Minesweeper Final AI Report

Team name: AlphaSweeper

Member #1: Litian Liang / 58905797

- **Minimal AI**

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

I wasn't really an interesting algorithm at that time (a lot more interesting after draftAI). I implemented rule of thumb and it works for 100% of the world.

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

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

- **Final AI**

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

My final AI Agent consists of 5 intelligence layers.
Here is how I will refer to my 5 intelligence layers in the explanation:

1. **Basic Rule of Thumb Logic (Basic)**
2. **Equal Subset Inference (Esub)**
3. **Unequal Subset Inference (UEsub)**
4. **Recursive Backtracking Search with heuristics (CSP)**
5. **History Inspection Guided Randomness (HIG)**

Also some specific things that I will refer to:

indecator number: the number Agent see after uncovering a tile
frontier: opened tiles on the board with >0 tiles around it.
frontier tiles: the tiles that are directly relevent to frontier

Overview:
Minesweeper is proven to be an NP-Complete problem. Without time constraint, deciding deterministic situations only requires CSP. However, to finish one world under 5 min, we need to speed up the process of finding a deterministic result. Therefore, Basic, Esub, UEsub are helpers to preprocess the board. When the situation becomes more complex, they will return failure and leave the current state to CSP. Here is a pseudocode for the procedure that I have described above.

getAction (state) {

```
basicResult = Basic(state)
if basicResult is not failure:
    return basicResult
esubResult = Esub(state)
if esubResult is not failure:
    return esubResult
...
```

}

Note: HIG will not return failure (always give an action, may be incorrect) according to my design.

Explanation of intelligence layers:

1. Basic

There are 2 rules:

Rule 1: Check every frontier cell [c] and get the number [k]. Then count the number of tiles [t] around it. If k = t, agent will flag all the tiles around c.

Rule2: Check every frontier cell [c] and get the number [k]. Then count the number of flags [f] around it. If f = k, agent will uncover all the tiles that are not flagged.

2. Esub

Let's consider the situation below (a, b, c) are uncovered tiles, (f) are flags:

F	F
a	3
b	2
c	1

In this case, Basic will return failure. Esub solves the problem by:

1. **Construct a set of variables (domain = {0, 1}) for every frontier cell with known sum k.**
    a. **In the above case: ({a, b}, 1), ({a, b, c}, 2), ({b, c}, 1)**
    b. **This means an equation system of: a + b = 1, a + b + c = 2, b + c = 1.**
2. **Check every 2 set, see if one is the subset of another one.**
    a. **In the above case: {a, b} is a subset of {a, b, c}, and same for {b, c} and {a, b, c}**
3. **If there is one, check to see if the difference of these 2 number is 1: 0 or 2: equal to its cardinality.**
    a. **In the above case: 2 – 1 = 1 and {a, b, c} – {a, b} = {c}**
4. **If the difference in number = |set difference|, then all variable in set difference are mine. If the difference in number = 0, then all variable in set difference are 0.**
    a. **In the above case: |{c}| = 1, which indicates that c is mine.**

After that, Basic will handle all the rest.

3. UEsub

Let's consider the situation below (a, b, c, x, y) are uncovered tiles.

x	a	b	c	y
?	1	2	1	?

In this case, Esub will fail. UEsub solves this problem by:

1. **Construct a Esub set of variables. Like the example above.**
    a. **({a, b. x}, 1) …**
2. **Given that every variable can only be 1 or 0, we can infer a set of unequalities.**

    a. **a + b <= 1 (using 1 on the left), a + b >= 1 (using 2 in the middle)**
3. **we can deduct that a + b = 1**
4. **we can, by looking at the sets that contains this set, and using the set difference rule to find out mine / safe.**
    a. **a + b + c = 2 -> c = 1 therefore, c is a mine.**

After this, Basic can handle the rest of the cells.

4. **CSP**
    1. **Varset = {a, b, c} (uncovered tiles), domain = {0, 1}, constraints constructed from the numbers on the board. Like a + b + c = 2, and a + b + x = 1, etc**
    2. **A recursive backtracking search that verifies the validity of partial assignments.**
    3. **Selects the next variable to explore using MRV + degree heuristics (I implemented using Euclidian distance and surrounding tile count to measure these two respectively).**
    4. **Finds all valid combinations, save them.**
    5. **If any block can be mine or flagged (all valid combination has 0 or 1 assigned to some variable), open or flag it.**

5. **History Inspection Guided Randomness**
    1. **The last try to open a mine with the minimum probability of mine in situations that are impossible to be deterministically solved. I assume that in 1000 matches, a perfect random set of board are generated.**
    2. **When solving board, it maintains two 2d array, HISTORY and UNKNOWN that has cell dimension corresponding to each level. When entering a new game I increment every cell in UNKNOWN by 1. If in a game, I discover some cell is not mine, UNKNOWN[x,y] -= 1 and HISTORY[x,y] += 0. If mine, HISTORY += 1.**
    3. **This way I will be able to estimate the probability by looking at how many times a cell has been mine. In my implementation, I will select the cell that was the most mines in previous games to open, since the probability of it is still mine this time will be the lowest.**

II.B. Describe your Final AI algorithm, focusing mainly on the changes since Minimal AI:

Board Size	Sample Size	Win Rate	Worlds Complete
5x5	1000	1.00	1000
8x8	1000	0.89	891
16x16	1000	0.87	872
16x30	3000	0.40	1255
Total Summary			

Further improvements:

I am thinking of using Monte Carlo simulation in non-deterministic situations to implement "forward-look-N" which is looking with UCB and try some possible combinations and simulate the possible state configuations to see which one has the best probability of winning the whole board. It will be interesting as I have never seen any paper use MCTS on minesweeper. I think will be a worthy try.