Intro to Artificial Intelligence 16:198:520

Minesweeper: Inference-Informed Action

Baber Khalid, Muhammad Aunns Shabbir, Jimmy Jorge, Kyungsuk Lee
https://github.com/baber-sos/Minesweeper

Questions & Writeup

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

Visually, a cell in the minesweeper board will contain a single value from -2 to 8.

- 0 represents a clear open cell with all neighboring cells safe
- 1-8 represent the clear with neighboring cells that contain mines given number of mines
- -1 represents a confirmed mine in the specific cell
- -2 on the board (shown in the figures below) represents an unexplored cell. -2 in the status list represents the specific cell and its neighbors have been fully explored.

We store the play of the game in a hash table, where the key represents the coordinate of the cell itself, and the value stores a tuple. The tuple contains the status of the cell (i.e. -2 to 8), and a list of tuples. Each tuple in this list contains the number of cells of that specific status and the set of coordinates containing those mines.

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?

For each iteration of our algorithm, there is new information that is constantly being processed and updated. The algorithm keeps track using a boolean variable if it receives new information in a specific iteration. If the information is concrete, it works on that by opening all the clear spaces or marking the mines. Otherwise, it tries to partition the existing neighbor sets into multiple disjoint sets to try and narrow down the position of the mines by calculating intersections with all the cells that have some common neighbors and using possible mine counts to calculate the number of satisfiability requirements. Since each cell is checked in each iteration, the algorithm deduces everything from a given cell before it tries to make a guess. For a better understanding, consider the following two cells with their respective neighbors:
(2, 4) : (2, [(0, {(1,3)}), (2, {(2, 3), (1, 5), (2, 5)})])
(1, 6): (1, [(1, {(1,5), (2, 5), (0, 6)})])

The above data means (2, 4) is surrounded by two mines in the possible cells (2, 3), (1, 5), (2, 5). (1, 6) is surrounded by one mine in the possible cells (1, 5), (2, 5), (0, 6). The algorithm will first open cell (1, 3) since it should have no mines according the algorithm, but when it tries to deduce more information, it will compare both (2, 4) and (1, 6) because they have common neighbors. After calculating the intersections and differences, we can divide the two sets in three parts:
{(0,6)}, {(1, 5), (2, 5)}, {(2, 3)}
Then, the algorithm will try to calculate the number of satisfying assignments by starting with a zero in the left most segment and going up to the min(len(set), no_of_mines). Since the left most cell is the unique neighbor to (1, 6), it can have a maximum of one mine, meaning two possible assignments of 0 and 1.
In the case of 0, it will realize the following satisfying assignment:
0 mine, (1 mine), 1 mine
In the case of 1, the first assignment encounters the following contradiction:
1 mine, 0 mine, 2 mines
Since there is only mine in the rightmost set it will know it cannot have more than one mine so there is only one satisfying assignment of mine count. Using this information, the resulting cell information will look like this:
(2, 4): (2, [(1, {(2, 3)}), (1, {(1, 5), (2, 5)})])
(1, 6): (1, [(0, {(0, 6)}), (1, {(1, 5), (2, 5)})])

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?

In each iteration, our algorithm goes through the list of opened cells sequentially and opens the cells which are confirmed clear or marks the mines. If neither is possible, then it calculates the intersections with the neighboring cells to see if it can add any new information by partitioning the existing sets into disjoint members. Since each time, we are looking at all the opened cells, our approach does not involve any risk until there comes a point when our algorithm is forced to make a guess because it cannot add any new information to the already available knowledge base. Then, our algorithm tries to make the best educated guess by finding a mine set with the least probability of having a mine. Probabilities represent the chance of neighboring cells having a mine when considered independently. This might be improved using conditional probabilities.

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?

On a 16x16 board with 40 mines, we ran the algorithm to solve the game beginning with the first cell at (0,0).

```
(0, 0) (0, [(0, {(0, 1), (1, 0), (1, 1)})])
[0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!
```

The first cell shows all 3 neighbors are open cells, denoted with '0'. The rest of the board is unexplored, denoted with '-2'. We keep track of cells in the tuple shown above the game with the coordinate of the current cell, followed by a list of the coordinates of neighbor cells and their status. In this case, the current cell is (0, 0), and the list of neighbor cells are (0, 1), (1, 0), and (1, 1) with a status of open, shown with a '0' at the beginning of the list.

```
(0, 0) (-2, [])
(0, 1) (0, [(0, {(1, 2), (0, 2)})])
(1, 0) (0, [(0, {(2, 0), (2, 1)})])
(1, 1) (0, [(0, {(2, 2)})])
[0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!(0, 0) (-2, [])
```

The next play, we look at the neighbor cells of the neighbor cells from the previous iteration. In this case, we look at the neighbors of (0, 1), (1, 0), and (1, 1). Every iteration, the list of each current cell's neighbors will constantly keep track of the status of all surrounding cells as the board is explored.

```
(0, 1) (-2, [])
(1, 0) (-2, [])
(1, 1) (-2, [])
(1, 2) (0, [(0, {(0, 3), (1, 3), (2, 3)})])
(0, 2) (0, [(0, set())])
(2, 0) (0, [(0, {(3, 0), (3, 1)})])
(2, 1) (0, [(0, {(3, 2)})])
(2, 2) (0, [(0, {(3, 3)})])
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, 1, 1, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!
```

```
(0, 0) (-2, [])
(0, 1) (-2, [])
(1, 0) (-2, [])
(1, 1) (-2, [])
(1, 2) (-2, [])
(0, 2) (-2, [])
(2, 0) (-2, [])
(2, 1) (-2, [])
(2, 2) (-2, [])
(0, 3) (1, [(1, {(1, 4), (0, 4)})])
(1, 3) (1, [(0, {(2, 4)}), (1, {(1, 4), (0, 4)})])
(2, 3) (1, [(1, {(3, 4), (1, 4)})])
(3, 0) (1, [(1, {(4, 1), (4, 0)})])
(3, 1) (1, [(0, {(4, 2)}), (1, {(4, 1), (4, 0)})])
(3, 2) (1, [(1, {(4, 1), (4, 3)})])
(3, 3) (0, [(0, {(4, 4), (3, 4), (4, 3)})])
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, 1, 1, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, 1, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!
```

```
(0, 0) (-2, [])
(0, 1) (-2, [])
(1, 0) (-2, [])
(1, 1) (-2, [])
(1, 2) (-2, [])
(0, 2) (-2, [])
(2, 0) (-2, [])
(2, 1) (-2, [])
(2, 2) (-2, [])
(0, 3) (1, [(1, {(1, 4), (0, 4)}), (0, set())])
(1, 3) (1, [(0, set()), (1, {(1, 4), (0, 4)})])
(2, 3) (1, [(1, {(1, 4)}), (0, set())])
(3, 0) (1, [(1, {(4, 1), (4, 0)}), (0, set())])
(3, 1) (1, [(0, set()), (1, {(4, 1), (4, 0)})])
(3, 2) (1, [(1, {(4, 1)}), (0, set())])
(3, 3) (-2, [])
(2, 4) (0, [(0, {(1, 5), (2, 5), (3, 5)}), (0, set())])
(4, 2) (0, [(0, {(5, 1), (5, 2), (5, 3)}), (0, set())])
(4, 4) (0, [(0, {(4, 5), (5, 4), (5, 5)})])
(3, 4) (0, [(0, set())])
(4, 3) (0, [(0, set())])
[0, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -1, 3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, 1, 3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, 1, 1, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -1, 1, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, 1, 1, 0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!
```

```
(0, 0) (-2, [])
(0, 1) (-2, [])
(1, 0) (-2, [])
(1, 1) (-2, [])
(1, 2) (-2, [])
(0, 2) (-2, [])
(2, 0) (-2, [])
(2, 1) (-2, [])
(2, 2) (-2, [])
(0, 3) (0, [(0, {(0, 4)})])
(1, 3) (0, [(0, set())])
(2, 3) (-2, [])
(3, 0) (0, [(0, {(4, 0)})])
(3, 1) (0, [(0, set())])
(3, 2) (-2, [])
(3, 3) (-2, [])
(2, 4) (-2, [])
(4, 2) (-2, [])
(4, 4) (-2, [])
(3, 4) (-2, [])
(4, 3) (-2, [])
(1, 4) (-1, [])
(4, 1) (-1, [])
(1, 5) (2, [(2, {(0, 6), (2, 6), (0, 5), (1, 6)})])
(2, 5) (2, [(2, {(2, 6), (1, 6), (3, 6)})])
(3, 5) (1, [(0, {(4, 6)}), (1, {(2, 6), (3, 6)})])
(5, 1) (0, [(0, {(6, 1), (6, 2), (6, 0), (5, 0)})])
(5, 2) (0, [(0, {(6, 3)})])
(5, 3) (0, [(0, {(6, 4)})])
(4, 5) (1, [(1, {(5, 6), (3, 6)})])
(5, 4) (0, [(0, {(6, 5)})])
(5, 5) (0, [(0, {(5, 6), (6, 6)})])
[0, 0, 0, 1, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -1, 3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, 1, 3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, 1, 1, 0, 0, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, -1, 1, 0, 0, 1, 1, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[1, 1, 1, 0, 0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[2, 2, 1, 0, 0, 0, 0, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]
press enter to continue playing!
```

As shown, the algorithm simply explores the board in order of the open cells found, and the number of lists of cells with status and coordinates grows to the limit of the board size. When there is a set() shown for a coordinate, it denotes a null set which is removed in the next iteration.

```
[0, 0, 0, 1, 1, 3, -1, 2, 1, -1, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -1, 3, -1, 2, 1, 3, -1, -2, -2, -2, -2, -2]
[0, 0, 0, 1, 1, 3, 2, 2, 0, 3, -1, -2, -2, -2, -2, -2]
[1, 1, 1, 0, 0, 1, -1, 1, 0, 2, -1, -1, 3, 3, -1, -1]
[1, -1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 2, 2, -1, 3, 2]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[-1, -1, 2, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]
[3, -1, 2, 1, -1, 1, 0, 0, 1, -1, 2, 2, 2, 2, 1, 0]
[2, 3, 3, 2, 1, 1, 0, 1, 3, 4, -1, 2, -1, -1, 1, 0]
[1, -1, -1, 3, 1, 1, 2, 4, -1, -1, 2, 2, 2, 2, 1, 0]
[1, 3, -1, -1, 1, 1, -1, -1, -1, 4, 2, 1, 1, 1, 0, 0]
[0, 2, 3, 3, 1, 1, 2, 3, 3, -1, 2, 2, -1, 1, 0, 0]
[0, 1, -1, 2, 1, 0, 0, 0, 1, 2, -1, 2, 1, 1, 0, 0]
[1, 2, 3, -1, 1, 0, 0, 0, 0, 1, 1, 2, 2, 2, 1, 0]
[1, -1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 1, -1, -1, 1, 0]
press enter to continue playing!
```

```
Going to make a guess from the following:  (0.3333333333333333, {(2, 13), (2, 11), (2, 12)})
Oops we lost
```

After a number of plays, we get to a point on the board where the algorithm has to make a guess between different cells in order to continue. In this case, the algorithm had to guess between cells (2, 13), (2, 11), and (2, 12) with a probability of 0.33 of a cell being a mine. For this play, the algorithm guess incorrectly and we lost.

```
Going to make a guess from the following:  (0.3333333333333333, {(2, 13), (2, 11), (2, 12)})
[0, 0, 0, 1, 1, 3, -1, 2, 1, -1, -2, -2, -2, -2, -2, -2]
[0, 0, 0, 1, -1, 3, -1, 2, 1, 3, -1, -2, -2, -2, -2, -2]
[0, 0, 0, 1, 1, 3, 2, 2, 0, 3, -1, -2, 3, -2, -2, -2]
[1, 1, 1, 0, 0, 1, -1, 1, 0, 2, -1, -1, 3, 3, -1, -1]
[1, -1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 2, 2, -1, 3, 2]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[-1, -1, 2, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]
[3, -1, 2, 1, -1, 1, 0, 0, 1, -1, 2, 2, 2, 2, 1, 0]
[2, 3, 3, 2, 1, 1, 0, 1, 3, 4, -1, 2, -1, -1, 1, 0]
[1, -1, -1, 3, 1, 1, 2, 4, -1, -1, 2, 2, 2, 2, 1, 0]
[1, 3, -1, -1, 1, 1, -1, -1, -1, 4, 2, 1, 1, 1, 0, 0]
[0, 2, 3, 3, 1, 1, 2, 3, 3, -1, 2, 2, -1, 1, 0, 0]
[0, 1, -1, 2, 1, 0, 0, 0, 1, 2, -1, 2, 1, 1, 0, 0]
[1, 2, 3, -1, 1, 0, 0, 0, 0, 1, 1, 2, 2, 2, 1, 0]
[1, -1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 1, -1, -1, 1, 0]
press enter to continue playing!
```

In this instance of rerunning the game, the algorithm guessed correctly and we are able to continue exploring the board.

```
[0, 0, 0, 1, 1, 3, -1, 2, 1, -1, 3, 2, 1, 0, 0, 0]
[0, 0, 0, 1, -1, 3, -1, 2, 1, 3, -1, -1, 2, 1, 1, 0]
[0, 0, 0, 1, 1, 3, 2, 2, 0, 3, -1, 5, 3, -1, 3, 2]
[1, 1, 1, 0, 0, 1, -1, 1, 0, 2, -1, -1, 3, 3, -1, -1]
[1, -1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 2, 2, -1, 3, 2]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[-1, -1, 2, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]
[3, -1, 2, 1, -1, 1, 0, 0, 1, -1, 2, 2, 2, 2, 1, 0]
[2, 3, 3, 2, 1, 1, 0, 1, 3, 4, -1, 2, -1, -1, 1, 0]
[1, -1, -1, 3, 1, 1, 2, 4, -1, -1, 2, 2, 2, 2, 1, 0]
[1, 3, -1, -1, 1, 1, -1, -1, -1, 4, 2, 1, 1, 1, 0, 0]
[0, 2, 3, 3, 1, 1, 2, 3, 3, -1, 2, 2, -1, 1, 0, 0]
[0, 1, -1, 2, 1, 0, 0, 0, 1, 2, -1, 2, 1, 1, 0, 0]
[1, 2, 3, -1, 1, 0, 0, 0, 0, 1, 1, 2, 2, 2, 1, 0]
[1, -1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 1, -1, -1, 1, 0]
```

Finally, we reach the end state of the game where the board is fully expanded and the algorithm has won the game.

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?

Utilizing the minesweeper online game given to us, we tested our program on the “Beginner” (9x9 with 10 mines), “Intermediate” (16x16 with 40 mines), and “Expert” (16x30 with 99 mines) modes with success. For some cases in the Beginner and Intermediate boards, nearly no guessing was utilized as all of the information was gathered periodically from neighboring cells. For Expert, our program highly depended on which cell it opened when it had to make a guess; although we were successful for each type of board, we decided to do the rest of our testing on an Expert sized board with a variable number of mines.

For 16x30 **randomly generated cell boards** with X number of mines. 1000 trials were run for each board with a specific number of mines.

Success Rate of Solving Minesweeper board

	Number of Mines									
Size	**50**	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**95**	**100**
16x30	81.3%	72.3%	63.5%	54.4%	46.5%	32.3%	22.2%	14.6%	8.0%	3.9%

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?

Implementing this program, a space constraint we encountered was due to the fact that we store the board for each iteration of the algorithm. This specific constraint would be an implementation specific constraint because as a future improvement, we could simply not store each board for every pass. The time constraints include inferring whether or not the board can safely open a cell to expand its knowledge base or if it has to calculate the probability of which cell would be safer to open in a set of multiple cells. This would all depend on the board size and the number of mines included within the board, thus this time constraint would be a problem specific constraint. With less mines to worry about in a given space, the time complexity would be a lot less as the algorithm would not have to constantly be checking for neighboring cells and calculating the probability of which cell is statistically safer to open at any given time.

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

This information can be modeled and included in our program via the use of a global tracking variable. During the running of the algorithm, the number of mines in the board (inputted by the user) would be the value of a set variable. This variable, for each mine found in the board, would be decremented until it reaches 0. Once at 0, our algorithm would understand that every single mine in the board has been found, thus our algorithm would not have to over-analyze the rest of the cells in the board and skip all of the condition checking. On top of this, knowing the total number of mines in the board would assist in more accurate guesses of specific subsets, when need be.

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?

Each time a cell tries to calculate the satisfying assignments after calculating intersections with other cells having some common neighbors, it may partition its mine into multiple disjoint sets. This information is updated at each iteration, and its effect spreads to other cells as it gets more specific. This effect of getting updated iteratively can tell which cells are affecting other cells at each iteration and can help us build a chain of influence due to the fact that a certain cell reaches a specific state starting from some other state.

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

Each iteration of the algorithm, we constantly intersect sets of possible mine cells in order to extract data for the next iteration. The more specific the data becomes, the more it helps cells with common neighbors. Therefore, the highest granularity we can achieve for a given mine set while trying to narrow down the location of mines, controls the length of the longest chain of influence.

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

The length of the longest chain of influence influences the efficiency of our solver directly. The longer the chain of influence, the greater number of cells opened with full certainty that they do not contain mines. Generally speaking, if a very long chain of influence is found, little to no guessing would have to be done, thus increasing the solve rate of the minesweeper board.

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

0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1
1	1	1	1	-1	1	1	-1	1
2	-1	2	1	1	1	1	1	1
2	-1	3	2	2	2	1	0	0
1	3	-1	4	-1	-1	2	0	0
0	2	-1	-1	4	-1	2	0	0
0	1	2	2	2	1	1	0	0
0	0	0	0	0	0	0	0	0

Starting at (0, 0), the beginning of the chain of influence was found. From there, we were able to expand our list of safe cells to further explore. The chain of influence continually grew until every mine was found without any guesses. This varies with the total number of mines proportionally. As the number of mines increases, the smaller the chain of influence becomes. In this particular case with 10 mines, the chain of influence was able to grow until every open cell was found without any guesses made. The length of chain of influence will vary inversely with an increment in number of mines. The reason for that, with the increasing number of mines board may be divided into different subsections (position of mines is also important) not having an effect on each other. Due to which, we will have to make more guesses because it will not be possible to obtain any useful information at one point.

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

Each cell will effect all the cells which have some common neighbors with it. Depending on the granularity of information available for a single cell, that information will be carried forward to the cells having some common neighbors with those cells. So assuming that whole board is connected these chains of influence can spread to the whole minesweeper board and help us solve the board without even a single guess. The board given in question 4, shows such an example.

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?

This is what our solving algorithm is already doing. For each cell, it finds the effect of that cell on only the cells which have a common neighbor with it. Those effects will start to spread as more and more iterations pass. This approach is more efficient as it helps the solver to only worry about the local effects. The effects turn global as more and more iterations pass and more information is obtained.

7. Is solving minesweeper hard?

This correlates directly to the question of the length of chain of influence. If a given board is such that it has particularly long chains of influences, then it can be solved with the least amount of guess like in question 4. However, if the board is such that the length of chains of influences are particularly short, then it forces the player to guess at certain points. Each time one has to guess, the uncertainty of being wrong increases, which makes the solution hard to reach. This directly relates to the length of chains of influence, position, and number of mines in the board.

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.***

- 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.***

- 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.***

1. How could you adapt your solver to each of these three cases? Try to implement at least one, and experiment with what kind of performance hit results, in terms of ability to clear the board.

FIRST CASE: When our solver queries the user board for the value at a certain index we can put a probabilistic factor using pseudo random number due to which that value may not be revealed. Then, our solver will have to figure out the value of these cells by either using information obtained from other cells or will have to guess about in the worst case.

SECOND CASE: In the second case, whenever we discover a 0 in a cell, we can't act on it. Instead, the only concrete information we can have is the location of mines (when the number of unknown neighbors is same as length of mine set). So we have to gain more information as we discover more mines and improve our guessing using that.

THIRD CASE: This case is similar to the previous in that we lost a piece of concrete information which is the confirmed number of mine locations. In this case, we are sure about the locations of clear cells (cells surrounded by a 0) but the mine locations are not accurate. So we have to improve our understanding of the board using the information about clear cells and then make a guess for clear cells using the rest of the information available.