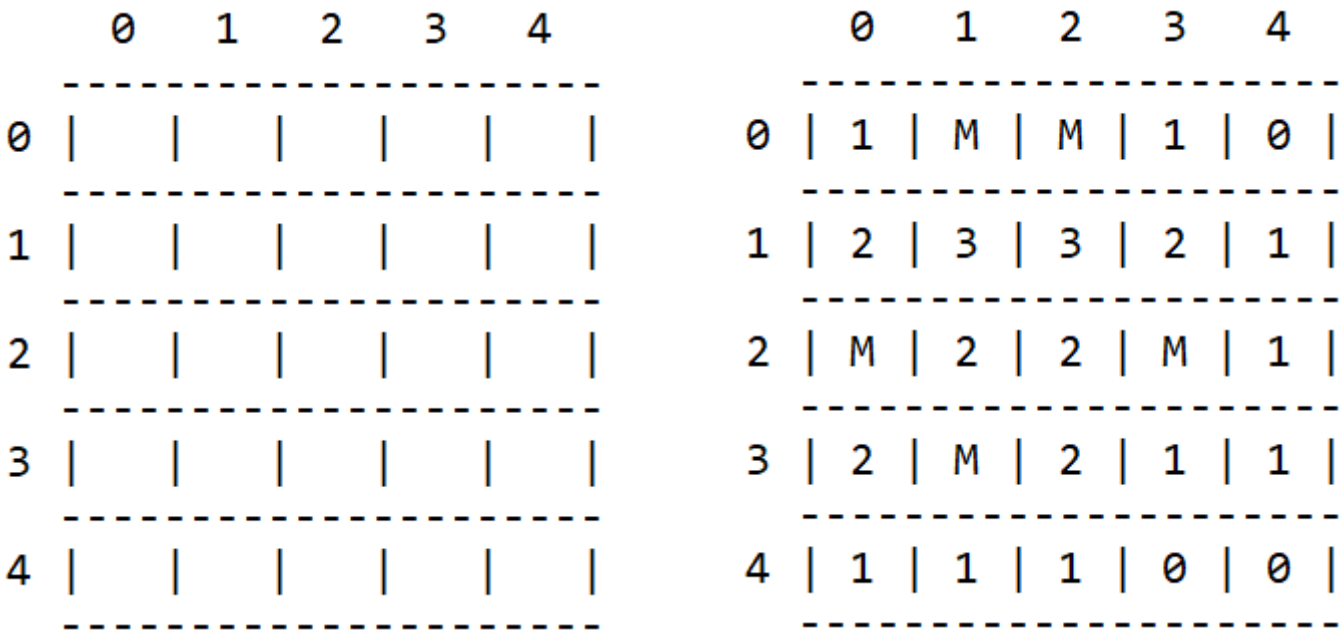
RUTGERS UNIVERSITY

MineSweeper AI

Prathamesh Kulkarni
October 31, 2020

1 INTRODUCTION

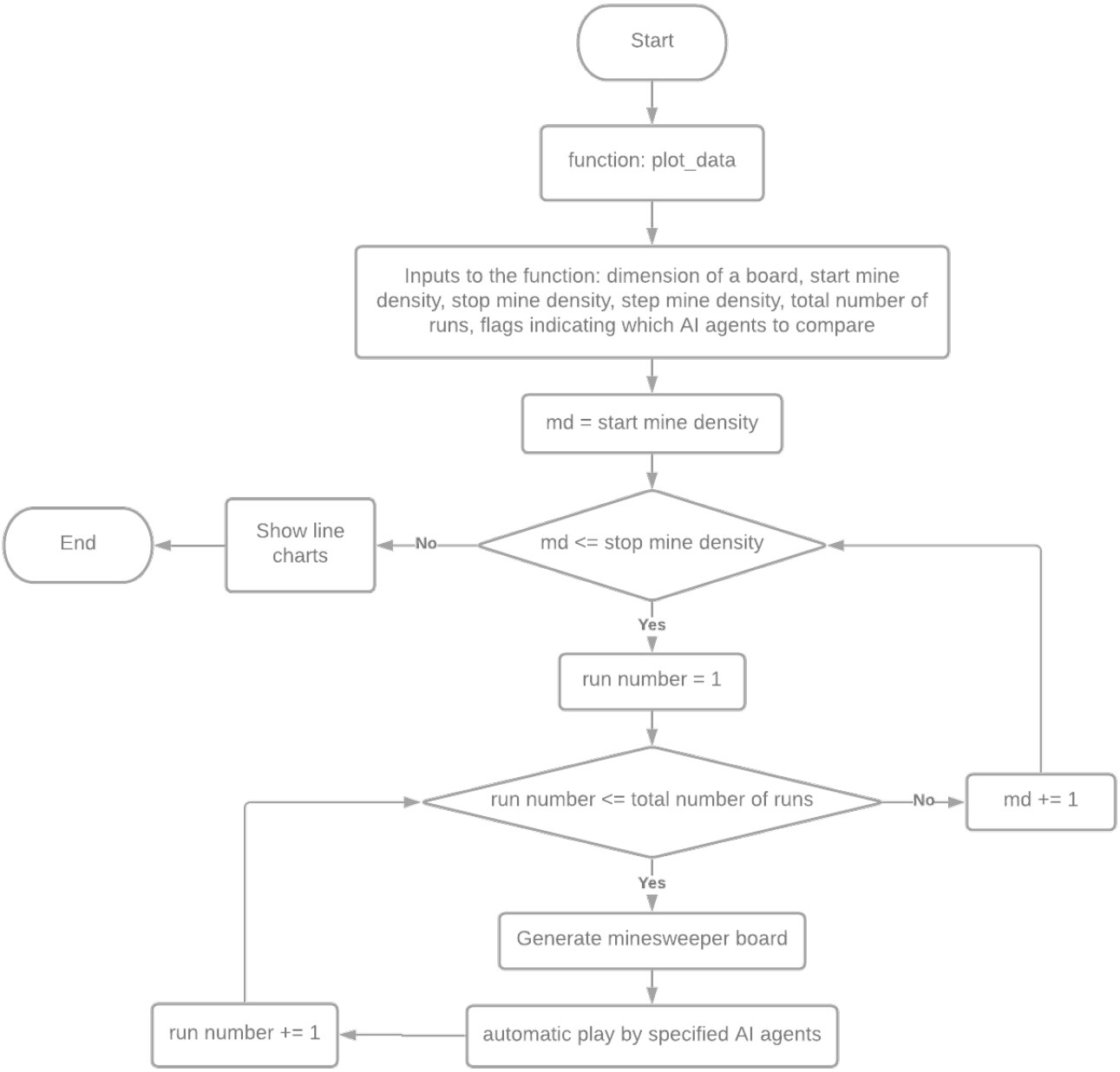
Board in a minesweeper is a square grid of cells where each cell is either safe or mine. Initially, every cell in a board is hidden. At every move, AI agent chooses to reveal a hidden cell. If a revealed cell is safe, then it shows count of neighboring mines as a clue. If revealed cell is a mine, then mine goes off and no clue is shown. In addition, agent can also flag a cell as mine based on inference. Agent continues in this way until every cell in a board is either flagged or revealed. There are three AI agents - basic agent, single improved agent and double improved agent - to play the Minesweeper. These agents differ in their inference capabilities. Final score of an agent is determined as the ratio of number of mines safely identified and total number of mines. Higher final score is an indication of better inference capabilities.



2 System Design

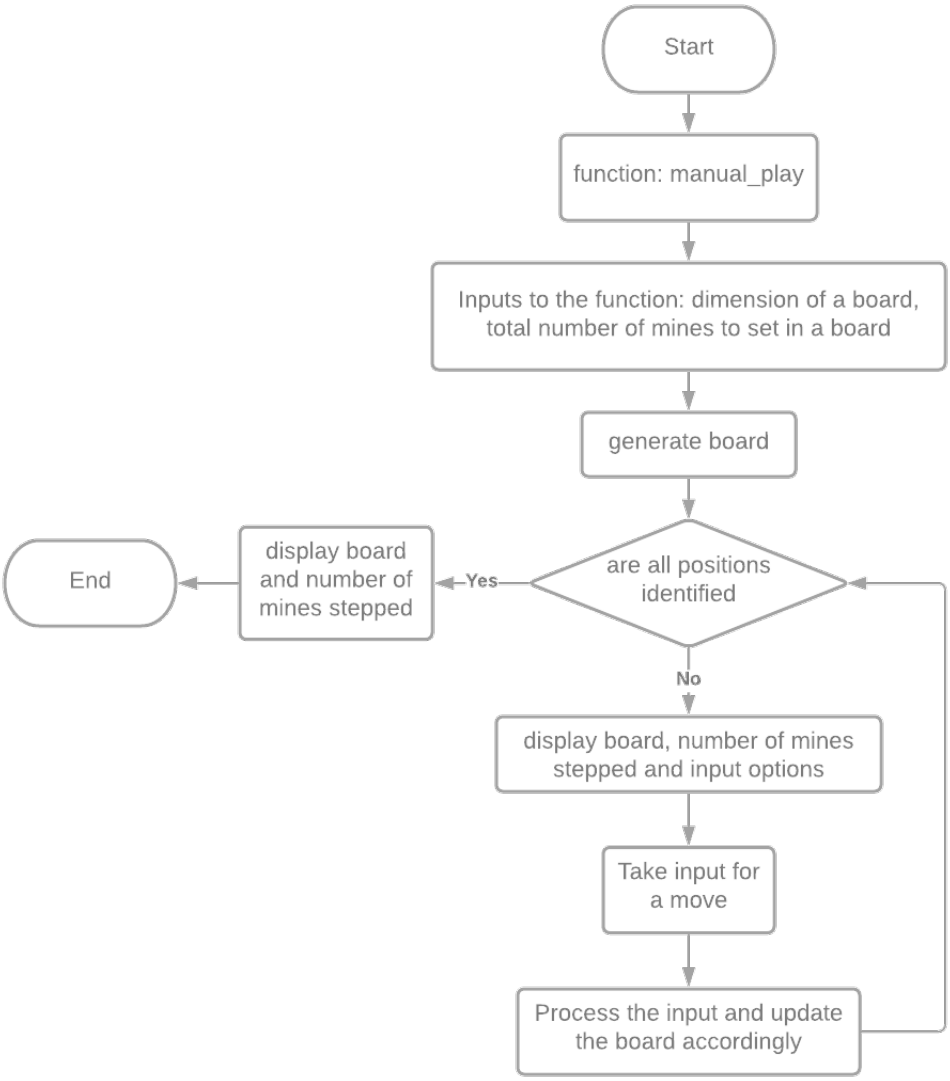
2.1 Automatic Player Architecture

Automatic player plays minesweeper game till the end without human intervention. It has 3 AI agents; basic agent, single improved agent and double improved agent; to automate the play. It computes average final scores of AI agents and displays comparison line charts with mine densities on x-axis and average final scores on y-axis. It takes following parameters as input: dimension of a board, start mine density, stop mine density, step mine density, total number of runs to compute average final score, indicator variable of basic agent, indicator variable of single improved agent, indicator variable of double improved agent. Indicator variables are used to specify the AI agents to be compared. For a single run, specified AI agents play the same minesweeper board to yield fairer comparison of average final scores. List of mine densities and lists of average final scores of specified AI agents are used to draw comparison line charts.



2.2 Manual Player Architecture

Manual player plays minesweeper game with human intervention. At every move, it displays current state of minesweeper board and asks for a move. It provides following options for a move: reveal position, flag position, move by basic agent, move by single improved agent, move by double improved agent, automate game from current state till the end. If move is to reveal or flag a position, it updates the input position in a board accordingly. If move is to be played by one of the three AI agents, it uses their corresponding inferences to reveal a position. After a move is played, it displays new state of the board and asks for a move again until all the positions in a board are identified (revealed or flagged). If move is to automate game-play till the end, then specified AI agent plays a move and displays intermediate state of a board until game ends.



2.3 Important Modules

1. Actual Board Generation
   Actual board is the true underlying minesweeper board that has all the information about locations of safe cells, mine cells and clues to show for the revealed safe cells. Actual board is stored as a 2-dimensional list in which each cell is a dictionary. Each cell stores it's status (0 = safe , 1 = mine), count of its mine neighbors and count of its total neighbors. Actual board is generated by function get_actual_board where every cell's status is safe and count of mine neighbors is 0. Given number of mines are set in actual board by a function set_mines. Once the mines are set, the count of mine neighbors of every cell is initialized by function initialize_cmn.

2. Visible Board Generation
   Visible board is the knowledge base of AI agent where it stores the information gained from querying the environment and inferences based on them. Visible board is stored as a 2-dimensional list in which each cell is a dictionary. Each cell stores its status (0 = safe , 1 = mine , ' ' = hidden), count of its mine neighbors (None if cell is hidden), count of its total neighbors, count of its identified safe neighbors, count of its identified mine neighbors and count of hidden neighbors. Visible board is generated by function get_visible_board where every cell's attributes are set as follows: status = ' ', count of mine neighbors = None, count of total neighbors = (3 or 5 or 8 depending upon the position of cell in a board), count of identified safe neighbors = 0, count of identified mine neighbors = 0, count of hidden neighbors = count of total neighbors.

3. Basic Agent
   Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. It keeps on playing a game until all the cells in a visible board are identified. It uses two basic rules to make inference about the hidden cells. If a cell is inferred to be a mine, it updates that position in a visible board (knowledge base) and removes same position from a list of unidentified cells. If a cell is inferred to be safe, it adds that cell to a set of inferred safe cells. If a set of inferred safe cells is not empty, it chooses to reveal a position from that set. If a set of inferred safe cells is empty, it makes inference before making a move. If inference does not conclude safety of any hidden cell, then it chooses to reveal a hidden cell at random.

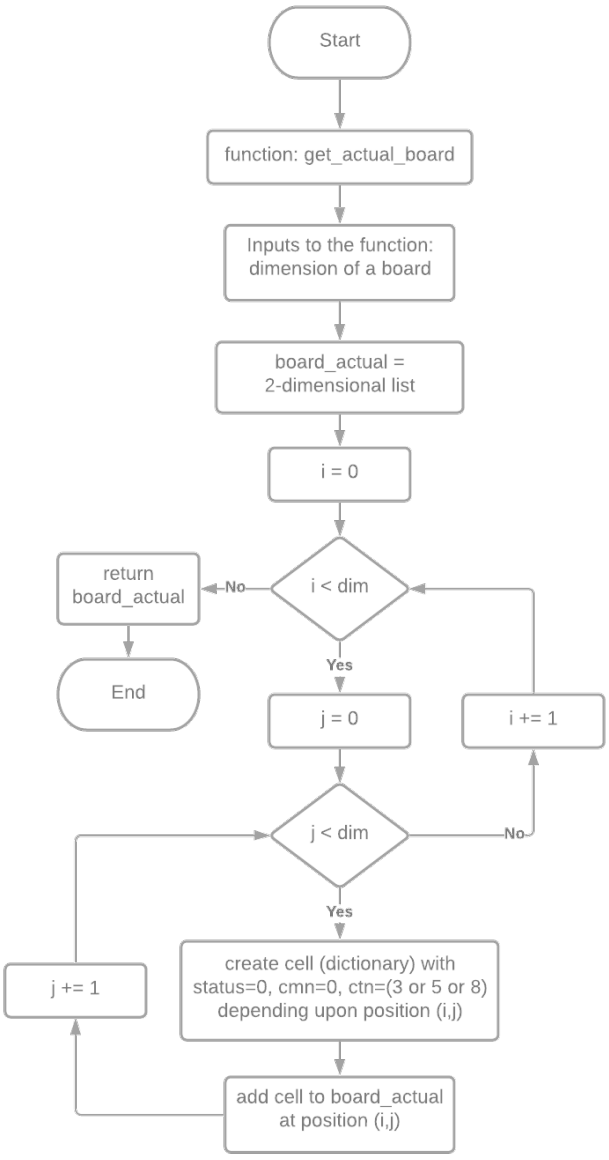
4. Single Improved Agent
   Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. It keeps on playing a game until all the cells in a visible board are identified. It uses constraint satisfaction based inference in addition to the two rules of basic agent. After the inference, it updates visible board, list of unidentified cells and a set of inferred safe cells in a same way as that of basic agent. If a set of inferred safe cells is not empty, it chooses to reveal a position from that set. If a set of inferred safe cells is empty, it first tries basic agent's inference and goes for constraint satisfaction based inference only if basic agent's inference does not conclude safety of any hidden cell. If both inferences do not conclude safety of any hidden cell, then it chooses to reveal a hidden cell at random.

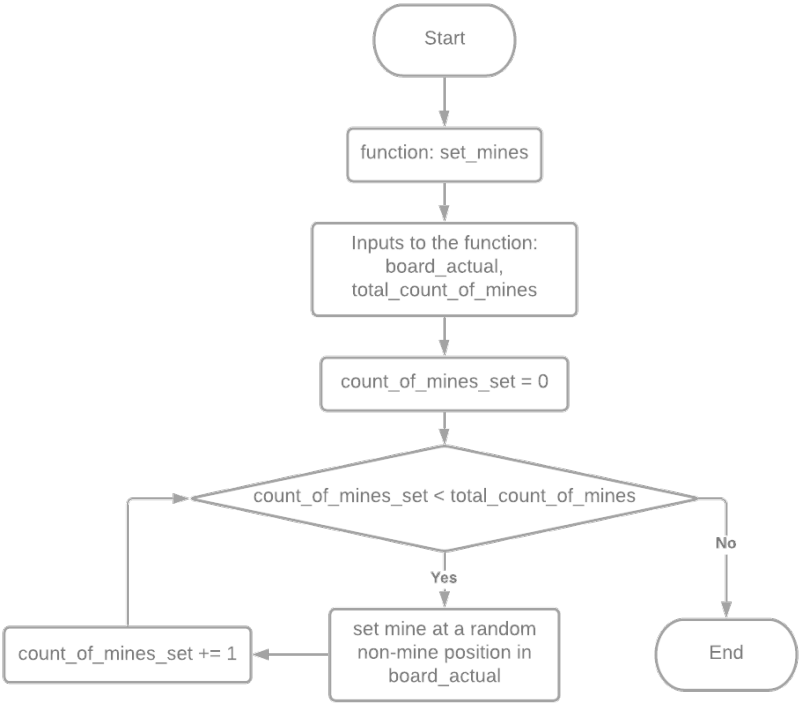
5. Double Improved Agent
   Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. It keeps on playing a game until all the cells in a visible board are identified. It uses probabilistic inference in addition to inferences used by single improved agent. It proceeds like single improved agent until constraint satisfaction based inference. If CSP based inference does not conclude safety of any hidden cell, then it performs probabilistic inference to make a move. It chooses to reveal a position that has least probability of being a mine.

3 Detailed Design

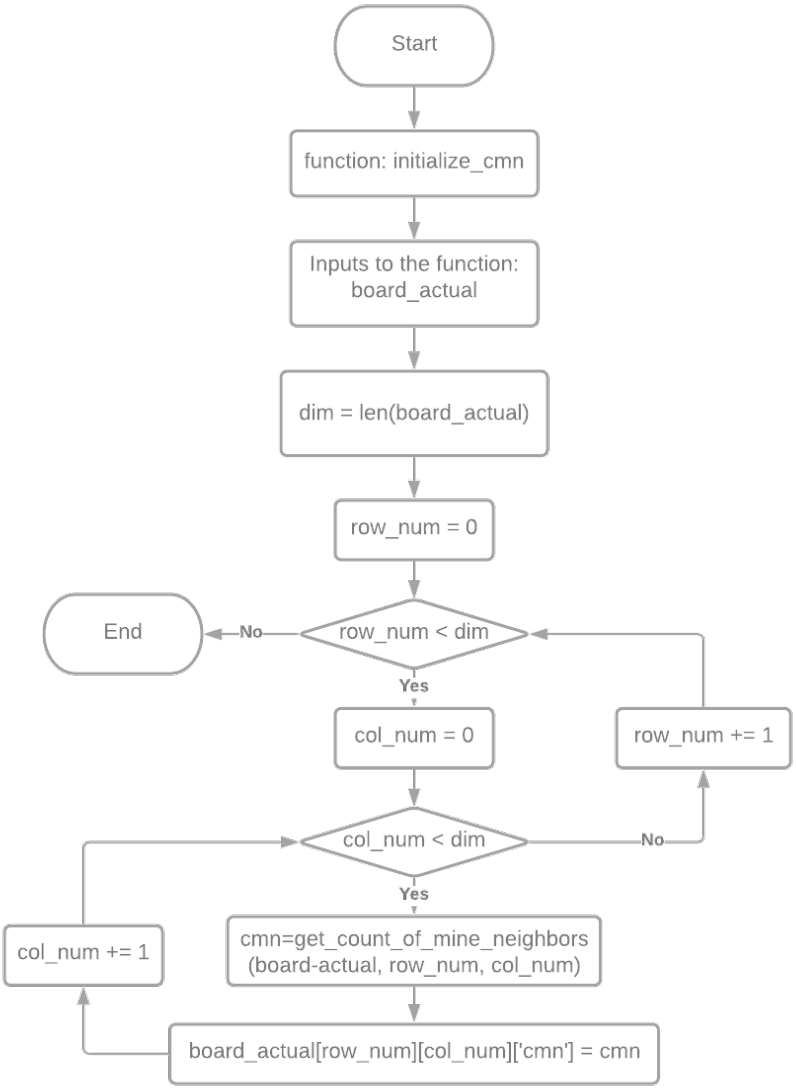
3.1 Actual Board Generation

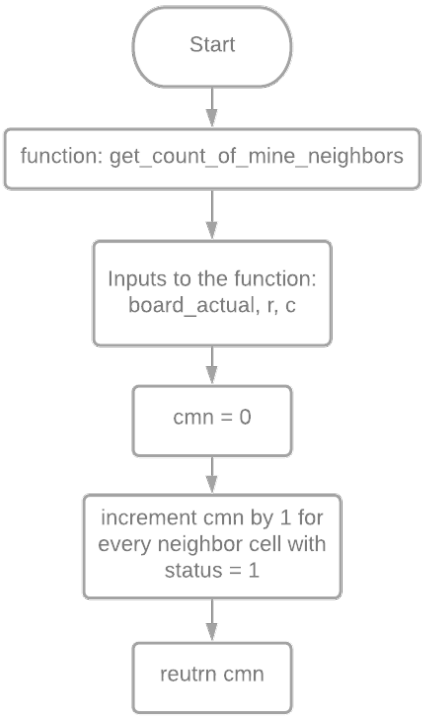
Actual board is the true underlying minesweeper board that has all the information about every cell. Actual board is stored as a 2-dimensional list in which each cell is a dictionary. Dictionary contains following keys - status, cmn (count of mine neighbors), ctn (count of total neighbors). Actual board is generated by function get_actual_board where status of every cell is 0 (safe), cmn of every cell is 0 and ctn is constant. ctn is set to following values: 3 (corner cells), 5 (edge but non-corner cells), 8 (non-edge cells). Given number of mines are set in board_actual by a function set_mines. It keeps setting mine at a random non-mine position in board_actual until given number of mines are set.





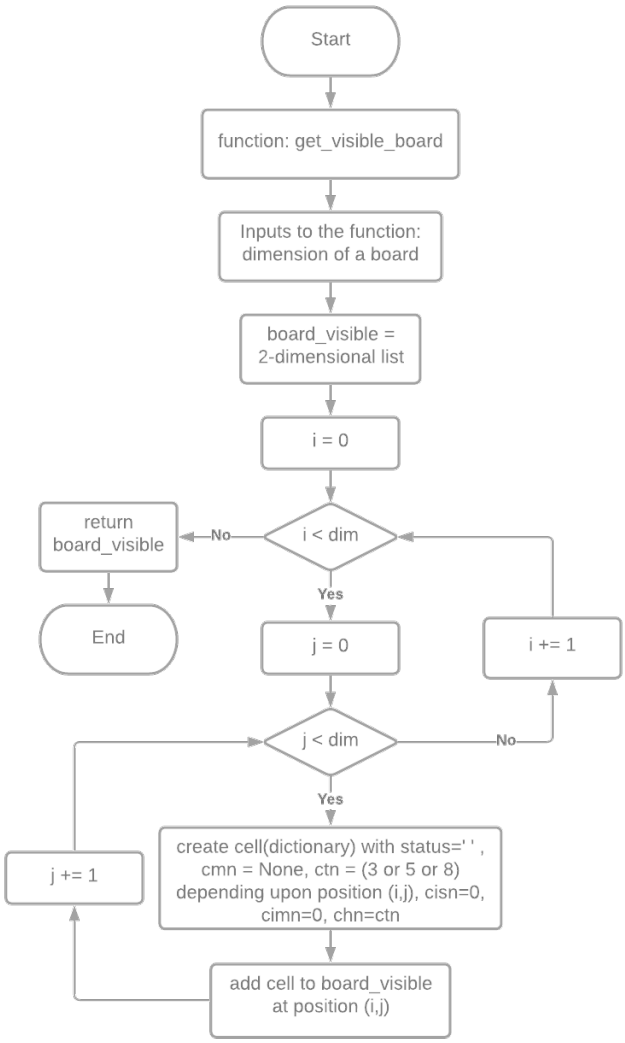
Once the mines are set, cmn of each cell in a board has to be updated according to status of its neighbors. Function initialize_cmn sets cmn of each cell in a board to its count of mine neighbors. Count of mine neighbors of a particular cell are computed by function get_count_of_mine_neighbors. Function get_count_of_mine_neighbors checks for the existence of a cell in all 8 directions and checks status of every existent cell. It returns a count of existent neighbors with status = 1 (mine).

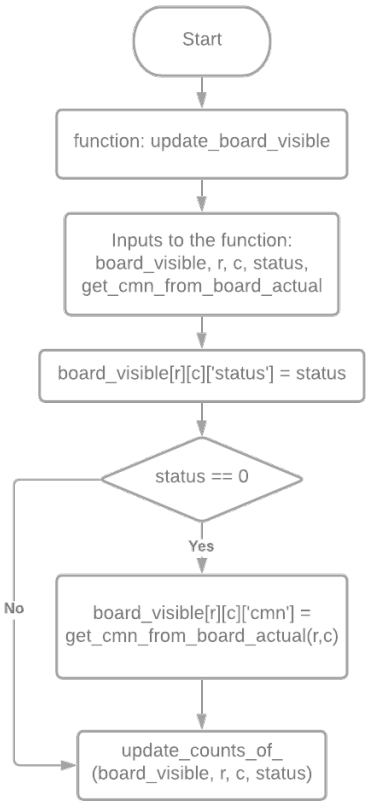


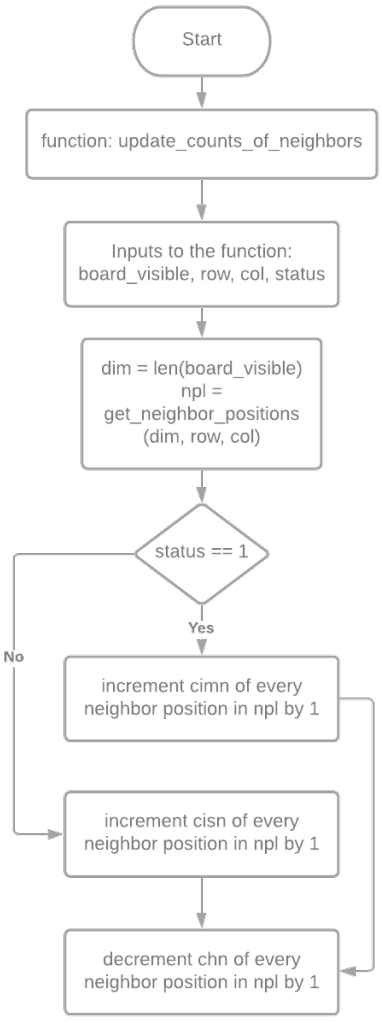


3.2 Visible Board Generation

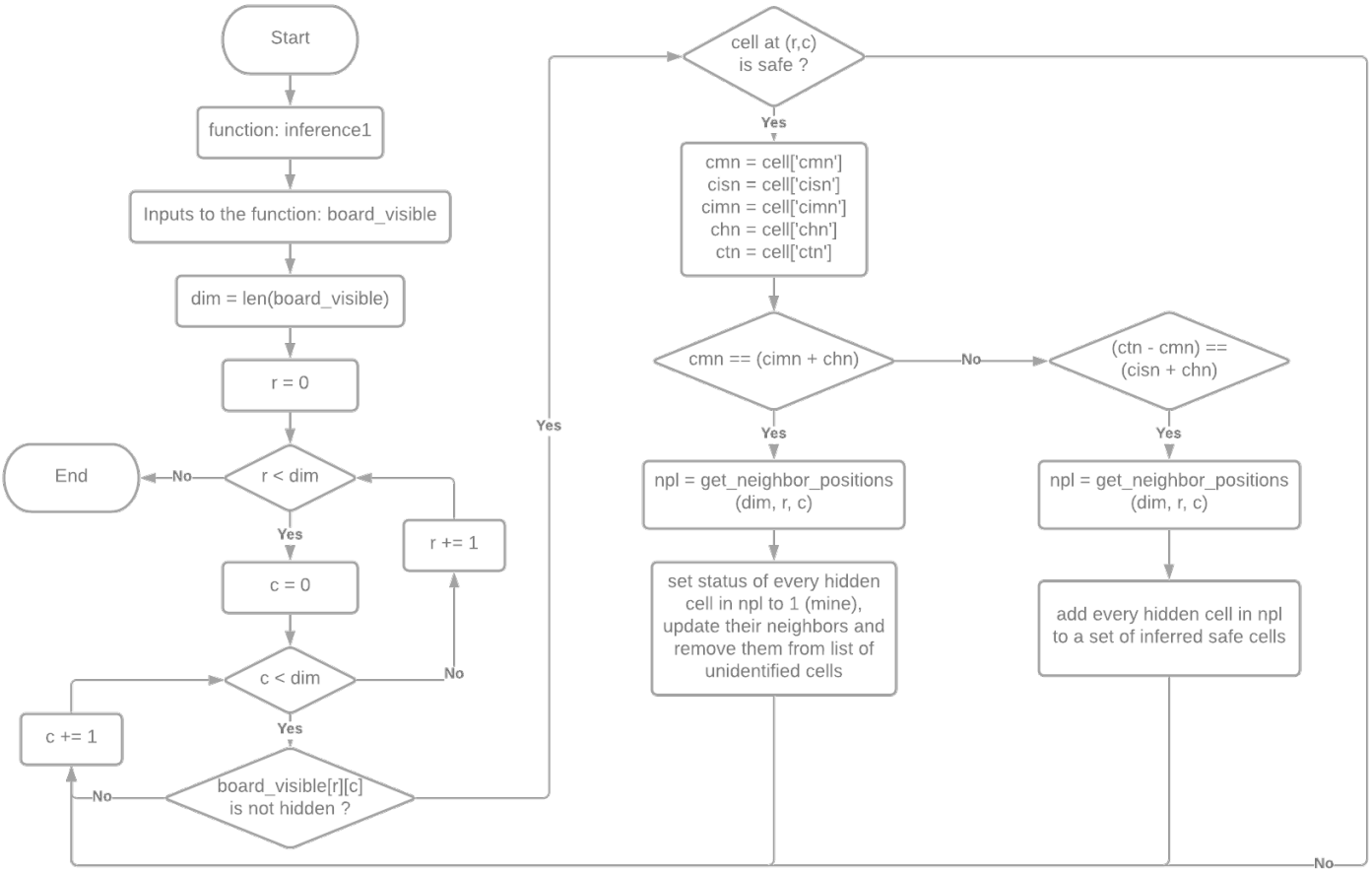
Visible board is the knowledge base of AI agent where it stores the information gained from querying the environment and inferences based on them. Visible board is stored as a 2-dimensional list in which each cell is a dictionary. Dictionary contains following keys - status, cmn (count of mine neighbors), ctn (count of total neighbors), cisn (count of identified safe neighbors), cimn (count of identified mine neighbors), chn (count of hidden neighbors). Visible board is generated by function get_visible_board where every cell's attributes are set as follows: status = ' ', count of mine neighbors = None, count of total neighbors = (3 or 5 or 8 depending upon the position of cell in a board), count of identified safe neighbors = 0, count of identified mine neighbors = 0, count of hidden neighbors = count of total neighbors. As AI agent gains more information through queries and inferences, it updates the knowledge base via update_board_visible function. It also updates counts of neighbors of queried or inferred cells via update_counts_of_neighbors function.



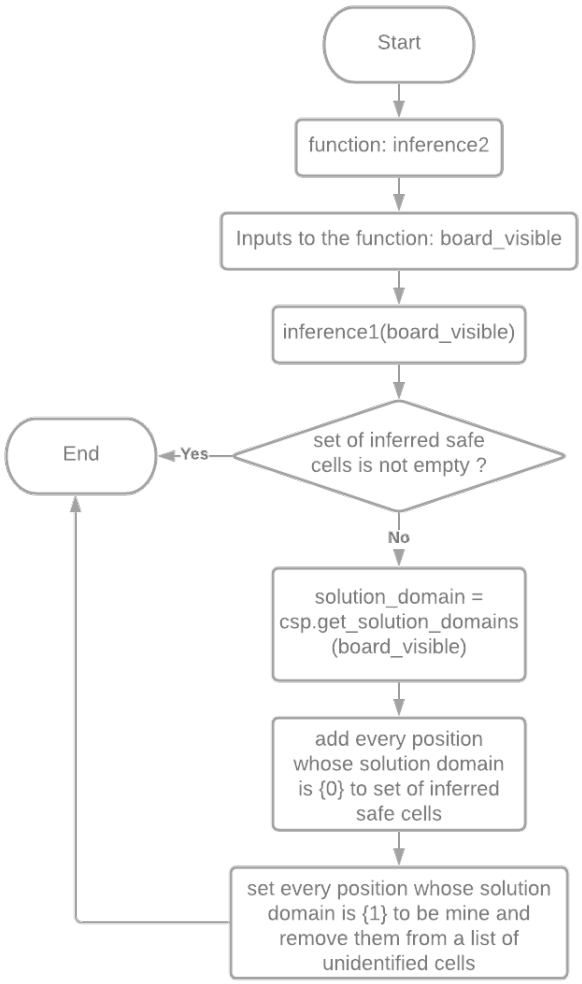




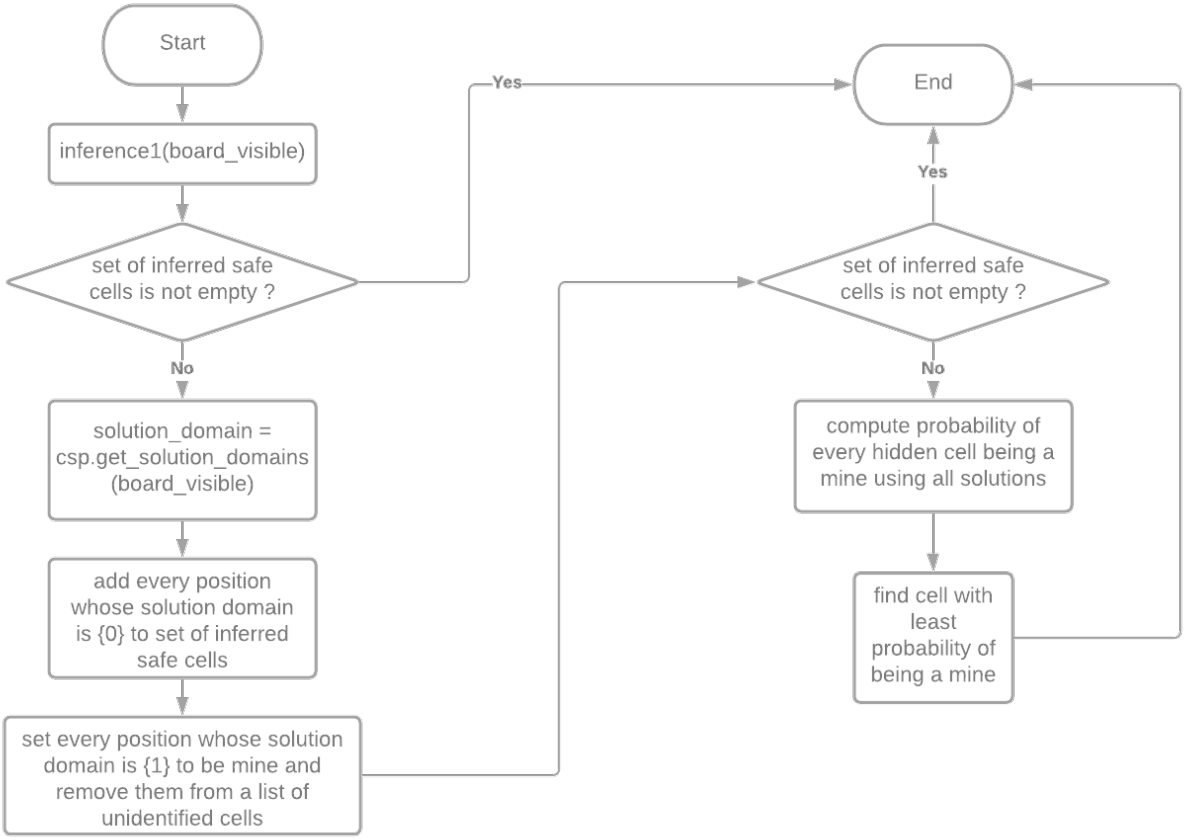
Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. List of unidentified cells is initialized to contain every cell in a visible board. Set of inferred safe cells is initialized to an empty set. Basic agent plays the game until all cells in a visible board are identified (revealed or flagged). It uses two basic rules to make inference about the hidden cells - 1) If count of mine neighbors (cmn) is equal to sum of count of identified mine neighbors (cimn) and count of hidden neighbors (chn), then every hidden neighbor is inferred to be a mine 2) If count of safe neighbors (ctn - cmn) is equal to sum of count of identified safe neighbors (cisn) and count of hidden neighbors, then every hidden neighbor is inferred to be safe. If a cell is inferred to be a mine, basic agent updates that position in a visible board (knowledge base) and removes same position from a list of unidentified cells. If a cell is inferred to be safe, it adds that cell to a set of inferred safe cells. If a set of inferred safe cells is not empty, basic agent chooses to reveal a position from that set. If a set of inferred safe cells is empty, it makes inference before making a move. If inference does not conclude safety of any hidden cell, then it chooses to reveal a hidden cell at random.



Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. List of unidentified cells is initialized to contain every cell in a visible board. Set of inferred safe cells is initialized to an empty set. Single improved agent plays the game until all cells in a visible board are identified (revealed or flagged). To possibly save computation time, it first tries basic agent's inference and checks if any hidden cell can be inferred to be safe. If basic agent's inference succeeds in inferring a hidden cell to be safe, it proceeds for a move according to basic agent as described above in 3.3. If basic agent's inference does not infer any hidden cell to be safe, single improved agent performs constraint satisfaction based inference. It frames a constraint satisfaction problem as follows: 1) All hidden cells are treated as indicator variables 2) Domain of every variable is 0,1 (0=safe , 1=mine). 3) Each identified safe cell gives one constraint on its hidden neighbor cells. Single improved agent uses backtracking search to find all possible solutions to a CSP. For a game-play to execute in reasonable computation time (1.5 minutes), single improved agent uses at most 16 most restricted variables (MRV) and constraints involving only them. This number of variables can be easily tuned to have better performing AI agent at the cost of more computation time. Once all possible solutions are found, values of a variable from all solutions are merged to find resultant domain of that variable. If resultant domain of a variable is 0, then it is inferred that a cell is safe. If resultant domain of a variable is 1, then it is inferred that a cell is mine. If resultant domain of a variable is 0,1, then nothing can be inferred about a cell. CSP based inference allows interaction between multiple clues to infer more information. If a cell is inferred to be a mine, agent updates that position in a visible board (knowledge base) and removes same position from a list of unidentified cells. If a cell is inferred to be safe, it adds that cell to a set of inferred safe cells. If both inferences do not conclude safety of any hidden cell, then it chooses to play a hidden cell at random.



Before making a move, it first initializes a list of unidentified cells and a set of inferred safe cells. It keeps on playing a game until all the cells in a visible board are identified. It uses probabilistic inference in addition to inferences used by single improved agent. It proceeds like single improved agent until constraint satisfaction based inference. If CSP based inference does not conclude safety of any hidden cell, then it performs probabilistic inference to make a move. Backtracking search of CSP based inference has already computed all possible solutions. These solutions can be used to compute probability of each cell being a mine. Probability of a cell being mine is the relative frequency of that cell (indicator variable) being 1 amongst all solutions. Cell with the least probability of being a mine is chosen by double improved agent. Probabilistic inference never updates the status of any hidden cell in a visible board (knowledge base) to safe or mine. Probabilistic inference is used only to decide which hidden cell to reveal. Similar to single improved agent, double improved agent can be improved further by tuning number of most restricted variables used in backtracking search.



4 Answers to the questions about design choices

4.1 Representation

Board (board_actual) is represented as a 2-dimensional list (list of lists). Each cell in a board is a dictionary. Keys in a dictionary are as follows: 1) cmn (count of mine neighbors) 2) ctn (count of total neighbors) 3) status (0=safe , 1=mine). Values in this board are never updated.
In order to disallow direct access to this board, it is stored within two nested functions (reveal_position and get_cmn_from_board_actual) that are created and returned by function get_board.
Function get_cmn_from_board_actual returns count of mine neighbors only for the cells that are safe (status = 0).

```
[[{'cmn': 0, 'ctn': 3, 'status': 0},
  {'cmn': 1, 'ctn': 5, 'status': 0},
  {'cmn': 1, 'ctn': 3, 'status': 0}],
 [{'cmn': 1, 'ctn': 5, 'status': 0},
  {'cmn': 2, 'ctn': 8, 'status': 0},
  {'cmn': 1, 'ctn': 5, 'status': 1}],
 [{'cmn': 1, 'ctn': 3, 'status': 0},
  {'cmn': 1, 'ctn': 5, 'status': 1},
  {'cmn': 2, 'ctn': 3, 'status': 0}]]
```

```
    0   1   2
  -------------
0 | 0 | 1 | 1 |
  -------------
1 | 1 | 2 | M |
  -------------
2 | 1 | M | 2 |
  -------------
```

Knowledge base (board_visible) is also represented as a 2-dimensional list (list of lists). Each cell in a knowledge base is a dictionary.
Keys in a dictionary are as follows: 1) chn (count of hidden neighbors) 2) cimn (count of identified mine neighbors 3) cisn (count of identified safe neighbors) 4) cmn (count of mine neighbors) 5) ctn (count of total neighbors) 6) status (0=safe , 1=mine , ' '=hidden).
New information about a cell is stored in the knowledge base by updating values in a corresponding dictionary.
AI agents also have a set to store cells that are inferred to be safe but not yet revealed.

```
[[{'chn': 3, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 3, 'status': ' '},
  {'chn': 5, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 5, 'status': ' '},
  {'chn': 3, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 3, 'status': ' '}],
 [{'chn': 5, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 5, 'status': ' '},
  {'chn': 8, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 8, 'status': ' '},
  {'chn': 5, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 5, 'status': ' '}],
 [{'chn': 3, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 3, 'status': ' '},
  {'chn': 5, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 5, 'status': ' '},
  {'chn': 3, 'cimn': 0, 'cisn': 0, 'cmn': None, 'ctn': 3, 'status': ' '}]]
```

```
    0   1   2
  -------------
0 |   |   |   |
  -------------
1 |   |   |   |
  -------------
2 |   |   |   |
  -------------
```

4.2 Inference

There are four possible ways to get new information:

1) AI agent reveals a random cell that turns out to be safe:
AI agent removes that cell from a list of unidentified cells. It then updates status key of corresponding dictionary in visible board to 0. As the cell is safe, it gets a count of mine neighbors from actual board by calling function get_cmn_from_board_actual. It also increments cisn and decrements chn of every neighbor of revealed cell by 1.

2) AI agent reveals a random cell that turns out to be mine:
AI agent removes that cell form a list of unidentified cells. It then updates status key of corresponding dictionary in visible board to 1. As the cell is mine, it does not get a count of mine neighbors. It also increments cimn and decrements chn of every neighbor of revealed cell by 1.

3) AI agent infers that a cell is safe:
AI agent only adds that cell to a set (set_of_inferred_safe_cells) where it stores cells that are inferred to be safe but not yet revealed.

4) AI agent infers that a cell is mine:
AI agent removes that cell form a list of unidentified cells. It updates status key of corresponding dictionary in visible board to 1. As the cell is mine, it does not get a count of mine neighbors. It also increments cimn and decrements chn of every neighbor of revealed cell by 1.

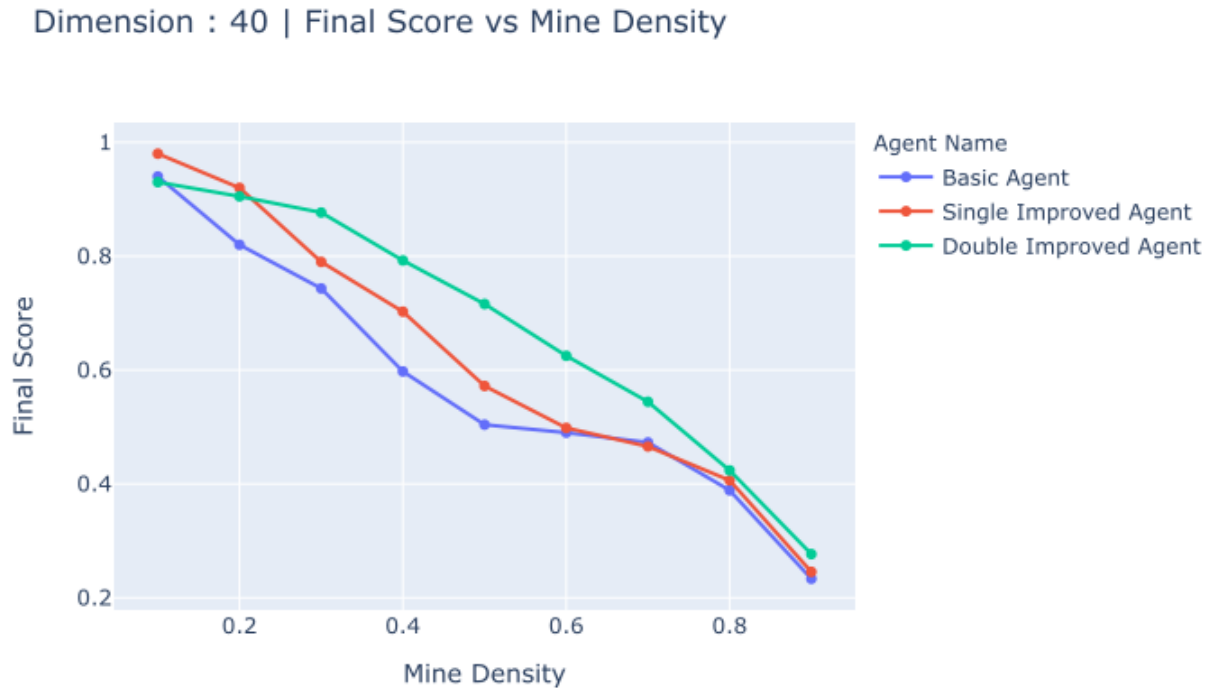
All three AI agents (basic, single improved, double improved) never infer a safe cell as a mine cell (No false positive) because:
a) Basic agent uses a rule - (total mine neighbors == (revealed mine neighbors + hidden neighbors)) implies that every hidden neighbors has to be a mine. (Logical implication with no uncertainty involved about inference).
b) Single improved agent uses solutions to constraint satisfaction problem in addition to basic agent's rule. It infers that a cell is a mine only if indicator variable of that cell is assigned 1 in all the solutions, i.e., resultant domain of that cell is {1}. New constraints can only reduce this domain but cannot add new value to it. On the other hand, domain {1} cannot be reduced any further as the indicator variable has to have a value of 0 or 1. This squeezing of the domain makes it certain that cell is a mine.
c) Double improved agent uses strategies of above two agents to infer that a cell is a mine. It never makes inference about cell's status based on its probability. It uses probability only to choose a hidden cell to be revealed.

4.3 Decisions

AI agent keeps a set of cells that are inferred to be safe but not yet revealed. For every move, agent chooses and removes a cell from this set if it is not empty. If set is empty, then agent executes its corresponding inference. If inference concludes safety of any hidden cell, then it chooses to reveal that cell. If inference does not conclude safety of any hidden cell, then basic and single improved agent choose a hidden cell at random. If inference does not conclude safety of any hidden cell, then double improved agent chooses a hidden cell that has least probability of being a mine.

4.4 Play-by-play Progression

Play-by-play progression of all three AI agents for a minesweeper board of size = 10 and total number of mines = 45 has been written in three separate files.
Links to these files are below:
Basic Agent:
https://github.com/Prathameshk2696/Minesweeper_AI/blob/main/example_basic_agent.txt
Single Improved Agent:
https://github.com/Prathameshk2696/Minesweeper_AI/blob/main/example_single_improved_agent.txt
Double Improved Agent:
https://github.com/Prathameshk2696/Minesweeper_AI/blob/main/example_double_improved_agent.txt



4.6 EFFICIENCY

Computation time of single and double improved agent increases exponentially as depth limit of backtracking search tree increases ($O(2^d)$). This problem specific constraint is encountered in both CSP-based as well as logic-based implementations.

For the game-play, all three AI agents require actual board ($O(dim^2)$), visible board ($O(dim^2)$), list of unidentified cells ($O(dim^2)$) and a set of inferred safe cells ($O(dim^2)$). In addition to that, single and double improved agents require space to store one path of backtracking search (O(depth)) and all the solutions computed by backtracking search ($O(2^d)$). Space required to store all solutions is an implementation-specific constraint. Instead of storing and then merging all solutions to compute resultant domains of hidden cells, every new solutions can be merged into current resultant domain immediately.

5 Conclusion

Single improved agent outperforms basic agent by considering interaction between multiple clues via constraint satisfaction. Double improved agent outperforms single improved agent and basic agent by considering probabilities of hidden cells being mine. Depending upon the availability of time, single and double improved agents can tune the depth limit of backtracking search to improve their performance even further. Their performance will be at peak when the backtracking search involves all the constrained hidden cells at any given move. The only case where performance order can possibly change is when game-play involves more number of random moves.

6 Future Work

1) Remove implementation-specific space constraint by merging new solutions into current resultant domains immediately.
2) Implement logical inference based AI agent.
3) Improve performance of CSP-based AI agents by implementing rule learning.

7 References

1) https://docs.python.org/3/
2) https://plotly.com/python/line-charts/
3) https://en.wikipedia.org/wiki/Constraint_satisfaction
4) https://en.wikipedia.org/wiki/Backtracking
5) https://www.oreilly.com/library/view/learning-python-5th/9781449355722/