Perfect Minesweeper AI Using Greedy and Probability Calculation

Hafidh Rendyanto 13517061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517061@std.stei.itb.ac.id
hafidhrendyanto@s.itb.ac.id

***Abstract*—In this paper we discuss the use of greedy algorithm and probability calculation to make a perfect AI for minesweeper that only have 5% chance of losing. Minesweeper is a popular game in which we are given mxn matrix of cell with k bombs and try to "guess" in which cell the bombs are.**

***Keywords*—Minesweeper, AI, Greedy, Probability Calculation.**

I. Introduction

Minesweeper is a very popular computer game that is very underestimated in the current age where we could be bombarded by countless games coming from hundreds of game developer big and small. The reason why minesweeper is very underestimated and forgotten is actually very clear, it is a very simple game in which the outcome of the game "seems" like it is only decided by chance, the luckier you are, the better you are at this game. But this notion is actually very misleading, minesweeper is actually a game that requires great analysis and logical skill. If you can use the hint that the game gave you, you could actually win the game almost every time.

In this paper we are going to demonstrate my claim above using algorithm that use greedy algorithm and probability to simulate the logic in how to solve a minesweeper problem

In Computer Science, greedy algorithm is a computer algorithm that try to approach the most optimal solution from a problem using the current most optimal solution or what we like to call "local solution". But before we delve deeper into greedy algorithm, we will focus our attention to what an algorithm actually is.

Algorithm is an ordered list of step to solve a problem. This problem can range from cooking a meal for dinner to making sure your back transaction can be resolved securely. This definition of algorithm might seems too broad but actually that is what algorithm really is.

There are likely thousand of algorithm that exist out there just to solve one thing but we are only interested in the ones that are good which is the one that have low complexity. Lower complexity means the execution time will be lower and most of the times the algorithm is more elegant than most.

There are many use of greedy algorithm, most of which is finding the most optimal solution of a problem in the most fastest way possible. Some example are, finding closest path from A to B, maze problem, knapsack (in which we want to carry as many as we can with limited space) and many more.

The neat thing about greedy is that you can use it to solve a hard problem that doesn't require too much precision, just like the problem that we are going to solve in this paper.

II. Brief explanation of algorithm

A. Definition

The word algorithm is used widely in computer science, mathematics, and science in general. The formal definition of this word is that algorithm is an unambiguous specification of how to solve a class of problem. Algorithm can perform calculation, data processing, and automated reasoning.

Algorithm is used in every part of our daily life, from solving complex problem, proofing mathematical theorem, and even food recipes can be categorized as algorithm. But most importantly, algorithm gives the means to convey our ideas to other, and to make our life easier.

In this paper we are going to talk about the type of algorithm used in data processing, vis, sort algorithm. Sort algorithm itself is a type of algorithm that is used to make an ordered set of an unordered one. This type of algorithm has many applications like what I have described in the preceding chapter.

B. Brief History

Over the course of human history, the word has developed from precise definition of process to something more abstract in this day and age, the reason will be explained shortly. I will describe briefly about the history of the word below,

Ancient Greece

There are two examples of the use of this word in ancient Greece, vis, Sieve of Eratosthenes, which was described in Introduction to Arithmetic by Nicomachus, and the Euclidean algorithm, which was first described in Euclid's Elements.

The Advent of Algebra

The advent of algebra gives birth to the use of variable to solve mathematical problem, which also gives birth to algorithm to solve such problem.

Advancement of Mathematics

Algorithm is expanded in the field of mathematics to solve even more problem and even given its own place of study in Discrete Mathematic, a subset of math.

The Advent of Computer

In this event of human history, the use of the word is not expanded, it is rather, shrink to be used mainly in the field of computer science to describe a set of process that computer go through in order to solve problem.

Current Age

In this day and age, the word expanded again to not only describe precise description of some process but also to model how intelligence work in the field of artificial intelligence. Even if this definition of "algorithm" does not match precisely with artificial intelligence, but I still believe that even the most sophisticated machine intelligence (using neural net or some other kind of advanced implementation) could still be described using algorithm.

C. Classification

There are many ways to classify algorithm, each with their own odds and ends.

By design paradigm

Paradigm can be described as point of view, or way of thinking. With different point of view to a problem, one can create a completely different algorithm to a problem.

a) Brute-force
   This is the naïve method of trying every possible solution to see which is best.
b) Divide and Conquer
   A divide and conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments.
c) Search and Enumeration
   Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.
d) Randomized Algorithm
   Such algorithms make some choices randomly (or pseudo-randomly). They can be very useful in finding approximate solutions for problems where finding exact solutions can be impractical (see heuristic method below).
e) Reduction of Complexity
   This technique involves solving a difficult problem by transforming it into a better-known problem for which we have (hopefully) asymptotically optimal algorithms.
f) Backtracking
   In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution. One of its popular use is its use in logic programming

Optimization Problem

For optimization problems there is a more specific classification of algorithms

Linear Programing

If the problem that is going to be solved have linear equity and inequality constraints, the constraint of this particular problem can be used directly in producing the optimal solution, such is the definition of this classification

Dynamic Programing

Dynamic Programing use the "infinite" memory of computer to its advantage, it save the result of some computation in memory to avoid recomputing it in the future.

The Greedy Method

A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution.

The Heuristic Method

I think heuristic algorithm use similar paradigm in what was used in calculus. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Just like calculus.

III. Analysis and discussion

To create our AI, we first need to define some rule that we are going to follow in order to solve our minesweeper problem. We are going to first define some logical rule

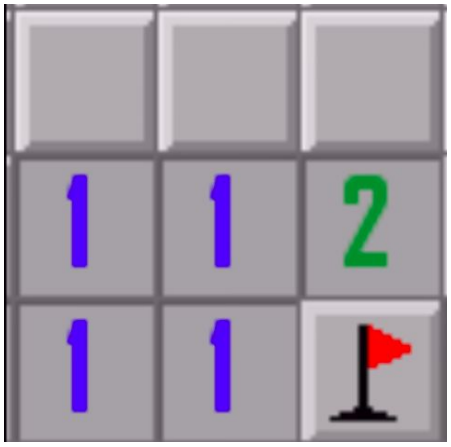
The first rule, *if a tile has the same amount of hidden tiles around it as unflaged bomb, then all the hidden tiles are bomb.*

This is a very simple rule, the basis of minesweeper. Consider this example,

The “2” in the center have only one flagged bomb in its lower right. Then that means the tile above it must be a bomb.

The second rule, *if a tile has the same amount of flags around it as the number on the square then all remaining hidden tiles around it are not bomb*

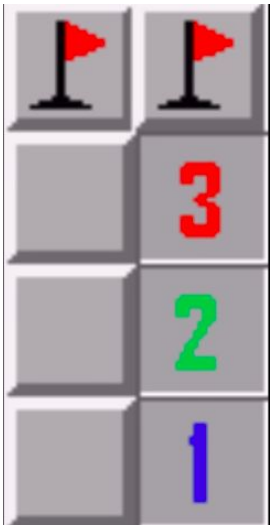
Consider this example,



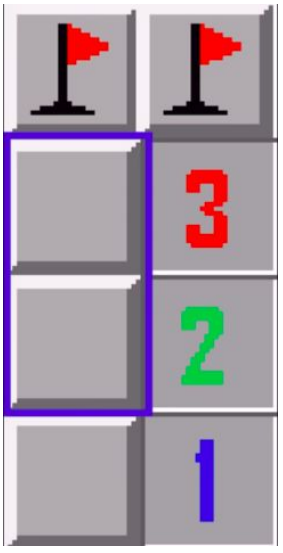
The one in the center already have one flagged tile in its lower right, that means all other tiles around it are safe.

Now, with this two rules in set, the AI already able to perform pretty well. But still has to guess far too much. So lets add more complex logic.

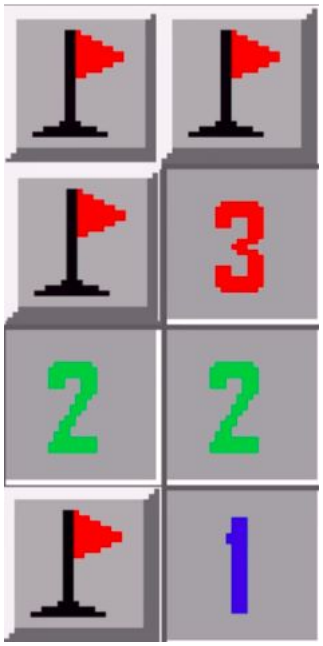
Consider this example,



With our current logical rules, the AI cannot solve this problem, but it can be easily solved. Note that the red “three” only needs one more bomb that means the other two tiles beside it can only have one bomb between them



Now, please look at the green “two”, it needs two bomb but it can only have one bomb in the highlighted space, so the tile beside the blue “one” must also have bomb.



We can keep adding hard coded rules for ages but we still can't make sure that we got them all. What we need it someway to *combine* all of these posible rules into a single rules. And that where the probability and greedy part of this paper comes in. If we can find a way to calculate the probability of each tiles being a bomb then we can use that as our greedy parameter. Choosing the one tile that has the lowest probability of being a bomb. This way we can actually emulate all possible rules that we tried to hardcode before.

So how can we calculate the probability of a bomb in a cell? Simple, all we need to do is enumerate all possible arrangements of bomb that still has not been found then count all the arrangement that have bomb in current tiles then divide it by all the arrangement.

With this information, we can use probability to find the most safest cell to progress our game.

Source Code

```
class Game {
    private:
        int nRow, nColl, bombCount;
        char** gameboard;
        bool isLose;

        int adjBomb(int _i, int _j);
        void reccOpen(int _i, int _j);

    public:
        /** ctor */
        Game(int, int, int);
        /** dtor */
        ~Game();

        bool Lose();
        bool Win();

        void printGameBoard();

        void input(int, int);
};

#include <stdio.h>
#include "Game.h"

Game::Game(int _nRow, int _nColl, int _bomb)
{
    srand(time(0));
    nRow = _nRow;
    nColl = _nColl;
    bombCount = _bomb;
    gameboard = new char*[nRow];
    for (int i = 0; i < nRow; i++) {
        gameboard[i] = new char[nColl];
    }
    for (int i = 0; i < nRow; i++)  {
        for (int j = 0; j < nColl; j++) {
            gameboard[i][j] = '-';
        }
    }
      for (int x, y, i = 0; i < bombCount; i++) {
        do {
            y = rand() % nRow;
            x = rand() % nColl;
        } while (gameboard[y][x] == '*');
        gameboard[y][x] = '*';
    }
    isLose = false;
}

Game::~Game() {
    for (int i = 0; i < nRow; i++) {
        delete gameboard[i];
    }
    delete[] gameboard;
}

void Game::printGameBoard() {
    printf("  ");
    for (int j = 0; j < nColl; j++) {
        printf("%d ", (j + 1)%10);
    }
    printf("\n");
    for (int i = 0; i < nRow; i++)  {
        printf("%d|", (i + 1)%10);
        for (int j = 0; j < nColl; j++) {
            if (gameboard[i][j] != '*') {
                                printf("%c|", gameboard[i][j]);
            } else {
                                 printf("-|", gameboard[i][j]);
            }
        }
        printf("\n");
    }
}

```

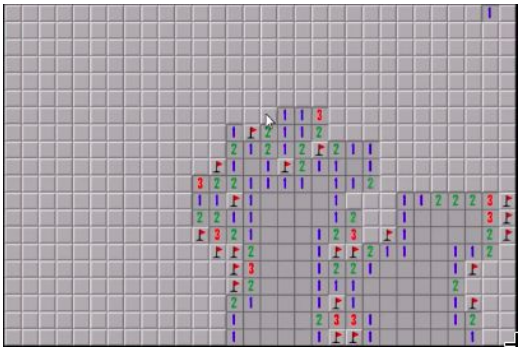
```
bool Game::Lose() {
    return isLose;
}

bool Game::Win() {
    bool isWin;
    isWin = true;
    for (int i = 0; i < nRow; i++)  {
        for (int j = 0; j < nColl; j++) {
            if (gameboard[i][j] == '-') {
                isWin = false;
                break;
            }
        }
    }
    return isWin;
}

int Game::adjBomb(int _i, int _j) {
    int count = 0;
     for (int i = (_i - 1); i < (_i - 1) + 3; i++) {
          for (int j = (_j - 1); j < (_j - 1) + 3; j++) {
              if ((i >= 0 && i < nRow && j >=0 && j < nColl) && gameboard[i][j] == '*') {
                    count++;
                }
            }
        }
        return count;
    }

    void Game::reccOpen(int _i, int _j) {
        if (adjBomb(_i, _j) != 0) {
                  gameboard[_i][_j] = '0' + adjBomb(_i, _j);
        } else {
            gameboard[_i][_j] = ' ';
             for (int i = (_i - 1); i < (_i - 1) + 3; i++) {
                 for (int j = (_j - 1); j < (_j - 1) + 3; j++) {
                     if ((i >= 0 && i < nRow && j >=0 && j < nColl) && (gameboard[i][j] != ' ')) {
                            //printf("hmmm %d %d", i, j);
                        reccOpen(i, j);
                    }
                }
            }
        }
    }

    void Game::input(int x, int y) {
        if (((x - 1) < nColl && (x - 1) >= 0) && (y - 1 < nRow) && (y - 1) >= 0) {
            if (gameboard[y-1][x-1] != '*') {
                  gameboard[y-1][x-1] = '0' + adjBomb(y-1, x-1);
                  if (gameboard[y-1][x-1] == '0') {
                      gameboard[y-1][x-1] = ' ';
                    reccOpen(y - 1, x - 1);
                }
            } else {
                isLose = true;
            }
        }
    }

    int main() {
        Game* G;
        int choice, inx, iny;
                                    cout       << "==============MINESWEEPER==============\n";
        ch:
                 cout << "Please choose your difficulty :\n";
            cout << "1. Easy\n";
            cout << "2. Medium\n";
            cout << "3. Hard\n";
            cin >> choice;
            if (choice == 1) {
                G = new Game(10, 10, 10);
            } else if (choice == 2) {
                G = new Game(18, 18, 20);
            } else if (choice == 3) {
                G = new Game(24, 24, 40);
            } else {
                cout << "Invalid input\n";
                goto ch;
            }

        while(!(G->Lose() || G->Win())) {
            G->printGameBoard();
            cin >> inx >> iny;
            G->input(inx, iny);
        }
        if (G->Lose()) {
            cout << "YOU LOSE!\n";
        } else if (G->Win()) {
            cout << "YOU WIN!\n";
        } else {
            cout << "Something wrong";
        }
        delete G;
    }
```

I only included the important function in the code snippet and exclude many unimportant parts so not to make the snippet too long.

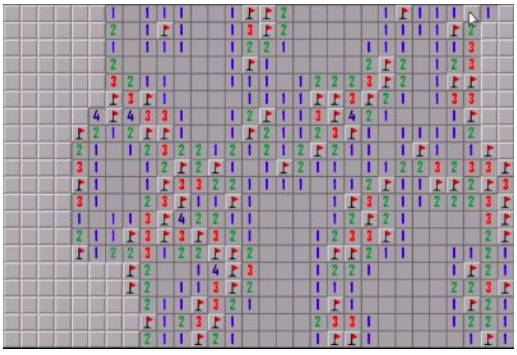
Test Case

I have also make some test case to test my algorithm,

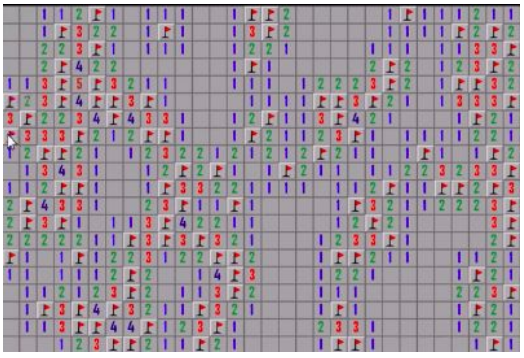
Test Case 1 Take 1



Test Case 1 Take 2



Test Case 1 Take 3



IV. Conclusion

In this paper, we have discussed algorithm in general, discussing it definition, classification all the way to it history, we continue our discussion to more specific type of algorithm that used to find the most optimal solution from a problem which is greedy algorithm. Then we use this algorithm to solve my long time favorite game of minesweeper.

We do this by finding the probability of a tile being a bomb then choose the safest tile given this parameter.

V. Acknowledgment

I would like to thank God for giving me power, will, and opportunity to finish this paper in time and I would like to thank my parent for giving me support and the chance to study here in computer science department of STEI.

References

[1] Bell, C. Gordon and Newell, Allen (1971), Computer Structures: Readings and Examples, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.

[2] Burgin, Mark (2004). Super-Recursive Algorithms. Springer. ISBN 978-0-387-95569-8.

[3] Minsky, Marvin (1967). Computation: Finite and Infinite Machines (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-165449-7.

[4] Bolter, David J. (1984). Turing's Man: Western Culture in the Computer Age (1984 ed.). The University of North Carolina Press, Chapel Hill NC. ISBN 0-8078-1564-0., ISBN 0-8078-4108-0pbk.

[5] Knuth, Donald (1997). Fundamental Algorithms, Third Edition. Reading, Massachusetts: Addison–Wesley. ISBN 0-201-89683-4.

[6] Church, Alonzo (1936a). "An Unsolvable Problem of Elementary Number Theory". The American Journal of Mathematics.

[7] Munir Rinaldi. DIktat Kuliah Strategi Algoritma, Program Studi Teknik Informatika ITB 2009.

[8] A. A. Markov (1954) Theory of algorithms. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954

[9] Kosovsky, N. K. Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms, LSU Publ., Leningrad, 1981

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

Hafidh Rendyanto 13517061