CS5011 A3 Report
Logic

1. Parts Implemented

Part 1: Random guess and Single Point Strategy
Part 2: Easy Equation Strategy

2. How to run

Run like this, where <map number> is the map that you wish to use. Numbers 1-5 are the easy maps, 6 - 10 the medium and 11 - 15 the hard ones.

java -jar Logic1.jar <map number 1 - 15>
java -jar Logic2.jar <map number 1 - 15>

Alternatively, to run all of the maps you can use:

java -jar Logic1RunAll.jar
java -jar Logic2RunAll.jar

3. Literature Review

Minesweeper & NP-Completeness: what is P vs NP and why do we care?

P stands for polynomial time and refers to a class of problems in which the time needed to solve them does not increase exponentially compared to the inputs, like multiplication and sorting. The asymptotic notation for these kinds of problems looks like this: O(n) or perhaps O(n^k) or O(kn) etc. where n is some measure of the number of inputs to the problem (e.g. the number of numbers to sum) and k is a constant. So these problems are not necessarily all easy to solve at the moment, (e.g. O(n^200) is still in P), but their difficulty does not exponentially increase relative to their complexity, and so to solve them all we need to do is wait for computers to get more powerful. [14]

NP stands for nondeterministic polynomial time, and includes problems like sudoku and minesweeper (and many real-world problems too, like routing and scheduling and other

important things), which become exponentially harder to solve as their complexity increases. The asymptotic notation for such problems looks like like this: O(k^n). [10] This is somewhat intuitive - a 10x10 minesweeper board is far more than twice as difficult as a 5x5 one. With NP problems, if given a solution you can *check* it in a reasonable amount of time - if given a completed minesweeper board or sudoku puzzle, it's easy to tell if it has been correctly solved. This is distinct from say, chess, in which if the problem is: 'What is the best move to make at board state X?' it is very difficult to judge whether a given solution is correct. [14]

So P and NP are useful ways of talking about the complexity of a given problem. A problem is in P if it can be solved in polynomial time, and NP if it can be verified in nondeterministic polynomial time. Thus, we can see that all P problem are in NP. But are all NP problems in P?

Russell and Norvig state that "One of the biggest open questions in computer science is whether the class NP is equivalent to the class P when one does not have the luxury of an infinite number of processors or omniscient guessing. Most computer scientists are convinced...that NP problems are inherently hard and have no polynomial-time algorithms. But this has never been proven." [10]

What is NP-Completeness?

NP-Complete problems are a subclass of NP, and consist of the hardest NP problems that we know of. It has been proven that if we can solve any NP-complete problem efficiently, we can solve all of them, and also that either no NP problems are in P or all of them are. Thus:

"An efficient solution to any **NP**-complete problem would imply **P** = **NP** and an efficient solution to every **NP**-complete problem..." [4]

Why do we care?

Because lots of important problems are in NP and NP-Complete, and if we could solve them efficiently the world we be amazingly different - complex optimisation problems would become trivial and we would see huge leaps forward in machine learning! "What we would gain from P = NP will make the whole Internet look like a footnote in history." [4]

Sadly, most people believe that P ≠ NP [3], although no-one has proven it yet.

Minesweeper is NP-Complete!

In 2000 Richard Kaye proved that Minesweeper is NP-Complete [5]. This is really exciting because it means that "an expert Minesweeper player could spot some pattern in the game that would eventually lead to an polynomial-time algorithm for solving it—and hence polynomial-time algorithms for all the NP-complete problems. Such an outcome would result in methods to crack all of the codes used in the internet and computers across the world." [8]

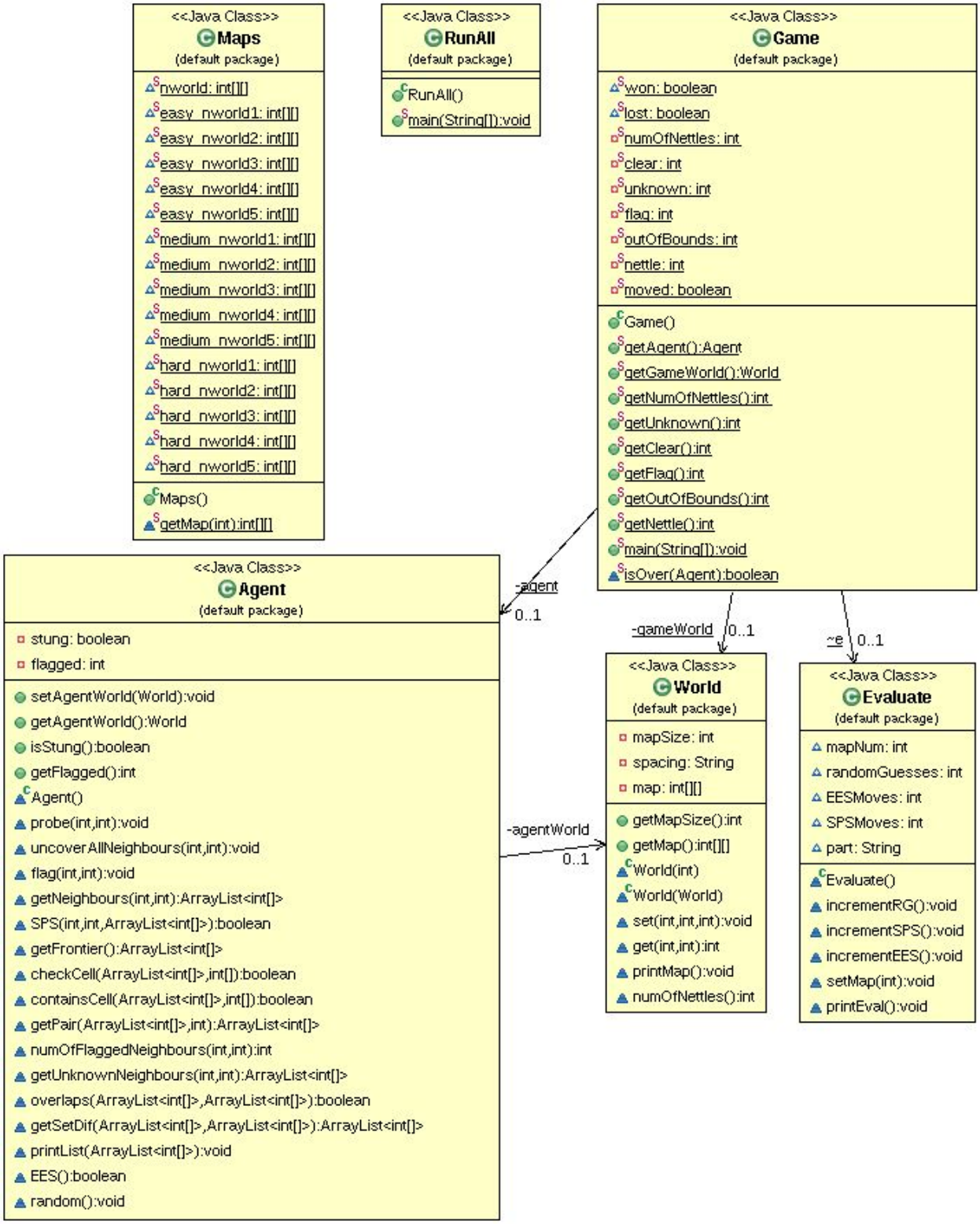
Unfortunately I don't think I've managed to solve P=NP in this assignment.

Playing Minesweeper

Minesweeper can be approached as a game of probability [17] in which the player hopes to assign either 0% or 100% probability to a certain tile containing a mine by reasoning logically based on the information available about the board. However, the player may come across situations in which it is impossible to reason with certainty about the contents of a hidden cell, and no further options remain but to take a guess at the most probable scenario [2]. Fortunately all the boards used in this assignment are solvable logically so our agent won't get blown up. (Or rather stung, since we're implementing Nettlesweeper.)

We can also approach Minesweeper as a constraint satisfaction problem, in which every tile is a variable with two possible values ('contains a mine' or 'does not contain a mine'), and every tile that does not contain a mine creates a sum constraint over its neighbours - for example, a tile containing the number 3 requires that 3 of its neighbours contain mines in order for the problem to be satisfiable [1].

4. Design & Implementation



Game Implementation

Since the agent has access to different information than the game does, I separated the two into distinct classes. The Agent class contains all the methods necessary to implement the three strategies, and the strategy implementations themselves. Both the Game class and the Agent class have an instance of the World class, which provides methods to get information about their specific representation of the world. This consists of a Map from the Map class (which simply contains 2d int arrays of all the maps and delivers whichever is requested), and methods for accessing the contents of that map at given coordinates, finding how many nettles exist in the world, printing the map to the console, updating the map for the agent when new information is uncovered, etc. There is also an Evaluate class, which just provides an easy way of collecting information about each game and then printing it to the console, and a RunAll class, which enables all maps to be run one after the other.

The game logic runs like this: while the game is not won or lost, loop through each of the cells in the agent's representation of the world (implemented as the the Agent's World's Map). For each cell, if its contents are more than 0, see if the agent can make a logical move to uncover or flag any of its covered neighbours (if it has any covered neighbours). If it can do this, go back to the start of the loop and resume checking each cell to see if a move can be made. If we reach the end of the map and no move can be made using SPS, either make a random guess (in Part1) or use EES (in Part2). Check if the game is over - if the agent has uncovered a nettle and thereby failed, or if all the nettles have been flagged. After each move, increment the Evaluator class to keep track of what strategy was used.

Agent & Strategy Implementation

The PEAS agent model

Performance measure:

- If stung (if a nettle is probed), the agent has failed.
- If all nettles are marked, the agent has succeeded.
- The fewer random guesses made, the more efficient the agent's strategy.

Environment:

- Probed cells show how many of their neighbours contain nettles.
- The neighbours of cells containing the number 0 are always safe.
- Flagged cells contain a nettle. (If the agent has flagged them correctly, which it should do if performing as expected.)

- The map contains a fixed number of nettles. (But the agent does not know how many.)

Actuators:

- The agent may probe one or more cells.
- The agent may flag one or more cells.

Sensors:
The agent can see the location of all probed, flagged and unknown cells.

The Agent does not have access to the same information as the rest of the game - rather, it has its own representation of the world implemented as described above. It has a number of methods that allow it to reason and implement strategies based on the contents of this world. Technical details of the implementation of these methods can be found in the code & comments.

Random Guess

- Pick some coordinates at random until an unknown cell is found.
- Probe that cell.

Single Point Strategy

- Select a cell that has a value greater than 0 and is not flagged.
- Sum the number of its flagged neighbours.
- Sum the number of its unknown neighbours.
- If the value of that cell is equal to the number of its flagged neighbours, we know that all the unknown neighbours are clear. Probe all unknown neighbours.
- If the value of the cell, minus the number of its flagged neighbours, is equal to the number of its unknown neighbours, then we know that those unknown neighbours must contain nettles. Flag all unknown neighbours.

Easy Equation Strategy

Find the frontiers (frontiers consist of any unmarked cells with an unknown neighbour), then find two adjacent cells within the frontier. Try each combination of adjacent cells (called c1 and c2) with the following strategy:

- Get the value in c1 and the value in c2 (called c1 and c2)
- Get the number of flagged cells in the the neighbours of cell1 (called f1) and in cell2 (called f2)

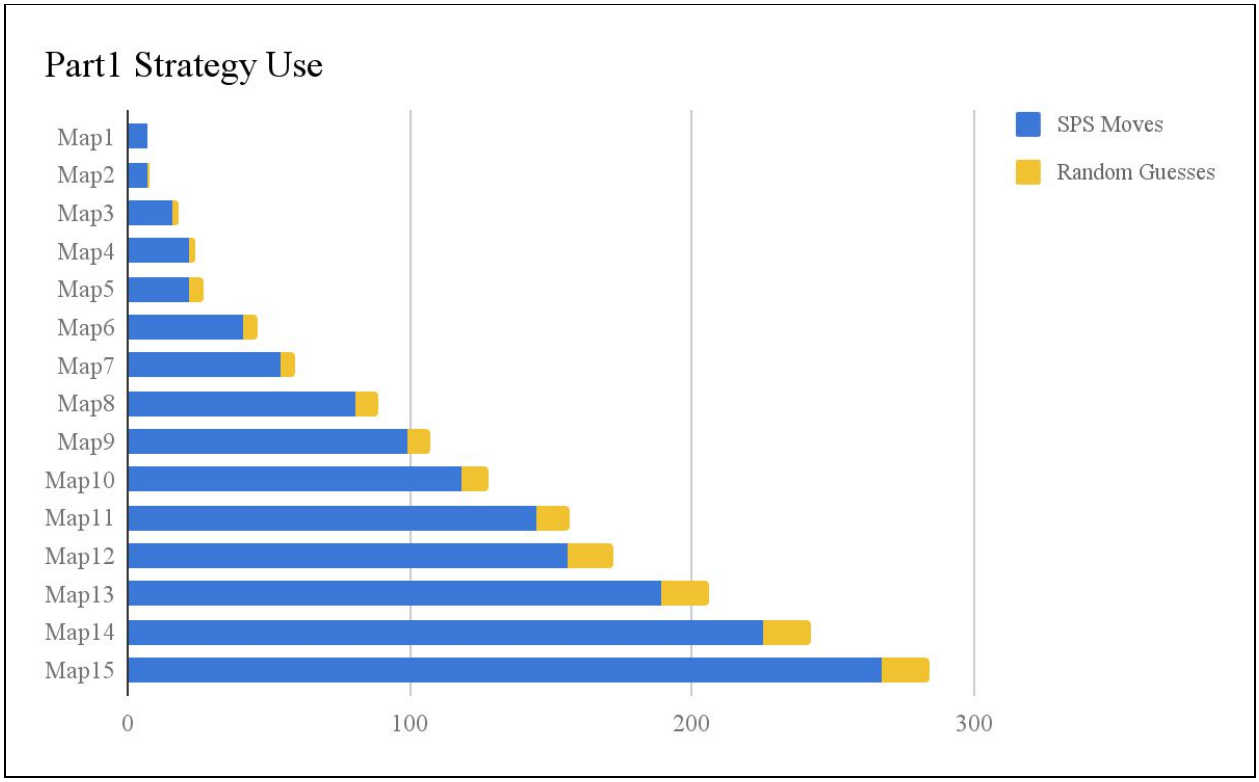
- Calculate the difference in how many nettles remain unflagged, in the neighbours of each cell: nDif = |(v1-m1) - (v2 - m2)|
- Then, find all unknown neighbours of c1 (called set 1), and of c2 (called set2)
- If one set fully overlaps the other, continue - else return.
- Get the difference between the two sets, eg: set1{a,b,c} - set2{a,b} = setDif{c}
- If the number of cells in the difference between the sets is equal to the difference in the number of nettles yet to be marked, we know that those cells must contain nettles! So mark all cells in the difference set.
- Or, if the difference in how many nettles remain unflagged in the neighbours of each cell (nDif calculated above) is 0, we know that the cells in the difference set must be clear, and so can probe them.

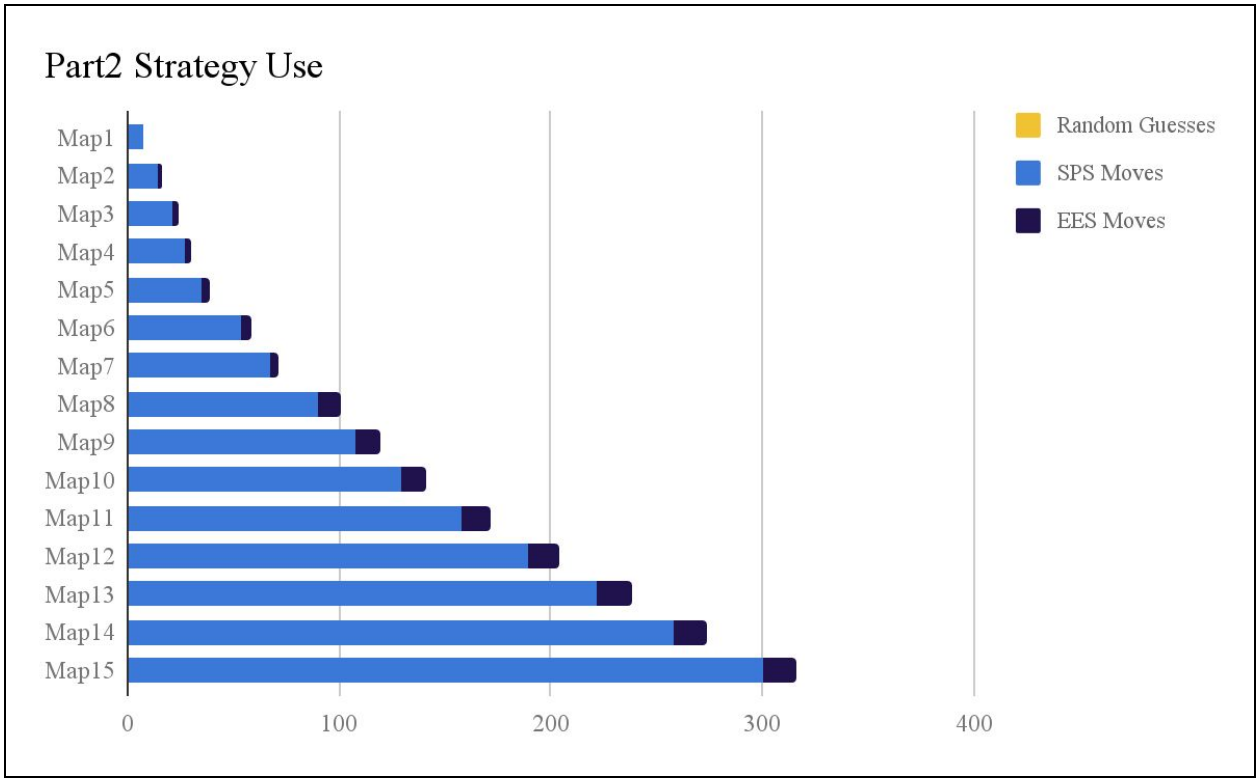
5. Performance

I created a simple evaluator class to keep track of the number of moves made by each strategy and print them out after each game to allow for easy evaluation.

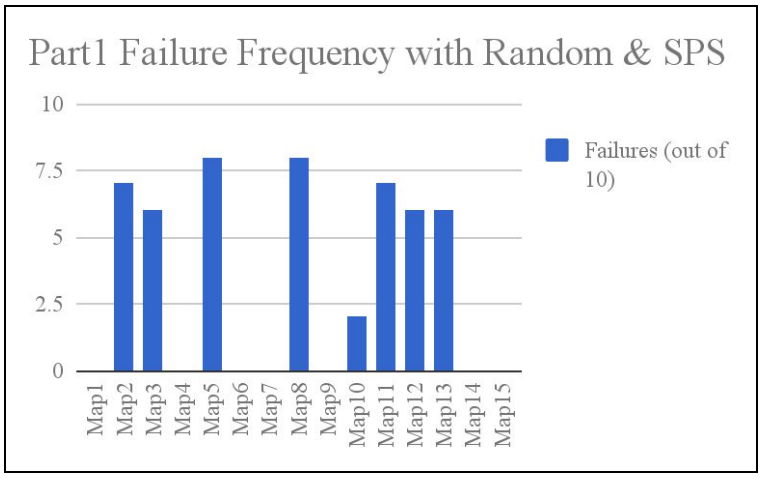
```
Flagged all the nettles!
End game.
Evaluation
Map: 15 Strategy: SPS, Random
Number of random guesses: 16
Number of SPS moves: 240
```

```
Flagged all the nettles!
End game.
Evaluation
Map: 15 Strategy: EES, SPS, Random
Number of random guesses: 0
Number of SPS moves: 300
Number EES moves: 16
```


Part1 Strategy Use
Map1
Map2
Map3
Map4
Map5
Map6
Map7
Map8
Map9
Map10
Map11
Map12
Map13
Map14
Map15
0
100
200
300
SPS Moves
Random Guesses


Part2 Strategy Use
Map1
Map2
Map3
Map4
Map5
Map6
Map7
Map8
Map9
Map10
Map11
Map12
Map13
Map14
Map15
0
100
200
300
400
Random Guesses
SPS Moves
EES Moves

Observations



Random guess alone failed most of the time, but occasionally succeeded through chance. It is not a good strategy for avoiding nettles.

SPS plus random guess is more successful. The success of this strategy did not seem affected very much by the size of the map, but rather the complexity of the reasoning required - I ran it ten times (not nearly enough to be conclusive but it can give us an intuition) and found that the average failure rate was around 50%. Clearly here some maps are easier than others - I don't believe it ever failed on map1.

Because the maps we were given are guaranteed to have a logical solution, EES succeeds on every map, and never has to resort to random guesses. This would not be the case in minesweeper, where it is possible to have tiles that are impossible to reason about with certainty.

6. Examples & Testing

```
Cell value is: 2
Flagged neighbours: 1
Unknown neighbours: 1
Flagging cell 3,2

    0002F
    0002F
    12121
    ??F20
    ???20
```

```
Flagging cell 9,9

    01111FF210
    01F2223F10
    024F201110
    01FF311211
    014F32F3F1
    002F33F311
    01333F2100
    12FF321000
    F223F32111
    11012FF11F

Flagged all the nettles!
```

```
Probing cell 7,1

    000000000
    111000000
    1F1001110
    111012F10
    00001F331
    122112???
    1FF211???
    23????????
    ?????????

Can't infer next move.

Chose cell at random 7,4
Probing cell 7,4
```

I have tested my strategies extensively through the use of print statements (which I have left in the code for posterity, but commented some of them out so the console isn't flooded) to make sure that everything was working as expected at each stage. This allowed me to very quickly

check if something odd was happening - e.g. if the agent was resorting to random guesses when it should not have been, implying that something was wrong with the logic implementation.

I also implemented a visual representation of the agent's map which is printed to the console at each step to allow for manual checking of the agent's reasoning. I walked through both SPS and EES step by step manually with a random selection of maps to ensure that at each stage the agent logic was working as intended.

```
Starting agent. Must flag 5 nettles.
Probing cell 0,0

    001??
    001??
    121??
    ?????
    ?????

Using EES.
Adding [0, 2] to frontier.
Adding [1, 2] to frontier.
Adding [2, 2] to frontier.
Adding [2, 0] to frontier.
Adding [2, 1] to frontier.
Frontier is: [0, 2][1, 2][2, 2][2, 0][2, 1]
Frontier cell is:
[0, 2]
Possible pairs are:
[1, 2]
Considering pair:
[0,2][1,2]
v1: 1
v2: 1
f1: 0
f2: 0
nDif: 0
set1:
[0, 3][1, 3]
set2:
[0, 3][1, 3][2, 3]
setDif:
[2, 3]
sDif: 1
nDif: 0
Probing cell 2,3

    001??
    001??
    1211?
    ?????
    ?????
```

7. Evaluation

All elements of my code work correctly, and I believe that the structure of my program and class and method decomposition are successful and logical. I did consider creating a separate cell class to represent individual cells which might have looked a bit neater, but in the end it seemed unnecessary as it would only have had a couple of methods, so I just used little arrays which was quick and easy. to I made an effort to factor out a lot of separate methods for each strategy and found that this made debugging and reusing code a lot easier. My implementation of both SPS and EES work as expected, and EES successfully solves every map without resorting to random guesses. If I had more time I would have solved P=NP too, but unfortunately that'll have to wait!

Wordcount: 2279

8. Bibliography

[1] Ken Bayer, Josh Snyder, Berthe Y Choueiry, and Ken ; Bayer. A Constraint-Based Approach to Solving Minesweeper. Retrieved November 12, 2017 from http://digitalcommons.unl.edu/cseconfwork

[2] Chris Studholme. Minesweeper as a Constraint Satisfaction Problem. Retrieved November 12, 2017 from https://www.mendeley.com/viewer/?fileId=1042e98c-3f57-53ed-0d79-468dbd6ad4b1&documentId=2eec9e23-ecc3-3519-bc19-b9bd3fd5ec1d

[3] Stephen Cook. THE P VERSUS NP PROBLEM. Retrieved November 12, 2017 from http://www.claymath.org/sites/default/files/pvsnp.pdf

[4] Lance Fortnow. 2009. The status of the P versus NP problem. *Commun. ACM* 52, 9 (September 2009), 78. DOI:https://doi.org/10.1145/1562164.1562186

[5] Richard Kaye. 2000. Minesweeper is NP-complete. *Math. Intell.* 22, 2 (2000), 9–15. Retrieved November 12, 2017 from https://link.springer.com/content/pdf/10.1007/BF03025367.pdf

[6] Richard Kaye. 2007. Some Minesweeper Configurations. (2007). Retrieved November 12, 2017 from http://web.mat.bham.ac.uk/R.W.Kaye/

[7] Richard Kaye. How Complicated is Minesweeper? Retrieved November 12, 2017 from http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ASE2003.pdf

[8] Richard Kaye. Minesweeper and NP-completeness. Retrieved November 12, 2017 from http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm

[9] Kevin Leyton-Brown, Holger H. Hoos, Frank Hutter, and Lin Xu. 2014. Understanding the empirical hardness of *NP* -complete problems. *Commun. ACM* 57, 5 (May 2014), 98–107. DOI:https://doi.org/10.1145/2594413.2594424

[10] Stuart Russell and Peter Norvig. 2010. Artificial intelligence : a modern approach. (2010), 1054.

[11] Ian Stewart. Ian Stewart on Minesweeper. Retrieved November 12, 2017 from http://www.claymath.org/sites/default/files/minesweeper.pdf

[12] NP-Completeness. Retrieved November 12, 2017 from https://www.ics.uci.edu/~eppstein/161/960312.html

[13] Complexity Theory - What is the definition of P, NP, NP-complete and NP-hard? - Computer Science Stack Exchange. Retrieved November 12, 2017 from https://cs.stackexchange.com/questions/9556/what-is-the-definition-of-p-np-np-complete-and-np-hard

[14] P vs. NP and the Computational Complexity Zoo - YouTube. Retrieved November 12, 2017 from https://www.youtube.com/watch?v=YX40hbAHx3s&t=47s

[15] Minesweeper and NP-completeness. Retrieved November 12, 2017 from http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm

[16] Programmer's Minesweeper. Retrieved November 9, 2017 from http://www.ccs.neu.edu/home/ramsdell/pgms/

[17] Minesweeper: Advanced Tactics. Retrieved November 9, 2017 from https://nothings.org/games/minesweeper/