МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ООП»

ТЕМА: СОЗДАНИЕ ИГРОВОГО ПОЛЯ

Студент гр. 9304 Прокофьев М.Д.

Преподаватель Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Обучение работе с классами, конструкторами на языке в C++.

Задание.

Написать класс игрового поля, которое представляет из себя прямоугольник (двумерный массив). Для каждого элемента поля должен быть создан класс клетки. Клетка должна отображать, является ли она проходимой, а также информацию о том, что на ней находится. Также, на поле должны быть две особые клетки: вход и выход.

При реализации поля запрещено использовать контейнеры из stl

Обязательные требования:

- Реализован класс поля
- Реализован класс клетки
- Для класса поля написаны конструкторы копирования и перемещения, а также операторы присваивания и перемещения
- Поле сохраняет инвариант - из любой клетки можно провести путь до любой другой
- Гарантированно отсутствует утечки памяти

Дополнительные требования:

- Поле создается с использованием паттерна Синглтон
- Для обхода по полю используется паттерн Итератор. Итератор должен быть совместим со стандартной библиотекой.

Основные теоретические положения.

Класс - это пользовательский тип данных.

Конструктор копирования используется для инициализации класса путем создания копии необходимого объекта.

Оператор присваивания копированием (или «копирующее присваивание») используется для копирования одного класса в другой (существующий) класс.

Конструкторы перемещения принимают ссылку на значение объекта класса и используются для реализации передачи владения ресурсами объекта параметра

Синглтон — порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Итератор — поведенческий шаблон проектирования. Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов.

Инвариант — это свойство некоторого класса (множества) математических объектов, остающееся неизменным при преобразованиях определённого типа.

Утечка памяти — процесс неконтролируемого уменьшения объёма свободной оперативной или виртуальной памяти компьютера, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные участки памяти, или с ошибками системных служб контроля памяти.

Метод в объектно-ориентированном программировании — это функция или процедура, принадлежащая какому-то классу или объекту. Как и процедура в процедурном программировании, метод состоит из некоторого количества операторов для выполнения какого-то действия и имеет набор входных аргументов.

Выполнение работы.

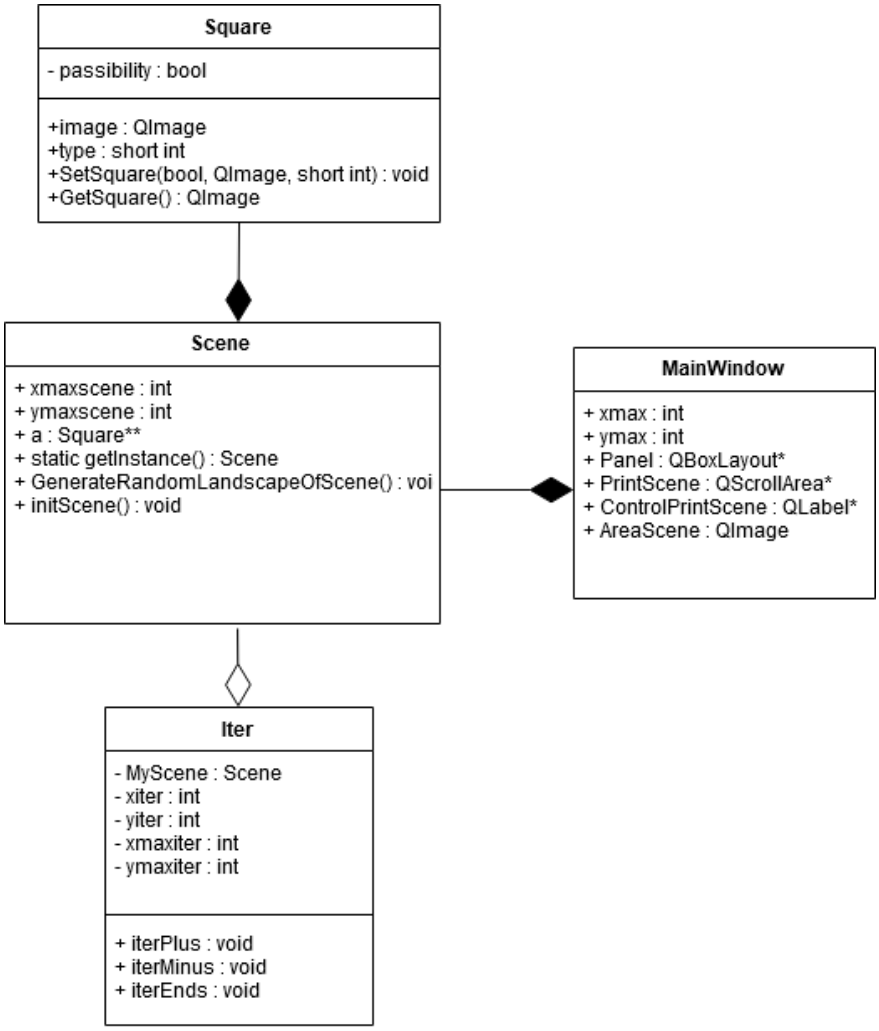

Рис.1 – UML-схема

Первоначально был создан класс square, он обозначает клетку. Этот класс содержит в себе такие параметры, как проходимость(passibility), она описана в private, изображение(image) и тип(type) они описаны в public, для того чтобы класс mainwindow, отвечающий за отображение, мог использовать этот класс. Также, класс Square содержит конструктор Square и метод void SetSquare. При вызове конструктора создается экземпляр соответствующего класса, в котором инициализируется стандарт для переменных passibility и image. Изначально это стена, поэтому passibility=0(непроходимая клетка) и qimage(". /rock. png", изображение каменного блока). SetSquare используется для изменения клетки

Далее, был создан класс Scene, который представляет из себя массив клеток(Square). Данный класс содержит в себе конструкторы копирования и перемещения и является синглтоном(static Scene* getInstance()) ожет быть только в одном экземпляре). Кроме того, для этого класса реализован итератор(Iter, обход поля).

Позднее был создан класс mainwindow, который отвечает за отображение поля в графике. В самом классе идет настройка окна, создание qlabel, в котором лежит поле рисования(AreaScene). Для рисования использован класс qt: qimage. сам принцип рисования построен по принципу:

1) Подгрузка изображение

2) Попиксельное рисование изображения на поле в определенных координатах(рисование идет начиная с лево-верхнего угла, кончая право-нижним)

Изначально все параметры поля(включая пути к изображениям клеток на поле) указываются в соответствующих классах(в классе поля и клетки). После этого уже происходит отображение самих клеток в классе mainwindow, естественно при этом используется синглтон для поля.

Также, в классе поля(Scene) присутствует случайная генерация карты, создана с помощью примитивного метода создания лабиринтов. Таким образом в самом поле сохраняется инвариант

Файл libs.h содержит все необходимые библиотеки.

Тестирование.

Запуск программы:

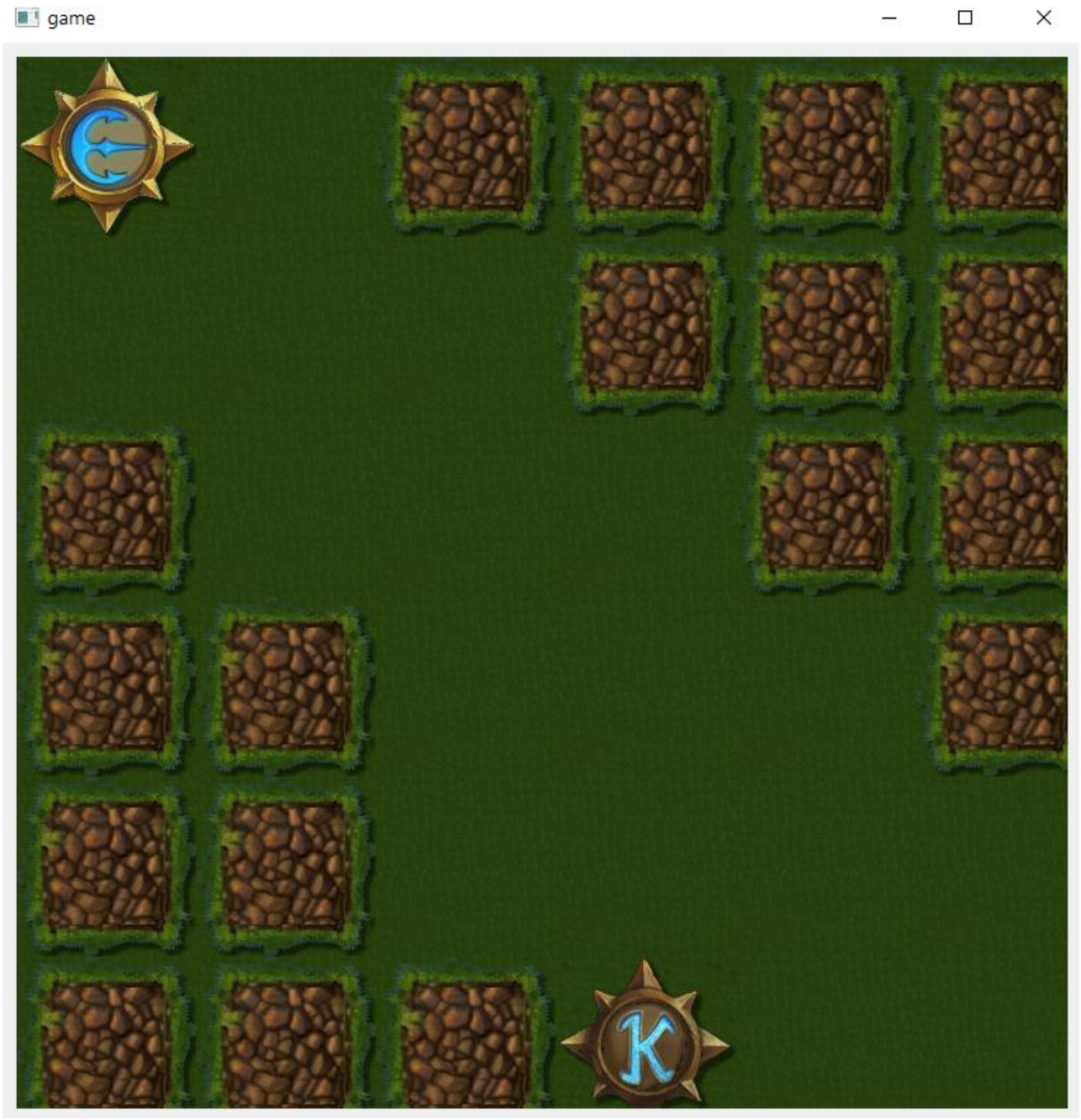

Рис.2 – Тестирование программы

Выводы

Получены знания о классах, конструкторах (копирования, перемещения), итераторе, синглтона, а также освоены навыки их использования в программировании.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "mainwindow.h"

#include <QApplication>



int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Название файла: libs.h

```
#ifndef LIBS_H
#define LIBS_H

#include <QMainWindow>
#include <QAction>
#include <QBoxLayout>
#include <QColor>
#include <QDesktopWidget>
#include <QDialog>
#include <QEvent>
#include <QHBoxLayout>
#include <QLabel>
```

```
#include <QLayout>
#include <QMainWindow>
#include <QMenu>
#include <QMenuBar>
#include <QMouseEvent>
#include <QMoveEvent>
#include <QPushButton>
#include <QStatusBar>
#include <QStyle>
#include <QToolBar>
#include <QWidget>
#include <QTextEdit>
#include <QLineEdit>
#include <QFileDialog>
#include <QRect>
#include <QIcon>
#include <QFile>
#include <QColor>
#include <QColorDialog>
#include <QScrollArea>
#include <QCommandLinkButton>
#include <QComboBox>
#include <QInputDialog>
#include <QSpinBox>
#include <QGroupBox>
#include <iterator>
#include <memory>

#include <cstring>
#include <fstream>
```

```
#include <iostream>
#include <math.h>

#include "QMessageBox"
#include <tchar.h>



#endif // LIBS_H
```

Название файла: Square.h

```
#ifndef POINT_H
#define POINT_H
#include "libs.h"
using namespace std;

const QImage standard("D:/MyLargeProjects/game/block01.png");
const QImage rock("D:/MyLargeProjects/game/block02.png");
const QImage entry("D:/MyLargeProjects/game/block03.png");
const QImage exite("D:/MyLargeProjects/game/block04.png");



class Square
{
   friend class mainwindow;
private:
   bool passibility;

public:
```

```
    QImage image;
    short int type; // 0 - свободная, 1 - стена, 2 - вход, 3 - выход
    void SetSquare(bool pass, QImage img, short int tp);
    QImage GetSquare();
    Square(bool pass=false, QImage img=rock, short int tp=1);
};

#endif // POINT_H
```

Название файла: Square.cpp

```
#include "Square.h"



Square::Square(bool pass, QImage img, short int tp)
{
    this->passibility=pass;
    this->image=img;
    this->type=tp;



}
void Square::SetSquare(bool pass, QImage img, short int tp)
{
    this->passibility=pass;
    this->image=img;
    this->type=tp;

}



QImage Square::GetSquare()
```

```
{
   return(this->image);
}
```

Название файла: Scene.h

```
#ifndef SCENE_H
#define SCENE_H
#include "Square.h"
//class Square;

class Scene
{
   Scene();

   Scene(const Scene& TimeScene);
   Scene& operator=(const Scene& TimeScene);

   Scene (Scene&& TimeScene);
   Scene& operator=(Scene&& TimeScene);

public:
   int xmaxscene=5, ymaxscene=5;
   Square** a;
  static Scene* getInstance();
   ~Scene();
   void GenerateRandomLandscapeOfScene();
   void initScene();

};
```

```
#endif // SCENE_H
```

Название файла: Scene.cpp

```
#include "scene.h"

Scene::Scene()
{

}

void Scene::initScene()
{
    this->xmaxscene;
    this->ymaxscene;



    Scene::a=new Square*[ymaxscene];
    for(int x=0; x<xmaxscene; x++)
        Scene::a[x] = new Square[xmaxscene];

}

Scene::Scene(const Scene &TimeScene){
    xmaxscene = TimeScene.xmaxscene;
    ymaxscene = TimeScene.ymaxscene;
    a = new Square* [xmaxscene];
    for (int x = 0; x < xmaxscene; x++) {
        a[x] = new Square[ymaxscene];
    }
    int x=-1,y=-1;
```

```
    while(++x<xmaxscene)
        while(++y<ymaxscene)
        {
            if(TimeScene.a[x][y].type == 2) a[x][y].type=2;
            if(TimeScene.a[x][y].type == 3) a[x][y].type=3;
            a[x][y] = TimeScene.a[x][y];
        }

}

Scene& Scene::operator= (const Scene &TimeScene){
    if(this != &TimeScene) {
        Square** an = new Square *[xmaxscene];
        for (int i = 0; i < TimeScene.xmaxscene; i++) {
            an[i] = new Square[ymaxscene];
        }
        int x=-1,y=-1;
        while(++x<xmaxscene) delete a[x];
        delete a;
        a = an;
        ymaxscene = TimeScene.ymaxscene; xmaxscene = TimeScene.xmaxscene;
        x=-1;
        while(++x<xmaxscene)
            while(++y<ymaxscene)
            {
                if(TimeScene.a[x][y].type == 2) a[x][y].type=2;
                if(TimeScene.a[x][y].type == 3) a[x][y].type=3;
                a[x][y] = TimeScene.a[x][y];
            }
    }
```

```
    return *this;
}

Scene::Scene (Scene&& TimeScene){
    ymaxscene = TimeScene.ymaxscene; xmaxscene = TimeScene.xmaxscene;
    a = TimeScene.a;
    TimeScene.a = nullptr;
}

Scene& Scene::operator=(Scene&& TimeScene){
    if (&TimeScene == this)
        return *this;
    int x=-1;
    while(++x<xmaxscene) delete a[x];
    delete a;
    xmaxscene = TimeScene.xmaxscene; ymaxscene = TimeScene.ymaxscene;
    a = TimeScene.a;
    TimeScene.a = nullptr;
    return *this;
}

void Scene::GenerateRandomLandscapeOfScene()
{
    int x=0,y=0, random;
    srand( time(0) );

   for(int i=0; i<2; i++)
   {
    x=0,y=0;
    while((x>=0)&&(y>=0)&&(x<xmaxscene)&&(y<ymaxscene))
```

```
    {
        random=rand()%2+0;
        switch(random)
        {
        case 0:
           if(x<xmaxscene) { Scene::a[x][y].SetSquare(1, standard, 0); x++; }
        case 1:
           if(y<ymaxscene) { Scene::a[x][y].SetSquare(1, standard, 0); y++; }
        }
    }
    }



    Scene::a[0][0].SetSquare(1, entry, 2);
    Scene::a[x-1][y-1].SetSquare(1, exite, 3);


}



Scene* Scene::getInstance()
{
    static Scene instance;
          return &instance;

}
Scene::~Scene(){
    for(int x = 0;x<xmaxscene;x++) {
```

```
            delete a[x];
        }
        delete[] a;
}
```

Название файла: iter.h

```
#ifndef ITER_H
#define ITER_H
#include "scene.h"


class iter
{
private:
    const Scene &MyScene;
    int xiter;
    int yiter;
    int xmaxiter=Scene::getInstance()->xmaxscene;
    int ymaxiter=Scene::getInstance()->ymaxscene;

public:
    iter(const Scene &MyScene);
    void iterPlus();
    void iterMinus();
    bool iterEnds();
    const Square& gc();
    void operator++();
    void operator--();
```

```
};

#endif // ITER_H
```

Название файла: iter.cpp

```
#include "iter.h"

iter::iter(const Scene& MyScene) : MyScene(MyScene)
{
   this->xiter=0;
   this->yiter=0;
}


void iter::iterPlus()
{
   if(yiter != ymaxiter)
   {
      if(xiter==xmaxiter) { xiter=0; yiter++;}
      else xiter++;
   }
}

void iter::iterMinus()
{
   if(yiter != -1)
   {
      if(xiter==xmaxiter) { xiter=0; yiter--;}
      else xiter--;
   }
}
```

```
bool iter::iterEnds()
{
    return(yiter == ymaxiter);
}

const Square& iter::gc()
{
    return Scene::getInstance()->a[xiter][yiter];
}
void iter::operator++()
{
    iterPlus();
}

void iter::operator--()
{
    iterMinus();
}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "scene.h"
#include "iter.h"
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    int xmax, ymax;
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QBoxLayout *Panel;
    QScrollArea *PrintScene;
    QLabel *ControlPrintScene;
    QImage *AreaScene;




};
#endif // MAINWINDOW_H
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
    xmax=6;ymax=6;
    // // // // Настройка поля
    Scene::getInstance()->xmaxscene=xmax;
    Scene::getInstance()->ymaxscene=ymax;
    Scene::getInstance()->initScene();
    Scene::getInstance()->GenerateRandomLandscapeOfScene();
    int winxmax=xmax*128, winymax=ymax*128;
    // // // // Работа с окном
```

```
        QWidget *cw = new QWidget(this);
        Panel = new QVBoxLayout(cw);
        PrintScene = new QScrollArea();

        ControlPrintScene = new QLabel(PrintScene);
        ControlPrintScene->setMouseTracking(true);
        ControlPrintScene->setSizePolicy(QSizePolicy::Ignored,
QSizePolicy::Ignored);

        Panel->addWidget(PrintScene);

        setMinimumSize(winxmax, winymax);
        setCentralWidget(cw);
        AreaScene = new QImage(winxmax, winymax,  QImage::Format_RGB888 );
        for(int i = 0; i < winxmax; i++){
           for(int j = 0; j < winymax; j++ ){
              AreaScene->setPixel( j, i, qRgb(0, 0, 0 ) );
              }
           }


        // // // // Рисуем само поле
        for(int x=0; x<xmax; x++)
        {
           for(int y=0;y<ymax;y++)
           {
              for(int i = 0; i < 128; i++){
                    for(int j = 0; j < 128; j++ ){
                       QRgb pixColor;
                       pixColor = Scene::getInstance()->a[x][y].image.pixel(j, i);
```

```
if((qRed(pixColor)!=0)&&(qGreen(pixColor)!=0)&&(qBlue(pixColor)!=0))
AreaScene->setPixel( i+128*x, j+128*y, pixColor);

                    }
                }
            }
            }
            ControlPrintScene->resize(winxmax, winymax);
            ControlPrintScene->setPixmap(QPixmap::fromImage(*AreaScene));



        }
        MainWindow::~MainWindow()
        {
        }
```