# MineSweeper

Project 1

Lei wan

11-1-2015

Content

1. Introduce
   1. Rules
2. Development
   1. Difficult points
   2. Call function
   3. Game function
3. Pseudo Code
4. Flowchart
5. Code

**1. Introduction**

**1.1 Rules**

Minesweeper is a classic game in Window. The rule is quiet simple. Players need find all the mines in the board.

**2. Development**

**2.1 Difficult points**

Because we are learning linklist, So the game I code the board by linklist. On the same time, I hope my linklist can imply function like 2D array, I used points to connect each Node to the up, down, left and right four Nodes. It help me simplify the method to find the numbers of the mines around each Node.

**2.2 Key points**

My code has a head file. It is a template class. I use linkList table to imply the board. All the operation about the board are realized here. In this class, the function open(Node\*,int x,int y) continuous open many nodes, if there are surround by non-mine nodes. It is a recuision.

In cpp file, there are three levels for choice. I used a few STL, Vector, set, map to deal with history record, and I used iterator, quicksort to find the data and sort them. I save and load the best 10 score by fstream library. Certainly, different difficult level saved in different file to save.

Wrong input controls were used in the whole grogram.

**2.3 Game function**

1. Player choose the difficult level.

2. Player input row, and column then choose to fill in non-mine or flag(assume mine)

3. Once all the mines are found, player win the game, and the time spent will be compared to history record. If the time spent is less than any time of the history table. The player’s time will be insert to proper place and the last record will be delete. Then, Console will require player input name and save the new table to file.

**3.Pseudo Code**

Do{

Let user to choose the board size

}while(invalid)

Save choice in variable input

Create a ChessBoard(class) by input

If(input==1){

Open file(“easy.txt”)

}else if(input==2)

Open file(“normal.txt”)

}else

Open file(“hard.txt”)

}

Insert in set<int>

Cout<<the first element;

Cout<<the last element;

Do

{

Do{

Player input

}while(invalid)

}while(ChessBoard.winOrLoose()!=0)

If(winOrLoose==1)

Cout<<win;

Else

Cout<<loose;

Record name and time

Compare with these data with data from file.

Print out by time or name sort

Head file

constructor create a four pointers node ChessBoard

fill mines

calculate the number of coordinate position’s surrounded mines

if no mine surrounded the position, fill 0;

if(the first step)

while(coordinate position (x,y)is mine or is a number){

clear the board;

fill mines;

calculate the numbers of mines and fill in;

}

Call recursion function open(), open as many as possible position.

}

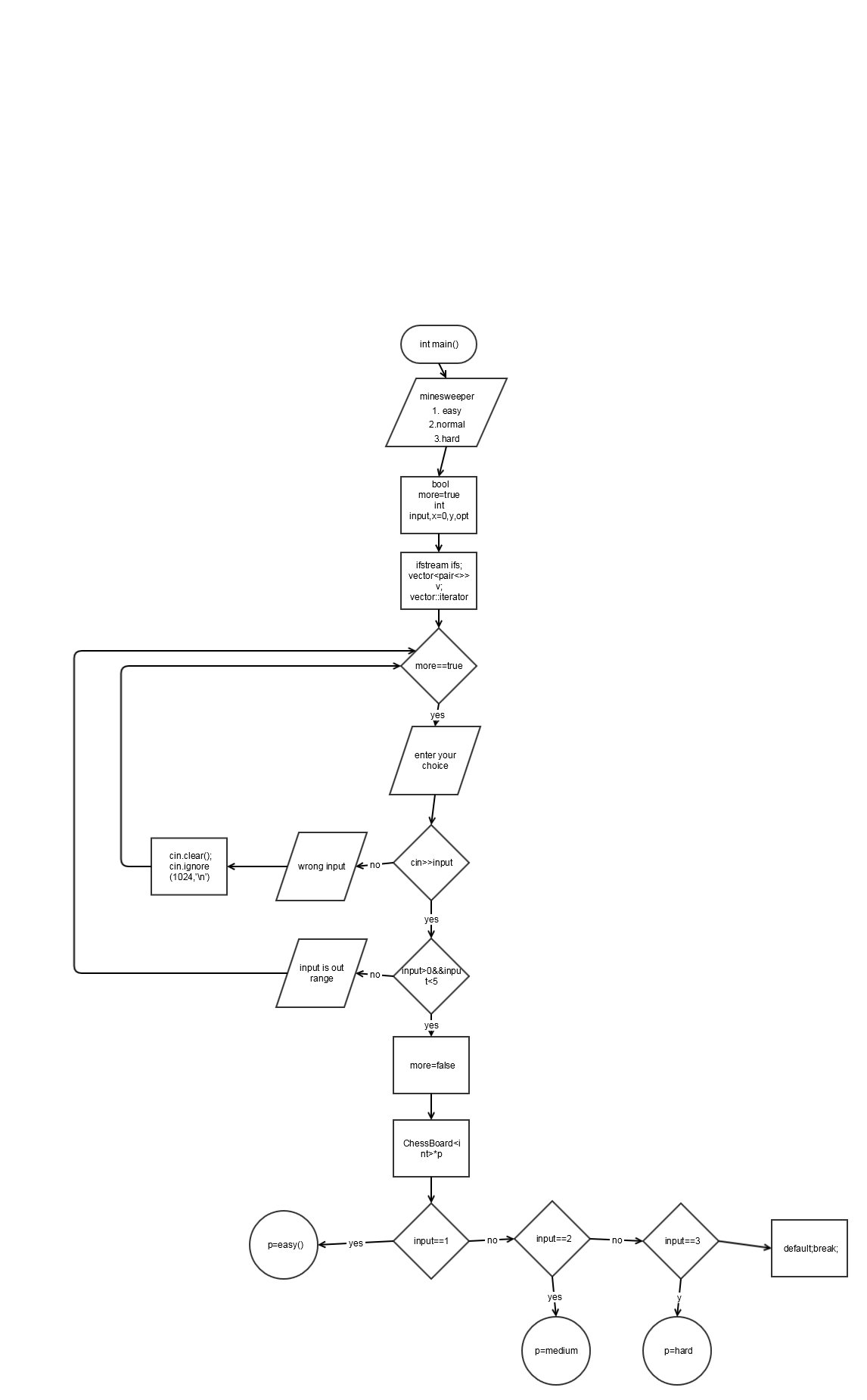
If(determine a mine position as no-mines solution)

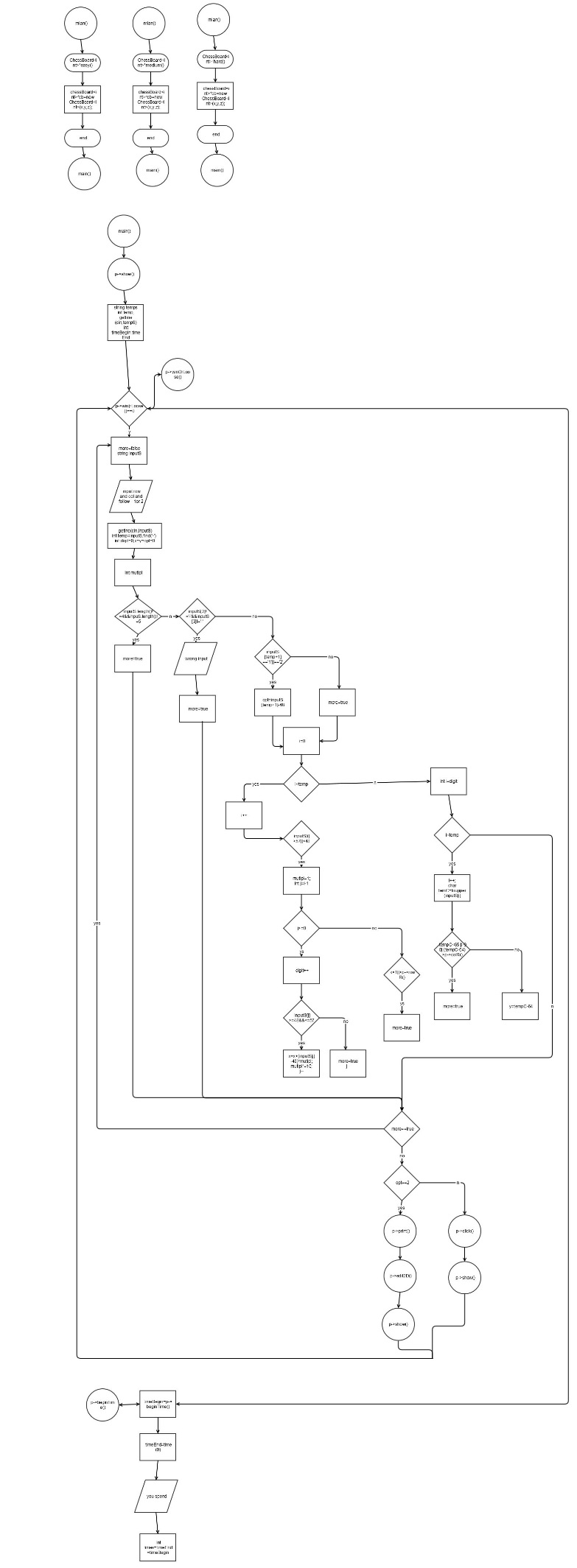
Return loose

If(all the no-mines positions are founded)

Return win

**4.Flowchart**





**5.Code**

#ifndef NEWFILE\_H

#define NEWFILE\_H

#include <cstdlib>

#include<iomanip>

#include<iostream>

using namespace std;

/\* -1 means mine, 0 means blank

\*/

template<class T>

class ChessBoard {

private:

struct Node {//background board

T val;

Node\* down;

Node\* right;

Node\* up;

Node\* left;

Node(const T& c = 0, Node\*p = 0, Node\*p1 = 0,

Node\*p2 = 0, Node\*p3 = 0) : val(c), down(p),

right(p1), left(p2), up(p3) {

}

};

Node\* head;

Node\* rear;

Node\* curr;

bool firstStep;

int noMines; //非雷数

int mines;

int boardSize;

int row;

int col;

int timeBegin;

int win; //1 为赢，-1 为输

char\*\*p; //player board

public:

ChessBoard(const int&, const int&, const int&);

~ChessBoard();

void fillMine(int); //fill mines

void fillNumber(); //fill numbers

int &operator[](const int&);

void click(int x, int y);

void open(Node\*, int x, int y); //recuision

void print(); //display the real board

void show(); //display the board player can see

void build(); //build fake board with '?'

void clearBoard() ;//clear the board

inline int beginTime(){

return timeBegin;

}

inline void edit2D(int x, int y) {

p[x - 1][y - 1] = '\*';

}

inline int winOrLoose() {

return win;

} //return win or lose;

inline int rowR() {

return row;

}

inline int colR() {

return col;

}

};

template<class T>

ChessBoard<T>::ChessBoard(const int& row, const int& col, const int&n) { //skip list

srand(time(0));

Node \* p[col]; // pointer point to upper row ,

noMines = 0; //the place player walk and the sametime no mines

firstStep = true; //the first step,control open

boardSize = row\*col;

mines = n;

win = 0;

timeBegin=0;

this->row = row;

this->col = col;

p[0] = head = rear = new Node();

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

rear->right = new Node();

rear->right->left = rear;

rear = rear->right;

if (i >= col) {//using the pointer to connected with the upper row node

//begin to connected up and down points from the seconde row

p[i % col]->down = rear;

p[i % col]->down->up = p[i % col];

}

p[i % col] = rear; //update the point to the rear point

}

/\*cut off the right side double pointer;

\*/

curr = head;

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

Node\*temp;

if (i % col == 0) {

temp = curr->right;

curr->right->left = 0;

curr->right = 0;

curr = temp;

} else

curr = curr->right;

}

build();

fillMine(mines);

fillNumber();

}

/\*clear the board

\*/

template<class T>

void ChessBoard<T>::clearBoard(){

Node\*temp = curr = head;

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

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

curr->val = 0;

curr = curr->right;

}

curr = temp->down;

temp=curr;

}

}

//return the char on the val of the chessboard

template<class T>

int &ChessBoard<T>::operator[](const int&n) {

curr = head;

int r = n / col;

while (r > 0) {

curr = curr->down;

r--;

}

int c = n % col;

while (c > 0) {

curr = curr->right;

c--;

}

return curr->val;

}

template<class T>

void ChessBoard<T>::build() {

p = new char\*[row];

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

p[i] = new char[col];

}

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

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

p[i][j] = '?';

}

}

}

//fill mine to chessboard

template<class T>

void ChessBoard<T>::fillMine(int n) {

int x;

int y;

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

do {

x = rand() % row;

y = rand() % col;

} while (operator[](x \* col + y) == -1);

operator[](x \* col + y) = -1;

}

}

template<class T>

void ChessBoard<T>::fillNumber() {

int count;

Node\*temp = head;

curr = head;

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

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

count = 0;

if (curr->val != -1) {//节点判断

if (curr->up != 0) {//up;

curr->up->val == -1 ? count++ : count += 0;

if (curr->up->left != 0) {//up-left

curr->up->left->val == -1 ? count++ : count += 0;

}

if (curr->up->right != 0) {//up-right

curr->up->right->val == -1 ? count++ : count += 0;

}

}

if (curr->down != 0) {//down

curr->down->val == -1 ? count++ : count += 0;

if (curr->down->left != 0) {//down-left

curr->down->left->val == -1 ? count++ : count += 0;

}

if (curr->down->right != 0) {//down-right

curr->down->right->val == -1 ? count++ : count += 0;

}

}

if (curr->left != 0) {//left

curr->left->val == -1 ? count++ : count += 0;

}

if (curr->right != 0) {//right

curr->right->val == -1 ? count++ : count += 0;

}

curr->val = count;

curr = curr->right;

} else curr = curr->right;

}

curr = temp->down; //right edge point has cut off, so depend on down pointer

temp = curr;

}

}

template<class T>

ChessBoard<T>::~ChessBoard() {

Node\*temp = head->down;

int r = row;

while (r > 1) {//can not be 0,because temp will point to a invalid allocation

while (head != 0) {

curr = head;

head = head->right;

delete curr;

}

head = temp;

temp = head->down;

r--;

}

while (head != 0) {

curr = head;

head = head->right;

delete curr;

}

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

delete []p[i];

}

delete [] p;

}

/\* '/' mean in the recursion the place had been passed and no number there before

\*/

template<class T>

void ChessBoard<T>::open(Node\* now, int x, int y) {//recurseion for zero mine allocation open

// cout<<"noMines is "<<noMines<<endl;

curr = now;

Node\*temp = now;

if (curr->up != 0) {

if (curr->up->val != 0) {

if (p[x - 1][y] != curr->up->val + 48) {

p[x - 1][y] = curr->up->val + 48;

noMines++;

}

} else {

if (p[x - 1][y] != '/') {

p[x - 1][y] = '/';

noMines++;

open(curr->up, x - 1, y);

curr = temp;

}

}

if (curr->up != 0 && curr->up->left != 0) {

if (curr->up->left->val != 0) {

if (p[x - 1][y - 1] != curr->up->left->val + 48) {

p[x - 1][y - 1] = curr->up->left->val + 48;

noMines++;

}

} else {

if (p[x - 1][y - 1] != '/') {

p[x - 1][y - 1] = '/';

noMines++;

open(curr->up->left, x - 1, y - 1);

curr = temp;

}

}

}

if (curr->up != 0 && curr->up->right != 0) {

if (curr->up->right->val != 0) {

if (p[x - 1][y + 1] != curr->up->right->val + 48) {

p[x - 1][y + 1] = curr->up->right->val + 48;

noMines++;

}

} else {

if (p[x - 1][y + 1] != '/') {

p[x - 1][y + 1] = '/';

noMines++;

open(curr->up->right, x - 1, y + 1);

curr = temp;

}

}

}

}

if (curr->left != 0) {

if (curr->left->val != 0) {

if (p[x ][y - 1] != curr->left->val + 48) {

p[x ][y - 1] = curr->left->val + 48;

noMines++;

}

} else {

if (p[x ][y - 1] != '/') {

p[x ][y - 1] = '/';

noMines++;

open(curr->left, x, y - 1);

curr = temp;

}

}

}

if (curr->right != 0) {

if (curr->right->val != 0) {

if (p[x][y + 1] != curr->right->val + 48) {

p[x][y + 1] = curr->right->val + 48;

noMines++;

}

} else {

if (p[x ][y + 1] != '/') {

p[x ][y + 1] = '/';

noMines++;

open(curr->right, x, y + 1);

curr = temp;

}

}

}

if (curr->down != 0) {

if (curr->down->val != 0) {

if (p[x + 1][y ] != curr->down->val + 48) {

p[x + 1][y ] = curr->down->val + 48;

noMines++;

}

} else {

if (p[x + 1][y] != '/') {

p[x + 1][y] = '/';

noMines++;

open(curr->down, x + 1, y);

curr = temp;

}

}

if (curr->down != 0 && curr->down->left != 0) {

if (curr->down->left->val != 0) {

if (p[x + 1][y - 1] != curr->down->left->val + 48) {

p[x + 1][y - 1] = curr->down->left->val + 48;

noMines++;

}

} else {

if (p[x + 1][y - 1] != '/') {

p[x + 1][y - 1] = '/';

noMines++;

open(curr->down->left, x + 1, y - 1);

curr = temp;

}

}

}

if (curr->down != 0 && curr->down->right != 0) {

if (curr->down->right->val != 0) {

if (p[x + 1][y + 1] != curr->down->right->val + 48) {

p[x + 1][y + 1] = curr->down->right->val + 48;

noMines++;

}

} else {

if (p[x + 1][y + 1] != '/') {

p[x + 1][y + 1] = '/';

noMines++;

open(curr->down->right, x + 1, y + 1);

curr = temp;

}

}

}

}

}

template<class T>

void ChessBoard<T>::click(int x, int y) {

cout << endl;

curr = head;

int x\_temp = x;

int y\_temp = y;

while (x > 1) {

curr = curr->down;

x--;

}

while (y > 1) {

curr = curr->right;

y--;

}

while (firstStep && curr->val != 0) {//if the first step is not empty,rand()mines again)

Node\*temp=curr;

clearBoard();//before refill mines, clear all the numbers of the board first

// cout<<"mines number is "<<mines<<endl;

fillMine(mines);

fillNumber();

curr=temp;

timeBegin=time(0);

// cout<<"after is "<<curr->val<<endl;

}

if (curr->val == -1) {

p[x\_temp - 1][y\_temp - 1] = 'M';

win = -1;

//下一步需要考虑递归中开出的不是雷的格子数目

} else if (curr->val == 0) {//

timeBegin=time(0);

p[x\_temp - 1][y\_temp - 1] = '/';

noMines++;

open(curr, x\_temp - 1, y\_temp - 1);

} else {

p[x\_temp - 1][y\_temp - 1] = curr->val + 48;

noMines++;

}

if (noMines == row \* col - mines) {

win = 1;

}

// cout<<"noMinse number is "<<noMines<<endl;

firstStep = false;

}

template<class T>

void ChessBoard<T>::show() {

cout << endl;

cout << " ";

for (int k = 0; k < col; k++) {

cout << right << setw(3) << char(k + 65);

}

cout << endl;

for (int k = 0; k < 3 \* col + 3; k++) {

cout << "\_";

}

cout << endl;

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

cout << right << setw(2) << i + 1 << "|";

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

if (p[i][j] == '/')

p[i][j] = ' ';

cout << right << setw(2) << p[i][j] << "|";

}

cout << left << setw(2) << i + 1;

cout << endl;

}

for (int k = 0; k < 3 \* col + 3; k++) {

cout << "-";

}

cout << endl;

cout << " ";

for (int k = 0; k < col; k++) {

cout << right << setw(3) << char(k + 65);

}

}

template<class T>

void ChessBoard<T>::print() {

for (int k = 0; k < 3 \* col; k++) {

cout << "\_";

}

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

if (i % col == 0) {

cout << endl << "|";

}

cout << setw(2) << operator[](i) << "|";

}

cout << endl;

for (int k = 0; k < 3 \* col; k++) {

cout << "-";

}

}

#endif /\* NEWFILE\_H \*/

#include <bits/stdc++.h>

#include "board.h"

using namespace std;

ChessBoard<int>\* easy(int, int, int);

ChessBoard<int>\* medium(int, int, int);

ChessBoard<int>\* hard(int, int, int);

void quickSort(vector<pair<string, int> >&, int, int);

/\*

\*

\*/

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

cout << "Minesweeper" << endl;

cout << "1. Easy (8\*8, 10)" << endl;

cout << "2. Normal (16\*16, 40)" << endl;

cout << "3. Hard (30\*16, 99)" << endl;

bool more = true;

int input;

int x = 0; //row

int y; //column

int opt; //flag or not;

ifstream ifs;

vector<pair<string, int> >v; //temply store the data from file

vector<pair<string, int> >::iterator iter;

set<int>s;

map<string, int> m;

while (more) {

cout << "enter your choice (1-4)" << endl;

if (cin >> input) {

if (input > 0 && input < 5) {

more = false;

} else cout << "input is out range" << endl;

} else {

cout << "wrong input" << endl;

cin.clear();

cin.ignore(1024, '\n');

}

}

ChessBoard<int>\* p;

switch (input) {

case 1: p = easy(8, 8, 10);

break;

case 2: p = medium(16, 16, 40);

break;

case 3: p = hard(16, 30, 99);

break;

default: break;

}

string tempS;

int temp;

//add STL set to the code by requirement

if (input == 1) {

ifs.open("easy.txt");

} else if (input == 2) {

ifs.open("normal.txt");

} else {

ifs.open("hard.txt");

}

try {

if (ifs) {

while (!ifs.eof()) {

ifs>>tempS;

ifs>>temp;

s.insert(temp);

// v.push\_back(make\_pair(tempS, temp));

}

} else throw "file missed";

} catch (const char\* e) {

cout << e << endl;

}

set<int>::iterator itrS;

itrS=s.begin();

cout <<"The best record is "<<\*itrS<<"s"<<endl;

itrS=s.end();

itrS--;

cout<<"The worst record in the table is "<<\*itrS<<"s"<<endl;

ifs.close();

p->show();

// string tempS;

// int temp;

getline(cin, tempS);

//loop realize the process of sweeping

int timeBegin, timeEnd; //recorder the time;

while (p->winOrLoose() == 0) {

do {

more = false;

cout << endl << "input the row(number) and column(character), then follow by \_1 or 2,"

"1 is non-mine place , 2 is estimated mine flag"

"example (2B-2), put a mine flag at row 2,column B" << endl;

string inputS;

getline(cin, inputS);

int temp = inputS.find('-');

int digit = 0;

x = y = opt = 0;

int mutipl;

if (inputS.length() != 4 && inputS.length() != 5) {

more = true;

} else if (inputS[2] != '-' && inputS[3] != '-') {

cout << "wrong input" << endl;

more = true;

} else {

if (inputS[temp + 1] == '1' ||

inputS[temp + 1] == '2') {

opt = inputS[temp + 1] - 48;

} else {

more = true;

}

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

if (inputS[i] > 57 || inputS[i] < 48) {//is not 0-9

mutipl = 1;

for (int j = i - 1; j >= 0; j--) {

digit++;

if (inputS[j] >= 48 && inputS[j] <= 57) {//0-9 char-int

// cout << "inputS[j[+48 is " << inputS[j] - 48 << endl;

x = x + (inputS[j] - 48) \* mutipl;

// cout << "x in process is " << x << endl;

mutipl \*= 10;

} else {//out of 0-9

more = true;

}

}

// cout << "final x is " << x << endl;

if (x < 1 || x > p->rowR()) {//out of 1-row

// cout<<"final x is "<<x<<endl;

more = true;

// cout << 3 << endl;

}

}

}

for (int i = digit; i < temp; i++) {

char tempC = toupper(inputS[i]);

if (tempC < 65 || tempC > 90 || (tempC - 64) > p->colR()) {

// cout << "wrong place and the value is " << tempC << endl;

more = true;

} else {

y = tempC - 64;

}

}

if (more == true) {

// cout << "wrong input" << endl;

}

}

} while (more);

// cout << "x is " << x << " : y is " << y << endl;

if (opt == 2) {

p->print();

p->edit2D(x, y);

// cout << "test" << endl;

p->show();

} else {

// p->print();

p->click(x, y);

// p->print();

p->show();

}

}

timeBegin = p->beginTime();

timeEnd = time(0);

cout << endl;

cout << "You spent " << timeEnd - timeBegin << "s" << endl;

int times = timeEnd - timeBegin;

if (p->winOrLoose() == 1) {

cout << "Congratuation! You win" << endl;

if (input == 1) {

ifs.open("easy.txt");

} else if (input == 2) {

ifs.open("normal.txt");

} else {

ifs.open("hard.txt");

}

try {

if (ifs) {

while (!ifs.eof()) {

ifs>>tempS;

ifs>>temp;

v.push\_back(make\_pair(tempS, temp));

}

} else throw "file missed";

} catch (const char\* e) {

cout << e << endl;

}

ifs.close();

bool firstTen = false;

for (int i = 0; i < v.size(); i++) {

if (times < v[i].second) {//find the time allocation in the vector

cout << "You are the " << i + 1 << "rd" << endl;

cout << "please input your name" << endl;

cin>>tempS;

firstTen = true; //in first 10 rd

v.push\_back(make\_pair(tempS, times));

break;

}

}

if (firstTen) {

quickSort(v, 0, v.size() - 1);

cout << right << setw(17) << "Sheet" << endl;

cout << setw(10) << left << "Name" << right << setw(15) << "Time" << endl;

for (int i = 0; i < v.size() && i < 10; i++) {//only the first ten name and time need to be print out

cout << left << setw(10) << v[i].first << setw(15) << right << v[i].second << endl;

}

ofstream ofs; //after insert new data, the sheet in the file need to be edited

if (input == 1) {

ofs.open("easy.txt");

} else if (input == 2) {

ofs.open("normal.txt");

} else {

ofs.open("hard.txt");

}

for (int i = 0; i < v.size() - 1 && i < 9; i++) {//only save the first 10 score;

ofs << v[i].first << endl;

ofs << v[i].second << endl;

}

ofs << v[9].first << endl << v[9].second;

ofs.close();

} else {//mot insert any data to vector, so do not need edit the file

cout << right << setw(17) << "Hero Sheet" << endl;

cout << setw(10) << left << "Name" << right << setw(15) << "Time" << endl;

for (iter = v.begin(); iter != v.end(); iter++) {

cout << left << setw(10) << iter->first << setw(15) << right << iter->second << endl;

}

}

}

if (p->winOrLoose() == -1) {

cout << "You loose" << endl;

if (input == 1) {

ifs.open("easy.txt");

} else if (input == 2) {

ifs.open("normal.txt");

} else {

ifs.open("hard.txt");

}

try {

if (ifs) {

while (!ifs.eof()) {

ifs>>tempS;

ifs>>temp;

v.push\_back(make\_pair(tempS, temp));

m.insert(pair<string,int>(tempS,temp));

}

} else throw "file missed";

} catch (const char\* e) {

cout << e << endl;

}

ifs.close();

cout << right << setw(17) << "Hero Sheet by Time" << endl;

cout << setw(10) << left << "Name" << right << setw(15) << "Time(s)" << endl;

for (iter = v.begin(); iter != v.end(); iter++) {

cout << left << setw(10) << iter->first << setw(15) << right << iter->second << endl;

}

cout << right << setw(17) << "Hero Sheet by Name" << endl;

map<string ,int>::iterator itrM;

cout << setw(10) << left << "Name" << right << setw(15) << "Time(s)" << endl;

for (itrM = m.begin(); itrM != m.end(); itrM++) {

cout << left << setw(10) << itrM->first << setw(15) << right << itrM->second << endl;

}

}

return 0;

}

ChessBoard<int>\* easy(int x, int y, int z) {

ChessBoard<int> \*cb = new ChessBoard<int>(x, y, z);

return cb;

}

ChessBoard<int>\* medium(int x, int y, int z) {

ChessBoard<int> \*cb = new ChessBoard<int>(x, y, z);

return cb;

}

ChessBoard<int>\* hard(int x, int y, int z) {

ChessBoard<int>\* cb = new ChessBoard<int>(x, y, z);

return cb;

}

void quickSort(vector<pair<string, int> >&v, int l, int r) {

if (l < r) {

int i = l;

int j = r;

vector<pair<string, int> >temp(1);

temp[0] = v[l];

while (i < j) {

while (i < j && v[j].second >= temp[0].second) {

j--;

}

if (i < j) {

v[i++] = v[j];

}

while (i < j && v[i].second < temp[0].second) {

i++;

}

if (i < j) {

v[j--] = v[i];

}

}

v[i] = temp[0];

quickSort(v, 0, i - 1);

quickSort(v, i + 1, r);

}

}