Oscar Mayorga

CSC 17A

Dr. Mark Lehr

Project 2

Minesweeper Game

June 08 2016

Introduction:

Minesweeper consists of uncovering the coordinates in this case without uncovering the mine which if you uncover the mine you lose the game. The Player has to uncover coordinates and try to not uncover the mine. There are only one mine in the easy mode, if the player chooses to play the normal and hard mode then there will be more mines and the player would have more chances to loose the game. Also the player has the opportunity to choose the size of the game board, the game board can be from 1X1 to 10X10.

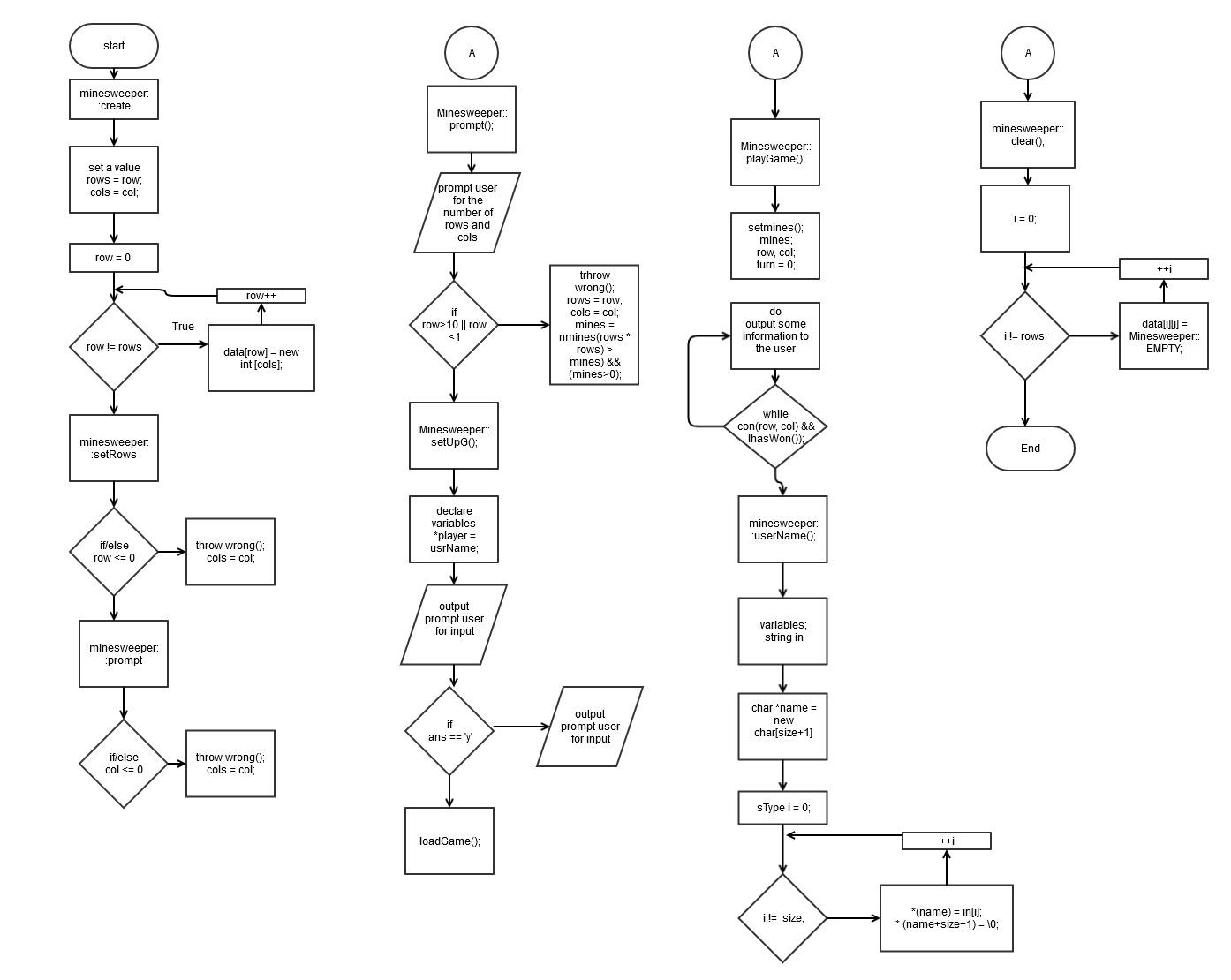
Summary:

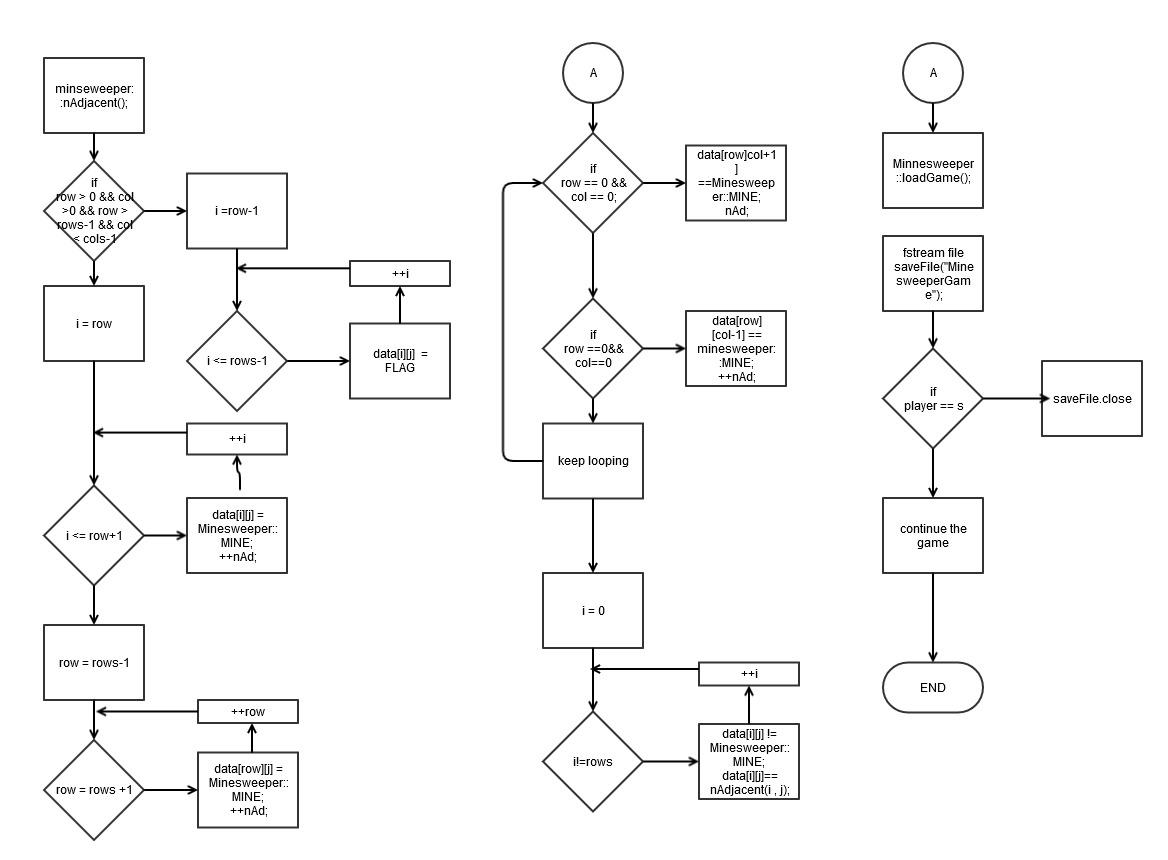
Number of lines: 915

Number of variables: 20

Number of classes: 4

Flowchart





Major Variables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Variable | Description | Location | Library |
| int | rows | This variable sets the number of rows for the game board (0-10) | Line number 20 | GameBoard.h |
|  | cols | This sets the number of columns for the size of the game board | Line number 21 | GameBoard.h |
|  | row | This gives a value to the rows variable in the class | Line number 16 | GameBoard.cpp |
|  | col | This gives a value to the cols variable in the class | Line number 17 | GameBoard.cpp |
|  | diff | This variable is used to store the difficulty level of the game Easy, normal or Hard when the player enters the choice. | Line number 54 | GameBoard.cpp |
|  | \*\*data | This 2D pointer allocates the address of the game board and stores the address of the rows and cols variables | Line number 19 | GameBoard.h |
| enum | difficulty | This enum holds the difficulty levels of the game. | Line number 21 | Minesweeper.h |
|  | Squares | This enum holds the Empty. Loser, mines and clear variables. | Line number23 | Minesweeper.h |

Pseudo Code;

/// dinamically create a Minesweeper /// Create each column /// Function returns true if input was valid /// make sure that the number of mines does not exceed

/// the number of spots available and that mines exist /// Get the user name /// ask user if they want to play /// play if answer is yes

/// Get game information from user /// Get new data only if user wants to continue /// Information was invalid /// Play a game of minesweeper /// User inputs how many rows and columns and the difficulty /// Select the row /// User wants to save the game

/// save the game and exit /// check bounds

/// Select the column /// Prepare to print completed Minesweeper /// Print the complete Minesweeper /// Function gets the user name as a string converts it to a char array /// for the 1d dynamic array requirement /// Function that clears the grid on which game will be played /// Make sure each square is empty /// Function return the Minesweeper::Difficulty type from /// the int variable

/// Function prints the Minesweeper with spaces hidden /// Print the column index /// Pad initial output of column indicator /// KEEP EMPTY spaces and MINEs hidden

/// print out the CLEARED area /// Function returns the number of mines to set based on Difficulty /// not on first or last row or first or last column

/// most of the searches take place in this area /// on the first row, not on first or last column /// on the last row, not on first or last column /// on the first column, not on first or last row

/// search to the right /// on the last column, not on first or last row

/// search to the left /// top left corner /// Function returns true if /// there are 0 landmines adjacent to selected square /// Clear an area whose values are CLEAR /// i.e 0 adjacent mines /// Function shows how many mines are adjacent to selected square /// for the entire Minesweeper /// Function reveals what is underneath the square that the user has selected /// and whether to continue based on what is revealed /// i.e selecting a mine means you lost, game over /// Square had adjacent mine

/// reveal the number to the user /// Function checks whether the player has won

/// if there are no EMPTY spaces left the game is won /// Function finds the perimeter of the cleared areas /// Function prints the data variable from the Minesweeper structure /// writen to a binary file ///print(); /// This is the class that holds the Minesweeper /// as well as the associated flags that occur when /// a user selects a square /// output this if user tries to set negative rows or columns

Abtracts.h

class Abstracts { protected: virtual void setRows(int)=0; virtual void setCols(int)=0; virtual int getRows() const =0; virtual int getCols() const =0; virtual void setUpG()=0; virtual void print() const = 0;

};

GameBoard.h

#include "Abstracts.h" /// Base class for games that require an n\*m array such as minesweeper

class GameBoard: public Abstracts{ private:

protected: int \*\*data; int rows; int cols; virtual void create(int, int);

public:

Code

Main.cpp

///System Libraries

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <vector>

#include <iomanip>

//user libraries

#include "Minesweeper.h"

#include "Templates.h"

using namespace std;

//function prototypes

void GameRules();

//execution begins here

int main(int argc, const char \* argv[]) {

//call function here to show the game rules

GameRules();

//Cast for random time seed generator

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

Game<GameBoard> m(new Minesweeper(10,10));

//throw the exeption

try {

m->setUpG();

}

//error output

catch (Minesweeper::wrong) {

cout << "Size was invalid\n";

}

catch (const char\* s) {

cout << s << endl;

}

return 0;

//Exit stage right

}

//function to output the rules at the beginning of the game

void GameRules()

{

cout<<"\*\*\*\*\*\*\*\*\*\* Minesweeper Rules \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"--- Please read the Rules carefully; GOODLUCK and ENJOY!!"<<endl;

cout<<"Welcome to minesweeper, first you are going to enter your username,"<<endl;

cout<<"then you would have to choose the difficulty of the game."<<endl;

cout<<"After you will have to choose how big the game board would be, "<<endl;

cout<<"you can enter choose from 1 X 1 to 10 X 10 (Rows X Columns). "<<endl;

cout<<"Then you will enter coordinates that would uncover mines,"<<endl;

cout<<"You would choose coordinates Until you find a mine, "<<endl;

cout<<"that would end the game and you would loose the game"<<endl;

cout<<""<<endl;

}

GameBoard.cpp

#include <iostream>

#include "gameBoard.h"

using namespace std;

// Function that creates the grid on which game will be played

void GameBoard::create(int row, int col) {

// dinamically create a Minesweeper

rows=row;

cols = col;

// Set up the rows

data = new int \*[rows];

// Create each column

for (int row = 0; row != rows; ++row)

data[row] = new int [cols];

}

// Function resets the GameBoard to initial in order to use it again

void GameBoard::clear() {

for (int i = 0; i != rows; ++i)

for (int j = 0; j != cols; ++j)

data[i][j] = 0;

}

// Function deallocates memory

void GameBoard::destroy() {

// delete each dynamically allocated row

for (int i = 0; i != rows; ++i)

delete[] data[i];

// delete the dynamically allocated structure

delete data;

}

void GameBoard::setRows(int row) {

if ( row <= 0 )

throw wrong();

rows = row;

}

void GameBoard::setCols(int col) {

if (col <= 0 )

throw wrong();

cols = col;

}

void GameBoard::print() const {

for (int i = 0; i != rows; ++i){

for (int j = 0; j != cols; ++j) {

cout << data[i][j] << " ";

}

cout <<endl;

}

}

void GameBoard::loadGame() {

cout << "Wrong data"<<endl;

}

void GameBoard::setUpG() {

cout << "Wrong data"<<endl;

}

MInesweeper.cpp

//System libraries

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

#include <iomanip>

//User Libraries

#include "Minesweeper.h"

using namespace std;

// Function gets the user name as a string converts it to a char array

// for the 1d dynamic array requirement

char\* Minesweeper::userName() {

cout << "Enter your username: "<<endl;

cout << "(No spaces, you can use only characters and numbers but no spaces!!) "<<endl;

string in;

cin >> in;

cout<<endl;

typedef string::size\_type sType;

sType size = in.size();

// make room for '\0'

char \*name = new char[size+1];

for (sType i = 0; i != size; ++i) {

\*(name+i) = in[i];

}

\*(name+size+1) = '\0';

return name;

}

void Minesweeper::setUpG() {

// Get the user name

char \*player = userName();

// ask user if they want to play

cout << "Hello " << player

<< ", Would you like to play minesweeper?"<<endl;

cout<<"Enter 'y' if yes or enter 'n' if not"<<endl;

char ans;

cin >> ans;

cout<<endl;

// play if answer is yes

if (ans == 'y' || ans == 'Y') {

//cout << "Please Enter n to start the game!!"<<endl;

//char ans2;

//cin >> ans2;

//cout<<endl;

//if ( ans2 == 'y' || ans2 == 'Y') {

//}

//else

// Get game information from user

prompt();

if (isValidIn()) {

while (ans == 'y' || ans == 'Y'&& isValidIn()) {

playGame();

cout << endl;

cin.ignore();

cout << "Would you like to play again " << player << "? "<<endl;

cin >> ans;

cout << endl;

// Get new data only if user wants to continue

if (ans =='y' || ans == 'Y') {

prompt();

clear();

}

}

}

// Information was invalid

else

throw wrong();

}

cout << "Game is Over."<<endl;

// Cleanup

delete player;

cout << endl;

cout << "Goodbye"<<endl;

}

void Minesweeper::prompt() {

cout << "Enter the number of rows(0-10)"<<endl;

cout<<"The Area will be rows x rows: "<<endl;

int row;

cin >> row;

// invalid sizes

if (row > 10 || row < 1)

throw wrong();

rows = row;

cols = row;

char diff;

cout << "Please Enter the difficulty (make sure is a lowercase letter)"<<endl;

cout<<"e=Easy"<<endl;

cout<<"n=Normal"<<endl;

cout<<"h=Hard"<<endl;

cin >> diff;

mines = nMines(intToDiff(diff));

}

void Minesweeper::create(int row, int col) {

// dinamically create a Minesweeper

rows=row;

cols = col;

// Set up the rows

data = new int \*[rows];

//Create each column

for (int row = 0; row != rows; ++row)

data[row] = new int [cols];

}

void Minesweeper::setRows(int row) {

if ( row <= 0 )

throw wrong();

rows = row;

}

void Minesweeper::setCols(int col) {

if (col <= 0 )

throw wrong();

cols = col;

}

// Function returns true if input was valid

bool Minesweeper::isValidIn() const{

//make sure that the number of mines does not exceed the number of spots available and that mines exist

return (((rows \* cols) > mines) && (mines>0));

}

// Play a game of minesweeper

// User inputs how many rows and columns and the difficulty

void Minesweeper::playGame() {

setMines();

prntObscr();

int row, col;

int turn = 0;

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

do {

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

cout << "Turn: " << turn++ << endl;

// Select the row

do {

cout << "Enter (-1) to exit"<<endl;

cout << "Enter the row " << 0 << "-" << rows-1 << ": "<<endl;

cin >> row;

// User wants to save the game

// save the game and exit

if ( row == -1) {

saveGame();

return;

}

// check bounds

} while (row < 0 || row >= rows);

// Select the column

do {

cout << "Enter the column " << 0 << "-" << cols-1 << ": "<<endl;

cin >> col;

// check bounds

} while (col < 0 || col >= cols);

// endTime

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

cout << "Turn took: " << endTime - begTime << " seconds.\n";

cout << endl;

} while (cont(row, col) && !hasWon());

// Prepare to print completed Minesweeper

if (hasWon()) {

cout << "You win"<<endl;

setSquares();

}

else{

cout << "You have, lost"<<endl;

setSquares();

data[row][col]= Minesweeper::LOSER;

}

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

cout << "Game was completed in " << finalTime - initialTime << " seconds."<<endl;

// Print the complete Minesweeper

print();

}

// Function that clears the grid on which game will be played

void Minesweeper::clear() {

// Make sure each square is empty

for (int i = 0; i != rows; ++i)

for (int j = 0; j != rows; ++j)

data[i][j] = Minesweeper::EMPTY;

}

// Function return the Minesweeper::Difficulty type from

// the int variable

Minesweeper::Difficulty Minesweeper::intToDiff(char choice) {

switch (choice) {

case 'e':

return Minesweeper::EASY;

break;

case 'n':

return Minesweeper::NORMAL;

break;

case 'h':

return Minesweeper::HARD;

default:

return Minesweeper::EASY;

break;

}

}

// Functions prints the Minesweeper with all the squares revealed.

// used mostly after player loses

void Minesweeper::print() const {

cout << "Here's what the board looked like"<<endl;

for (int row = 0; row != rows; ++row){

for (int col = 0; col != cols; ++col) {

//

if ( \*(\*(data+row) + col) == Minesweeper::LOSER)

cout << "t ";

else if (\*(\*(data+row) + col) == Minesweeper::MINE)

cout << "x ";

else if (!isClear(row, col))

cout << nAdjacent(row, col) << " ";

else

cout << "0 ";

}

cout << endl;

}

cout << endl;

}

// Function prints the Minesweeper with spaces hidden

void Minesweeper::prntObscr() const{

// Print the column index

for (int i = 0; i != cols; ++i){

// Pad initial output of column indicator

if (i==0)

cout << " ";

cout << setw(3) << i;

}

cout << endl;

for (int row = 0; row != rows; ++row){

for (int col = 0; col != cols; ++col){

if(col == 0 && row < 10) cout << row << " ";

if (col == 0 && row >= 10) cout << row << " ";

// KEEP EMPTY spaces and MINEs hidden

if (data[row][col] == Minesweeper::EMPTY ||

data[row][col] == Minesweeper::MINE)

cout << setw(3) << right << "\* ";

// print out the CLEARed area

else if (data[row][col] == Minesweeper::CLEAR)

cout << setw(2)<< 0 << " ";

// Print out the actual value of the square

else

cout << setw(2)<< data[row][col] << " ";

}

cout << endl;

}

cout << endl;

}

// Function returns the number of mines to set based on Difficulty

int Minesweeper::nMines(Minesweeper::Difficulty d) const {

if (d==Minesweeper::EASY)

return (rows\*cols)/10;

else if (d==Minesweeper::NORMAL)

return (rows\*cols)/5;

else

return (rows\*cols)/3;

}

// Function places mines in grid

void Minesweeper::setMines() {

int minecpy = mines;

// keep looping through Minesweeper until all mines are set

while (minecpy) {

for (int i = 0; i != rows; ++i) {

for (int j = 0; j != cols; ++j) {

// place mines if result of rand()%15 == 0

if ((rand() % 100) % 10 == 0){

//only place mines if mines are still available

// and current space is empty

if (minecpy && data[i][j] == Minesweeper::EMPTY) {

// set the mine

data[i][j] = Minesweeper::MINE;

// decrement number of mines available

--minecpy;

}

}

}

}

}

}

// Function returns how many 'flag' elements surround a given square

int Minesweeper::nAdjacent(int row, int col, int FLAG) const{

int nAd=0; /// the number of adjacent mines

// not on first or last row or first or last column

// most of the searches take place in this area

if ( row > 0 && col > 0 && row < rows-1 && col < cols-1) {

// search the 3x3 grid surrounding a cell

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

for (int j = col-1; j <= col+1; ++j)

if (data[i][j] == FLAG)

++nAd;

}

}

// on the first row, not on first or last column

else if ( row == 0 && col > 0 && col < cols - 1) {

for (int i = row; i <= row+1; ++i) {

for (int j = col-1; j <= col+1; ++j)

if (data[i][j] == Minesweeper::MINE)

++nAd;

}

}

// on the last row, not on first or last column

else if ( row == rows-1 && col > 0 && col < cols - 1) {

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

for (int j = col-1; j <= col+1; ++j)

if (data[i][j] == Minesweeper::MINE)

++nAd;

}

}

// on the first column, not on first or last row

// search to the right

else if ( col == 0 && row > 0 && row < rows - 1) {

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

for (int j = col; j <= col+1; ++j)

if (data[i][j] == Minesweeper::MINE)

++nAd;

}

}

// on the last column, not on first or last row

// search to the left

else if ( col == cols-1 && row > 0 && row < rows - 1) {

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

for (int j = col-1; j <= col; ++j)

if (data[i][j] == Minesweeper::MINE)

++nAd;

}

}

// top left corner

else if (row == 0 && col == 0) {

if (data[row][col+1] == Minesweeper::MINE) ++nAd;

if (data[row+1][col] == Minesweeper::MINE) ++nAd;

if (data[row+1][col+1] == Minesweeper::MINE) ++nAd;

}

// top right corner

else if (row == 0 && col == cols-1) {

if (data[row][col-1] == Minesweeper::MINE) ++nAd;

if (data[row+1][col] == Minesweeper::MINE) ++nAd;

if (data[row+1][col-1] == Minesweeper::MINE) ++nAd;

}

// bottom left corner

else if (row == rows-1 && col == 0) {

if (data[row-1][col] == Minesweeper::MINE) ++nAd;

if (data[row-1][col+1] == Minesweeper::MINE) ++nAd;

if (data[row][col+1] == Minesweeper::MINE) ++nAd;

}

// bottom right corner

else if (row == rows-1 && col == cols-1) {

if (data[row-1][col-1] == Minesweeper::MINE) ++nAd;

if (data[row-1][col] == Minesweeper::MINE) ++nAd;

if (data[row][col-1] == Minesweeper::MINE) ++nAd;

}

// return number of mines from appropriate if statement

return nAd;

}

// Function returns true if

// there are 0 landmines adjacent to selected square

bool Minesweeper::isClear(int row, int col) const {

if (nAdjacent(row, col))

return false; // nAdjacent returned 1 or more

return true; // nAdjacent returned 0

}

// Clear an area whose values are CLEAR i.e 0 adjacent mines

void Minesweeper::showZeros(int row, int col) {

// check bounds

if ( row >= rows || row < 0 || col >= cols || col < 0)

return;

if (isClear(row, col) && data[row][col] != Minesweeper::CLEAR){

data[row][col] = Minesweeper::CLEAR;

// go up one row

showZeros(row+1, col);

// go down one row

showZeros(row-1, col);

// go right one col

showZeros(row, col+1);

// go left one col

showZeros(row, col-1);

}

// space was not clear or already shown

else

return;

}

// Function shows how many mines are adjacent to selected square

// for the entire Minesweeper

void Minesweeper::setSquares() {

for (int i = 0; i != rows; ++i)

for (int j = 0; j != cols; ++j)

// don't look for adjacent mines in areas where

// mine is already located

if (data[i][j] != Minesweeper::MINE)

data[i][j] = nAdjacent(i, j);

}

// Function reveals what is underneath the square that the user has selected

// and whether to continue based on what is revealed

// i.e selecting a mine means you lost, game over

bool Minesweeper::cont(int row, int col) {

// check if user selected a losing square

if (data[row][col] == Minesweeper::MINE)

return false;

// Square is a zero, clear the surrounding area if necessary

else if (isClear(row, col) ){

showZeros(row, col); // show cleared area

setPerim();

prntObscr();

return true;

}

// Square had adjacent mine

// reveal the number to the user

else {

data[row][col] = nAdjacent(row, col);

prntObscr();

return true;

}

}

// Function checks whether the player has won

// if there are no EMPTY spaces left the game is won

bool Minesweeper::hasWon() const {

for (int i = 0; i != rows; ++i)

for (int j = 0; j != cols; ++j)

// if there are empty spaces player has not won

if (data[i][j] == Minesweeper::EMPTY)

return false;

// there were no empty spaces left. Player has won

return true;

}

// Function finds the perimeter of the cleared areas

void Minesweeper::setPerim() {

for (int row = 0; row != rows; ++row ) {

// avoid searching at left and right edge of array

for (int col = 0; col != cols; ++col) {

// when you're not on the bounds of the array

if (row > 0 && row < rows-1

&& col > 0 && col < cols-1){

if (data[row][col] == Minesweeper::CLEAR) {

// check that the previous number has mines adjacent

if (data[row][col-1] != Minesweeper::CLEAR)

data[row][col-1] = nAdjacent(row, col-1);

// check if the next number has mines adjacent

if (data[row][col+1] != Minesweeper::CLEAR)

data[row][col+1] = nAdjacent(row, col+1);

if (data[row-1][col] != Minesweeper::CLEAR)

data[row-1][col] = nAdjacent(row-1, col);

// check if the next number has mines adjacent

if (data[row+1][col] != Minesweeper::CLEAR)

data[row+1][col] = nAdjacent(row+1, col);

// check the adjacent corners

if (data[row-1][col-1] != Minesweeper::CLEAR)

data[row-1][col-1] = nAdjacent(row-1, col-1);

if (data[row-1][col+1] != Minesweeper::CLEAR)

data[row-1][col+1] = nAdjacent(row-1, col+1);

if (data[row+1][col-1] != Minesweeper::CLEAR)

data[row+1][col-1] = nAdjacent(row+1, col-1);

if (data[row+1][col+1] != Minesweeper::CLEAR)

data[row+1][col+1] = nAdjacent(row+1, col+1);

}

}

}

}

}

void Minesweeper::saveGame() {

fstream saveFile("gameSave", ios::out | ios::binary);

saveFile.write(reinterpret\_cast<char\*>(this), sizeof(\*this));

saveFile.close();

}

//Function prints the data variable from the Minesweeper structure

//writen to a binary file

void Minesweeper::loadGame() {

fstream saveFile("gameSave", ios::in | ios::binary);

if (!saveFile.is\_open())

throw "No previous settings found\n";

saveFile.read(reinterpret\_cast<char\*>(this), sizeof(\*this));

print();

saveFile.close();

}

Minesweeper& Minesweeper::operator=(const Minesweeper &rhs) {

create(rhs.getRows(), rhs.getCols());

for (int i = 0; i != rhs.getRows(); ++i) {

for (int j = 0; j != rhs.getCols(); ++j)

data[i][j] = rhs[i][j];

}

return \*this;

}

Templates.h

#ifndef TEMPLATES\_H

#define TEMPLATES\_H

template<class T>

class Game {

private:

T\* p;

public:

Game():p(0){}

Game(T\* t):p(t){}

~Game() {delete p;}

Game<T>& operator=(const T&);

operator bool(){return p;}

T\* operator->() const;

T& operator\*() const;

};

template<class T>

T\* Game<T>::operator->() const {

// only return p if it points to something

if (p)

return p;

return 0;

}

template<class T>

T& Game<T>::operator\*() const{

if (p)

return \*p;

}

template<class T>

Game<T>& Game<T>::operator=(const T& rhs) {

p = &rhs;

}

#endif /\* TEMPLATES\_H \*/

Minesweeper.h

#ifndef MINESWEEPER\_H

#define MINESWEEPER\_H

//System Library

#include <string>

//User Library

#include "GameBoard.h"

//This class holds the minesweeper

class Minesweeper:

public GameBoard {

private:

// Determines how many mines to set

enum Difficulty {EASY, NORMAL, HARD};

// Flags representing various square possibilities

enum Squares {EMPTY=10, MINE, CLEAR, LOSER};

// number of mines

int mines;

void create(int, int);

//Destroy

Minesweeper::Difficulty intToDiff(char);

bool isValidIn() const;

int nMines(Minesweeper::Difficulty) const;

void setMines();

void setSquares();

int nAdjacent(int, int, int = Minesweeper::MINE) const;

bool isClear(int, int) const;

void setPerim();

void showZeros(int, int);

bool hasWon() const;

bool cont(int, int);

void prompt ();

char \*userName();

public:

//constructors

Minesweeper(int row, int col):GameBoard(row, col)

{clear();}

// Main Game Functions

void setRows(int);

void setCols(int);

int getRows() const {return rows;}

int getCols() const {return cols;}

void print() const;

void prntObscr() const;

void setUpG();//set up the game

void playGame();

void clear();

void saveGame();

void rules();

void loadGame();

int getMines() const { return mines;}

Minesweeper& operator=(const Minesweeper&);

};

#endif /\* MINESWEEPER\_H \*/

GameBoard.h

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include "Abstracts.h"

// Base class for the games

class GameBoard: public Abstracts{

private:

protected:

int \*\*data;

int rows;

int cols;

virtual void create(int, int);

public:

//If user tries to set negative rows or columns

class wrong{};

GameBoard(int rows, int cols) {create(rows,cols);clear();}

virtual ~GameBoard(){destroy();}

virtual void destroy();

virtual void setRows(int);

virtual void setCols(int);

virtual int getRows() const {return rows;}

virtual int getCols() const {return cols;}

virtual void clear();

//setup the game

virtual void setUpG();

//load the game

virtual void loadGame();

//Print

virtual void print() const;

int\* operator[](int index) { return data[index];}

int\* operator[](int index) const { return data[index];}

};

#endif /\* GAMEBOARD\_H \*/

Abstracts.h

#ifndef ABSTRACTS\_H

#define ABSTRACTS\_H

class Abstracts {

protected:

virtual void setRows(int)=0;

virtual void setCols(int)=0;

virtual int getRows() const =0;

virtual int getCols() const =0;

virtual void setUpG()=0;

virtual void print() const = 0;

};

#endif /\* ABSTRACTS\_H \*/