**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 классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Калмак Д.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

## **Цель работы.**

Создать классы, конструкторы и методы классов.

## **Задание.**

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.

При реализации класса поля запрещено использовать контейнеры из stl

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

## **Выполнение работы.**

Был реализован класс Interface\_cell, в котором прописан интерфейс клетки. Реализованы виртуальные функции получения информации об объекте на клетке и состоянии клетки, также изменения объекта и состояния клетки.

Класс Cell наследуется от класса Interface\_cell. В нем задается информация о клетке: ее состояние – проходимость, вход, выход, объект на ней. Виртуальные функции переопределены. Добавлена возможность изменения пользователем информации о клетке.

Реализован класс Field. Данный класс определяет игровое поле. В нем хранится динамический двумерный массив клеток, а также его размеры. Определены конструкторы копирования и перемещения и соответствующие операторы. В классе определен метод вывода поля. Для класса Field класс Builder дружеский. Он служит для генерации поля.

Разработанный код см. в Приложении А.

Разработанную диаграмму классов UML см. в Приложении Б.

## **Тестирование.**

Результаты тестирования представлены в Таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 – Результаты тестирования | | | |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | 10 10 | 0000000000  i010100110  010001110o  0010000000  0010101000  0000101000  0101010100  0110000100  0100010000  0110100110 | Есть проход от входа к выходу. Все верно. |

|  |  |  |  |
| --- | --- | --- | --- |
| Продолжение таблицы 1 | | | |
| 2 | 15 15 | 000000000000000  010111001100000  001100010010000  001100100001110  011010100000010  011011001100000  i0100100110011o  010000001101010  001101011001110  000110000000010  010111110110010  001110111011000  000100111111100  001100011011100  001011101010110 | Есть проход от входа к выходу. Все верно. |

## **Выводы.**

Таким образом, были реализованы классы, методы, конструкторы, конструкторы копирования, перемещения и соответствующие операторы.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Interface\_cell.h:**

#include <iostream>

using namespace std;

class *Interface\_cell*{

    public:

        virtual *string* get\_object(){

        }

        virtual *string* get\_condition(){

        }

        virtual void change\_object(){

        }

        virtual void change\_object(*string* *obj*){

        }

        virtual void change\_condition(){

        }

        virtual void change\_condition(*string* *condit*){

        }

};

**Файл Cell.h:**

#include "Interface\_cell.h"

class Cell: public *Interface\_cell*{

    private:

        string condition; // 0 passable 1 impassable i in o out

        string object;

    public:

        Cell(){

            condition = " ";

            object = "Empty";

        }

        Cell(*string* *condit*, *string* *obj*): condition(condit), object(obj){}

        Cell(*string* *condit*): condition(condit){}

*string* get\_object(){

            //cout << "Объект на клетке: " << object << endl;

            return object;

        }

*string* get\_condition(){

            //cout << "Состояние клетки: " << condition << endl;

            return condition;

        }

        void change\_object(){

            cout << "Ввод объекта на клетку: ";

            cin >> object;

        }

        void change\_object(*string* *obj*){

            object = obj;

        }

        void change\_condition(){

            cout << "Ввод состояние клетки: ";

            cin >> condition;

        }

        void change\_condition(*string* *condit*){

            condition = condit;

        }

};

**Файл Field.h:**

#include <cstdlib>

#include <ctime>

#include "Cell.h"

class *Field*{

    friend class *Builder*;

    protected:

        int n, m;

*Cell*\*\* arr;

    public:

        Field(int *n\_size*, int *m\_size*){

            n = *n\_size*;

            m = *m\_size*;

            arr = **new** *Cell*\*[n];

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

                arr[i] = **new** *Cell*[m];

            }

        }

        Field(const *Field*& *object*){

            n = *object*.n;

            m = *object*.m;

            arr = **new** *Cell*\*[n];

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

                arr[i] = **new** *Cell*[m];

            }

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

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

                    arr[i][j] = *object*.arr[i][j];

                }

            }

        }

*Field* & operator=(const *Field* & *object*){

            if (*this* != &*object*){

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

**delete** arr[i];

                }

**delete[]** arr;

                n = *object*.n;

                m = *object*.m;

                arr = **new** *Cell*\*[n];

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

                    arr[i] = **new** *Cell*[m];

                }

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

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

                        arr[i][j] = *object*.arr[i][j];

                    }

                }

            }

            return \**this*;

        }

        Field(*Field*&& *object*){

            n = *object*.n;

            m = *object*.m;

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

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

                    arr[i][j] = *object*.arr[i][j];

                }

            }

*object*.arr = nullptr;

*object*.n = 0;

*object*.m = 0;

        }

*Field* & operator=(*Field* && *object*){

            if (*this* != &*object*){

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

**delete** arr[i];

                }

**delete[]** arr;

                n = *object*.n;

                m = *object*.m;

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

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

                        arr[i][j] = *object*.arr[i][j];

                    }

                }

*object*.arr = nullptr;

*object*.n = 0;

*object*.m = 0;

            }

            return \**this*;

        }

        ~Field(){

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

**delete** arr[i];

            }

**delete[]** arr;

        };

        void print\_Field(){

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

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

                    cout << arr[i][j].get\_condition();

                }

                cout << endl;

            }

        }

};

**Файл Builder.cpp:**

#include "Field.h"

class Builder{

    private:

        int n;

        int m;

    public:

        Builder(int *n*, int *m*): n(n), m(m){}

*Field* fill\_Field(){

            Field field(n, m);

            srand(time(0));

            for (int i = 1; i < n; i++){

                for (int j = 1; j < m-1; j++){

                    field.arr[i][j].change\_condition(to\_string(rand() %2));

                }

            }

            for (int i = 0; i < m; i ++) field.arr[0][i].change\_condition("0");

            for (int i = 0; i < n; i ++) field.arr[i][0].change\_condition("0");

            for (int i = 0; i < n; i ++) field.arr[i][m-1].change\_condition("0");

            field.arr[rand() %10][0].change\_condition("i");

            field.arr[rand() %10][m-1].change\_condition("o");

            return field;

        }

};

int main(){

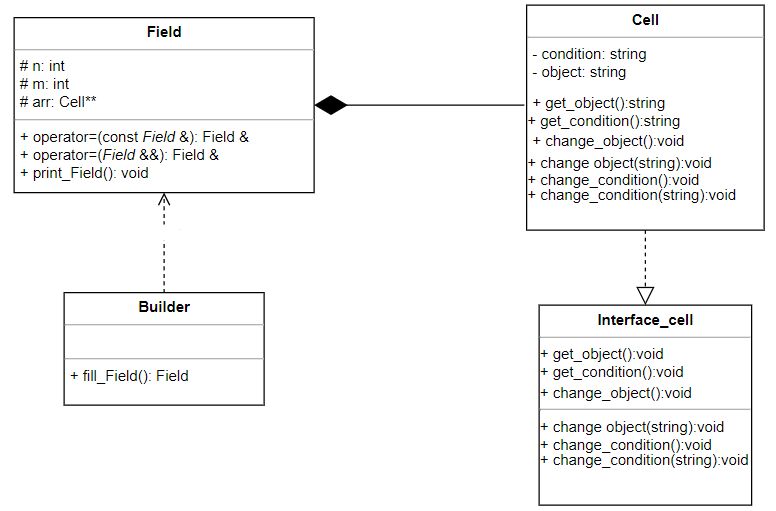
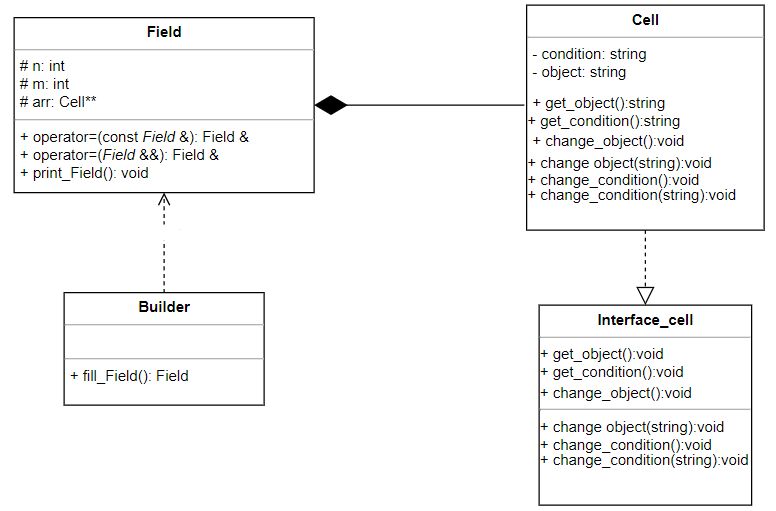
    Builder field\_gen(10, 10);

    field\_gen.fill\_Field().print\_Field();

    return 0;

}

# **ПРИЛОЖЕНИЕ Б UML ДИАГРАММА КЛАССОВ**



-

-