Introduction to Logic Project

Problem: Minesweeper

By Jiramet Harnjan 68011354

"Minesweeper is a puzzle video game. In the game, mines are scattered throughout a board, which is divided into cells. Cells have three states: unopened, opened and flagged. An unopened cell is blank and clickable, while an opened cell is exposed. Flagged cells are those marked by the player to indicate a potential mine location.

A player selects a cell to open it. If a player opens a mined cell, the game ends. Otherwise, the opened cell displays either a number, indicating the number of mines vertically, horizontally or diagonally adjacent to it, or a blank tile (or "0"), and all adjacent non-mined cells will automatically be opened. Players can also flag a cell, visualised by a flag being put on the location, to denote that they believe a mine to be in that place. Flagged cells are still considered unopened, and a player can click on them to open them."[1]

Propositional atoms:

x denote columns from the left starting from zero.

y denote rows from the top starting from zero.

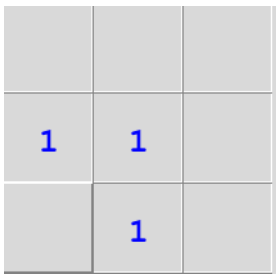
$m_{x,y}$ denotes a mine exists at cell (x, y)

$\neg m_{x,y}$ denotes a mine does not exists at cell (x, y)

Formula:

A cell with number n where $0 > n \geq 8$ must only have amount of adjacent mines less than or equal to n.

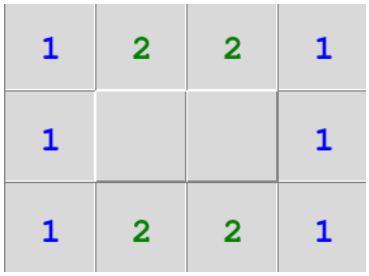
For trivial cases where a number cell that matches exactly the same amount of adjacent unopened cell, it can be deduced that all of those unopened cells is a mine. For example:



Cell (1,1) has 1 mines nearby–
and has a one adjacent unopened cell (0,2).

We can construct the formula

$$m_{0,2}$$



Cell (1,0) has 2 mines nearby–
and has two adjacent unopened cells (1,1) and (1,2).

We can construct the formula

$$m_{1,1} \wedge m_{1,2}$$

[1] https://en.wikipedia.org/wiki/Minesweeper_(video_game)

By applying the trivial cases repetitively, revealing more numbers and clues, many of the games can be solved to the end without SAT solver.

But there are cases where a single number cell cannot determine exactly which the adjacent unopened cells is a mine and must use multiple number cells to be able to deduce the mines location.

We can imagine a simpler cases first. In each cases below, we will be focusing on just one number cell as the center point of view where its number is "n nearby mines".

Let denotes each adjacent cells that is a mine as a, b and c. We're only looking within locally adjacent cells so the cells position can be arbitrary as long as it's adjacent, so we don't use $m_{x,y}$ for clarity.

- 2 Adjacent cells, 1 nearby mines:

  $$(a \land \neg b) \lor (\neg a \land b)$$

  Can be translated to CNF as

  $$\equiv (a \lor b) \land (\neg a \lor \neg b)$$

- 3 Adjacent cells, 1 nearby mines:

  $(a \land \neg b \land \neg c) \lor (\neg a \land b \land \neg c) \lor (\neg a \land \neg b \land c)$

  Translated to CNF as

  $$\begin{aligned}
  &(\ \ a \lor \ \ b \ \ \lor \ \ c) \land \\
  &(-a \lor -b \ \ \lor \ \ c) \land \\
  &(-a \lor \ \ b \ \ \lor -c) \land \\
  &(\ \ a \lor -b \ \ \lor -c) \land \\
  &(-a \lor -b \ \ \lor -c) \land \\
  &(-a \lor -b) \land \\
  &(-a \lor -c) \land \\
  &(-b \lor -c) \land
  \end{aligned}$$

- 3 Adjacent cells, 2 nearby mines:

  $(\neg a \land b \land c) \lor (a \land \neg b \land c) \lor (a \land b \land \neg c)$

  Can be translated to CNF as

  $$\begin{aligned}
  &(\ \ a \lor \ \ b \lor \ \ \ c \lor) \land \\
  &(-a \lor \ \ b \lor \ \ \ c \lor) \land \\
  &(\ \ a \lor -b \lor \ \ \ c \lor) \land \\
  &(\ \ a \lor \ \ b \lor - c \lor) \land \\
  &(-a \lor -b \lor - c \lor) \land \\
  &(\ \ b \lor \ \ c \lor) \land \\
  &(\ \ a \lor \ \ b \lor) \land \\
  &(\ \ a \lor \ \ c \lor) \land
  \end{aligned}$$

As we can see, it's very simple to construct the formula which is in DNF. But as there are more possibilities for the mine locations, it gets unwieldy when converted to CNF. For example, with the

maximum possible adjacent cells, 8, and with 1 nearby mines, the converted CNF has 5282 clauses, computed via a Python script. Neglecting maximum possible adjacent cells, I were able to compute up to 15 cells with 1 nearby mines under a reasonable amount of time, resulting in about 14 million clauses. Since these computation can take a large amount of time and affect user experience, the submitted program contains all the CNF clauses already computed and stored within the folder `cnf_cached`. The script used to generate the cache is called `compute_cnf.py`.

After translating each local CNF atoms to a uniquely identifiable atom within the entire board, eg. a to $m_{x,y}$ and combine all local CNF clauses together, we will be able to solve for the mine locations via the SAT solver. Unfortunately, there are cases where there are multiple solutions but only 1 is correct due to how the game is designed and how the board is generated. The SAT solver used in the program only produces 1 out of the many possible solutions and I can't seems to construct more clauses.

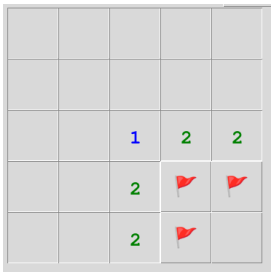
What I decided to do is run the SAT solver in the first step and collect the model from the solver. And by negating an assignment for some single variable, e.g. from `True` to `False` or from `False` to `True`. Then assumes the variable with this new assignment and running the SAT solver again.

If the CNF is unsatisfiable, it means that the variable cannot be changed. This guarantees if the cell is a mine or is not a mine.

If the CNF is satisfiable, it means that the cell can either be a mine or not a mine. It's not safe to assume either.

With this exhaustion method, we can ensure that the model will not result in us stepping on a mine.

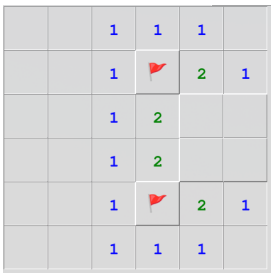
Lastly, for the algorithm, there are cases where we are no longer able to construct CNF from adjacent cells. Such as this board below:



But if we know that there are only 1 mine left, that unopened cell must be a mine. Or if we know that there are no mines left, that unopened cell must not be a mine.

If such a case or similarly appears (by checking that the previous CNF clauses is not satisfiable or no safe solution exists), the algorithm will construct another clauses from the amount of mines left and the remaining unopened cells. It uses the same cached CNF clauses (With limitation up to 12 remaining unopened cells). Then combine this new clause with the previous adjacent cells clauses, and also runs it through the same exhaustion method, will results in this board being solved.

And there are cases where the board can only be solved by guessing, such as this board below:



There are 2 mines left, but there are 2 possibilities, with no information to decide between them. The solver will stops in this case and leave up for the user to probe for mines.

The Python program can be view here: https://github.com/peach-on-the-way/tkinter-minesweeper