

**A REPORT ON**  
**Directory Tree Generator**

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE  
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENT  
FOR

**FUNDAMENTALS OF DATA STRUCTURES**

**SUBMITTED BY**

**RAKESH KUMAR TIWARI (3239)**  
**DIGVIJAY SINGH (3214)**



**DEPARTMENT OF COMPUTER ENGINEERING**

**ARMY INSTITUTE OF TECHNOLOGY**

**DIGHI HILLS, ALANDI ROAD, PUNE 411015**

**SAVITRIBAI PHULE PUNE UNIVERSITY**  
**2020-2021**



This is to certify that the project report entitles

SUBMITTED BY

**RAKESH KUMAR TIWARI (3239)**  
**DIGVIJAY SINGH (3214)**

are bonafide students of this institute and the work have been carried out by them under the supervision of **Prof. Vaishali Ganganwar** and it has been approved for the partial fulfilment of the requirement of mini-project for the subject Fundamentals of Data Structures.

Place: Pune

Date:

03/12/2020

## **ABSTRACT**

- This project is implementation of game Minesweeper.
- We play on a square board and we have to click on the board on the cells which do not have a mine. And obviously we don't know where mines are. If a cell where a mine is present is clicked then we lose, else we are still in the game.
- **There are three levels for this game-**
  - Beginner –  $9 * 9$  Board and 10 Mines
  - Intermediate –  $16 * 16$  Board and 40 Mines
  - Advanced –  $24 * 24$  Board and 99 Mines
- **Probability of finding a mine –**
  - 
  - Beginner level –  $10/81$  (0.12)
  - Intermediate level –  $40/256$  (0.15)
  - Advanced level –  $99 / 576$  (0.17)

### **❖ How to Play**

- When we click on a cell having adjacent mines in one or more of the surrounding eight cells, then we get to know how many adjacent cells have mines in them. So we can do some logical guesses to figure out which cells have mines.
- If you click on a cell having no adjacent mines (in any of the surrounding eight cells) then all the adjacent cells are automatically cleared, thus saving our time.
- So we can see that we don't always have to click on all the cells not having the mines (total number of cells – number of mines) to win. If we are lucky then we can win in very short time by clicking on the cells which don't have any adjacent cells having mines.

## **PROJECT IMPLEMENTATION**

- Also there are two boards- realBoard and myBoard. We play our game in myBoard and realBoard stores the location of the mines. Throughout the game, realBoard remains unchanged whereas myBoard sees many changes according to the user's move.
- We can choose any level among – BEGINNER, INTERMEDIATE and ADVANCED.
- Once the level is chosen, the realBoard and myBoard are initialized accordingly and we place the mines in the realBoard randomly.
- We can cheat before playing (by knowing the positions of the mines) using the function – cheatMinesweeper(). In the code this function is commented . So if you are afraid of losing then uncomment this function and then play !
- Then the game is played till the user either wins (when the user never steps/clicks on a mine-containing cell) or lose (when the user steps/clicks on a mine-containing cell). This is represented in a while() loop. The while() loop terminates when the user either wins or lose.
- The makeMove() function prompts the user to enter his own move.

- Also to guarantee that the first move of the user is always safe (because the user can lose in the first step itself by stepping/clicking on a cell having a mine, and this would be very much unfair), we put a check by using the if statement – if (currentMoveIndex == 0)
- The lifeline of this program is the recursive function – playMinesUtil()
- This function returns a true if the user steps/clicks on a mine and hence he loses else if he step/click on a safe cell, then we get the count of mines surrounding that cell. We use the function countAdjacentMines() to calculate the adjacent mines. Since there can be maximum 8 surrounding cells, so we check for all 8 surrounding cells.
- If there are no adjacent mines to this cell, then we recursively click/step on all the safe adjacent cells (hence reducing the time of the game-play). And if there is atleast a single adjacent mine to this cell then that count is displayed on the current cell. This is given as a hint to the player so that he can avoid stepping/clicking on the cells having mines by logic.
- Also if you click on a cell having no adjacent mines (in any of the surrounding eight cells) then all the adjacent cells are automatically cleared, thus saving our time.
- So we can see that we don't always have to click on all the cells not having the mines (total number of cells – number of mines) to win. If we are lucky then we can win in very short time by clicking on the cells which don't have any adjacent cells having mines.
- The user keeps on playing until he steps/clicks on a cell having a mine (in this case the user loses) or if he had

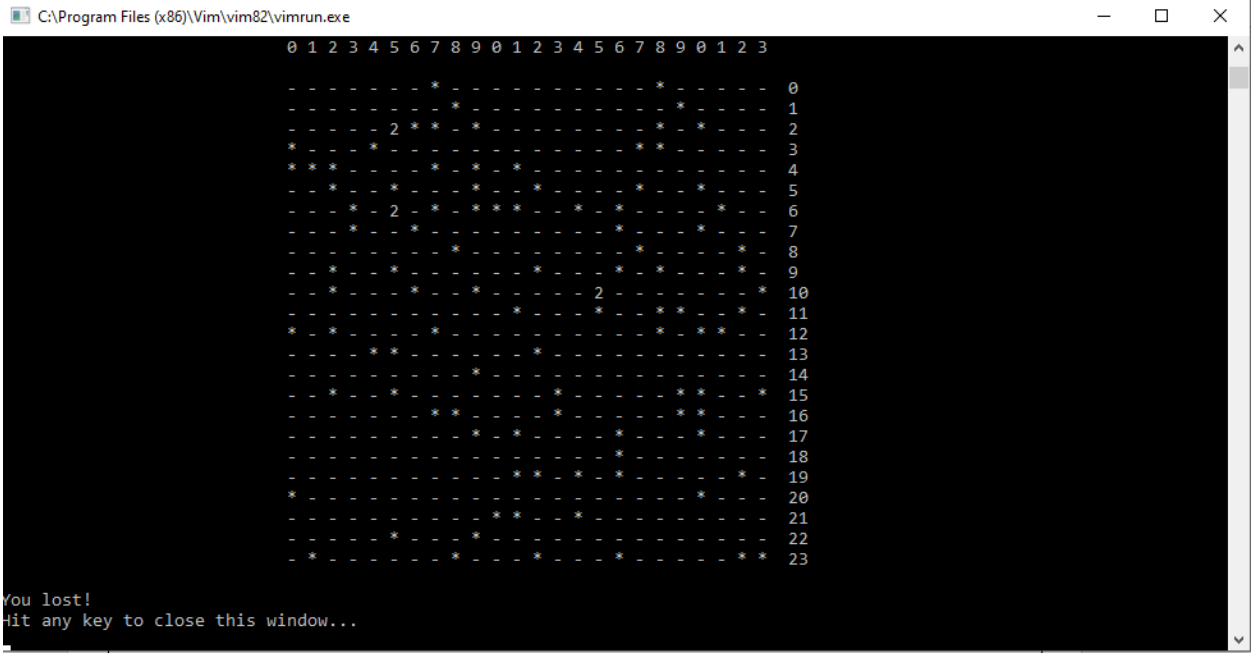
clicked/stepped on all the safe cell (in this case the user wins).

## **TOOLS AND TECHNOLOGIES USED**

- Language used:
  - C++
- Libraries used:
  -
- Algorithms Used:
  - We used Recursions in playminesuntil() to skip non mines cells related to the input cell to save time.

## RESULT

- Implemented the game : Minesweeper.
- MineSweeper Generated in output window:

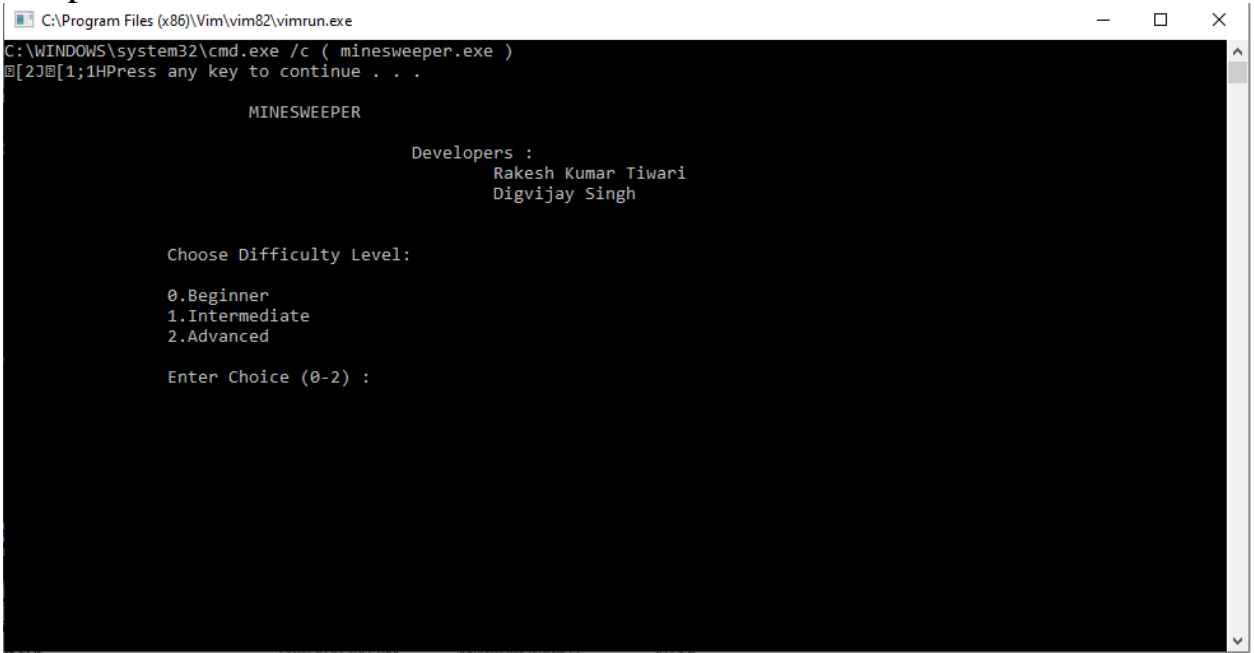


```
C:\Program Files (x86)\Vim\vim82\vimrun.exe

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
- - - - - * - - - - - * - - - - - 0
- - - - - * - - - - - * - - - - - 1
- - - - - 2 * * * - - - - - * * - - - 2
* - - - * - - - - - - - - - * * - - - 3
* * * - - * - * * * - - - - - * - - - 4
- * - * - - - * * * - - * - - * - - - 5
- - - * - 2 - * * * * - * - - - * - - 6
- * - - * - - - - - - * - - - * - - - 7
- - - * - - - * - - - - * - - - * - - 8
- - * - * - - - - * - - - * - - - * - 9
- - * - - * - - * - - - 2 - - - - * 10
* - - - - - - * - - - * - * * - * - 11
* - * - - * - - - - - - - * * * - - 12
- - - * * - - - - * - - - - - - - - 13
- - - - - - - * - - - - - - - - - - 14
- - * - * - - - - * - - - - * * - * 15
- - - - - * * - - - * - - - * * - - 16
- - - - - - - * * - - - * - - - * - - 17
- - - - - - - - - - - * - - - - - - 18
- - - - - - - - - - * * - * - - - * - 19
* - - - - - - - - - * - - - - * - - 20
- - - - - - - * * - - * - - - - - - 21
- - - - - * - - - * - - - - - - - - 22
- * - - - - * - - - * - - - - - * * 23

You lost!
Hit any key to close this window...
```

- Graphical User Interface:



```
C:\Program Files (x86)\Vim\vim82\vimrun.exe
C:\WINDOWS\system32\cmd.exe /c ( minesweeper.exe )
[2J[1;1HPress any key to continue . . .

MINESWEEPER

Developers :
    Rakesh Kumar Tiwari
    Digvijay Singh

Choose Difficulty Level:

0.Beginner
1.Intermediate
2.Advanced

Enter Choice (0-2) :
```

## CODE

```
/* Minesweeper
 * By Rakesh kumar Tiwari(3239) & Digvijay Singh (3214)
 * FDS Mini Project
 */
#include <bits/stdc++.h>
using namespace std;

#define max_mine 99
#define max_side 25
#define max_move 526

int SIDE;
int MINES;

void clear(){
    cout << "\33[2J\033[1;1H";
    system("pause");
}

bool isValid(int row,int col){ // If the cell is inside the board
    return (row>=0)&&(row<SIDE)&&(col>=0)&&(col<SIDE);
}

bool ismine(int row,int col,char board[][max_side]){ // If there is a mine
    if(board[row][col]=='*')
        return (true);
    else
        return (false);
}

void make_move(int *x,int *y){
    while(true) // Correct input
    {
        cout<<"\nEnter your move: [row] [column] -> ";
        cin>>*x>>*y;
        // Check values to make sure they're not larger than the board size.
        if ((*x < SIDE) && (*y < SIDE)){
            return;
        }
        else cout << "\t (Wrong Input)";
    }
}

void printboard(char myboard[][max_side]){
    clear();
    cout<<"\n\n\t\t\t\t\t ";
    for(int i=0;i<SIDE;i++)
```



```

{
    if (i>9)
        cout<<i/10<<" ";
    else
        cout<<" ";
}

cout<<"\n\t\t\t\t ";

for(int i=0;i<SIDE;i++)
    cout<<i%10<<" ";

cout<<"\n\n";

for(int i=0;i<SIDE;i++)
{
    cout<<"\t\t\t\t ";
    for(int j=0;j<SIDE;j++){
        cout<<myboard[i][j]<<" ";
    }
    cout<<" "<<i;
    cout<<"\n";
}
return;
}

int countadjacent(int row,int col,int mines[][2],char realboard[][max_side]) // Counts the
no of adjacent cell that does no contain a mine
{
    int count=0;
    if(isvalid(row-1,col)==true)
    {
        if(ismine(row-1,col,realboard)==true)
            count++;
    }
    if(isvalid(row+1,col)==true)
    {
        if(ismine(row+1,col,realboard)==true)
            count++;
    }
    if(isvalid(row,col+1)==true)
    {
        if(ismine(row,col+1,realboard)==true)
            count++;
    }
    if(isvalid(row,col-1)==true)
    {
        if(ismine(row,col-1,realboard)==true)
            count++;
    }
}

```

```

    if(isvalid(row-1,col-1)==true)
    {
        if(ismine(row-1,col-1,realboard)==true)
            count++;
    }
    if(isvalid(row-1,col+1)==true)
    {
        if(ismine(row-1,col+1,realboard)==true)
            count++;
    }
    if(isvalid(row+1,col-1)==true)
    {
        if(ismine(row+1,col-1,realboard)==true)
            count++;
    }
    if(isvalid(row+1,col+1)==true)
    {
        if(ismine(row+1,col+1,realboard)==true)
            count++;
    }
    return (count);
}

bool playminesuntil(char myboard[][max_side],char realboard[][max_side],int mines[][2],int
row,int col,int *moves_left)
{
    if(myboard[row][col]!='-') //
        return false;

    int i,j;
    if(realboard[row][col]=='*')
    {
        myboard[row][col]='*';
        for(i=0;i<MINES;i++)
            myboard[mines[i][0]][mines[i][1]]='*';

        printboard(myboard);
        cout<<"\nYou lost!\n";
        return (true);
    }
    else
    {
        int count=countadjacent(row,col,mines,realboard);
        (*moves_left)--;
        myboard[row][col]= count+'0';
        if(!count)
        {
            if(isvalid(row-1,col)==true)
            {
                if(ismine(row-1,col,realboard)==false)

```

```

        playminesuntil(myboard, realboard, mines, row-1, col, moves_left);
    }
    if (isvalid (row+1, col) == true)
    {
        if (ismine (row+1, col, realboard) == false)
            playminesuntil(myboard, realboard, mines, row+1, col, moves_left);
    }
    if (isvalid (row, col+1) == true)
    {
        if (ismine (row, col+1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row, col+1, moves_left);
    }
    if (isvalid (row, col-1) == true)
    {
        if (ismine (row, col-1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row, col-1, moves_left);
    }
    if (isvalid (row-1, col+1) == true)
    {
        if (ismine (row-1, col+1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row-1, col+1, moves_left);
    }
    if (isvalid (row-1, col-1) == true)
    {
        if (ismine (row-1, col-1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row-1, col-1, moves_left);
    }
    if (isvalid (row+1, col+1) == true)
    {
        if (ismine (row+1, col+1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row+1, col+1, moves_left);
    }
    if (isvalid (row+1, col-1) == true)
    {
        if (ismine (row+1, col-1, realboard) == false)
            playminesuntil(myboard, realboard, mines, row+1, col-1, moves_left);
    }
}
return (false);
}
}

void placemines(int mines[][2],char realboard[][max_side])
{
    bool mark[max_side*max_side];

    memset(mark,false,sizeof(mark));

    for(int i=0;i<MINES;)
    {

```

```

    int random=rand()%(SIDE*SIDE);
    int x=random/SIDE;
    int y=random%SIDE;

    if(mark[random]==false) //add mine if not present at position random
    {
        mines[i][0]=x;
        mines[i][1]=y;

        realboard[mines[i][0]][mines[i][1]]='*';
        mark[random]=true;
        i++;
    }
}

void initialise(char realboard[][max_side],char myboard[][max_side])
{
    srand(time(NULL)); //initilising random so that same config doesn't arise

    int i,j;
    for(i=0;i<SIDE;i++)
        for(j=0;j<SIDE;j++)
        {
            myboard[i][j]=realboard[i][j]='-';
        }
    return;
}

void replacemine(int row,int col,char board[][max_side])
{
    for(int i=0;i<SIDE;i++)
    {
        for(int j=0;j<SIDE;j++)
        {
            if(board[i][j]!='*')
            {
                board[i][j]='*';
                board[row][col]='-';
                return;
            }
        }
    }
}

void play()

```

```

{
    bool gameover=false;

    char realboard[max_side][max_side],myboard[max_side][max_side];

    int moves_left=SIDE*SIDE-MINES;
    int x,y;
    int mines[max_mine][2]; //stores (x,y) of all mines

    initialise(realboard,myboard);

    placemines(mines,realboard);

    int currentmoveindex=0;

    while(gameover==false)
    {
        cout<<"Current Status of Board : \n";
        printboard(myboard);
        make_move(&x,&y);

        //if first move is mine
        if(currentmoveindex==0)
        {
            if(ismine(x,y,realboard)==true) //first attempt is a mine
                replacemine(x,y,realboard); //replace the mine at that location
        }

        currentmoveindex++;

        gameover = playminesuntil(myboard,realboard,mines,x,y,&moves_left);

        if((gameover==false)&&(moves_left==0))
        {
            cout<<"\nYou won !\n";
            gameover=true;
        }
    }
}

int Menu(){
    clear();
    cout<<"\n\t\t\t\t\tMINESWEEPER";
    cout<<"\n\n\t\t\t\t\tDevelopers : \n\t\t\t\t\t\t\tRakesh Kumar
Tiwari\n\t\t\t\t\t\t\tDigvijay Singh\n";
    cout<<"\n\n\t\t\t\t\tChoose Difficulty Level: ";
    cout<<"\n\n\t\t\t\t\t0.Beginner\n\t\t\t\t\t1.Intermediate\n\t\t\t\t\t2.Advanced";
    cout<<"\n\n\t\t\t\t\tEnter Choice (0-2) : ";
    int choice;

```

```

    cin >> choice;
    return choice;
}
void choosedifficulty()
{
    int choice = Menu();
    if(choice==0)
    {
        SIDE=9;
        MINES=10;
    }
    else if(choice==1)
    {
        SIDE=16;
        MINES=40;
    }
    else if(choice==2)
    {
        SIDE=24;
        MINES=99;
    }
    else
        exit(0);
}

int main()
{
    choosedifficulty();
    play();
    return 0;
}

```