Minesweeper Game Documentation

Introduction

The first game developed by me (Jay Vanidani) is minesweeper.

Minesweeper is a very much popular and classic game that came installed in most of the computers which was made in 2000's. Nowadays the game doesn't comes installed but then also we can search on Google the minesweeper game and Google has the game online and has more greater graphics. In minesweeper game, first we have to choose the difficulty mode of the game. The difficulty of the game will decide the size of the board and the number of mines. After choosing the difficulty, the board will randomly place mines in the board and hide all the mines with #(in my program). Then user will be asked to enter the row and column of the place where you want to click as till now there is no option in my game to click on the block. Then if the block contains a blank tile or the block contains the number 0 that means there is no bombs around that block. If block contains bomb(X), the user loses the game and game will be over at that time. If a number is shown, the number indicates the number of mines touching that block. This number of mines include bombs touching at all four sides as well as the diagonal blocks too. The overall objective of this minesweeper game if to reveal all the numbers without choosing the bomb block and identifies all the block containing the bombs. Once all the numbers are revealed without choosing the bomb block then user will win that game.

Variables declared

1) **sides:** Used to define size of the sides of the grid.
2) **mines:** Used to define the number of mines.
3) **board[][]:** Used to declare the board.
4) **row:** Variable to identify the row.
5) **col:** Variable to identify the column.
6) **mode:** Variable used to declare to choose the mode of the game.
7) **minesArr[]:** Stores the number of mines around the block.
8) **playBoard[][]:** It is the board in which the user plays.
9) **cnt:** It counts the adjacent mines of the particular block in function adjMines().
10) **currboard[][]:** It is the board in which all the mines are placed and with reference to that the user will play in the playBoard.
11) **mark[][]:** Stores the status of the blocks.
12) **randMines:** Stores the value of the mine location generated by the random function.
13) **count:** It counts the adjacent mines in function recMineSweeper
14) **movesLeft:** Variable used to store the moves left to win the game.
15) **gameStatus:** Stores the status of the game.
16) **noOfMoves:** Stores the value of number of moves done.

Functions Used

1) **isValidBlock():** This function checks that the block is valid or not.
2) **isMine():** This function checks that the block contains mine or not.
3) **chooseMode():** This function lets the user to choose the difficulty mode of the game.
4) **move():** This function takes the move from user.
5) **adjMines():** This function counts the number of mines of every block.
6) **printBoard():** This function prints the board.
7) **replaceMineloc():** This function replaces the location of mine and it is used to replace the location of the mine when the starting block contains mines as user shall not lose at first move.
8) **arrangeMines():** This function arranges all the mines in the board.
9) **initializingBoard():** This function initialises the board with hiding all the mines with '#'.
10) **recMineSweeper():** This function is the recursive function of minesweeper game which is used to run all the functions till the game is over.
11) **mineSweeperMain():** This function is used to run minesweeper game in a sequence.
12) **main():** This is the main function as it's there in all the programs.

Stages of Project Build

Basic Functions

Valid Block

```
bool isValidBlock(int row,int col)
{
    if(((row>=0)&&(row<N))&&((col>=0)&&(col<N)))
     return true;
    else
     return false;
}
```

- The above function isValidBlock() checks that the coordinate which user has chosen or the neighbour coordinate exists or not in the program.

Checking mine

```
bool isMine(int row,int col,char a[][N])
{
    if(a[row][col]=='X')
     return true;
    else
     return false;
}
```

- The above function isMine() checks that the coordinate which user has chosen contains mine or not.

- This function is also helpful for counting the adjacent mines as for coordinate user has chosen there we can create many conditions using this function for the adjacent coordinates.

Stage 1:

Arranging/Placing mines

```
void arrangeMines(int minesArr[][2],char playBoard[][MAXSIDES])
{
     bool mark[MAXSIDES*MAXSIDES];
     memset(mark,false,sizeof(mark)); //marking all positions in array
mark as false
     for(int i=0;i<mines;)
     {
          int randMines=rand()%(sides*sides);
          int row=randMines/sides;
          int col=randMines%sides;

          //condition for finding if there is no mine then
          //it will place mine and increment i
          if(mark[randMines]==false)
          {
               //taking row of the mine
               minesArr[i][0]=row;
               //taking column index of the mine
               minesArr[i][1]=col;
               playBoard[minesArr[i][0]][minesArr[i][1]]='X';//placing
the mine
               mark[randMines]=true;  //marking the place mine as true
               i++;
          }
     }
     return;
}
```

- The code given above is the arrangeMines() function given in which there is a 2D-Array whose size depends upon the mode of game i.e., the difficulty level of the game.
- First the above function will mark all the places of the array as false for the mines placed.
- Then after that the loop will run and inside that the random function(rand()) will generate any number and according to that row and column number will be generated
- After getting row and column first we will check whether the mine is already placed or not and if the mine is not placed then on that place mines will be placed.
- The number of mines will be decided when the mode of the game will be chosen.

```
void printBoard(char currboard[][MAXSIDES])
{
     cout<<"\t";
     for(int i=0;i<sides;i++)
```

```
    {
        cout<<i<<"  ";
    }
    cout<<endl;
    for(int i=0;i<sides;i++)
    {
        cout<<i<<"\t";
        for(int j=0;j<sides;j++)
        {
            cout<<currboard[i][j]<<"  ";
        }
        cout<<endl;
    }
    return;
}
```

- The above printBoard() function will print the board i.e., the status of all the points.

Counting mines

(i) Counting mines in a row

```
  void countNoOfMines(char a[N][N])
  {
    for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            a[i][j]='#';
        }
    }
    int noOfMines;
    int row,col;
    int choice;
    int x;
    int sPoint,sideSize;
    int count=0;
    cout<<"Enter no of mines to be placed:";
    cin>>noOfMines;
    for(int i=0;i<noOfMines;i++)
    {
        cin>>row>>col;
        a[row][col]='X';
    }
    printBoard(a);
    cin>>choice>>x>>sPoint>>sideSize;
    if(choice==1)
    {
```

```
          for(int i=sPoint;i<(sPoint+sideSize);i++)
          {
               if(a[x][i]=='X')
               {
                    count++;
               }
          }
          cout<<"There are "<<count<<" mines in row "<<x<<" from column
  "<<sPoint<<" to "<<(sPoint+sideSize-1)<<endl;
     }
     else if(choice==2)
     {
          cout<<"The number of mines in the square are:";
     }
     else
     {
          cout<<"Wrong choice!!!!!";
     }
  }
```

- In above countNoOfMines() function, first it initialises the 2D-array and take no. of mines and placed it.
- Then after placing the mines and printing the array it takes the input of

    1) The case i.e., for case 1 it find the mines in a row and for case 2 it will find the case in the square.

    2) Row number

    3) Starting Point

    4) Length of the side/ side of the square
- Then after taking all the inputs it runs the loop for that and counts the no. of mines in that particular area and prints it.

(ii) Count adjacent number of mines

```
int adjMines(int row,int col,int minesArr[][2],char playBoard[][MAXSIDES])
{
     int cnt=0;
     //for upper left block
     if(isValidBlock(row-1,col-1)==true)
          if(isMine(row-1,col-1,playBoard)==true)
               cnt++;

     //for upper block
     if(isValidBlock(row-1,col)==true)
          if(isMine(row-1,col,playBoard)==true)
```

```
                cnt++;

        //for upper right block
        if(isValidBlock(row-1,col+1)==true)
             if(isMine(row-1,col+1,playBoard)==true)
                  cnt++;

        //for left block
        if(isValidBlock(row,col-1)==true)
             if(isMine(row,col-1,playBoard)==true)
                  cnt++;

        //for right block
        if(isValidBlock(row,col+1)==true)
             if(isMine(row,col+1,playBoard)==true)
                  cnt++;

        //for lower left block
        if(isValidBlock(row+1,col-1)==true)
             if(isMine(row+1,col-1,playBoard)==true)
                  cnt++;

        //for lower block
        if(isValidBlock(row+1,col)==true)
             if(isMine(row+1,col,playBoard)==true)
                  cnt++;

        //for lower right block
        if(isValidBlock(row+1,col+1)==true)
             if(isMine(row+1,col+1,playBoard)==true)
                  cnt++;

        return cnt;
}
```

- In the above function adjMines(), it already takes row and column of a particular coordinate.
- With the help of that the function will check first the neighbouring block exists or not and if yes then it checks whether that neighbour coordinate contains mine or not.
- If the mine is their so the count variable's value will increase and by this at last we will get the number of adjacent mines at a particular coordinate.

```
#include<iostream>
#define N 10
using namespace std;
bool isValidBlock(int row,int col)
{
    if(((row>=0)&&(row<N))&&((col>=0)&&(col<N)))
     return true;
```

```
    else
     return false;
}
bool isMine(int row,int col,char a[][N])
{
    if(a[row][col]=='X')
     return true;
    else
     return false;
}
void makeMove(char a[][N],char playBoard[][N],int row,int col,int
movesLeft)
{
    playBoard[row][col]='0';
    if(isValidBlock(row-1,col)==true)
    {
     if(isMine(row-1,col,a)==false)
     {
          if(playBoard[row-1][col]!='0')
          {
               playBoard[row-1][col]='0';
               (movesLeft)--;
          }
     }
    }
    if(isValidBlock(row,col-1)==true)
    {
     if(isMine(row,col-1,a)==false)
     {
          if(playBoard[row][col-1]!='0')
          {
               playBoard[row][col-1]='0';
               (movesLeft)--;
          }
     }
    }
    if(isValidBlock(row,col+1)==true)
    {
     if(isMine(row,col+1,a)==false)
     {
          if(playBoard[row][col+1]!='0')
          {
               playBoard[row][col+1]='0';
               (movesLeft)--;
          }
     }
    }
    if(isValidBlock(row+1,col)==true)
    {
     if(isMine(row+1,col,a)==false)
     {
```

```
          if(playBoard[row+1][col]!='0')
          {
               playBoard[row+1][col]='0';
               (movesLeft)--;
          }
      }
     }
}
void printBoard(char a[][N])
{
     for(int i=0;i<N;i++)
     {
      for(int j=0;j<N;j++)
      {
           cout<<a[i][j]<<" ";
      }
      cout<<endl;
     }
}
void placeMines(char a[N][N],int noOfMines)
{
     int row,col;
     for(int i=0;i<N;i++)
     {
      for(int j=0;j<N;j++)
      {
           a[i][j]='#';
      }
     }
     cout<<"Enter number of mines:";
     cin>>noOfMines;
     for(int i=0;i<noOfMines;i++)
     {
      cout<<"Enter coordinates of mines:";
      cin>>row>>col;
      a[row][col]='X';
     }
     printBoard(a);
}
void initializeBoard(char a[][N],char playBoard[][N])
{
     for(int i=0;i<N;i++)
     {
      for(int j=0;j<N;j++)
      {
           a[i][j]=playBoard[i][j]='#';
      }
     }
     return;
}
void playGame()
```

```
{
    bool gameStatus=false;
    int row,col;
    int noOfMines;
    char a[N][N],playBoard[N][N];
    initializeBoard(a,playBoard);
    placeMines(a,noOfMines);
    int movesLeft=N*N-noOfMines;
    while(gameStatus==false)
    {
     cout<<"Current board:"<<endl;
     printBoard(playBoard);
     cout<<"\nEnter row and column of move:";
     cin>>row>>col;
     if(isMine(row,col,a)==true)
     {
          cout<<"Game Over"<<endl;
          gameStatus=true;
     }
     else if((gameStatus==false)&&(movesLeft==0))
     {
          cout<<"\nCongratulations Game won!!!!!"<<endl;
          gameStatus=true;
     }
     else
     {
          makeMove(a,playBoard,row,col,movesLeft);
     }
    }
    return;
}
int main()
{
    playGame();
    return 0;
}
```

- This is above program is just a pseudocode of the whole program of minesweeper.
- The above program will show only the neighbouring coordinates of the coordinate selected, where as the minesweeper game shows some more coordinates.