CS5011: A2 - Logic - Obscured Sweeper

Assignment: A2 - Assignment 2

Deadline: 9 March 2022

Weighting: 25% of module mark

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical aims to implement an agent that is able to play and solve a logic game.

2 Competencies

- Design and implement a logical agent that is able to perceive certain facts about the world it occupies, and act accordingly.
- Use of logical reasoning to solve a puzzle game.

3 Introduction

3.1 Background

The Obscured Sweeper game is inspired by the well-known Mineswepeer computer game[1]. Similar to the classic Minesweeper, an Obscured Sweeper world is a grid of $N \times N$ cells, and M mines are scattered among the cells. However, some of the cells are "blocked". A blocked cell is a cell not containing mines which does not give any clue to the number of neighbouring mines, and therefore cannot be probed.

The rules of the Obscured Sweeper game are similar to those of Minesweeper, but the Obscured Sweeper worlds are a small subset of those. A logical agent playing the Obscured Sweeper game aims to uncover all cells on the board but those containing a mine. If the agent probes a cell that contains a mine, the game is over. Otherwise, a clue appears on the cell indicating the number of mines in the surrounding neighbours of the probed cell. In this practical, your aim is to investigate logic-based strategies for an agent to solve the Obscured Sweeper game.

[1] `http://www.minesweeper.info/wiki/Main_Page`

3.2 The board

Each Obscured Sweeper world is represented as a Java array of chars, where each char represents cell with the following code:

- `'m'` means that the cell contains a mine
- `'0'–'8'` is the number of mines in the 8 neighbouring cells
- `'b'` means that the cell is blocked, i.e. it does not contain a mine and it cannot be probed

Each cell is represented with its coordinates $[x, y]$.
The Obscured Sweeper game can be played at 3 sizes, each containing a varying number of mines M:

- Small worlds: in which $N = 5$ indicating a 5 by 5 board.
- Medium worlds: in which $N = 7$ indicating a 7 by 7 board.
- Large worlds: in which $N = 9$ indicating a 9 by 9 board.

In Figure 1 an example of a 3 by 3 Obscured Sweeper world with 3 mines is shown.

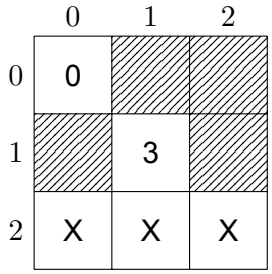

Figure 1: Example Obscured Sweeper World, `TEST1`. The shaded areas are blocked cells while the cells marked with 'X' are cells containing mines.

3.3 The rules

The rules of the game are as follows:

1. The total number of mines, M, is known by the agent but should not be used to determine whether the game is over.
2. At all times, the agent is aware of the size of the board.
3. The agent is also aware of the locations of the blocked cells.
4. At the initial step, the non-blocked cells are all covered and the agent has no other information.
5. Two starting clues are given to the agent: the agent can safely probe the cell at the top left-hand corner, and the cell in the centre of the board at the start of the game (e.g. $[0, 0]$ and $[1, 1]$ in Figure 1).
6. At each step, the agent probes a covered non-blocked cell, and receives information about the content of the cell. The cell will then be marked as probed/uncovered.
7. If the probed cell is a mine, the game is over and the agent loses the game.

8. If the cell contains a value 0 meaning that no adjacent cells contain mines, all the non-blocked neighbouring cells will also be uncovered.

9. If the content of the cell has a value greater than 0, this is a clue about the number of mines existing in the non-blocked neighbouring cells. The agent should make inferences about its view of the world and decide which cell should be probed next.

10. If all but M cells are probed without a game over, the agent wins the game.

11. The agent may use flags to signal the presence of a mine in covered cells.

There are a number of possible reasoning strategies that the agent can employ, the agent goal is to employ one that limits the amount of guesswork probing needed to solve the game, in order to increase the chances of winning. For more details on the strategies, please refer to the lecture slides.

4 The tasks

The task for Part 1 is to implement a basic Obscured Sweeper agent by probing cells in order. The main objective of this part is to ensure that you are familiar with the rules of the Obscured Sweeper game, and are able to print the output in the required format (See Section 4.6).

For Parts 2 to 4, your task is to investigate how far the different strategies will get you in the given Obscured Sweeper worlds. The logical agent can only flag or probe cells that are proven to contain or not contain a mine. Hence, with correct implementation, the agent should never lose the game, but may arrive at a situation where the game is not won but the strategy(-ies) no longer yields any logical conclusion that a cell could be flagged or probed. Your task is to find this situation in each of the worlds provided and output the agent's view at this point.

4.1 Part 1 - Basic Obscured Sweeper Agent

A basic Obscured Sweeper agent is one that can play the Obscured Sweeper game by only probing cells on the board. You are required to write a program that takes any of the Obscured Sweeper worlds provided and permits an agent to play the game by probing the cells in order. The agent should hold a knowledge base indicating the cells that are yet to be probed, and the information about the cells already probed. The agent's view of the world should be clearly distinct from the board representing the world. In addition, the agent should be able to probe non-blocked covered cells **in order** (left to right on each row; top row to bottom row), receive and process the information contained in that cell. The agent should also be able to establish whether the game is over.

4.2 Part 2 - Beginner Obscured Sweeper Agent

A beginner Obscured Sweeper agent is one that is able to flag potential mines to avoid a losing, taking into consideration the knowledge acquired so far. Instead of probing in order, this agent plays the Obscured Sweeper game using the single point reasoning strategy (SPS) from lectures, adapted for the Obscured Sweeper game. You may probe or flag the cells in any order you like, but you may only probe or flag cells that are proven to be safe or unsafe. As mentioned before, your task is to find the situation where the the strategy ceases to provide any more logical information, and print the agent's view at this point (see Section 4.6).

4.3 Part 3 - Intermediate Obscured Sweeper Agent

An intermediate Obscured Sweeper agent plays the Obscured Sweeper game using the satisfiability test reasoning strategy (SATS) from lectures, adapted for Obscured Sweeper game, in addition to the SPS. The agent should be able to transform its current partial view of the world into a logic sentence, and use satisfiability results to select the next move. For this part, you should use the "DNF encoding" from lectures to write the logic sentences. The library LogicNG[2] should be used to prove that a given cell does or does not contain a mine. As in Part 2, your program should stop when no other deduction could be made on whether a cell is safe or not.

4.4 Part 4 - Intermediate Obscured Sweeper Agent 2

As in Part 3, the agent in this part also uses the SPS and SATS to play the Obscured Sweeper game. However, you should write your logic sentence directly in conjunctive normal form, transform your sentence into DIMACS format, before feeding it into a SAT solver. For this part, you should use the "CNF encoding" from lectures to write the logic sentences, and you should use SAT4J Core[3] to solve the satisfiability problems. Note that you are **not** allowed to use LogicNG for this part of the practical, nor are you allowed to use other libraries or own code that transforms logical sentences into conjunctive normal form. If done correctly, the final agent's view should be the same as that of Part 3. In addition, you should compare and contrast these two encoding in your report.

4.5 Part 5 - Additional Advanced Logical Agent

It is strongly recommended that you ensure you have completed the requirements for previous parts before attempting any of these additional requirements. You may consider one or two additional functionalities:

1. Suggest an additional strategy to improve the performance of the logic agent (some inspiration may be drawn from the Minesweeper wiki[4]).
2. Implement and compare other methods of encoding the knowledge base (for example other encoding in this resource[5]).
3. Include additional items or rules in the game, write the rules for these items and extend the implementation of the game to include these new rules (some inspiration may be drawn from the Wumpus game).
4. Permit the agent to play on boards of a different shape (e.g. triangular or hexagonal).
5. Use the satisfiability approach to solve other similar games on a grid, such as Hitori[6] or Soduku[7]. Test this approach only on small environments/reduced boards of those games.
6. Suggest your own additional requirements: for a significant extension, you may increase the complexity of the game and extend your reasoning strategies to guide the agent to solve the problem, or identify additional games that can be solved with similar logic strategies.

[2] `https://github.com/logic-ng/LogicNG`

[3] `http://www.sat4j.org/products.php#core`

[4] `http://www.minesweeper.info/wiki/Windows_Minesweeper`

[5] Frisch, Alan M. & Paul A. Giannaros. "SAT encoding of the At-Most-k Constraint: Some Old, Some New, Some Fast, Some Slow." (2010).

[6] `https://en.wikipedia.org/wiki/Hitori`

[7] `https://en.wikipedia.org/wiki/Sudoku`

4.6 The output

Your program should output the agent's view at the end of the game. So, for Part 1, that is the situation where you have won or lost the game, whereas in Parts 2 to 4, it is the situation where your strategy does not yield any more logical moves or when you have won the game.

An agent's view is a map with the following as keys:

- `'?'` to mean that the cell covered,
- `'*'` to mean that the cell is flagged for a mine,
- `'b'` to means that the cell is blocked,
- `'0'–'8'` to mean the probed cells not containing mines, with the number is number of mines in the neighbouring cells on the underlying map.

Your program should also output the result of the game, i.e. game lost, game won or no other logical moves (see starter code for the exact wordings).

In addition to that, your program should also allow an additional parameter. If set to `verbose`, the agent's view at each step should be printed. Otherwise, your program should only output the final agent's view. Sample code to do these is provided in the starter code and you are also given some sample output to check if you meet these requirements (see Section 5.4).

5 Code Specification

5.1 Code Submission

The program must be written in Java and your implementation must compile and run without the use of an IDE. Your system should be compatible with the version of Java available on the School Lab Machines (Amazon Corretto 11 – JDK11). The libraries LogicNG and SAT4J are included in the started code under directory called **libs/**, which should include any other external libraries you choose to use (note that you may need to edit `playSweeper.sh` to include external libraries). Please do *not* use libraries that implement minesweeper algorithms, but other libraries that are secondary to the objectives of the practical can be used. Your source code should be placed in a directory called **src/**. The code must run and produce outputs as described in the next sections.

Please note that code that does not adhere to these instructions may not be accepted.

5.2 Starter Code

For this practical, two starter classes (`World.java` and `A2main.java`). The class `World.java` provides the Obscured Sweeper worlds to be used, 36 in total, defined as Enum Type[8]. The Obscured Sweeper worlds are named according to different board sizes: `SMALL, MEDIUM, LARGE`. Six additional worlds (`TEST1` – `TEST6`) are included.

The `A2main.java` is a skeleton class containing code that showcases how to set the board to be used, and how to print it. Note that the indices shown on the board in Figure 1 correspond to the indices of the two dimensional char array given in the starter code. You are free to use this starter code as is or you may modify it, however, please ensure that you do not change the worlds to be tested (you may include additional ones), and that the outputs are formatted in the same way as demonstrated in `A2main.java`.

[7] `https://aws.amazon.com/corretto/`

[8] `http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html`

You are also given a simple script `playSweeper.sh` that includes the libraries, compile and run your program. The primary purpose of the script is to support automated checking (see Section 5.4). Please ensure that you edit this file to include any other libraries you are using.

5.3 Running

Your code should run using the following command:

```
java A2main <P1|P2|P3|P4> <ID> [verbose] [<any other param>]
```

where ID is the world name. For example, to run an agent of Part 1 on the board in Figure 1 where the agent's view after each action is displayed, we use:

```
java A2main P1 TEST1 verbose
```

As another example, to run the intermediate agent of Part 3 on the world named `TEST2` without the verbose option we use:

```
java A2main P3 TEST2
```

For Part 1 and the non-verbose version of Parts 2 – 4, the output of the system must be as described as we will perform some automatic tests on the required output. For advanced agent functionalities (Part 5), please include clear running instructions.

5.4 Automatic Testing

We provide some stacscheck tests to help you check that your output is structured with the required format. The tests only check the 3 by 3 boards `TEST1` – `TEST6`, for Part 1 and the non-verbose version of Part 2. They can be found at `/cs/studres/CS5011/Practicals/A2/Tests` and can be run with `stacscheck`.

In order to run the automated checker on your program, place your program in a directory in the School lab machines, ssh into one of the School computers running Linux if you are working remotely, then change directory to your CS5011 A2 directory and execute the following command:

```
stacscheck /cs/studres/CS5011/Practicals/A2/Tests
```

Please note that these tests are not designed to check that your program works correctly, please do test your system well and provide information on your own testing in the report.

6 Report

You are required to submit a report describing your submission in PDF with the structure and requirements presented in the additional document *CS5011_A_Reports* found on `studres`. The report includes 5 sections (Introduction, Design & Implementation, Testing, Evaluation, Bibliography) and has an advisory limit of 2000 words in total. The report should include clear instructions on how to run your code. Consider the following points in your report:

1. Describe the implementation of the game infrastructure.
2. Describe the implementation of agents and their strategies.
3. Evaluate the agent performance on the given Obscured Sweeper worlds. Comment on the percentage of covered cells at the end of the game where the logical strategies do not yield any other deductions, and any other performance feature of the agents using different strategies.

7 Deliverables

A single ZIP file must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected.

Your ZIP file should contain:

1. A PDF report as discussed in Section 6
2. Your code as discussed in Section 5

8 Assessment Criteria

Marking will follow the guidelines given in the school student handbook. The following issues will be considered:

- Achieved requirements
- Quality of the solution provided
- Examples and testing
- Insights and analysis demonstrated in the report

Some guideline descriptors for this assignment are given below:

- For a mark of 8 to 10: the submission implements the basic agent of Part 1, with the ability of probing cells and determining whether the game is over, adequately documented or reported.
- For a mark of 10 to 13: in addition to the basic agent, the submission implements the beginner agent of Part 2, in full at the higher end of this range. The code submitted is of an acceptable standard and is tested appropriately, and the report describes clearly what was done, with good style.
- For a mark of 13 to 16: the submission implements the agents of Parts 1 and 2, as well as the intermediate agent in Part 3, with complete implementations of all parts at the higher end of this range. It contains clear and well-structured code that is tested well, and a clear report showing a good level of understanding of design and evaluation of logical agents.
- For a mark of 16 to 18: the submission implements Parts 1 – 3 of the practical, as well as the requirements in Part 4, with complete implementations of all parts at the higher end of the range. It contains clear, well-designed code with comprehensive tests, and a clear and well-written report showing real insight into the design and evaluation of logical agents.
- For a mark above 18: the submission completes Parts 1 – 4 of the practical, and one or two (max) advanced agent functionalities. The submission should demonstrate unusual clarity of implementation, together with an excellent and well-written report showing evidence of extensive understanding of design and evaluation of logical agents.

9 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Mun See Chang & Alice Toniolo

cs5011.lec@cs.st-andrews.ac.uk

February 10, 2022

Revisions:

- Changed the ”PART”s to ”P”s in Section 5.3
- First version: February 8,2022