Questions and Write-up

**Representation**

For the code presented in this project, minesweeper board and information is stored using numpy matrix. The input system to set the dimension of the board is started as a popup window where the user can enter dimension for the board and mine density. If the input fields are left blank then The default size should be 10x10 with mine density of 15.

As this project is intended to use inference rules and constraint satisfaction problem the constraint equations are stored in knowledge base as [[Variables , Value] ] – in this case the variables are cell indexes (e.g in inference rules we use A,B,C as examples in our case it would be (0,0), (0,1) and so on ) and Value is the equation value

A picture containing crossword

Description automatically generated

The picture on the right shows the generated maze board solved, where the cell with a flag indicates cell that is flagged, cell with a mine indicates it was open because not enough information was available to deduce or it was opened on a random and the numbers on the cells indicate clues which indicate number of mines present in its adjacent neighboring cells.

(Note: The image on the left are mines on board and is taken from terminal due to some framework issues used in this project )

**Inference:**

Our agent keeps total information of

* Each cell it has visited and flagged (Each cell has a structure to store information)
* Total mines that exists on the board
* List of safe cells
* List of cells visited and unvisited
* Every csp equation formed and stored in the knowledge base
* Duplicate csp equations that already exists and equations that might have a different Equation Value that can play a role in determining whether a cell is a mine or not

This project solves the basic structure that is presented in the project description and an improved algorithm

Basic Algorithm: Follows the description present in the Project outlines and makes comparisons based on singular cells rather than multiple cells, and does not make use of inference rules or the information stores in knowledge base

Advanced Algorithm: This solves the board by considering multiple clues, solving equations and forming new csp equations to deduce new clues for future unrevealed cells.

**Explanation:**

All cells are visited by the agent, and the action to either open or flag the cell depends solely on what information and clues were deduced by making use of the inference rules.

**How the constraint equation for variables are formed:**

When a non-mine variable is opened, an equation is formed that includes adjacent neighboring cell of the open cells, and the equation value is set as the clue the current cell provided. Whenever a cell is opened and it turns out to be a safe cell, it is removed from other equations in the knowledge base and its equation value is updated to 0 (0 indicates as safe) and it is also added to a list of safe cells.

Text

Description automatically generated

How csp equations are stored for inference rules and deduction

From the above image, it can be determined that the non-repeating variable positions (left) with with 1 value are mines, but repeating variable positions with 0 and 1 are not clear clue to deduce anything and they usually result in either random decision or wrong decision. More on the results of how our agent identifies cells and gets past ambiguity is described below

When a cell is either flagged or randomly opened and it is a mine variable, then we first check if it exists in other equations. If it does then we remove it from those equations and make sure we also subtract the variable value from those equations, so it does not affect anything in the future

Whenever a cell is opened or flagged, equations for its adjacent neighbors are formed and stored in the knowledge base

How CSP equations are solved and cells are flagged:

Constraint Satisfaction Problem Equations: Our knowledge base stores the equations in the form of A + B + C = Value, A+B = Value and A = Value (A,B,C are all cell positions on the board to store information corresponding to it) . Each equation is solved in a very systematic way. If the equation is of the length 3, then either two of the options are chosen. If enough information is present then we can either simply apply the value of variables stored in our knowledge base, or partial information is present then try solving in through subset. If not enough information is present in our knowledge base then more information is determined by counting the variables and compared it to the value of the equation. For example, if our equation is A + B + C = 3 and no information is present that can be used (for instance, placing in the values of A, B or C in equation to see what we can get) then for the time being we will set A,B and C to a mine status and placed in a list where we come back to it later when we have more information. Before that happens we try solving it through subsets. For A + B + C = 3, we go through out knowledge base and check whether we have any variables in the knowledge base that are present in our equation. If they are then we first verify if that specific equation satisfies as a subset for our equation in hand. After verifying we get two equations, lets and A+B =2 and B+C=1.

First we try placing A+B in A + B +C and see what the results hold. Through inference rules it is safe to deduce that c = 1 considering A+B = 2 and A + B + C = 3 ( 2 + C = 3 would mean C=3-2)

The same can be done with B+C

We determine that C =1. Also, we can further break down A + B = 2 by checking our information if we have values for A and B. If we do, then we can simply put the values in and confirm if A + B = 2 is true. If we have only one value out of the two we can determine the value of the other variable as well. If A + B = 2 and A=1 then we can determine that B=1 and that way A + B = 2 is satisfied. Using the same information from this we can either place it in A+B or B+C and we can deduce the same result and mark C as a mine.

A picture containing table

Description automatically generated

When we are unable to determine if a cell is safe or not then our decision is either based off pure randomity, in that case we either flag or open the cell, or we try to deduce results from whatever information we have. Above image indicates the decision our agent made when it had to decide if a cell is worth flagging just based off the partial information it had . You can see notice it managed to flag all mines, but also ended up flagging safe cells as well. This is the result it produced due to ambiguity (it can be argued one way to get rid of ambiguity and mark cells even more accurately even with partial information would be to calculate probabilistic heuristic for each decision – in our case I did not implement that bonus feature)

Decision

Performance