Minesweeper

By: John Haller

**Introduction:**

My purpose of my program is to create a game of minesweeper. I chose this game because i felt that it was a game that was doable for me, while still maintaining a complex enough theme to challenge me. I also wanted to choose this game because it is one that i used to play a lot, whenever the internet went out, and one that I had a lot of fun playing.

**Summary:**

My program is 335 lines of code

There are 19 major variables

As well as many counter variables used for loops

This program took me approximately a week and a half spread out over the last month to make it.

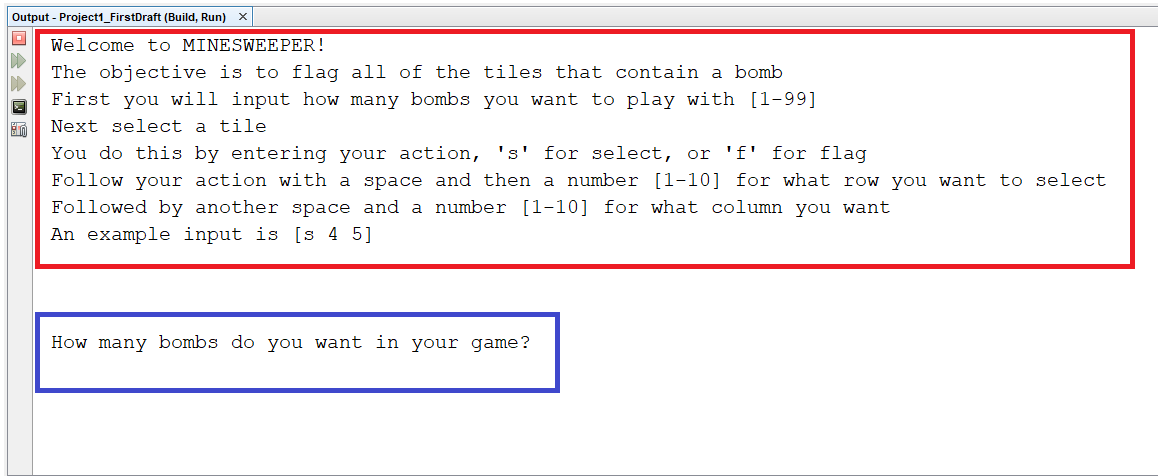
The most challenging part of the project was by far figuring out the logic and implementation of how to reveal tiles if you clicked on one that was not surrounded by bombs. It took me probably over 10 hours on just that, including time in lab getting assistance from Michael. But eventually I figured it out, and that was the most rewarding part of the project.

**Description:**

The general flow of the program starts with the idea that there is two arrays constantly in use, one (bombAry) is the grid that actually has the bombs on it and after it sets the bombs is never changed. The second (playAry) is what the player sees and interacts with. So the first step is to fill bombAry such that it is filled with ‘X’ with a border of ‘b’ around the playable field. This pattern is then copied to playAry, and after it is copied bombAry is filled with however many bombs the user chooses to play with. Now is when the user starts making their selections as to where they pick and what action to do. The two actions available in this game are ‘f’ to flag a tile, or ‘s’ to select a tile. Note: the user is also able to flag a tile that has already been flagged to unflag it. Following the action they input which row and column they want to select, in this notation. s 4 5 . Command would select the tile located on the 4th row and the 5th column.

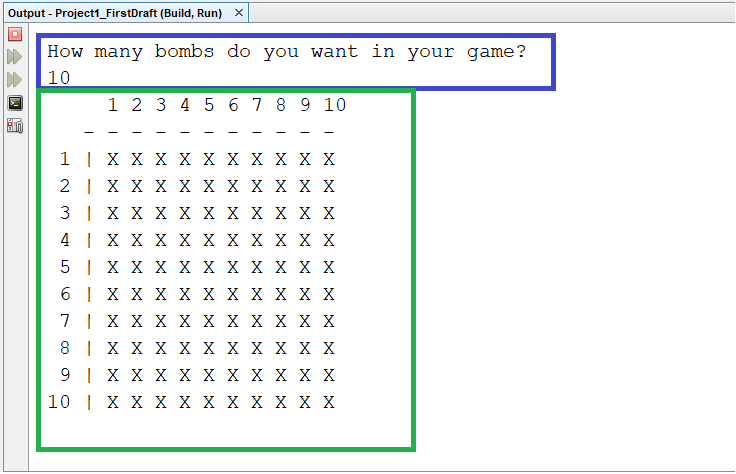
After noting the user’s input the program checks to see if that tile is a bomb, if so they lose. If not it will check all 8 tiles around that space and set the chosen tile to the number of bombs around it. For example a chosen tile with 3 bombs around it will then be displayed as a ‘3’. A tile with no bombs around it will be displayed as ‘ ‘. Then if and only if there were no bombs, it moves on to reveal surrounding tiles. It will repeat the same checking on the tiles in the cardinal directions until a tile in that direction is anything but a ‘ ‘. Meaning as soon as it finds a tile with a bomb near it, it will stop going in that direction. The game ends when the user has selected a bomb tile, or when the user has flagged all of the bombs on the grid.

**Sample I/O**



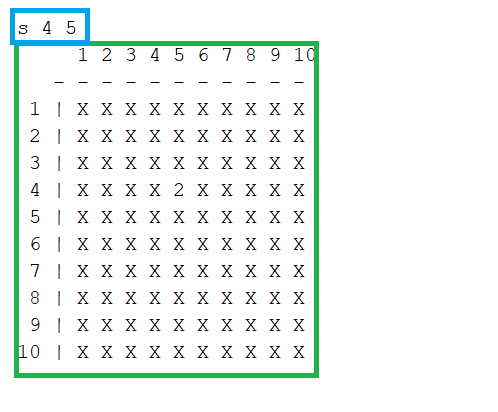
The red box is the start of game output describing the game and how you input your turns.

The blue box is the game prompting you for the number of bombs you want.



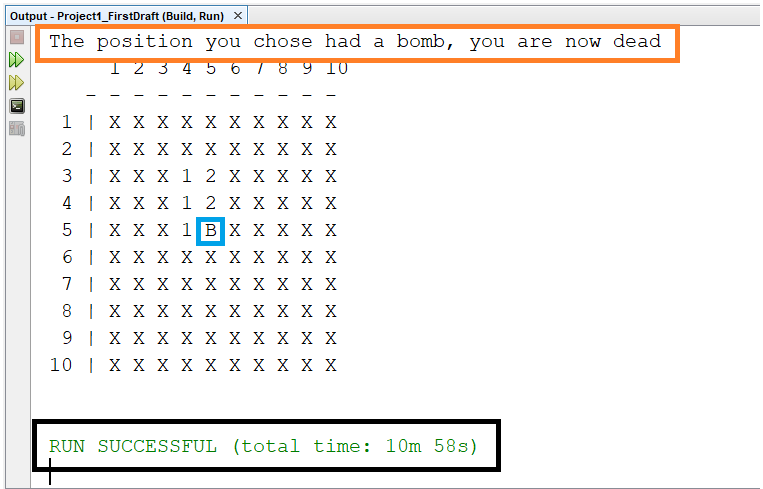
The blue box is the same prompt from before but now it has the user’s input, in this case it is 10.

The green box is the first instance of displaying the users grid, Note: this is not the bomb grid.



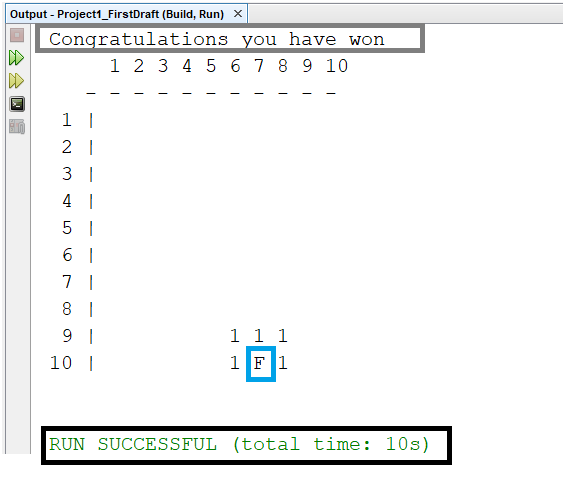
This time the blue box is the user’s choice for what they want to do and where to do it.

The green box is the grid after calculating the user’s move. Note: There are 2 bombs around the user’s tile.



This time the user’s tile (blue) was a bomb, this cause them to lose and the text in orange was displayed to show that.

Also the black box shows that the program ends after selecting a bomb.



In this image the user’s tile was flagged, and as you can see it was the only bomb on the grid.

Causing the game to end (black) and the text (gray) displaying that you have won.

**Flowcharts:**

Flowcharts for this program would take far too many pages so instead they are being included in a separate folder in the Project1 folder.

**Pseudocode:**

Set random number seed

input output operator

open file in bin mode for input/output

Declare intro strings

Declare enumeration for use in a stupid way for points

Copy welcome string to 1

write contents of wlcome1 to binary file

read in contents of binary file to string

Display instructions

Declare Variables and initialize variables for game

Fill Arrays and print playAry

Choose a position and search for bombs

checks to see if player is alive and has not won yet

inputs user's choice and checks for bombs

Reveal surrounding tiles if applicable

If dead display defeat screen

If win display victory screen

Destroy dynamic components

Exit stage right!

**Doxygen:**

The doxygen html file will be included in the Project1 folder that is turned in.

**Major Variables:**

int strngSz - The size for welcome strings

char wlcome1[strngSz] - Welcome string

char wlcome2[strngSz] - Welcome string

char wlcome3[strngSz] - Welcome string

enum stupid - For use in a stupid way to get points

int rows - Rows for 2d array

char playAry[rows][COLS] - 2D player array

char \*\*dynAry=initAry(rows) - 2D array to be copied to struc

Bomb \*array2D - Pointer to struc

bool alive - Used to tell if player is alive

int nFlags - Number of flags player has set

int bombFlg - Number of flags succesfully placed on bombs

bool win - Used to tell if player has won

bool live - players life status

char action - Player’s action

int chseRow - Row chosen by player

int chseCol - Col chosen by player

int bombCnt - number of bombs found around tile

**Concepts used:**

Pointer Variables - 134

Arrays/Pointers - 79

Function Parameters - 201

Memory Allocation - 134

Return parameters - 131

C-Strings - 62

Strings - 107

Arrays - Bomb.h file, 20

Nested - 144

Function Arguments - 97

Function Return - 138

Pointers - 134

Enumeration - 58,61

Formatting - 158

Function Parameters - 25-39

Binary Files - 48

Records with Structures - Bomb.h file, 17

Input/Output simultaneous - 48

**References**

The only reference I used was a piece of code written by Dr. Mark Lehr, for his spring 17a class.

The link to it is here - <https://github.com/ml1150258/2018_Fall_17a_LehrMark/tree/master/Class/Structure_of_2D_Dynamic_Array>

I used this for help with setting up a structure and how to properly access it.

**Code for program:**

Main:

/\*

\* File: main.cpp

\* Author: John Haller

\* Created on August 29, 2018, 12:58 PM

\* Purpose: Class work

\*/

//System Libraries

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <ctime>

#include <cstring>

#include <fstream>

using namespace std;

//User Libraries

#include "Bomb.h"

//Global Constants - Math/Physics Constants, Conversions,

// 2-D Array Dimensions

const int COLS=12;//Two more than necessary to have border

//Function Prototypes

void intro(string);

char \*\*initAry(int);

Bomb \*filStrc(char \*\*,int);

void fillAry(Bomb \*,char [][COLS]);

void copyAry(char [][COLS],Bomb \*);

void prntAry(char [][COLS],int);

void prntAry(Bomb \*);

bool choice(Bomb \*,char [][COLS],int,int,int,bool &);

void bombs(Bomb \*);

void cntBomb(Bomb\*);

bool check(Bomb \*,int,int,char [][COLS]);

void reveal(Bomb \*,int,int,char [][COLS]);

void death();

bool victory(Bomb \*,int,int);

void destroy(Bomb \*);

//Execution Begins Here

int main(int argc, char\*\* argv) {

//Set random number seed

srand(static\_cast<unsigned int>(time(0)));

//File stuff

fstream file;//input output operator

file.open("intro.dat",ios::in|ios::out|ios::binary);//open file in bin mode

//for input/output

//Declare intro strings

int strngSz=30;//size for welcome strings

char wlcome1[strngSz]= " ";//first welcome string

char wlcome2[strngSz]= "Welcome to MINESWEEPER!";//second welcome string

char wlcome3[strngSz]= "";//third welcome string

//Declare enumeration for use in a stupid way for points

enum stupid{one,two};

//Copy welcome string to 1

if(one<two){//stupid way to use enumeration

strcpy(wlcome1,wlcome2);//copy string 2 onto string 1

}

//write contents of wlcome1 to binary file

file.write(wlcome1,sizeof(wlcome1));//write contents of wlcome1 to bin file

file.close();//close file

//read in contents of binary file to string

file.open("intro.dat",ios::in|ios::out|ios::binary);//reopen file for input

file.read(wlcome3,sizeof(wlcome3));//read in from file

//Display instructions

intro(wlcome3);//display first line of intro

//Declare Variables and initialize variables for game

int rows=12;//Two more than necessary for border

char playAry[rows][COLS];//Grid that player sees and uses

char \*\*dynAry=initAry(rows);

Bomb \*array2D=filStrc(dynAry,rows);//copy contents of dynamic array to bomb

bool alive=true;//If player is alive or dead

int nFlags=0;//counter for number of flags

int bombFlg=0;//counter for number of flags on bombs

bool win=false;

//Fill Arrays and print playAry

initAry(rows);//initialize array to 'b' for border

fillAry(array2D,playAry);//fill in usable part of array with 'X'

prntAry(playAry,rows);//Print the array

// prntAry(array2D); //used for testing to manually compare grids

// cntBomb(array2D); //used for testing to verify bomb count

//Choose a position and search for bombs

while(alive&&!win){//checks to see if player is alive and has not won yet

alive=choice(array2D,playAry,rows,nFlags,bombFlg,win);//inputs user's choice and checks for bombs

}

//Destroy dynamic components

destroy(array2D);

//Exit stage right!

return 0;

}

void intro(string welcome){//intro/instructions for game

cout<<welcome<<endl;

cout<<"The objective is to flag all of the tiles that "

<<"contain a bomb"<<endl;

cout<<"First you will input how many bombs you "

<<"want to play with [1-99]"<<endl;

cout<<"Next select a tile"<<endl;

cout<<"You do this by entering your action, 's' for select, "

<<"or 'f' for flag"<<endl;

cout<<"Follow your action with a space and then a number [1-10] "

<<"for what row you want to select"<<endl;

cout<<"Followed by another space and a number [1-10] for what "

<<"column you want"<<endl;

cout<<"An example input is [s 4 5]"<<endl<<endl<<endl<<endl;

}

char \*\*initAry(int row){

char \*\*a=new char\*[row];

for(int i=0;i<row;i++){

a[i]=new char[COLS];

}

for(int i=0;i<row;i++){

for (int j=0;j<COLS;j++){

a[i][j]='b';//initialize border to 'b'

}

}

return a;//return the array

}

Bomb \*filStrc(char \*\*a,int row){

Bomb \*d2=new Bomb;

d2->rows=row;

d2->cols=COLS;

(\*d2).bombAry=a;

return d2;

}

void fillAry(Bomb \*d2,char playAry[][COLS]){

for(int row=1;row<d2->rows-1;row++){

for(int col=1;col<d2->cols-1;col++){

d2->bombAry[row][col]='X';//Fill in the play area with X's

}

}

copyAry(playAry,d2);//copy bomb grid to player grid

bombs(d2);//insert bombs onto bomb grid

}

void copyAry(char playAry[][COLS],Bomb \*d2){

for(int i=0;i<d2->rows;i++){

for(int j=0;j<d2->cols;j++){

playAry[i][j]=d2->bombAry[i][j];//copy bomb grid to player grid

} //before bombs are set

}

}

void prntAry(char playAry[][COLS],int rows){

cout<<setw(4)<<' '; //Formatting output

for(int i=1;i<COLS-1;i++){

if(i==10)cout<<' ';

cout<<setw(2)<<i;

}

cout<<endl;

for(int i=0;i<COLS-1;i++){ //Format output

if(i==0)cout<<setw(2)<<" ";

cout<<setw(2)<<'-';

}

cout<<endl;

for(int row=1;row<rows-1;row++){

cout<<setw(2)<<row;

cout<<setw(2)<<'|'; //format output

for(int col=1;col<COLS-1;col++){

cout<<setw(2)<<playAry[row][col];//output player grid

}

cout<<endl;

}

cout<<endl;

}

void prntAry(Bomb \*d2){//used for testing purposes to print bomb array

cout<<setw(4)<<' ';

for(int i=1;i<d2->cols-1;i++){

if(i==10)cout<<' ';

cout<<setw(2)<<i;

}

cout<<endl;

for(int i=0;i<d2->cols-1;i++){

if(i==0)cout<<setw(2)<<" ";

cout<<setw(2)<<'-';

}

cout<<endl;

for(int row=1;row<d2->rows-1;row++){

cout<<setw(2)<<row;

cout<<setw(2)<<'|';

for(int col=1;col<d2->cols-1;col++){

cout<<setw(2)<<d2->bombAry[row][col];//output bomb elements

}

cout<<endl;

}

cout<<endl;

}

bool choice(Bomb \*d2,char playAry[][COLS],int rows,int nFlags,int bombFlg,

bool &win){

bool live=true;// bool for player's life state

char action;//which action the player will take [flag/selection]

int chseRow;//chosen row

int chseCol;//chosen col

cin>>action>>chseRow>>chseCol;//input action/row/col

if(chseRow<1||chseRow>10){//check for valid row input

cout<<"Invalid row number, ending program"<<endl;

exit(EXIT\_FAILURE);//exit if invalid

}

if(chseCol<1||chseCol>10){//check for valid col input

cout<<"Invalid col number, ending program"<<endl;

exit(EXIT\_FAILURE);//exit if invald

}

if(action=='f'){

nFlags++;//increment flag counter

if(playAry[chseRow][chseCol]=='F'){//if flagging a flag

playAry[chseRow][chseCol]='X';//option to unflag by flagging it again thanks mom

nFlags--;//Decrement since flag was undone

}

else playAry[chseRow][chseCol]='F';//set tile to 'F' for flag choice

if(d2->bombAry[chseRow][chseCol]=='B')bombFlg++;//increment bomb flag counter

}else if(action=='s'){

live=check(d2,chseRow,chseCol,playAry);//Select tile and send to check

}else{//check for invalid action input

cout<<"Invalid input for action, ending program"<<endl;

exit(EXIT\_FAILURE);//exit if invalid

}

// if(live)reveal(a,chseRow,chseCol);

prntAry(playAry,rows);//print array after choice

win=victory(d2,nFlags,bombFlg);//check if player has won

if(win)prntAry(playAry,rows);//print array final time if they won

return live;//return life state

}

void bombs(Bomb \*d2){//used to generate bombs in bomb grid

cout<<"How many bombs do you want in your game?"<<endl;

cin>>d2->nBombs;//input number of bombs for game

for(int bombs=0;bombs<(d2->nBombs);bombs++){//generates bombs until nBombs unique bombs

int bombRow=rand()%10+1;//chooses random row for bomb

int bombCol=rand()%10+1;//chooses random col for bomb

if(d2->bombAry[bombRow][bombCol]=='B'){//tests if bomb already in that spot

bombs-=1;//if bomb placed on old bomb do generate new one

}

d2->bombAry[bombRow][bombCol]='B';//sets bomb position

}

}

void cntBomb(Bomb \*d2){//used for testing to count bombs in bomb grid

int count=0;//bomb counter

for(int row=1;row<d2->rows-1;row++){

for(int col=1;col<d2->cols-1;col++){

if(d2->bombAry[row][col]=='B'){//search for bombs

count++;//increment bomb count

}

}

}

cout<<"There are "<<count<<" bombs"<<endl;//output bombs found

}

//Check function searches for bombs around the chosen tile and sets the tile = to that number

//if no bombs found it sets it to ' '

//if chosen tile is bomb it returns false to set live to false

//after calculations are done it sends the chosen tile to reveal function-

//for further checks

bool check(Bomb \*d2,int chceRow,int chceCol,char playAry[][COLS]){

if(playAry[chceRow][chceCol]!='X')return true;//checks if tile has been checked already

int bombCnt=0;//how many bombs are around tile

if(d2->bombAry[chceRow][chceCol]=='B'){//checks if hit bomb

playAry[chceRow][chceCol]='B';//if so set tile to 'B'

death();//call death function

return false;//return false for live bool

}

if(d2->bombAry[chceRow-1][chceCol-1]=='B')bombCnt++;//checks top left corner

if(d2->bombAry[chceRow-1][chceCol]=='B')bombCnt++;//check middle top

if(d2->bombAry[chceRow-1][chceCol+1]=='B')bombCnt++;//checks top right

if(d2->bombAry[chceRow][chceCol-1]=='B')bombCnt++;//checks middle left

if(d2->bombAry[chceRow][chceCol+1]=='B')bombCnt++;//checks middle right

if(d2->bombAry[chceRow+1][chceCol-1]=='B')bombCnt++;//checks bottom left

if(d2->bombAry[chceRow+1][chceCol]=='B')bombCnt++;//checks bottom middle

if(d2->bombAry[chceRow+1][chceCol+1]=='B')bombCnt++;//checks bottom right

playAry[chceRow][chceCol]=bombCnt+48;//convert bomb count to char

if(bombCnt==0)playAry[chceRow][chceCol]=' ';//if no bombs set to ' '

if(playAry[chceRow][chceCol]==' '){//if no bombs near go to reveal function

reveal(d2,chceRow,chceCol,playAry);

}

return true;

}

//The reveal function only serves to resend the surrounding tiles back into the

//check function

//This is used to reveal an area around the chosen tile if no bombs were found

//nearby

void reveal(Bomb \*d2,int chosRow,int chosCol,char playAry[][COLS]){

if(playAry[chosRow-1][chosCol]!='B'){//checks top middle

check(d2,chosRow-1,chosCol,playAry);

}

if(playAry[chosRow][chosCol-1]!='B'){//checks middle left

check(d2,chosRow,chosCol-1,playAry);

}

if(playAry[chosRow+1][chosCol]!='B'){//checks bottom middle

check(d2,chosRow+1,chosCol,playAry);

}

if(playAry[chosRow][chosCol+1]!='B'){//checks middle right

check(d2,chosRow,chosCol+1,playAry);

}

if(playAry[chosRow-1][chosCol-1]!='B'){//checks top left

check(d2,chosRow-1,chosCol,playAry);

}

if(playAry[chosRow+1][chosCol-1]!='B'){//checks bottom left

check(d2,chosRow,chosCol-1,playAry);

}

if(playAry[chosRow-1][chosCol+1]!='B'){//checks top right

check(d2,chosRow+1,chosCol,playAry);

}

if(playAry[chosRow+1][chosCol+1]!='B'){//checks bottom right

check(d2,chosRow,chosCol+1,playAry);

}

}

void death(){//used to output that you died incase user didn't notice

cout<<"The position you chose had a bomb, you are now dead"<<endl;

}

bool victory(Bomb \*d2,int nFlags,int bombFlg){//Used to determine if player won

if(nFlags==d2->nBombs&&bombFlg==d2->nBombs){

cout<<"Congratulations you have won"<<endl;

return true;

}

return false;

}

void destroy(Bomb \*d2){//delete dynamic elements

for(int i=0;i<d2->rows;i++){

delete []d2->bombAry[i];

}

delete []d2->bombAry;

delete d2;

}

Bomb.h File:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*

\* File: Bomb.h

\* Author: rcc

\*

\* Created on October 18, 2018, 10:57 AM

\*/

#ifndef BOMB\_H

#define BOMB\_H

struct Bomb{

int rows;

int cols;

char \*\*bombAry;

int nBombs;

};

#endif /\* BOMB\_H \*/