Minesweeper Write-up

Anirban Chakraborty

Representation:

The minesweeper game board is made up of 2d array of values on whether it has been clicked, contains a mine, or a flag. Our agent would interact with the board with a random choice on the first move. Based on that decision and if the choice was not a mine, the agent would take in the number of mines in the picked position as a constraint. The information that our agent would have is a list of solved squares so that it doesn't pick it again, a list of mine squares that would be flagged, a temporary and updating list of surrounding squares around the picked square, and a list of constraints that stores the clues (number of mines indicated by the picked squares).

Inference:

When a new clue is gained by the agent, the agent uses this constraint to pick the next square. The agent only consults the most recent constraint to reduce the complexity of the program. For example, the agent picks a square (0th row,0th col) and it gives a clue of 1. It takes note of the surrounding squares and takes them in (e.g 0,1 | 1,1 |1,0). An arbitrary threshold binds the agent to only pick the surrounding squares if the threshold is greater than the clue value / number of squares. (To maintain a consistent probability to the user's preference). So if the threshold was .5, ⅓ is less than the threshold, and it would pick, at random, of the surrounding squares. If it is another clue value, the process repeats, with a new set of surrounding squares, the previous picked square into solved square list, and a new clue. This approach would theoretically reduce complexity and time needed to solve the board, depending on how greedy the threshold is.

Decisions:

The agent picks the next square based on the information of the most recent clue stored. It will decide to flag and only flag all surrounding squares at once if the clue value equals the number of surrounding squares. It is to be noted that surrounding squares do not contain solved squares, so the list of surrounding squares only takes in the squares which haven't been picked yet or not flagged. If the threshold is less than the clue value / number of surrounding squares, then the agent picks another square at random, repeating the process. Over the number of turns, the agent will have reached its goal state if all the squares picked have either become solved or flagged, and if the number of mines is less than the number of flagged mines, it will pick the flagged mine square list at random until a mine is found or the flagged mines = actual mines. The biggest risk involved is that the agent sacrifices greater information of the board in order to achieve a faster time of s solved board, since it only bases its decision on the most recent clue value of the picked square. Another risk is that if the threshold is too greedy (too low), then failure increases, or too safe(too high), then the average number of turns becomes longer and there are more chances of failure with more chances to fail. We face this risk by having a threshold less than or equal to .5 for a faster and greedy approach regardless, since the alternative is worse.

Performance/Efficiency/Improvements:

Since I could not effectively code the agent, I played Minesweeper by picking random squares to my own intuition, and played a simultaneous game where I only picked squares that my agent would pick to its own constraints. Unfortunately, my agent did not perform as well as expected, as the number of turns would decrease on my end as I had information of all the clues listed on the game, unlike my agent who would only have information of one clue at a time and a static threshold to decide on whether to pick the surrounding squares or not. The obvious improvements to the agent would be to model its decisions based on a larger information set of clue values, namely the entire board, rather than one clue at a time. Perhaps an additional data set of a respective probability matrix instead of a binary threshold would achieve more success than the rudimentary agent. While I wanted to avoid complexity to decrease the average amount of turns taken, sacrificing information for the agent to access is not a viable choice for decreasing time. I also question the efficiency of the agent, as I think the data structure of a constraint should hold more information than the clue value. Perhaps a list of clue values of the entire game board + a probability matrix with the same dimensions of the game board, and each square of the p. Matrix holds a constant arbitrary probability that is derived from the respective clue value (e.g .5 for 4/8 squares or .33 for ⅓ squares . The agent that compares varying probabilities of the matrix and picks the one that is the highest. Thus, the decision is binded by varying probabilities instead of a binary choice of one clue value and redundant random choices.