CS 520 Introduction to Artificial Intelligence
Professor Wes Cowan
Assignment 2: Minesweeper

Group Members:
Keya Desai (kd706)
Prakruti Joshi (phj15)
Rushabh Bid (rhb86)

Representation:

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

Environment:

The environment for a minesweeper game will be the game board generated in the form of a grid. There are two grids used in this approach. One is the actual environment grid (*env_grid*) that stores the completed state of the generated game, while the other is the agent grid (*user_grid*) which, as the name suggests, is the one that the user (agent) solves. The user grid is initially full of hidden cells and is eventually revealed as the agent works on it.

Typically, there are two parameters for a minesweeper game: grid size and mine density. For ease of implementation and understanding, a square grid is used with side length '*d*'. The mine density is the number of mines in the grid *'num_mines'* as a percent of the grid size '*d x d*'. The grids are represented by numpy arrays of size '*d x d*'.

Game generation:

To get started, numpy array of size '*d x d*' is initialized. Then, we add the given number of mines at random locations in the grid. The value of associated to every "clear cell" is given by the number of mines in the immediate neighborhood. The result (environment grid) is the final state of the game. The agent is provided with an empty grid and it starts solving it. Each cell in the grid can take different values based on its nature (shown in Table 1). When the agent selects a cell to explore, the agent grid updates the value of that cell according to its value in the environment. The agent then infers whatever it can from the information obtained.

The obvious information that the agent obtains are the cell values. The agent can use "baseline" conditions on these cell values to infer new information.

The other approach that the agent can take is the "knowledge base" approach. This requires the use of a knowledge base in which all the information is stored. This information is then checked for internal relations and new information is inferred. Given the same state of the game, this approach extracts more information than

the baseline approach. The knowledge is stored in the knowledge base as a tuple of all the unexplored neighbors of a given cell and the associated number of unidentified mines.

Knowledge - ([set of unexplored neighbors]; # of unidentified neighboring mines; set length/Number of unexplored neighbors)

The knowledge base is a list of all such tuples that can be generated from the information gained from the environment.

Cell value	Nature of cell
-9	Burst mine
-1	Armed/disarmed mine
0 <= k < 9	Safe cell with k neighboring mines
9	Hidden cell
20	Safe cell with no information

Table 1: Cell values and corresponding nature of the cell

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 agent infers all that it can given the approach and the knowledge of the current state of the environment. When there are no moves left for the agent to make, it chooses an arbitrary cell to explore. Based on the information available, the choice of this arbitrary point of exploration can either be informed or uninformed. This case can be associated to a local goal state and thus we solve it using a random restart.

The first one is the baseline approach. The "baseline" approach provides two conditions of inference for each cell. Firstly, it states that if the number of hidden neighbors for a cell is the same as the number of unidentified mines around it, all of it's hidden neighbors are mines. On the contrary, the second condition states that if the number of identified mines around a cell is the same as the value of that cell, all of it's hidden neighbors are safe. This is the basic approach that deals with all the neighbors of a given cell independently.

The second approach is the advanced approach. This approach uses knowledge extracted from the situation to infer new knowledge. A knowledge base is maintained to store the information. The agent then checks for relations within the knowledge base. For each set of unexplored neighbors that is a subset of some other set, new information is identified for their set difference.

For example:

Knowledge Base -

Set 1 - ([x1, x2, x3]; 2; 3)
Set 2 - ([x1,x2]; 1; 2)

Inference -

Set 3 - ([x3]; 1; 1) ⇒ *The cell x3 definitely has a mine*

This approach uses dependencies to infer knowledge that the baseline approach would not be able to infer because of its independent search methodology.

These two approaches keep applying till they can no longer find any new information or the game has reached its end. This becomes the exit condition for these approaches and if the game has not ended, we use the random restart strategy again.

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?

A situation of the board can be broadly classified into two cases: either there are some cells that can be identified as safe or unsafe without a doubt, or there are none. The approach to decide which cell to search next is different in each case.

Case - 1: Some knowledge about surely safe or unsafe cells

As discussed earlier, there are two approaches used to search cells in this case. The first approach is the 'baseline' approach and the other one is the 'optimized' approach which uses a knowledge base. Both the approaches discussed under this case fetch only certain information, thereby eliminating the involvement of risks. Since, this case does not involve any risks, it doesn't require critical decision making, and thus isn't relevant to the topic of discussion.

Case - 2: No knowledge about surely safe or unsafe cells

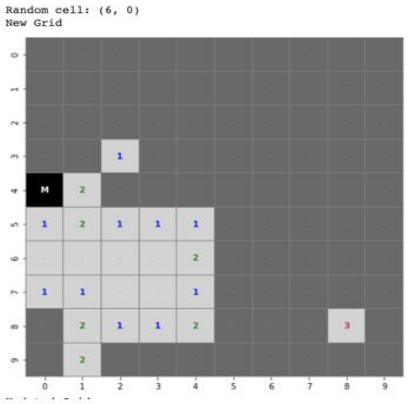
Contradictory to the previously discussed case, in this case we do not have any information that will make us believe that a cell is safe or unsafe with complete certainty. The two discussed approaches reach a state where they cannot help us decide which cell to search next. Here, we have to decide to search for a cell based on its likelihood of being a mine. The factor of risk is introduced due to this uncertain situation. Now, how do we decide which cell to search next? Should it be a cell with some explored neighbors?

In typical situations, we can assume that the mine density of a given grid is not very high. As the mine density increases, the grid becomes harder to solve. So, in such typical situations, the probability of stepping on a mine around an explored cell could be much higher. For instance, consider a 400 cell grid with 40 mines. If we choose to search a cell next to an explored cell, the best case would be a cell with value 1, with all 8 neighbors unexplored. The probability of each of them being a mine is 0.125. But if we choose a cell with no explored neighbors, the probability of it being a mine is approximately 40/400 or 0.1 (considering most of the grid is still unexplored). As the game approaches its end, the chance of experiencing this case (Case 2) reduces significantly due to the large amount of knowledge that we poses. And even if we do get such a situation, the chance of there being a cell with all unexplored neighbors is reduced.

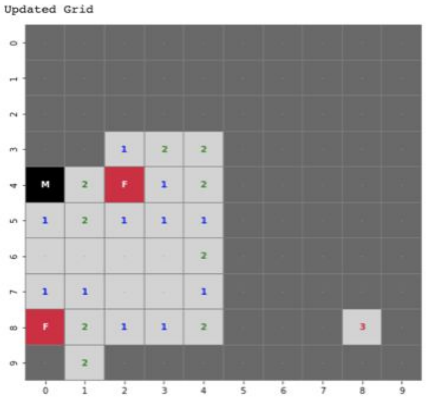
In general, for grids with low mine densities (about 10 - 20 %), it is a safer option to randomly pick a cell with all unexplored neighbors than a cell with some explored neighbors. This saves a lot of computation while still having a good chance of moving ahead in the game.

Performance:

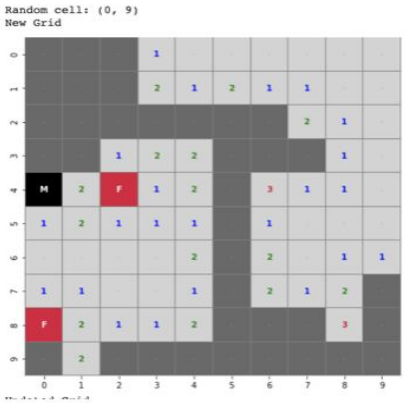
For a reasonably-sized board and a reasonable number of mines, including a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program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



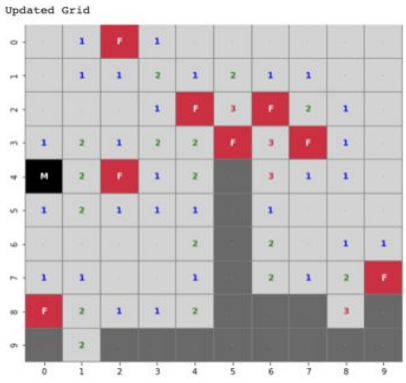
Randomly opened cells.



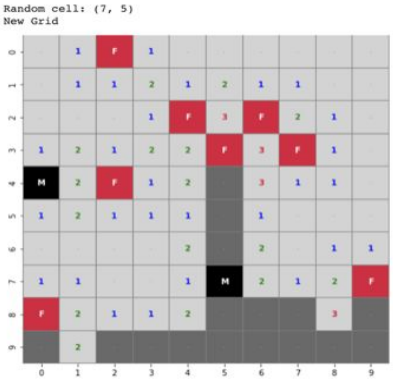
Baseline Inference



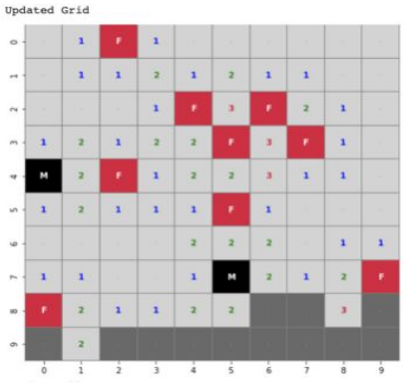
Random cell (0,9) opened



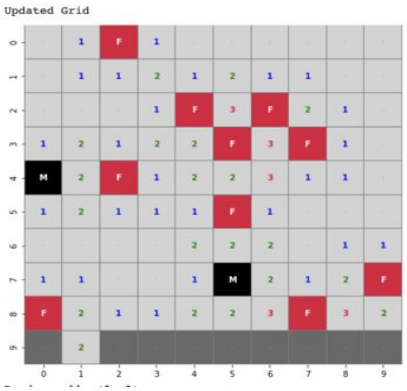
Baseline Inference



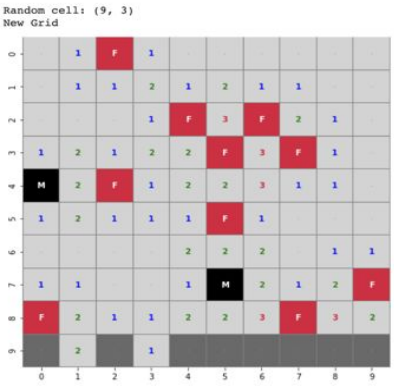
Randomly a mines is opened



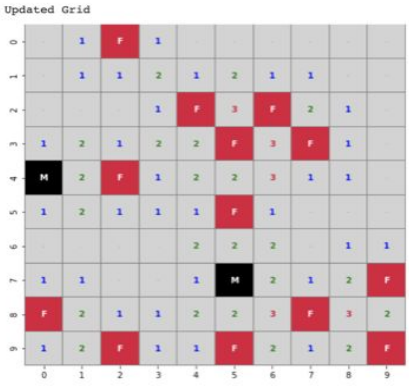
Baseline Inference



Baseline Inference -Iteration 2



Randomly cell (9,3) opened



Baseline Inference - Grid solved

Figure 1: Agent solving the game using only baseline approach. Grid size: (10x10), #Mines = 15, Score = 13

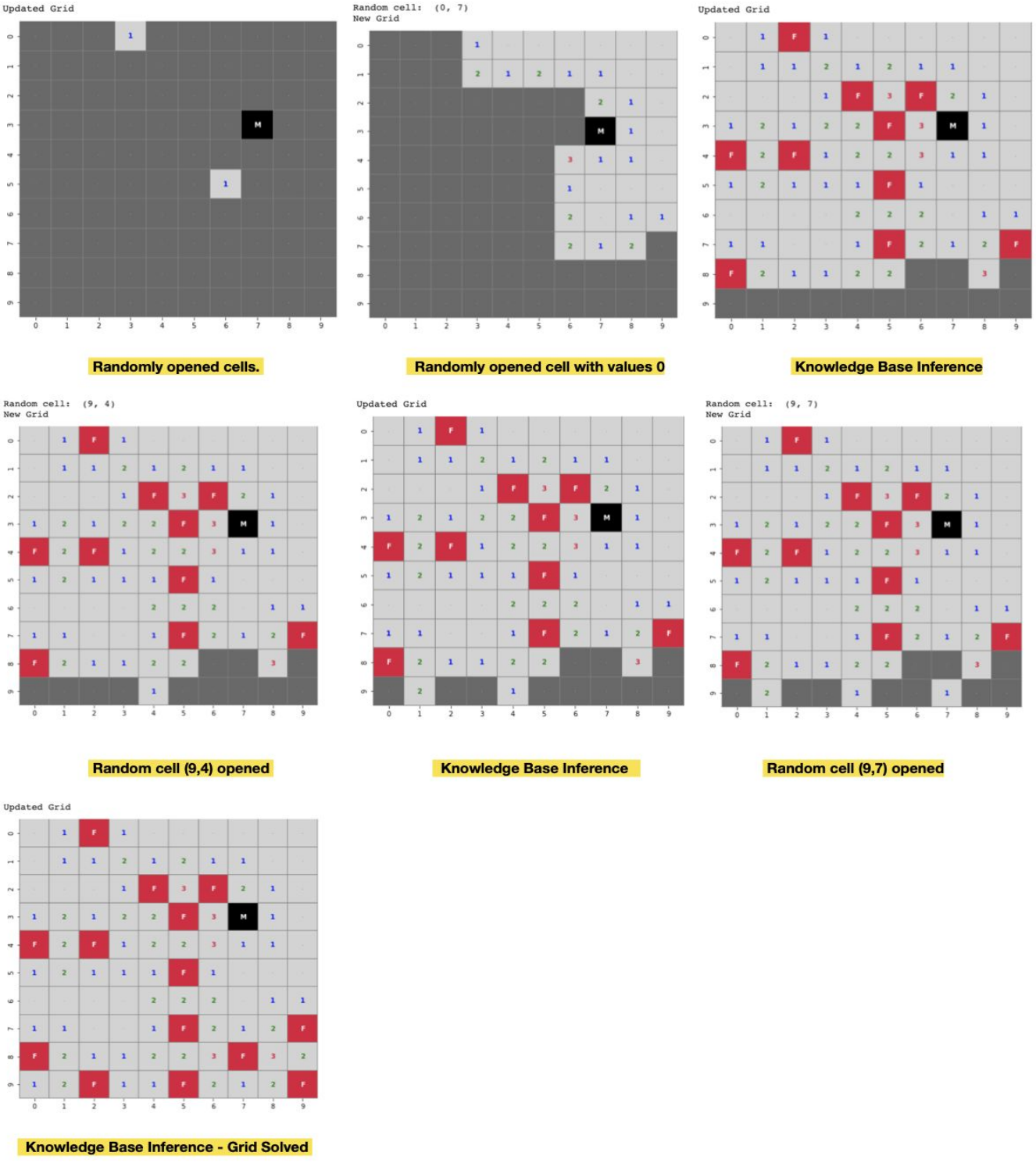


Figure 2: Agent solving the game using knowledge base approach. Grid size: (10x10), #Mines = 15,Score =14

In our program, we have used two inference based techniques- one simply inferring the cell as mine or safe depending on the local neighborhood information (baseline approach) and the other knowledge base inference which uses the neighborhood information of the neighbors to classify the cells. We have integrated both the inference approaches into a common approach known as the '*optimized_approach'*.

Any inference of mine will be determined by the knowledge of number of hidden neighbors, number of discovered mines and number of safe neighbors of a cell. This information for a cell is propagated in the knowledge base representation and inference approach.

If the number of hidden neighbors equals the value of the cell minus the number of discovered cells, then all the hidden neighbors are mines. If the number of discovered mines of a cell is equal to the cell value, then all the hidden neighbors are safe. Since this base rule is applied everywhere, the discovery of mines is progressing as it should according to the constraints and logic of the game. Thus, there was no decision that surprised us. There were discoveries that were not very obvious to a human player, but that can be inferred with some additional effort via logic.

Performance:

For a fixed, reasonable size of board, plot as a function of mine density the average final score (safely identified mines / total mines) for the simple baseline algorithm and your algorithm for comparison. 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'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why?

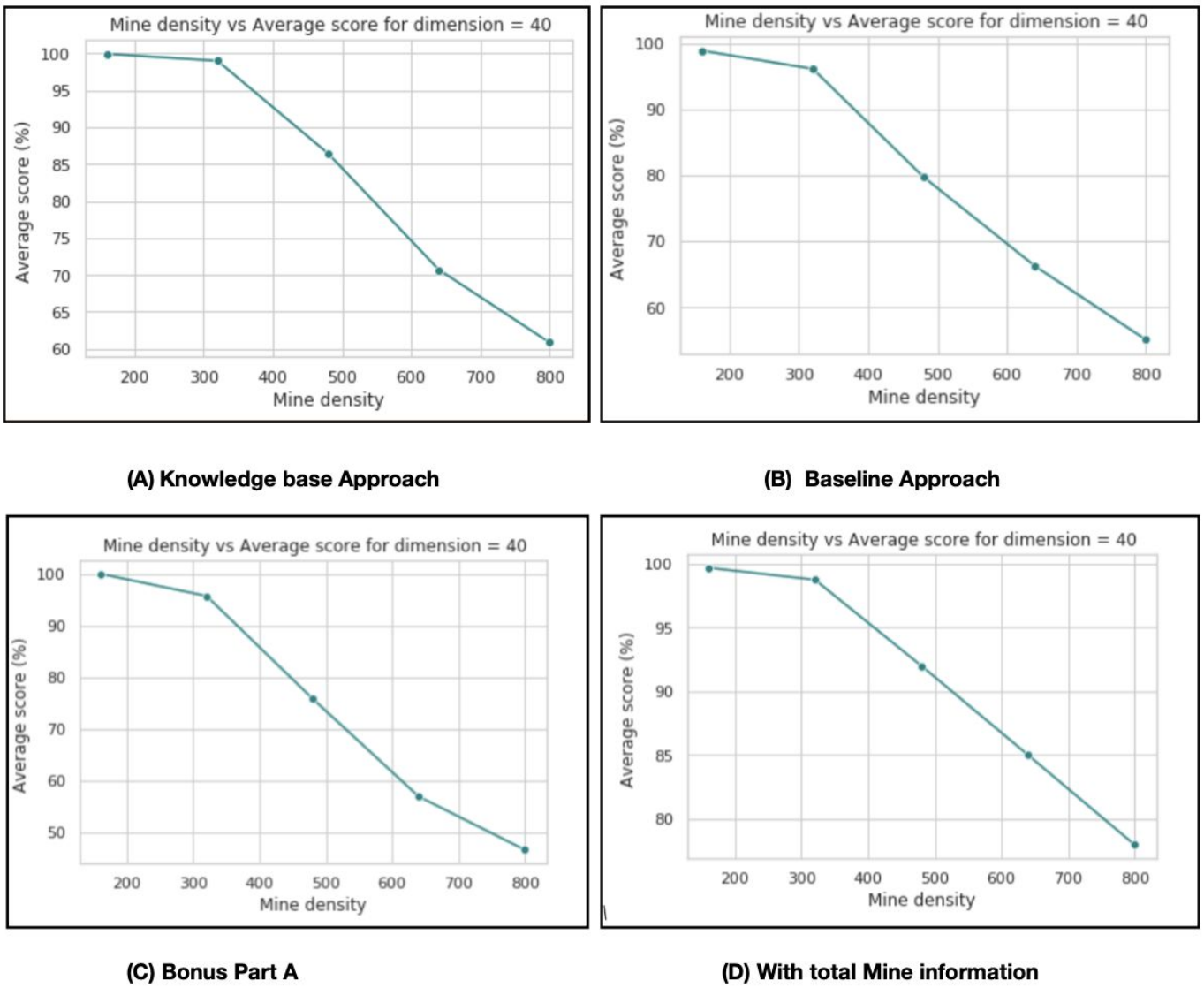


Figure 3: Mine density(0.1, 0.2, 0.3, 0.4, 0.5) vs Average Score

The need of more random selects increases with increasing mine density and the probability of having a mine at a cell is also more since number of mines in the grid are more. Hence the score should decrease as the mine density increases. The plot of mine density vs average score confirms this intuition.

The minesweeper becomes ‘computationally’ hard when the dimension is high and the mine density is greater than 0.3. We tested the optimized knowledge base approach for 100x100 dimension. It produced results rapidly when the number of mines was 2000. For number of mines greater than 3000, the algorithm became drastically slow.

The difference between the two approaches is in terms of their ability to extract information and their complexity. As far as the score is concerned, the baseline approach can never beat the optimized approach as there is no information that baseline finds and optimized approach does not. The knowledge base will always check for the baseline conditions that deal with individual cell information independently. Additionally, it also checks for relations within different elements of the knowledge base, thus always staying a step ahead of the baseline strategy. There might be instances where the baseline strategy demands multiple restarts while the knowledge base solves the game in one go.

However, there are cases where the baseline strategy could be preferred over the optimized approach. The baseline strategy only checks for two conditions for each cell and hence completes inference rapidly. Even though it might not extract all the information that was possible to be extracted, it is evidently quicker than the knowledge base approach. Also, the maintenance of a knowledge base adds to the space complexity while the baseline strategy does not demand any additional space. Thus, in scenarios in which the score isn’t a critical factor or the dimension of the grid is low or the mine density of the game is low, it might be a better idea to go for the baseline strategy.

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?

A cell in the minesweeper grid provides a clue about the number of mines in the neighborhood. A cell with the value '0' means that all it's eight neighbors are safe and can be opened for more clues. In a grid with low mine density, the number of '0' cells will be high, and thus there is a high probability of such '0' cells to be adjacent to each other. Thus, opening one of such '0' cells will lead to exploration of a lot of safe cells. The approach uses recursive calls to explore the large portions of safe cells given by these adjacent '0' cells. As the size of the grid increases, if the mine density is not sufficiently large, the recursive calls give rise to a space constraint. However, this is an implementation specific constraint and can be tackled by eliminating recursion and using loop iteration instead.

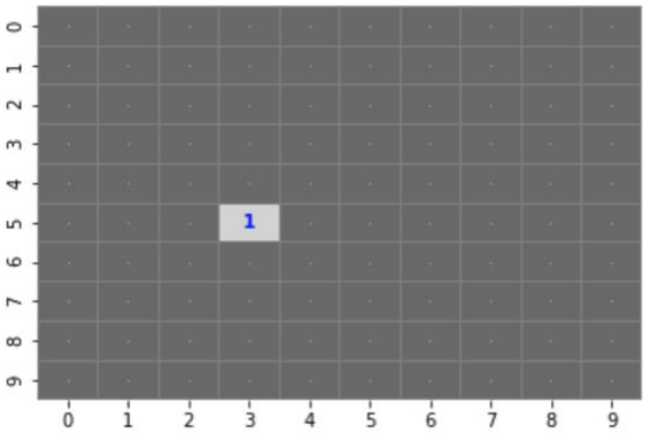
The advanced approach to solving the grid employs a knowledge base to infer decisions based on the knowledge explored. The addition of knowledge provided by 'k' cells takes O(k) time. The inference from this knowledge base takes O(k^2) time. Notice that the time complexity depends on the number of cells, and not the side length of the grid. The number of cells grows quadratically as the size of the grid increases. When the game is about to end, almost all cells are explored, i.e. number of cells explored 'k' is approximately equal to the square of the side length of the grid 'd'. Thus the knowledge based inference runs in O(d^4) time. This time constraint is problem specific.

Improvements:

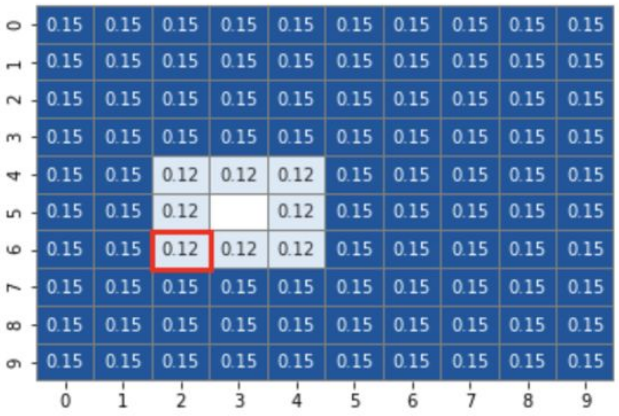
Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action? How can you use this information to effectively improve the performance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

A solver to the minesweeper game looks at the current situation of the environment and identifies cells that are certainly safe or have mines under them. It can use the numbers to evaluate probabilities of presence of mine at a given location using the information in the environment, but as discussed before, it is a better idea to randomly pick a cell that is not opened. However, the development in the situation where the solver knows the total number of mines present, might make room for improvement.

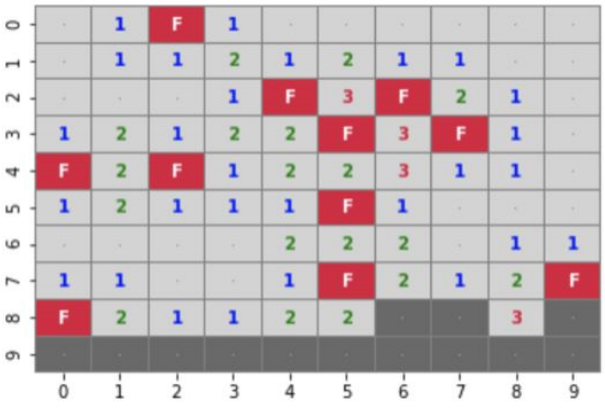
Firstly, when we were not aware of the number of mines present in the grid, it didn't make sense to take a probabilistic approach because we had no information about the probabilities of the unexplored regions. But, the provision of total number of mines adds clarity to the probabilities of the unexplored regions as well. This helps us make an informed decision about which cell to choose when we have no certain inference from the environment. The requirement is to calculate probabilities of presence of mine at each unexplored location. All the data we need is available in the knowledge base. The only thing we need to add to the knowledge base is the set of all unexplored cells and the total number of unexplored mines. Now the probability of mine at each location can be considered as the mean of probabilities of it being a mine given the value of each of its neighbor independently. Each of those probabilities are given by dividing the number of unexplored mines of a set by the set length. Once we have all the probabilities we can choose the minimum probability cell to explore next.



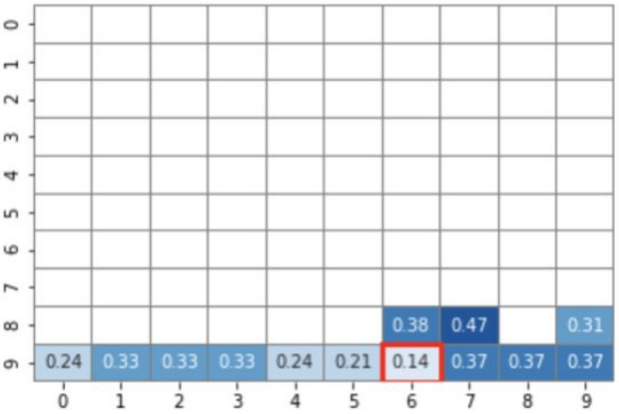
(A) Cell (5,3) opened randomly.



(B) The cells surrounding (5,3) has lower probability of having a mine than the rest of the cells. Hence a cell (6,2) is selected to be opened next.

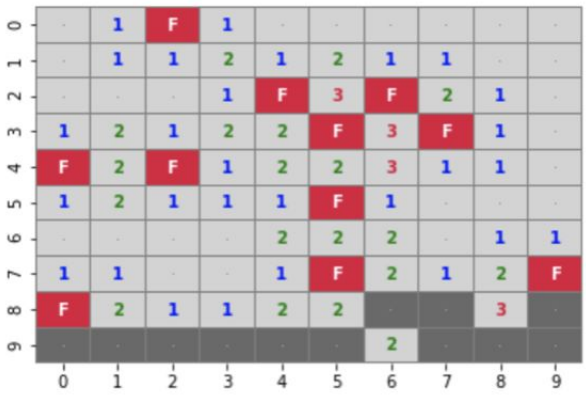


(C) Agent grid after inferencing using knowledge base approach.

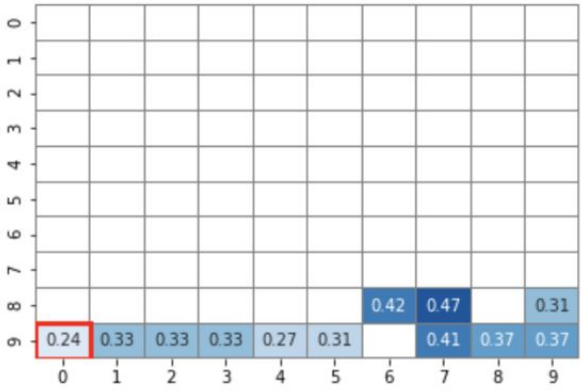


(D) Probabilities of having a mine out of the cells yet to be opened. (9,6) has the lowest probability. Hence it is opened next.

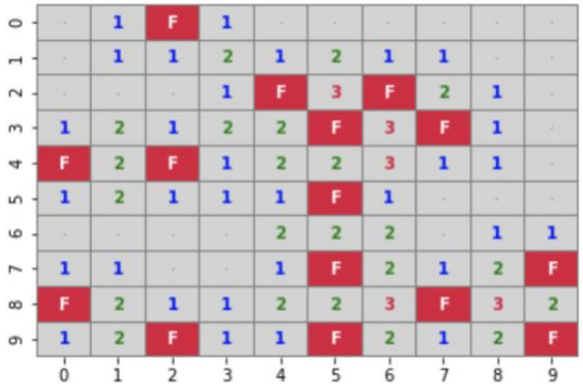
Figure 4: Choosing a random cell to open based on the probability of having a mine on that cell.



(E) Cell (9,6) opened. Agent grid after inferencing using knowledge base approach.



(F) Probabilities of having a mine out of the cells yet to be opened. (9,0) has the lowest probability. Hence it is opened next.



(G) Cell (9,0) opened. The agent is able to solve the entire grid.

Figure 4: Choosing a random cell to open based on the probability of having a mine on that cell.

We used the extra information, but can we do more? The information added to the knowledge base about the number of remaining mines can be used as another equation to solve for mines. This kind of information usually helps in the 'end game' strategy. Suppose we have a situation where there are two possible permutations of mines that satisfy the current situation. One of them uses 3 mines and the other uses 2. If the information about the number of remaining mines says that only 2 mines are left, we find which of the two permutations is correct. Thus, by implementing this, we can speed up the solving process as the game approaches its end without adding any uncertainty or room for error.

Bonus: Dealing with Uncertainty

1. When a cell is selected to be uncovered, if the cell is `clear' you only reveal a clue about the surrounding cells with some probability. In this case, the information you receive is accurate, but it is uncertain when you will receive the information.

Assumption: The mine information of the grid is always revealed. Only the neighborhood value of the safe cell is revealed with some probability.

The first model that adds uncertainty to the game still provides certain information. So, the approach to solve the game is still valid given we handle the obstacles set by this model. We are uncertain if we will get the information that we need, when we need it. There is some probability for a clear cell to stay hidden when it is selected to open. This introduces randomness in the exploration of the grid. To adapt to this situation, the information of a given cell is revealed with some probability, but in the case if is not revealed, the relevant query indicating that the cell is clear (does not contain mine) is **not** popped out of the knowledge base. Thus, the information is not leaked out and we don't have to rely on opening the cell randomly at a later stage. When a cell is opened via random restart, if the cell is safe and the information is not revealed, we add a query in the knowledge base adding the information that the particular mine is safe. However, we cannot interpret anything about the neighborhood information in this case.

Despite of handling this randomness, the performance of the algorithm is hit evidently by this model. The reason behind this is the randomness itself. All information is not made available when deserved, and thus a state of minimal inference can be reached. At this point we have to randomly open a cell and this increases the probability of hitting a mine. Also, if the revelation of information is random, it takes additional time to try and find the same information. Thus, this model hits the performance of the solver in terms of score as well as time efficiency.

2. *When a cell is selected to be uncovered, the revealed clue is less than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of underestimating the number of surrounding mines. Clues are always optimistic.*

In this case, the revealed value of a cell is uncertain in the sense that it is always less than or equal to the true number of surrounding mines. For example if the revealed value of the cell is 2, the true number of surrounding mines will be from the range [2,8]. As an adaption to this case, the knowledge base representation of neighbourhood mines information remains the same as earlier. Here, the cell value (the number of unidentified mines in the neighbourhood) will depend on the value that is revealed instead of its true value. The adaption in the inference of the underlying knowledge becomes that instead of using the cell value as the information about the neighboring mines, it is used as a lower bound on the number of neighboring mines. This model allows the inference of mines when *the length of a set of hidden neighbors is the same as its value (revealed cell value minus the number of discovered mines)*. We know from the value that there are at least a given number of mines which is same as the number of neighbors left. This obviously implies that all the remaining neighbors are mines. Additionally, we can discard a few cells from a set by simply subtracting the number of cells discarded from the value of the knowledge and the inequality still holds true.

3. *When a cell is selected to be uncovered, the revealed clue is greater than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of overestimating the number of surrounding mines. Clues are always cautious.*

The model introduces uncertainty to the value of the cells revealed. If the cell is a mine, it is unaffected in this model. However, if the cell is clear, the information we obtain is an overestimate of the true value. The value obtained for any clear cell is greater than or equal to the true value. For instance, a cell with 3 mines in its neighborhood will actually take some value in the range [3,8]. To adapt the solution to this model, the knowledge representation remains unaffected but the inference from it changes. The value for each knowledge is now an overestimate, and hence cannot be used for inter-knowledge inference. The most straightforward information available through this model is when a cell value is revealed as '0'. Since we know it is an overestimate, there can be no more than '0' mines in its neighborhood. Thus, we can infer from this information that all its neighbors have to be safe.

Comparison of all the methods:

Index	**Approach**	**Average score:** For 30x30 grid and 300 mines	**Average time:** For 30x30 grid and 300 mines (seconds)
1.	Baseline Inference	225.6	0.855
2.	Knowledge based inference (depending on global neighborhood information)	239.5	3.534
3.	Knowledge based inference with actual total mines information	267.0	6.145
4.	Bonus (Part 1) Accurate information is revealed but with some probability (Here, prob = 0.6)	207.7	1.587

Table 2: Comparison of score vs time for all methods

Conclusion:

As we can see from the above table, using only inference from baseline approach (based entirely on local data and comparisons), the algorithm is much faster. However, due to weak inference, the approach has to perform random restart multiple times. This increases the risk of opening a mine and thus the score is less as compared to other stronger inference based techniques such as knowledge base which inculcates information about neighborhood of neighbor too.

The highest score is when the information about the total number of mines present in the environment minesweeper game is known to the agent beforehand. Using this information, the probability of presence of mine for all the cells is calculated and the minimum is chosen. Also, the total mine information present in the knowledge base also helps to infer mine information about cells. This helps us mitigate the risk of opening a mine during random restart. However, this comes with a tradeoff of higher computation as for each random restart, the probability of presence of mine for the hidden cells (not determined cells) has to be calculated.

The bonus question modification of dealing with uncertainty reduces the average score. Since the information is revealed with a certain probability, the known information at every level is less. Thus, the extent of inference is reduced at each level as compared to the previous approach where we were able to deduce the neighborhood information from the safe cell revealed. Thus, the random restarts increase and so does the risk of opening a mine. Thus, the average score is drastically low as compared to the knowledge base approach where full information is revealed.

Contributions by team members:

The discussion and analysis related to each part of the problem were purely group based wherein each of us provided inputs and justifications based on individual understanding. After the discussion about the implementation strategies, the coding part of the algorithm was divided amongst us as follows:

Keya Desai	Baseline approach, Visualization and plots, Optimization, Improvement with total mine information (probability calculation for random restart), Bonus Part 2,3
Prakruti Joshi	Knowledge base approach, Optimization, Bonus- Part1, Debugging and Testing, Comparison between approaches, Documentation, Bonus Part 2,3
Rushabh Bid	Knowledge base approach, Bonus- Part1, Optimization, Debugging and Testing, Total mine information in the knowledge base, Documentation, Bonus Part 2,3