Group Member:
Xin Yang(xy213)
Zhuohang Li(zl299)

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

3. Questions and Writeup

1. **Representation:**
   We use a class Cell to represent cells in the board. It has a property CellType which is an enum class consists of different cell types including MINE, CLEAR, UNKNOWN, CLUE etc to represent the current state of the cell, and another property integer value to store the number of mines in the adjacent area. Board is represented by a 2-dimensional array of cells.

2. **Interference:**
   The status of the board will be updated after every query request using functions such as setCellType() and setVal(). We think our program is able to deduce everything it can from a given clue before continuing by utilizing the following functions:
   a. run(): we first traverse all cells to check if there is any cell that satisfies: 1) the number of revealed mines in its adjacent area is equal to its clue value, in which case all unknown cells in adjacent area should be safe 2) the clue value minus the number of revealed mines in its adjacent area is equal to the number of unknown cells in its adjacent area, in which case those unknown cells are all mines.
   b. assume(): If the function in step a didn't find any useful information, we then use assume() function to go through all possibilities to find potential mines and clear cells.
   c. randomGuess(): If we still can't find any useful information after step b, which means all information on board has been considered, we then have to take a random move.
3. **Decisions:**

This program has three strategies to take, first is to make direct operations according to the current knowledge base, this strategy take steps without any risk, e.g., surrounding mines are all marked and all unknown cells must be clear or clues, in this case, the program queries all surrounding unknown cells and this will always be safe. This strategy will be executed most frequently, and each unsafe strategy is followed by a direct loop until we have to take a risk.

The second strategy is to make assumptions based on the knowledge we have. In this case, the program will go through all clue cells and see if there is one that has enough clues to be inferred, the threshold can be set up by the user, here we set it to be 3, i.e., if there are less than 3 unknown neighbours, it can go into the assumption process . If so this program will add it to the assumable list and go through all possible permutations of neighbouring unknown cells to see if there is a meet. To determine if such a permutation exists, this strategy will go through all related clue cells and see if such an assumption won't break any rule. If there exist multiple assumptions, we only take the one that has the fewest assumed cells.

The last strategy is a random guess. This strategy will only be executed if the above ones can't move any further. E.g., at the very beginning of the game. After one random guess, the program will immediately return to the direct strategy to see if the random operation can lead to any new safe moves.

Risks are that there could be situations where multiple assumptions exist, and can be against each other. Taking any of these assumptions could be a risky decision but still have a relatively large possibility to move on. Also the random guess is always a dangerous option.

4. **Performance:**

   What I don't agree with is when the program can't make direct decisions, it will try an assumption, but the order of assumption is based on the traverse.

So when later visited cells have better assumptions than the first visited ones, the program will still do the more risky assumption.

Since I set a proper priority and made the program to run the direct strategies in a loop, and this loop will continue after an assumption is made. Our program will always do the safest way, rather than make more assumptions. A random guess will only be executed under extreme conditions.

5. **Performance:**
   For a advanced game, which has a 16x30 board, exclude the games that hit a mine at the first several queries, i.e., fail because of bad luck, we run our test based on 50 rounds random generated games. Our program has a nearly 36% success rate for 80 mines. For the same board with 70 mines, the success rate will increase to 60%. For a standard medium level game, which has a 16x16 board and 40 mines, the success rate is 48%.

   We set the program to print the game and at which step the game ends. 99% of the failure is caused by random guess, and over 70% of the valid games have almost found all mines. Since our assumption strategy is relatively conservative, almost all failures are caused by a random guess. And after looking at the game, honestly, even for a human, it is still hard to find a better solution since all available clues cannot reveal further.

6. **Efficiency:**
   Given a board with length M and width N, the time complexity of our program will be O(M*N) due to the fact that we need to go through all cells on board. The space complexity will also be O(M*N) to keep track of the status of every cell. We think those are problem specific constraints.

7. **Improvements:**
   The number of mines can help finish the game earlier if all mines have been found and if the number of all unknown cells equals to unveiled mines, then we can guarantee all left cells are mines. Also, we can calculate the

probability of a certain cell is a mine real-time, in this way we can compare with the winning probability of an assumption and decide which one is less risky.

4. Bonus: Chains of Influence

1. Our implementation returns all qualified queries in a list and queries the user one by one, in this approach, the list is right the influence chain of the cell that generates such a list.

2. The number of mines, the size of the board, and the distribution of the mines and clues decide the length of the chain. If all mines can be directly found based on the known knowledge without any assumption or guess, all the cells are inside the influence chain To make the chain longer, in an extreme case, if each cell can only be determined after the last one is known, the length would be the number of all cells. And if the available knowledge can't move forward, the chain is broken and a new guess or risky assumption will be the start of a new chain. Another factor that causes a short chain is that each level of such a chain can recursively unveil more cells, and thus only a very short chain can win the whole game, this usually happens when there are fewer mines and a larger board.

3. If the length is shorter and the number of cells in such a chain is greater, it's more efficient to solve the game in that it can always take a safe step and don't need to traverse the whole board again, the results of the last step can make more cells be discovered. Otherwise, the assumption strategy would consume a lot of time and space to try all possible permutations and even fall into a random guess step, which is likely to end the game, and all this would only generate a starting point of a new chain rather than make the previous chain longer.

4. For a fixed 16x30 board, if there are less than 10 mines, the length is short, and when the number of mines becomes larger, The length becomes longer

but the number of cells in each level becomes fewer, also there are more chains generated in one game. If we have over 90 mines, both the number in one level of a chain and the length of chain become shorter.

5. In a small board with sparse mines, the influence chain will go through the whole board and the game ends. On a larger board, e.g., the advanced board, the chain usually ends fast because the available knowledge base is useless. For a 16x30 board with 80 mines, 40-50 cells would be an ideal quantity for a single chain of influence, normally just 20-30 cells. But for a medium level board, the influence chain will contain most of the cells and only leave less than 10 unknown cells.

6. When we have multiple available steps to take, we can always take the one that might build a shorter chain of influence and have more children in its subchain.

7. Solving a minesweeper largely depends on the luck when the game is challenging enough. For a simple board with a few mines, the program can easily build a longer chain without being interrupted. But when the game is advanced, a dead end can always happen and we have to take a new guess. This is quite dangerous and luck based if the total number of mines are not given. Exclude the fact of luck, this game is quite easy, which only need to do basic math and naive assumptions.

5. Bonus: Dealing with Uncertainty

1. Our solver is capable of dealing with such uncertainty but it comes with the drawback that it may decrease the probability of solving a board. If the solver cannot get the information it needs, it may fell into a random guess situation earlier, which could bring an early end to the game. The modification could be if a query for a certain cell is not responded, adding that cell into a list and revisit that list of cells after finishing current queries.

2. In this case, the first strategy will no longer work. We have to run multiple assumptions to check the surrounding cells before finally decide whether a cell is a mine or not. Solver facing optimistic clues will be unaware of certain mines. Therefore, collecting more information from surrounding cell before making moves is crucial.

3. When making an assumption, assume that the number of cells is as shown by the clue. This would give us a board with more mines. Then finalize the board according to the total number of mines.

6. Work Distribution

Xin Yang: I implemented the data structure of the program, and implemented part of the direct strategy, the random guess and the assumption strategy.

Zhuohang Li: I implemented the advanced strategy, the naive version of the direct strategy and the random guess.