Assignment 2 – MineSweeper : Inference-Informed Action

Group members and Contributions

Divyaprakash Dhurandhar(dd839):
Involved in the development of base classes and algorithmic understanding of the simple square analysis algorithm for solving the minesweeper maze. Developed parts of the simple square analysis algorithm. Generated performance results for the minesweeper solver and contributed in the final report.

Abdulaziz Almuzaini (aaa395):
Worked on implementation in creating classes and the algorithms. Involved in answering the questions and analyzing the algorithms, data and the results. Created graph, tables and other visualizations for the reports.

Sri Gautham Subramani (ss2999):
Worked on the implementation of randomized selection of the algorithm. Involved in computing test cases for answering questions, explaining the inference, generating results and evaluating performance of the algorithm.

Shubhada Suresh(ss2970):
Contributed in development of algorithms and involved in report writing especially the bonus questions. Involved in analyzing the algorithm, generating results and evaluating performance of the algorithm.

Representation

The representation is a 2D array of squares, with each square storing several fields. For each cell we have the following information:

status = False	*Check if a specific cell has a mine or not.*
isVisited = False	*Check if a cell is opened or not.*
flagged = False	*Is the cell marked or not?.*
visible= "-"	*Initially all cells are blocked.*
value = 0	*The value inside the cells; 1 - 8 provides the clue information For the neighbours which have mines or 9 if it is a mine.*

Through the program these attributes are going to be updated accordingly and the clue cell information will be used to determine which cells to be flagged and which one are safe to be opened.

See the below figure for clear understanding.

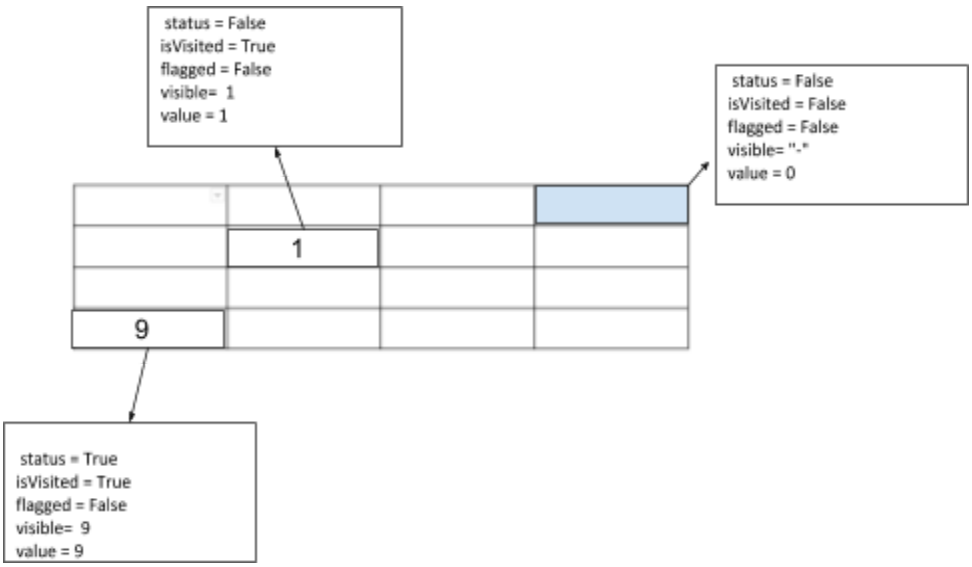


Figure 1: A simple illustration of our representation

The process:

1. We first initialize our board with the height and width specified by users.
2. Place mines randomly in the board. Beginner mode has 10 mines, Intermediate has 40 mines and Advanced has 99 mines.
3. The locations of placed mines are stored in a **set**. We need it later for the algorithm to check if they opened cell is a mine or not.
4. Then, we calculated the number of adjacent cells in order to collect information for the clue cells. This process is done by looking for the 8 adjacent neighbours and find how many mines are nearby. After this process the board will be similar to Figure 2.

```
The Actual Board
9  1  0  0  0  0  0  1  9
1  1  0  0  0  0  0  1  1
0  0  1  1  2  1  1  0  0
0  1  2  9  2  9  1  0  0
0  1  9  2  2  1  1  1  1
1  2  2  1  0  0  0  1  9
1  9  1  0  0  0  0  1  1
1  2  3  2  1  1  1  1  0
0  1  9  9  1  1  9  1  0
```

Figure 2 : The underlying board

5. Then we the game start we represent a closed board:

Figure 3 : The closed board

6. After the game start we randomly choose a cell and open it and open all zero cells and the clue information around it. In this example we are going to use the first cell(0,0) and open all adjacent cells; make them revealed.

Figure 4: opening all zeros and clue cells next to the initial cell

Inference

We have three main sets the algorithm needs to be deal with to check, process and collect information. These sets are the following:

- Opened Sets
    - Contains all cells that are already opened.
- Flagged Sets
    - Contains all cells that are suspected to be mines or we don't have much information to safely open it.

Besides these two sets, we have two main dictionaries/HashMaps that are going to be used with the opened/flagged sets. These dictionaries are:

- **Neighbours_dictionary**
    - Basically, for every cell we store all its neighbours.
    - The key is the cell and the values are a set of neighbours. See Figure 5 for illustration.

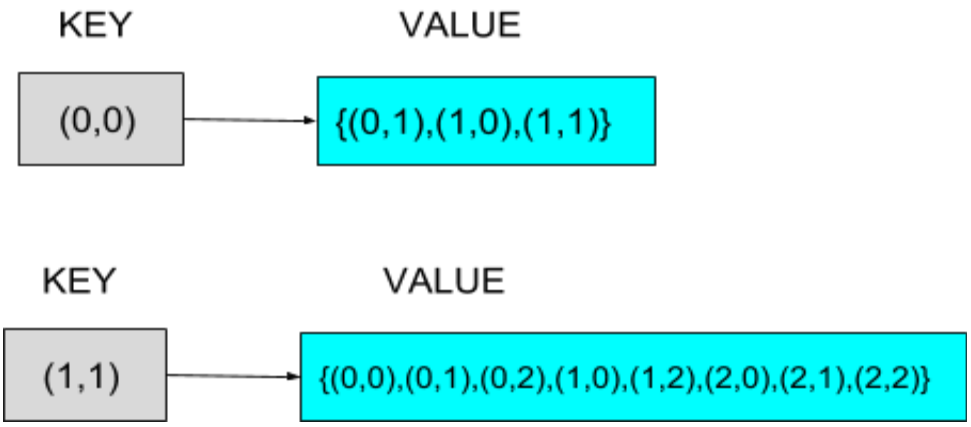


Figure 5: Neighbours_dictionary

- **Mines_nearby_dictionary**
  - This is just a simple dictionary where for every cell we store the clue information. Meaning how many mines are nearby after we finished process 4 above.

The main reason we have chosen to combine our board with these multiple data structures is that for every iteration it will be much easier to collect information for every cell and use the sets operations in order to make an inference about the surrounding cells.

The main algorithm is summarized in the next two points:

- For every cell value; a piece of clue of information, if all its neighbours that could be mines are already flagged, then the rest are considered safe to be opened.

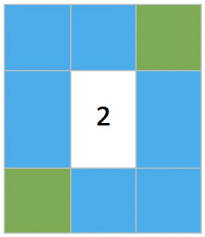


Figure 6: Green: Flagged, Blue: Safe

- For every cell value; a piece of clue of information, if its value is equal to the number of neighbours that are not opened, then the rest are considered to be suspicious; subsequently, all the rest will be flagged.

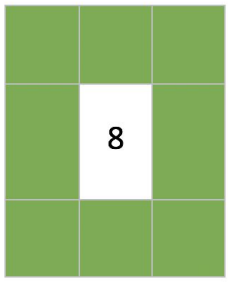


Figure 7: Green: Flagged

So the algorithm will first start with the *initial cell* , and all sets are initially empty. Open the initial cell, if it is a mine report dead and exit. Otherwise, open it and add it to the open set and check open all adjacent cells if they are zeros or clue and. For all neighbors for the initial cell, check the constraint mentioned above to either flag/open cells and add them to the designed sets. The algorithm will do all these steps recursively until there is no possible way to collect much information from all cells.

Let's see an example: for 9*9 board we have 10 mines.

Example 1: When the algorithm cleared the whole board.

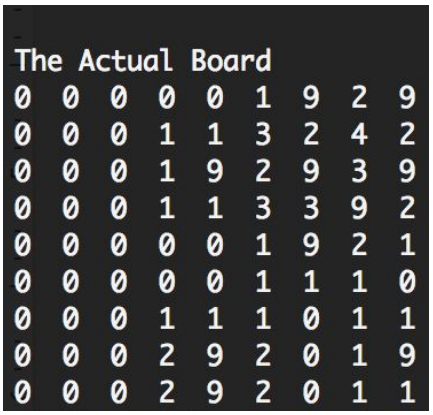


Figure 8: The actual Board

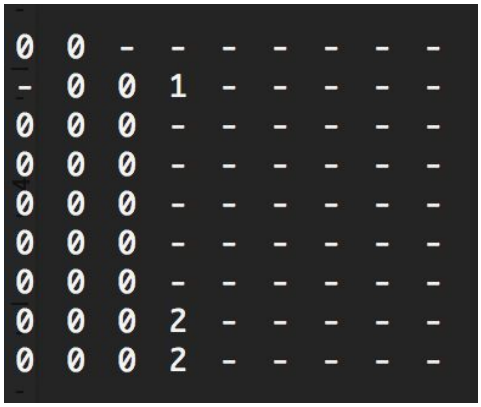


Figure 9: One of middle steps

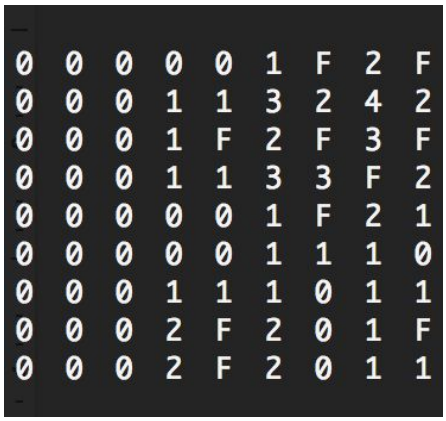


Figure 10: The last state in clearing the board

Explanation, thoughts and suggestion:

For most cases, our algorithm works perfectly fine when we have the mines fairly distant from one another. As seen from the example above, most of the grid values in the initial columns have a value of 0 which means there are no mines nearby, making it more easier for our algorithm to solve. The same trend was also seen when we increased the grid size to 16*16 and when we had 40 mines which shows that when we have 12-15% of the grid with mines, our algorithm has no problem solving it.

Example 2: When the algorithm cleared most of the board

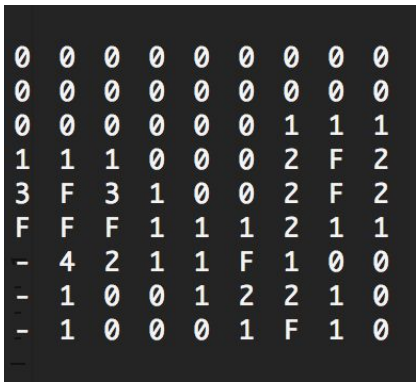


Figure 11: The last state in example 2

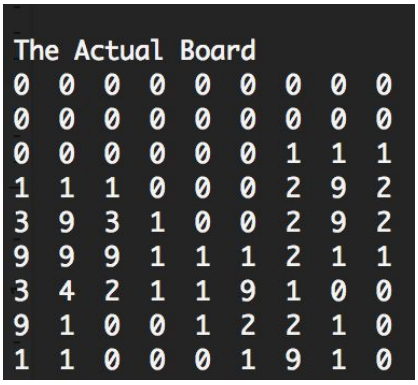


Figure 12: Actual board in example 2

Explanation, thoughts and suggestion:

In the above case, our algorithm stopped working when we have an uncertain situation. As seen above, all the three positions i.e, [6][0], [7][0], [8][0] have a chance of having a mine, our algorithm reached a conflicting situation and was forced out as a result. But when looked at from our perspective, we can see that due to the presence of 1 in both [7][1], [8][1], we can pretty much conclude a mine in either one of the positions to their left. Also, since [6],[1] with a value of 4 has already 3 mines flagged near it, there has to be a mine in either [6][0] or [7][0] to satisfy the value. So ultimately, based on our inference, we can conclude that there should be a mine with [7][0] which eventually results in solving the maze. The main point to look at here is that the algorithm could not solve a particular scenario which was actually solvable by a human. This can be improved by imputing our algorithm with probability based search. That is, say the value 4 at [6][1] will impute a probability of 0.5 in each of [6][0], [7][0]. And value of 1 in each of [7][1], [8][1] will again induce a probability of 0.5 in each of [7][0],[8][0]. In both cases, we can see that [7][0] has high probability of having a mine thus leading us to the result that this is the cell that has the mine. And based on the count of the number of mines, we could have declared that the rest of the unexplored cells were safe.

Example 3: When the algorithm is stuck in the first iteration

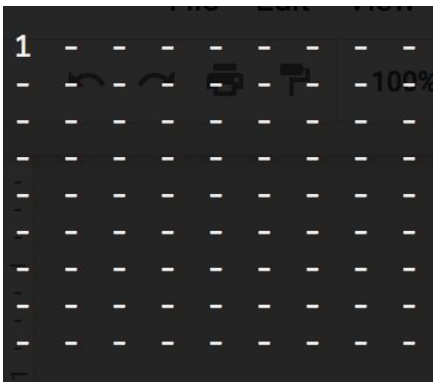


Figure 13: The last state in example 3

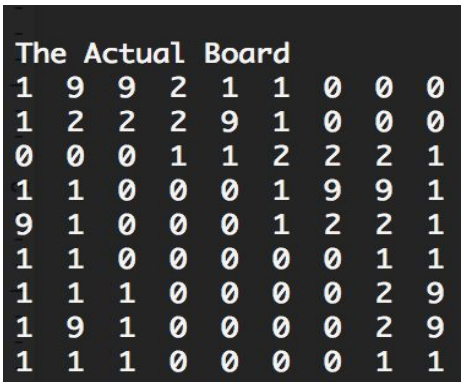


Figure 14: Actual board in example 3

Explanation, thoughts and suggestion:

The main problem in this particular scenario is that our algorithm did not take a decisive move as the probability of a mine in all of the neighbouring cells of the initial cell is the same. Since there is no other information in the knowledge base to infer from, it was forced out. As a human, we also may encounter a situation like this almost all the time. There is no guaranteed safe move at this point because of the limited information about the other cells. So in general, the ideal move at this point is to make a random selection from the neighbouring cells and go with it.
The drawback of this is that this random selection may yield a mine which results to game over but that is a possibility we have to risk. And that is exactly what our algorithm should also do if we were to overcome this particular scenario. This may not yield the complete result but if the randomly selected cell is not a mine, we get more information about the maze to work on and may be essential for solving this maze.

Example 4: When the algorithm is stuck after a couple of iterations

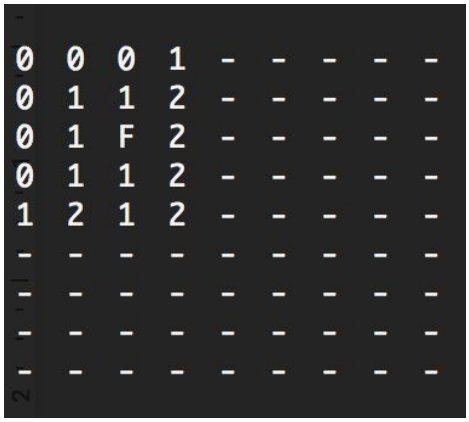


Figure 15: The last state in example 4

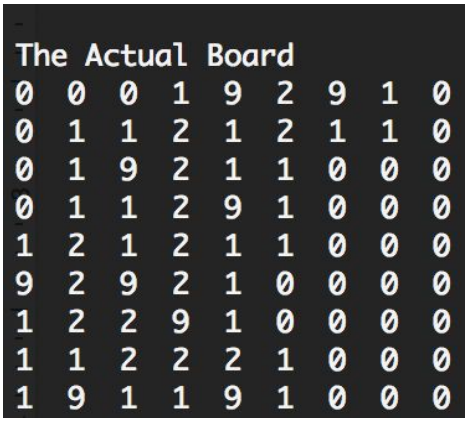


Figure 16: Actual board in example 4

Explanation, thoughts and suggestion:

This is also the same scenario as the last one but the difference being, the program is stuck after a couple of iterations when it reaches a conflicting situation of multiple cells having equal probability of having a mine. We can see that the probabilities are not actually the same for all the cells. From [4][0], we can see that both [5][0], and [5][1] have equal probabilities of 0.5 of having a mine. Then, based on [4][2], we can assume that [5][1], [5][2], [5][3] have equal probabilities of 0.33 of having a mine. Hence from these we can conclude that [5][1] has the highest probability but in reality, this is not the case. Thus even having the use probabilistic search may not yield the correct positions sometimes and thus random selection is the right way in this case too.

Decisions

There are two variants of the algorithm. The first variant does not allow a random movement if the algorithm doesn't know where to search next as it happens in example 3 and 4. In this variant, the algorithm will never step in a mine and in most of the cases it will clear most of the board. For the second variant, we are allowing the algorithm to choose a random neighbour cell of the current cell to discover and open all zero cells and the clue information around it and make them revealed. One potential problem can be that, this cell can be a mine, which ends the game. This is something we need to face it. But in an ideal minesweeper game, the scenario that we cannot deduce any further and that we have to choose a random cell, will be very rare. And the ideal game will not have a mine in this random cell.

Example 5: When we used a random search

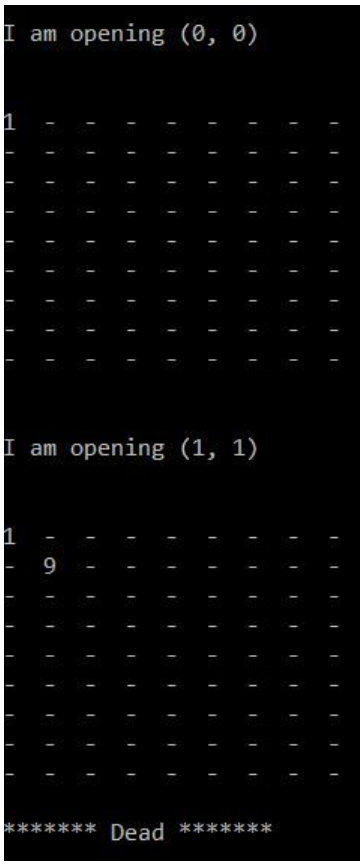


Figure 16: The last 2 states in clearing the board

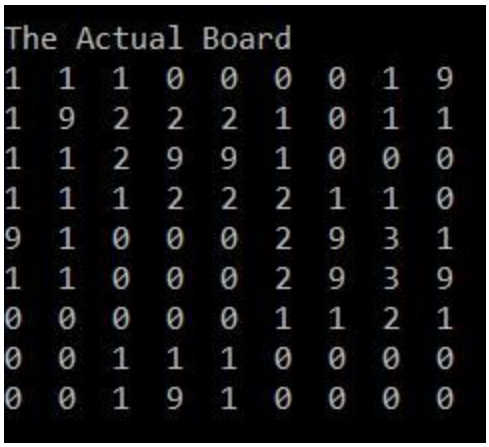


Figure 17: The actual board

Explanation, thoughts and suggestion:

This is a particular scenario where making the random search resulted in a mine forcing the program to end. This was because the conflict was making a choice between multiple cells having the same probability of having the mine. This is not an algorithm specific problem as it may yield a similar result when a human is playing as well. The main point to note is that, the algorithm is able to assess the situation and make a decision similar to what a human would take given this particular scenario.

Performance

Scenario (click here)

Explanation, thoughts and suggestion:

Attached is a particular scenario where the algorithm made decision by itself to solve the given maze completely. The parts of this solution that surprised us was line 547, 561. Basically, the program ended up in position where it had to guess between two cells with equal probability and then it randomly chose the cell which is not the mine which was really important to solve the rest of the maze. The take away from this part is that, it was completely random, and also there is a high chance for a human in that position to make a wrong choice. There is no logical reason for the algorithm to select the correct one as the program was designed to get all the neighbouring cells with conflicting probabilities and make a completely random selection among those and in conclusion, this selection is from a mere choice but that fact that it was the deciding point in the solution surprises us.

We ran the traditional game with the defined difficulty levels and obtained the accuracy levels of the algorithm as shown below:

Level of Difficulty	No. of maze solved out of 50	Accuracy
Beginner (9x9 with 10 mines)	5	10% ~ 15%
Intermediate (16x16 with 40 mines)	3	6%
Advanced (30x30 with 185 mines)	1	2%

.

Also, we ran the maze with significantly large number of rows and columns (50X50 Grid) keeping the mine density low (10% of the blocks are mines). We observed that the algorithm was able to solve the large maze with low mine density 50% of the times.

For our algorithm, the program is able to solve mazes were 15-20% of the maze has mines. This is because, the more the number of mines is increased, the less the number of cells with the value 0. This is crucial because having 0 valued cells can expose more safe cells in the maze which can in turn give more information about the board. As the number of mines increases, the number of cells with the value 0 decreases and also, the number of cells having multiple conflicting probabilities also increases thus leading to more than one randomized choices in each game. As suggested in the scenarios above,

the randomized selection works good for scenarios with high probability. As the number of mines increases, we have multiple mines with the same high probability which again leads to random selection among them. This iterative randomization has a very high bound failure rate because of its uncertainty.

Efficiency

For our implementation, we observed that the algorithm does not have any problem until the size of the maze is extremely large. It can solve 50x50 minesweeper maze with low mine density(10% mines). After 50x50, the computer runs out of memory or the system crashes. This is a problem-specific constraint as the Minesweeper problem is an NP-Complete problem. As the dimensions of the maze increases, the memory requirement will increase exponentially.

The problem that the algorithm faces is where it has to guess between two choices which have equal probability of being a mine. This is an implementation specific constraint as there will always be such scenarios where the algorithm will have to make a guess.

Improvements

When the user inputs the dimension of the board and the number of mines to place, with the original implementation being to start from the node [0][0], the program can be designed to start from the middle of the board (say width/2 and height/2) so that the initial chance of landing on a mine is decreased. Our implementation can be varied that this particular node that is in the middle of the board to be designed can have an initial check of whether that is the first click and can be modified such that the first click is devoid of landing on the mine. Moreover, having the program to select the middle of the board, even if it is holding a value other than '0', the probability of selecting a random cell is increased from ⅓ to ⅛. Again this also depends on the number of mines defined, but the upper bound of successfully completing the maze would be increased considerably.

Bonus: Chains of Influence

Based on your model and implementation, how can you characterize and build this chain of influence? Hint: What are some 'intermediate' facts along the chain of influence?

Every time, a new cell is discovered in the following ways. Every cell along the implementation is marked as either a mine, or a open cell based on some features defined in the Cell class. This class is also how we get the neighbouring cells as well as their status.

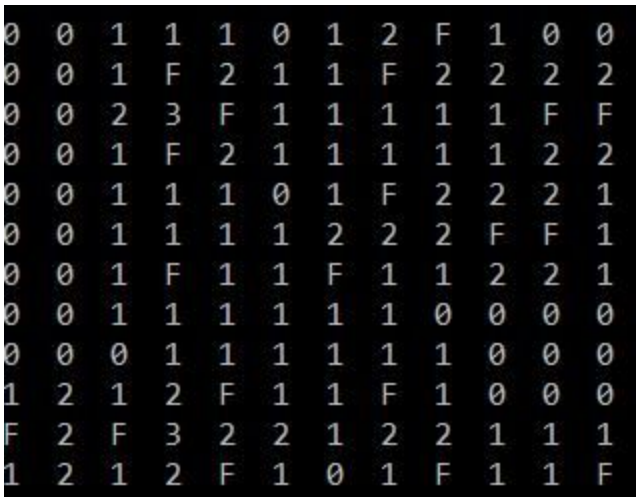
1. Initialise the algorithm by querying the first grid [0][0]
2. When the cell has a value of zero, all its neighbours are marked as safe and are iterated recursively to get more information.
3. When the number of unexplored cells around the active cell is equal to the number of mines, i.e., the value of the active cell, all the neighbours are flagged as mines and the loop backtracks.
4. When the number of flagged mines around a particular cell is equal to the value in the current cell, then all the neighbours are marked as safe and are iterated recursively.

Our algorithm follows these steps recursively to get all the mines from the maze.

What influences or controls the length of the longest chain of influence when solving a certain board?
The longest chain of influence depends on the number of mines defined. If the number of mines defined is very high, revealing a particular cell can be in such a way that it only influences the prediction of one other cell. In other words, if there is only one to one influence between a parent cell and child cell in a maze, the chain of influence will be longest. But in an ideal case, this is almost not possible, as if a mine has a value of '0', it influences to open all its surrounding neighbours thus having more than one path to explore.

How does the length of the chain of influence influence the efficiency of your solver?
Like mentioned in the above answer, the performance of the solver is increased if it has the longest chain of influence. The deeper the tree the better the performance. This is interpreted as, if every cell in the the maze influences only one child, then that means the knowledge base is defined perfectly and the solver is able to make accurate predictions for the mines and the probability of the solver running into a mine is drastically decreased.

Experiment. Can you find a board that yields particularly long chains of influence? How does this vary with the total number of mines?



The above example here has the largest chain of influence based on our algorithm. As the number of mines increase, it is more likely to break the chain on influence for our implementation. Also if the number of mines decrease, it may lead to short chain with more number of branches.

Experiment. Spatially, how far can the influence of a given cell travel?
Given the root cell [0] [0], the chain of influence can travel till the last cell that can be explored provided the designed knowledge base does not fail to evolve as it obtains more information. In our implementation, the ideal COF would be from [0][0] to [dim][dim].

Can you use this notion of minimizing the length of chains of influence to inform the decisions you make, to try to solve the board more efficiently?
By definition, decreasing the length of the chain of influence is nothing but decreasing the number of mines or increasing the number of branches to explore. Thus, having the count of the number of mines prior to the start of our solver, if we find all the mines before all the cells are explored, we can mark all the remaining mines as safe and end the implementation which makes the job much easier. Hence we can minimize the length of chains of influence to solve the board more efficiently.

Is solving minesweeper hard?

Based on the basic idea of a minesweeper game, to design an algorithm to solve for all mines in a board becomes challenging when the algorithm reaches a point where it has to query a random cell as it is not able to gain any more information using the knowledge base. Other than that, given a maze where the placement of mines doesn't impute conflicting probabilities on cells, solving minesweeper is not so hard at all.

Bonus: Dealing with Uncertainty

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.

We can use the concept of binomial probability distribution and Bayes' theorem to include information about the probability of surrounding cells. We express the surrounding cell value in terms of a decimal number.

If m = number of mines and c = number of cells then (c-1) choose (m-1) / (c choose m) gives the initial probability of the grid. For a 30 * 16 grid with 100 mines, the initial probability is (479 choose 99) / (480 choose 100) = 0.21.

When we reveal 1 as clue at the first click, then the probability of clue cells is 1/8 = 12.5% and the probability of all the remaining cells is 99/471 = 21.02%.

At the second click suppose we get 2 as clue adjacent to the first clue. We can have three configurations here, column 1 probability, column 2 and 3 probability, column 4 probability. Now the probability of adjacent cells is given by

a	b	b	c
a	1	2	c
a	b	b	c

For a for a 30 * 16 grid the final probabilities after second click revealing 2 are as follows:

0.05	0.21	0.21	0.39
0.05	1	2	0.39
0.05	0.21	0.21	0.39

This works well if we know the value of the clue cell. Thus working in tandem, probability and clue cell value together can efficiently solve minesweeper. However, in cases where we only reveal a clue about the surrounding cell in terms of probability and do not know the value of the clue cell (i.e., the actual number of mines in the surrounding cells), it becomes increasingly difficult to solve minesweeper only based on the probability values of surrounding cells, and the performance hit is extremely high. Because the game might end before we can even find and use the complete information from clue cells. Thus, it results in encountering mines more frequently and the game ends early.

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.

Whenever we uncover a clue, say for example 3, we can count how many remaining unknown cells there are, say 6,and predict with a uniform distribution that the true clue may be 3, 4, 5, 6 with each ¼ probability. In this way, it will be much more difficult to solve large puzzles because we will have to consider exponentially more scenarios for each clue.

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.

Whenever we uncover a clue, say for example 3, we can count how many remaining unknown cells there are, say 6,and predict with a uniform distribution that the true clue may be min(3, 6), min(3,6) -1, ..., 1 each with 1/(min(3,6) probability. Similarly, it will be much more difficult to solve large puzzles because we will have to consider exponentially more scenarios for each clue.