Minesweeper Final AI Report

Team number_____61____
Member #1 (name/UCI netid)_____Thar/thzaw__________________________________
Member #2 (name/UCI netid)_____Husam/solachuh____________________________

I. Minimal AI

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

Our minimal AI algorithm covered the simple rules of thumb, without any sophisticated or guessing logic. Our algorithm could solve about 30% of beginner worlds and 100% of super easy worlds. We had functions to find the number of covered and unmarked neighbors and the covered and marked neighbors around a certain tile. We used these functions along with the perceptive number of a tile to find if an uncoverable neighbor exists. Our algorithm was unique in that we kept track of “parent” and “child” tiles, e.g., if a tile on the board had a label of 0, we made that tile the parent and kept choosing the neighbors around it as children to keep uncovering, before moving on to other tiles. Since we did not cover ambiguous cases, the algorithm returns LEAVE immediately.

II. Final AI

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

Since the minimal AI, we’ve implemented sophisticated logic upon coming across ambiguous cases. Specifically, we’ve added backtracking search and model checking to efficiently find all possible worlds that satisfy world constraints. We initialize 2 frontier lists (C and V), one list being the uncovered tiles and the other being the covered tiles that both surround the perimeter of all minesweeper islands. We then use the frontierV list and treat it as a stack, and we also make a popped stack (a reciprocal of frontierV). We use these stacks in a loop; if we consider a tile from frontierV’s to be a mine, and which satisfies our constraint function, we can pop that tile from the frontierV stack and append it to the popped stack. If it doesn’t satisfy the constraint function, we then consider the tile to be an uncoverable tile and do the same thing. If it still doesn’t satisfy, we backtrack by popping a tile from the popped stack and appending it back to the frontierV, and loop back again. If all of the tiles from frontierV have been popped, all tiles have been assigned a value that satisfies constraints, and the values are appended to a list of possible solutions, and we continue backtracking. We stop the loop when we’re backtracking, but there are no tiles to pop from the popped stack. We uncover the tile that has the least amount of mines from the list of solutions.

III. In about 1/4 page of text or less, provide suggestions for improving the performance of your system.

We have a try except clause for a specific portion of our code, as our algorithm doesn’t work with 100% of worlds (runtime error that comes up rarely for specific worlds). Debugging this problem could help find the root of some larger problem which could help speed up the algorithm. We can also add more attributes in our Tile class to save the number of covered and uncovered neighbors as well as its effective label in each tile, instead of repeatedly running our functions when we need access to them. We don’t check if our AI runs for more than 5 minutes on a world, but implementing a timer and a check at the beginning of our algorithm would be essential. The popped stack in our implementation might also be somewhat unnecessary, as we can still complete our algorithm with just one stack, so it could be a waste of memory and operations. Additionally, we should limit the number of tiles we can have in our frontierV list, so that it doesn’t take so much time to run our algorithm. We would only be trading a bit of accuracy for greater reduction of time.

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

Board Size	Sample Size	Score	Worlds Complete
5x5	100	0	100
8x8	100	34	34
16x16	100	44	22
16x30	100	0	0
Total Summary	400	78	156

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

Board Size	Sample Size	Score	Worlds Complete
5x5	100	0	100
8x8	100	81	81
16x16	100	158	79
16x30	100	99	33
Total Summary	400	338	293