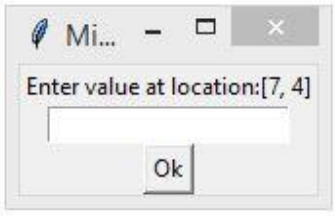
CS 520
Assignment 2
Minesweeper

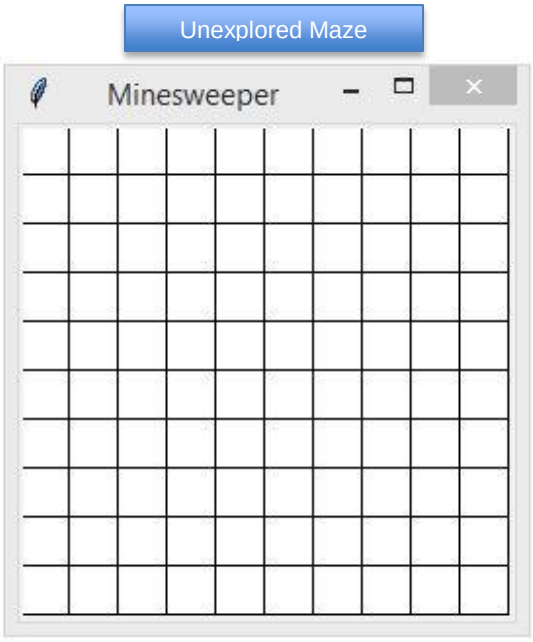
October 28, 2018

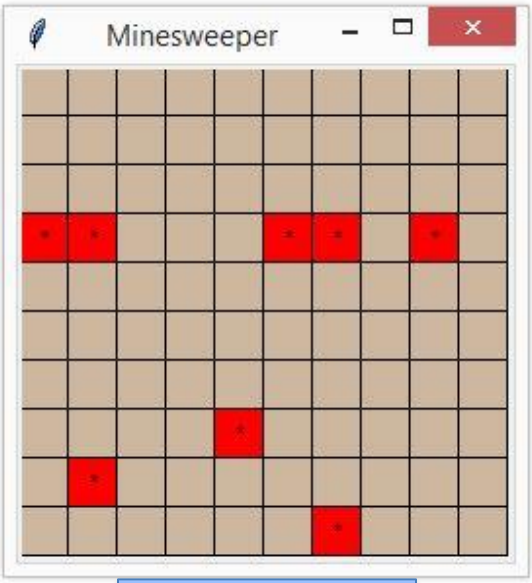
Adhish Shrivastava, Hiral Nagda, Yash Nisar

1. Representation:
   We represented the board as a 2D array of dimensions row*col which were selected by the user. We indicate states of each cell using different values:
   1. Unexplored cells are represented by 9.
   2. Explored cells which are not mines are represented by a value (0-8) which is equivalent to the number of mines in its surrounding cells.
   3. Cells which contain mines are represented by the value -2. When the AI explores a cell having -2, the game ends.

   We represented the knowledge as follows:
   - a list of mine cells
   - a list containing explored cells
   - a matrix which stores the probability of each cell being a mine cell (Initial cell probability being Z).







2. Inference:
   The game runs until either all the safe cells have been explored or the solver explores a mine cell. When the solver explores a cell, the following cases can be encountered:

   Case 1:
   If the cell explored has value 0, indicating none of its neighbors are mines. Hence we explore all the neighbors recursively and continue until all the neighbors of explored cells with 0 value have been explored.

   Case 2:
   When the number of mines around the cell is equal to cell value, this implies that all the remaining neighbor cells are safe cells and thus marked safe and explored.

   Case 3:
   When the value of the cell is equal to the number of unexplored neighbors, all the unexplored neighboring cells are marked as mines.

   We also check if any unknown neighbors of 2 explored cells (mine or clear cell) have common neighbors by subtracting the values of the cell.

   For example:

A	B	C
2	3	

   A + B + C = 3
   A + B = 2

   Subtracting both the equations we get C = 1. This implies that cell is a mine.

   At the intersection of the neighborhood of two cells are considered and based on the difference between the values of the 2 cells under consideration cells are either cleared or declared as bombs

   Until the game ends, we encounter either of the cases mentioned above. If we encounter a state in which we cannot infer any new knowledge from the explored cells or from the knowledge base, we use a probabilistic approach. Initially, all the cells are given a probability 1 and it is updated when the above mentioned situation is encountered. The new probability is computed by

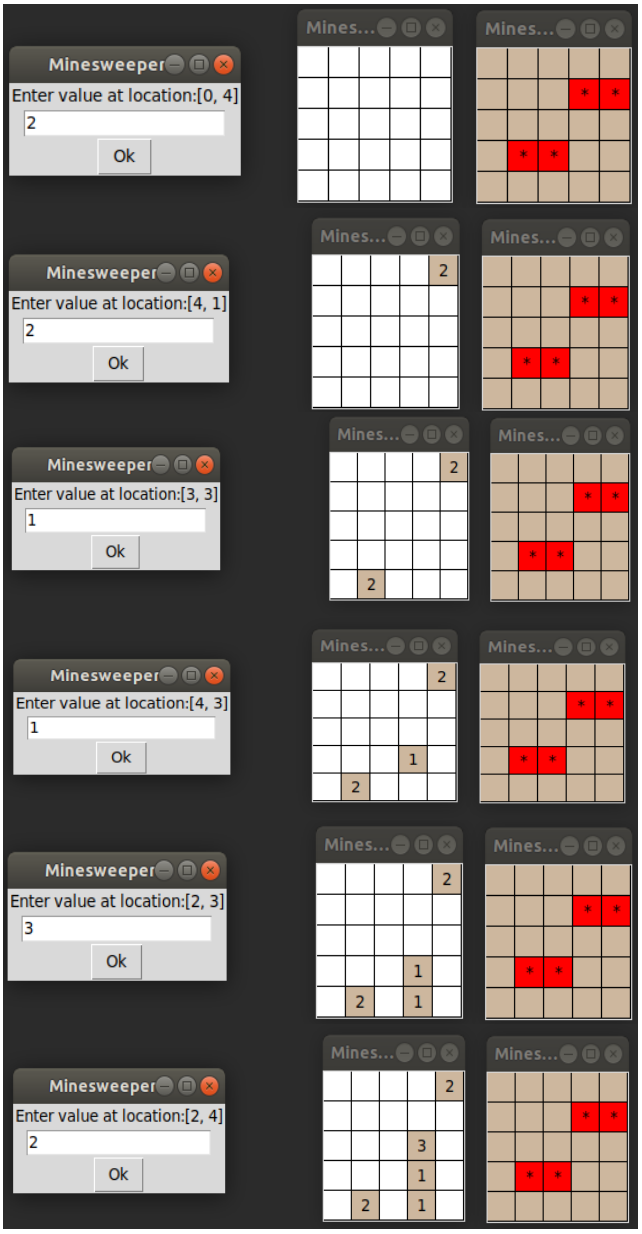
$$\frac{cell\ value - number\ of\ known\ mines}{number\ of\ unexplored\ neighoring\ cells}$$

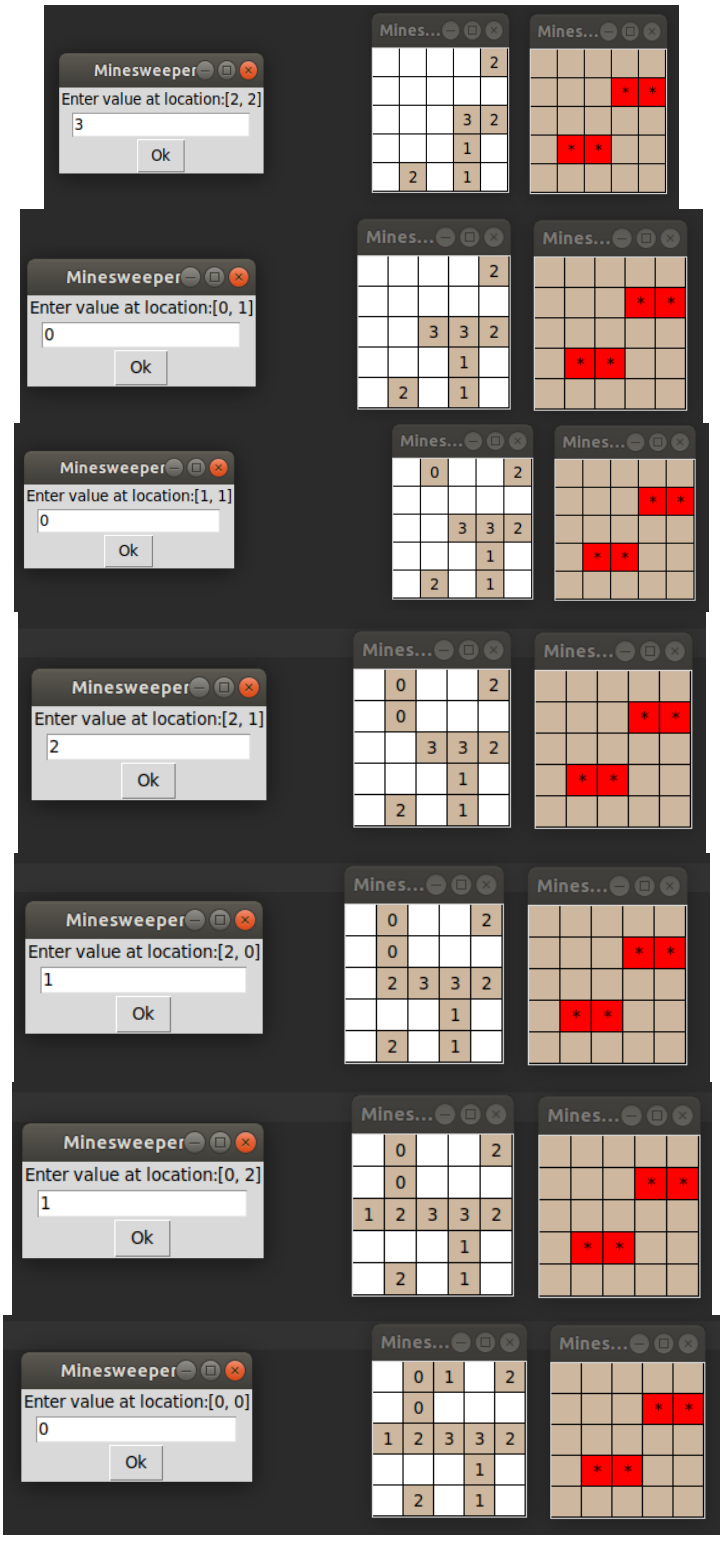
   The cell which is already explored and is known to be safe, a probability of 2 is assigned. The cell which has the least chance of being a mine is taken and explored. This approach can

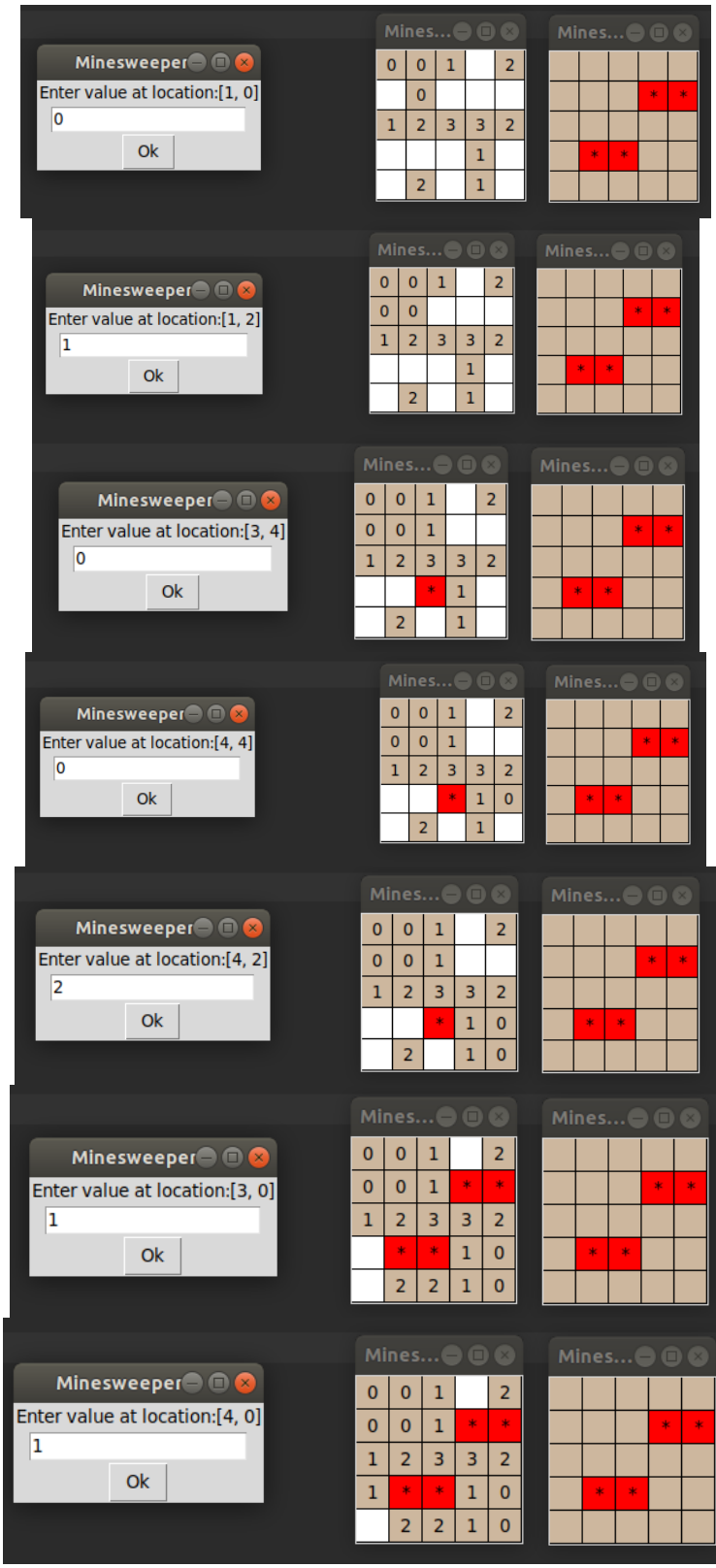
be improved to give a more accurate identification of the cell status if more constraints were taken into account. This is possible when more cells have been explored as it will eliminate cells thus show an effect on cell probability.

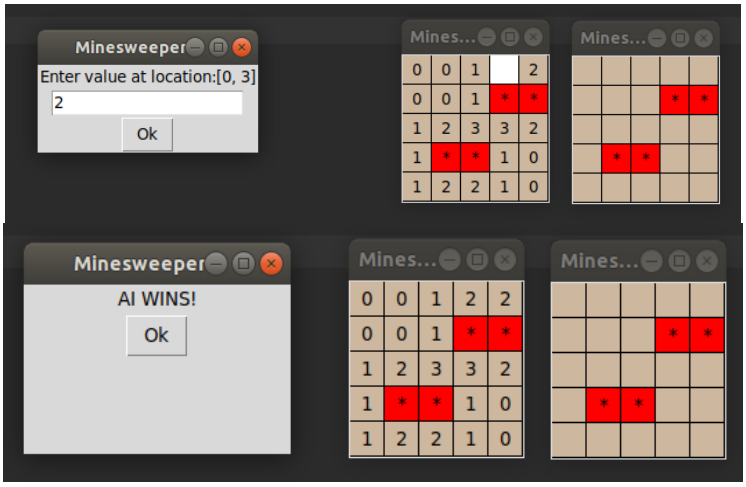
3. Decisions:
   Given the current state, cells are checked if they are mines or safe cells. If there are no cells left to explore, a cell with the least probability of having a mine is chosen to search next. Here we are taking a risk that is there might be a possibility of multiple cells having the least probability. In this case we select at random from these cells, without being sure that it cannot be a mine.

4. Performance:



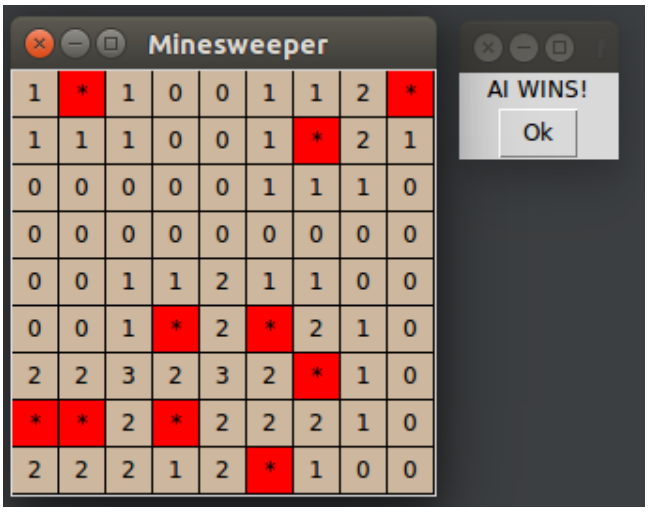

Minesweeper
Enter value at location:[2, 2]
3
Ok
Mines...
Minesweeper
Enter value at location:[0, 1]
0
Ok
Mines...
Minesweeper
Enter value at location:[1, 1]
0
Ok
Mines...
Minesweeper
Enter value at location:[2, 1]
2
Ok
Mines...
Minesweeper
Enter value at location:[2, 0]
1
Ok
Mines...
Minesweeper
Enter value at location:[0, 2]
1
Ok
Mines...
Minesweeper
Enter value at location:[0, 0]
0
Ok
Mines...

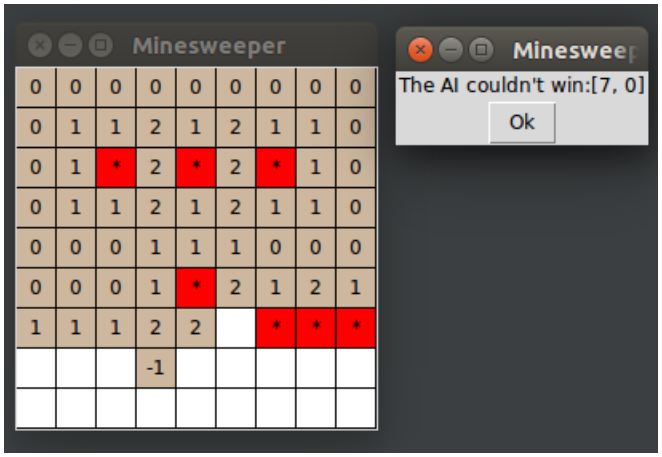

Minesweeper
Enter value at location:[1, 0]
0
Ok
Minesweeper
Enter value at location:[1, 2]
1
Ok
Minesweeper
Enter value at location:[3, 4]
0
Ok
Minesweeper
Enter value at location:[4, 4]
0
Ok
Minesweeper
Enter value at location:[4, 2]
2
Ok
Minesweeper
Enter value at location:[3, 0]
1
Ok
Minesweeper
Enter value at location:[4, 0]
1
Ok
Mines...
Mines...



Our program didn't make any decision that surprised us as the decisions based on logic and probability.

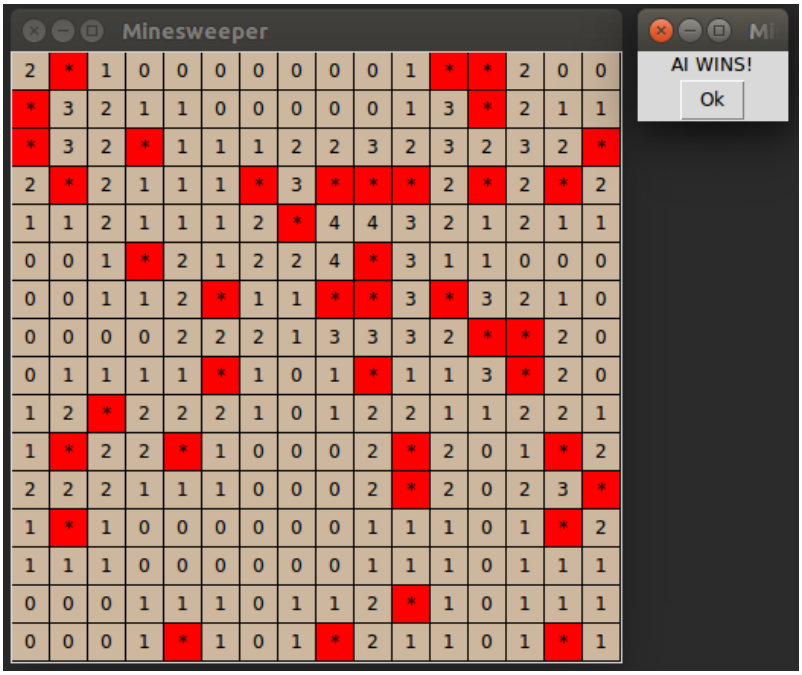
5. Performance:
   Beginner:
   Row = 9
   Column = 9
   Mines = 10

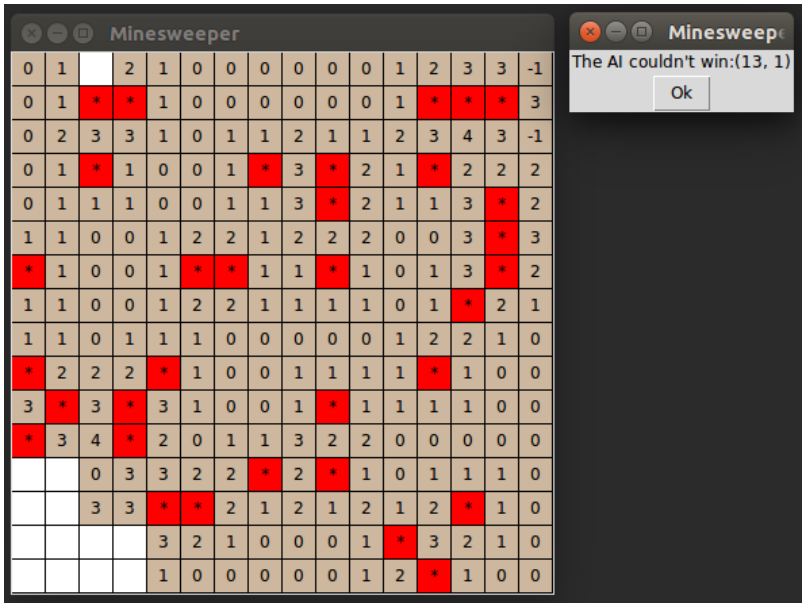




%times the AI won = 81%

Intermediate:
Row = 16
Column = 16
Mines = 40





%times the AI won = 21%

In Expert mode, i.e. 16*30 with 99 mines, the solver is forced to randomly guess in most of the situations. In cases where it needed to guess, the solver was not able to win the game.

6. Efficiency:
   Space Complexity: $O(n^2)$
   2D Array: Board of size row*col
   2D Array: Cell Probability
   Time Complexity: $O(n^3)$

   These are implementation specific constraints. If we perform targeted checking, complexities can be improved.

7. Improvements:
   Currently, we are not aware of the number of mines that are there on the board. If the number of mines is known, a counter can be maintained and this counter gets updated every time we mark a new cell to be a mine cell. The code can then keep a track as to all the mines have been discovered or not. If all the mines have been discovered then the solver has successfully completed the game and can exit out of the game. This can possibly reduce the time complexity as the program need not run the entire cycle even after all the mines have been identified.

Bonus: Chains of Influence

1. We can maintain a dictionary which maps a cell to its parent cell (parent cell is the cell from which the state of this cell is derived).

   Case 1:
   When a Cell X is revealed to have value 0. All the surrounding cells can be derived from X and can be marked as safe.

Case 2:
When the number of mines around the cell X is equal to cell value, all the neighboring cells are marked safe.

Case 3:
When the value of the cell X is equal to the number of unexplored neighbors, all the unexplored neighboring cells are marked as mines.

Map {Neighbor of X: X}

2. The solver usually takes a decision and explores a new cell only after looking up in the knowledge base to be sure that it is not a mine. This data is received when one of the neighbor cells confirm this. The chain of influence breaks when a situation is encountered when the solver can no longer explore a new cell based on the knowledge base and is forced to take a risk by using a probabilistic approach. This approach might make the solver encounter a mine.

3. The length of the chain directly has an effect on the efficiency of the solver. If the entire board is solve without breaking the chain or in case of the longest chain, the chances of winning is highest as the solver won't be forced to take a risk. If the length of the chain is reducing it means that the code is forced to take more decisions, thus increasing the chance of losing. Hence reducing efficiency.

4. A board will yield a long chain of influence when the mines are densely concentrated in groups. In this case, exploring a cell will help to know information about more cells and thus reducing the chances of taking risks. When there are a large number of mines and are sparsely placed, the chances of the chain to break are higher.

5. Theoretically, the chain of influence can travel the entire board. This depends on the position of the mines.

7. Yes, there are cases when the solver is forced to take risks as it can no longer continue with the present level of knowledge. Due to this, there is a chance that the solver hits a mine and losses the game.

Work Division:

Adhish Shrivastava: GUI, Analysis, Report
Hiral Nagda: Implementation (Explore function), Analysis, Report
Yash Nisar: Implementation, Analysis