CS 520: Assignment 2 - MineSweeper, Inference-Informed Action

Haoyang Zhang, Han Wu, Shengjie Li, Zhichao Xu

October 28, 2018

1 Introduction, group members and division of workload

In this group project, we implemented a minesweeper solver that far exceeded our expectation. Not only can our program solve the normal minesweeper, but also it can solve minesweeper with inaccurate information. Our program also has a gorgeous GUI and can visualize the progress of solving minesweeper by animation.

Name RUID	Workload
Haoyang Zhang 188008687	Implemented the minesweeper solver. Finished the writing of report for most of the questions.
Han Wu 189008460	Ran tests and generated figures for question 2.4. Finished the writing of report for question 2.4
Shengjie Li 188008047	Designed and implemented the GUI of our program. Implemented a function that can generate animation of the progress of solving minesweeper. Finished the format design of whole report using LaTeX.
Zhichao Xu 188008912	Proofread the report. Ran tests and generated figures for question 2.5 and question 4.1. Finished the writing of report for question 2.5 and question 4.1.

2 Questions and Writeup

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

   In our program, the basic board is represented by a matrix, which is constructed by a 2-dimensional array. For example, a 64×64 board is represented by an array of 64 elements, and each element is an array with length of 64.

   The board is represented by 4 matrices: "covered", "flag", "_mine" and "_clue".

   - The matrix "covered" saves exploration status.
   - The matrix "flag" records blocks that agents regard as mines.
   - The matrix "_mine" represents the distribution of mines.
   - The matrix "_clue" is a preprocessed matrix that records each block's hint number. The hint number here is used to save time when solving minesweeper.

   The knowledge base is represented by 9 matrices: "covered", "safe", "flag", "hint", "warn", "left", "done", "nebr" and "prob".

   - Matrix "covered" and "flag" are the same as the board's matrix.
   - Matrix "safe" records the blocks that agents regard as safe blocks.
   - Matrix "hint" saves the hint number the agents have got from the minesweeper.

- Matrix "warn" represents the number of each hint block's un-flagged mines.
  Namely, warn = hint − (the number of neighbor flags).
- Matrix "left" represents the number of each block's inconclusive neighbors.
  Namely, left = 8−(the number of neighbor flags + the number of neighbor uncovered)−(edge cost).
- Matrix "done" records blocks whose neighbors are all conclusive. Such blocks could be ignored when agents are solving the minesweeper.
- Matrix "nebr" is a hash value matrix. It records the inconclusive neighbors of each block. Agents use it when trying to solve equations like "$A + B + C = 2$, $A + B = 1$".
- Matrix "prob" is actually a tensor. It records each block's probability of being a mine block, given each neighbor's data. Therefore, the size of "prob" is $rows \times columns \times 9$ (with an extra basic probability).

2. 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?

   There are at least 3 things we need to do:

   (a) Compute "warn" value of the block we have just explored.

   (b) Update all neighbors' "left" and "nebr" value.

   (c) Update all these 9 blocks' neighbor's "prob" value.

   If the updated "prob" value of a block is 0 or 1, it is now conclusive. Therefore, we add this block to the waiting line, to be explored or marked flag in the future.

   Sure that there are many other things we might make deductions, for instance, trying to combine it with a clue we have already known. However, there is a reason for stopping combining clues. Combining clues is especially time-consuming without a proper direction or heuristic. For a given knowledge base, the statements we can deduce do not change, even if we add more hints into the knowledge base. Therefore, it is not economical to spend so much time combining clues when we can explore and make deductions from somewhere else, because new hints may easily lead to exactly the same statements.

   If we have nothing else to do, there are still several ways to combine them (the detailed introduction is in "\docs\Solution Algorithm Explanation.html"). The structure of some hints (value relationship and geometric relationship) tends to be solvable, and then we can focus on those structures. At least, we can use proof by contradiction to combine a large number of hints at one time, though it is also time-consuming.

3. 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?

   There are mainly 4 things we can do:

   (a) If there is a block in the waiting line, process it.
   This brings no risk and is the easiest thing we can do.

   (b) Use pattern recognition method to find a group of blocks that usually solvable, and try to solve it.
   This performs surprisingly well, and takes not that much time, comparing with proof by contradiction.

   (c) Use constraint satisfaction and proof by contradiction to find a conclusive block.
   Pattern recognition could miss complex solvable structures, and therefore, we use the former one to improve. But generally, it takes too much time. Also, it has limitations. For example, it chooses some blocks randomly to search rather than choose all inconclusive blocks. Note that this limitation also leads to the missing of some solvable blocks.

(d) Should all attempts above fail, find a block that is worthy to take the risk to open.
The block should be either with the lowest risk, or "well-paid". A "well-paid" block is a block whose clue is expected to play an important role in the solving process in the future if the block itself is not a mine.

4. 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?

   When we check the play-by-play progression of the game, we can find some interesting phenomena. The algorithm made many decisions to solve the problem. Some of them are consistent with human beings' decisions, while some of them are not and surprised me (I am a human being). Let's take a 10×10 board as an example.

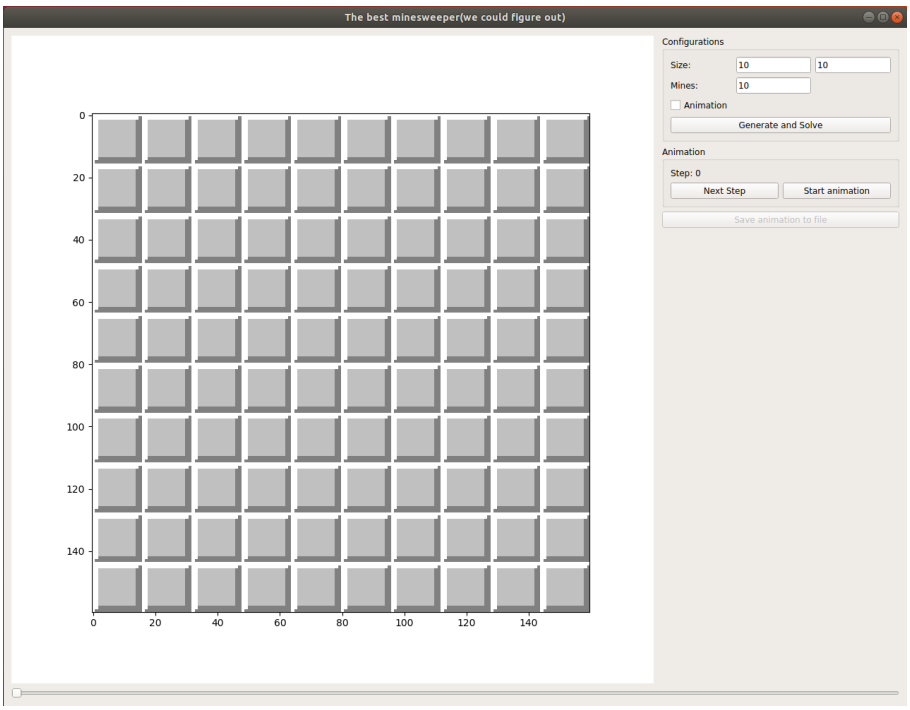

Figure 1: A 10×10 board.

(This screenshot is taken after we have done the experiment and the GUI has been changed since then, so the screenshots below may look a little different from this one.)

As Figure 1 shown, it is a 10 by 10 board. There are 100 cells on the board. We use (i, j) to represent the cell in the board. "i" represents the row of the cell and "j" represents the column of the cell. For example, (1, 1) represents the cell in the upper left corner and (10, 1) represents the cell in the lower left corner. The coordinate of the cell which is uncovered in this figure is (10, 6).

The first decision which the algorithm surprises me is step 13. The state of step 12 is shown below as Figure 2.

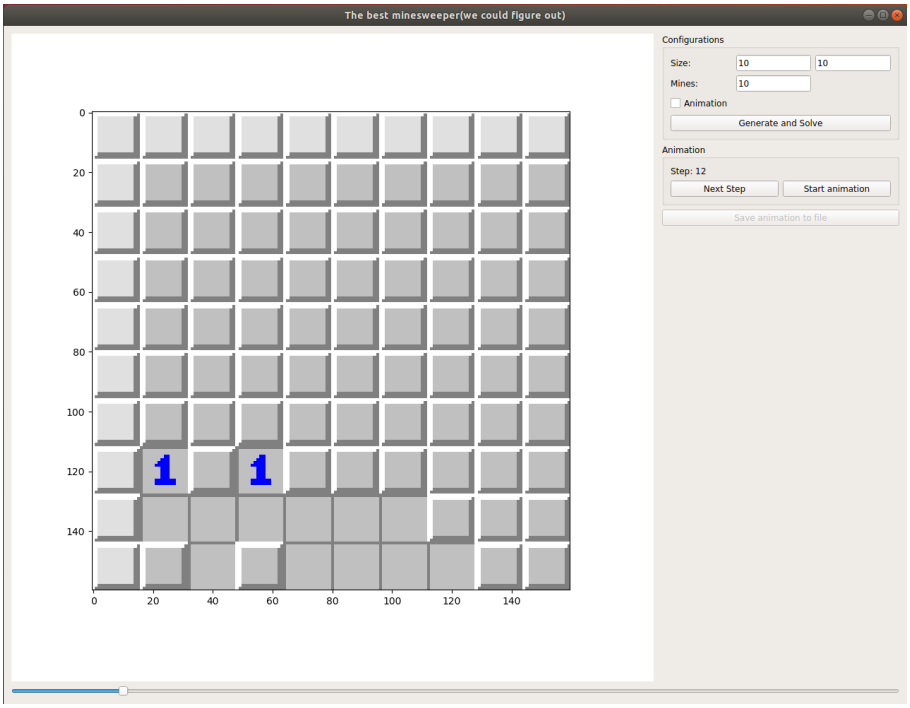

Figure 2: Step 12.

There are already two "1" on the board. As a human being, I choose to explore the cells near two "1", because exploring more cells there can help me find the position of mine and mark it. However, the algorithm chose to check (10, 9) in the next step, which is shown in the figure below as Figure 3.

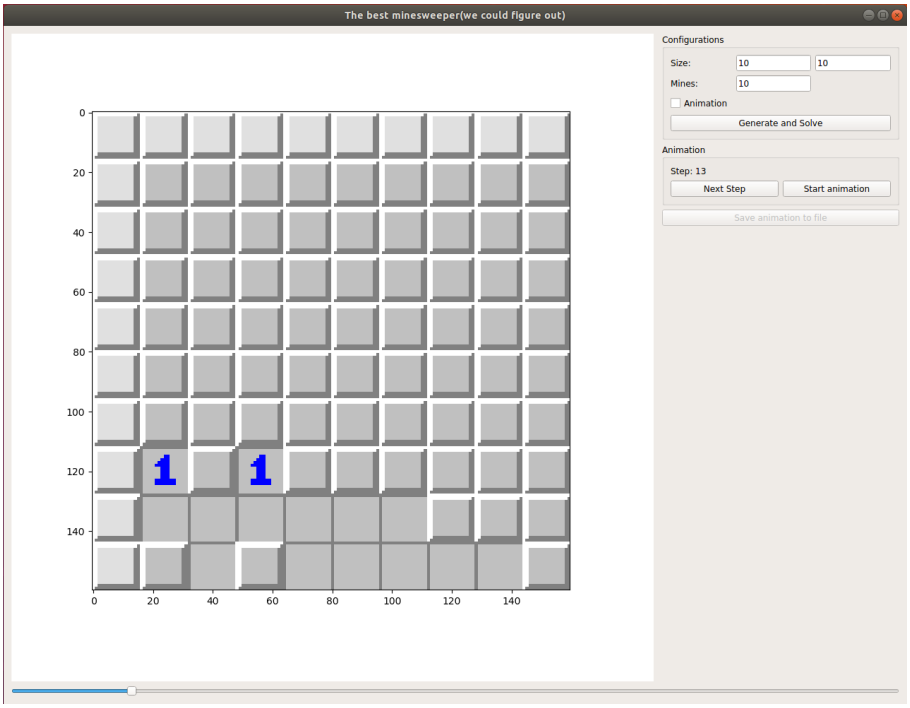

Figure 3: Step 13.

Cell (10, 9) is far away from the two "1" on the board. It is safe because no number is shown on (10, 8), but it provides no help to find the mine near two "1". In fact, the algorithm chose to uncover (10, 4) in step 14. The choice is not a bad choice but the algorithm doesn't know about focusing on problems, which is different from human beings and surprises me.

Another thing that surprises me is in step 16 and 17.

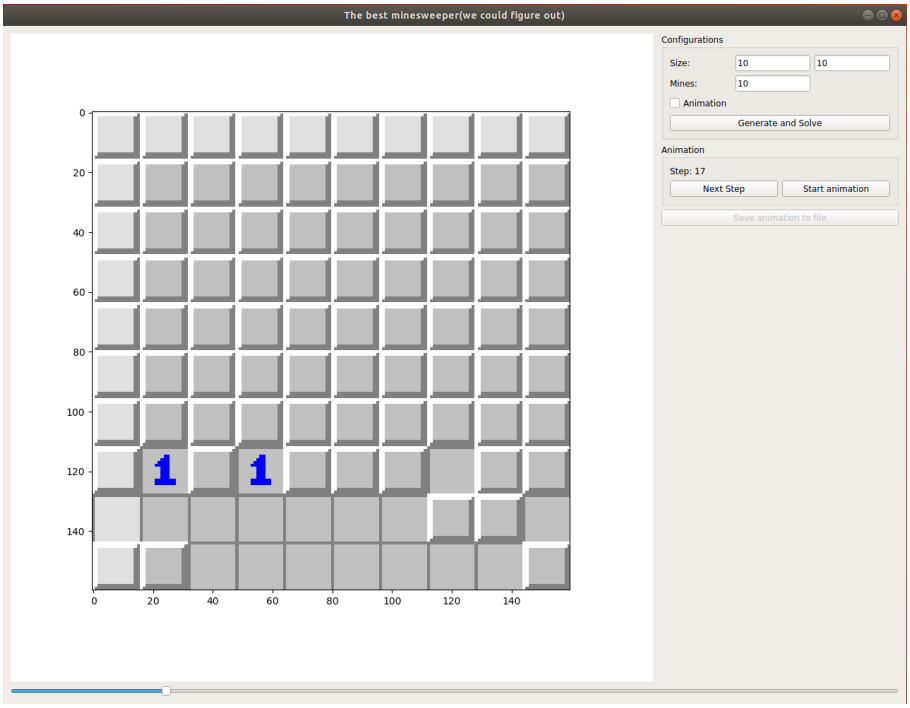

Figure 4: Step 17.

The algorithm uncovered (8, 8) in step 16 and (9, 10) in step 17. Because of (9, 7) and (10, 9), we know that these two cells are safe. However, these two cells are not close to each other. I, as a human being, will not do like that. Human beings often uncover the neighbors of one cell and then continue to uncover the neighbors of the neighbors if they know they are safe. The randomness of this algorithm surprises me.

Sometimes, the algorithm also made some decisions that I do not agree with. In step 42, the algorithm uncovered the cell (7, 1). We can conclude that (7, 3) is a mine according to the "1" shown in (8, 2). I will mark (7, 3) in the next step.

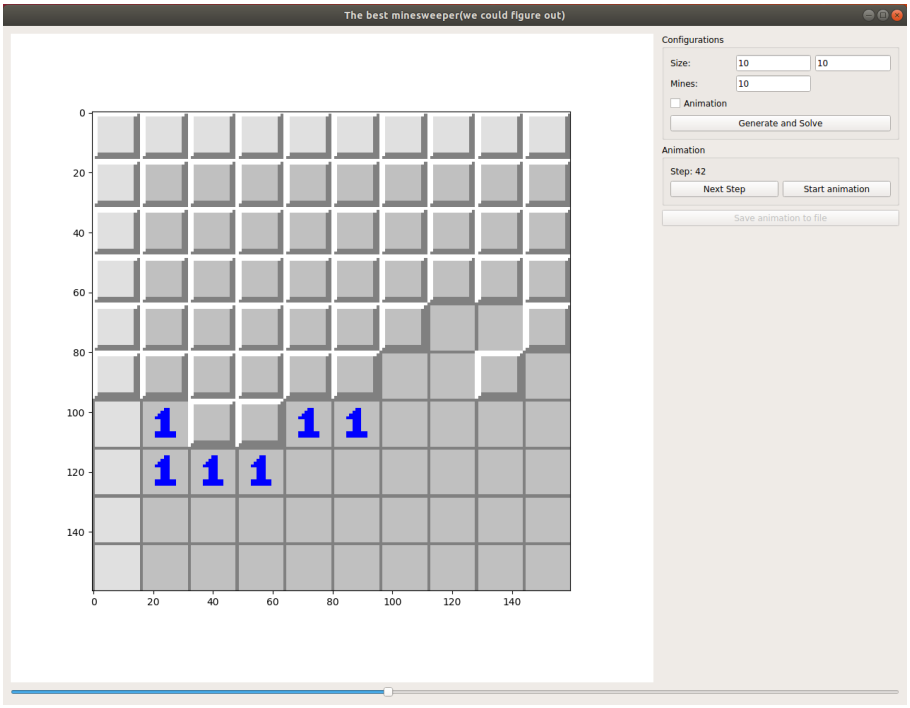

Figure 5: Step 42.

However, the algorithm explored (4, 10) in the next step. It didn't mark (7, 3) as mine until step 64.

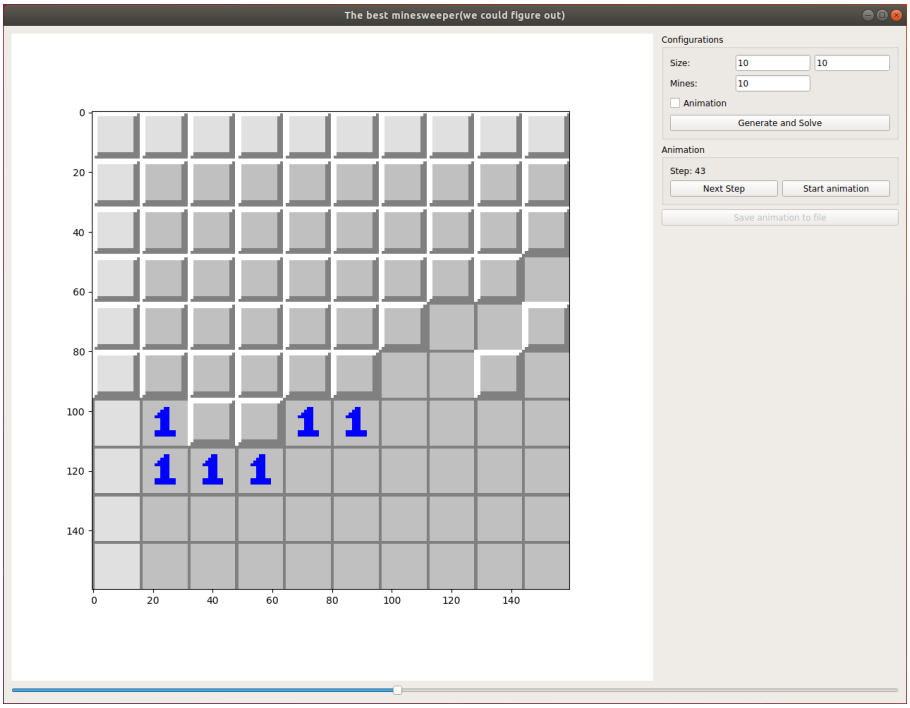

Figure 6: Step 43.

The same situation happened between step 50 and step 51. In step 50, we can find that (6, 5) is a mine. However, the algorithm chose to check (3, 10) in step 51. The phenomenon is shown in

Figure 7 and Figure 8.

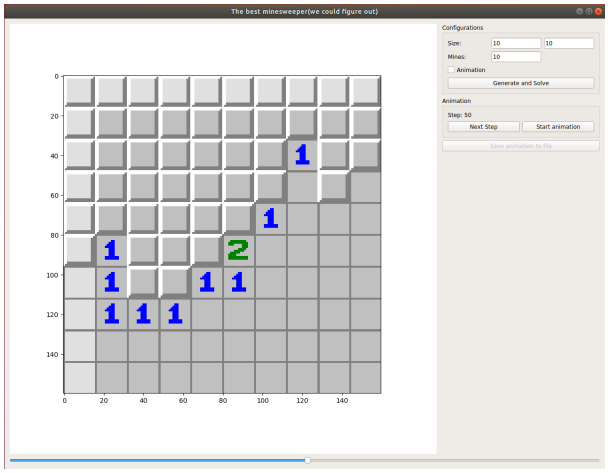

Figure 7: Step 50.

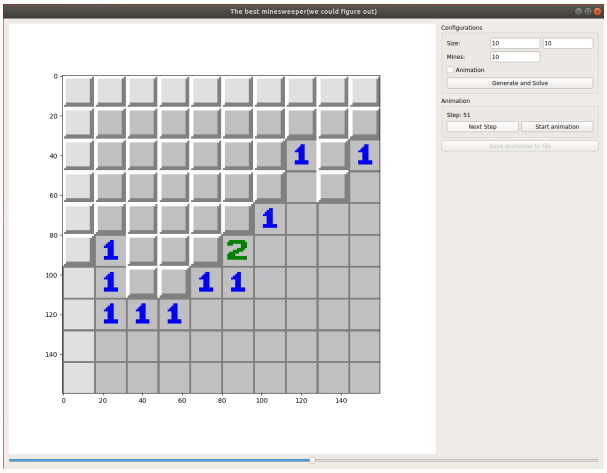

Figure 8: Step 51.

In step 58, we can conclude that (6, 5) and (5, 5) are mines according to "2" in (6, 6). However, the algorithm chose to explore (3, 9) in step 59. The phenomenon is shown in Figure 9 and Figure 10.

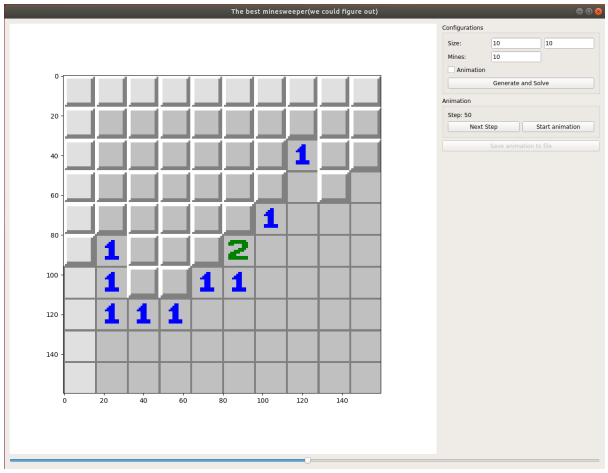

Figure 9: Step 58.

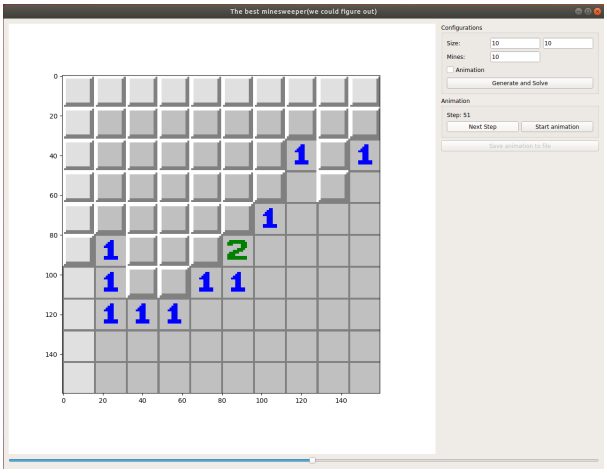

Figure 10: Step 59.

Cell (6, 5) is marked as mine in step 63 and Cell (5, 5) is marked as mine in step 65.

Sometimes, the algorithm does something that surprises me. In other times, it does something that I think is not appropriate. This is the difference between the algorithm and human beings.

5. 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 used the size of 64×64 as the board size. From our preliminary testing result, our program can solve board of less than 20% mine density easily. When the mine density is over 20%, it usually takes a much longer time. The threshold for the board to become not solvable for our program is 23%. When the mine density is over 23%, nearly all the boards are not solvable for our program. We increased the number of experiments, setting mine density to less than 20%, the result is showed in Figure 11.

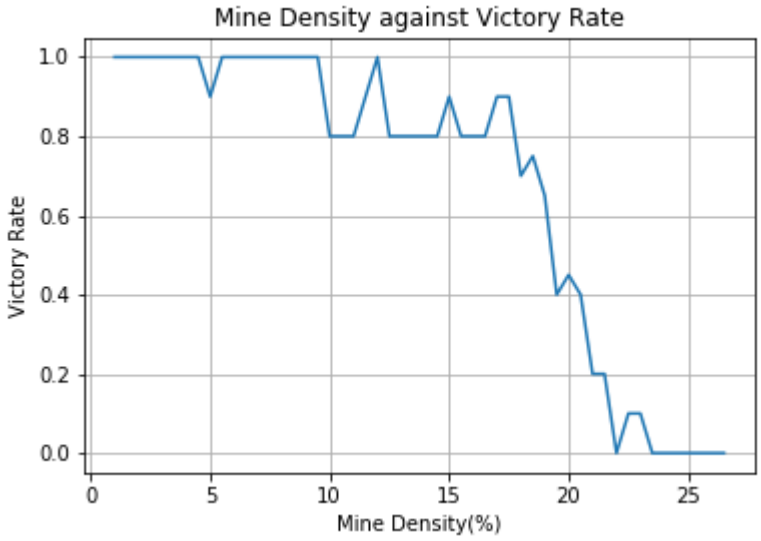

Figure 11: Mine Density against Victory Rate.

We have come to the conclusion that our program can confidently solve boards of less than 20% mine density. As mine density increases, it takes more time to make a deduction. At the same time, there are more unsolvable scenarios where we have to take risks to explore new blocks. That's where our program begins to struggle.

6. 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?

   Overall space complexity: $O(n^2)$
   step-by-step time complexity: $O(1)$ in one iteration
   step-aside time complexity: $O(n^2)$
   keep-in-step time complexity: $O(n^2)$
   elixir time complexity: $O(2^n)$ if without depth limitations
   leap-of-faith time complexity: $O(n^2)$

   Space complexity is problem-specific. But we need an n^2 matrix at least to store the board.

   To be improved: "Step-by-step" is the best we can do. "step-aside" can be accelerated in 2 ways:

   (a) Use more space to store the unsolvable override relations. Ignore those relations when solving override relations.

   (b) Instead of searching the whole board when "step-by-step" fails, update relations in "step-by-step". But it will increase the time complexity of "step-by-step" and may lead to the result that some blocks are updated too frequently.
   "keep-in-step" can be improved in the same way as we could use to improve "step-aside".
   "elixir" is actually problem-specific constraints because minesweeper is an NP-complete problem. But still, we can use some limitations to prevent it from taking too much time. (Details are in "\docs\Solution Algorithm Explaination.html".)
   "leap-of-faith" can also be preprocessed in "step-by-step", but it still leads to frequently updating.

7. 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.

Now we can keep a counter to count how many mines left. There are 3 ways to use this information:

(a) In "elixir", we can limit maximum mine numbers in the proofed range, which may decrease the total number of possible distributions of mines.

(b) In "leap-of-faith", we can estimate the probability of being mines of blocks isolated from the knowledge base. It is more possible to choose a less risky block to explore.

(c) At the endgame stage (less than 15 blocks inconclusive), we can use "elixir" to search the whole board with exact mine number. Some cases are solvable.

3 Bonus: Chains of Influence

1. 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?*

   Influence chain can be divided into 2 parts:

   (a) How can we use the information that a block should be a mine block?

   (b) How can we organize several hints to deduce new information?

   For the first part, if we find a block which should be a mine block, there are at least 3 things we could do:

   (i) Mark this block to flag, and increase the flag counter by 1.

   (ii) Decrease the left value of all neighbors of this block by 1, because now this block is conclusive.

   (iii) Decrease the warn value of all hint blocks' neighbors by 1, and update their neighbors' prob value.

   (iv) If a block's updated prob is 0 or 1, it is now conclusive. Therefore, add it to the waiting line to explore it or to mark it to flag in the future.
   We do not know whether the situation listed in (iv) is accessible, but we also include it in this report.

   For the second part, refer to "Solution Algorithm Explaination.html".

2. What influences or controls the length of the longest chain of influence when solving a certain board?

   For the first part, it mainly depends on the distribution of mines. Because all new information is deduced in step (iv). But if a mine block's neighbors are all mine blocks or inconclusive blocks, we cannot update any hint block's warn value. So if mines are too dense, influence chains are usually short.
   For the second part, it is more about how long would a chain be. The longer a chain is, the more time we need to combine those hints.

3. How does the length of the chain of influence influence the efficiency of your solver?

   For the first part, it will not influent the efficiency. Actually, the more we depend on it, the less we will depend on the second part. The longer the chain of influence is, the faster the agents can be.
   However, for the second part, as a chain gets longer and longer, it takes more and more time to combine hints, and it tends to be less and less probable to be solved. (Find more in "Solution Algorithm Explanation.html".

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

   Yes. For example, a 64×64 board with 10 mines. In this case, the chains of influence would be very long.
   A nearly-empty board (without mines) yields pretty long chains. Because "step-by-step" is totally based on the notion of influence chain and in this case, this board can be solved only based on "step-by-step".
   As mine density goes up, "step-by-step" tends to be stuck in dead ends. Therefore, the length of chains decreases when mine density increases. Note that if the total number of mines and board size increase at the same time (density staying unchanged), the length of the chain will increase, because there are more blocks that can be processed by "step-by-step".

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

   The given cell can travel across the whole board. Because a board nearly empty can be completely solved by "step-by-step".

6. 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?

   For the first part, we would like the length to be as long as it can be. In this scenario, we present the whole board as matrices and use "step-by-step" to spread any newly conclude blocks' information to their neighbors, and neighbors of neighbors.
   For the second part, we limit the maximum length.
   "Step-aside" is limited into 5*5 range. Many "cascaded" override relations are ignored because they are usually unsolvable.
   "Keep-in-step" is limited into 3*3 range. As 2 twin blocks get further, they will share fewer common blocks. In this case, a large number of unique neighbors usually lead to an unsolvable situation.
   When using "Elixir", the larger the range is, the more likely that it is solvable. But there is a downside. A large range will inevitably make it spend an unacceptable time.
   Therefore, we choose a range of 9*9 as the limitation for 2 reasons:

   (a) It should be larger than "step-aside" and "keep-in-step", otherwise nearly all solvable structures will have been solved by them.
   (b) With a limitation of 9*9, it usually takes about 1 second to solve each block, which is acceptable.

7. Is solving minesweeper hard?

   For most (about 85%) boards whose mine density is around 20%, it is hard to solve completely. When the mine density is above 20%, the solver becomes powerless. However, it is pretty easy to solve 95% of the board whose mine density is under 20% as long as we have a good start.

4 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.*

   When it is uncertain about the time to receive the information, we explore these blocks and get to know they are safe. In the following execution of our algorithm, we just ignore the blocks without feedback, because they do not provide any useful information.

For the experiment part, we set the probability of revealing a clue from 0.06 to 0.15, and repeatedly solve the maze using our algorithm. The plot is showed below.

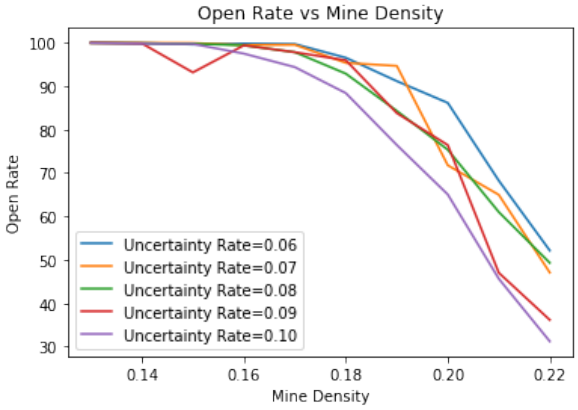

Figure 12: The influence of different uncertain rate on open rate under different mine density.

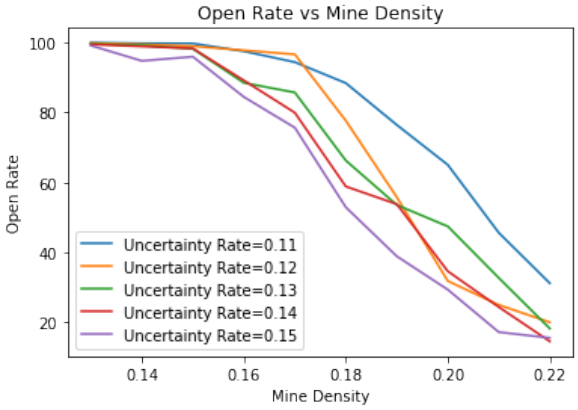

Figure 13: The influence of different uncertain rate on open rate under different mine density.

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.*

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.*

   For the second and third scenarios, we use conditional probability $P(hint|clue)$ to represent our deduction process. But we did not have enough time to do experiments on these two scenarios.