Санкт-Петербургский Политехнический Государственный Университет

факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

## Курсовая работа по курсу «Программирование»

## Тема работы: «Разработка приложения с графическим интерфейсом на языке C++ с использованием библиотеки QT»

Работу выполнил:

студентка 1 курса

группы 1081/3

Назарова К. Е.

Работу принял:

Пышкин Е. В.

Санкт-Петербург 2012

## Образ и границы проекта

Целью данного проекта является разработка проекта с графическим интерфейсом на языке C++ на примере игры «*Sunny Seeds*».

Суть игры заключается в том, чтобы очистить поле от всех чисел.

Правила игры:

1. На входе таблица случайных чисел от 1 до 9;
2. Требуется найти пары одинаковых чисел или чисел, сумма которых равна 10, в случае чего они исчезнут;
3. Числа, которые можно удалить, должны располагаться на одном уровне по горизонтали или вертикали, при этом между ними не должно быть других чисел;
4. При необходимости можно добавить новую группу чисел, состоящую из оставшихся на поле цифр (это дает гарантию выигрыша, но для того, чтобы игра не была бесконечной, доступное количество добавлений будет ограничено).

## Техническое задание

Разработать приложение на языке C++ с использованием библиотеки Qt для реализации графического интерфейса в двух режимах:

1. Консольное приложение, предназначенное для тестирования и проверки правильности работы приложения;
2. Интеграция кода программы для консоли с графической библиотекой Qt, и создание графического интерфейса.

Основные требования

Меню приложения:

1. Новая игра;
2. Продолжить сохраненную игру;
3. Выход.

Сценарии выполнения программы

1. Новая игра

При выборе этого пункта перед пользователем появляется игровое поле;

1. Продолжить сохраненную игру

При нажатии на эту кнопку меню загрузится последняя сохраненная игра (данные хранятся в текстовом формате);

1. Выход

Выход из приложения. При этом все данные незаконченной игры будут потеряны.

Требования к графическому интерфейсу

1. На входе стартовое меню в виде кнопок;
2. Удаление пар чисел и выбор других действий производится посредством нажатия левой кнопки мыши.

Системная архитектура

В программе предполагается использование следующих классов:

1. Игровое поле. Этот класс создает матрицу, размер которой зависит от выбранного разработчиком числа. По мере необходимости добавляет ячейки с требуемыми значениями.
2. Строка. Этот класс хранит в себе строку из 7 элементов, которая заполняется случайными числами.
3. Проверка хода. С помощью методов этого класса в ходе игры будут проверяться все ходы, сделанные игроком, а так же проверяется конец игры.
4. Тест. Класс, тестирующий некоторые функции других классов.
5. Абстрактные классы графического интерфейса.

Тестирование работы приложения.

Файл test.h

#ifndef TEST\_H

#define TEST\_H

#include "field.h"

class Test{

public:

Test();

bool nearSameNumber();

bool emptyOrFilledCell();

bool sameNumThroughEmptyCell();

};

#endif // TEST\_H

Файл test.cpp

#include "test.h"

Test::Test()

{

if (nearSameNumber())

cout << "Test #1 passed."<<endl;

else cout << "Test #1 didn't pass."<<endl;

cout<<endl;

if (emptyOrFilledCell())

cout << "Test #2 passed."<<endl;

else cout << "Test #2 didn't pass."<<endl;

cout<<endl;

if (sameNumThroughEmptyCell())

cout << "Test #3 passed."<<endl;

else cout << "Test #3 didn't pass."<<endl;

cout<<endl;

}

bool Test::nearSameNumber(){

cout<<"Test #1. The choice of identical adjacent numbers."<<endl;

Field f(3);

cout<<f<<endl;

f.field[0].setData(0, 1);

f.field[0].setData(1, 1);

cout<<f<<endl;

return f.deleteCell(1, 1, 1, 2);

}

bool Test::emptyOrFilledCell(){

cout<<"Test #2. The choice of empty and filled cell."<<endl;

Field c(3);

cout<<c<<endl;

c.field[0].setData(0, 0);

c.field[0].setData(1, 1);

cout<<c<<endl;

return c.deleteCell(1, 1, 1, 2);

}

bool Test::sameNumThroughEmptyCell(){

cout<<"Test #3. The choice of identical numbers, among which is an empty cell."<<endl;

Field f(3);

cout<<f<<endl;

f.field[2].setData(4, 1);

f.field[1].setData(4, 0);

f.field[0].setData(4, 1);

cout<<f<<endl;

return f.deleteCell(3, 5, 1, 5);

}

Первый тест заключается в том, чтобы проверить рядом стоящие одинаковые числа. Создается поле 3х7, в котором в ячейках с координатами [0, 0] и [0, 1] стоят единицы. Выбирается эта пара цифр, затем удаляется. Следовательно, тест пройден, так как соблюдены все правила.

Второй тест заключается в том, чтобы проверить рядом стоящие не одинаковые числа. Создается поле 3х7, в котором ячейках с координатами [0, 0] пустая, а в ячейке [0, 1] стоит единица. Выбирается эта пара ячеек, затем выходит сообщение, что тест не пройден, так как выбраны неверные значения.

Суть третьего теста в том, чтобы проверить одинаковые числа, между которыми стоят пустые ячейки. Создается поле 3х7, в котором в ячейках с координатами [2, 4] и [0, 4] стоят единицы, а ячейка с координатами [1, 4] пустая. Выбирается пара, затем удаляется. Следовательно, тест пройден, так как соблюдены все правила.

## Текст программы

main.cpp

#include <QtGui>

#include "mainwindow.h"

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

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("Sunny Seeds");

w.show();

return a.exec();

}

field.h

#ifndef FIELD\_H

#define FIELD\_H

#include "row.h"

#include <vector>

class Field{

int rows;

bool saveField;

public:

Field();

Field(int a);

~Field();

vector<Row>field;

bool deleteCell(int, int, int, int);

void deleteEmptyRow();

void addCells();

void getState(bool);

void saveGame();

void loadGame();

int getRows();

};

#endif // FIELD\_H

row.h

#ifndef ROW\_H

#define ROW\_H

#include <time.h>

#include <vector>

using namespace std;

const int maxColumns=7;

const int maxRows=35;

class Exception {};

class Row{

vector<int> arr;

vector<bool> isPressed;

public:

Row();

int getData(int);

void setData(int number, int value);

void clearCell(int a);

void setStateIsPressed(int, bool);

bool getStateIsPressed(int);

};

#endif // ROW\_H

check.h

#ifndef CHECK\_H

#define CHECK\_H

#include "field.h"

class Check{

Field f;

bool checkSameValues(int, int, int, int);

bool checkSumValues(int, int, int, int);

bool checkPlace(int, int, int, int);

bool nullValue(int, int, int, int);

bool checkSamePlace(int, int, int, int);

bool checkLoss();

bool checkWin();

public:

Check();

Check(Field&);

bool checkMove(int, int, int, int);

bool getLoss();

bool getWin();

};

#endif // CHECK\_H

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QtGui>

#include <QMainWindow>

#include <QAbstractScrollArea>

#include "field.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

protected:

bool firstClick;

int startX, startY, finishX, finishY;

virtual void *paintEvent*(QPaintEvent \*);

void *mousePressEvent*(QMouseEvent \*event);

void leftButtonPressEvent(QMouseEvent\* event);

void *closeEvent*(QCloseEvent \*event);

bool outLoss();

bool outWin();

private slots:

void on\_actionNew\_game\_activated();

void on\_pushButton\_clicked();

void on\_actionExit\_activated();

void on\_verticalScrollBar\_valueChanged(int value);

void on\_actionContinued\_activated();

private:

Ui::MainWindow \*ui;

Field f;

int myVar;

};

#endif // MAINWINDOW\_H

field.cpp

#include "field.h"

#include "check.h"

#include "mainwindow.h"

#include "QtGui"

#include <stdlib.h>

#include <fstream>

Field::Field(): field() {

saveField=false;

rows=5;

srand((int)time(NULL));

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

field.push\_back(Row());

}

Field::Field(int a): field(){

saveField=false;

rows=a;

srand((int)time(NULL));

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

field.push\_back(Row());

}

Field::~Field() {}

bool Field::deleteCell(int a1, int b1, int a2, int b2){

Field newField;

newField.field=field;

Check check(newField);

if (check.checkMove(a1, b1, a2, b2)){

field[--a1].clearCell(b1);

field[--a2].clearCell(b2);

return true;

}

else return false;

}

void Field::addCells(){

vector<Row>f;

f.resize(rows);

int i\_new = 0;

int j\_new = 0;

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

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

if (field[i].getData(j)!=0){

if (j\_new >= maxColumns){

i\_new++;

j\_new = 0;

}

f[i\_new].setData(j\_new, field[i].getData(j) );

++j\_new;

}

}

j\_new++;

for (j\_new; j\_new<=maxColumns; j\_new++){

f[i\_new].clearCell(j\_new++);

j\_new--;

}

for (unsigned int i=0; i<=i\_new; i++)

field.push\_back(f[i]);

rows=field.size();

}

void Field::getState(bool state){

saveField=state;

}

void Field::saveGame(){

ofstream out("Save.txt");

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

for (int j=0; j<maxColumns; j++)

out<<field[i].getData(j)<<" ";

out<<endl;

}

out.close();

}

void Field::loadGame(){

vector<Row>f;

int amt=0;

int\* arr=new int[maxRows\*maxColumns];

ifstream in("Save.txt");

for (int i=0; !in.eof(); i++){

in>>arr[i];

amt++;

}

f.resize((amt+1)/maxColumns);

int k=0;

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

for (int j=0; j<maxColumns; j++)

if(k<amt){

f[i].setData(j, arr[k]);

k++;

}

field=f;

in.close();

}

int Field::getRows(){

return rows;

}

void Field::deleteEmptyRow(){

for (vector<Row>::iterator it=field.begin(); it!=field.end(); it++){

bool isEmpty = true;

for (int i=0; i<maxColumns; i++)

if (it->getData(i) != 0){

isEmpty = false;

break;

}

if (isEmpty)

field.erase(it);

}

}

row.cpp

#include "row.h"

#include <stdlib.h>

Row::Row(){

arr.resize(maxColumns);

isPressed.resize(maxColumns);

for(int i=0; i<maxColumns; i++)

{

arr[i]=1+rand()%9;

isPressed[i] = false;

}

}

int Row::getData(int a){

if (a<=maxColumns)

return arr[a];

else throw Exception();

}

void Row::clearCell(int a){

if (a<=maxColumns)

arr[--a]=0;

else throw Exception();

}

void Row::setStateIsPressed(int i, bool j){

isPressed[i]=j;

}

bool Row::getStateIsPressed(int i){

return isPressed[i];

}

void Row::setData(int number, int value){

if (number<=maxColumns)

arr[number] = value;

else throw Exception();

}

check.cpp

#include "check.h"

#include "cstdlib"

Check::Check() {}

Check::Check(Field&field){

f=field;

}

bool Check::checkMove(int a1, int b1, int a2, int b2){

if ((checkSameValues(a1, b1, a2, b2) || checkSumValues(a1, b1, a2, b2)) && checkPlace(a1, b1, a2, b2) && nullValue(a1, b1, a2, b2)

&& (checkLoss()==false) && (checkSamePlace(a1, b1, a2, b2)) && (checkWin()==false))

return true;

else return false;

}

bool Check::getLoss(){

return (checkLoss());

}

bool Check::getWin(){

return (checkWin());

}

bool Check::checkSameValues(int a1, int b1, int a2, int b2){

if (f.field[--a1].getData(--b1)==f.field[--a2].getData(--b2))

return true;

else return false;

}

bool Check::checkSumValues(int a1, int b1, int a2, int b2){

if (f.field[--a1].getData(--b1)+f.field[--a2].getData(--b2)==10)

return true;

else return false;

}

bool Check::checkPlace(int a1, int b1, int a2, int b2){

if (a1==a2 || b1==b2){

if (a1==a2){

for(int i=++b1; i<--b2; i++)

if (f.field[--a1].getData(--i)!=0)

return false;

}

else return true;

if (b1==b2){

for(int i=++a1; i<--a2; i++)

if (f.field[--i].getData(--b1)!=0)

return false;

}

else return true;

}

else return false;

}

bool Check::nullValue(int a1, int b1, int a2, int b2){

if (f.field[--a1].getData(--b1)==0 || f.field[--a2].getData(--b2)==0)

return false;

else return true;

}

bool Check::checkSamePlace(int a1, int b1, int a2, int b2){

if (a1==a2 && b1==b2) return false;

else return true;

}

bool Check::checkLoss(){

if(f.field.size()>=maxRows)

return true;

else return false;

}

bool Check::checkWin(){

int amt=0;

for (int i=0; i<f.field.size(); i++)

for (int j=0; j<maxColumns; j++)

if (f.field[i].getData(j)==0)amt++;

if (f.field.size()\*maxColumns==amt) return true;

else return false;

}

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "QPainter"

#include "field.h"

#include "check.h"

#include "row.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow){

ui->setupUi(this);

firstClick = true;

startX=startY=finishX=finishY=0;

myVar=0;

}

MainWindow::~*MainWindow*(){

delete ui;

}

void MainWindow::*paintEvent*(QPaintEvent \*){

int x=30, y=45;

for(int i=myVar; i<f.field.size(); i++){

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

QRect sq(x, y, 30, 30);

QPainter pict;

pict.begin(this);

pict.setRenderHint(QPainter::Antialiasing, true);

pict.setBrush(QBrush(Qt::lightGray));

pict.setPen(QPen(Qt::black));

pict.drawRoundedRect(sq, 1, 1);

pict.end();

QPainter text;

text.begin(this);

text.setRenderHint(QPainter::Antialiasing, true);

if (f.field[i].getStateIsPressed(j)==true) text.setPen(QPen(Qt::cyan, 3, Qt::DashLine));

if (f.field[i].getData(j)==0) text.drawText(sq, " ", QTextOption(Qt::AlignCenter));

else text.drawText(sq, QString::number(f.field[i].getData(j)), QTextOption(Qt::AlignCenter));

text.end();

x=x+30;

f.deleteEmptyRow();

f.field[i].setStateIsPressed(j, false);

}

x=30;

y=y+30;

}

}

void MainWindow::on\_actionNew\_game\_activated(){

if (QMessageBox::question(this, "Question", "Open a new game?",QMessageBox::Yes, QMessageBox::No)==QMessageBox::Yes){

Field field(5);

f=field;

repaint();

}

}

void MainWindow::*mousePressEvent*(QMouseEvent \*event){

if (event->button()==Qt::LeftButton) leftButtonPressEvent(event);

}

void MainWindow::leftButtonPressEvent(QMouseEvent \*event){

if(firstClick){

QPoint p;

p=event->pos();

startX=((p.ry()-45)/30)+1+myVar;

startY=((p.rx()-30)/30)+1;

f.field[startX-1].setStateIsPressed(startY-1, true);

firstClick=false;

}

else {

QPoint p;

p=event->pos();

finishX=((p.ry()-45)/30)+1+myVar;

finishY=((p.rx()-30)/30)+1;

f.field[finishX-1].setStateIsPressed(finishY-1, true);

f.deleteCell(startX,startY,finishX,finishY);

firstClick = true;

}

repaint();

}

void MainWindow::*closeEvent*(QCloseEvent \*event){

if (QMessageBox::question(this, "Question", "Save game?",QMessageBox::Yes, QMessageBox::No)==QMessageBox::Yes) {

f.saveGame();

emit exit(1);

}

else emit exit(1);

}

bool MainWindow::outLoss(){

Check check(f);

if (check.getLoss()==true){

if (QMessageBox::question(this, "You lose!!!", "Open a new game?",QMessageBox::Yes, QMessageBox::No)==QMessageBox::Yes){

Field field;

f=field;

repaint();

}

else exit(1);

return true;

}

else return false;

}

bool MainWindow::outWin(){

Check check(f);

if (check.getWin()==true){

if (QMessageBox::question(this, "You win!!!", "Open a new game?",QMessageBox::Yes, QMessageBox::No)==QMessageBox::Yes){

Field field;

f=field;

repaint();

}

else exit(1);

return true;

}

else return false;

}

void MainWindow::on\_pushButton\_clicked(){

if (outLoss()==0 && outWin()==0) emit f.addCells();

repaint();

}

void MainWindow::on\_actionExit\_activated(){

if (QMessageBox::question(this, "Question", "Save game?",QMessageBox::Yes, QMessageBox::No)==QMessageBox::Yes) {

f.saveGame();

emit exit(1);

}

else emit exit(1);

}

void MainWindow::on\_verticalScrollBar\_valueChanged(int value){

myVar=value;

repaint();

}

void MainWindow::on\_actionContinued\_activated(){

f.getState(true);

f.loadGame();

repaint();

}