Fall 2018 CS 520: #2 Wes Cowan

Due by Oct. 26 11:55 pm.

GroupMember1: Yujie Ren ID: 167006818, NetID: yr181
GroupMember2: Lemin Wu ID: 171008940, NetID: lw514
GroupMember3: Guo Chen ID: 173006648, NetID: gc538

3 Questions and Writeup

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

The board in my program is defined as a class in java. This class has two main members and each of which is a 2D boolean array. One records whether a cell is a mine or not, the other records whether a cell is covered or not. And one fundamental method in this class is return the result of a query from user whether a given cell is a mine or not. If it's a mine, then game over, if it's not a mine, then it's returns a value which indicates the number of mines around this cell.

For my AI program, I use a Class ***Cell*** to store clues of a given cell, including the following: what is the state of current Class (Safe, Unknown or Mine); the number of surrounding mines; a list of all its surrounding cells; whether this cell is adjacent to an existing safe cell or not; the probability of this cell to be a mine. In addition, I defined three fringes: ***decision set***, ***mine set*** and ***remain set***. The ***decision set*** is a set (implemented in a LinkedList in Java) stores the cells that my AI program could either determine or guess it's safe to explore for the next round. The ***mine set*** is a set (implemented in a ArrayList in Java) stores the cells that my AI program is able to be 100% sure to be a mine. The ***remain set*** is a set (implemented in a minimum PriorityQueue in Java) stores the cells that are not yet explored.

• 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?

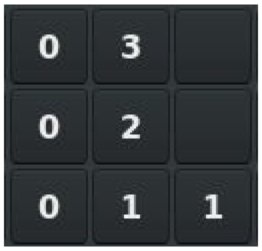


Figure. 1a

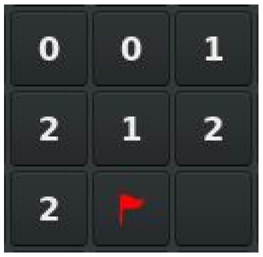


Figure. 1b

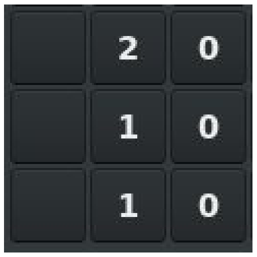


Figure. 1c

When a new clue is collected by my AI program, There's following different scenario:

- A query of a given cell ***A*** indicates that it's not a mine and a number of surrounding mines is returned. Now the AI program checks the surrounding cells of ***A***. As it shown in *Figure. 1a.* If the number of uncovered surrounding cells equals to the remain number of mines of ***A*** (A.uncovered == A.totalMineNum - A.alreadyCoveredMineNum), then all the surrounding cells must be mines, and AI program add them to mine set; They are absolutely mines.

- As it shown in *Figure. 1b.* If the number of uncovered surrounding cells equals to the remain number of mines of ***A*** (A.totalMineNum == A.alreadyCoveredMineNum), then all the surrounding cells must be mines, and AI program add them to decision set; They are safe to explore in the next few rounds.

- As it shown in *Figure. 1c.* If the number of uncovered surrounding cells greater than the remain number of mines of ***A*** (A.uncovered > A.alreadyCoveredMineNum), then we cannot conclude any valuable deterministic clues, so cannot AI. However, at least, we could conclude from *Figure. 1c.* that the probability of uncovered cells around the center cell to be a mine is ⅓, and my AI program will put this clue in each of these three uncovered cells for further use.

The above process is processed until my AI program cannot determine any cell to be clear or to a mine. The above inference seems to be complete, however those clues are inferences on a single cell. What could we do if we look at adjacent multiple cells? Could we get some additional clues? The answer is yes! And actually, I could easily list some examples.

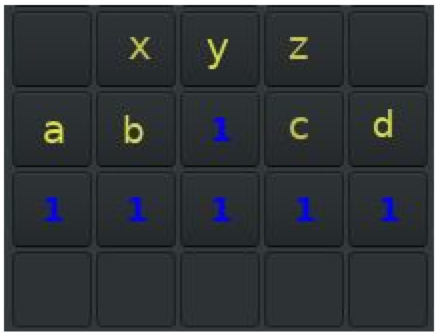


Figure. 2

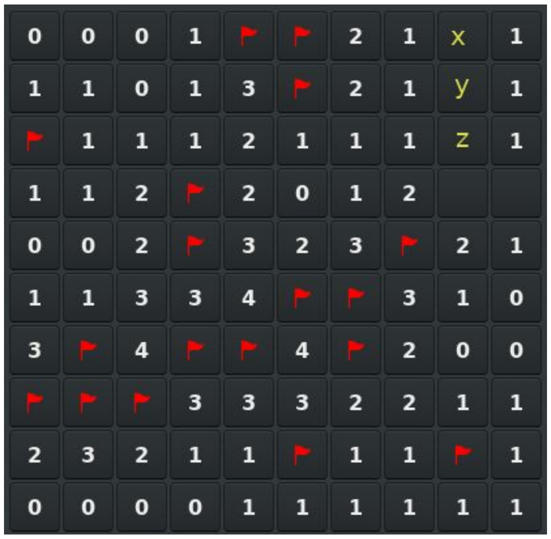
For the Figure shown above, we could conclude the following:

$$b + c = 1$$
$$b + c + x + y + z = 1$$

And we could conclude that x + y + z = 0. This indicates x, y, z are all clear. This is just one example of multiple cells inference. It's easy to see that if we add all the cells those are related, my AI program will be much more powerful. Now this problem becomes a ***CSP*** (constraint satisfaction problem). To implement this, two fundamental problems need to be defined. First, what is the criteria that two cells are related? Second, how could we solve these linear equations?

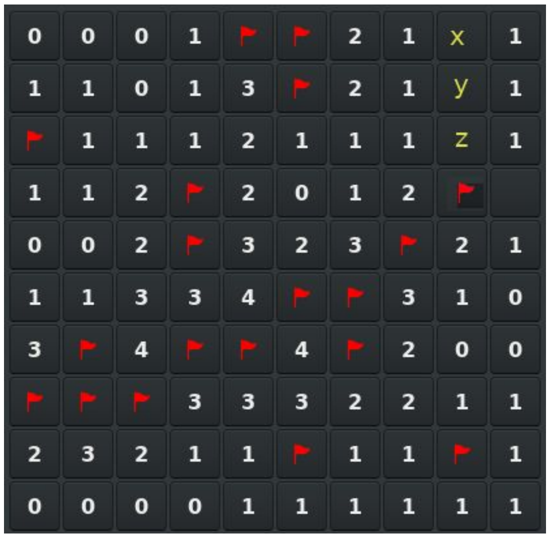
Two cells are related if and only if they have overlapped surrounding cells. To solve linear equations, we could use Gauss-Jordan Eliminations. However, general Gauss-Jordan Elimination failed to accomplish my goal for the linear equation defined in my case is quite different. First, my equation variables only have two values, 0 or 1. Second, negative values in an augmented matrix are not acceptable. To apply Gauss-Jordan Elimination in my case. I made several modifications.

Multi-cell inference is important and most of time it's the key to win a game. And it becomes dominant when the density of mine increases. Look at the following example.



0	0	0	1	🚩	🚩	2	1	x	1
1	1	0	1	3	🚩	2	1	y	1
🚩	1	1	1	2	1	1	1	z	1
1	1	2	🚩	2	0	1	2		
0	0	2	🚩	3	2	3	🚩	2	1
1	1	3	3	4	🚩	🚩	3	1	0
3	🚩	4	🚩	🚩	4	🚩	2	0	0
🚩	🚩	🚩	3	3	3	2	2	1	1
2	3	2	1	1	🚩	1	1	🚩	1
0	0	0	0	1	1	1	1	1	1

Figure. 3a



0	0	0	1	🚩	🚩	2	1	x	1
1	1	0	1	3	🚩	2	1	y	1
🚩	1	1	1	2	1	1	1	z	1
1	1	2	🚩	2	0	1	2	🚩	
0	0	2	🚩	3	2	3	🚩	2	1
1	1	3	3	4	🚩	🚩	3	1	0
3	🚩	4	🚩	🚩	4	🚩	2	0	0
🚩	🚩	🚩	3	3	3	2	2	1	1
2	3	2	1	1	🚩	1	1	🚩	1
0	0	0	0	1	1	1	1	1	1

Figure. 3b

At Figure.3a, my AI program encounters a dilemma that the cannot proceed any more. To dig more clues, I add multi-cell inference on the ***cell[0][8]*** and ***cell[0][9]*** in the above figure, and get the following equations:

$$x + y = 1$$

$$x + y + z = 1$$

And then the AI program get a new clue that ***z*** must be clear. Since ***z*** is clear, then ***cell[3][8]*** must be a mine according to the surrounding clues get by ***cell[2][8]*** as it shown in Figure.3b. Then the game is easy to solve.

• 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?

As I illustrated above, my AI program maintains a fringe called ***decision set*** in which stores all the cells to be safe determined by my AI program based on clues or some cells that are likely to be safe. My AI program follows a strategy that the AI will not make any guesses until all safe cells in the ***decision set*** are uncovered. If there are no more safe cells in the ***decision set***, the AI program will extract a cell from remain set. The ***remain set*** is implemented in a minimum priority queue sorted by the probability of a cell to be a mine. Thus, if the AI program must make a guess to continue, it will pick the one with minimum probability to be a mine to uncover in the next step; on the other side, the ***decision set*** is a FIFO queue, such data structure also guarantees the cells determined to be safe will be uncovered before those are guessed to be safe.

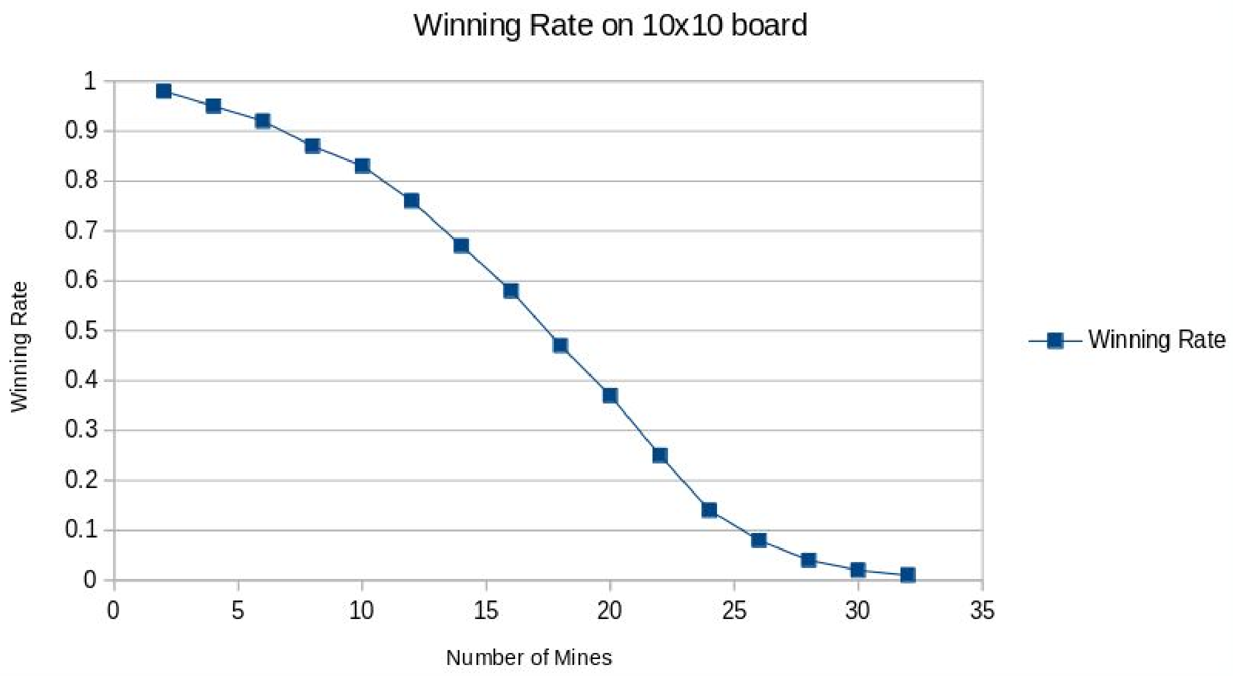
Yes, there are risks. As I said, if the AI program is facing a situation that all the clues have been digged out and it cannot keep going any further, it will take a risk to make a guess by picking a cell with minimum probability to be a mine from ***remain set***. The probability of each cell to be a mine is updated very often. Even if this probability model is simple, it's very effective.

There is a thought that when AI program cannot proceed any more, it should return the decision making back to human. However, in this case, I don't think it's a good idea. Because when the board is very large, my AI program could record the probability to be a mine of every cell in the remain set, and pick the minimum one to uncover in the next round; In addition, these probabilities are updated very often. It might not be so easy for a human to record and recalculate all of them every time. The most important reason to support my claim is that picking a minimum one to uncover next is the most rational decision in this case.

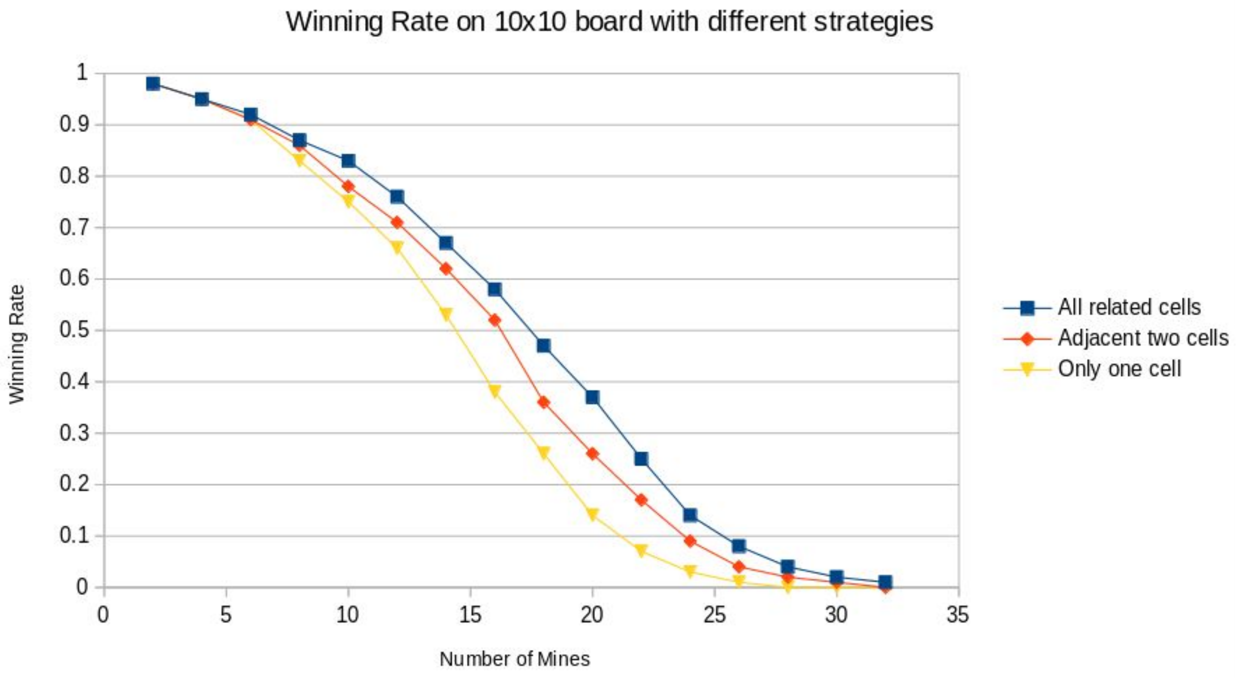
• Performance: For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?
My AI program hardly makes any decision that I don't agree with. As I illustrated above, my AI program roughly contains two components: deterministic solver and uncertain solver. The deterministic solver will uncover all cells that are guaranteed to be safe based on clues (one-cell logical inference and multiple adjacent logical inference). Once there's no clues left, the deterministic solver cannot proceed, nor could human. Then, the uncertain solver will take risk to make a guess. Sometimes, it will make a wrong guess and the game is over. However, I think it's the best that a MineSweeper AI program could do. Just consider my AI program, even if the uncertain solver made a wrong guess, at least it make a most rational decision - picking a cell with least probability to be a mine. I was thinking what will I do if I facing such situation. Probably, I'll make a random guess. Apparently, my decision is less admissible than that from my AI program.

Since my AI program will always make the most rational decision, it NEVER surprise me. This is both a good thing and bad thing. The good thing is my AI program will hardly make any mistakes when the clues are already there while human even make silly mistakes from time to time; However, the bad thing is that I, in some aspects, cannot say my MineSweeper AI program is intelligent. My AI program could make decisions, and these decisions are made based on some clues. However, it is just like a normal computer program and the decision it could made is pre-programmed; it do some basic logical inferences as I told it. So in this aspect, I would say, it doesn't have the ability of thinking and learning.

• Performance: For a fixed, reasonable size of board, what is the largest number of mines that your program can still usually solve? Where does your program struggle?
We made several experiments on this questions. The board size we pick is 10*10, and we made the mine number from 2 to 32 (stepping by 2).



The experiment result is self-explainable. The winning rate will decrease as number of mines goes up. This figure is not very interesting. We also made another experiment, we use different strategies of analyzing clues in my AI program. We get the following result:



The simplest method uses only one cell's clue as a constraint; An enhanced method is using graphically adjacent two cell's clues; And the most powerful method is using all of the related cell's clues. From the experiment result, we could see that when the mine number is small and sparse, those three methods doesn't make much difference in winning rate. This makes sense because if number of mine is small, then mines is very likely to be distributed sparsely on the board. Therefore, multi-cell's clues are very less likely to be related to each other. However, when the mine density goes up, it's intuitive that multi-cell's clues will influence each other. Thus there must be much more valuable clues if we add more related cells' clue together as a constraint.

• 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?
Since DRAM and storage size increases, space are no longer a big problem for most programs. For my AI program, there is only limited space used. As I said above, my AI program maintains three sets: ***decision set***, ***mine set*** and ***remain set***. The sizes of those sums up to be the number of cells on a board. And to solve the CSP, I use a modified Gauss-Jordan Elimination approach. The space to store the matrix is very limited and could be freed after each iteration. So, in general, my AI program will not consume a lot of memory space.

As for time complexity, it's also easy to analyze. One important observation is that each cell will be accessed no more than the size of the board. One extreme cases is that an arbitrary cell will be accessed again whenever other cells is being uncovered. Thus, given the board size ***n***,the time complexity for my AI program is ***$O(n^2 * n^2)$*** = ***$O(n^4)$***. This is polynomial time.

Since I formulated a CSP problem to a linear programming model, and use Gauss-Jordan Elimination to solve these linear equations, my AI program is more efficient than those brute-force methods in both time and space. One thing I could improve in my AI program is the following. When determining which cells in ***remain set*** are related to each other, I use a two nested loop for each cell, if any two cells are related, I join them together in a data structure. Such two nested loop will make the join set contains duplicated sets. Since I don't have much time left on this project, I didn't make optimization on this part.

• Improvements: Consider augmenting your program's knowledge in the following way - when the user inputs the size of the board, they also input the total number of mines on the board. 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.
Actually, acknowledging the total number of mines on the board is useful. When sweeping the board, the AI program could record how many remaining mines left given the total number of mines. It could be used in the following two situations:

- If the number of remaining mines equals to the number of remaining cells. All the remaining cells must be mine. If the number of remaining mines equals to zero. All the remaining cells must be clear.
- When the sweeping process is near the end, the remaining mine number could be used as one additional constraint. And for some game, such additional constraint is the key point to win the game.

My AI program already implemented the features above.

4 Bonus: Chains of Influence

For any cell that is marked clear or mined, we can think about the cells that led to or informed that conclusion. In Fig. 1 above, Cell (0, 0) [upper left corner] containing the clue 3 is enough to determine the value of Cells (1, 0),(1, 1),(0, 1). Cell (1, 2) on the other hand (depending on your specific implementation) depended on Cell (2, 4) which revealed the values of Cell (1, 4) and Cell (1, 3) - these, combined with the clue in Cell (2, 3), was enough to mark the value of Cell (1, 2) as clear.

In this way, for every cell we can determine the value of, we can construct a chain of influence that led to the marking - starting at revealed cells, cells those allowed to be marked, and working through the chain of inference to the conclusion for the value of a certain cell. Every new cell will inherit the chain of influence of any cell that influenced it - you can imagine this as a very large directed acyclic graph branching out from the initial cell you reveal (or any unjustified / guessed cell you reveal).

• 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?
Intuitively, the chain of influence is equal to the number of necessary 'intermediate' facts a reasoning process on the game board would need, which are about essentially constraints of some cells to be mined or clear, to mark a cell to be mined or clear in certain, starting from a given revealed cell. In the minesweeper game, a player can firstly randomly select a cell and check if it is clear or mined. If it turns out to be clear (the player would lose the game otherwise), then the player obtain one piece of fact from number in the cell, which reveals how many adjacent cells are mined. Then based on this fact the player may be able to discover more cells around the starting one, obtaining more such facts such that more cells can be marked as clear or mined in sure, and so on so forth. Finally, as sufficient 'intermediate' facts are discovered by the player, most cells can be determined to mark as mined or clear, till the end of the game.

Specially, based on the rule of the game, we find several cases where a single revealed cell can help make more cells to be revealed with 100% certainty, or in other words, generate more useful 'intermediate' without necessity of checking any other cells:

1. The revealed cell indicates no mines in all of its adjacent cells, with its number being zero;
2. The number of unmarked adjacent cells is equal to the number of mines the revealed cell indicates, which means all of them are surely mined;
3. The number of adjacent cells marked as mined is equal to the number of mines the revealed cell indicates, which means all unmarked adjacent cells are surely clear.

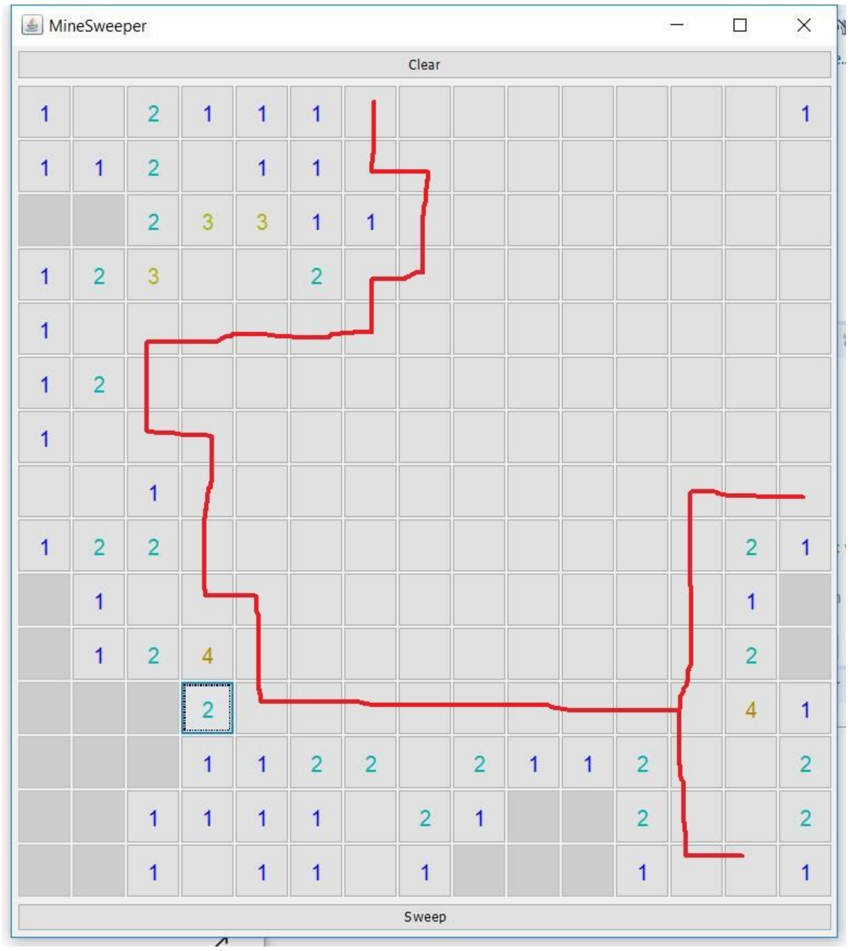
• What influences or controls the length of the longest chain of influence when solving a certain board?
Intuitively, we find three factors that impact on the length of the chain together:

1. The size of the game board;
2. The number of mines hidden in the board;
3. The distribution/layout of the mines in the board.

• How does the length of the chain of influence influence the efficiency of your solver?
As in the first problem, we've known that the length of chains of influence somehow indicates how much information is needed to determine a certain cell to be mined or clear, besides revealed information obtained from the starting cell. In this case, the longer the length becomes, more information is needed to make clear conclusions. Therefore, the longer the length becomes, the solver would find itself less efficient to work because it would have to spend more time collecting information rather than making solutions directly.

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

We found a chain of influence which involves 38 cells to be marked. The game board, which is as large as 15 * 15, is shown as below.



Suppose we begin at the cell located at (0, 6). If this cell is determined, then we can go down to determine the cell at (1, 6) since all surrounding cells has been explored and they are all determined. In this way, we can go along the chain of influence and make determinations.

As for the relationship between the number of mines and the length of chain of influence, the later one increases as the former one goes up. Intuitively, as the number of mines goes up, then it becomes harder to mark the cells because it becomes more likely to “step” onto a mined cell. In this case, it becomes more critical to draw accurate conclusions. In order to do so, more information is needed to determine cells. Therefore, with information needed for making decisions increases, the chain of influence would expand accordingly.

• Experiment. Spatially, how far can the influence of a given cell travel?

In the extreme case, the influence of a given cell can travel all the game board, which means all 'intermediate' facts depend on such a cell to determine all unknown cells to be mined or clear. One possible example is shown as below, which is a game board with size 1 * 20. In this game, only half of the cells are determined to be not mined while all others are undetermined at all.

a	1		1		1		1		1		1		1		1		1		1

In this case, we can easily see that we cannot determine any unmarked cells until the very first cell a is determined. If a is determined to be mine, then the states of all undetermined cells are (C, M, C, M, C, M, C, M, C). Otherwise, the states of all undetermined cells are (M, C, M, C, M, C, M, C, M). Without the cell a being determined, these two possible solutions are satisfiable.

• 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?
The key point here is to minimize the length of chains of influence, which means we need try not to introduce more information to determine certain cells. In this case, it equally means that we need to make use of previously obtained 'intermediate' facts better and expect them to generate more useful 'intermediate' facts for reasoning process. From this point of view, we will utilize local search as more often as possible, which means we restrict the length of chain of influence in reasoning process to be 1.

• Is solving minesweeper hard?
The level of difficulty for the game depends on how many mines are placed in the game board and how they are distributed. Ideally, a solver can work much more efficiently in a game board with 'sparse' mines. On the other side, however, more information needs to be collect to make clear decisions, in other words, the solver can work in an much less efficient way. One example for this is that it may encounter a such scenario: there is no way to make reasoning process anymore since no certain facts are there, an enumeration method would be carried out instead with a exponential search space.

5 Bonus: Dealing with Uncertainty

My solutions for 3 uncertainty models are implemented based on the version including only probability part (excluding solving equations). In the original probability version, every unknown neighbor cell's probability of being a mine is decided according to 1)if the neighbor cell's probability is 0, keep it 0. 2)if the neighbor cell's probability is calculated by the marginal cell for the first time, assign the probability to the cell no matter its value, 3)else only assign the probability to the cell when it's larger.

For the 1st uncertainty model (When a cell is selected to be uncovered, the information received is accurate, but it is uncertain if the information will be received) an additional input, uncertainProb, is added to the model to indicate the probability of telling the clue when a clear marginal cell is selected. Also, a new parameter, info = uncertainProb*number of unknown neighbor cells, is used to indicate the amount of information will be revealed if the cell is selected. The main logic of this solution is similar as the original probability version. The difference is that we choose the cell with the lowest probability of being a mine and highest information value as the next cell to dig. For 20*20 maps with 20% mines, there are about 5% winning rate for 100% sure to tell the clue, 1% winning rate for 90% sure to tell the clue, 0.1% winning rate for 80% sure to tell the clue. For 5*5 maps with 20% mines, there are about 35% winning rate for 100% sure to tell the clue, about 32% winning rate for 90% sure to tell the clue, about 28% winning rate for 80% sure to tell the clue, about 6% winning rate for 50% sure to tell the clue, about 8% winning rate for 30% sure to tell the clue. There seems to be a trend that the winning rate decreases when the probability of telling the clue decreases in the range from 1 to 0.5 and increases afterwards.

For the 3rd uncertainty model (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)), I plan to change the assigning probability to the sum of original probability*the probability that the original probability is true. For example, originally 0.8 will be assigned to the 5 neighbor cells with a 4 clue. In this solution the assigning clue will becomes 0.8*0.5+1*0.5 = 0.9. And the similar way of calculating probability can also be used in the 2nd uncertainty model (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)).