” Minesweeper playing AI Agent”

Name: Muhammad Rizwan Khan (netid: mrk150)

1 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 board is generated with default size of 10x10 dimension with mine density of 20 mines scattered on the board. As this project is intended to use inference rules, every possible information and knowledge is used by the agent in order to learn and try to solve the board. For this purpose, the agent makes use of equation forming and for this we treat the equations as constrain satisfaction problem equations (which will be referred as csp equations) as we treat this problem as a constraint satisfaction problem. These 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

```
[[0 0 1 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

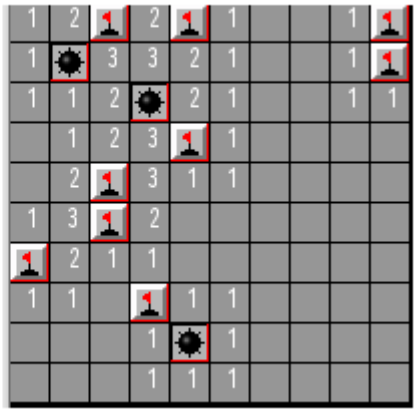


Figure 1: Minesweeper GUI

The Image on the right in Figure 1 shows a generated board that is solved

by our agent, where the cell with a flag indicates a flagged cell, cell with a mine indicates as wrong move from our agent, as it was open because not enough information was available to deduce anything, or it was opened randomly.The numbers on the cells indicate clues which tells us number of mines present in its adjacent neighboring cells. The image on the left are mines present on the board, and is an image taken from terminal (Note: due to some GUI framework issues we had to use the terminal to show locations of the mine).

2 Inference

Our agent keeps information of these factors below when solving the board

- 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 un-visited
- Every csp equation formed and stored in the knowledge base
- Duplicate csp equations that already exists and equations that might have a different value and that can play a role in determining whether a cell is a mine or not

This project solves the minesweeper board using two algorithms

  - Basic Algorithm: Follows the description present in the Project outline and makes comparisons based decisions based on just single cell at a time rather than multiple cells or multiple clues, and does not make use of inference rules or the information stored in knowledge base other than the one that is mentioned in the description.
  - Improved Algorithm: solves the board by considering multiple clues, solving equations and forming new csp equations to deduce new clues for unrevealed cells.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 equations for variables are formed:

When a non-mine variable is opened, an equation is formed that includes adjacent neighboring cell of the current cell, 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.

```
[[7, 8], [0]],
[[9, 7], [1]],
[[8, 7], [1]],
[[8, 7], [1]],
[[8, 7], [9, 7],[1]],
[[7, 7], [1]],
[[7, 7], [1]],
[[7, 7], [0]],
[[7, 7], [8, 7], [9, 7], [1]],
[[7, 7], [8, 7], [0]],
[[9, 7], [0]], [[9, 7], [0]],
[[6, 9], [1]], [[6, 8], [1]],
[[6, 8], [6, 9], [1]],
[[6, 8], [1]]
```

Figure 2: Equation in Knowledge Base

The image above shows how those equations are formed and stored. Equations with value of 1 on the right end of each equations indicate they are mines, but also some equations are being repeated with both 0 and 1 values – as we continue to find new clues and form and solve new equations, we are left with information that does not give us any new clues. This usually results in either random decision making, wrong decisions or complete ambiguity which leads to more computation in trying to deduce new clues from the information we have. 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 our algorithm first thinks about flagging a cell, it stores it in a separate list and proceeds to gain new clues. After another step is taken, it goes back to the list where our cell is located to be flagged, tries to determine if it's a mine (by going through knowledge base, scanning results of csp equation where this cell exists) and if the correct information is received by it then flags it, and if not then it leaves it in the list to test it again later (pushes it down the order).

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 cells on the board to store information corresponding to it e.g A could be [0,0], B could be [0,1] and so on) . Each equation is solved in a very systematic way. If the equation is of the length 3, then two options are available to try. One, If enough information is present then we can either simply apply the value of variables stored in our knowledge base to the equation at hand, or second option where it uses partial information present in the knowledge base and use that to try solving it by dividing the equations in to a subsets of two.

If both of these methods are not applicable then the algorithm tries to determine new information by simply counting the total number of variables in the equation and compares it to the value of the equation. If both are equal then it can use that to determine if a cell is safe or not. 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 place it in a list where we will come back to it later when we we think we have more information for it.

If we think we might have some information then 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 information is present, then we first verify if that specific equation satisfies as a subset for our equation in hand. Lets say, our knowledge base has equations of two common variables , 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, place value of A+B in A+B+C we get 2 + C = 3. We determine that if we consider C as a 1 then it would satisfy the constraint. We tag C as 1 and that way we update C in our knowledge base (you can also say that C equals 1 because 3 – 2 equals 1) The same can be done with B+C by using the same logic. We determine that C = 1 .

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.

When we are unable to determine if a cell is safe or not then our decision is either random, 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 . In Figure 3, the image below you can notice that our agent managed to flag all mines, but also ended up flagging safe cells

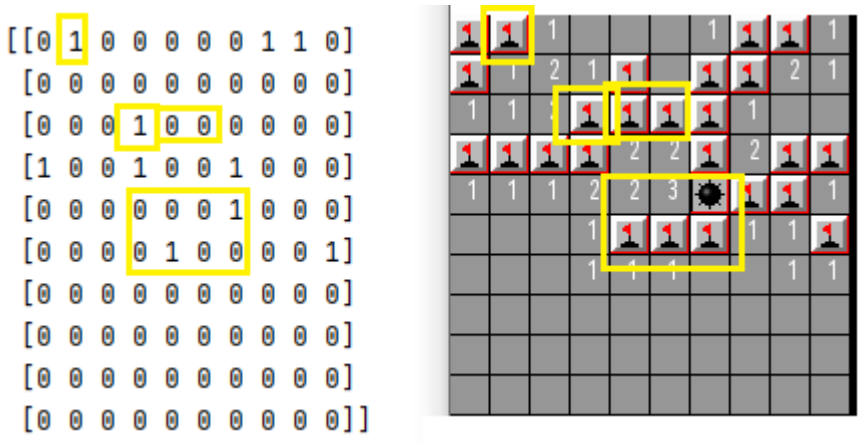


Figure 3: Decision by the agent and difference

as well. This is the result it produced due to ambiguity where it made decisions of flagging cell from the new information it was gathering (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 this case I did not implement that bonus feature)

3 Decision

Given the current state of the board environment and the state of our knowledge base about the environment – After the first click to start the movement of the agent, our agent would first calculate and form equations of its unrevealed neighbor for the first move. Then store a list of its current neighbor, and check if any of those cells can be determined to be safe from the information it just stored. If all cells are marked as uncertain (it should be noted they are not yet flagged or opened) then it would randomly open any of the cell in the next move. If not, it would select the neighbor from the list and try to determine if it is safe. That way the agent starts its movement from the first click. If a cell happens to be a mine, it stores that information and then selects a new cell from the list (our agent is storing each neighboring cell indexes in a list to visit after forming and deducing new clues and storing them in the knowledge base to use).

Our agent makes decision to open or flag a cell based on these steps

- First check if our knowledge base has any kind of information stored in it that would indicate if cell is a mine or safe, in this case it searches in the knowledge base for a single variable equation (e.g A=0 or A=1) which contains current cell information and then determine from there if it is

safe to open it or flag it.

- If no information is found, it then it would try to solve it through subset solver. In this, it searches the knowledge base for any equation that includes the current cell and try to break those equations down using subset solver

- Random – if no information is found then it will simply come back to it after few steps, and if it still cant determine about the same step then it will open the cell, but if due to any ambiguity it finds duplicate information exist in the knowledge base for some reason (you have A=1 and A=0 present in the knowledge in the numbers) then it would simply count the number of 1's and 0's it has and carry out a comparison on them. If it can be determined if 1's is greater than 0 then it will flag the cell and vice versa. if not then it simply opens the cell.

4 Performance

Given the current state of the board environment and the state of our knowledge base about the environment – After the first click to start the movement of the agent, our agent would first calculate and form equations of its un-revealed neighbor for the first move. Then store a list of its current neighbor, and check if any of those cells can be determined to be safe from the information it just stored. If all cells are marked as danger (it should be noted they are not yet flagged or opened) then it would randomly open any of them and in the next move. If not, it would select the neighbor from the list and determine if it is safe. That way the agent starts its movement. If a cell happens to be a mine, it stores that information and then selects a new cell from the list (our agent is storing each neighboring cell indexes in a list to visit after forming and deducing new clues and storing them in the knowledge base to use).

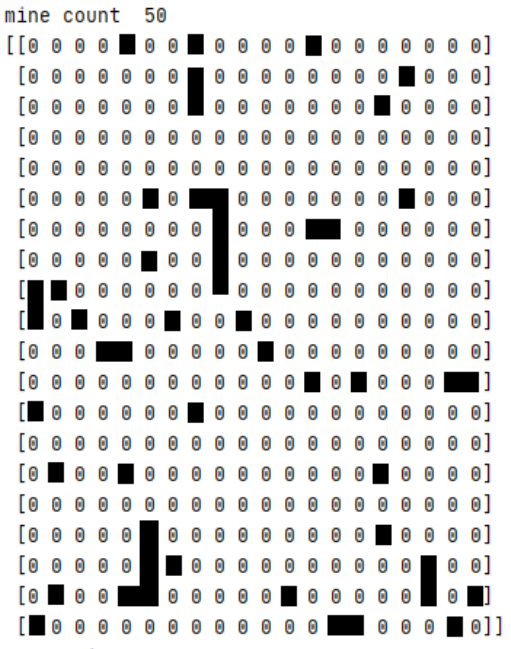


Figure 4: Mine blocks on the board represented in terminal

Figure 4 is taken from the terminal and the black box's in Figure 4 indicate location of the mines on the board and 0 as clear cell (Note : Due to some problem in my GUI framework I was not able to two graphical user canvas to generate graphics of both boards)

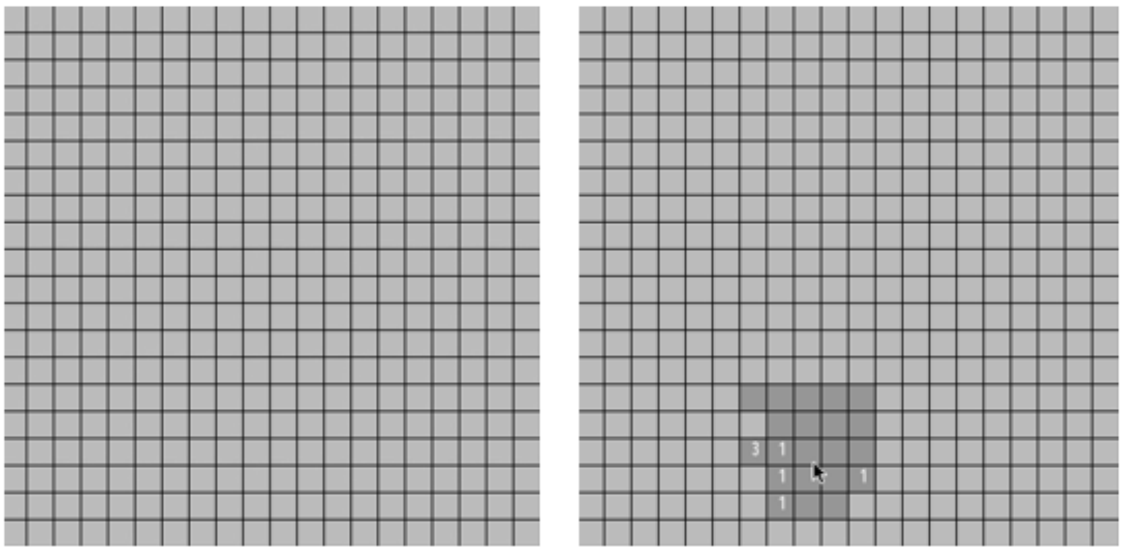


Image 1 Image 2

Figure 5: Beginning phase

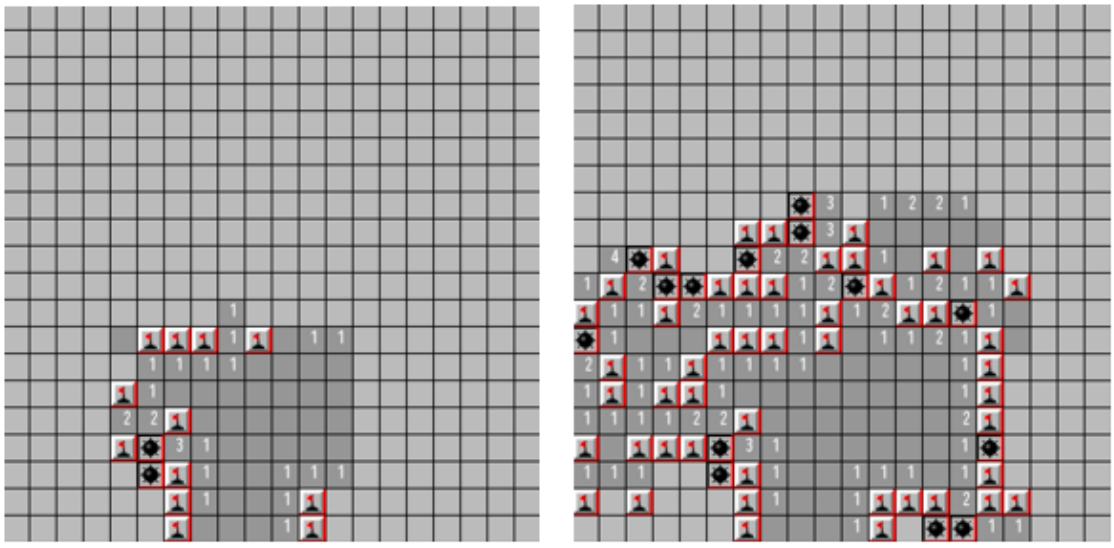


Figure 6: Middle phase

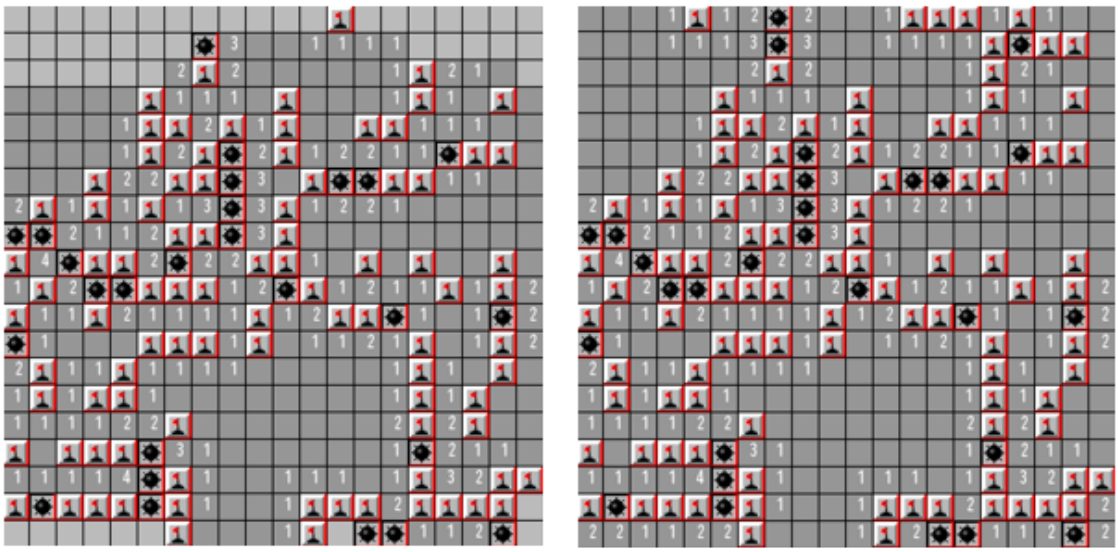
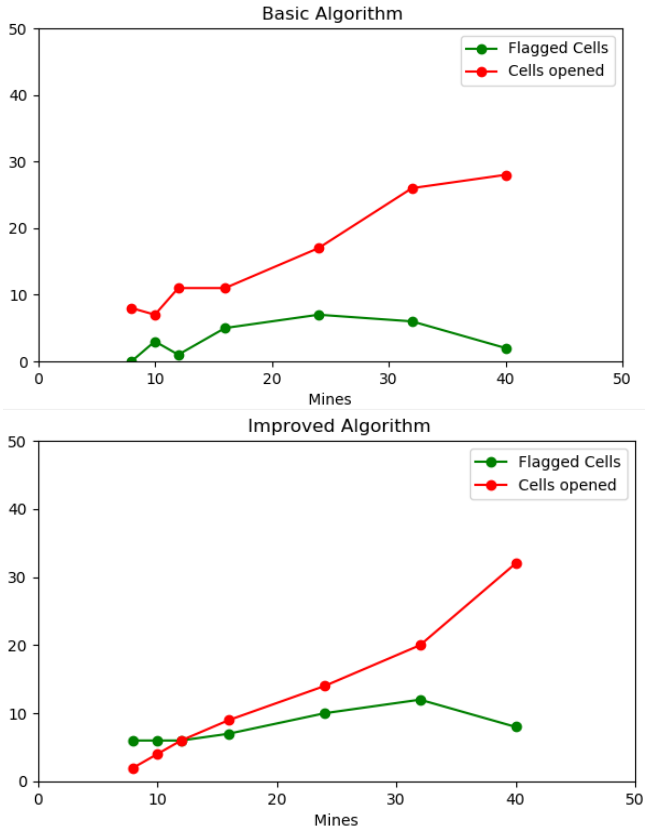


Figure 7: End phase

In the images above, 50 mines exists on a 20 by 20 board, our agent was able to successfully identify most of the cells that were actually mines and flag them. Around 20 cells were decided to be opened by the agent and they happened to be mines. These were the wrong decisions made by our agent that lead to it opening cells with mines in them. One thing that is very noticeable is, that most mines that are revealed are opened when the agent is on the edge of the hidden cells and no clues can be concluded to determine anything about unknown and un-revealed cells on the other side of the board, so the agent has to deduce all clues from whatever information it has or try to randomly open the cells. If it was given the chance to find a cell by searching from both ends then it would have done better job in determining it. One more important factor to take into account is the number of flagged cells. Not all the flagged cells are mines, and if the agent was given the ability to backtrack and go back and open flagged cells that it is sure to open the ones that are safe (Note: When all cells are visited or flagged, our agent goes back to the list of flagged cells and then tries to determine if a flagged cell is safe or not - all through information stored in our knowledge base).

5 Performance graphs



The above graph is a result of 8x8 board and results are determined from average scores from very low mine density to higher density. The x-line label 'mine' indicate the the number of mines present on the board in correspondence to mine density (Note: The graph shows low to high density of mines from left to right). Red line in improved algorithm graph indicates opened cells that happened to be mines, and it starts from 0 and gradually increase with increasing higher mine density. The green line indicates correctly flagged cells and as you can see, it correctly solves the board for low density and gets most to half of the flagged cells correct in Improved Algorithm. That is why you see green line in the graph constantly improve until it hits a higher density threshold. The main difference between simple and advanced algorithm is, improved algorithm is able to flag half or even more cells correct up from low to medium density of mines but as mine density gets in the 'higher' threshold it tends to not keep up and starts performing poorly then. It is safe to assume from the result that improved algorithm will correctly flag half the cells in low to medium density in any given situation (we are ignoring situations where it does correctly flag

all correct cells). In comparison of basic algorithm to the improved one , basic one it tends to perform very poorly in all aspects. It should be noted that Basic algorithm do flag cells, but its performance is worse than our improved algorithm

6 Efficiency

One of the main issues for this was the time constraint. Knowledge base used makes use of lists to store equations and those equations themselves are nested lists containing variables and values and equations. Whenever we wanted to use information or deduce anything new, we had to traverse through the knowledge base looking for the information. In this case, getting values for single variable equations to use in other equations or finding equations for subsets, we would need to go through complete knowledge base list. That same issue was also common when we needed to update or remove variables and equations from the knowledge base. One improvement that could be made on this would be the use of dictionary in comparison to use of lists, and applying indexes, string codes or character keys to find our variables and equations by tying similar subsets, equations or cell positions to their vicinity and using those indexes or keys to quickly access the information rather than traversing through a complete list.

7 References

https://www.python-course.eu/tkinter