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 _____ Бояркин Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить механизм работы создания классов, их конструкторов и деструкторов, конструкторы копирования и перемещения и специальные методы класса, используя стандартную библиотеку языка C++.

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.

Порядок выполнения поставленной задачи программой:

1) Реализован класс клетки Cage, которая хранит информацию о ее состоянии, а также того, что на ней находится.

Описаны перечисления:

- *TypeObj* - тип объекта (ничего, игрок, враг, предмет)
- *TypeCage* - тип клетки (старт, конец, проходимая клетка, непроходимая клетка, пустая клетка)

Поля класса:

- *int typeObj* - тип объекта
- *int typeCage* - тип клетки
- *int x* - координата x
- *int y* - координата y
- *CageEntity* entity* - то что находится внутри клетки (какой именно враг, какой именно предмет и т.п.)

Реализованы два конструктора - без аргументов и с аргументами (x и y).

Реализованы функции void *setTypeCage(TypeCage typeCage = TypeCage::EMPTY)* и *void setTypeObj(TypeObj typeObj = TypeObj::NOTHING)* для установки типа клетки и типа объекта на клетки соответственно. Также, обратные им функции *TypeCage getTypeCage() const* и *TypeObj getTypeObj() const*, которые позволяют узнать какой тип клетки и какой объект находится на клетки. Помимо этого, функции *int getX() const* и *int getY() const* позволяют взять значения X и Y у клетки.

2) Для него реализован класс CageView (просмотр, что находится внутри этой клетки)

Для него разработан только один конструктор, которому в качестве аргумента передается ссылка на объект типа Cage.

Реализованы функции *void printCage() const* и *void printObj() const*, которые позволяют печатать тип клетки и сам объект.

3) Реализован класс поля Field, который хранит набор клеток в виде двумерного массива.

Класс Field имеет следующие приватные поля:

- *int width* - ширина поля
- *int height* - высота поля
- *Cage start* - клетка старта
- *Cage finish* - клетка финиша
- *Cage** board* - двумерный массив клеток

Для класса поля Field реализованы два конструктора: Field() и Field(int w, int h). В первом случае, полям width, height присваивается нулевое значение, а полю board - nullptr. Во втором случае, сначала идет проверка на корректность высоты и ширины (если значения некорректны, то будет создано поле 2 на 2). Далее идет выделение памяти под двумерный массив и он заполняется клетками с типом клетки “пустой” и объектом “ничего”. После этого, изменяется значение клеток start и finish с помощью функции *Cage** *entry_exit_creat(Cage** arr)*, которая в свою очередь работает так: рандомно выбираются две клетки на границах поля с помощью функции *Cage generateBorderPoint() const* (и с помощью функции *bool isCorrectDistStartFinish(Cage start, Cage finish) const* проверяется, что расстояние между стартом и финишем корректно).

4) Для неё реализован класс клетки FieldView (просмотр всех клеток)

В этом классе реализованы две функции, которые позволяют печатать все клетки у поля:

- *void printField(Cage** board) const* - печатает клетки с учетом их типа клетки (старт, конец, проходимая, непроходимая, пустая)
- *void printFieldObj(Cage** board) const* - печатает клетки с учетом их объекта (ничего, игрок, враг, предмет)

5) Обеспечил появление клеток входа и выхода на поле. Данные клетки не появляются рядом. См. пункт 3.
6) Для класса поля реализованы конструкторы копирования и перемещения, а также соответствующие операторы:
   - *Field(const Field& obj)* - конструктор копирования
   - *Field& operator=(const Field& obj)* - оператор присванивания с копированием
   - *Field(Field&& obj) noexcept* - конструктор перемещения
   - *Field& operator=(Field&& obj) noexcept;* - оператор перемещения
7) Гарантированно отсутствие утечки памяти.

Разработанный программный код см. в приложении А.

UML-диаграммы см. в приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Field model(10, 10); FieldView modelView(model); Cage** table = model.getField();	ОК	Создано пустое поле с появлением точек входа и выхода. Данные точки не находятся рядом.

	modelView.printField(table);
	std::cout << "\n";
	modelView.printFieldObj(table);		

2.	Field model2(3, 3);
	model2= std::move(model);
	FieldView model2View(model2);
	Cage** table2 = model2.getField();
	model2View.printFieldObj(table2);
	std::cout << "\n";
		OK	Конструктор перемещения и оператор присваивания корректны

3.	Field copyfield = model;
	copyfield = model2;
	FieldView copyfieldView(copyfield);
	Cage** table3 = copyfield.getField();
	copyfieldView.printFieldObj(table3);
	std::cout << "\n";
		OK	Конструктор копирования и оператор присваивания корректны

Выводы.

Был изучен механизм работы создания классов, их конструкторов и деструкторов, конструкторы копирования и перемещения и специальные методы класса, используя стандартную библиотеку языка C++.

Приложение А

Исходный код программы

Название файла: main.cpp

```
#include <iostream>
#include "Cage.h"
#include "Field.h"
#include "FieldView.h"

int main(){

//    Cage model;
//    model.setTypeCage();
//    model.setTypeObj();
//    int a = model.getTypeCage();
//    int b = model.getTypeObj();
//    std::cout << a << " " << b << std::endl;
//    CageView view(model);
//    view.printCage();
//    view.printObj();
    int w = 7;
    int h = 7;
    Field model(w, h);
    FieldView modelView(model);
    Cage** table = model.getField();
    modelView.printField(table);
    std::cout << "\n";
    modelView.printFieldObj(table);

//    std::cout << "\n";
//    Field model2(3, 3);
//    model2 = std::move(model);
//    FieldView model2View(model2);
//    Cage** table2 = model2.getField();
//    model2View.printField(table2);
//    std::cout << "\n";
//
//
//    Field copyfield = model;
//    copyfield = model2;
//    FieldView copyfieldView(copyfield);
//    Cage** table3 = copyfield.getField();
//    copyfieldView.printField(table3);
//    std::cout << "\n";

    return 0;
}
```

Название файла: Cage.h

```
#pragma once
```

```
#include <iostream>
#include "CageEntity.h"

enum TypeObj{  // исправил на вверхний регистр, т.к. это константы
    NOTHING,
    PLAYER,
    ENEMY,
    INTERACTION_ELEMENTS
};

enum TypeCage{
    START,
    END,
    PASSABLE,
    IMPASSABLE,
    EMPTY
};

class Cage{
private:
    int typeCage;
    int typeObj;
    int x, y;
public:
    CageEntity* entity;
    friend class CageView;
    Cage();                // prev: Cage(): x(0), y(0) {}
     Cage(int x, int y);  // prev: Cage(int x, int y): x(x), y(y)
{}
    ~Cage();
    void setTypeCage(TypeCage typeCage = TypeCage::EMPTY);
    void setTypeObj(TypeObj typeObj = TypeObj::NOTHING);
    [[nodiscard]] TypeCage getTypeCage() const;
    [[nodiscard]] TypeObj getTypeObj() const;
    [[nodiscard]] int getX() const;
    [[nodiscard]] int getY() const;
};
```

Название файла: Cage.cpp

```
#include "Cage.h"
#include <iostream>

Cage::Cage() {
    this->x = 0;
    this->y = 0;
    this->typeObj = TypeObj::NOTHING;
    this->typeCage = TypeCage::EMPTY;
    this->entity = nullptr;
}

Cage::Cage(int x, int y) {
    this->x = x;
```

```
        this->y = y;
        this->typeObj = TypeObj::NOTHING;
        this->typeCage = TypeCage::EMPTY;
        this->entity = nullptr;
    }

    Cage::~Cage() = default;

    void Cage::setTypeCage(TypeCage typeCage) {
        this->typeCage = typeCage;
    }

    void Cage::setTypeObj(TypeObj typeObj) {
        this->typeObj = typeObj;
    }

    TypeCage Cage::getTypeCage() const{  // prev: int
            return  static_cast<TypeCage>(this->typeCage);    //  prev:
this->typeCage
    }

    TypeObj Cage::getTypeObj() const {  // prev: int
             return  static_cast<TypeObj>(this->typeObj);    //  prev:
this->typeObj
    }

    int Cage::getX() const {
        return this->x;
    }

    int Cage::getY() const {
        return this->y;
    }
```

Название файла: CageView.h

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

    class CageView{
    private:
        Cage& cell;
    public:
        explicit CageView(Cage& obj);
        void printCage() const;
        void printObj() const;
    };
```

Название файла: CageView.cpp

```
    #include "CageView.h"
    #include <iostream>
    #include <map>
```

```
#include <string>


CageView::CageView(Cage &obj): cell(obj){}

void CageView::printObj() const {
    // нужно убрать switches
//    switch (cell.getTypeObj()) {
//        case NOTHING:
//            std::cout << "[ ]";
//            break;
//        case PLAYER:
//            std::cout << "[P]";
//            break;
//        case ENEMY:
//            std::cout << "[E]";
//            break;
//        case INTERACTION_ELEMENTS:
//            std::cout << "[I_E]";
//            break;
//    }
     std::map<TypeObj, std::string> out;  // можно ли использовать
контейнер string?
     out[TypeObj::NOTHING] = "[ ]";
     out[TypeObj::PLAYER] = "[P]";
     out[TypeObj::ENEMY] = "[E]";
     out[TypeObj::INTERACTION_ELEMENTS] = "[I_E]";
     std::cout << out[cell.getTypeObj()];
}

void CageView::printCage() const {
    // нужно убрать switches
//    switch (cell.getTypeCage()) {
//        case START:
//            std::cout << "[START]";
//            break;
//        case END:
//            std::cout << "[END]";
//            break;
//        case PASSABLE:
//            std::cout << "[PASS]";
//            break;
//        case IMPASSABLE:
//            std::cout << "[IPASS]";
//            break;
//        case EMPTY:
//            std::cout << "[EMPTY]";
//    }
     std::map<TypeCage, std::string> out;  // можно ли использовать
контейнер string?
     out[TypeCage::START] = "[START]";
     out[TypeCage::END] = "[END]";
     out[TypeCage::PASSABLE] = "[PASS]";
```

```
        out[TypeCage::EMPTY] = "[EMPTY]";
        std::cout << out[cell.getTypeCage()];
    }
```

Название файла: CageEntity.h

```
#pragma once

// интерфейс
class CageEntity{};
```

Название файла: Field.h

```
#pragma once
#include "Cage.h"
#include "CageView.h"

class Field{
private:
    int width;
    int height;
    Cage start;
    Cage finish;
    Cage** board;
public:
    Field();  // Field():width(0), height(0), board(nullptr){};
    Field(int w, int h);
    Field(const Field& obj);  // конструктор копирования
    Field& operator=(const Field& obj);  // оператор присванивания
с копированием
    Field(Field&& obj) noexcept; // конструктор перемещения
        Field& operator=( Field&& obj) noexcept; // оператор
перемещения
    ~Field();
    friend class FieldView;
//      Cage **array_generator(unsigned int dim1, unsigned int
dim2);
//    Cage **fill_array(Cage** arr);
//    void array_destroyer(Cage **arr, unsigned int dim2);
    [[nodiscard]] Cage **getField() const;
     [[nodiscard]] bool isCorrectDistStartFinish(Cage start, Cage
finish) const;
    [[nodiscard]] Cage generateBorderPoint() const;
    Cage** entry_exit_creat(Cage** arr);
};
```

Название файла: Field.cpp

```
#include "Field.h"
#include <iostream>
#include <cstdlib>
```

```
Field::Field() {
    this->width = 0;
    this->height = 0;
    this->board = nullptr;
}

Field::Field(int w, int h){
    if (w < 2 and h < 2){
        w = 2;
        h = 2;
    }
    this->width = w;
    this->height = h;
//    board = array_generator(w, h);
//    board = fill_array(board);
//    board = entry_exit_creat(board);
    board = new Cage * [height];
    for (int i = 0; i < height; i++) {
        board[i] = new Cage[width];
    }
    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; j++) {
            board[i][j] = Cage(j, i);  // Cage(i, j)
            board[i][j].setTypeCage(TypeCage::EMPTY);
            board[i][j].setTypeObj(TypeObj::NOTHING);
        }
    }
    board = entry_exit_creat(board);
}

Field::~Field(){
    //array_destroyer(board, height);
    for (int i = 0; i < height; i++) {
        delete [] board[i];
    }
    delete [] board;
}

Field::Field(const Field& obj):width(obj.width),
height(obj.height){
    // исправить костыли! (исправлено)
      //TypeCage typesCage[5] = {TypeCage::START, TypeCage::END,
TypeCage::PASSABLE, TypeCage::IMPASSABLE, TypeCage::EMPTY};
      //TypeObj typesObj[4] = {TypeObj::NOTHING, TypeObj::PLAYER,
TypeObj::ENEMY, TypeObj::INTERACTION_ELEMENTS};

    start = obj.start;
    finish = obj.finish;
    board = new Cage * [height];
    for (int i = 0; i < height; ++i) {
        board[i] = new Cage[width];
    }
```

```
        for (int i = 0; i < height; ++i) {
            for (int j = 0; j < width; ++j) {
                board[i][j] = Cage(i, j);
                                                                    //    prev:
board[i][j].setTypeCage(typesCage[obj.board[i][j].getTypeCage()]);
                                                                    //    prev:
board[i][j].setTypeObj(typesObj[obj.board[i][j].getTypeObj()]);

board[i][j].setTypeCage(obj.board[i][j].getTypeCage());
                board[i][j].setTypeObj(obj.board[i][j].getTypeObj());
            }
        }
    }


    Field& Field::operator=(const Field& obj){
        // исправить костыли (исправлено)
          //TypeCage typesCage[5] = {TypeCage::START, TypeCage::END,
TypeCage::PASSABLE, TypeCage::IMPASSABLE, TypeCage::EMPTY};
          //TypeObj typesObj[4] = {TypeObj::NOTHING, TypeObj::PLAYER,
TypeObj::ENEMY, TypeObj::INTERACTION_ELEMENTS};

        if (this != &obj){
            for (int i = 0; i < height; ++i) {
                delete[] board[i];
            }
            delete[] board;
            width = obj.width;
            height = obj.height;
            start = obj.start;
            finish = obj.finish;
            board = new Cage * [height];
            for (int i = 0; i < height; ++i) {
                board[i] = new Cage[width];
            }
            for (int i = 0; i < height; ++i) {
                for (int j = 0; j < width; ++j) {
                    board[i][j] = Cage(j, i);  // Cage(j, i)

board[i][j].setTypeCage(obj.board[i][j].getTypeCage());

board[i][j].setTypeObj(obj.board[i][j].getTypeObj());
    //
board[i][j].setTypeCage(typesCage[obj.board[i][j].getTypeCage()]);
    //
board[i][j].setTypeObj(typesObj[obj.board[i][j].getTypeObj()]);
                }
            }
        }
        return *this;
    }

    Field::Field(Field&& obj) noexcept{
```

```
        std::swap(width, obj.width);
        std::swap(height, obj.height);
        std::swap(start, obj.start);
        std::swap(finish, obj.finish);
        std::swap(board, obj.board);
    }

    Field& Field::operator=( Field&& obj) noexcept {
        if (this != &obj){
            std::swap(width, obj.width);
            std::swap(height, obj.height);
            std::swap(start, obj.start);
            std::swap(finish, obj.finish);
            std::swap(board, obj.board);
        }
        return *this;
    }

    Cage** Field::getField() const{
        return this->board;
    }

    bool  Field::isCorrectDistStartFinish(Cage  start,  Cage  finish)
const{
        int distStartFinish = std::max(((width + height) / 2), 2);
        return abs(start.getX() - finish.getX()) +
               abs(start.getY() - finish.getY()) >= distStartFinish;
    }

    Cage Field::generateBorderPoint() const {
        switch (rand() % 4) {
            case 0:
                return {0,
                        rand() % height};
            case 1:
                return {this->width - 1,
                        rand() % height};
            case 2:
                return {rand() % width,
                        0};
            case 3:
                return {rand() % width,
                        this->height - 1};
        }
        return {0, 0};
    }

    Cage** Field::entry_exit_creat(Cage **arr) {
        srand(time(nullptr));
        while (!isCorrectDistStartFinish(start, finish)) {
            start = generateBorderPoint();
            finish = generateBorderPoint();
        }
```

```
        arr[start.getX()][start.getY()].setTypeCage(START);
        arr[finish.getX()][finish.getY()].setTypeCage(END);
        return arr;
    }

    //Cage**  Field::array_generator(unsigned  int  dim1,  unsigned  int
dim2) {
    //    Cage** ptrary = new Cage * [dim1];
    //    for (int i = 0; i < dim1; i++) {
    //        ptrary[i] = new Cage[dim2];
    //    }
    //    return ptrary;
    //}

    //void Field::array_destroyer(Cage **arr, unsigned int dim1) {
    //    for (int i = 0; i < dim1; i++) {
    //        delete [] arr[i];
    //    }
    //    delete [] arr;
    //}

    //Cage** Field::fill_array(Cage **arr) {
    //    for (int i = 0; i < height; i++) {
    //        for (int j = 0; j < width; j++) {
    //            arr[i][j] = Cage(i, j);
    //            arr[i][j].setTypeCage(TypeCage::EMPTY);
    //            arr[i][j].setTypeObj(TypeObj::NOTHING);
    //        }
    //    }
    //    return arr;
    //}
```

Название файла: FieldView.h

```
#include "Field.h"

class FieldView{
private:
    Field& grid;
public:
    explicit FieldView(Field& obj);
    void printField(Cage** board) const;
    void printFieldObj(Cage** board) const;
};
```

Название файла: FieldView.cpp

```
#include "FieldView.h"

FieldView::FieldView(Field &obj): grid(obj){}

void FieldView::printField(Cage** board) const{
    for (int i = 0; i < grid.height; i++) {
```

```
        for (int j = 0; j < grid.width; j++) {
            CageView tmpCage(board[i][j]);
            tmpCage.printCage();
        }
        std::cout << "\n";
    }
}

void FieldView::printFieldObj(Cage **board) const {
    for (int i = 0; i < grid.height; i++) {
        for (int j = 0; j < grid.width; j++) {
            CageView tmpCage(board[i][j]);
            tmpCage.printObj();
        }
        std::cout << "\n";
    }
}
```

Приложение Б

UML-диаграмма