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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объектно-ориентированное программирование»

Тема: Создание классов, конструкторов и методов класса

Студент гр. 0383 _____ Кусмарцев А.И.

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.

Задание.

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.

Выполнение работы.

В ходе работы были реализованы следующие классы: *class field, class field_image, class cell, class pass_cell, class entity.* Класс *field_image* — интерфейс, который предоставляет пользователю методы, по работе с программой. Класс *field* реализует класс-интерфейс *field_image*. Класс *field* хранит в своих полях: экземпляры класса *cell* в виде двумерного массива, для

представления игрового поля, и экземпляры классов *pass_cell*, для хранения координат клеток входа и выхода. В свою очередь класс *cell* хранит в себе экземпляр класса *entity*, для представления сущности, располагающейся на конкретной клетке.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Была разработана программа по созданию игрового поля и заполнения его проходимыми клетками, непроходимыми клетками, клетками входа и выхода. Для этого были созданы классы и методы в них для решения данной задачи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/main.h

```
#pragma once
#include "userAction.h"
#include "field.h"
#include "field_image.h"
#include "pass_cell.h"
#include "entity.h"
#include "cell.h"
#include <stddef.h>

int main();
```

Название файла: src/main.cpp

```
#include "field.h"
#include "userAction.h"

int main(){
    srand(static_cast<unsigned int>(time(0)));
    field game{};
    field_image game_image{game};
    userAction(game_image);
    return 0;
}
```

Название файла: src/userAction.h

```
#pragma once
#include "field.h"
#include "field_image.h"
#include "pass_cell.h"
#include "entity.h"
#include "cell.h"

void userAction(field_image& obj);
```

Название файла: src/userAction.cpp

```
#include "userAction.h"
void userAction(field_image& obj){
    obj.read_features();
    obj.modify_board();
    obj.print_board();
}
```

Название файла: src/field.h

```
#pragma once
#include <vector>
#include "pass_cell.h"
#include "entity.h"
#include "cell.h"

class field{
    std::size_t width;
    std::size_t height;
    cell** board;
    std::size_t q_unpass;
    pass_cell entry;
    pass_cell exit;
public:
        field(std::size_t width = 0, std::size_t height = 0,
std::size_t q_unpass = 0, pass_cell entry = pass_cell(), pass_cell
exit = pass_cell());
    field(const field& other);
    field& operator=(const field& other);
    field(field&& other);
    field& operator=(field&& other);
    ~field();
    void generate_io();
    void generate_unpass();
    void set_features(std::vector<std::size_t> temp);
    std::size_t get_width();
    std::size_t get_height();
    cell** get_board();
    std::vector<std::size_t> get_entry();
    std::vector<std::size_t> get_exit();
};
```

Название файла: src/field.cpp

```
#include "field.h"

field::field(std::size_t width, std::size_t height, std::size_t
q_unpass,    pass_cell    entry,    pass_cell    exit):width(width),
height(height), q_unpass(q_unpass), entry(entry), exit(exit){
    if(width && height){
        board = new cell*[width];
        for(std::size_t i = 0;i < width; i++){
            board[i] = new cell[height];
        }
    }
}
field::field(const       field&       other):width(other.width),
height(other.height),   q_unpass(other.q_unpass),   entry(other.entry),
exit(other.exit){
    if(width && height){
        board = new cell*[width];
        for(std::size_t i = 0; i < width; i++){
           board[i] = new cell[height];
        }
```

```
    }
    for(std::size_t i = 0; i < width; i++)
        for(std::size_t j = 0; j < height; j++)
            board[i][j] = other.board[i][j];
}
field& field::operator=(const field& other){
    if(this != &other){
        if(width && height){
            for(std::size_t i = 0; i < width; i++)
                delete[] board[i];
            delete[] board;
        }
        width = other.width;
        height = other.height;
        q_unpass = other.q_unpass;
        entry = other.entry;
        exit = other.exit;
        if(other.width && other.height){
            board = new cell*[other.width];
            for(std::size_t i = 0; i<other.width; i++)
                board[i] = new cell[other.height];
            for(std::size_t i = 0; i < other.width; i++)
                for(std::size_t j = 0; j < other.height; j++)
                    board[i][j] = other.board[i][j];
        }
    }
    return *this;
}
field::field(field&& other){
    std::swap(width, other.width);
    std::swap(height, other.height);
    std::swap(q_unpass, other.q_unpass);
    std::swap(board, other.board);
    std::swap(entry, other.entry);
    std::swap(exit, other.exit);
}
field& field::operator=(field&& other){
    if(this != &other){
        std::swap(width, other.width);
        std::swap(height, other.height);
        std::swap(board, other.board);
        std::swap(height, other.height);
        std::swap(entry, other.entry);
        std::swap(exit, other.exit);
    }
    return *this;
}
field::~field(){
    if(width && height){
        for(size_t i = 0; i<width; i++)
            delete[] board[i];
        delete[] board;
    }
}
void field::generate_io(){
    std::size_t temp_ix, temp_iy;
```

```
            std::size_t temp_ox, temp_oy;
            temp_ix = std::rand() % (width-2) + 1;
            temp_iy = std::rand() % (height-2) + 1;
            temp_ox = std::rand() % width;
            temp_oy = std::rand() % height;
            entry.set_cell(temp_ix, temp_iy);
             while(temp_ox == temp_ix && temp_ox == temp_ix -1 && temp_ox
==  temp_ix  +1  &&  temp_oy  ==  temp_iy  &&  temp_oy  ==  temp_iy  -1  &&
temp_oy == temp_iy +1){
                temp_ox = std::rand() % width;
                temp_oy = std::rand() % height;
            }
            exit.set_cell(temp_ox, temp_oy);
        }
        void field::generate_unpass(){
            std::size_t temp_x, temp_y;
            temp_x = std::rand() % width;
            temp_y = std::rand() % height;
            std::vector<std::size_t> check_i;
            std::vector<std::size_t> check_o;
            check_i = entry.get_cell();
            check_o = exit.get_cell();
            for(std::size_t i = 0; i < q_unpass; i++){
                 while(((temp_x == check_i[0] && temp_y == check_i[1]) ||
(temp_x  ==  check_o[0]  &&  temp_y  ==  check_o[1]))  ||  !board[temp_x]
[temp_y].get_pass()){
                    temp_x = std::rand() % width;
                    temp_y = std::rand() % height;
                }
                board[temp_x][temp_y].set_pass();
            }
        }
        void field::set_features(std::vector<std::size_t> temp){
            if(width && height){
                for(std::size_t i =0; i < width; i++){
                    delete[] board[i];
                }
                delete[] board;
            }
            this->width = temp[0];
            this->height = temp[1];
            this->q_unpass = temp[2];
            if(width && height){
                board = new cell*[width];
                for(std::size_t i = 0; i < width; i++){
                    board[i] = new cell[height];
                }
            }
            return;
        }
        std::size_t field::get_width(){
            return this->width;
        }
        std::size_t field::get_height(){
            return this->height;
        }
```

```
cell** field::get_board(){
    return this->board;
}
std::vector<std::size_t> field::get_entry(){
    return entry.get_cell();
}
std::vector<std::size_t> field::get_exit(){
    return exit.get_cell();
}
```

Название файла: src/field_image.h

```
#pragma once
#include <fstream>
#include <vector>
#include "field.h"


class field_image{
    field& obj;
public:
    field_image(field& obj);
    void read_features();
    void print_board();
    void modify_board();
};
```

Название файла: src/field_image.cpp

```
#include "field_image.h"

field_image::field_image(field& obj):obj(obj){}
void field_image::read_features(){
    std::ifstream in("../parametrs.txt");
    char word[20];
    std::vector<size_t> temp;
    for(int i = 0; i<3; i++){
        in >> word;
        temp.push_back(atoi(word));
    }
    obj.set_features(temp);
    return;
}
void field_image::print_board(){
    remove("./board.txt");
    cell** temp_board = obj.get_board();
    std::vector<size_t> check_i = obj.get_entry();
    std::vector<size_t> check_o = obj.get_exit();
    std::ofstream out("./board.txt", std::ios::app);
    if(out.is_open()){
        for(size_t i=0; i<obj.get_width();i++){
            for(size_t j = 0; j< obj.get_height(); j++){
                if(check_i[0] == i && check_i[1] ==j){
                    out << "I";
                } else if(check_o[0] == i && check_o[1] ==j){
                    out << "O";
```

```
                } else if(temp_board[i][j].get_pass()){
                    out << ".";
                } else
                    out << "X";
            }
            out << "\n";
        }
    }
    out.close();
}
void field_image::modify_board(){
    obj.generate_io();
    obj.generate_unpass();
}
```

Название файла: src/cell.h

```
#pragma once
#include "entity.h"

class cell{
    bool pass;// 1 - да, 0 - нет
    entity unit;
public:
    cell(bool pass = 1, entity unit = entity());
    void set_pass();
    bool get_pass();
};
```

Название файла: src/cell.cpp

```
#include "cell.h"
cell::cell(bool pass, entity unit):pass(pass), unit(unit){}
void cell::set_pass(){
    pass = 0;
}
bool cell::get_pass(){
    return this->pass;
}
```

Название файла: src/pass_cell.h

```
#pragma once
#include <vector>


class pass_cell{
protected:
    std::size_t x;
    std::size_t y;
public:
    pass_cell(std::size_t x = 0, std::size_t y = 0);
    void set_cell(std::size_t x, std::size_t y);
    std::vector<std::size_t> get_cell();
};
```

Название файла: src/pass_cell.cpp

```
#include "pass_cell.h"
pass_cell::pass_cell(std::size_t x, std::size_t y): x(x), y(y){}
void pass_cell::set_cell(std::size_t x, std::size_t y){
    this->x = x;
    this->y = y;
}
std::vector<std::size_t> pass_cell::get_cell(){
    std::vector<std::size_t> ret;
    ret.push_back(this->x);
    ret.push_back(this->y);
    return ret;
}
```

Название файла: src/entity.h

```
#pragma once
#include <string>

class entity{
    std::string name;
public:
    entity(std::string name = "unknown");
};
```

Название файла: src/entity.cpp

```
#include "entity.h"
entity::entity(std::string name): name(name){}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Входные данные :

```
1 25
2 50
3 100
```

Выходные данные: