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

Changlin Jiang(cj360), Ruolin Qu(rq40), Shengdong Liu(sl1563)

Questions and Write-up

1. Representation

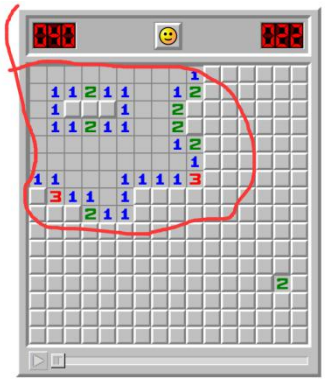
In this assignment, the board is set as a 2D array of Class Cell.
Cell contains the types (named CellType) of “Unknown”, “Clear”,” Mine”, “Number” ,”END”. Each types represent the state of each cell.

Cell also have the property (named val) stores the information of each cell hint (the number of mines near the cell).
Our mine board is generated randomly represented as a 2D int type array, the number in where represents the clue provide by this element.

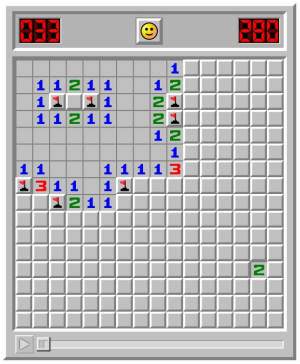
2. Inference

Let’s think about how to play MineSweeper as human:
Simple algorithm(for convenience, we called it **Search**)

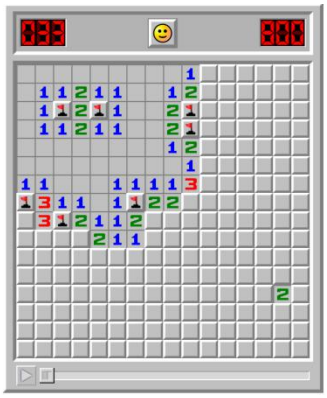
a. Pick a random cell, reveal it. If it is a number, reveal another one until a bigger space is revealed. Just like the following board showing until the cells in the red ring is revealed. If we step on a mine at this time, restart.



b. Then we mark the “obvious” mines in the board which is inferred by the information of cell.



c. Now we marked some mines, which implies some “safety” cells we can explore now, then we explore them. The last explored cells will infer more “obvious” mines and with an iteration of step b. and c. If no “obvious” mines are inferred. We pick another random cell and do b. and c. until end or success.



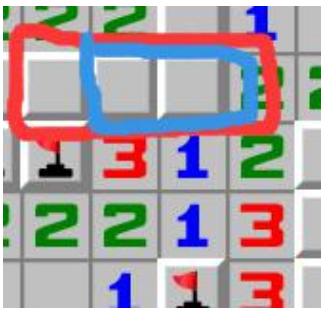
Our simple algorithm program works in the same way like below.

a. Random reveal a cell at the very beginning. (which is implement in RandomGuess())
b. If it is “number”, random reveal again until get a “Clear” cell.
c. put its neighbors to query and then reveal the neighbors, if some neighbors are “clear”, do c. on these neighbors until we got a “clear” zone surrounded by numbers.
d. Mark the “obvious” mines where “the number of neighbors = the clue of mines”
e. Back to (a.) random pick again.

These steps are the same as it remained at Section 2 Program Specification.

Advanced algorithm (for convenience, we called it **Infer**):

Before we consider using random guess as a “helpless” start, Let’s think about the following situation:



Some “advanced” game players might notice that “the number “3” and the mine in the left” imply this knowledge “there are 2 mines in the red cycle” and “the number “1” in the right of number “3”” imply the following knowledge “there are only 1 mine in the blue cycle”. With these 2 clues we can conclude the left-up cell next to number “3” is a mine.

To do the inference above , we set a **knowledgebase** include **clue** represented like [zone, number of mines], zone is a array of [i ,j] which represented the position of cells.

while our simple program run to an end, our inference algorithm runs before randomly picking up the next cell.

Pseudocode of infer algorithm

When two numbers have common area, infer new information by this algorithm:

```
Define:
   n1 = number of unknown mines adjacent to first node
   n2 = number of unknown mines adjacent to second node
   c = {nodes in the common area of n1 and n2}
   u1 = {nodes that only belongs to n1}
   u2 = {nodes that only belongs to n2}
   knowledge = [n = number of mines in this area,
              area = {nodes in this area}]

Algorithm:
   main(){
      newKnowledge(n1)
      newKnowledge(n2)
      update()
   }

   newKnowledge(n):{
      knowledge = [n,{nodes adjacent to n}]
      knowledgeBase.append(knowledge)
   }

   update(){
     for knowledge1 in knowledgeBase:{
         for knowledge2 in knowledgeBase:{
            if( intersection of area in two knowledge is not empty){
               u1 = {nodes only belong to knowledge1}
               u2 = {nodes only belong to knowledge2}
               c = {common nodes}
               min_c = max(max(n1-length(u1),0),max(n2-length(u2),0))
            #  min c represents the least mines can be placed in common nodes
               max_c = min(min(n1,length(c)),min(n2,length(c)))
               if(min_c=length(c)):
                  mark all nodes in C as "Mine"
               else if (max_c=0):
                  mark all nodes in C as "Safe"
               if(n1-max_c>=length(u1)):
                  mark all nodes in u1 as "Mine"
               else if(n1-min_c=0):
                  mark all nodes in u1 as "Safe"
               if(n2-max_c>=length(u2)):
                  mark all nodes in u2 as "Mine"
               else if(n2-min_c=0):
                  mark all nodes in u2 as "Safe"
               }
            }
         }
```

}

As what we provides above, we are sure that our advanced program deduce everything it can from all given clues.

3. Decisions

At the very beginning, the program will pick random cell to reveal. Here it will take the risks of $\frac{total\ mine}{board\ size}$.

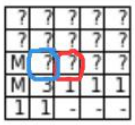
After random search at beginning, the program will calculate every unknown cells' possibility (p). and pick the next step on the less possibility of stepping mines. to get more details, take a look at **Improvements** section.

Every exploration in **Search** and **Infer** are absolutely safe.

Our program will only fail at guessing progress and we also try to calculate the guessing possibility to optimize the results.

4. Performance

1. Surprised result



```
Inferred Position: [[2, 3], [2, 3], [2, 2], [2, 2]]
True
```

As a human player, the inference in blue cycles are enough for breaking the dead end. However, our **Infer** program will find all "safe zones" before marking mines and keeping **Search** after **Infer.**

2.

5. Efficiency

Our program has to traverse every cell on the board. In this case, the time complexity of this program will be O(L*W) where L represents the length and W represents the width of the board. And the space we take are also O(L*W) to save the status of cell. The knowledge base will only be created after **Search** and the space cost is less than O(L*W) which could be ignored.

6. Improvements

For this improvement approach, we use the function getMinesTotal() return the total num ber of mines, and countMines() return the number of mines we have been found so far. These information can make a big improvement in random guess selection. For those ce lls on the board have not been discovered with no CIUE cells around it, we calculate the possibility of these cells as (board.getMinesTotal()-board.countMines())/board.countUnknowns(). For those cells that have CLUE cell aroun

d, the possibility of them to be the mine should be 1/p(1)*1/p(2)*1/(p3)...1/p(n). Number n is sequence of the clue cells, this result is not exact accurate because the we use this a pproximate way to calculate, however we argue that's close enough for our mine sweep er to make the right decision.

7. Bonus

1.Our solver is capable of sweeping the mine with such uncertainty without revea ling the "clue"cell. If the solver cannot get the clue information, it is same as the a gent waste one chance obtaining more information, which cause the early end to the game. The modification can be if cell is uncover with no clue, add the cell into the query again after the current one.