CPSC 481 PROPOSAL - MINESWEEPER SOLVER

Group Member: Nghia Phan

Problem Description:

Minesweeper is a popular single-player puzzle game where the player is tasked with clearing a board without detonating hidden mines. My project aims to develop a Minesweeper solver that can play the game automatically with logic and probability deduction. I also experiment with the approach of incorporating machine learning for auto-player in this project.

Programming Language: Python for both the interface and the solver.

Datasets: Datasets for Artificial Neural Network will be generated by the Probability Player. The approach will be mentioned in Algorithm/Approach section.

Existing Code:

The already-made game interface for Minesweeper will be used. AI auto player using logic and probability will be added to play the game. Data analysis of the game play will be added as a summary.

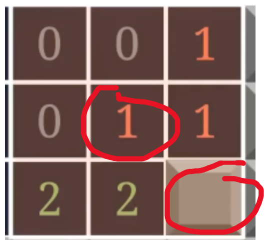
Algorithm/Approach:

My approach to developing the Minesweeper solver is based on 3 strategies: Logic Rules, Heuristic probability and Artificial Neural Network (ANN model)

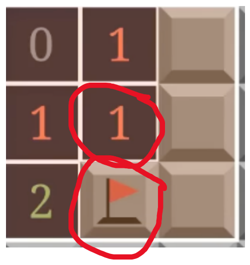
I. LOGIC RULES:

Algorithm: Pseudocode Logic Player

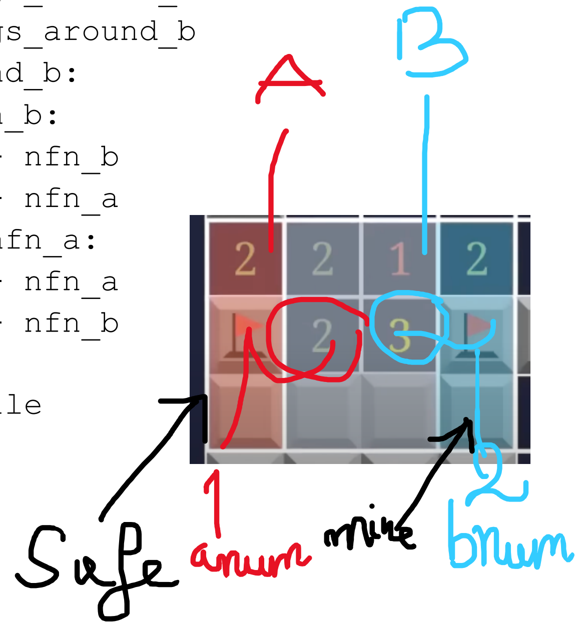
```
1. First move: randomly open at one of the four corners of the
   board

2. Strategy 1:

   For each open tile,
       check the number of unopened tiles around it
       if tile_number =  len(unopened_tiles):
```



```
                    flag all the unopened_tiles # certain to be mine

               if tile_number =  num_of_flags,
                    open all the unopened_tiles
                    # certain to not have mine
```



```
   3. Strategy 2:

               For a pair (a,b) of opened_tile adjacent to each other:

               let nfn_a = set(unopened tiles around a)
               let nfn_b = set (unopened tiles  around b)
               let unknown_around_a = a_num - flags_around_a
               let unknown_around_b = b_num - flags_around_b
               if unknown_around_a > unknown_around_b:
                    if a_num - b_num = nfn_a - nfn_b:
                         flag all tiles in nfn_a - nfn_b
                         open all tiles in nfn_b - nfn_a
                    elif b_num - a_num = nfn_b - nfn_a:
                         flag all tiles in nfn_b - nfn_a
                         open all tiles in nfn_a - nfn_b

   4. If no move can be made, randomly open a tile
```

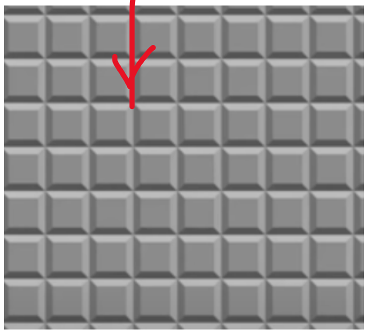


II. HEURISTIC APPROXIMATION FOR PROBABILITY

Algorithm: Pseudocode Probability Player

```
1. Extends all functionalities of the Logic Player
2. def probability():
     For each of unopened_tiled:
          count = 0 # count the repeated occurrence of a tile
          neighbors = all of its opened neighbors
```

$p = \frac{\text{Mine}}{\text{unopened}}$



```
          if len(neighbors) = 0:
               prob = mine left / number of unopened_tiles
               count += 1
     # all of the 9 tiles in the square are unopened,
     global probability
          Else:
               for each tile in neighbors:
                    u_neighbor = unopened neighbors
                    flags = number flag around the tile
                    mine_left = tile_num - flags
                    prob += mine_left/ len(u_neighbor)
                    count +=1
     # local probability, heuristic approximation
          # normalize probability
          final_prob = prob / count
```

```
3. If Logic algorithm stuck,
          apply the probability function.

4. Greedily choose the one with least probability

5. At the same gather information and put all data in csv file for
   training the ANN model later on.
```

III. ARTIFICIAL NEURAL NET

I also incorporate the Artificial Neural Network to make decision on top of the Greedy Policy.

Rather than the player greedily choose the one with lowest probability, choose one based on the decision of the ANN model.

The ANN model will return prediction of if there is mine based on the heuristic probability. Through training, the ANN will pick up and correct the accuracy of the Greedy Probability Player

Advantages:

- Small datasets requires (some thousands - tens of thousands is sufficient)
- Work best for big map

Disadvantages:

- Work on par as the probability player on small map but requires more computation power to run

I generate the data for the Neural network player the while Probability Player is running, because the ANN also needs to know the probability data of the current unknown tiles.

<u>Approach:</u> Pseudocode Neural Net Player

```
1. Extends all the functionalities of Probability Player

2. ANN policies:
   If the Logic Player stuck
        Probability Player calculate the probability of an
        unopened tile being mine

        Greedily select groups of Lowest, Second_to_lowest,
        Third_to_lowest of probability

        In turn feed each candidate of a group to the ANN model
        using model.predict()

             If the ANN says there is likely a mine, go to next
             candidates.

             If all candidates of group denied by ANN model, go
             to the next group.

             Probability will be updated before being added to
             enrich the dataset.
```

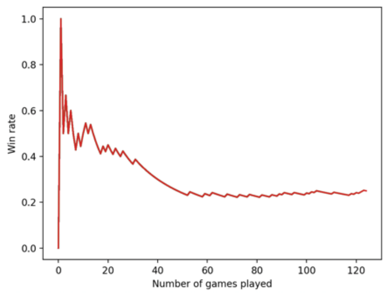
Complexity:

Mine Sweeper is NP-Complete.

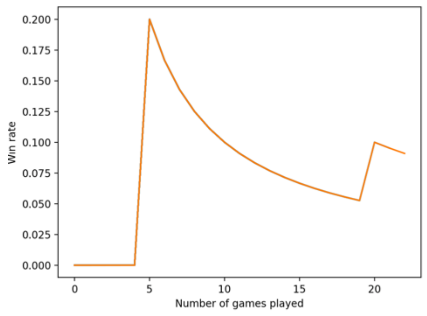
Statistics:

Probability Player:

On **30 x 16 expert mode** with **99 mines**



On **80 x 45 custom mode** with **740 mines**



Classes - Knowledge Requirement Involved:

1. CPSC 481 - Artificial Intelligence
2. CPSC 483 - Introduction to Machine Learning
3. MATH 338 - Statistics Applied to Natural Science
4. CPSC 121 - Object Oriented Programming
5. CPSC 223P - Python Programming