CS5011: Assignment 2
Logical Agents - Daggers and Gold Sweeper

(Words in the essay: 2160, Time spent: 30hrs) **Student ID:180029539**

1. Parts Implemented:

- I have implemented part 1 which contains both the algorithms for random probing and single point strategy to solve the game.
- I have implemented part 2 which contains the implementation for Satisfiability Test strategy to solve the game.
- I have also implemented part 3. I have provided a text interface for the user to enter the cell he wants to probe. If CPU probes more gold than user then CPU wins else user wins.

2. Working Functionalities:

- All the functionalities in both the part 1 and part 2 working as expected. The worlds that are supposed to be solved using SPS and ATS are being solved as per the correct logic.

3. Literature review:

- The use of logical inference to solve problems has been around for a long time now
- The use of logical inference helps us understand the problem space better and explore the parts of problem space that are not possible without inference
- The base to solving problems or games like minesweeper is to build a knowledge base of the current state of the game
- Then we use propositional logic to perform operations on the knowledge base to derive new information about the problem space
- Minesweeper is a very good example of this.
- In Minesweeper, we first use techniques like single point strategy unlock as much board as we can.
- Then we use different kinds of solvers like DPLL which take knowledge base as input and tells us if one particular cell has a mine or not.
- It's an NP-complete problem and also fits in perfectly with the conditions of a Constraint Satisfaction problem.

4. Design:

- PEAS model:
    - Performance: The main performance metric would be the number of guesses made before the game was completed. But, we can also compare the ratio of wins over losses.

- Environment: A grid of m rows and n columns with a finite number of daggers and gold placed in some cells. The other cells contain the number of daggers in its neighbour cells.
- Actuators: The agent is allowed to probe a cell, mark a cell, and identify if it contains a clue regarding the neighbours.
- Sensors: The number of dagger and number of unexplored neighbours might be used as sensors to perceive the knowledge of the present cell and number of dagger and number of unexplored cells in the world can be used to perceive the knowledge base of the world explored so far.

Part 1 - Process Flow:

- The user is prompted whether he wants the agent to solve using random probing or single point strategy.
- The program first starts out by taking an input of difficulty of the game from the user and then the game class selects a world randomly for the user.
- Next, the game calculates the number of daggers in the world and returns it back to the agent playing the game.
- Then, the agent initializes a map with the same number of rows and columns and initializes all the cells in the grid as unexplored with a letter “u”. Then initializes the number of lives with 1.
- Throughout the game, whenever the agent probes a dagger it loses a life and gains a life when it probes a gold.
- Whenever the count of lives hits zero, the game is lost.
- **Note: My grid starts from 1,1 and not 0,0. As I have used an extra row on the top and bottom and an extra column on the left and right as padding for my grid. It is done to make the checking of neighbours for the cells at boundaries easy.**
- So, the agent probes the first cell [1,1] then the algorithm starts.
  Random Probing:
- First, the algorithm goes through the grid cell by cell and checks the value of each cell.
- If it is “0” or “g”, it probes all the unexplored neighbours and moves to the next cell.
- If it completes the grid, it goes back to the beginning. But, in every loop, it keeps note of the updated cells for that loop. If no cell is probed in that loop by the time it reaches the end of the loop, it randomly probes a cell.
- Then, it starts with the algorithm again.

- This goes on until either all cells in the grid are probed or the number of daggers of the world is equal to the sum of unexplored cells and marked cells and daggers probed.

Single Point Strategy:

- First, the algorithm goes through the grid cell by cell and checks the value of each cell.
- If it is “0” or “g”, it probes all the unexplored neighbours and moves to the next cell.
- If it is a clue and has unexplored neighbours, it calculates the sum of marked and uncovered dagger neighbours and checks if it is equal to the value of the cell. If it is equal, it probes all the unexplored neighbours.
- Then, if both the above conditions are not true, it will check if the value is equal to the sum of unexplored neighbours, marked neighbours and dagger neighbours. If it is equal, then it marks all the unexplored neighbours as a dagger.
- If it completes the grid, it goes back to the beginning. But, in every loop, it keeps note of the updated cells for that loop. If no cell is probed in that loop by the time it reaches the end of the loop, it randomly probes a cell.
- Then, it starts with the algorithm again.
- This goes on until either all cells in the grid are probed or the number of daggers of the world is equal to the sum of unexplored cells and marked cells and daggers probed.

Part 2 - Process Flow:

- The user is prompted whether he wants the agent to solve using random probing or single point strategy.
- The program first starts out by taking an input of difficulty of the game from the user and then the game class selects a world randomly for the user.
- Next, the game calculates the number of daggers in the world and returns it back to the agent playing the game.
- Then, the agent initializes a map with the same number of rows and columns and initializes all the cells in the grid as unexplored with a letter “u”.
- Then, the agent probes cell [1,1]. Till here the working is the same as part 1. But the algorithm changes for part 2.
- We will use the SAT4J library and implement Satisfiability Test Strategy.

Satisfiability Testing Strategy:

- The agent first implements Single point Strategy on the grid and keeps track of the updated cells for each iteration of the loop.
- As soon as it completes a loop without updating any cells Satisfiability testing strategy is invoked.

- As a part of this, we first generate the Knowledge Base for the current state of the game.
- To do this we loop through the current state of the grid cell by cell and if any cell has unexplored neighbours and contains a clue, we build a formula based on the no of daggers that can be placed in the neighbouring cells.
- Then it is appended to the Knowledge Base and moves on to the next cell.
- After the knowledge base is constructed, we append the “&<formula of the cell>” to be tested and we pass it to the SAT solver.
- If it returns not satisfiable, then we probe the cell.
- Then we return back to SPS and see if we SPS can unlock any of the cells
- If SPS fails, we try the satisfiability test strategy again. If still unable to unlock any cells then we revert to random probing.
- Generating Knowledge Base:
    - First, we start exploring the cells one by one to get all the cells with a value and unexplored neighbours.
    - Then, each cell with value is used to generate a formula for its neighbours.
    - Next, we merge all the individual formulae and combine them using “&”.
    - This results in the KBU of the current state of the game.

Common features between both the parts:

- The game functionalities are separated from the agent functionalities in both the parts.
- The Game class retrieves the world to work on and returns the state of the board to the agent.
- The agent executes the algorithms and tries to uncover the world.
- The agent communicates with the game only in two situations. When it probes a cell, it asks the Game class the contents of the cell. Next, when the game is initiated the agent gets the number of daggers in the game from the Game class.
- Apart from the main algorithm of the individual parts, the other functionalities are the same between both the parts and are decomposed into methods.
- Some of the methods used are getMap(), displayMap(), getNoOfUnexploredCells(), getNoOfMarkedCells(), getNoOfDaggerCells(), getNoOfGoldCells() which return information about the current state of the game.
- The other methods used are getNoOfUnexploredNeighbours(), getNoOfMarkedNeighours(), getNoOfDaggerNeighbours(), probeAllNeighbours(), markAllUnexploredNeighbours(), probeCell(), isCellANumber() return the properties of the cell we are exploring.
- Methods like useSATSolver(), getKBU(), getFormulaForCell(), getUnexploredNeighbours(), checkSatisfiability(), clauseToIntArray() are specific to part 2.

- All the method names give the precise description of the purpose of the method.

Testing and Examples:

Part 1:

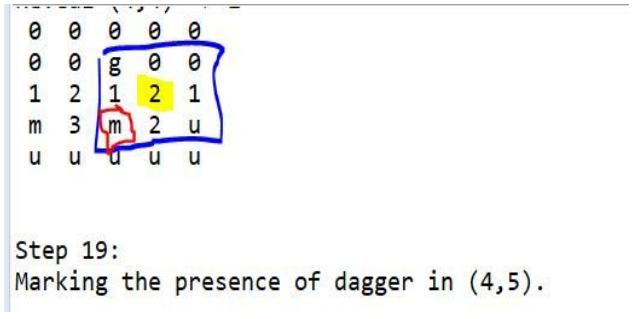
- I have tested all the functionalities of the algorithms and they are working fine.
- Let me discuss some scenarios to demonstrate the working of the game.
    - Scenario 1: Only when the world cannot be solved using SPS, it shifts to random probing.
        - Test:

```
Step 8:
Reveal (3,3) -> 2
 0  2  m  u  u
 0  2  m  u  u
 1  2  2  u  u
 u  u  u  u  u
 u  u  u  u  u


Using Random Probing on (4,4)

Step 9:
Reveal (4,4) -> 1
```

    - Scenario 2: Marks all the unexplored as daggers only if the value of the cell is equal to the sum of unexplored neighbours, marked neighbours and dagger neighbours.
        - Test:



- Scenario 3: Probes all neighbours if the value is “0” or “g”
  - Test:

```
Step 0:
Reveal (1,1) -> g
Goldmine in (1,1), new life count: 2.

Step 1:
Reveal (1,2) -> 1

Step 2:
Reveal (2,1) -> 1

Step 3:
Reveal (2,2) -> 2
 g  1  u  u  u
 1  2  u  u  u
 u  u  u  u  u
 u  u  u  u  u
 u  u  u  u  u
```

- Scenario 4: Probes all neighbours when the value of a cell is equal to the sum of the dagger and marked neighbours.
  - Test:

```
 0  2  m  2  g
 0  2  m  3  1
 1  2  2  2  u
 1  d  1  1  u
 u  u  u  u  u


Step 18:
Reveal (5,1) -> 1

Step 19:
Reveal (5,2) -> 1
```

Part 2:

- Scenario 1: Use SAT Solver only when SPS cannot solve the current status of the game
  - Test:

```
 0  0  0  2  m
 0  g  0  2  m
 1  1  2  2  2
 u  u  u  u  u
 u  u  u  u  u


Using SAT Solver to unlock (4,1).
```

- Scenario 2: Revert to SPS after ATS probes a cell
  - Test:

```
Using SAT Solver to unlock (4,1).

Step 15:
Reveal (4,1) -> 1
SAT Solver ends

SPS Starts


Step 16:
Marking the presence of dagger in (4,2).
```

Performance evaluation:

Part 1:

- The Random probing strategy wins about one or two games after making a significant amount of guesses before finishing the game.
- It becomes even less likely when tested on medium and hard worlds.
- The Single point strategy wins about 70% of the time by making around 0 to 3 guesses approximately.
- But as we test it on medium and hard levels, the winning percentage lags just by a small percentage but the number of guesses has a high increase.

Part 2:

- The SAT Solver has an 80-90 % winning rate but still has small situations where random probing is required.

World and Algorithm Comparison:

- Below you can see which algorithms can solve which given worlds.
- Some worlds require both SAT Solver and SPS to solve them. These are marked with an “A” in the table.
- The worlds that can be solved using only SPS can also be solved using SAT Solver. So, they are marked with both “S” and “A”.
- “G” specifies the worlds that definitely need random probing to solve them.

	Easy	Medium	Hard
World 1	G	S, A	A
World 2	G	A	A
World 3	S, A	S, A	A
World 4	S, A	A	S, A
World 5	G	A	A
World 6	S, A	G	G
World 7	A	G	G
World 8	A	G	A
World 9	S, A	G	G
World 10	S, A	A	G

Symbol	Meaning
S	Can be solved using SPS
A	Can be solved using SAT Solver
G	Random Probing Required

Evaluation Analysis:

- The use of SAT Solver really boosts the chances of winning and reduces the number of guesses required to make.
- But, the use of SAT Solver becomes really complex for worlds with a higher number of cells.
- The knowledge base exponentially increases with increase in rows and columns of the world, which takes huge amounts of memory and time by SAT solver to parse the formula and test for satisfiability.

Running Instructions:

- You can directly run both the jars without any parameters.
- For part 1, you can run java -jar Logic1.jar
    - First, it will ask you whether you want to use random probing or single point strategy.
    - After you pick a number between 1 & 2, you have to select the difficulty of the game.
    - After you enter a number between 1 to 3, you will get the result of the game and whether you want to play again or not.
- For part 2, you can run java -jar Logic2.jar
    - First, it will ask you for the difficulty of the game.
    - After you enter a number between 1 to 3, it will play the game and give you the result.
    - Finally, you will get an option whether you would like to play again or not.
- For part 3, you can run java -jar Logic3.jar
    - First, it will ask you for the difficulty of the game.
    - After you enter a number between 1 to 3, it will probe a cell.
    - Then you have to enter the x and y coordinates of the cell you want to probe
    - Then the CPU probes.
    - This goes on until all cells are explored, then the winner will be declared.

References:

1. **Richard Kaye. Minesweeper is NP-complete. The Mathematical Intelligencer, 22(2): 9–15, 2000.**
2. **Stuart J. Russell and Peter Norvig. Artificial intelligence: a modern approach. Prentice Hall series in artificial intelligence, 2010.**