Steinar Jennings

Portfolio Project – Minesweeper Game

EC: Adjustable Input Size +1

<u>Brief Description of Game:</u>

Minesweeper is a game where there is traditionally, a 16x16 grid with 40 bombs placed on it. The user seeks to "sweep" the minefield and identify the locations of all the mines, revealing all the safe spaces at the same time. The user will select an initial tile at random to flip and investigate. There are three possible outcomes from flipping a tile.

The first outcome: It was a bomb! This triggers a game over message.

The second outcome: It was a tile near a bomb or bombs.

> You will be able to infer how many of a tiles (up to 8) immediate neighbors are a bomb by the display. If a tile is surrounded by 3 bombs for example, the tile will display the number 3. Since you clicked on a tile that was near a bomb, no algorithms are implemented to reveal additional tiles, since we cannot investigate all the neighbors without investigating a bomb.

The third outcome: The tile did not have any neighboring bombs.

> In this instance we know that all the neighboring (up to 8) cells are bomb free, so we can go check them out. This is where the implementation of my algorithm, which was a BFS, starts. See the algorithm section below for details on the algorithm and how it works.

You can also choose to flag a tile that is unrevealed if you are suspicious that it may have a bomb under it. This is done so the user can remember where they should be cautious of clicking.

The game continues until the player triggers a bomb or clears the board. The goal is to avoid clicking near tiles that have many bombs as neighbors. The more tiles revealed, the easier it is to see where the bombs truly are.

Please refer to my README.md file for more details on the exact implementation of my version of minesweeper, which follows in general all the rules described above.

In my version of the game, I created a game class, that created a grid with cells objects, containing information about them, such as whether it is a bomb, the number of neighbors, etc. This was done to help support the searching algorithms, so I could store more information in each cell.

<u>Algorithm:</u>

The algorithm that performs the solving of the board is my "revealBFS" algorithm. <u>This algorithm runs in O(r*c) time</u>, with "r" representing the number of rows on the gameboard and "c" representing the number of columns.

This algorithm will evaluate the users input and inspect the first, the input cell. If the cell object is not a mine, we will then perform our BFS on each of the neighboring cells. As I mentioned before, in the game of minesweeper, we are "safe" to visit any neighbor, so long as the cells we are visiting has no immediate neighboring bombs. Because of this, we perform our BFS on each neighboring cell and repeat the process from the neighboring cells, if said cell has no immediate bombs. The algorithm will run until there are no remaining, neighboring cells with 0 adjacent bombs.

This algorithm also tracks the cells that we have already visited to save us time from performing extra searches.

If the user has triggered a tile that reveals all non-bomb tiles, a check will be performed to evaluate to see if the number of hidden cells is equal to the number of bombs.

<u>NP Completeness:</u>

The verification of whether a partially revealed minesweeper board can be solved is in NP-Complete. This can be verified by looking at a partially completed minesweeper board. This does not hold true for all instances of board state. That is why this program falls in the realm of NP problems. There is not a surefire way to solve a game. If we can algorithmically deduce where the bombs are given neighboring cell information, then we are able perform a verification, but this cannot be done at all stages in play. It works best we all remaining cells are surrounded by edges or numbered cells. This problem is known as the consistency problem.

To prove minesweeper is in NP Complete we must show that there exists a decision problem P, such that P is in NP. Additionally, we must prove that minesweeper is reducible to a known NP-Hard problem.

In the case of minesweeper, we know that given a certain, game board that is finished, we can determine if it is a solvable game by evaluating all the provided board information is accurate, wherein all the number of adjected bombs and other data representation is correct. Essentially, we can in poly time evaluate whether a gameboard HAS a solution, we are not actually solving it. This can obviously be done in poly time by visiting each cell and inspecting its neighbors to verify that we have the correct information. If we do not have correct information, for example, if a cell says 6, but really there is only 1 bomb by it, we could not verify that the game could be solved. This would take O(cells*8) although not every cell has 8 neighbors. Since we have confirmed that we can verify a boards playability in simple poly time, we need to prove that the problem is reducible.

The obvious way to test this is to see if we can solve a specific game layout using a known NP-Complete algorithm. We know that each cell in minesweeper represents one of two things, it is either a bomb, or it is not. Knowing this we can make decisions and pass-through scenarios along the bordering cells (where we have adjacent bombs) based on an initial guess, we can see how each of the other pieces in the “circuit” would interact, given that it should have enough information to deduce where a bomb is given the location of a “known” one. The solution would pass possible combinations of bomb locations through the circuit until we reach a solution that satisfies all the adjacency requirements of the known cells.

As we look at the unknown cells bordering our known cells, we have established that they individually either are, or are not a bomb cell, when we expand this problem, we are looking at a circuit of decisions. That circuit for example can look at two unknown cells and tell us, this cell x or this cell y are the bomb, but not both, we also can see instances where a decision is arrived deducing this cell x AND this cell y is the bomb and so on. This chain(“circuit”) of decisions will ultimately lead to a solution that satisfies every decision gate. Symbolically satisfying the decision gates means that we have identified the proper locations of the bombs to decide. As described above, the problem can be reduced to what is known as Circuit-Sat, which is known to be NP-Complete through this class (which implies that it is also in NP-Hard) We are making decisions based on decisions gates, and at each cell there is a gate(s).

Because we know that we can verify the legitimacy of a game board in Poly time, we also know that the subproblem can be reduced to a known NP-Complete problem, we can conclude that the decision problem in minesweeper (clearing the map, methodically) is in NP-Complete.