

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:

Самара Р.Д.

Преподаватель:

Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Научиться создавать собственные классы, их конструкторы и деструкторы, конструкторы копирования и перемещения и специальные методы класса.

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 `std`

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.

Порядок выполнения поставленной задачи программой:

- 1 Все файлы программы были разбиты на две директории: `include` — заголовочные файлы классов с разрешением, и `src` — сами классы.
- 2 Реализован класс `Game`, запускающий основную бизнес-цепочку игры, создающий игровое поле с указанными пользователем размерами поля.
- 3 Реализован класс поля `Field`, состоящей из клеток `Grid`, который хранит набор клеток в виде двумерного массива, а также параметры, задающие размеры поля. Класс `Grid` хранит всю информацию о каждой клетке — её координаты, проходима ли она, является ли входом или выходом.
- 4 В классе `Field` созданы конструкторы копирования и перемещения, а также операторы для них. В перегрузке оператора перемещения мы очищаем память, выделенную под сетку, копируем из объекта параметры и координаты клеток. В перегрузке оператора копирования мы делаем то же самое, но без очищения памяти.
- 5 Создан класс `GridEntity`, который в будущем будет хранить информацию о сущностях, расположенных на клетках.
- 6 В классе `Field` происходит отрисовка поля с помощью метода `drawField`.
- 7 Реализовано случайное появление клеток входа и выхода на поле. Данные клетки не появляются рядом.
- 8 Реализован класс `Draw`, отвечающий за отрисовку поля в консоли.
- 9 Гарантированно отсутствие утечки памяти.

Разработанный программный код см. в приложении А

Диаграмму с зависимостями классов см. в приложении Б

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	Результат	Комментарии
1	10 10 Field field = new Field(10, 10);	<pre>##### # _ _ _ _ _ # # S _ _ _ _ _ # # _ _ _ _ _ # # _ _ _ _ _ # # _ _ _ _ _ # # _ _ _ _ _ # # _ _ _ _ _ E _ # # _ _ _ _ _ # #####</pre>	Поле сгенерировано и отрисовано.
2	Field field2 (15, 15) field = field2; cout « field.getHeight()	15	Конструктор копирования и оператор присваивания корректны.
3	field=std::move(field2) cout«field2.getHeight()	0	Конструктор перемещения и оператор присваивания корректны

1 Выводы.

В ходе работы было изучены способы создания собственных классов, их конструкторов и деструкторов, полей и методов.

Разработана программа по типу мини игры, генерирующая поле с непроходимыми клетками и стартом/финишем.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "../include/game.h"

int main(){
    int x, y;
    cin >> x;
    cin >> y;
    Game game (x, y);
    game.runGame(x, y);
    return 0;
}
```

Название файла: game.h

```
using namespace std;
#pragma once
#include "draw.h"

class Game{
private:
    Field *field;
    Draw drawer;
public:
    Game(int, int);
    ~Game();
    void runGame(int x, int y);
};
```

Название файла: game.cpp

```
#include "../include/game.h"

Game::Game(int x, int y){}
void Game::runGame(int x, int y){
    this->field = new Field(x, y);
    drawer.drawField(field);
}
Game::~~Game(){
    delete field;
}
```

Название файла: GridEntity.h

```
#pragma once

class GridEntity{}; //класс, отвечающий за будущее добавление сущностей в игру.
```

Название файла: GridEntity.cpp

```
#include "../include/GridEntity.h"
```

Название файла: grid.h

```
#include <iostream>
using namespace std;
#pragma once
#include "../include/GridEntity.h"

class Grid{
private:
    //GridEntity entity;
    int x = 0;
    int y = 0;
    bool passable;
    char value = '_';
public:
    enum Type {BORDER, START, END, EMPTY};
    Type type;
    Grid();
    Grid(int x, int y, Type type);
    int getX();
    int getY();
    Type getTypeOfGrid();
    void setTypeOfGrid(Type type);
    void setValue(Type type);
    bool isPassable();
    char getValue();
};
```

Название файла: grid.cpp

```
#include "../include/grid.h"

Grid::Grid(int x, int y, Type type):
    x(x),
    y(y),
    type(type){
    passable = (type != BORDER);
    setValue(type);
}
```

```

Grid::Grid(){
    };

int Grid::getX(){
    return x;
}

int Grid::getY(){
    return y;
}

char Grid::getValue(){
    return value;
}

Grid::Type Grid::getTypeOfGrid(){
    return type;
}

void Grid::setValue(Type type){
    if (type == BORDER)
        value = '#';
    else if (type == START)
        value = 'S';
    else if (type == END)
        value = 'E';
    else
        value = '_';
}

void Grid::setTypeOfGrid(Type type){
    this->type = type;
}

bool Grid::isPassable(){
    return passable;
}

```

Название файла: field.h

```

using namespace std;
#pragma once
#include "grid.h"
#include <string.h>

```

```

class Field{
private:
    int height = 0;
    int width = 0;
    void createField(int width, int height);
    void createField();
    void createField(int size);
public:
    Grid** grid;
    Field(int width, int height);
    Field(const Field &); //копирование
    Field &operator=(const Field &);
    Field(Field &&);
    Field &operator=(Field &&);
    ~Field();
    Grid getCell(int, int);
    int getHeight();
    int getWidth();
};

```

Название файла: field.cpp

```

#include "../include/field.h"

```

```

Field::Field(int w, int h)
    :width(w),
    height(h)
{
    createField(width, height);
}

void Field::createField(int width, int height){
    srand(time(NULL));
    int x_start, y_start, x_end, y_end; //задание случайных координат на границах поля
    x_start = rand()%(width/4) + 1; //создание случайных координат входа и выхода внутри поля
    y_start = rand()%(height/4) + 1;
    x_end = rand()%(width/4) + width*3/4 - 1;
    y_end = rand()%(height/4) + height*3/4 - 1;

    grid = new Grid*[width];
    for (int i = 0; i < width; i++) {
        grid[i] = new Grid[height];
        for (int j = 0; j < height; j++) {
            if (i * j == 0 || i == width - 1 || j == height - 1) { //назначение краев карты
                grid[i][j] = Grid(i, j, Grid::BORDER);
            }
        }
    }
}

```



```

    }
    else
        grid[i][j] = Grid(i, j, Grid::EMPTY); //остальные пустые клетки
    }
}
grid[x_start][y_start] = Grid(x_start, y_start, Grid::START);
grid[x_end][y_end] = Grid(x_end, y_end, Grid::END);
}

```

```

Field::Field(const Field &other): //конструктор копирования
    width(other.width),
    height(other.height)
{
    for(int i = 0; i < width; i++){
        for(int j = 0; j < width; j++){
            grid[i][j] = other.grid[i][j];
        }
    }
}

```

```

Field &Field::operator=(const Field &other){ //оператор копирования
    if(&other == this) return *this;
    for(int i = 0; i < width; i++)
        delete [] grid[i];
    delete [] grid;

    width = other.width;
    height = other.height;
    grid = new Grid*[width];
    for(int i = 0; i < width; i++){
        grid[i] = new Grid[height];
    }
    for(int i = 0; i < width; i++){
        for (int j = 0; j < height; j++){
            grid[i][j] = other.grid[i][j];
        }
    }
    return *this;
}

```

```

Field::Field(Field &&other): //конструктор перемещения
    width(other.width),
    height(other.height),
    grid(other.grid)

```

```

{
    std::swap(grid, other.grid);
    std::swap(width, other.width);
    std::swap(height, other.height);
}

Field &Field::operator=(Field &&other){ //оператор перемещения
    if(this == &other) //попытка присвоить объект самому себе
        return *this;
    std::swap(grid, other.grid);
    std::swap(width, other.width);
    std::swap(height, other.height);
    return *this;
}

void Field::createField(){
    createField(10, 10);
}

void Field::createField(int size){
    createField(size, size);
}

Grid Field::getCell(int x, int y){
    return grid[x][y];
}

int Field::getWidth(){
    return width;
}

int Field::getHeight(){
    return height;
}

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

```

Название файла: draw.h

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

```
class Draw{
public:
    void drawField(Field *field);
};
```

Название файла: draw.cpp

```
#include "../include/draw.h"

void Draw::drawField(Field *field){
    int width = field->getWidth();
    int height = field->getHeight();
    string char_to_str = ""; //преобразование char в string для отрисовки поля
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            char_to_str += field->grid[i][j].getValue();
            cout << " " + char_to_str;
            char_to_str = "";
        }
        cout << "" << endl;
    }
}
```

ПРИЛОЖЕНИЕ Б

UML-ДИАГРАММА КЛАССОВ

