**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 Object, который описывает объект на клетке. Для него реализованы методы вывода и ввода.

Был реализован класс Interface\_cell, в котором прописан интерфейс клетки. Реализованы виртуальные функции получения информации о состоянии клетки, также изменения состояния клетки.

Класс Cell наследуется от класса Interface\_cell. В нем задается информация о клетке: ее состояние – проходимость. Виртуальные функции переопределены. Также реализованы классы Entrance и Exit, которые наследуются от класса Cell в которых также переопределен метод вывода информации.

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, а также методы очистки, копирования и перемещения. Для класса Field класс Builder дружеский. Он служит для генерации поля. Для класса Field также реализован дружеский класс field\_View, который необходим для вывода 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 Object.h:**

#ifndef OBJECT\_H

#define OBJECT\_H

#include <iostream>

using namespace std;

class *Object*{

    private:

*string* obj;

    public:

*string* get\_object();

        void change\_object();

        void change\_object(*string* *object*);

};

#endif

**Файл Object.cpp:**

#include "Object.h"

*string* *Object*::get\_object(){

    return obj;

}

void *Object*::change\_object(){

    cout << "Ввод объекта на клетку: ";

    cin >> obj;

}

void *Object*::change\_object(*string* *object*){

    obj = *object*;

}

**Файл Interface\_cell.h:**

#ifndef INTERFACECELL\_H

#define INTERFACECELL\_H

#include "Object.h"

class *Interface\_cell*{

    protected:

*Object* obj;

    public:

        virtual *string* get\_condition(){

        }

        virtual void change\_condition(){

        }

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

        }

};

#endif

**Файл Cell.h:**

#ifndef CELL\_H

#define CELL\_H

#include "Interface\_cell.h"

class *Cell*: public *Interface\_cell*{

    private:

*string* condition;

    public:

        Cell(){

            condition = "0";

            obj.change\_object("Empty");

        }

*string* get\_condition(){

            return condition;

        }

        void change\_condition(){

            cout << "Ввод состояние клетки: ";

            cin >> condition;

        }

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

            condition = *condit*;

        }

};

#endif

**Файл Entrance.h:**

#include "Cell.h"

class *Entrance*: public *Cell*{

    private:

*string* condition;

    public:

        Entrance(){

            condition = "i";

            obj.change\_object("Empty");

        }

*string* get\_condition(){

            return condition;

        }

};

**Файл Exit.h:**

#include "Cell.h"

class Exit: public Cell{

    private:

        string condition;

    public:

        Exit(){

            condition = "e";

            obj.change\_object("Empty");

        }

        string get\_condition(){

            return condition;

        }

};

**Файл Field.h:**

#include <cstdlib>

#include <ctime>

#include "Cell.h"

class *Field*{

    friend class *Builder*;

    friend class *field\_View*;

    private:

        void clear();

        void copy(const *Field* & *object*);

        void move(*Field* && *object*);

    protected:

        int n, m;

*Cell*\*\* arr;

    public:

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

        Field(const *Field*& *object*);

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

        Field(*Field*&& *object*);

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

        ~Field();

};

**Файл Field.cpp:**

#include "Field.h"

void *Field*::clear(){

    for (int i = 0; i < n; i++){

**delete** arr[i];

    }

**delete[]** arr;

}

void *Field*::copy(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];

        }

    }

}

void *Field*::move(*Field* && *object*){

    swap(n, *object*.n);

    swap(m, *object*.m);

    swap(arr, *object*.arr);

}

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

    copy(*object*);

}

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

    if (*this* != &*object*){

        clear();

        copy(*object*);

    }

    return \**this*;

}

*Field*::Field(*Field* && *object*){

    move(std::move(*object*));

}

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

    if (*this* != &*object*){

        clear();

        move(std::move(*object*));

    }

    return \**this*;

}

*Field*::~Field(){

    clear();

}

**Файл Builder.h:**

#ifndef BUILDER\_H

#define BUILDER\_H

#include "Field.h"

#include "Entrance.h"

#include "Exit.h"

class *Builder*{

    private:

        int n;

        int m;

    public:

        Builder(int *n*, int *m*);

*Field* fill\_Field();

};

#endif

**Файл Builder.cpp:**

#include "Builder.h"

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

*Field* *Builder*::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");

*Entrance* i;

    field.arr[rand() %n][0].change\_condition(i.get\_condition());

*Exit* e;

    field.arr[rand() %n][m-1].change\_condition(e.get\_condition());

    return field;

}

**Файл field\_View.cpp:**

#include "Builder.h"

class *field\_View*{

    public:

        field\_View(){}

        void print\_F(*Field* *object*){

*Field* field\_for\_print = *object*;

            for (int i = 0; i < field\_for\_print.n; i++){

                for (int j = 0; j < field\_for\_print.m; j++){

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

                }

                cout << endl;

            }

        }

};

int main(){

*Builder* field\_gen(15, 15);

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

*Field* field = field\_gen.fill\_Field();

*field\_View* f;

    f.print\_F(field);

    return 0;

}

# **ПРИЛОЖЕНИЕ Б UML ДИАГРАММА КЛАССОВ**

