SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING
MYSORE-570006
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MINESWEEPER and LENGTH CONQUER

Submitted by

Name	Roll No.	Usn
Jayavant M Nijagal	61	4JC09CS126
Nandini Bhavasar N. A	29	4JC09CS060
Sachin Pandit	45	4JC09CS090
Vinay.V.Vasista	58	4JC09CS120

Guidance of
P.M.SHIVAMURTHY
Lecturer
Dept of CS&E,SJCE Mysore.

Affiliated to
VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM

Contents

1 Introduction

1.1 Minesweeper

Minesweeper is a single-player game. The object of the game is to clear an abstract minefield without detonating a mine. The game has been written for many system platforms in use today, including the Minesweeper for the Windows platform, which has come bundled with versions of the operating system from 3.1 and on.
Minesweeper cannot always be solved with 100 percent certainty, and may require the occasional use of probability to flag the square most likely to have a mine. In other words, one must sometimes guess to solve a minesweeper puzzle.
The player is initially presented with a grid of undistinguished squares. Some randomly selected squares, unknown to the player, are designated to contain mines. Typically, the size of the grid and the number of mines are set in advance by the user, either by entering the numbers or selecting from defined skill levels depending on the implementation.
The game is played by revealing squares of the grid, typically by clicking them with a mouse. If a square containing a mine is revealed, the player loses the game. Otherwise, a digit is revealed in the square, indicating the number of adjacent squares (typically, out of the possible eight) that contain mines. In typical implementations, if this number is zero then the square appears blank, and the surrounding squares are automatically also revealed. By using logic, the player can in many instances use this information to deduce that certain other squares are mine-free, in which case they may be safely revealed, or mine-filled, in which they can be marked as such (which, in typical implementations, is effected by right-clicking the square and indicated by a flag graphic).
The game is won when all mine-free squares are revealed, meaning that all mines have been located.
There are many patterns of numbered squares that may arise during a game that can be recognized as allowing only one possible configuration of mines in their vicinity. In the interest of finishing quickly, it is often easiest to process the known patterns first, and continue on with the uncertain parts later. There are a few broad methods for solving problems in minesweeper games without guessing.
There are two special cases that are of extra interest when solving a board that can be solved using analysis of only one square and its surrounding squares
∗ If the number of unrevealed (blank or flagged) squares adjacent to a numbered square is equal to the number on that square, all these unrevealed squares must be mines. ∗ For any numbered square, if the number of flagged mines located adjacent to that square is equal to the number of the square, all other squares adjacent to that numbered square must be 'safe'

The objective of this project is to implement this game and to make it more interesting as well as more challenging to the player.

1.2 Length Conquer

Length Conquer is a game played by 2 players. The goal of the game is to cover a total length in a scale. To achieve the goal three small sub-units are used. Each player is given a chance alternatively to choose a sub-unit. If the whole scale is covered exactly then the game is a draw. Else the player who last bid will be the winner

The objective of this project is to implement this game with one of the player being the computer. The aim is to make the computer unbeatable. This is achieved by designing an algorithm which tells the computer as to which is the safest next move based on all the probable moves of the other player. Care is taken such that the computer always stays in the safer zone. The game will either end up in a draw or the computer will win.

2 Design of algorithm

2.1 Minesweeper

Algorithm 1: Takes the difficulty level from the user

```
input  :  The choice of user ( easy ==1, medium = 2, hard = 3)
output: Number of mines
begin
    if mines == 1
    return 13
    else mines == 2
    return 28
    else mines == 3
    return 40
    else return 0
end
```

Algorithm 2: Procedure to set the values to each cell depending upon the surrounding mines

```
input  : The position of current cell
output: Number of mines surrounding it
begin
    count ← 0
    for i ← m−1 to m+1 do
    for j ← n−1 to n+1 do
    if (i ! = m OR j ! = n) AND (board[i][j] = '*')
    count ← count + 1 num[m][n] ← count+'0'
end
```

Algorithm 3: Depending on the difficulty level chosen by the user number of mines are decided

```
   input  : position of row and column
   output: Assigns the value to each cell
1  begin
2      countmines ← 0
3      nomines ← numofmines(mines)
4      while nomines ! = 0 do
5      i ← random number, 0−9
6      j ← random number, 0−9
7      if( i > 0 AND i < 9) AND (j > 0 AND j < 9 ) AND board[i+1][j+1]
       ! = '∗'
8      board[i+1][j+1] ¡- '∗'
9      countmines ← countmines + 1
10     nomines ← nomines − 1
11     countmines ← countmines − 1
12     return countmines
13 end
```

Algorithm 4: Dynamic Board

```
   input  :
   output:
1  begin
2      for i ← x−1 to x+1 do
3      for j ← x−1 to x+1 do
4      if num[i][j] = '0' AND board[i][j] ! = '∗')
5      if i > 0 AND i < 10 AND j > 0 AND j < 10 AND newnum[i][j] = '−'
6      num[i][j] ← ash
7      newnum[i][j] ← ash
8      dynamic(board, i, j)
9      if board[i][j] ! = '∗' AND newnum[i][j] = '−')
10     newnum[i][j] ← num[i][j]
11     return
12 end
```

Algorithm 5: countstar(board, newnum,*countu, *countf)

```
input  :
output:
begin
    $*$countu $\leftarrow$ 0
    $*$countf $\leftarrow$ 0
    for i $\leftarrow$ 1 to 9 do
    for j $\leftarrow$ 1 to 9 do
    if newnum[i][j] = ash OR newnum[i][j] >= '1' AND newnum[i][j] <= '9'
    $*$countu $\leftarrow$ $*$countu + 1
    if newnum[i][j] = 'F' AND board[i][j] = '$*$'
    $*$countf $\leftarrow$ $*$countf + 1
    return
end
```

2.2 Length Conquer

Algorithm 6: Sorts the given three sub-units in non-decreasing order

```
input  : 3 pointers pointing to the sub-units
output: Given 3 sub-units in non-decreasing order
begin
    if(*x > *y)
    SWAP(x, y);
    if(*y > *z)
    SWAP(y, z);
    if(*x > *y)
    SWAP(x, y);
end
```

Algorithm 7: Checks if the given sub-units are valid

```
   input  :  Uses the global variables L, x, y and z
   output:  Validity flag
1  begin
2      valid ← -1;
3      if (L < 4∗z) OR (x <= 1) OR (x = y OR x = z OR y = z) valid ← 0
4      return (valid)
5  end
```

Algorithm 8: Determines the no of sub-units that can be used

```
   input  :  The current index of the array
   output: Number of options the player has in chosing the next step
1  begin
2      limit ← bool(i+x <= L) + bool(i+y <= L) + bool(i+z <= L)
3      return (limit)
4  end
```

Algorithm 9: Determines the zone to which the zone-value maps to

```
   input  :  Zone-value, usually the content of a zone array element
   output:  Zone to which the zone-value comes under
1  begin
2      if zone-value ¡ 0 return (UNSTEPPED) if zone-value ¡ 10 return
       (SAFE) if zone-value ¡ 100 return (DRAW)
3      return (UNSAFE)
4  end
```

Algorithm 10: Calculates the priority-value of locations on the Length according to

```
   input  : Three zones to which the locations, led to by the three
            sub-units, belongs tozone1 - via x, zone2 - via y, zone3 - via z
   output: priority-value of the current location
1  begin
2      return ( bool(zone1 ! = NIL) ∗ pow(10,zone1) + bool(zone2 ! = NIL)
       ∗ pow(10,zone2) + bool(zone3 ! = NIL) ∗ pow(10,zone3) )
3  end
```

Algorithm 11: Based on its priority, this procedure divides locations into suitable zones

```
   input  : void
   output: zone-array[1..L] with it's locations categorized under suitable
           zones along with priority
 1 begin
 2     zone[L] ← Priority(DRAW, NIL, NIL)
 3     for i ← L−1 downto L−x+1 do zone[i] ← Priority(UNSAFE, NIL,
       NIL) done
 4     zone[L−x] ← Priority(DRAW, NIL, NIL) i ← i−1
 5     for i downto L−y+1 do zone[i] ← Priority( Zone(zone[i+x]), NIL, NIL
       ) done
 6     for i downto L−z+1 do zone[i] ← Priority( Zone(zone[i+x]),
       Zone(zone[i+y]), NIL ) done
 7     for i downto x do zone[i] ← Priority( Zone(zone[i+x]),
       Zone(zone[i+y]), Zone(zone[i+z]) ) done
 8     for i downto 1 do zone[i] ← UNSTEPPED; done
 9     for i ← x+1 until (i < y AND i < z AND i < 2∗x) do zone[i] ←
       UNSTEPPED; done
10 end
```

Algorithm 12: Procedure to get the optimal move from the computer

```
   input  : The number of allowed moves among x, y, z - 'limit' The current
           position in the array board - 'pos'
   output: The 'step' chosen by the computer as its next move
 1 begin
 2     if limit == 0 return (invalid) blindstep ← x
 3     if lim == 3 blindstep ← z if lim >= 2 if zone[pos+y] <
       zone[pos+blindstep] blindstep ← y if lim > 1 if zone[pos+x] <
       zone[pos+blindstep] blindstep ← x
 4     step ← blindstep
 5     return (step)
 6 end
```

Algorithm 13: Manager of the Game after the fixture of the Length and sub-units

```
input  : void
output: Result of the game, i.e the winner
begin
    boardpos ← 0,
    ask user to select his turn
    limit ← OptionLimit(boardpos)
    display the empty board
    while limit ! = 0 do if human is first player ask human's move else
    GetPosition(boardpos)
    flag ← 1
    update the board display the board
    limit ← OptionLimit(boardpos) if limit ! = 0 if human is second
    player ask human's move else GetPosition(boardpos)
    flag ← 2
    update the board display the board
    limit ← OptionLimit(boardpos) done
    if boardpos = Length result ← draw else if human played as flag
    player result ← human else result ← bot
    return (result)
end
```

3 Implementation

3.1 Minesweeper

The procedure where in the mines are marked and the adjacent cells are initialised are shown below

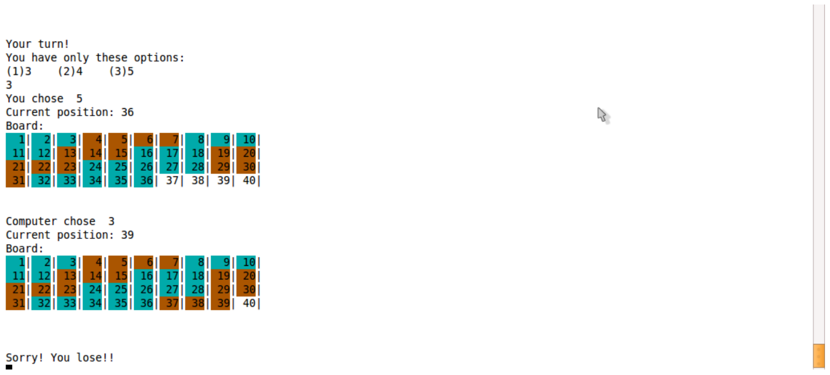
```
// DYNAMIC UNCOVERING OF CELLS IN A BOARD
void dynamic(char board[11][11], int x, int y){
        for(int i = x−1 ; i <= x+1 ; i++ ){
                for( int j = y−1 ; j <= y+1 ; j++){
                                if( num[i][j] == '0' && board[i][j] != '*'){
                                        if( i > 0 && i < 10 && j > 0 && j < 10 && new_num[i][j] =
                                                num[i][j] = '#';
                                                new_num[i][j] = '#';
                                                dynamic(board, i, j);
                                        }
                                }
                                if( board[i][j] != '*' && new_num[i][j] == '-')
                                        new_num[i][j] = num[i][j];
                }
        }
        new_num[x][y] = '#';
        return;
}

// ASSIGN NO: OF BOMBS IN ITS SURROUNDINGS
void assign_num(char board[11][11], int m ,int n){
        int count = 0;
        char var;
        for(int i = m−1 ; i <= m+1 ; i++ ){
                for(int j = n−1 ; j <= n+1 ; j++ ){
                        if(( i != m || j != n) && board[i][j] == '*'){
                                count++;
                        }
                }
        }
        num[m][n] = count+'0';
}
```

3.2 Length Conquer

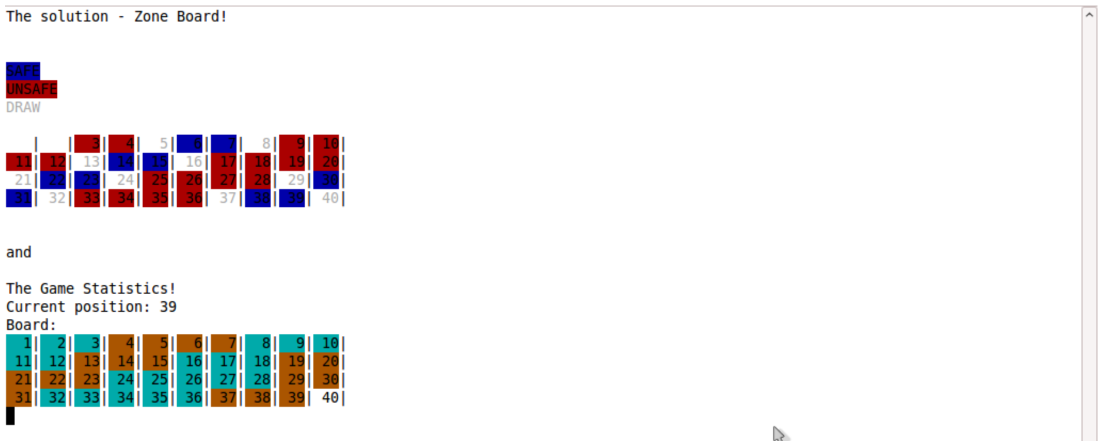
Here is the code of the procedure where the zones are marked.

```
void mark_zones()
{
  int
    i;      //index

  //printf("\nin mark_zones:\narg1 = %d\targ2 = %d\targ3 = %d\targ4 = %d\n", x, y, z, L);

  zone[L−1] = priority(DRAW, UNSAFE−1, UNSAFE−1);

  for(i = L−1; i > L−x; i−−)
    zone[i−1] = priority(UNSAFE, UNSAFE−1, UNSAFE−1);

  zone[L−x−1] = priority(DRAW, UNSAFE−1, UNSAFE−1);
  i−−;

  for( ; i > L−y; i−−)
    zone[i−1] = priority(ZONE(zone[i−1+x]), UNSAFE−1, UNSAFE−1);

  for( ; i > L−z; i−−)
    zone[i−1] = priority(ZONE(zone[i−1+x]), ZONE(zone[i−1+y]), UNSAFE−1);

  for( ; i >= x; i−−)
    zone[i−1] = priority(ZONE(zone[i−1+x]), ZONE(zone[i−1+y]), ZONE(zone[i−1+z]));
  for( ; i > 0; i--)
    zone[i-1] = UNSTEPPED;


  for(i = x+1; i < y && i < z && i < 2*x; i++)
    zone[i-1] = UNSTEPPED;

}
```

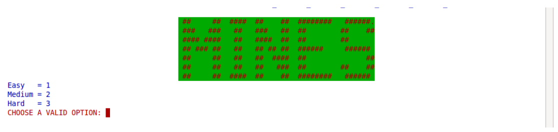
4 Snapshots



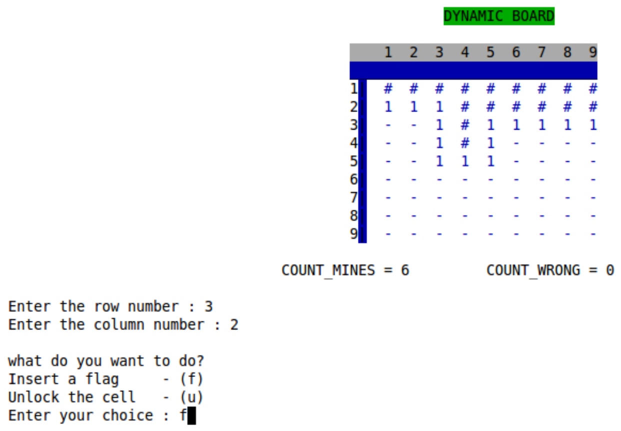
```
Your turn!
You have only these options:
(1)3    (2)4    (3)5
3
You chose  5
Current position: 36
Board:
  1|  2|  3|  4|  5|  6|  7|  8|  9| 10|
 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
 21| 22| 23| 24| 25| 26| 27| 28| 29| 30|
 31| 32| 33| 34| 35| 36| 37| 38| 39| 40|


Computer chose  3
Current position: 39
Board:
  1|  2|  3|  4|  5|  6|  7|  8|  9| 10|
 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
 21| 22| 23| 24| 25| 26| 27| 28| 29| 30|
 31| 32| 33| 34| 35| 36| 37| 38| 39| 40|



Sorry! You lose!!
```



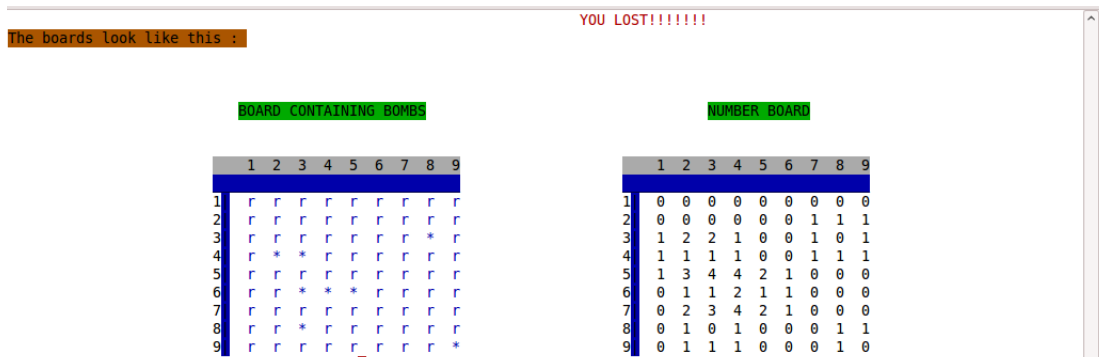
```
The solution - Zone Board!


SAFE
UNSAFE
DRAW

   |   |  3|  4|  5|  6|  7|  8|  9| 10|
 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
 21| 22| 23| 24| 25| 26| 27| 28| 29| 30|
 31| 32| 33| 34| 35| 36| 37| 38| 39| 40|


and

The Game Statistics!
Current position: 39
Board:
  1|  2|  3|  4|  5|  6|  7|  8|  9| 10|
 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
 21| 22| 23| 24| 25| 26| 27| 28| 29| 30|
 31| 32| 33| 34| 35| 36| 37| 38| 39| 40|
```



```
##     ##  ####  ##    ##  ########   ######.
###   ###   ##   ###   ##  ##        ##    ##
#### ####   ##   ####  ##  ##        ##
## ### ##   ##   ## ## ##  ######     ######
##     ##   ##   ##  ####  ##              ##
##     ##   ##   ##   ###  ##        ##    ##
##     ##  ####  ##    ##  ########   ######

Easy   = 1
Medium = 2
Hard   = 3
CHOOSE A VALID OPTION:
```



```
                                DYNAMIC BOARD

                         1  2  3  4  5  6  7  8  9

                     1   #  #  #  #  #  #  #  #  #
                     2   1  1  1  #  #  #  #  #  #
                     3   -  -  1  #  1  1  1  1  1
                     4   -  -  1  #  1  -  -  -  -
                     5   -  -  1  1  1  -  -  -  -
                     6   -  -  -  -  -  -  -  -  -
                     7   -  -  -  -  -  -  -  -  -
                     8   -  -  -  -  -  -  -  -  -
                     9   -  -  -  -  -  -  -  -  -

                COUNT_MINES = 6          COUNT_WRONG = 0

Enter the row number : 3
Enter the column number : 2

what do you want to do?
Insert a flag     - (f)
Unlock the cell   - (u)
Enter your choice : f
```



```
                                                  YOU LOST!!!!!!!
The boards look like this :


            BOARD CONTAINING BOMBS                         NUMBER BOARD


         1  2  3  4  5  6  7  8  9                 1  2  3  4  5  6  7  8  9

     1   r  r  r  r  r  r  r  r  r             1   0  0  0  0  0  0  0  0  0
     2   r  r  r  r  r  r  r  r  r             2   0  0  0  0  0  0  1  1  1
     3   r  r  r  r  r  r  r  *  r             3   1  2  2  1  0  0  1  0  1
     4   r  *  *  r  r  r  r  r  r             4   1  1  1  1  0  0  1  1  1
     5   r  r  r  r  r  r  r  r  r             5   1  3  4  4  2  1  0  0  0
     6   r  r  *  *  *  r  r  r  r             6   0  1  1  2  1  1  0  0  0
     7   r  r  r  r  r  r  r  r  r             7   0  2  3  4  2  1  0  0  0
     8   r  r  *  r  r  r  r  r  r             8   0  1  0  1  0  0  0  1  1
     9   r  r  r  r  r_ r  r  r  *             9   0  1  1  1  0  0  0  1  0
```

5 Results

Minesweeper:

input :unlock the positions in the board by keepin the count of the mines such that you unlock entire board without clicking any mines.Here, (.)locked positions,(-)unlocked positions.

```
. |. |. |. |. |. |. |. |. |
- |- |. |. |. |1 |- |. |. |
- |1 |. |. |2 |- |- |. |. |
- |- |1 |. |1 |- |- |. |. |
- |- |2 |. |1 |- |. |. |. |
- |- |- |. |. |. |. |. |. |
. |. |. |. |. |. |. |. |. |
1 |- |- |- |- |- |. |. |. |
. |. |. |. |. |. |. |. |. |
```

output :If u unlock all positions without selecting any mine, you WIN.Else you LOSE.

Length Conquer Game:

input :The total length of the zone say l and three sun-units x, y and z.

```
01|02|03|04|05|06|07|08|09|10|
11|12|13|14|15|16|17|18|19|20|
21|22|23|24|25|26|27|28|29|30|
```

Here l=30,x=3,y=4,z=5.

output :if the player does not make it to 30 in his chance without any options,he LOSES.
Else if he makes it exactly to 30. the game is a DRAW. If the opponent does not make it to 30 without any draw, player WINS.

6 Conclusion

A popular technique in Minesweeper is known as ”chording”. Some of the fastest times ever accomplished in Expert level have been said to be achieved with this technique. A strafe is performed by using the following steps: 1) Flag a cell by pressing, but not releasing, the right mouse button, 2) drag the mouse, with the right mouse button still pressed, to a number that is fully flagged, 3) with the right mouse button still pressed, press and release the left mouse button to clear the fully flagged number. This game is solved basically using brute-force techniques and it does not have any ideal algorithms,or we can say we can only achieve average efficiency.

Length conquer game is based mainly on greedy' techniques of solving and comes under the ”zero sum problem”.This algorithm we have written has a very good efficiency and consumes less time in solving it and also the output can be analysed easily.

This project provided us the exposure to the real life problems related to the above techniques.We got to know the implementation of these problems in a C as well as c++ program.