AI Challenge 1: Cleaning Agents

COSC4550/COSC5550
Artificial Intelligence
University of Wyoming

1 Overview

In this challenge you will have to implement the AI of a Simple Reflex Agent. Graduate students will also have to implement a Model-Based Reflex Agent. The goal of these agents will be to clean as much dust as possible within a limited time.

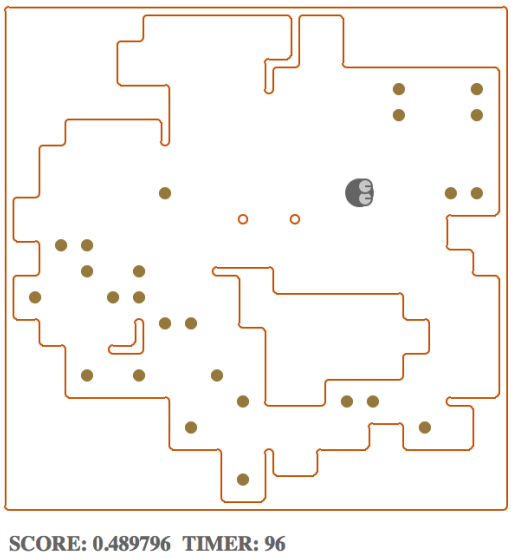

Figure 1: The screen of the cleaning agent simulation

Acknowledgment: This assignment is based on the one created by Dan Klein and John DeNero given as part of Berkeley's CS188 course. This assignment was also inspired by the modifications made by Peter Stone in his CS343 course of 2012. We thank Dan and John for creating the assignment and granting the permission to use it and we thank Peter for the ideas on how to adapt the assignment for this course.

1.1 Chapters

Chapters are from the book 'Artificial Intelligence, A Modern Approach', third edition, by Stuart Russel and Peter Norvig. The relevant chapter for this challenge is chapter 2, especially section 2.4. Some of the search-algorithms from chapter 3.4 may also be helpful (although the next assignment will handle those in-depth).

1.2 Program files

The zip file provided with this challenge contains eight python files. These files have been tested on, and should work with, Python version 2.7.3 and Python version 2.6.6. More recent versions of Python 2.x probably work as well, but these files do not work with Python 3.x.

Most of those files are just to simulate the environment or display things on the screen, but the following files need your attention:

your_agents.py	This is the file that you have to extend with the logic of your agents. This is also the file you have to hand in for this assignment.
clean.py	This is the main file of the program. It is used to test your agents.
game.py	This file contains most of the important classes that you will use to communicate with the environment.

First, to test whether your setup is working, open a terminal, navigate to the assignment directory and type:

```
python clean.py
```

This should open a screen similar to the one shown in figure 1, showing a randomly generated environment with the cleaner robot (grey) and patches of dust or dirt (brownish yellow). At the bottom of the screen you will find a score counter and a timer. The score counter indicates what percent of the dirt has been cleaned while the timer indicates how much time is left for this run. When the timer reaches zero the simulation will end and the final score is written back to the terminal. If the screen does not immediately pop-up, it is possible that the window is hidden behind other screens. Use alt-tab or a similar method to retrieve it. Now, if you open *your_agents.py* you will find the following classes:

```
class SimpleReflexAgent(Agent):
  "A reflex agent you have to implement"

  def getAction(self, state):
```

```
    "The agent receives an AgentState (defined in game.py)."
    # YOUR CODE HERE #
    util.raiseNotDefined()
    return action

class ModelBasedReflexAgent(Agent):
  "A reflex agent you have to implement"
  def __init__(self):
    # YOUR CODE HERE #
    util.raiseNotDefined()

  def getAction(self, state):
    "The agent receives an AgentState (defined in game.py)."
    # YOUR CODE HERE #
    util.raiseNotDefined()
    return action
```

If, after writing the code for it, you wish to test one of your agents, you will have to provide the *-p* option with the name of the class as an argument. For example, if you want to test your SimpleReflexAgent, type:

```
python clean.py -p SimpleReflexAgent
```

This should run the program using your implementation as the AI for the cleaner robot. If you feel your robot is moving too slow, try the option *--frameTime=0* to increase its speed. This will only work if the slowdown is caused by the rendering, it will not help if your agent is just slow in making decisions or if you are running in quiet mode (*-q*).

Now, because the environments are randomly generated, your agent has to perform reasonable in a variety of different environments. To test your agent's performance on 100 different environments type:

```
python clean.py -p SimpleReflexAgent -n 100 -q
```

Here *-n 100* indicates the number of runs you wish to perform while *-q* indicates that the program should not use the graphics display for these runs.

1.3 The cleaning robot

The cleaning robot has four types of sensors: a GPS, a compass, a wall detector and a dust detector. The data of all these sensors is contained in the *state* argument of the *getAction(self, state)* function. Below is a table on how to access the data of these sensors.

Sensor	Access	Description
GPS	getPosition()	Returns the current position of the robot as the pair *(x, y)*.
Compass	getDirection()	Returns the direction of the robot as a string from Directions (see game.py).
Wall sensor	rightWallSensor, leftWallSensor, frontWallSensor, backWallSensor	Returns the distance (in squares) to the first wall detected in the indicated direction.
Dust sensor	rightDustSensor, leftDustSensor, frontDustSensor	Returns the dust concentration in the indicated direction.

Here the dust sensor returns the 'concentration' of two tiles in its direction. When both tiles contain dust the sensor returns 1.5. When only the closer tile contains dust it returns 1.0. When only the farther tile contains dust the sensor returns 0.5. When neither tile contains dust the sensor returns 0.0.

The cleaner has 6 actions it can perform. All actions are part of the *RobotActions* class, meaning you will have to access them via this class. For example, to get the *TURN_LEFT* action, type: *RobotActions.TURN_LEFT*

Action	Description
TURN_LEFT	Turns the robot left by 90 degrees.
TURN_RIGHT	Turns the robot right by 90 degrees.
TURN_AND_MOVE_LEFT	Turns the robot left by 90 degrees and moves forward one square.
TURN_AND_MOVE_RIGHT	Turns the robot right by 90 degrees and moves forward one square.
FORWARD	Moves the robot forward one square.
STOP	The robot does not move or turn.

1.4 The environment

The environment is randomly generated every run. However, there are a few constants that you may use to your advantage.

- The maximum size of the environment is: 20 by 20. Because the environment is stored in a matrix starting with index $(0, 0)$, the minimum coordinate your robot can be in is $(0, 0)$ and the maximum coordinate your robot can be in is $(19, 19)$.
- The origin of the environment, which has the coordinate $(0, 0)$, is the bottom left of the environment and is not always reachable.

- The starting position of your robot is always $(10, 10)$.
- The maze will always have at least 10 patches of dust in it.
- The probability of a floor tile containing dust is 20%.

More information about the environments (such as the algorithm that produces the environments) can be found in *layout.py*.

1.5 Deliverables

For this assignment you have to hand in a completed version of the *your_ agents.py* file. Please, rename the file to *[yourName]_ agents.py* before handing it in.

Important: Make absolutely sure that your implementation will run all questions without any modifications being necessary on our part. It should run on either python 2.7.3 or python 2.6.6 and when in doubt you can always test your implementation on hive. You will only receive partial credit for implementations that do not run.

To run your solution on hive, first copy your project to hive (hive.cs.uwyo.edu) through any protocol accepted by hive (such as scp), then ssh onto hive and run your solution. Do not forget to use the `-q` option, as you will not have a display on hive and the program will crash if you try to run it without the `-q` option. The `-q` option is not necessary for the autograder as the autograder does not require any visualizations.

2 Questions

2.1 Question 1 (4 points): Simple Reflex Agent

Implement a simple reflex agent for our robot. Remember that the simple reflex agent is not allowed to store any data, it may only use the current data from its sensors.

Execute a single run:

```
python clean.py -p SimpleReflexAgent
```

Execute a batch of 500 runs as used for grading:

```
python clean.py -p SimpleReflexAgent -n 500 -q
```

2.1.1 Hints and Observations

- The simple reflex agent often gets stuck in endless loops, consider using randomized actions to escape these situations.

2.1.2 Grading: 4 points

If you implemented a proper reflex agent you will get points depending on your average score over 500 runs according to the table below.

Average score	**Points COSC 4550**	**Points COSC 5550**
$score < 0.80$	2	2
$0.80 \leq score < 0.84$	3	3
$0.84 \leq score < 0.90$	4	4
$0.90 \leq score$ (current record)	+1 extra credit	+1 extra credit

2.2 Question 2 (4 points): Model-Based Agent

(Mandatory for those enrolled in the graduate student version of the class. Bonus points for undergraduates)

Implement a model-based reflex agent for our robot. In contrast to the simple reflex agent, the model-based agent should maintain a state that contains the current knowledge the agent has about its surroundings and update this state according to new sensor information and its world-model. Luckily our world-model can be very simple, meaning the main challenge is to create an accurate knowledge state and to effectively use this knowledge state to make your decisions.

Execute a single run:

```
python clean.py -p ModelBasedReflexAgent
```

Execute a batch of 500 runs as used for grading:

```
python clean.py -p ModelBasedReflexAgent -n 500 -q
```

2.2.1 Hints and Observations

- Create a good and usable knowledge state that accurately reflects the agents current knowledge about the world. This state should probably contain information about detected dust patches, walls, and unexplored territory.
- Once you have a good representation of the agent's knowledge, a simple search algorithm might help in finding the best location to go to next.
- The correct high-level behavior often includes a series of lower level actions. Consider keeping a queue or similar data-structure to store actions that need to be executed in the future.

2.2.2 Grading: 4 points

If you implemented a proper model-based agent you will get points depending on your average score over 500 runs according to the table below.

Average score	**Points COSC 4550**	**Points COSC 5550**
$score < 0.85$		1
$0.85 \leq score < 0.92$		2
$0.92 \leq score < 0.95$	+0.5 extra credit	3
$0.95 \leq score < 0.98$	+1 extra credit	4
$0.98 \leq score$ (current record)	+1.5 extra credit	+1 extra credit

3 FAQ

Q: *How do I get Python?*

A: You can get Python from: https://www.python.org/downloads. Remember to download version 2, since version 3 will not work for this assignment.

Q: *How do I compile Python files?*

A: You do not compile Python files, Python is an interpreted language. Once you have installed Python you should be able to run the main file for this assignment as follows: `python clean.py`

Q: *How do I edit Python files?*

A: You can edit Python files in any text-editor, although a text-editor that supports syntax highlighting is recommended. If you prefer a simple text-editor something like notepad++ is good choice (`http://notepad-plus-plus.org`). If you prefer a more advanced IDE you might look into the PyDev extension of Eclipse (`http://pydev.org`), or the community edition of PyCharm (`https://www.jetbrains.com/pycharm/download`).

Q: *Are we allowed to import a data structure?*

A: Yes, you are allowed to import a data structure such as a Stack, Queue, ect. Most other imports are off-limits though.

Q: *What is the `def __init__(self)` method?*

A: The `def __init__(self)` method is the constructor of a class. This is were you should initialize your world model. For example, if you wanted to store the last action your robot performed, you could add `self.last_action = None` to this function. Now, anywhere in the `def getAction(self, state)` function, you can access this variable with the self keyword: `self.last_action`. For more information about Python constructors you can look at one of the many tutorials available, such as this one:
`http://www.tutorialspoint.com/python/python_classes_objects.htm`.

Q: *What constitutes a world model?*

A: A 'world model' is anything that stores information about the world that can not be inferred from the current sensor data. You might, for example, just store the last action that your robot performed, and this would count as a 'world model', be it a rather minimalistic one. To get a good score on this challenge you probably need to store the positions of dust, walls, as well as to mark areas that you have already explored.

Q: *How do I make an agent perform more than one action?*

A: The `def getAction(self, state)` function can only return one action at a time. When you want the robot to perform multiple actions in a row, you can create a list of actions in the constructor (`self.actions = []`), and then, in the `def getAction(self, state)` function, immediately remove and return the first action in that list if the list is not empty (see: `http://www.tutorialspoint.com/python/python_lists.htm` for more information about lists).

Q: *Will we be graded on the from and readability of our code?*

A: No, your code will be checked for plagiarism and cheating (such as directly modifying the variables that affect your score), but other than that it is only the performance of your robot that matters. That said, you are encouraged to write readable code simply because that will make it a lot easier for you to debug your code.