COMPUTATIONAL INTELLIGENCE PROJECT REPORT

ExpectiMax approach to solve Minesweeper

Submitted by:

Harith R

Rahul Menon

Adarsh Kuruppath

Computer Science & Engineering

S6

April, 2017

Contents

1. INTRODUCTION

Minesweeper is a single-player puzzle video game. The objective of the game is to find and mark all the randomly arranged mines on rectangular board, with help of clues about the number of neighbouring mines for each tile. The game originates from the 1960s, and has been written for many computing platforms in use today.

1.1 GRID GENERATION

Initially, the player is presented with a grid of identical squares. The grid size and the number of mines in the grid are selected by the player. The mines are placed randomly onto the grid.

The grid is generated as follows:

1. Size of the grid (M x N) and Number of mines (k) are taken as input
2. The grid is implemented as a 2-D matrix of size M x N.
3. The k mines are then randomly placed into the matrix.
4. The whole matrix is then scanned cell by cell to determine the neighbouring mine count for each cell. This is done by counting the total number of mines in the 3×3 grid centred at that particular cell.

Let’s take an example:

Consider the grid size to be 3×3 and the number of mines to be 3. The configuration of the grid after randomly placing the mines could be:

M	2	1
2	3	M
M	2	1

1.2 PROBING ALGORITHM

Once the grid has been generated, the solving process begins. Since the player has no information about the position of the mines, the first cell chosen to probe is random. Here's how the probing algorithm works:

1. Let's assume the first cell to be chosen was x.
2. If x is a mine, game is over and the player loses.
3. If x is not a mine, and if there are no mines under any of the 8 adjacent cells, then whole 3×3 grid centred at x gets uncovered.
4. If x is not a mine and there is at least one mine in the adjacent 8 cells, then only x gets uncovered.
5. For each uncovered adjacent cell, the steps 3-5 apply.

The following sample grid illustrates this process :

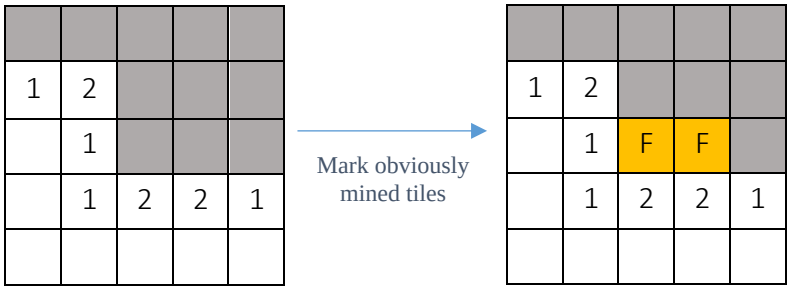
				1	2	2	1	
				1	M	M	1	
1	1		1	2	3	2	1	
M	1		1	M	2	1		
1	1		1	2	M	1		
1	1	1		1	1	1		
1	M	2	1	1	1	1	1	
1	1	2	M	1	1	M	2	1
		1	1	1	1	2	M	1

- If the choice is cell [2][6] (highlighted in dark red), then the game is lost
- If the choice is cell [5][3] (highlighted in black), then all of the 3x3 grids centred at [5][3], [4][3], [3][3] and [2][2] (highlighted in grey) would be uncovered.
- If the choice is cell [8][6] (highlighted in green), then only that cell will get uncovered

2. SOLVING THE GRID

2.1 INITIAL NAIVE HEURISTIC

Initially the algorithm is designed to play the game in the way how humans would play it: By marking the **obviously mined tiles** and then making a probabilistically safe choice. Consider the grid below

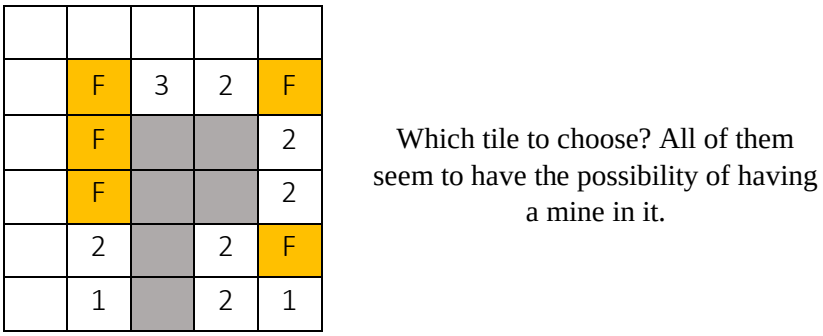


There are two thumb-rules which humans use to play this game:

1. If a numbered tile has the same number of uncovered neighbors as the number on the tile, then all of these neighbors are "**obviously mined**".
2. If a numbered tile has the same number of obviously mined neighbors as the number on the tile, then the rest of its uncovered neighbors are **safe or unmined**.

After marking the **obviously mined** and **safe tiles**, the human chooses a **safe tile**, (in the example above, the tile in the 3rd row and 5th column is deduced to be **safe**).

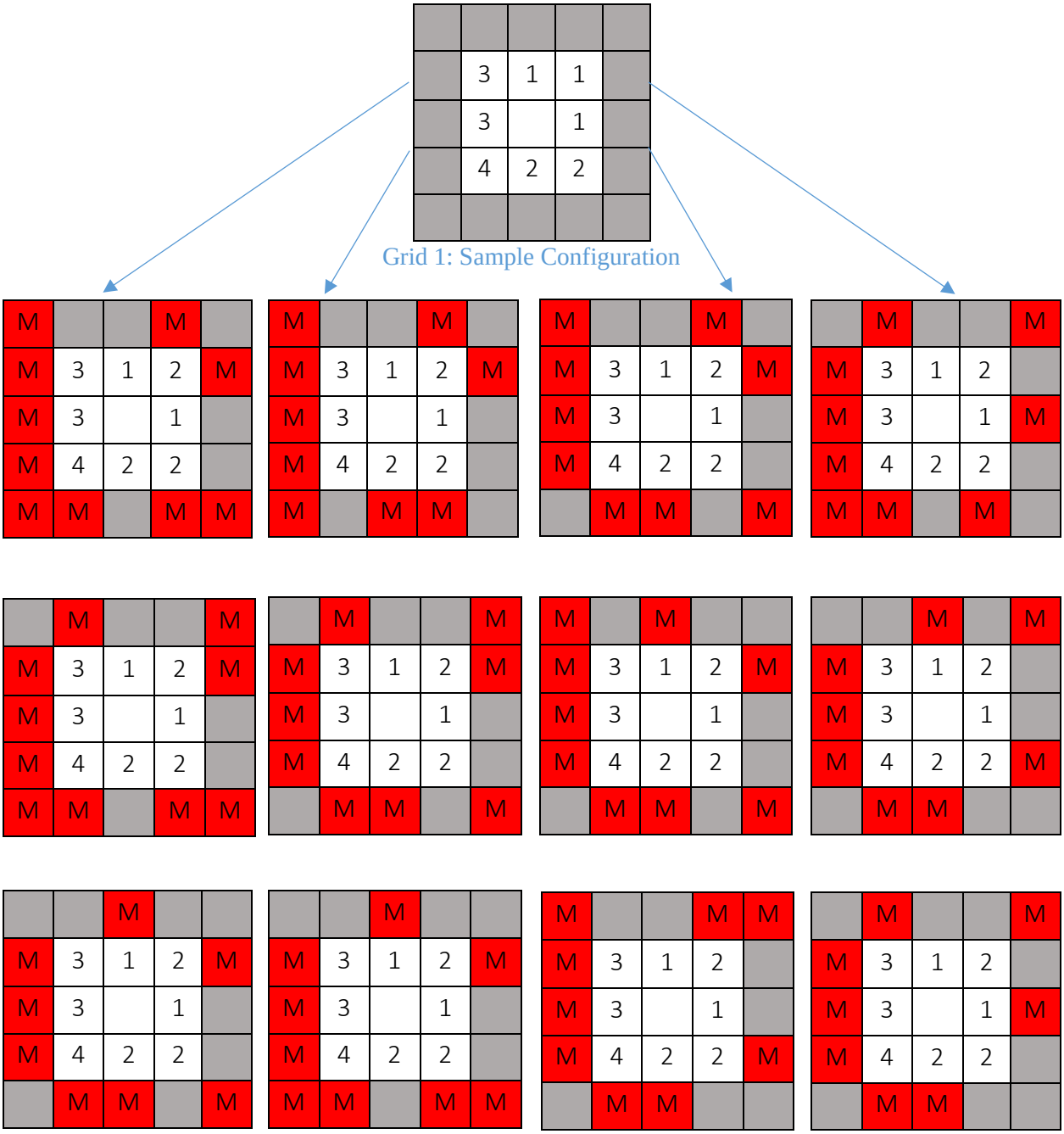
This approach fails when in every possible arrangement of mines, for every border tile there is **at least one** arrangement in which that tile has a mine. At this point, the human is forced to **guess**. Here's an example of such a situation:



The danger count heuristic, discussed below, illustrates how to make an informed decision in this kind of situation.

2.2 CALCULATION OF "DANGER COUNT" HEURISTIC

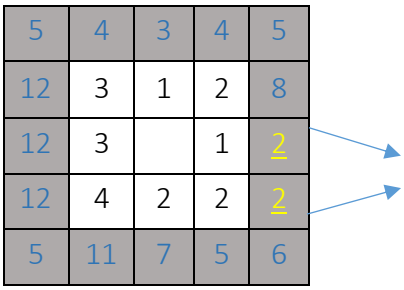
The procedure of calculating this heuristic is best explained with an example. Consider the following configuration of a section of an m*n minesweeper grid, with a total of k mines:



Grid 1: Sample Configuration

All possible arrangements

Grid with Danger Count for border tiles



A border tile is any covered tile neighboring a numbered tile. At any given stage:

b = *no. of border tiles* **u** = *no. of uncovered tiles* **o** = *no. of obviously mined tiles*

o_b = *minimum no of mines in border tiles among all unlucky arrangements*

P_e = *probability of losing the game upon clicking a non-border tile,* $(k-o-o_b)/(m*n-u-b)$

N_t = *no. of arrangements of mines for the current set of border tiles*

DC (for border tiles) = *no. of arrangements in which that border tile has a mine*

P_b = *probability of losing upon choosing a border tile* (DC / N_t)

2.3 GENERATING MINE ARRANGEMENTS

Generation of all arrangements of mines is computationally intensive. For the first move, knowledge of these arrangements is irrelevant, as it's always random. Hence, it is optimal to generate the arrangements **only after the first move**. Keep them in a set A.

An **unlucky arrangement** is any arrangement of mines in which the border tile with the lowest danger count has a mine.

2.4 ALGORITHM

1. After each move, flag the **obviously mined** and **safe tiles**.
2. Update P_e.
3. With the information of numbered, border, obviously mined and safe tiles, removing all the invalid arrangements from set **A** would give a valid set of arrangements of mines for the current set of border tiles.
4. Update P_t for all border tiles.
5. Let **S** be the set of border tiles, with the lowest value of P_t. If this value is lower than P_e, choose any border tile in **S**. If not, choose any non-border tile.

Repeat this process until **k=o**, implying that all of the unmined tiles have been uncovered.

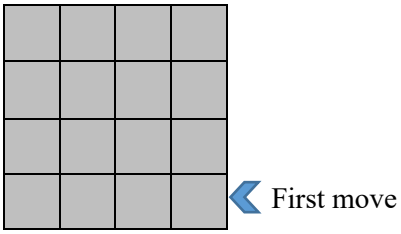
2.5 SOLUTION OF AN EXAMPLE GRID

The following steps illustrates a run of the algorithm on a sample grid.

M	2	1	
2	M	1	
2	2	1	
M	1		

Grid Size : 4x4 **Mine Count : 3**

1. The player is given a completely blank 4x4 grid, with the number of mines in the grid stated to be 3. The objective is to find the hidden mines.:



2. Start by randomly clicking one tile. Suppose the player clicks the bottom right tile. Then, the game reveals the following.

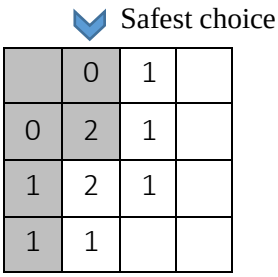
		1	
		1	
	2	1	
	1		

3. Now there are two possible arrangements of the entire grid, where M denotes a mine.

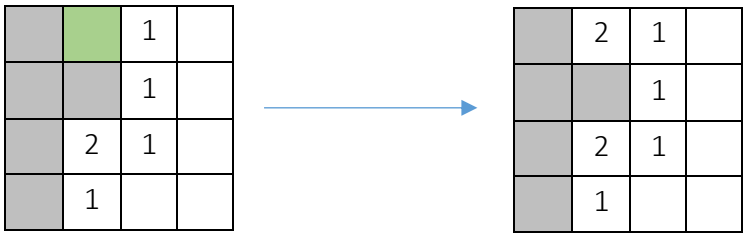
M		1	
	M	1	
	2	1	
M	1		

M		1	
	M	1	
M	2	1	
	1		

4. Now assign the danger count on all the border tiles which will act as the heuristic to decide the next tile to click. The danger count of a tile is the total number of arrangements in which the tile will have a mine (M).

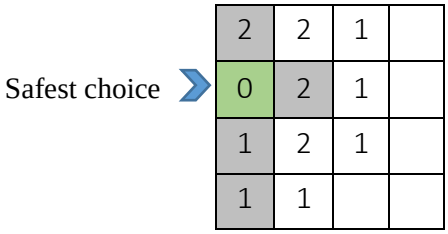


5. Choose the tile with the lowest heuristic (highlighted in green).



6. Repeat the process. Here are the possible mine combinations:

7. Set the danger count heuristic for each border tile



8. Select the lowest heuristic tile:

	2	1	
2		1	
	2	1	
	1		

9. Generate the possible mine combinations. There is only one possibility.

M	2	1	
2	M	1	
	2	1	
M	1		

10. Assign the heuristic to each uncovered tile.

	1	2	1	
	2	1	1	
Safest choice	0	2	1	
	1	1		

11. Select the lowest heuristic tile

1	2	1	
2	1	1	
2	2	1	
1	1		

12. There are only 3 uncovered tiles remaining which is equal to the total number of mines. The game is now complete:

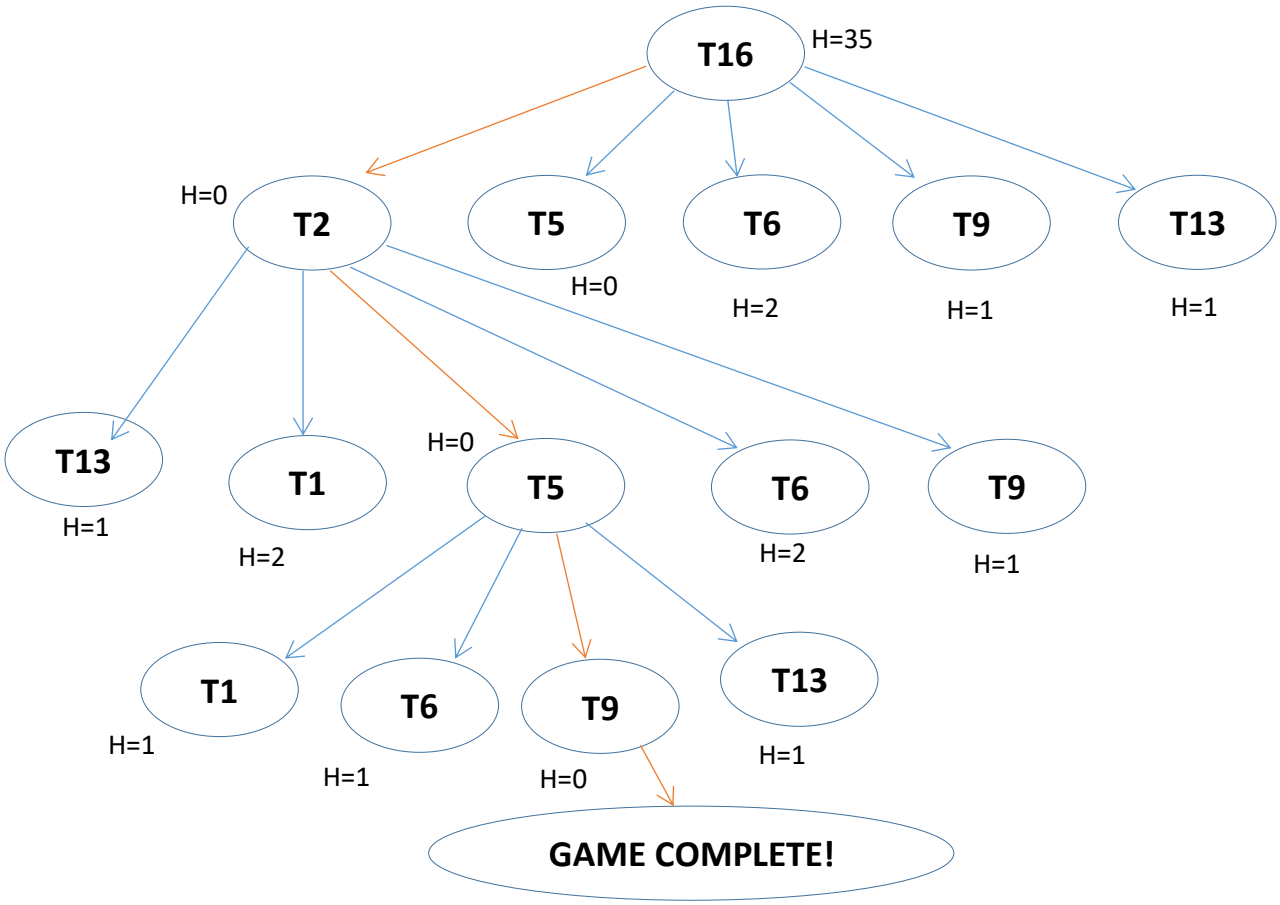
M	2	1	
2	M	1	
2	2	1	
M	1		

2.6 GRAPHICAL REPRESENTATION

The tiles are numbered as shown in the following grid :

T1	T2	T3	T4
T5	T6	T7	T8
T9	T10	T11	T12
T13	T14	T15	T16

In the graph shown below, the root node denotes the first tile chosen, which is random. At every level, each node denotes an available border tile. The number of possible arrangements for this grid is ${}_{16}C_3 = 560$. Therefore, for any given tile, there are 35 arrangements in which the tile has a mine, implying that the **danger count** of any tile for the first move is 35. In the tree shown below, going down a level indicates that a move has been made. This can be considered as an **ExpectiMax** search on a state space graph, where the value of each node would be 1 - (probability of losing by choosing the safest tile) = 1 - (**min** (P_b, P_t)). The ExpectiMax search would choose the highest valued node at each level.



2.7 PARALLELIZABLE COMPUTATIONS

There are several computations in this algorithm that are parallelizable and can take advantage of multiple processors. Considering a quad-core processor:

1. **Generation of all arrangements of mines** – each processor can take one quadrant of the grid (assuming the total no. of mines < area of the quadrant) and generate arrangements in which the mines are contained within the quadrant. Each quadrant can be divided further, until the quadrant area is less than the number of mines.
2. **Marking obviously mined and safe tiles** – the set of numbered tiles can be split into 4 subsets for each processor.
3. **Removing invalid arrangements** (**step 3 of the algorithm**) – the strategy is similar to that of generation of all arrangements of mines.

3. CONCLUSION

The minesweeper solving algorithm works very well for almost every grid arrangement. The caveat with this approach to solving the game is that at one point or the other, the player is forced to make a random guess if the lowest danger count is possessed by more than one tile, or if it's safer to choose a non-border tile. While the aim of this algorithm is only to win the game, regardless of the number of moves, a modified ExpectiMax search algorithm can determine the best set of choices to achieve a high score.