Report -Assignment 2-Minesweeper

Sharvani Pratinidhi (NetID: spp133)
Amit Patil (NetID: amp508)
Pranita Eugena Burani (NetID: peb63)

Solved Minesweeper

Yellow cells- Cells visited by the agent
Grey cells- Mines identified by agent

Questions and Writeup

• *Representation: How did you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal?*

Representation of board:
The board is by an object of type Environment which has a matrix to store the minefield and the following functions to interact with the minefield:

1. generateMineFieldValues : To create mines at random locations.
2. generateInformation : To generate clues about mines.
3. queryBox : To return information about the queried box to other programs.

Every element of the minefield matrix is an object of type Box which has the following attributes:

1. row and col: The position of the box in the matrix
2. mine: Indicates whether the box is a mine (IS_MINE) or not (IS_NOT_MINE).
3. visited: Indicates whether the box is visited by the agent (IS_VISITED) or not (IS_UNVISITED).
4. clue: Denotes the clue about neighboring mines. The value ranges from 0 to 8. Initially set to NO_CLUE

Representation of Knowledge/ Information that clue cells reveal
The MainProgram.py creates an object of the Environment and an object of the Agent. Both of these have a minefield of their own. The minefield is represented by a matrix where each element is an object containing the relevant information.

1. The Environment's minefield is generated by giving the row column information (Box[i,j]) to generateInformation(),which stores the number corresponding to number of mines in the neighboring cells (i.e. clue).
2. The Agent's observed minefield has no information about the clues or about the mines. It only knows the row and column. These elements get updated as the agent moves along the cells collecting the clues.

• *Inference: When you collect a new clue, how do you model / process / compute the information you gain from it? i.e., how do you update your current state of knowledge based on that clue? Does your program deduce everything it can from a given clue before continuing? If so, how can you be sure of this, and if not, how could you consider improving it?*

The initial knowledge base is TRUE, whenever a new clue is revealed to the agent, the addToKnowledgeBase() function generates all possible combinations(expressions) of the constraints based on the clue and the neighboring cells. If any of the neighbor has already revealed as the clue (False) or mine (True) then we substitute that value in the above equation. And then we convert it into CNF.
For example, If the clue reveals the number 2 for cell A00 and there are 3 neighbors of that cell A10, A11, A01 then the we create the following equation:

(A01 & A11 & ~A10) | (A10 & A11 & ~A01) | (A10 & A01 & ~A11)

We update the KB depending on the status of the current cell. If the current cell is identified as mine, we update that cell in KB with true value or else with false value. After this the CNF is added to the knowledge base. (KB & expr) Then it combines the equations in the knowledge base to find consistent assignments. This is nothing but converting Minesweeper to Constraint Satisfaction Problem and solving it.
Yes, our program does deduce everything it can from the given clue, because its basically a CSP and every time a new expression is added it takes into account all possible assignments possible and checks if it is consistent or not, but there are cases where the clue cannot exactly conclude what to assign to the variables . Then, it chooses the next cell randomly.

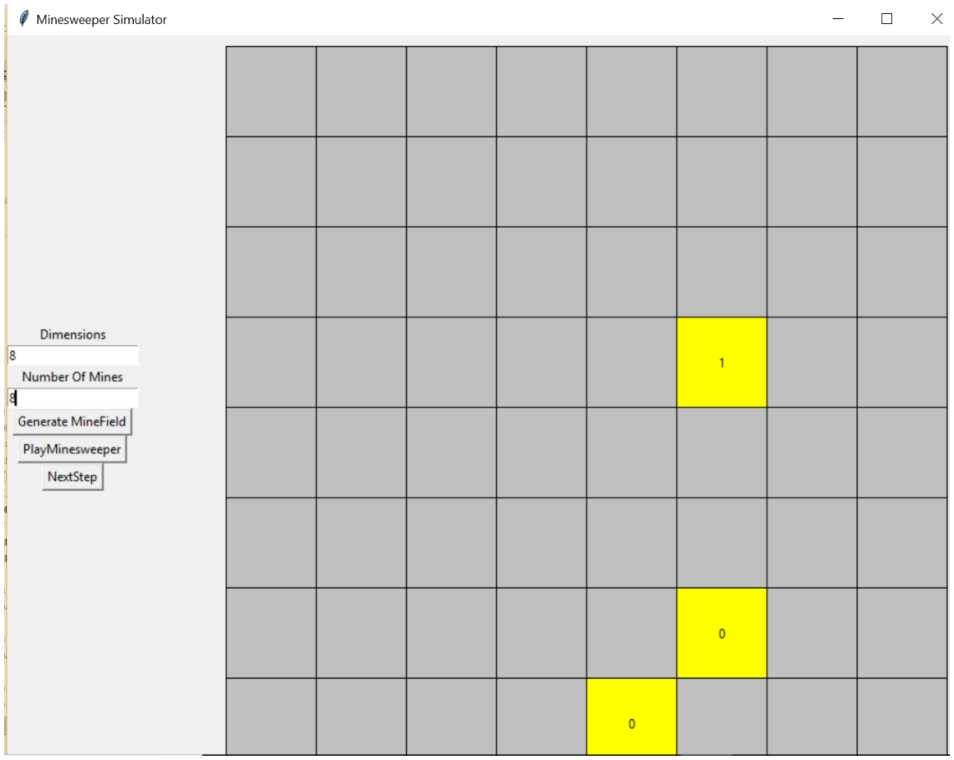
• *Decisions: Given a current state of the board, and a state of knowledge about the board, how does your program decide which cell to search next? Are there any risks, and how do you face them?*

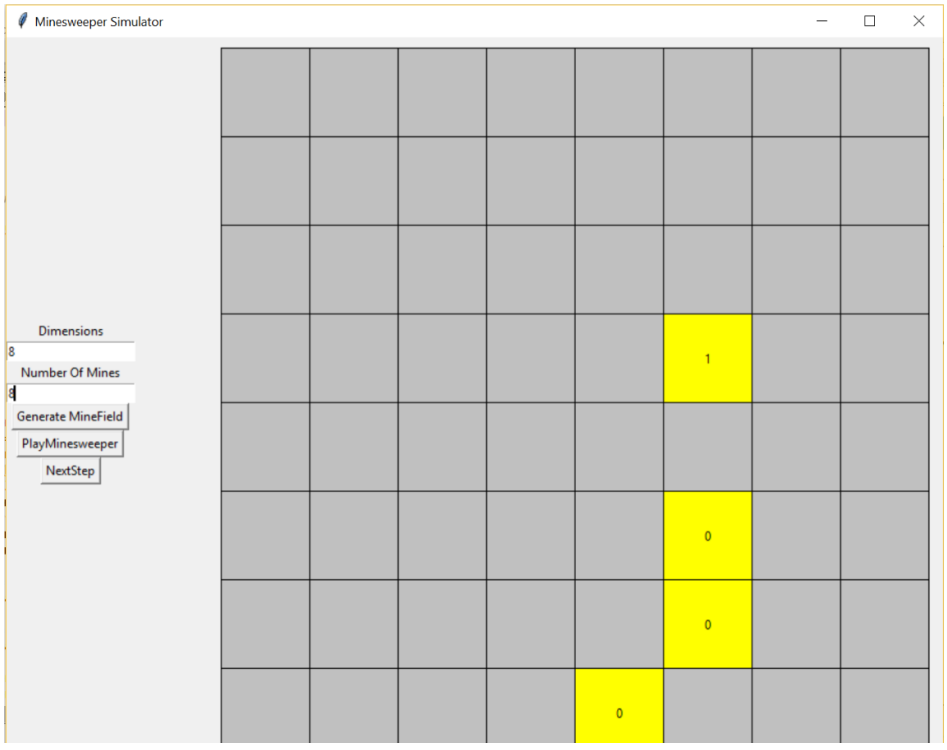
We have maintained one class attribute 'prob' which indicates the probability of cell having the mine. After we solve the constraints satisfaction problem in 'updateKnowledgeBase' function we get the multiple models through the 'satisfiable' function which returns the multiple model for the given CNF. If in each model a particular variable 'Vij' is true, then we have marked that Box (i,j) as Mine. For each box that is in fringe, we are calculating the ratio of number of true values to a particular symbol 'Vij' over all the models to the total number of models. If this ratio is 1 then we are marking that Box (i,j) as mine. If not 1, we store that value in 'prob' value of the Box(i,j).

While selecting the next box to reveal we first sort the boxes on the fringe according to their 'prob' value and select the one with the lowest probability. The maximum probability that a selected box can have is 0.2. We determined this value experimentally and it showed an improvement in the final score. If there are no boxes satisfying the condition, a random unsolved box is selected to query it.

• *Performance: For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?*

For this we chose board dimension to be 8x8 and 8 mines and noticed step by step progress of the agent.


Minesweeper Simulator
Dimensions
8
Number Of Mines
8
Generate MineField
PlayMinesweeper
NextStep
1
0
0


Minesweeper Simulator
Dimensions
8
Number Of Mines
8
Generate MineField
PlayMinesweeper
NextStep
1
0
0
0

At this step I would choose the boxes surrounding the box with clue 0 (cells around A74), but the agent chose some other box(A55) this is because of the way we designed our fringe, for every box that the agent is visiting, instead of fringe being just the neighboring boxes we are appending the new neighbors to the previous fringe and using this as our new fringe. And the next move is according to the method explained under Decision.
The reason for us to do this because we dint want to miss out on the case when the clue is 0: meaning all boxes around it are safe, If we choose one of its neighboring box, the fringe will change, the agent will not have a chance to select the other 3 possible cells from the previous fringe.

F1	F1	F1	
F1	0	F1	
F1	F1	F1	

F1	F2	F2	F2
F1	V	3	F2
F1	F2	F2	F2

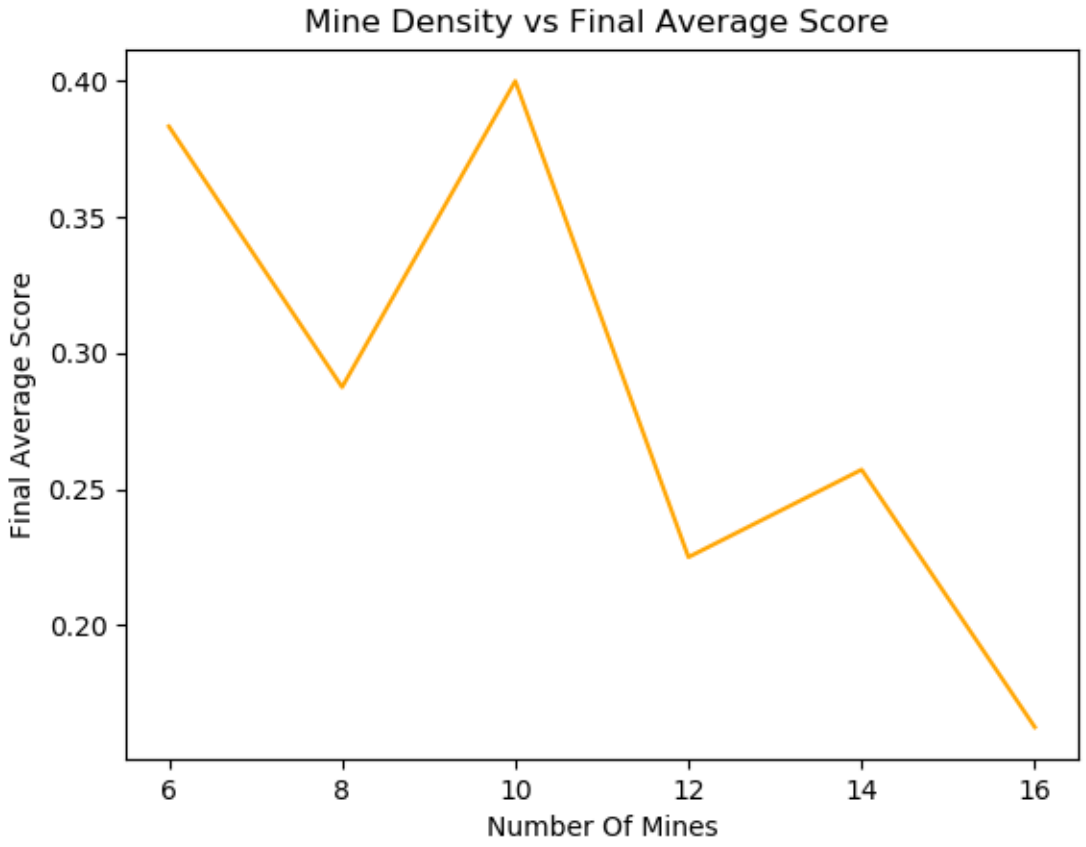
F	F	F	F
F	V	3	F
F	F	F	F

Illustration to show how the fringe is considered

• *Performance: For a fixed, reasonable size of board, plot as a function of mine density the average final score (safely identified mines / total mines). This will require solving multiple random boards at a given density of mines to get good average score results. Does the graph make sense / agree with your intuition? When does minesweeper become 'hard'?*

For this we chose the board dimension to be 8x8 solved the board for different mine densities.

The Final average score is obtained by taking mean of 10 random 8x8 boards



The graph becomes very hard for 12 and more mines in 8x8 board.

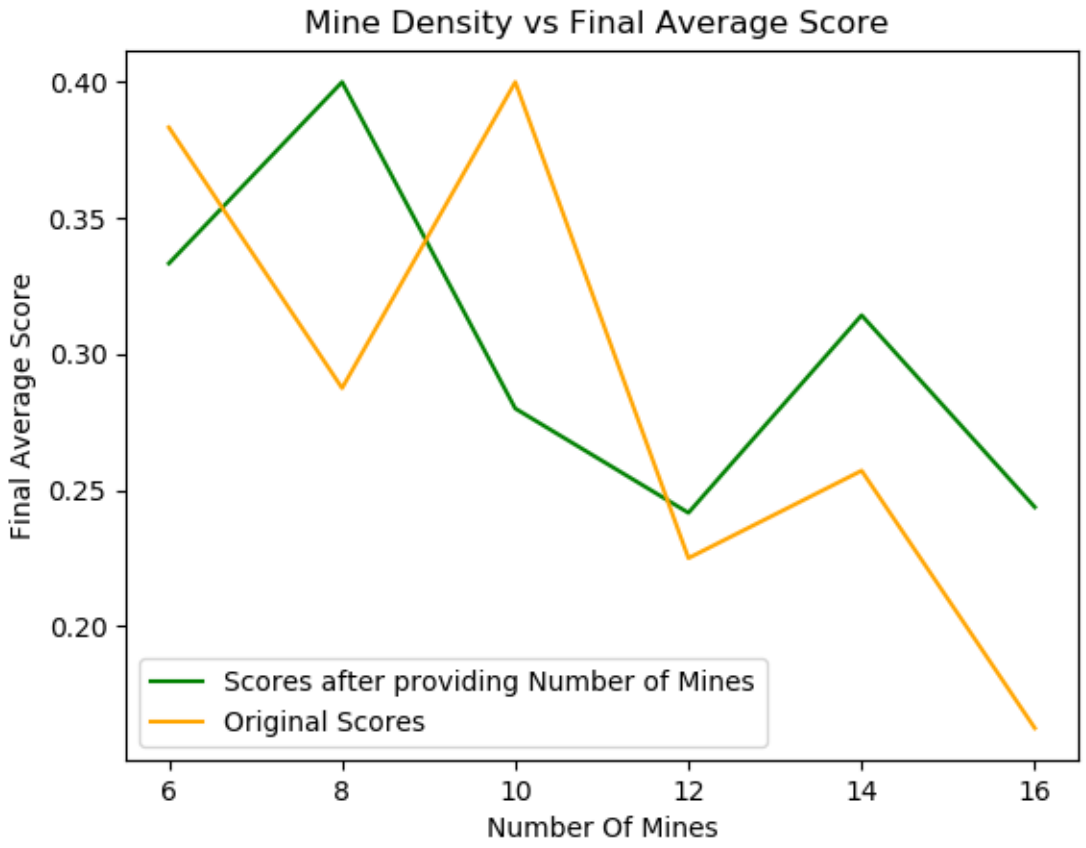
• *Efficiency: What are some of the space or time constraints you run into in implementing this program? Are these problem specific constraints, or implementation specific constraints? In the case of implementation constraints, what could you improve on?*

The idea with which are solving the minesweeper is to convert it into Constraint Satisfaction Problem, and to achieve this every clue is converted to its equivalent CNF.
Therefore, the complexity for solving this CSP is as follows
Time complexity: $O(2^n)$
Space complexity: O(n)
Where n is the number of variables in the knowledge base
This is problem specific constraint.

• *Improvements: Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action? How can you use this information to effectively improve the*

performance of your program, particularly in terms of the number of mines it can effectively solve? Re-generate the plot of mine density vs expected final score, when utilizing this extra information.

In the case where the agent knows the total number of mines in the minefield, the 'prob' for each box is initially set to $\frac{number\ of\ mines}{total\ number\ of\ boxes}$ i.e. the probability that any box will have a mine.
The probability changes as the agent uncovers the cells based on the number of mines identified and number of mines remaining.



Minesweeper Agent with Overestimated Clues

Here, we have implemented the solution for overestimated clue. The environment will return a clue that is overestimated, meaning that the clue may be larger than the true value. We have created one class MineSweeperOverestimatedAgent which inherits from the standard Agent class. The only difference to the standard agent lies in addToKnowledgeBase function that convert the clue of a cell into a logic expression.

The expression can be described as,

$$\mathrm{L}(\mathrm{clue}) \equiv \vee_{c=0}^{\mathrm{clue}} \mathrm{P}(\mathrm{c})$$

where $L(clue)$ means that there are at least clue number of mines in neighbors, and $P(c)$ as there are exactly c mines in neighbors.

Or alternatively,

$$\mathrm{L}(\mathrm{clue}) \equiv \neg(\neg \mathrm{L}(\mathrm{clue})) \equiv \neg \vee_{c=\mathrm{clue}+1}^{8} \mathrm{P}(\mathrm{c}) \equiv \wedge_{c=\mathrm{clue}+1}^{8} \neg \mathrm{P}(\mathrm{c})$$

By providing this information to the agent we can solve harder mazes