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 классов, конструкторов и методов классов

Студент гр. 0383 _____ Козлов Т.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Игровое поле представляет из себя прямоугольную плоскость разбитую на клетки. На поле на клетках в дальнейшем будут располагаться игрок, враги, элементы взаимодействия. Клетка может быть проходимой или непроходимой, в случае непроходимой клетки, на ней ничего не может располагаться. На поле должны быть две особые клетки: вход и выход. В дальнейшем игрок будет появляться на клетке входа, а затем выполнив определенный набор задач дойти до выхода.

Требования:

- Реализовать класс поля, который хранит набор клеток в виде двумерного массива.
- Реализовать класс клетки, которая хранит информацию о ее состоянии, а также того, что на ней находится.
- Создать интерфейс элемента клетки.
- Обеспечить появление клеток входа и выхода на поле. Данные клетки не должны быть появляться рядом.
- Для класса поля реализовать конструкторы копирования и перемещения, а также соответствующие операторы.
- Гарантировать отсутствие утечки памяти.

- *Потенциальные паттерны проектирования, которые можно использовать:*
- *Итератор (Iterator) - обход поля по клеткам и получение косвенного доступа к ним*
- *Строитель (Builder) - предварительное конструирование поля с необходимым параметрами. Например, предварительно задать кол-во непроходимых клеток и алгоритм их расположения*

Ход работы:

Был реализован класс клетки *Cell* (который по ходу выполнения других лабораторных работ будет масштабироваться) с полями координатами, а так же *_item* поле - указатель, ссылающийся на экземпляр некоторого класса, наследуемого от интерфейса *IEntity* и отвечающего за объект, находящийся в клетке. (использован паттерн «Стратегия»). Так же для координат реализованы геттеры. Перегружены конструкторы копирования, перемещения, а так же соответствующие операторы.

От класса *Cell* публично наследуется разновидность клетки *GraphicCell* с методами для изображения клетки и ее содержимого, а так же полями координатами в пикселях на графическом поле. Перегружены конструкторы копирования, перемещения, а так же соответствующие операторы.

От класса *GraphicCell* публично наследуются три класса (разновидности графических клеток): *Entrance*, *Exit*, *Way* – классы входа, выхода и «пути» соответственно. В дальнейшем спавн игрока будет происходить на клетке

выхода, целью будет добраться до выхода, а перемещение будет осуществляться через клетки «пути».

Был создан интерфейс класса клетки *IEntity*, от которого будут наследоваться классы, экземпляры которых могут находится в клетке (игрок, враги, предметы).

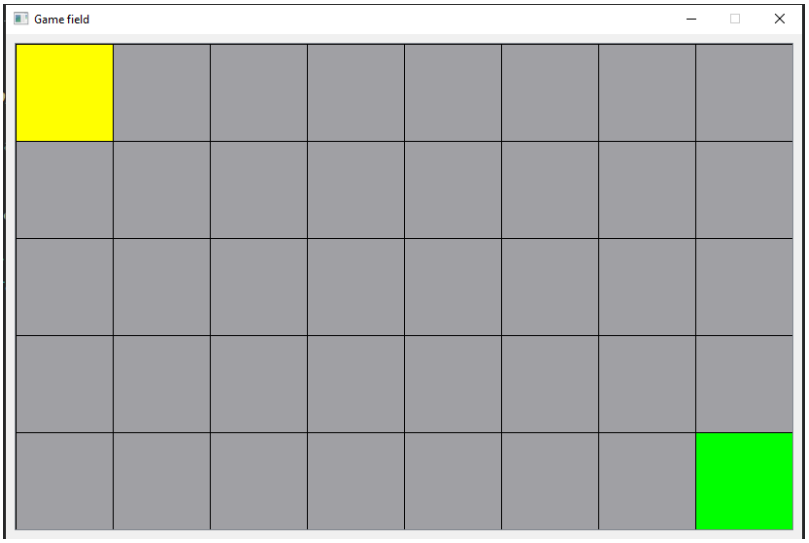
Был реализован класс поле *Field*, с реализованными конструкторами копирования и перемещения и перегруженными операторами присваивания и присваивания-перемещения. Также *Field* имеет *protected* поля *_heightOfCell*, *_widthOfCell* – отвечающие за размер клетки поля в пикселях и *_heightInCells*, *_widthInCells* –отвечающие за размеры поля в «клетках» (что необходимо для работы с полем: инициализации двумерного массива клеток и прочее..) Так же были реализованы геттеры для этих полей.

Так же в *Field* находится указатель на указатель на указатель на *Cell* – это массив из клеток, что в главной степени и определяет поле. Инициализация происходит в конструкторе, память под Cell выделяется в куче.

От класса *Field* наследуется разновидность поля класс - *GraphicField* (который также наследуется от класса *QGraphicsItem* библиотеки QT, что необходимо для реализации GUI). Этот класс дополняет родительский методами и полями, необходимыми для функционирования GUI (переопределяет два метода *QGraphicsItem*: ***paint() и boundingRect***).

Реализован класс *MainWindow*, наследуемый от класса QT *QMainWindow* которое является десктопным приложением. Класс содержит поля *GraphicField* и *GraphicScene* _scene, а так же метод, который отвечает за добавление *GraphicField* на графичесую сцену.

Приложение реализуется через метод *show()* который вызывается в клиентской функции *main*, при создании экземпляра класса *MainWindow* сразу передаются параметры графического поля. (разработанный код см. в ПРИЛОЖЕНИИ А).



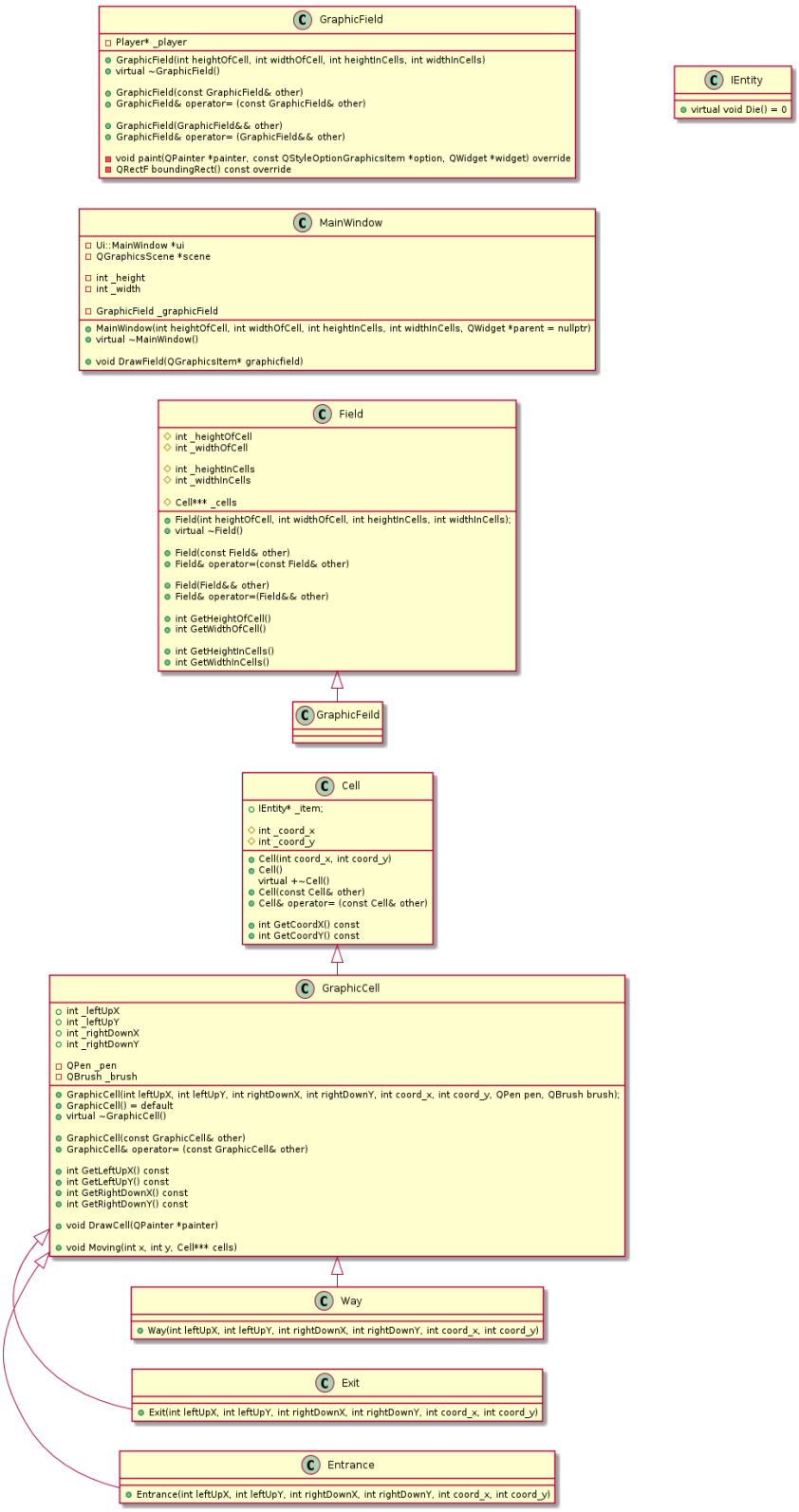
(размер клетки – 100 на 100 px, размер поля в клетках: 5 на 8)

Программа корректно завершается:

```
FightOrDie
04:16:05: C:\QtProjects\OOP\build-FightOrDie-Desktop_Qt_6_2_0_MinGW_64_bit-Debug\debug\FightOrDie.exe exited with code 0

04:46:11: Starting C:\QtProjects\OOP\build-FightOrDie-Desktop_Qt_6_2_0_MinGW_64_bit-Debug\debug\FightOrDie.exe ...
04:46:54: C:\QtProjects\OOP\build-FightOrDie-Desktop_Qt_6_2_0_MinGW_64_bit-Debug\debug\FightOrDie.exe exited with code 0
```

UML-диаграмма:



Выводы:

По завершению данной лабораторной работы была заложена основа для будущей игры FightOrDie, были изучены умные указатели, был применен паттерн «Стратегия», для данного этапа был реализован GUI и сделан UML-отчет.

ПРИЛОЖЕНИЕ А

ientity.h

```
#pragma once

#include "Avatar/avatar.h"

class IEntity
{
public:
    virtual void Die() = 0;
};
```

cell.h

```
#pragma once

#include"Entities/Player/player.h"
#include<memory>
#include<iostream>

class Cell
{
public:
    Cell(int coord_x, int coord_y);
    Cell() = default;
    virtual ~Cell();

    Cell(const Cell& other);
    Cell& operator= (const Cell& other);

    int GetCoordX() const;
    int GetCoordY() const;

    //void Moving(int coord_x, int coord_y);
    //std::shared_ptr<IEntity> _item;

protected:
    int _coord_x, _coord_y;

public:
    IEntity* _item; // Find out at what point in time shared_ptr will call
"delete"
};
```

cell.cpp

```
#include "cell.h"
#include "graphiccell.h"

Cell::Cell(int coord_x, int coord_y)
    : _coord_x(coord_x)
    , _coord_y(coord_y)
{

}

Cell::~Cell()
{

}

Cell::Cell(const Cell& other)
    : _coord_x(other._coord_x)
    , _coord_y(other._coord_y)
{
    _item = other._item; // Add copy constructor for concrete type of enemy
}

Cell& Cell::operator=(const Cell& other)
{
    _coord_x = other._coord_x;
    _coord_y = other._coord_y;

    _item = other._item; // Add copy constructor for concrete type of enemy

    return *this;
}

int Cell::GetCoordX() const { return _coord_x; }
int Cell::GetCoordY() const { return _coord_y; }
```

field.h

```
#pragma once

#include "Cells/entrance.h"
#include "Cells/exit.h"
#include "Cells/way.h"

class Field
{
public:
    Field(int heightOfCell, int widthOfCell, int heightInCells, int
widthInCells);
    virtual ~Field();

    Field(const Field& other);
    Field& operator=(const Field& other);

    Field(Field&& other);
    Field& operator=(Field&& other);
```

```
    int GetHeightOfCell();
    int GetWidthOfCell();

    int GetHeightInCells();
    int GetWidthInCells();

protected:
    int _heightOfCell;
    int _widthOfCell;

    int _heightInCells;
    int _widthInCells;

    Cell*** _cells; // Will think how to make it private

//    Cell* _entrance;
//    Cell* _exit;
};
```

field.cpp

```
#include "field.h"

Field::Field(int heightOfCell, int widthOfCell, int heightInCells, int
widthInCells)
    : _heightOfCell(heightOfCell)
    , _widthOfCell(widthOfCell)
    , _heightInCells(heightInCells)
    , _widthInCells(widthInCells)
{
    _cells = new Cell**[_heightInCells];

    // Just memory allocation
    for(int i = 0; i < _heightInCells; i++)
    {
        _cells[i] =  new Cell*[_widthInCells];
    }
}

Field::~Field()
{
    for(int i = 0; i < _heightInCells; i++)
    {
        delete[] _cells[i];
    }
    delete[] _cells;
}

Field::Field(const Field& other)
    : _heightOfCell(other._heightOfCell)
    , _widthOfCell(other._widthOfCell)
    , _heightInCells(other._heightInCells)
    , _widthInCells(other._widthInCells)
{
    _cells = new Cell**[_heightInCells];

    for(int i = 0; i < _heightInCells; i++)
    {
        _cells[i] =  new Cell*[_widthInCells];
        for(int j = 0; j < _widthInCells; j++)
        {
            *_cells[i][j] = *other._cells[i][j];
```

```
        }
    }
}

Field& Field::operator=(const Field &other)
{
    if(&other == this)
        return *this;

    _heightOfCell = other._heightOfCell;
    _widthOfCell = other._widthOfCell;
    _heightInCells = other._heightInCells;
    _widthInCells = other._widthInCells;

    _cells = new Cell**[_heightInCells];

    for(int i = 0; i < _heightInCells; i++)
    {
        _cells[i] =  new Cell*[_widthInCells];
        for(int j = 0; j < _widthInCells; j++)
        {
            *_cells[i][j] = *other._cells[i][j];
        }
    }

    return *this;
}

Field::Field(Field&& other)
    : _heightOfCell(other._heightOfCell)
    , _widthOfCell(other._widthOfCell)
    , _heightInCells(other._heightInCells)
    , _widthInCells(other._widthInCells)
{
    _cells = other._cells;
    other._cells = nullptr;
}

Field& Field::operator=(Field&& other)
{
    if(&other == this)
        return *this;

    _heightOfCell = other._heightOfCell;
    _widthOfCell = other._widthOfCell;
    _heightInCells = other._heightInCells;
    _widthInCells = other._widthInCells;

    delete _cells;

    _cells = other._cells;
    other._cells = nullptr;

    return *this;
}

int Field::GetHeightOfCell() { return _heightOfCell; }
int Field::GetWidthOfCell() { return _widthOfCell; }

int Field::GetHeightInCells() { return _heightInCells; }
int Field::GetWidthInCells() { return _widthInCells; }
```

graphicfield.h

```
#pragma once

#include "field.h"
#include <QGraphicsItem>

class GraphicField : public QObject, public Field, public QGraphicsItem
{
    Q_OBJECT
public:
    GraphicField(int heightOfCell, int widthOfCell, int heightInCells, int
widthInCells);
    virtual ~GraphicField();

    GraphicField(const GraphicField& other);
    GraphicField& operator= (const GraphicField& other);

    GraphicField(GraphicField&& other);
    GraphicField& operator= (GraphicField&& other);
public slots:
    void MovingPlayer(int x, int y);
signals:
    void MovingItemCells(int x, int y, Cell*** cells);
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget) override;
    QRectF boundingRect() const override;
    Player* _player;
};
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QGraphicsScene>
#include "QGraphicsItem"
#include "ui_mainwindow.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(int height, int width, QWidget *parent = nullptr);
    ~MainWindow();
    void DrawField(QGraphicsItem* graphicfield);

private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;

    const int _height;
    const int _width;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"


MainWindow::MainWindow(int height, int width,  QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , _height(height)
    , _width(width)
{
    ui->setupUi(this);

    setFixedSize(_width + 20, _height + 20);

    scene = new QGraphicsScene();
    ui->graphicsView->setScene(scene);
    scene->setSceneRect(0, 0, _width, _height);
    //ui->graphicsView->setRenderHint(QPainter::Antialiasing);
    ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
}


void MainWindow::DrawField(QGraphicsItem* graphicfield)
{
    scene->addItem(graphicfield);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```