CS5011: Assignment 3 - Logic - Tornado Sweeper

Student ID: 220029176

Date of Submission: 29/03/2023

Word Count: 1942

Contents

1. Introduction

The objective of this assessment was to design and implement a logical agent that is able to perceive certain facts about the world it occupies, and act accordingly.

1.1 Parts Implemented

Strategy	Status
Basic	Attempted, Tested, Fully Working.
Beginner	Attempted, Tested, Fully Working.
Intermediate	Attempted, Tested, Fully Working.
Intermediate V2	Attempted, Tested, Fully Working.
Extension: Random	Attempted, Tested, Fully Working.
Extension: Square	Attempted, Tested, Fully Working.

1.2 Compilation and Execution Instructions

1.2.1 Running the Code

To execute the code, please do the following:

1. Download the zip file "CS5011-Assignment3-220029176" and unzip it to a chosen folder location.
2. Open a terminal prompt from within the src folder location.
3. Enter the following command into the terminal "javac A3main.java"
4. Enter the following command into the terminal "java A3main" following by the desired logical agent, map ID and optional verbose and square, please see structure within 1.2.2.

1.2.2 The command structure

The agent runs on the following structure:

java A3main <P1|P2|P3|P4|P5> <ID> [verbose] [square]

<P1|P2|P3|P4|P5>: The logical agent to play the game.

<ID>: The map ID.

[verbose]: If you wish to display the steps involved. **[Optional]**

[square]: Used to configure the logical agent setting to square maps **[Optional]**

Note: Square parameter can only be used with a square map, please see worlkd.java for possible map configurations.

2. Design and Implementation

2.1 PEAS Model

The first step to achieving a logical agent was to design the PEAS model. The PEAS model is broken down into 4 sections: Performance measure, Environment, Actuators and Sensors.

Performance: The performance measure for the logical agent is to uncover all cells on the board except those containing Tornadoes, without losing the game.

Environment: The tornado sweeper world is deterministic, partially observable, static and unknown to the logical agent where the board is in the shape of a rhombus with N cells per side, where each cell is of hexagonal shape. T tornadoes are scattered among the cells.

Actuators: The agent can probe a cell, which will either reveal a Tornado or a clue indicating the number of Tornadoes in the 6 adjacent neighbours of the probed cell. The agent can also flag a cell to signal the presence of a Tornado within parts 2 to 5.

Sensors: The agent receives information about the content of the cell when it probes a covered cell. If the cell contains a tornado, then the game is lost. The agent is also given two starting clues about safe cells it can probe at the beginning of the game.

2.2 Game Infrastructure

The A3main.java file contains the game infrastructure, including the board view based on user arguments, which is represented as a char[][]. It also includes functions that determine whether the game has been won. In contrast, the knowledgebase.java file contains the game board used by the agent, represented as a matrix of cells with the initial clue as '?'. This board is built independently of the actual board, but its dimensions are taken from the original.

2.3 Agent Infrastructure

The Agent.Java file contains the implementation of the agent infrastructure, including the class definition that represents an instance of the agent. This class includes methods that facilitate the PEAS model, such as probing and marking cells, and retrieving free and marked neighbours. In the probing cell functionality, the character of the given cell is taken from the initial game board, and the function checks for a tornado. If a tornado is found, the game is over. Otherwise, if a 0 is found, the neighbours of the cell are probed as well. The marking functionality places a flag on the provided cell to indicate that there is a tornado, which is represented by the '*' character according to the specification. The knowledge base contains two lists: order and probed. Initially, all cells are added to the order list to be probed. Once a cell has been probed, it is moved between the two lists.

2.4 Agent Strategies

For this assessment there were four necessary agent strategies: basic, beginner, intermediate and intermediate v2. An additional strategy was developed to improve upon the beginner strategy called random. Each strategy was built as an extension of the previous level, utilising the functionality.

Basic Strategy

This was a straightforward strategy as no logical steps are taken when a unknown cell is present. The cells are probed in order from left to right and top to bottom. As cells cannot be flagged, the winning criteria for this strategy was to have the number of unknown cells match the number of tornados on the board.

Beginner Strategy

The agent uses this strategy to mark or probe cells based on the single point strategy. Initially, the top left cell is probed and expanded accordingly. Then all the known cells are scanned one by one for unknown neighbours. If an unknown neighbour is found, it is added to the single point strategy, which first checks if the unknown cell has either all free neighbours or all marked neighbours. If the cell has all free neighbours, then all the unknown neighbours are probed. If the cell has all marked neighbours, then all the neighbours are marked as containing a tornado. If the agent does not have enough information about the cell yet, it is placed at the back of the order list for reinspection later. By using this strategy, the agent systematically probes and marks cells on the game board, gradually

increasing the amount of information it has about the board and improving its chances of winning the game.

Intermediate Strategy

This strategy implements the satisfiability test reasoning strategy (SATS) from the lectures. Initially, the top left cell is probed and expanded accordingly. Then the disjunctive normal form (DNF) encoded knowledge base by converting the knowledge base into a logical sentence using looping through all the unknown cells that neighbour known cells. The logical sentence contains all the possibilities of where tornados could be created in DNF. Once the DNF encoded knowledge base has been created, each unknown neighbour is checked for whether it is safe or a tornado using the LogicNG library. The logical sentence is appended with the cell to be checked and tested for satisfiability using the SATSolver and Tristate classes from LogicNG. If the cell is determined to be safe, it is probed else if it is a tornado then it is flagged. This process continues for each unknown neighbouring cell. Once all cells are checked the next cell from the order is taken and the process loops until the order is exhausted.

Intermediate V2 Strategy

This strategy uses the SPS and the SATS to play the game. Like the intermediate strategy, a knowledge base is built to determine whether an unknown cell is safe or a tornado. However, the approach varies considerably. Within this strategy, the logical sentence is constructed in conjunctive normal form (CNF) by getting the number from the known neighbour's hint (N) and retrieving all possibilities where the unknown neighbours are at most N tornados and at least (N – neighbouring flagged cells) safe. The logical sentence is then converted to DIMACS format, this is done by breaking the sentence down into the individual clauses and converting each clause into an integer array. Once all the clauses are converted, each unknown neighbouring cell is checked whether the cell is safe or a tornado using the SAT4J library's ISolver and IProblem functionalities.

Extension: Random Strategy

Based on a real Mine Sweeper strategy, by randomly guessing cells when all other options are depleted, we can increase our chance of winning the game at the risk of hitting a tornado (Mine Sweeper, 2021). The beginner strategy often cannot infer the safety or danger of a cell based just on the SPS. Therefore, to increase the chances of success within the beginner strategy, a random strategy was extended. The random strategy involves a continuous loop of SPS and random probing where if the agent fails by SPS then a cell is randomly probed until the game is won or lost.

Extension: Square

Though tornado sweeper poses its own challenges and fun for the player and/or agent the most popular sweeper game is Mine Sweeper. Mine Sweeper implements a square board with each cell have nine neighbours. Like tornado sweeper, mines are placed randomly on a given board and are static. To mimic this popular game, map configurations were added to the world.java file as well as additional functionality was added to the knowledge base to allow the agent to take the additional neighbours needed for a square grid. Please see Section 1.2 for running instructions.

Note, the agent can only run correctly off the new maps as the hints are correctly configured for a nine-neighbour grid.

3. Testing

Once each of the four strategies had been created, they were extensively tested to ensure for accuracy and efficiency. Each of the four strategies were tested against the provided studres tests. All the strategies successfully passed these tests.

4. Evaluation

This project aimed to implement an agent that is able to play and solve a logic game, namely Tornado Sweeper. Initially a basic Tornado Sweeper was implemented that simply probed non-blocked covered cells in order of left to right, top to bottom, to receive and process the cell information. Then, a beginner Tornado Sweeper was created, utilising the single point reasoning strategy (SPS) to be able to flag potential tornados and avoid losing. This was then built upon to include satisfiability test reasoning strategy (SATS) transforming the agents partial view into a logic sentence. In Part 4, the conjunctive normal form was transformed into a DIMACS format before being fed into the SAT solver retrieving the same results as Part 3. During the testing of part 3 and 4, part 4 took less iterates to determine a final output. Therefore, CNF is more efficient than DNF within this game.

Finally, two advancements were made to improve the strategies. Firstly, a continuous loop of SPS and random probing was added to the Basic Agent to enable it to infer the safety of a cell. And secondly, a square board was implemented to enable the agent to coordinate with nine other neighbours. Both extension tasks required a complex understanding to implement. I found this an enjoyable challenge and benefitted from the creative nature of the task.

On reflection, during the testing stage, one of the Basic studres tests initially failed when it was run. When investigating this issue, it was because the test was looking for a mine and not a tornado. Because of this, in the Basic code you may find the term ‘mine’ or ‘bomb’ instead of tornado, this is to suit the requirements of these tests.

Due to time limitations, I was unfortunately unable to complete any further testing. This was attempted but I was unfortunately not able to resolve any issues before the deadline, and therefore was removed. In further research, this must be completed to test the extremes of the system. This will ensure for the reliability and efficiency of the system.

Furthermore, during the initial stages of development the game board was kept as an array and then was used in both the original and agent view. Within the agents view, the game board arrays sits inside the knowledge base and not in its own separate class. In hindsight this would have been better implemented as its own separate class for the original and agent view.

Overall, this project has designed and implemented a logical agent that is able to perceive certain facts about the world it occupies, and act accordingly. If time allowed, further testing should have been completed to test the fully capacity of the system.

References

Mine Sweeper. (2021, April 18). *Strategy.* Retrieved from minesweepergame: https://minesweepergame.com/website/authoritative-minesweeper/wiki/Strategy