Therefore, we designed an algorithm to clean every cell and return home systematically since our agent is deterministic and has memory. We designed the algorithm so that our agent performs the sweeping pattern across the room as in figure 2.

## 4.2 Algorithm and Rules

Our agent performs the following algorithm:

[2]Without loss of generality, we can use one state variable STATE that takes on integers $[0,7]$ instead of three explicit state bits STATE1, STATE2 and STATE3 because $0 = 000, 1 = 001, 2 = 010, ..., 7 = 111$.

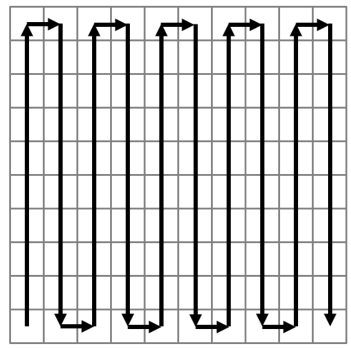


Figure 2: The deterministic model-based reflex agent starts at the bottom-left corner and sweeps north, east, south, east, etc. as shown in the figure.

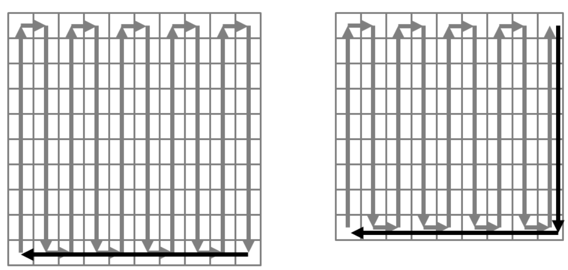


Figure 3: Once the agent is done cleaning, it moves to state 6, which is just a series of reflex actions similar to the memoryless agent. In the case when the agent is done cleaning at the top-right corner (left figure), it moves south and then west back to the home cell along the world boundaries. In the case when the agent is done cleaning at the bottom-right corner (right figure), it moves west back to the home cell along the world boundary.

1. Begin at the home cell.
2. Go north and suck until it hits the south boundary.

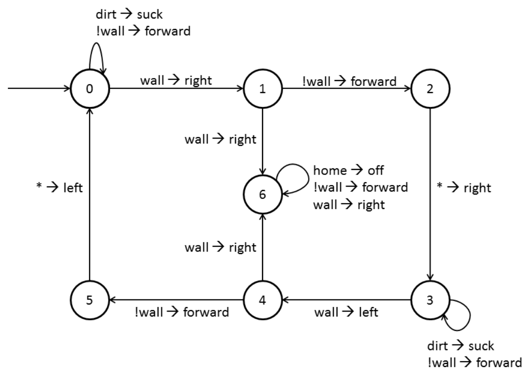



Figure 4: State machine of the model-based reflex agent.

3. Go one cell east.

4. Go south and suck until it hits the south boundary.

5. Go one cell east.

6. Repeat steps 2-5 until there is a wall that prevents going one cell east.

7. Traverse the world boundary until it comes home.

We can construct the algorithm by using 7 states. We illustrate the state machine in figure 4.

In plain language and referencing figure 4, state 0 represents a series of reflex actions for sucking and moving north. When the agent hits the top boundary of the world, states 1-2 assist the agent in moving east one square to prepare sucking and moving south. State 3 (symmetric to state 0) represents a series of reflex actions for sucking and moving south. When the agent hits the bottom boundary of the world, states 4-5 (symmetric to states 1-2) assist the agent in moving east one square to prepare sucking and moving north. The cycle repeats until the agent can no longer move east, as detected in states 1 and 4. If that is the case, then the agent moves to state 6, which is a series of reflex actions to navigate the agent around the edges of the map until it reaches home.

An alternative representation is the following formal list of if-then rules, which is equivalent to the state diagram. It is also coded into the agent as the program:

```
if STATE=0 and DIRT then SUCK
if STATE=0 and NOT WALL then FORWARD
if STATE=0 and WALL then STATE := 1, RIGHT
if STATE=1 and WALL then STATE := 6, RIGHT
if STATE=1 and NOT WALL then STATE := 2, FORWARD
if STATE=2 then STATE := 3, RIGHT
if STATE=3 and DIRT then SUCK
if STATE=3 and NOT WALL then FORWARD
if STATE=3 and WALL then STATE := 4, LEFT
if STATE=4 and WALL then STATE := 6, RIGHT
if STATE=4 and NOT WALL then STATE := 5, FORWARD
if STATE=5 then STATE := 0, LEFT
if STATE=6 and HOME then OFF
if STATE=6 and NOT WALL then FORWARD
if STATE=6 and WALL then RIGHT
```

Therefore, we have also verified that the state diagram is equivalent to the algorithm. The algorithm is correct by construction of the state machine.

# 5 Experiments

First, we define a performance measurement in terms of the ideal performance as follows:

$$P = \sum_i \frac{Number(action_i)}{i}$$

where we assume that the ideal number of actions should be the same as the number of steps. We will use this measurement for scoring each agent.

## 5.1 Memoryless Reflex Agent

In this case, the agent cannot finish the cleaning, which cleaned up 36 cells in 76 steps. The performance curve is shown in figure 5. The memoryless agent got a score of 0.503.

## 5.2 Randomized Reflex Agent

We ran 50 trials for the randomized reflex agent. The number of actions to take to clean 90% of the room is shown in table 2. We also show the

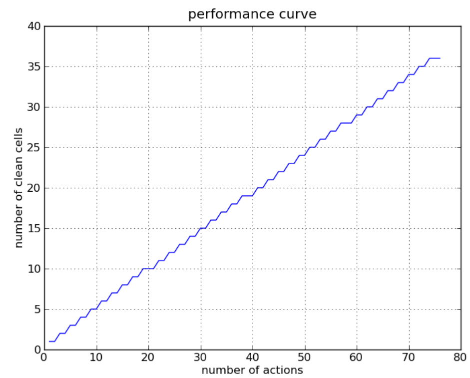



Figure 5: Performance of Memoryless Deterministic Agent

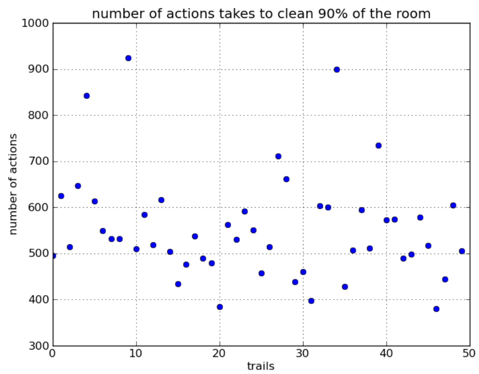
performance curves for each trial in figure 6. The average actions of the best 45 trials is 524.96. The score of this agent is 0.118. We didn't see the extreme case in our experiment, which may always able to choose the right action for making an optimal path. The randomized agent wastes a lot of steps to finish the cleaning but at least it has no limitations, so it is very possible to clean up the room before turning off. This is much better than the memoryless one.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| trials 1-10 | 495 | 625 | 514 | 647 | 842 | 614 | 550 | 532 | 532 | 924 |
| trials 11-20 | 510 | 585 | 519 | 616 | 504 | 434 | 477 | 538 | 489 | 479 |
| trials 21-30 | 385 | 562 | 531 | 591 | 551 | 458 | 515 | 711 | 662 | 439 |
| trials 31-40 | 461 | 398 | 604 | 601 | 900 | 428 | 507 | 594 | 512 | 735 |
| trials 41-50 | 572 | 574 | 490 | 498 | 578 | 517 | 380 | 444 | 605 | 506 |

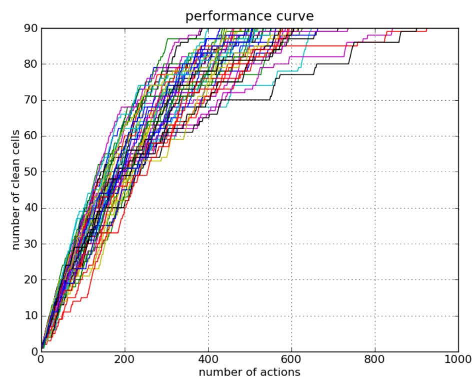
Table 2: Number of actions to take to clean 90% of the room

### 5.3 Deterministic Model-based Agent

For any $n > 0$, $m > 0$ and $0 \leq p \leq 1$, the agent is able to completely clean the room and shut off itself properly. The algorithm assumes the home cell is always the bottom-left corner of the world and the world is always a rectangular nxm array of cells with walls only at the boundarie of the rectangle. If the home cell is somewhere else, then it may miss cells to the west and south of its starting location. If the world is not rectangular, then it




(a) Number of actions




(b) Performance curves

Figure 6: Performance of randomized reflex agent

may make a premature assumption that the world is clean or it may navigate incorrectly such that it misses some cells.

For $n = 10$, $m = 10$ and $p = 1.0$ as in the default case, the agent took 230 steps. The model-based agent got a score of 0.472. We show the performance curve in figure 7.

# 6 Discussion

We have implemented three reflex agents and have shown that simple tasks can be done with carefully designed rules. However, it is not a good idea for

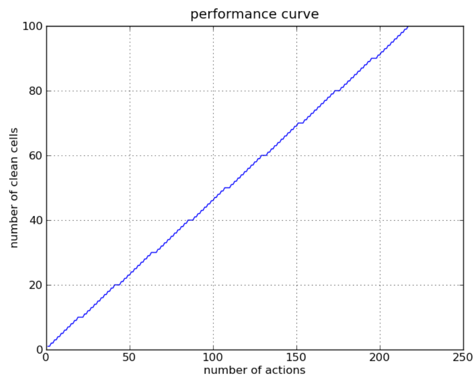



Figure 7: Performance of Model-Based Agent

creating an intelligent machine which has to handle many complex tasks. A better solution may let it learn how to do a certain type of tasks by itself. Actually, we can learn model parameters or mapping rules for our agents. We will try to use Q-learning for this improvement. But such learning agent is beyond the scope of this project.