CS5011 A3 Report

170008773

20th November 2017

1 Parts completed

- We successfully implemented all the requirements for part 1
- We again, sucessfully implemented all the requirements for part 2.

2 Parts not completed

- At time of writing we did not attempt to implement the SAT-solver strategy.

3 Literature review

The rules Minesweeper consists of a rectangular board of cells. At the start of the game, all the cells are covered, and some cells will contain mines. The Player/agent can perform two actions in this game: Flagging or uncovering a cell. If a cell containing a mine is uncovered the agent has lost the game. If a cell that does not contain a mine is uncovered it will reveal a number. This number is equal to the number of cells that are adjacent to the uncovered cell and contain a mine. If a cell is uncovered that is not adjacent to any nettles, all of its neighbours will be uncovered. The agent has won when all of the cells that do not contain a mine are uncovered.

P vs. NP Complexity is a measure of how "hard" a problem is. Kaye (2000) tells us that complexity-theory is a way of estimating the amount of time needed to solve a problem, given the *length* of the input. The first class of problems is a class called P, for *polynomial-time computable* problems. These are the problems that, when given an input of length n, can be solved in n^k steps for some exponent k. Kaye writes that these problems are precisely the ones that are practically solvable. Conversely, NP or *Non-deterministic Polynomial-time computable* is a class of problems that are solvable in polynomial time using "non-deterministic" algorithms (i.e. algorithms where the computer is allowed to make some guesses). These algorithems are deemed to be too hard to be solved in practical terms.Kaye (2000) proved that minesweeper is NP-Complete, by proving that it reduces to boolean satisfiability which was previously proven by Cook (1971) to be NP-Complete.

Current work In his work, Kaye (2000) also writes that "Cook's result shows that any NP problem can be reduced to SAT". Because of this property, both minesweeper and its variants and Boolean satisfiability (SAT) have enjoyed a multitude of research such as the work of Su et al. (2016); Mejia-Lavalle et al. (2016) and Meel (2014) on new ways of approaching SAT and the work of Golan (2011, 2014) on variations of minesweeper.

4 Design

We will now follow with a discussion of the design of our implementation.

Overal setup We attempted to design our setup with three concepts in mind: flexibility, extendibility and independence. To this end, we divided the design into three major parts: the game, the agent and the instantiation and reporting framework.

Earlier work Much of the I/O code was taken from assignment 2. For this reason, the file format used closely resembles that of the previous assignment. Much of the `MapCell` and array handling code was also taken and adapted from assignment 2.

The game The game is perhaps the simplest component of the whole setup. It is primarily responsible for giving the agent information about the world when it is requested and to check whether a mistake has been made. Furthermore, to increase independence between the game and the reporting framework, it is capable of running the game loop, and instantiating and reporting on the game if this is not done by the framework, but this is not necessary.

The agent The agent is where all of the logic happens. To increase flexibility, we implemented the strategies as an interface. The framework uses this fact quite handily to easily switch between the provided implementations. We tried to design the agent in such a way that the components were responsible for one thing in such a way that respected the levels of abstraction. In this instance the agent acts as a hub, connecting the strategy to the knowledge base and interacting with the outside world

The knowledge base We wanted to reflect the fact of which cells are flagged, probed and hidden in the knowledge base instead of the cells themselves. That is why we chose to represent the hidden, flagged and revealed cells as `Set`s in the knowledge base. We also store the cells in an array so we can efficiently find neighbours. This does mean some information duplication but it saves in computation, which we thought was a good trade-off. It also has the benefit of improving extendibility, since if we want to change the representation of the map, to, for example, a 3D array, a hexagonal, triangular or a graph, most of the knowledge base code would remain unchanged.

The strategy As mentioned before, the strategies inherit from one another, and implement an interface to increase flexibility and maintainability. We also choose to have the strategies return a list of `MapCell`s which the agent should either flag or probe. We thought this was a good idea because strategies like the easy equation can identify multiple cells at once to be flagged or probed. Finally, we needed a way to know if the returned list is to be probed or flagged. We decided to implement this as a function in the interface and implement it in the base class. That way we only have to set the appropriate flag. This does increase the coupling between the agent and the strategies, but they already work so closely together that we did not think this a big problem.

Reporting framework To be able to test and evaluate the different strategies across the different maps efficiently we made a structure around the game and agent which is able to instantiate the relevant components according to a provided `enum` of the implementations. If another implementation was added, one would only have to add it to this `enum`, and the framework would include it in the test results. Similarly, it will seek out any new experiments in the root directory provided, so that running experiments on new data is also automatic.

The framework has three main ways of reporting data: `VERBOSE, TABLE, REPORT`. In table format, the framework will run all experiments and print all the gathered data in table format as detailed in section 5.2. In `REPORT` mode, the same will happen but the output structure will be different. Here the data will be grouped per map, instead of per variable. Finally `VERBOSE` prints out all the data and intermediate states the agent achieves.

To enhance independence between the framework and the game and agent, we implemented this setup using an observer-observable partner. This way, the game and agent can be used independently of the reporting framework.

5 Examples and Testing

5.1 Testing

Initial testing During the early stages of development we mainly used two forms of testing. Manual inspection of states and outputs and `assert` statements. The `assert` served as micro unit tests, making sure that the things that worked still worked. Furthermore, we visually inspected most of the output and states of the agent and the strategy to verify that the programmes worked correctly.

Framework After most of the strategies and game logic had been implemented, we implemented a way to automatically run tests with different implementations, and print the results in a readable format. This then allowed us to compare results across both the algorithms and maps so that we could correct several bugs in the logic of the game, agent and strategies.

5.2 Examples

A single run A single run of the programme using the easy equation strategy looks as follows:

```
java -jar Logic2.jar ../worlds/easy/nworld1/
                Starting new game
                Probing: (0,0)
                 0  ?  ?  ?  ?
                  ?  ?  ?  ?  ?
                  ?  ?  ?  ?  ?
                  ?  ?  ?  ?  ?
                  ?  ?  ?  ?  ?
                Revealing: (1,0)
                  0  0  ?  ?  ?
                  ?  ?  ?  ?  ?
                  ?  ?  ?  ?  ?
```

```
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,0)
  0   0   0   ?   ?
  ?   ?   ?   ?   ?
    ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (3,0)
  0   0   0   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (1,1)
  0   0   0   2   ?
  ?   0   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (0,1)
  0   0   0   2   ?
  0   0   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (0,2)
  0   0   0   2   ?
  0   0   ?   ?   ?
  1   ?   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (1,2)
  0   0   0   2   ?
  0   0   ?   ?   ?
  1   2   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,1)
  0   0   0   2   ?
  0   0   0   ?   ?
  1   2   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (3,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   ?   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,2)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   ?   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (3,2)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (0,2)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (1,2)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,2)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (3,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (0,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (1,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (2,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (0,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Revealing: (1,1)
  0   0   0   2   ?
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
SPS
Checking Cell (1,2)
Checking Cell (2,2)
Checking Cell (3,2)
Checking Cell (3,1)
Checking Cell (3,0)
Flagging: (4,0)
  0   0   0   2   F
  0   0   0   2   ?
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
Flagging: (4,1)
  0   0   0   2   F
  0   0   0   2   F
  1   2   1   2   ?
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
SPS
Checking Cell (1,2)
Checking Cell (2,2)
Checking Cell (3,2)
Checking Cell (3,1)
Probing: (4,2)
  0   0   0   2   F
  0   0   0   2   F
  1   2   1   2   1
  ?   ?   ?   ?   ?
  ?   ?   ?   ?   ?
SPS
Checking Cell (1,2)
Checking Cell (2,2)
Checking Cell (3,2)
Checking Cell (4,2)
Probing: (3,3)
  0   0   0   2   F
  0   0   0   2   F
  1   2   1   2   1
  ?   ?   ?   2   ?
  ?   ?   ?   ?   ?
```

```
Probing: (4,3)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  ?  ?  ?  2  0
  ?  ?  ?  ?  ?
Revealing: (3,4)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  ?  ?  ?  2  0
  ?  ?  ?  2  ?
Revealing: (4,4)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  ?  ?  ?  2  0
  ?  ?  ?  2  0
SPS
Checking Cell (1,2)
Checking Cell (3,4)
Flagging: (2,3)
  0  0  0  2  F
 0  0  0  2  F
  1  2  1  2  1
  ?  ?  F  2  0
  ?  ?  ?  2  0
Flagging: (2,4)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  ?  ?  F  2  0
  ?  ?  F  2  0
SPS
Checking Cell (1,2)
Checking Cell (2,2)
Probing: (1,3)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  ?  3  F  2  0
  ?  ?  F  2  0
SPS
Checking Cell (1,2)
Flagging: (0,3)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  F  3  F  2  0
  ?  ?  F  2  0
Probing: (0,4)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  F  3  F  2  0
  1  ?  F  2  0
Probing: (1,4)
  0  0  0  2  F
  0  0  0  2  F
  1  2  1  2  1
  F  3  F  2  0
  1  3  F  2  0
        Final number of random guesses: 0
        Final number of probes: 7
        Final number of flags: 5
        Number of runs until success: 1
```

Table report A run form the `ProduceExperimentReport.jar` looks like this:

```
java -jar ProduceExperimentReport.jar   ../worlds/
flags
                          EASY_EQUATION,  RANDOM_GUESS,  SINGLE_POINT,
../worlds/easy/nworld1                5,             5,             5,
../worlds/easy/nworld2                9,             5,             8,
../worlds/easy/nworld3                8,             5,             7,
../worlds/easy/nworld4                7,             5,             7,
../worlds/easy/nworld5                8,             5,             7,
../worlds/hard/nworld1               20,             0,            20,
../worlds/hard/nworld2               34,             0,            33,
../worlds/hard/nworld3               33,             0,            20,
../worlds/hard/nworld4               34,             0,            34,
../worlds/hard/nworld5               34,             0,            34,
../worlds/medium/nworld1             16,             0,            16,
../worlds/medium/nworld2             10,             0,            10,
../worlds/medium/nworld3             16,             0,            17,
../worlds/medium/nworld4             10,             0,            10,
../worlds/medium/nworld5             16,             0,            16,
probes
                          EASY_EQUATION,  RANDOM_GUESS,  SINGLE_POINT,
../worlds/easy/nworld1                7,             5,             7,
../worlds/easy/nworld2                7,             9,             8,
../worlds/easy/nworld3                6,             7,             7,
../worlds/easy/nworld4                3,             5,             3,
../worlds/easy/nworld5                3,             7,             5,
../worlds/hard/nworld1               26,             3,            26,
../worlds/hard/nworld2               18,             3,            20,
../worlds/hard/nworld3               22,             6,            30,
../worlds/hard/nworld4               26,             3,            26,
../worlds/hard/nworld5               29,             6,            29,
../worlds/medium/nworld1             12,             4,            12,
../worlds/medium/nworld2             11,             3,            11,
../worlds/medium/nworld3             23,             7,            23,
../worlds/medium/nworld4             18,             2,            18,
../worlds/medium/nworld5             12,             4,            11,
randomGuesses
                          EASY_EQUATION,  RANDOM_GUESS,  SINGLE_POINT,
../worlds/easy/nworld1                0,             4,             0,
../worlds/easy/nworld2             0,            8,            4,
../worlds/easy/nworld3             0,            6,            1,
../worlds/easy/nworld4             0,            4,            0,
../worlds/easy/nworld5             0,            6,            3,
../worlds/hard/nworld1             0,            2,            2,
../worlds/hard/nworld2             0,            2,            4,
../worlds/hard/nworld3             0,            5,            2,
../worlds/hard/nworld4             0,            2,            0,
../worlds/hard/nworld5             0,            5,            0,
../worlds/medium/nworld1           0,            3,            0,
../worlds/medium/nworld2           0,            2,            0,
../worlds/medium/nworld3           0,            6,            5,
../worlds/medium/nworld4           0,            1,            0,
../worlds/medium/nworld5           0,            3,            1,
runsUntilSuccess
                        EASY_EQUATION, RANDOM_GUESS, SINGLE_POINT,
../worlds/easy/nworld1             1,          120,            1,
../worlds/easy/nworld2             1,          583,            6,
../worlds/easy/nworld3             1,          432,            1,
../worlds/easy/nworld4             1,          327,            1,
../worlds/easy/nworld5             1,           19,            1,
../worlds/hard/nworld1             1,         1000,            4,
../worlds/hard/nworld2             1,         1000,            2,
../worlds/hard/nworld3             1,         1000,            4,
../worlds/hard/nworld4             1,         1000,            1,
../worlds/hard/nworld5             1,         1000,            1,
../worlds/medium/nworld1           1,         1000,            1,
../worlds/medium/nworld2           1,         1000,            1,
../worlds/medium/nworld3           1,         1000,            2,
../worlds/medium/nworld4           1,         1000,            1,
../worlds/medium/nworld5           1,         1000,            1,
```

6 Running

1. Several `.jar` files are included with the submission. All of the `LogicN.jar` files should be run in the same manner: `java -jar LogicN.jar <testDirectory>` The programme expects there to be a file in this directory called `map.txt`. The format of this file is as follows. The first three lines of the file should contain just one integer. The first two should be the length and width of the world respectively. The third should be the number of nettles present in the world. Then the array of the world should follow in CSV format (i.e. rows of integers separated by commas and rows should be separated by new lines). For example:

   ```
   5
   5
   5
   0, 0, 0, 2, -1
   0, 0, 0, 2,-1
   1, 2, 1, 2, 1
   -1, 3, -1, 2, 0
   1, 3, -1, 2, 0
   ```

   Further examples of the file and directory structure that the programmes expect are included.

2. There is another `.jar` file included with the submission called `ProduceExperimentReport.jar`. This file expects the root directory of the experiments as an argument. It will then recursively go through this directory tree looking for files called `map.txt` and running the experiments it finds with all provided implementations and records the data which those experiments report. When all the experiments are done it will output the result in a table format (one for every variable)

7 Evaluation

Variables used In evaluating our systems we monitored four variables across our runs: The number of cells flagged, the number of probed cells, the number of random guesses the agent made and the number of runs it took for a strategy to be successful, which are detailed in table 1. 2, 3 and 4 respectively.

Variable justifications We chose to record the number of runs it took to complete a map by some strategy instead of whether the strategy succeeded or not to better account for the randomness. We did this because the more cells are already uncovered by the agent, the higher the probability of randomly guessing a mine. To reflect this, we thought it better to record how many times it would take to complete a map in order to give the strategies that would have to make random guesses a better chance. This has the added advantage that it carries over well to generated maps which are not guaranteed to be solvable by inference alone. To avoid almost-infinite loops on the harder maps we capped the number of runs the agent was allowed to make at 1000.

We recognise that although giving the agents a number of runs to complete the world is an improvement over the single run, it is still not optimal. One could, for example, have set up everything such that we'd compare an average performance across a fixed number of runs. This would probably have given an even better interpretation. We, however, did not have time to implement this. We also hypothesise that it would have made no difference in the conclusions we draw in this report since the strategies and maps are not complex enough to warrant these sophisticated analyses.

Further more it is important to note that the number of probed cells does not include cells revealed because they are the neighbour of a cell of value 0. If these were included and the algorithm succeeds, the number of probes will always be equal to the size of the map minus the number of nettles. We decided that this would be a less useful metric since it would not show any possible optimisations an agent could have made by clever deduction. Furthermore, this also reflects the cells the agent actively tries to reveal and the cells which are revealed by the world.

7.1 Interpretation of results

Relative performance of the strategies Looking at the data we see a clear hierarchy. The easy equation strategy performs better than the single point strategy which in term is better than the random guessing. This was be expected since every algorithm is an extension of the previous.

Flagged When looking at the flagged numbers we see that the random guess strategy flagged 5 cells in the easy maps and none of the others. That is because the random guess strategy flags all cells if and only if all the remaining cells are nettles (which it knows because this number is provided). So it will only flag cells if it succeeds by randomly guessing. Here we see that it only did so on the easy maps, which was to be expected. It could have also succeeded at the harder maps if we increased the run cap, but we did not find this a relevant experiment.

Probed In the way we set up the experiment the number of cells probed is not a very useful metric, but we included it for good measure anyway. This is because our algorithms are not complicated enough to make clever deductions which could potentially save on the number of probes that would be necessary. This could, however, become a useful metric if one were to develop more sophisticated algorithms. At the moment the number of cells probed is more of a reflection of how the map is laid out (are the nettled clustered or not/ does it contain a lot of 0s?) than how our algorithm performs. Note however that if the algorithm fails, then the probe counter reflects how far the algorithm came on its last run.

Random guesses The amount of random guessing that the agent has to perform is the variable we originally set out to optimise. However, because the maps were guaranteed to be solvable by inference alone, it is hard to make a good comparison between them. We again see the hierarchy we mentioned earlier emerge here, but beyond that, there is not much we can conclude from this data.

Number of runs Again because the experiments and the strategies lack a lot of complexity, there is not an incredible amount we can deduce from the data here. We see that the easy equation strategy completes all of the maps in a single run, as was guaranteed. Although this does provide a nice reassurance that the strategy was correctly implemented. We do see something interesting when we look at the data for the single point strategy, namely that it also completes almost all maps successfully in one run. If we look back at the number of random guesses made, we see that some, although not all of the runs that were completed in one go were solved completely without making a random guess. The maps where the agent did have to randomly guess have relatively few runs and guesses, which leads us to believe that the agent had to make a few guesses early on when still plenty of safe cells were covered and was able to solve the rest of the map from there. This suggests that a strategy which attempts to minimise guesses down the line by making guesses early on could yield significant results.

Flagged Cells			
	EASY_EQUATION	RANDOM_GUESS	SINGLE_POINT
worlds/easy/nworld1	5	5	5
worlds/easy/nworld2	9	5	9
worlds/easy/nworld3	8	5	7
worlds/easy/nworld4	7	5	7
worlds/easy/nworld5	8	5	8
worlds/hard/nworld1	20	0	20
worlds/hard/nworld2	34	0	20
worlds/hard/nworld3	33	0	20
worlds/hard/nworld4	34	0	34
worlds/hard/nworld5	34	0	34
worlds/medium/nworld1	16	0	16
worlds/medium/nworld2	10	0	10
worlds/medium/nworld3	16	0	10
worlds/medium/nworld4	10	0	10
worlds/medium/nworld5	16	0	16

Table 1: Table containing the number of flagged cells by each algorithm per map.

Probed Cells			
	EASY_EQUATION	RANDOM_GUESS	SINGLE_POINT
worlds/easy/nworld1	7	6	7
worlds/easy/nworld2	7	7	7
worlds/easy/nworld3	6	6	7
worlds/easy/nworld4	3	5	3
worlds/easy/nworld5	3	5	2
worlds/hard/nworld1	26	3	28
worlds/hard/nworld2	18	7	26
worlds/hard/nworld3	22	2	32
worlds/hard/nworld4	26	2	26
worlds/hard/nworld5	29	4	29
worlds/medium/nworld1	12	2	12
worlds/medium/nworld2	11	7	11
worlds/medium/nworld3	23	2	24
worlds/medium/nworld4	18	2	18
worlds/medium/nworld5	12	3	11

Table 2: Table containing the number of probed cells by each algorithm per map. Here it is important to note that this does not include cells revealed because they are the neighbour of a cell of value 0

Random Guesses			
	EASY_EQUATION	RANDOM_GUESS	SINGLE_POINT
worlds/easy/nworld1	0	5	0
worlds/easy/nworld2	0	6	1
worlds/easy/nworld3	0	5	1
worlds/easy/nworld4	0	4	0
worlds/easy/nworld5	0	4	1
worlds/hard/nworld1	0	2	5
worlds/hard/nworld2	0	6	3
worlds/hard/nworld3	0	1	1
worlds/hard/nworld4	0	1	0
worlds/hard/nworld5	0	3	0
worlds/medium/nworld1	0	1	0
worlds/medium/nworld2	0	6	0
worlds/medium/nworld3	0	1	9
worlds/medium/nworld4	0	1	0
worlds/medium/nworld5	0	2	1

Table 3: Table containing the number of random guesses made by each algorithm per map.

8 Conclusion

In this report, we implemented an agent designed to play games of nettlesweeper, a simplified version of minesweeper. We implemented several strategies and interpreted the results. This leads us to believe that a strategy which attempts to minimise guesses down the line by making guesses early on, could yield significant results. We could not make any further deductions because the experiments and the strategies involved did not contain large amounts of complexity.

word count: 2382

Runs until successful			
	EASY_EQUATION	RANDOM_GUESS	SINGLE_POINT
worlds/easy/nworld1	1	133	1
worlds/easy/nworld2	1	377	1
worlds/easy/nworld3	1	100	1
worlds/easy/nworld4	1	21	1
worlds/easy/nworld5	1	31	2
worlds/hard/nworld1	1	1000	1
worlds/hard/nworld2	1	1000	4
worlds/hard/nworld3	1	1000	1
worlds/hard/nworld4	1	1000	1
worlds/hard/nworld5	1	1000	1
worlds/medium/nworld1	1	1000	1
worlds/medium/nworld2	1	1000	1
worlds/medium/nworld3	1	1000	2
worlds/medium/nworld4	1	1000	1
worlds/medium/nworld5	1	1000	2

Table 4: Table containing the number of runs until a strategy was successful by each algorithm per map. The number of runs was capped a 1000 so this value means that the algorithm was not successful at all.

References

Cook, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd {IEEE} Symposium on the Theory of Computation*, pages 151–158.

Golan, S. (2011). Minesweeper on graphs. *Applied Mathematics and Computation*, 217(14):6616–6623.

Golan, S. (2014). Minesweeper strategy for one mine. *Applied Mathematics and Computation*, 232:292–302.

Kaye, R. (2000). Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9.

Meel, K. S. (2014). Sampling Techniques for Boolean Satisfiability. *CoRR*, abs/1404.6.

Mejia-Lavalle, M., Jose Ruiz, A., Joaquin Perez, O., and Marilu Cervantes, S. (2016). Modified Neural Net for the Boolean Satisfiability Problem. *Proceedings - 2015 International Conference on Mechatronics, Electronics, and Automotive Engineering, ICMEAE 2015*, pages 64–69.

Su, J., Tu, T., and He, L. (2016). A quantum annealing approach for boolean satisfiability problem. *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, pages 1–6.