Minesweeper Final AI Report

Team name Group20
Member #1 (name/id) Brian Yu / 15388961
Member #2 (name/id) Tzu Hsuan Huang / 67913387

I. Minimal AI

I.A. Briefly describe your Minimal AI algorithm. What did you do that was fun, clever, or creative?

Our Minimal AI first creates an empty board which it updates over time. Then it uncovers all the safe tiles that are guaranteed to not have a mine. If the covered tiles surround 0 then they get added to a queue containing all safe tiles. If the uncovered tile does not have a number 0 then it gets added to a queue containing nonzero tiles. The algorithm continues to go through the safe queue until all safe tiles are exhausted. Then it looks at the nonzero queue for any common patterns by using the rules of thumb. In the case of Minimal AI, it looks for the pattern where there is only one covered unmarked tile bordering an uncovered 1 tile. In other words, the effective label of that uncovered 1 tile is equal to the number of unmarked neighbors. It flags it as a mine, pops this uncovered 1 tile from the nonzero queue, and continues checking other uncovered tiles in the nonzero queue until the board is finished. Usually, it only needs to check the nonzero queue at most twice since the order of the uncovered tiles can affect when we find the mine. We think the most creative parts were the use of a queue and the fact that we created a class to store tile information for easy storage in the queue. The fun part was planning the AI and figuring out some clever ways to solve the problems we had. For example, since the board contains one mine, we only need to use rules of thumb to find it. Effective label gave us clear information about where the mine is located.

I.B Describe your Minimal AI algorithm's performance:

Board Size	Sample Size	Score	Worlds Complete
5x5	1000		1000
8x8	1000	0	0
16x16	1000	0	0
16x30	1000	0	0
Total Summary	4000	0	0

II. Final AI

II.A. Briefly describe your Final AI algorithm, focusing mainly on the changes since Minimal AI:

The biggest change to our Minimal AI was the addition of backtracking. As the number of mines increases, it is possible that we have to check the nonzero queue more than twice. Therefore, when the size of the nonzero queue does not change after checking pattern, we create two sets of frontiers using two maps. One map contains the uncovered tiles from the nonzero queue, with each tile having the value of its covered neighbors. The other map consists of these covered neighbors, where each covered tile has the value of its uncovered neighbors. Therefore, we can use these two maps for backtracking by initially assuming the covered tile is a mine. We then check the effective label and the number of adjacent covered/unmarked tiles for each uncovered tile neighbor of that covered tile. If the information does not follow the logic, it cannot be a mine, so we revert back and label it as safe, continuing to assume the next covered tile as a mine. If it does follow the logic, we continue to assume the next covered tile as a mine. We repeat this process until all covered tiles in the map have a vaild state, mine or safe. We store the information and then we return to the previous step to check for other possible solutions. Once we have found all the solutions, we can calculate the probability of each covered tile. Then we used mergesort to sort the tiles in order of probability and chose to uncover the tile with the 0% probability of being a mine and label the tile with 100% probability as a mine. In addition, when backtracking and pattern checking was unable to find a tile that can be uncovered or label as a mine, we created a function that essentially guessed to uncover a tile that has the lowest probability.

Some other changes were mainly bug fixes. For example when we ran our AI on expert worlds we realized that it segfaults. Through the use of gdb we discovered that the issue lied in the fact that we confused the association between the x,y coordinates and row,col. As a result, there were a lot of instances in our code where x was compared to row and y was compared to column. As a result sometimes boundary checking would not work properly, and the algorithm would try to access memory out of bounds of the board array.

II.B Describe your Final AI algorithm's performance:

Board Size	Sample Size	Score	Worlds Complete
5x5	1000	1000	1000
8x8	1000	784	784
16x16	1000	1582	791
16x30	1000	978	326
Total Summary	5000	3344	2901

III. In about 1/4 page of text or less, provide suggestions for improving this project (*this section does NOT count as past of your two-page total limit.*)

Some ways we can improve this project is to increase the accuracy of the algorithm. For example, we could add a constraint satisfaction-based approach. This way if pattern checking does not work or backtracking does not work, we could use constraint satisfaction so that we can resort less to guessing. In addition, now looking back at the algorithm we should have added a time out clock so that we can keep track of time and allow the algorithm to randomly guess as a last resort. Not only could this help during debugging by allowing us to determine which maps were solved correctly within the time limit but it could improve our chances. Although at the time we figured that if the algorithm couldn't finish in time, it was already pretty much a lost cause. In addition, there were times when the program would take a very long time to do an expert map. This is probably due to how much backtracking it had to do and given more time optimizing the backtracking would be a good way to improve the algorithm.