Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Факультет электронно-информационных систем

Кафедра ИИТ

Лабораторная работа №1-2

за 5 семестр

По дисциплине: «ОСиСП»

Выполнила:

студентка 3 курса

группы ПО-4(1)

Воробей А.П.

Проверила:

Дряпко А.В.

2021

Лабораторная работа №1-2

Цель работы: приобрести практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt.

Вариант 4

Задание:

4)Игра «Пакман». Реализовать игру с одним уровнем и 3 врагами-привидениями. При получении бонуса герой (Пакман) «съедает» привидение. Бонус действует ограниченное время (15 секунд). Приведения движутся рандомно. При столкновении с героем, если бонус не действует, игра заканчивается.

**Текст программы:**

1. **ColorLinesBuilder.h**

/\*\* @file

\* @author Kalinovskiy.

\*/

#ifndef COLORLINESBUILDER\_H

#define COLORLINESBUILDER\_H

#include "COMMON/INTERFACES/AbstractGameBuilder.h"

class **ColorLinesBuilder** : public AbstractGameBuilder

{

public:

void ***createGame***(AbstractColorLinesGame \*&game, QWidget \*&widget) override;

QString ***getName***() const override;

QIcon ***getIcon***() const override;

};

#endif // COLORLINESBUILDER\_H

1. **GameData.h**

/\*\* @file

\* @author Kalinovskiy.

\*/

#ifndef ColorLinesGameData\_H

#define ColorLinesGameData\_H

#include <random>

#include <vector>

#include <QObject>

#include "JustAnotherLines.h"

#include "COMMON/MAP/ColorLinesTileMap.h"

enum

{

/// Количество тайлов в строке или столбце.

DIMENSION = 9,

/// Минимальная длина линии одного цвета.

MIN\_LINE = 5,

};

class **AbstractGameState**

{

public:

virtual void ***update***() = 0;

virtual void ***tileClicked***(int col, int row) = 0;

virtual ~***AbstractGameState***(){};

};

class **ElementMovingState** : public AbstractGameState

{

ColorLinesGameData \*data;

public:

explicit **ElementMovingState**(ColorLinesGameData \*data);

void ***update***();

void ***tileClicked***(int col, int row);

};

class **WaitingState** : public AbstractGameState

{

ColorLinesGameData \*data;

public:

explicit **WaitingState**(ColorLinesGameData \*data);

void ***update***();

void ***tileClicked***(int col, int row);

};

class **ColorLinesGameData**: public QObject

{

Q\_OBJECT

bool lose;

/// Последний успешно проделанный путь между тайлами.

public:

std::default\_random\_engine randomEngine;

std::vector<ColorLinesTile \*> path;

/// Коэффициент, на который умножатся очки за ход в случае еще одного

/// удачного хода подряд.

int combo;

/// Набранные очки.

int coins;

ColorLinesTileMap tileMap;

/// Выделенный тайл с фишкой.

ColorLinesTile \*selection;

QString statistics;

std::vector<ColorLinesTile \*>::iterator pathIterator;

std::unique\_ptr<ElementMovingState> elementMovingStatePointer;

std::unique\_ptr<WaitingState> waitingStatePointer;

AbstractGameState \*currentState;

NextColors nextColors;

explicit **ColorLinesGameData**(JustAnotherLines \*game);

private:

JustAnotherLines \*game;

**ColorLinesGameData**();

explicit **ColorLinesGameData**(ColorLinesGameData&);

ColorLinesGameData &operator =(const ColorLinesGameData&);

public slots:

/\*\*

\* @brief Очистка последнего пути в памяти.

\*/

void **pathClean**();

public:

/\*\*

\* @brief Очищает горизонтальную, вертикальную и две диагональных линии

\* одного цвета достаточной длины, проходящие через тайл.

\* @param tile Проверяемый тайл, относительно которого построятся

\* @return True если удалена линия.

\*/

bool **clearIfLined**(ColorLinesTile \*tile);

/\*\*

\* @brief Сравнивает длину линии с минимальной требуемой,

\* если длина достаночна, очищает тайлы.

\* @param list Список тайлов одного цвета, выстроенных в одну линию.

\* @return Начисленные очки.

\*/

int **clearIfLined**(std::vector<ColorLinesTile \*> line);

/\*\*

\* @brief Добавляет случайные финки в случайные свободные тайлы.

\*/

void **placeBalls**();

void **randomColors**();

/\*\*

\* @brief Проверяет наличие пути между тайлами через свободные ячейки.

\* @param from Начальная точка.

\* @param to Конечная точка.

\* @return true если такой путь существует.

\*/

bool **buildPath**(ColorLinesTile \*from, ColorLinesTile \*to);

void **goToWaitingState**();

void **goToElementMovingState**();

bool **isLose**();

};

#endif // ColorLinesGameData\_H

1. **JustAnotherLines.h**

/\*\* @file

\* @author Kalinovskiy.

\*/

#ifndef JUSTANOTHERLINES\_H

#define JUSTANOTHERLINES\_H

#include "COMMON/INTERFACES/AbstractColorLinesGame.h"

#include <memory>

#include <vector>

class **JustAnotherLines**;

class **AdapterData**;

/// Класс объекта-модели, подсказывающего следующий ход.

class **Adapter** : public AbstractColorLinesGame

{

Q\_OBJECT

/// Приватная реализация

std::unique\_ptr<AdapterData> data;

/// Объект игры, для которой выполняется подсказка.

JustAnotherLines \*game;

**Adapter**();

**Adapter**(Adapter&);

private slots:

void **update**();

public:

**Adapter**(JustAnotherLines \*game, QObject \*parent = 0);

~***Adapter***();

int ***getRowCount***() const override;

int ***getColCount***() const override;

ColorLinesTile \****getRootTile***() const override;

ColorLinesTile \****getSelectedTile***() const override;

std::vector<ColorLinesTile \*> const&***getPath***() const override;

};

class **ColorLinesGameData**;

enum

{

/// Количество фишек, появляющихся за один раз.

BALLS\_IN\_STEP = 3,

};

/// Набор цветов фишек, появляющихся за один ход

struct **NextColors**

{

ColorLinesTile::Color color[BALLS\_IN\_STEP];

};

/// Реализация игры "Цветные линии"

class **JustAnotherLines** : public AbstractColorLinesGame

{

Q\_OBJECT

friend class Adapter;

/// Приватная реализация

std::unique\_ptr<ColorLinesGameData> data;

/// Возвращает цвета фишек, которые появятся на следующим ходе.

NextColors **getNextColors**() const;

public:

explicit **JustAnotherLines**(QObject \*parent = 0);

~***JustAnotherLines***();

int ***getRowCount***() const override;

int ***getColCount***() const override;

int **getCoins**() const;

int **getCombo**() const;

const QString &***getStatistics***() const override;

ColorLinesTile \****getRootTile***() const override;

ColorLinesTile \****getSelectedTile***() const override;

std::vector<ColorLinesTile \*> const&***getPath***() const override;

/// Обрабатывает случай поражения.

void **lose**();

/// Возвращает указатель на объект-модель,

/// подсказывающий следующий ход.

AbstractColorLinesGame \***getAdapter**();

private slots:

void **update**();

public slots:

void ***tileClicked***(int col, int row) override;

};

#endif // JUSTANOTHERLINES\_H

1. **MainWidget.h**

# /\*\* @file

\* @author Kalinovskiy

\*/

#ifndef MAINWIDGET\_H

#define MAINWIDGET\_H

#include <QWidget>

#include "JustAnotherLines.h"

/// Color Lines game window

class **MainWidget** : public QWidget

{

Q\_OBJECT

public:

**MainWidget**(JustAnotherLines \*game, QWidget \*parent = 0);

};

#endif // MAINWIDGET\_H

1. **ChampionsTable.cpp**

#include "ChampionsTable.h"

#include <QCoreApplication>

#include <QSettings>

#include <QInputDialog>

#include <QMessageBox>

#include <algorithm>

namespace

{

const QString key("champions");

}

ChampionsTable::**ChampionsTable**(QString company, QString application, unsigned count, bool greater)

: company(company),

application(application),

count(count),

greater(greater)

{

QCoreApplication::setOrganizationName(company);

QCoreApplication::setOrganizationDomain("");

QCoreApplication::setApplicationName(application);

QSettings s;

QString defaultValue("NULLVALUE");

QString key = "champions";

QVariant champions = s.value(key, defaultValue);

QString championsString(champions.toString());

if(champions != defaultValue){

QStringList list = championsString.split(";");

for (QString item: list) {

QStringList itemList = item.split("#");

if(itemList.size() == 2){

long long coins = itemList[0].toLongLong();

QString name = itemList[1];

map.push\_back(Entry{coins, name});

}

}

}

}

void ChampionsTable::**setResult**(long long coins)

{

auto greaterCoinsFunctor= [&](const Entry &e1, const Entry &e2)

{

return e1.coins > e2.coins;

};

auto lessCoinsFunctor= [&](const Entry &e1, const Entry &e2)

{

return e1.coins < e2.coins;

};

if(greater)

std::sort(map.begin(), map.end(), greaterCoinsFunctor);

else

std::sort(map.begin(), map.end(), lessCoinsFunctor);

if((map.size() < count) || (coins > map.rbegin()->coins)){

QString name = QInputDialog::getText(0, "Enter your name", "Name");

map.push\_back({coins, name});

if(greater)

std::sort(map.begin(), map.end(), greaterCoinsFunctor);

else

std::sort(map.begin(), map.end(), lessCoinsFunctor);

QString outputTable;

QString text;

for(unsigned i = 0; i < count && i < map.size(); ++i){

const Entry &e = map[i];

if(i != 0){

outputTable += ";";

}

outputTable += QString("%1#%2").arg(e.coins).arg(e.name);

text += QString("%1. %2 %3\n").arg(i + 1).arg(e.name).arg(e.coins);

}

text += QString("%1 (c) %2 2021").arg(application).arg(company);

QMessageBox::information(0, "Champions", text);

QSettings s;

s.setValue(key, outputTable);

}

}

1. **ColorLinesWidget.cpp**

#include "ColorLinesWidget.h"

#include <algorithm>

#include <QResizeEvent>

#include <QMouseEvent>

#include <QPainter>

#include <QTimer>

ColorLinesWidget::**ColorLinesWidget**(AbstractColorLinesGame \*game, QWidget \*parent)

: QWidget(*parent*),

game(game),

side(0),

plotToWidgetScale(0),

widgetToPlotScale(0),

horizontalBorder(0),

verticalBorder(0),

plotWidth(10000),

cellWidth(1),

elementType(RECTANGLE),

elementWidth(0.5)

{

float k = float(std::max(game->*getColCount*(), game->*getRowCount*())) /

std::min(game->*getColCount*(), game->*getRowCount*());

int h;

int w;

if(game->*getColCount*() > game->*getRowCount*()){

w = 500;

h = 500 / k;

}else{

h = 500;

w = 500 / k;

}

setMinimumSize(w, h);

resize(w, h);

QTimer \*updateTimer= new QTimer(this);

connect(updateTimer, SIGNAL(timeout()), SLOT(update()));

updateTimer->start(10);

setFocusPolicy(Qt::StrongFocus);

setSizePolicy(QSizePolicy(QSizePolicy::Expanding,

QSizePolicy::Expanding));

}

ColorLinesWidget::~***ColorLinesWidget***()

{

}

void ColorLinesWidget::**drawElement**(QPainter &p,

ColorLinesTile \*tile,

int x,

int y)

{

p.setBrush(tile->getQColor());

float rectSide = cellWidth \* elementWidth;

float spanSize = cellWidth / 2 - rectSide / 2;

QPointF upLeft(x + spanSize, y + spanSize);

QPointF downRight(x + spanSize + rectSide, y + spanSize + rectSide);

QRectF elementRect(upLeft,downRight);

switch (elementType) {

case RECTANGLE: p.drawRect(elementRect);

break;

case CIRCLE:

default: p.drawEllipse(elementRect);

break;

}

}

void ColorLinesWidget::**drawPath**(QPainter &p)

{

for(ColorLinesTile \*t: game->*getPath*()){

QPoint point = game->*getRootTile*()->getTileCoordinate(t);

QPen pen(Qt::darkBlue);

pen.setWidth(plotWidth / 100);

p.setPen(pen);

p.setBrush(QBrush());

int x = cellWidth \* point.x();

int y = cellWidth \* point.y();

p.drawRect(x, y, cellWidth, cellWidth);

}

}

void ColorLinesWidget::***paintEvent***(QPaintEvent \*e)

{

Q\_UNUSED(e);

QPainter p(this);

p.scale(plotToWidgetScale, plotToWidgetScale);

p.translate(horizontalBorder, verticalBorder);

for(int row = 0; row < game->*getRowCount*(); row++){

for(int col = 0; col < game->*getColCount*(); col++){

ColorLinesTile \*tile = game->*getRootTile*()->getTile(col, row);

if(game->*getSelectedTile*() == tile){

p.setBrush(Qt::cyan);

}else{

p.setBrush(QBrush());

}

int x = cellWidth \* col;

int y = cellWidth \* row;

QPen pen(Qt::black);

pen.setWidth(plotWidth / 1000);

p.setPen(pen);

p.drawRect(x, y, cellWidth, cellWidth);

if(tile->getColor() != ColorLinesTile::NONE){

drawElement(*p*, *tile*, x, y);

}

}

}

QFont f;

f.setPixelSize(plotWidth / 100 \* 4);

p.setPen(Qt::black);

p.setFont(f);

drawPath(*p*);

}

void ColorLinesWidget::***resizeEvent***(QResizeEvent \* event)

{

int h = event->size().height();

int w = event->size().width();

if((float(h) / w) < float(game->*getRowCount*()) / game->*getColCount*()){

side = h;

cellWidth = float(plotWidth) / game->*getRowCount*();

}else{

side = w;

cellWidth = float(plotWidth) / game->*getColCount*();

}

widgetToPlotScale = float(plotWidth) / side;

plotToWidgetScale = float(side) / plotWidth;

horizontalBorder = (w / plotToWidgetScale -

cellWidth \* game->*getColCount*())/ 2;

verticalBorder = (h / plotToWidgetScale -

cellWidth \* game->*getRowCount*())/ 2;

}

void ColorLinesWidget::***mouseReleaseEvent***(QMouseEvent \*e)

{

int xOnPlot = e->pos().x() \* widgetToPlotScale - horizontalBorder;

int yOnPlot = e->pos().y() \* widgetToPlotScale - verticalBorder;

int col = xOnPlot / cellWidth;

int row = yOnPlot / cellWidth;

if((col >= 0) && (col < game->*getColCount*()) &&

(row >= 0) && (row < game->*getRowCount*())){

game->*tileClicked*(col, row);

}

}

void ColorLinesWidget::***keyPressEvent***(QKeyEvent \*e)

{

if(!e->isAutoRepeat()){

switch(e->key()){

case Qt::Key\_Pause: game->*pauseToggle*();

break;

case Qt::Key\_Escape: game->*escape*();

break;

default: game->*keyPressed*(e->key(), e->modifiers());

break;

}

}

}

void ColorLinesWidget::***keyReleaseEvent***(QKeyEvent \*e)

{

if(!e->isAutoRepeat()){

game->*keyReleased*(e->key());

}

}

void ColorLinesWidget::**setElementType**(elementType\_t elementType)

{

this->elementType = elementType;

}

void ColorLinesWidget::**setElementWidth**(float elementWidth)

{

this->elementWidth = elementWidth;

}

1. **StatusBar.cpp**

#include "StatusBar.h"

#include <QTimer>

StatusBar::**StatusBar**(AbstractColorLinesGame \*game, QWidget \*parent) :

QLabel(*parent*),

game(game)

{

{

QTimer \*t = new QTimer(this);

t->start(100);

connect(t, SIGNAL(timeout()), SLOT(update()));

}

setAlignment(Qt::AlignCenter);

}

void StatusBar::**update**()

{

setText(game->*getStatistics*());

}

1. **AbstractColorLinesGame.cpp**

#include "AbstractColorLinesGame.h"

#include <QApplication>

AbstractColorLinesGame::**AbstractColorLinesGame**(QObject \*parent) : QObject(*parent*)

{

}

AbstractColorLinesGame::~***AbstractColorLinesGame***()

{

}

const QString &AbstractColorLinesGame::***getStatistics***() const

{

return emptyString;

}

void AbstractColorLinesGame::***tileClicked***(int col, int row)

{

Q\_UNUSED(col);

Q\_UNUSED(row);

}

void AbstractColorLinesGame::***keyPressed***(int key, Qt::KeyboardModifiers modifiers)

{

Q\_UNUSED(key);

Q\_UNUSED(modifiers);

}

void AbstractColorLinesGame::***keyReleased***(int key)

{

Q\_UNUSED(key);

}

void AbstractColorLinesGame::***pauseToggle***()

{

}

void AbstractColorLinesGame::***escape***()

{

QApplication::quit();

}

1. **AbstractGameBuilder.cpp**

#include "AbstractGameBuilder.h"

AbstractGameBuilder::~***AbstractGameBuilder***()

{

}

**10.ColorLinesTile.cpp**

#include "ColorLinesTile.h"

#include <assert.h>

#include "COMMON/MISC/sign.h"

ColorLinesTile::**ColorLinesTile**(Color color) :

left(0),

right(0),

top(0),

bottom(0),

color(color)

{

}

ColorLinesTile \*ColorLinesTile::**getLeftTile**() const

{

return left;

}

ColorLinesTile \*ColorLinesTile::**getLeftTile**(int index) const

{

const ColorLinesTile \*tile = this;

for (int i = 0; i != index; i += SIGN(index)){

tile = tile->left;

}

return const\_cast<ColorLinesTile \*>(tile);

}

ColorLinesTile \*ColorLinesTile::**getRightTile**() const

{

return right;

}

ColorLinesTile \*ColorLinesTile::**getRightTile**(int index) const

{

const ColorLinesTile \*tile = this;

for (int i = 0; i != index; i += SIGN(index)){

tile = tile->right;

}

return const\_cast<ColorLinesTile \*>(tile);

}

ColorLinesTile \*ColorLinesTile::**getTopTile**() const

{

return top;

}

ColorLinesTile \*ColorLinesTile::**getTopTile**(int index) const

{

const ColorLinesTile \*tile = this;

for (int i = 0; i != index; i += SIGN(index)){

tile = tile->top;

}

return const\_cast<ColorLinesTile \*>(tile);

}

ColorLinesTile \*ColorLinesTile::**getBottomTile**() const

{

return bottom;

}

ColorLinesTile \*ColorLinesTile::**getBottomTile**(int index) const

{

const ColorLinesTile \*tile = this;

for (int i = 0; i != index; i += SIGN(index)){

tile = tile->bottom;

}

return const\_cast<ColorLinesTile \*>(tile);

}

void ColorLinesTile::**setLeftTile**(ColorLinesTile \*tile)

{

left = tile;

tile->right = this;

}

void ColorLinesTile::**setRightTile**(ColorLinesTile \*tile)

{

right = tile;

tile->left = this;

}

void ColorLinesTile::**setTopTile**(ColorLinesTile \*tile)

{

top =tile;

tile->bottom = this;

}

void ColorLinesTile::**setBottomTile**(ColorLinesTile \*tile)

{

bottom = tile;

tile->top = this;

}

ColorLinesTile \*ColorLinesTile::**getTile**(int col, int row) const

{

return getRightTile(col)->getBottomTile(row);

}

QPoint ColorLinesTile::**getTileCoordinate**(const ColorLinesTile \*tile) const

{

QPoint ret;

int row = 0;

const ColorLinesTile \*left = this;

while(left){

int col = 0;

const ColorLinesTile \*current = left;

while(current){

if(current == tile){

return QPoint(col, row);

}

col++;

current = current->right;

}

left = left->bottom;

row++;

}

return ret;

}

ColorLinesTile::Color ColorLinesTile::**getColor**() const

{

return color;

}

QColor ColorLinesTile::**getQColor**() const

{

QColor c;

switch (color) {

case RED:

c = QColor(Qt::red);

break;

case GREEN:

c = QColor(Qt::green);

break;

case BLUE:

c = QColor(Qt::blue);

break;

case BLACK:

c = QColor(Qt::black);

break;

case YELLOW:

c = QColor(Qt::yellow);

break;

case MAGENTA:

c = QColor(Qt::magenta);

break;

case CYAN:

c = QColor(Qt::cyan);

break;

case COUNT:

default:

assert(false);

break;

}

return c;

}

void ColorLinesTile::**setColor**(Color color)

{

this->color = color;

}

std::vector<ColorLinesTile \*> ColorLinesTile::**getTileList**() const

{

std::vector<ColorLinesTile \*> list;

const ColorLinesTile \*left = this;

while(left){

const ColorLinesTile \*current = left;

while(current){

list.push\_back(const\_cast<ColorLinesTile \*>(current));

current = current->right;

}

left = left->bottom;

}

return list;

}

std::vector<ColorLinesTile \*> ColorLinesTile::**getHorizontalColorList**() const

{

std::vector<ColorLinesTile \*> list;

list.push\_back(const\_cast<ColorLinesTile \*>(this));

ColorLinesTile \*cur = right;

while((cur != 0) && (cur->getColor() == color)){

list.push\_back(cur);

cur = cur->right;

}

cur = left;

while((cur != 0) && (cur->getColor() == color)){

list.push\_back(cur);

cur = cur->left;

}

return list;

}

std::vector<ColorLinesTile \*> ColorLinesTile::**getVerticalColorList**() const

{

std::vector<ColorLinesTile \*> list;

list.push\_back(const\_cast<ColorLinesTile \*>(this));

ColorLinesTile \*cur = getTopTile();

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = cur->getTopTile();

}

cur = getBottomTile();

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = cur->getBottomTile();

}

return list;

}

std::vector<ColorLinesTile \*> ColorLinesTile::**getDiagonal1ColorList**() const

{

std::vector<ColorLinesTile \*> list;

list.push\_back(const\_cast<ColorLinesTile \*>(this));

ColorLinesTile \*cur = (left != 0) ? left->top : 0;

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = (cur->left != 0) ? cur->left->top : 0;

}

cur = (right != 0) ? right->bottom : 0;

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = (cur->right != 0) ? cur->right->bottom : 0;

}

return list;

}

std::vector<ColorLinesTile \*> ColorLinesTile::**getDiagonal2ColorList**() const

{

std::vector<ColorLinesTile \*> list;

list.push\_back(const\_cast<ColorLinesTile \*>(this));

ColorLinesTile \*cur = (left != 0) ? left->bottom : 0;

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = (cur->left != 0) ? cur->left->bottom : 0;

}

cur = (right != 0) ? right->top : 0;

while((cur != 0) && (cur->getColor() == getColor())){

list.push\_back(cur);

cur = (cur->right != 0) ? cur->right->top : 0;

}

return list;

}

**11. ColorLinesTileMap.h**

#include "ColorLinesTileMap.h"

void ColorLinesTileMap::**set**(ColorLinesTile \*tile, ColorLinesTile::Color color)

{

if(color == ColorLinesTile::NONE){

free(*tile*);

}else{

tile->setColor(color);

ownedList.push\_back(tile);

freeList.erase(std::remove(freeList.begin(), freeList.end(), tile), freeList.end());

}

}

void ColorLinesTileMap::**free**(ColorLinesTile \*tile)

{

ownedList.erase(std::remove(ownedList.begin(), ownedList.end(), tile), ownedList.end());

freeList.push\_back(tile);

tile->setColor(ColorLinesTile::NONE);

}

void ColorLinesTileMap::**clear**()

{

std::vector<ColorLinesTile \*> list = topLeft->getTileList();

freeList = list;

ownedList.clear();

for (ColorLinesTile \*t: list) {

t->setColor(ColorLinesTile::NONE);

}

}

ColorLinesTileMap::**ColorLinesTileMap**(int colCount, int rowCount) :

topLeft(0),

colCount(colCount),

rowCount(rowCount)

{

topLeft = 0;

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

ColorLinesTile \*leftTile = (row == 0) ? topLeft : 0;

ColorLinesTile \*currentTile = leftTile;

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

ColorLinesTile \*newTile = new ColorLinesTile;

if(topLeft == 0){

topLeft = newTile;

}

if(currentTile != 0){

currentTile->setRightTile(*newTile*);

}

if(row > 0){

newTile->setTopTile(*topLeft->getTile(col,* *row* *-* 1*)*);

}

currentTile = newTile;

}

}

freeList = topLeft->getTileList();

}

ColorLinesTileMap::~**ColorLinesTileMap**()

{

deleteBottomTiles(*topLeft*);

}

int ColorLinesTileMap::**getRowCount**()

{

return rowCount;

}

int ColorLinesTileMap::**getColCount**()

{

return colCount;

}

void ColorLinesTileMap::**deleteBottomTiles**(ColorLinesTile \*tile)

{

if(tile->getBottomTile()){

deleteBottomTiles(*tile->getBottomTile()*);

}

deleteRightTiles(*tile*);

}

void ColorLinesTileMap::**deleteRightTiles**(ColorLinesTile \*tile)

{

if(tile->getRightTile()){

deleteRightTiles(*tile->getRightTile()*);

}

delete tile;

}

**12. PathSearchToWidth.cpp**

#include "PathSearchToWidth.h"

#include "MISC/container\_convenience.h"

bool PathSearchToWidth::**search**(ColorLinesTile \*from, ColorLinesTile \*to, std::vector<ColorLinesTile \*> &path)

{

bool ret = false;

tileParentMap.insert(from, 0);

openQueue.enqueue(from);

while(!openQueue.empty()){

ColorLinesTile \*current = openQueue.takeFirst();

if(current == to){

/// Конструирование пути

path.clear();

ColorLinesTile \*pathTile = current;

while(pathTile != from){

path.push\_back(pathTile);

pathTile = tileParentMap[pathTile];

}

path.push\_back(from);

ret = true;

std::reverse(path.begin(), path.end());

break;

}else

{

enqueueChild(*current->getLeftTile()*, *current*);

enqueueChild(*current->getRightTile()*, *current*);

enqueueChild(*current->getTopTile()*, *current*);

enqueueChild(*current->getBottomTile()*, *current*);

}

}

return ret;

}

void PathSearchToWidth::**enqueueChild**(ColorLinesTile \*child, ColorLinesTile \*parent)

{

bool childExists = (child != 0);

if(childExists && isFree(*child*) && notEnqued(*child*)){

openQueue.enqueue(child);

tileParentMap.insert(child,parent);

}

}

bool PathSearchToWidth::**isFree**(ColorLinesTile \*tile)

{

return tile->getColor() == ColorLinesTile::NONE;

}

bool PathSearchToWidth::**notEnqued**(ColorLinesTile \*tile)

{

return !Container::contains(tileParentMap, tile);

}

**13. ColorLinesBuilder.cpp**

/\*\* @file

\* @author Kalinovskiy.

\*/

#include "ColorLinesBuilder.h"

#include "JustAnotherLines.h"

#include "COMMON/GUI/ColorLinesWidget.h"

#include "COMMON/GUI/StatusBar.h"

#include <QVBoxLayout>

void ColorLinesBuilder::***createGame***(AbstractColorLinesGame \*&game,

QWidget \*&widget)

{

JustAnotherLines \*newGame = new JustAnotherLines;

game = newGame;

widget = new QWidget;

QVBoxLayout \*mainLayout = new QVBoxLayout(*widget*);

{

StatusBar \*bar = new StatusBar(*game*);

mainLayout->addWidget(*bar*);

}

{

/// Виджет с подсказкой о фишках, которые выпадут в следующий ход

ColorLinesWidget \*linesSmallWidget = new ColorLinesWidget(*newGame->getAdapter()*);

linesSmallWidget->setElementType(ColorLinesWidget::CIRCLE);

linesSmallWidget->setElementWidth(0.6);

mainLayout->addWidget(*linesSmallWidget*);

linesSmallWidget->setFixedHeight(50);

}

{

/// Виджет игрового поля

ColorLinesWidget \*linesWidget = new ColorLinesWidget(*game*);

linesWidget->setElementType(ColorLinesWidget::CIRCLE);

linesWidget->setElementWidth(0.6);

mainLayout->addWidget(*linesWidget*);

//mainLayout->setMargin(0);

}

}

QString ColorLinesBuilder::***getName***() const

{

return "Color Lines";

}

QIcon ColorLinesBuilder::***getIcon***() const

{

return QIcon(":/icons/icons/lines\_ico.png");

}

**14. GameData.cpp**

/\*\* @file

\* @author Kalinovskiy.

\*/

#include "GameData.h"

#include <time.h>

#include "COMMON/PathSearchToWidth.h"

ColorLinesGameData::**ColorLinesGameData**(JustAnotherLines \*game)

: lose(false),

randomEngine(time(NULL)),

combo(1),

coins(0),

tileMap(DIMENSION, DIMENSION),

selection(0),

elementMovingStatePointer(new ElementMovingState(this)),

waitingStatePointer(new WaitingState(this)),

currentState(waitingStatePointer.get()),

game(game)

{

statistics = QString("Coins=%1 Combo=%2").arg(coins).arg(combo);

randomColors();

}

bool ColorLinesGameData::**clearIfLined**(ColorLinesTile \*tile)

{

std::vector<ColorLinesTile \*> hList = tile->getHorizontalColorList();

std::vector<ColorLinesTile \*> vList = tile->getVerticalColorList();

std::vector<ColorLinesTile \*> d1List = tile->getDiagonal1ColorList();

std::vector<ColorLinesTile \*> d2List = tile->getDiagonal2ColorList();

int cash = clearIfLined(hList) +

clearIfLined(vList) +

clearIfLined(d1List) +

clearIfLined(d2List);

if(cash > 0){

coins+= cash;

combo++;

}else{

combo = 1;

}

return cash > 0;

}

int ColorLinesGameData::**clearIfLined**(std::vector<ColorLinesTile \*> line)

{

int cash = 0;

if(line.size() >= MIN\_LINE){

cash+= combo \* line.size();

for (ColorLinesTile \*t: line) {

tileMap.free(*t*);

}

}

return cash;

}

bool ColorLinesGameData::**buildPath**(ColorLinesTile \*from, ColorLinesTile \*to)

{

PathSearchToWidth search;

bool ret = search.search(*from*, *to*, *path*);

return ret;

}

void ColorLinesGameData::**placeBalls**()

{

int freeCells = tileMap.freeList.size();

int ballsWeCanPlace = std::min(freeCells, (int)BALLS\_IN\_STEP);

if(ballsWeCanPlace == 0){

lose = true;

}else{

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

int cell = randomEngine() % freeCells;

ColorLinesTile \*randomTile = tileMap.freeList[cell];

tileMap.set(*randomTile*, nextColors.color[i]);

clearIfLined(*randomTile*);

--freeCells;

}

if(tileMap.ownedList.empty()){// Чтобы с пустым полем не остаться

placeBalls();

}

randomColors();

}

if(tileMap.freeList.empty()){

lose = true;

}

}

void ColorLinesGameData::**randomColors**()

{

for(int i = 0; i < BALLS\_IN\_STEP; i++){

int randomColor = randomEngine() % ColorLinesTile::COUNT;

nextColors.color[i] = (ColorLinesTile::Color) randomColor;

}

}

void ColorLinesGameData::**goToElementMovingState**()

{

currentState = elementMovingStatePointer.get();

}

void ColorLinesGameData::**goToWaitingState**()

{

currentState = waitingStatePointer.get();

}

void ColorLinesGameData::**pathClean**()

{

path.clear();

}

ElementMovingState::**ElementMovingState**(ColorLinesGameData \*data) :

data(data)

{

}

void ElementMovingState::***update***()

{

if((data->pathIterator + 1) != data->path.end()){

ColorLinesTile \*tile = \*data->pathIterator;

ColorLinesTile::Color color = tile->getColor();

data->tileMap.free(*tile*);

++data->pathIterator;

data->tileMap.set(*\*data->pathIterator*, color);

}else{

data->goToWaitingState();

ColorLinesTile \*tile = data->path.back();

data->pathClean();

if(!data->clearIfLined(*tile*)){

data->placeBalls();

}else if(data->tileMap.ownedList.empty()){// Чтобы с пустым полем не остаться

data->placeBalls();

}

data->selection = 0;

data->statistics = QString("Coins=%1 Combo=%2").arg(data->coins).arg(data->combo);

}

}

void ElementMovingState::***tileClicked***(int col, int row)

{

Q\_UNUSED(col);

Q\_UNUSED(row);

}

WaitingState::**WaitingState**(ColorLinesGameData \*data) :

data(data)

{

}

void WaitingState::***update***()

{

}

void WaitingState::***tileClicked***(int col, int row)

{

ColorLinesTile \*tile = data->tileMap.topLeft->getTile(col, row);

if(data->selection == 0){

if(tile->getColor() != ColorLinesTile::NONE){

data->selection = tile;

}

}else{

if(tile->getColor() != ColorLinesTile::NONE){

data->selection = tile;

}else{

if(data->buildPath(*data->selection*, *tile*)){

data->goToElementMovingState();

data->pathIterator = data->path.begin();

}

}

}

}

bool ColorLinesGameData::**isLose**()

{

return lose;

}

**15.JustAnotherLines.cpp**

/\*\* @file

\* @author Kalinovskiy.

\*/

#include "JustAnotherLines.h"

#include <time.h>

#include <assert.h>

#include <QMessageBox>

#include <QTimer>

#include <QApplication>

#include "COMMON/GUI/ChampionsTable.h"

#include "COMMON/MAP/ColorLinesTileMap.h"

#include "GameData.h"

class **AdapterData**

{

public:

**AdapterData**();

ColorLinesTileMap tileMap;

std::vector<ColorLinesTile \*> path;

};

AdapterData::**AdapterData**() :

tileMap(BALLS\_IN\_STEP, 1)

{

}

Adapter::**Adapter**(JustAnotherLines \*game, QObject \*parent) :

AbstractColorLinesGame(*parent*),

data(new AdapterData),

game(game)

{

QTimer \*t = new QTimer;

t->start(10);

QObject::connect(t, SIGNAL(timeout()), this, SLOT(update()));

}

Adapter::~***Adapter***()

{

}

void Adapter::**update**()

{

assert(data);

ColorLinesTile \*tile = data->tileMap.topLeft;

NextColors nextColors = game->getNextColors();

for(int i = 0; i < data->tileMap.getColCount(); i++){

data->tileMap.set(*tile*, nextColors.color[i]);

tile = tile->getRightTile();

}

}

int Adapter::***getRowCount***() const

{

assert(data);

return data->tileMap.getRowCount();

}

int Adapter::***getColCount***() const

{

assert(data);

return data->tileMap.getColCount();

}

ColorLinesTile \*Adapter::***getRootTile***() const

{

assert(data);

return data->tileMap.topLeft;

}

ColorLinesTile \*Adapter::***getSelectedTile***() const

{

return 0;

}

std::vector<ColorLinesTile \*> const& Adapter::***getPath***() const

{

assert(data);

return data->path;

}

JustAnotherLines::**JustAnotherLines**(QObject \*parent)

: AbstractColorLinesGame(*parent*),

data(new ColorLinesGameData(this))

{

data->placeBalls();

QTimer \*t = new QTimer(this);

QObject::connect(t, SIGNAL(timeout()), this, SLOT(update()));

t->start(100);

}

JustAnotherLines::~***JustAnotherLines***()

{

}

void JustAnotherLines::**lose**()

{

assert(data);

ChampionsTable t("Vladimir Kalinovskiy", "Just Another Lines");

t.setResult(data->coins);

QMessageBox::StandardButton b =

QMessageBox::question(0, tr("Game over!"), tr("Do you want to replay?"));

if(b == QMessageBox::Yes)

{

data.reset(new ColorLinesGameData(this));

data->placeBalls();

}else{

emit quitToMenu();

}

}

AbstractColorLinesGame \*JustAnotherLines::**getAdapter**()

{

return new Adapter(this);

}

int JustAnotherLines::***getRowCount***() const

{

return DIMENSION;

}

int JustAnotherLines::***getColCount***() const

{

return DIMENSION;

}

int JustAnotherLines::**getCoins**() const

{

assert(data);

return data->coins;

}

int JustAnotherLines::**getCombo**() const

{

assert(data);

return data->combo;

}

const QString &JustAnotherLines::***getStatistics***() const

{

assert(data);

return data->statistics;

}

ColorLinesTile \*JustAnotherLines::***getRootTile***() const

{

assert(data);

return data->tileMap.topLeft;

}

ColorLinesTile \*JustAnotherLines::***getSelectedTile***() const

{

assert(data);

return data->selection;

}

std::vector<ColorLinesTile \*> const&JustAnotherLines::***getPath***() const

{

assert(data);

return data->path;

}

void JustAnotherLines::***tileClicked***(int col, int row)

{

assert(data);

data->currentState->*tileClicked*(col, row);

}

void JustAnotherLines::**update**()

{

assert(data);

if(data->isLose()){

lose();

}else{

data->currentState->*update*();

}

}

NextColors JustAnotherLines::**getNextColors**() const

{

assert(data);

return data->nextColors;

}

**16. main.cpp**

/\*\* @file

\* @author Kalinovskiy.

\*/

#include <QApplication>

#include <memory>

#include "ColorLinesBuilder.h"

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

{

QApplication a(*argc*, *argv*);

AbstractColorLinesGame \*game;

QWidget \*w;

ColorLinesBuilder b;

b.*createGame*(*game*, *w*);

std::unique\_ptr<AbstractColorLinesGame> gameP(game);

std::unique\_ptr<QWidget> widgetP(w);

widgetP->show();

return a.exec();

}

**18. MainWidget.cpp**

/\*\* @file

\* @author Kalinovskiy.

\*/

#include "MainWidget.h"

#include "COMMON/GUI/ColorLinesWidget.h"

#include <QVBoxLayout>

MainWidget::**MainWidget**(JustAnotherLines \*game, QWidget \*parent) :

QWidget(*parent*)

{

QVBoxLayout \*mainLayout = new QVBoxLayout(this);

/// Виджет с подсказкой о фишках, которые выпадут в следующий ход

ColorLinesWidget \*linesSmallWidget = new ColorLinesWidget(*game->getAdapter()*);

linesSmallWidget->setElementType(ColorLinesWidget::CIRCLE);

linesSmallWidget->setElementWidth(0.6);

mainLayout->addWidget(*linesSmallWidget*);

linesSmallWidget->setFixedHeight(50);

/// Виджет игрового поля

ColorLinesWidget \*linesWidget = new ColorLinesWidget(*game*);

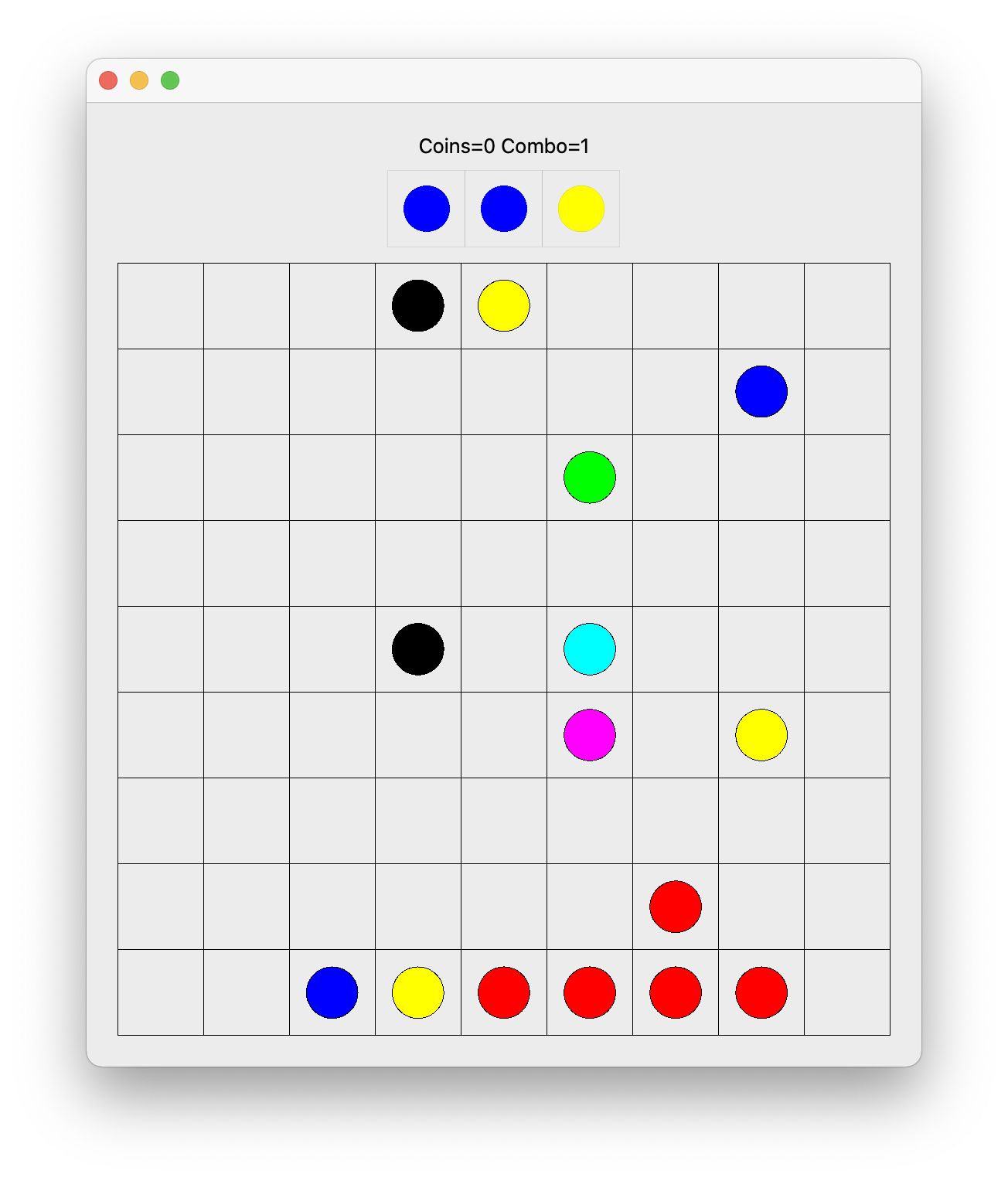
linesWidget->setElementType(ColorLinesWidget::CIRCLE);

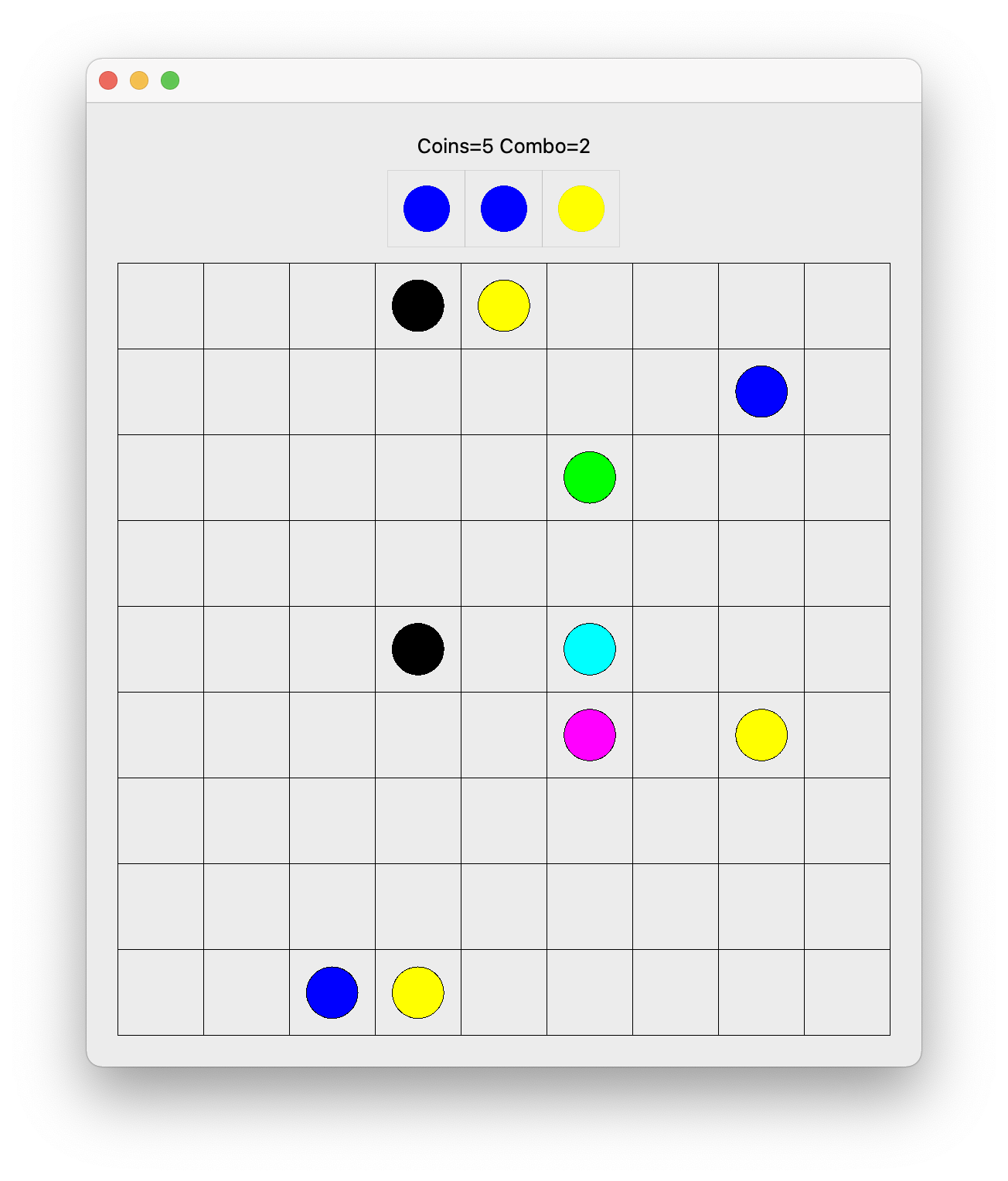
linesWidget->setElementWidth(0.6);

mainLayout->addWidget(*linesWidget*);

}

**Результаты тестирования программы:**

****

****

**Вывод:** приобрел практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt.