Minesweeper Project

Frank Fondell

March 16, 2021

1 Representation:

In my program, the board is represented as a 2D numpy array, all of the information on the board is represented in character form. One grid is generated for the mine locations, and another is used as a knowledge base. In the knowledge base a '?' character represents an unrevealed tile, an 'm' is a confirmed mine tile, and the numbers 0-8 represent the number of mines on an unrevealed tile. Inferred relationships, such as amount of surrounding mines, is generated in the knowledge base when that tile is queried. Neighbors are checked on the mine grid and a number is sent back to be put in the knowledge base as a clue.

2 Inference:

As mentioned above, a new clue is only collected in the case of a queried tile. To update the current state of knowledge, my program begins first by checking if the guess made for that round is actually a mine, then it checks for for the two simple inferences in the project directions to confirm hidden mines or safe tiles. Only after the process is the tile clue added to the knowledge base. My program deduces all information at once after a guess is made, this information however only includes the two simple rules for confirmed hidden mines and confirmed safe tiles. My program doesn't deduce all possible information that it can, however there are many ways to improve this. Many rules exist online apart from the one implemented in my improved agent, these rules are very complicated and look at specific locations on the board and reveal hidden mines based on specific formations of clue numbers.

3 Decisions:

My program has a few ways of deciding what squares to choose at a given state of the board. If it's the first pick, my program will always guess a square at random. Next in priority is the fringe, each round, the board is scanned for tiles with a '0', all neighbors of that tile are added to the fringe. A guess is then selected from the first item in the fringe. In my original implementation, If there are no more guesses in the fringe, a random unrevealed coordinate is selected and queried.

4 Performance:

For the improved method, there is a play by play below. This represents six rounds of the game, the grid has one more space filled in at each step. The first three are shown here:

```
[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['?' '2' '?' '?' '?']
 ['?' '?' '?' '?' '?']
 ['?' '?' '?' 'm' '?']]


[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['0' '2' '?' '?' '?']
 ['?' '?' '?' '?' '?']
 ['?' '?' '?' 'm' '?']]


[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['0' '2' '?' '?' '?']
 ['0' '?' '?' '?' '?']
 ['?' '?' '?' 'm' '?']]
```

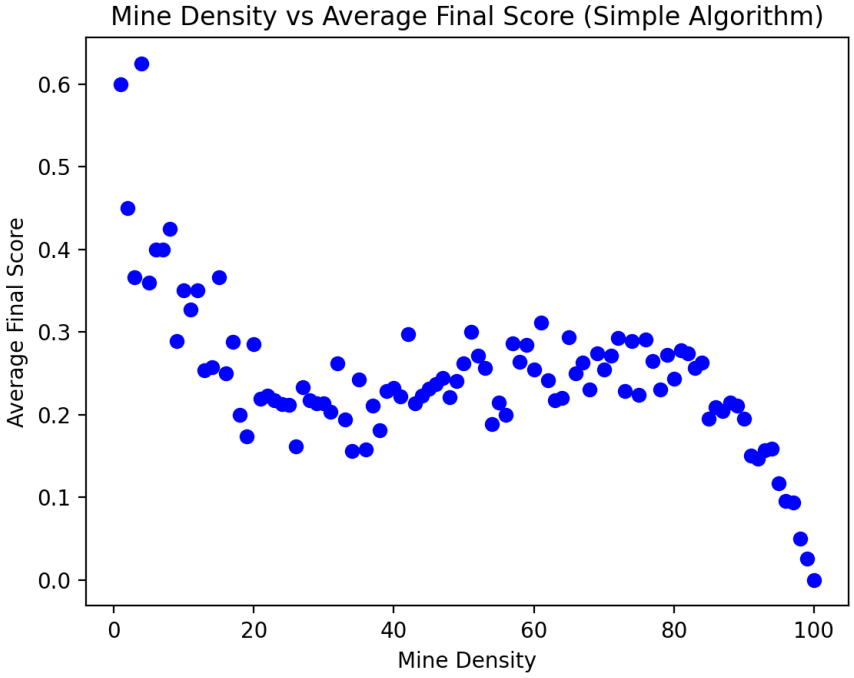
The last round above shows the grid being setup for my One Two Punch rule to take affect. There aren't any steps where my program makes a decision that I don't agree with, my program is designed in an order that takes the most optimal guesses first. I was delightfully surprised that my algorithm could detect the One Two Punch rule as soon as the necessary spaces were filled to do so.

```
[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['0' '2' '?' '?' '?']
 ['0' '?' '?' '?' '?']
 ['?' '?' '?' 'm' '?']]


[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['0' '2' '?' '?' '?']
 ['0' '2' '?' '?' '?']
 ['?' '?' '?' 'm' '?']]
Found hidden mine with method


[['0' '1' '?' '?' '?']
 ['0' '1' '?' '?' '1']
 ['0' '2' '?' '?' '?']
 ['0' '2' 'm' '?' '?']
 ['0' '?' '?' 'm' '?']]
```

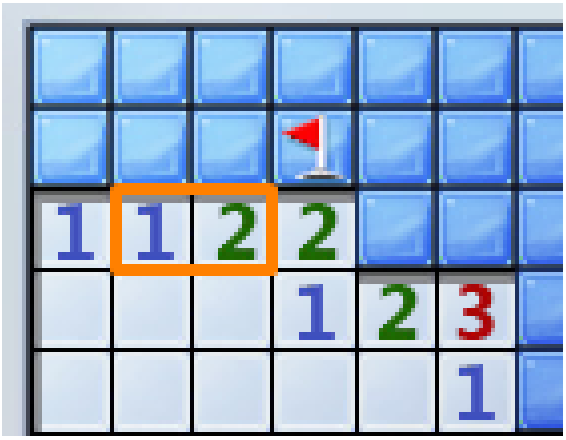
As displayed above, the mine is filled in at (3,2) as soon as the rule becomes

true. After my trigger message is sent, the knowledge base can be seen with the mine.

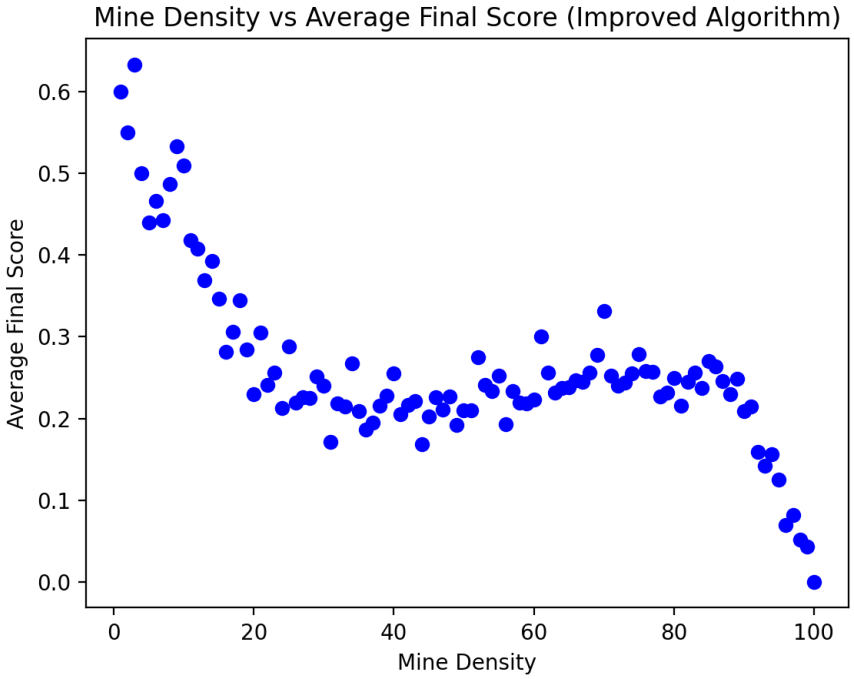
To achieve results with trend, for a range of mine densities 1 to 100, I used a dimension of 10 by 10, and included 10 trials for each mine density to get a quality average. The graph makes sense with my intuition for the most part. It seems to slowly get easier as mine density passes from 30 to 80 which is interesting. I would think that increasing mine density would directly correlate with a lower final score but it does not appear to be the case. Minesweeper becomes hard as it reaches 80%, with a sharp down trend on the way to 100%. Pictured below is the graph:



My improved algorithm uses a logic rule to have an edge over the simple algorithm. The algorithm is called the One Two Punch. When a one is next to a two in any orientation and there is a wall of hidden blocks parallel to them, the block diagonally away from the two in the opposite direction of the one will always be a mine. An example of the orientation is shown below:



As shown in the performance graph displayed below, the improved algorithm shows a few differences between itself and the simple. There appears to be a decently higher average final score when mine density is between 0 and 20 compared to the simple algorithm. The Improved algorithm also shows slightly better performance near the end, an uptrend can be seen that lasts until mine density approaches 90%. This uptrend lasts longer than the simple algorithm and shows that the improved algorithm can withstand a higher percent of mine density. The only edge the simple algorithm seems to have is in the middle of the graph, between 40% and 70%. The average final score can be seen to just be slightly lower in the improved algorithm between 40% and 70%. I concluded that this variance is just random because I don't see a way that this new implementation could harm the performance of the algorithm. I can see however though that the improved algorithm is much better at lower mine densities because it works with only 1s and 2s, at very high mine densities, the total amount of 1s and 2s on the board is lower and therefore decreases the chance of triggering the rule. In direct comparison, my improved algorithm tends to work things out 1% more than the simple.



The simple algorithm final average successes for 3 different graphs with 10000 trials are shown below:

```
0.24585884674717068
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.24228749562835164
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.2428324111706662
```

The improved algorithm results show the performance increase in the improved algorithm:

```
0.24372115452358356
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.2560143361178739
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.2554053608734517
```

5 Better Decisions:

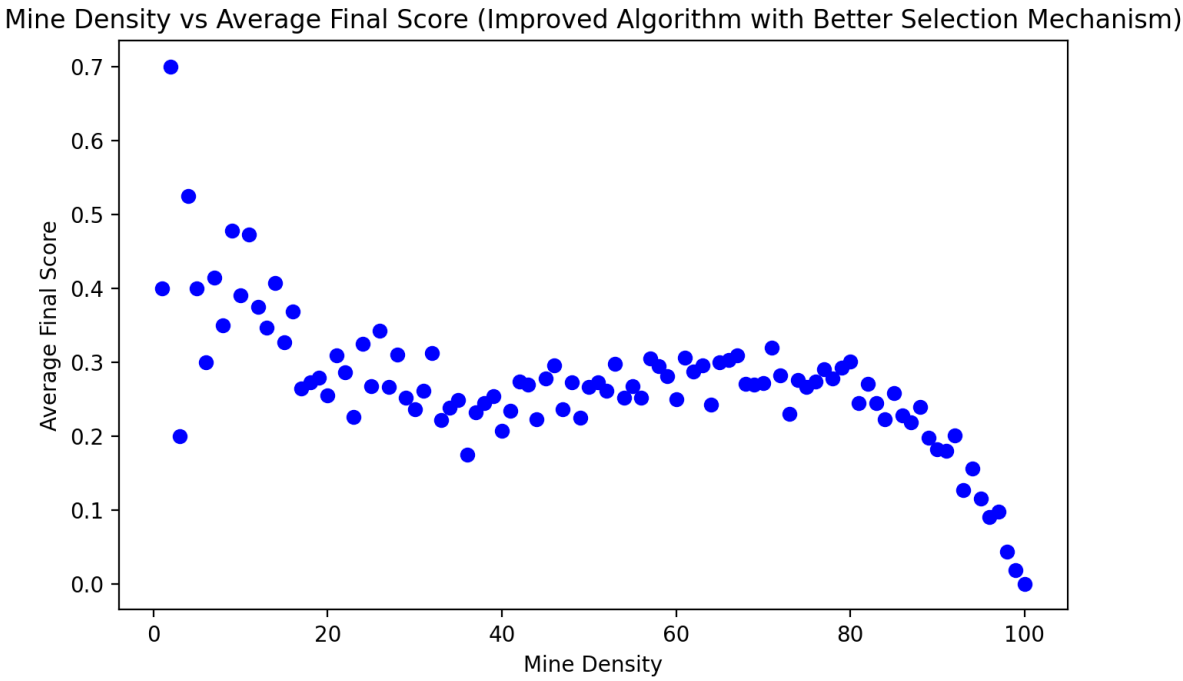
To improve the success rate of my program, I came up with a new way to select next guesses other than just getting a random coordinate. My program retrieves the corner tiles first, then the edge tiles, and then adds them to the end of the fringe, still giving priority to tiles with neighbors of 0. Since corner tiles have only 3 hidden neighbors, they have the least chance to have a mine surrounding them, giving them a greater chance to reveal a '0'. Furthermore, in the directions, the first rule states that "If, for a given cell, the total number of mines (the clue) minus the number of revealed mines is the number of hidden neighbors, every hidden neighbor is a mine." And the second rule states "If, for a given cell, the total number of safe neighbors (8 - clue) minus the number of revealed safe neighbors is the number of hidden neighbors, every hidden neighbor is safe." With this in mind, if there are less hidden neighbors to work with for a given tile, the chance of this either of these rules triggered is increased, and it shows in the results. I printed the final average success rate as an output to see the exact number instead of looking at the graph first. The first screenshot of the terminal below represents the final average score of 3 different graphs generated with 10000 different scores each using the improved algorithm with random guesses:

```
0.24372115452358356
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.2560143361178739
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.2554053608734517
```

The second screenshot of the terminal below represents the average final score when the educated guesses were implemented. As there were so many trials to strengthen the data, there appears to be a clear 2% increase in performance:

```
0.2787299798705679
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.277570540552588
(base) Franks-MBP:~ frankfondell$ python /Users/frankfondell/Documents/Minesweep
er.py
0.28125086125693965
```

To visualize it I've included the graph below also. At first glance looking at the graph, it seems to have a harder time performing at lower mine densities. However, it seems to perform nearly 5-10% better on average between the mine density range of 30-80.



6 Efficiency:

During the writing of this program, there were a few space and time constraints. Problem specific constraints that I could not control include space used to store size of board, and time it took to query and fill in every tile for a clue. In terms of implementation, there were a few things I could improve upon if I could figure it out. For large board with low mine density, the fringe would have to use space to contain guesses and contain those guesses for quite a few rounds. My fringe grows fast and shrinks slowly. My implementation of guesses could have been more optimal too. The algorithm (both simple and improved) iterate through the board more often as the board gets filled with squares that cannot be guessed anymore. I could improve this someone by find a way to only reference or determine the next best guess without looking at the entire board at once.