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 игрового поля"

Студент гр. 9383 _____ Корсунов А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Написать программу в ООП стиле согласно заданию. Углубить знания об ООП, по возможности изучить предложенные паттерны.

Задание.

Написать класс игрового поля, которое представляет из себя прямоугольник (двумерный массив). Для каждого элемента поля должен быть создан класс клетки. Клетка должна отображать, является ли она проходимой, а также информацию о том, что на ней находится. Также, на поле должны быть две особые клетки: вход и выход.

При реализации поля запрещено использовать контейнеры из stl

Обязательные требования:

- Реализован класс поля
- Реализован класс клетки
- Для класса поля написаны конструкторы копирования и перемещения, а также операторы присваивания и перемещения
- Поле сохраняет инвариант - из любой клетки можно провести путь до любой другой
- Гарантированно отсутствует утечки памяти

Дополнительные требования:

- Поле создается с использованием паттерна **Синглтон**
- Для обхода по полю используется паттерн **Итератор**. Итератор должен быть совместим со стандартной библиотекой.

Выполнение работы.

Для выполнения работы использовалось библиотека SFML, предназначенная для работы с 2D графикой.

Класс Cell:

Данный класс является реализацией класса клетки:

он имеет три свойства (все булевого типа), где pass – проходимость клетки (в конструкторе класса она устанавливает в единицу — т. е. изначально все клетки — проходимые), out – является ли клетка выходом (в конструкторе устанавливается в нуль — не выход), in — является ли клетка входом (в конструкторе устанавливается в нуль — не вход). Предусмотрен деструктор, а также все get-ры и set-ры (методы, с помощью которых можно менять свойства класса и получать их);

Класс Field:

Данный класс является реализацией класса поля и является «родителем» класса клетки. Класс имеет 4 свойства:

ptr – указатель на двумерный массив клеток (изначально ссылается на нулевой указатель), width и height – количество строк и столбцов в массиве, object – указатель на объект класса Field (его можно будет получить при вызове метода get_Field, который обеспечивает существование только одного поля, что подразумевает под собой использование паттерна Синглтон), а также методы, для изменения свойства клетки. Согласно обязательным условиям, написаны конструкторы копирования, перемещения и их операторы присваивания, а также деструктор. К тому же, здесь объявляется о существование дружественного класса drow (он нужен для реализации SFML-составляющей).

Класс drow:

Данный класс служит для рисования уже созданного поля с помощью библиотеки SFML:

32 — размер клетки, gradView – массив из 11 значений (на деле нужно, конечно, всего 4, но, возможно, другие пригодятся при выполнения следующих заданий). Сами же значения есть части картинок их загружаемого спрайта. Картинка рисуется путем наложения частей из спрайта (что удобно, ведь спрайт один). Значение 10 (выпухлая клетка) — проходимая клетка, 0 —

вход (обычная клетка), 11 — выход (флажок), 9 — непроходимая клетка (бомба). Иконки взяты из классической игры «Сапер».

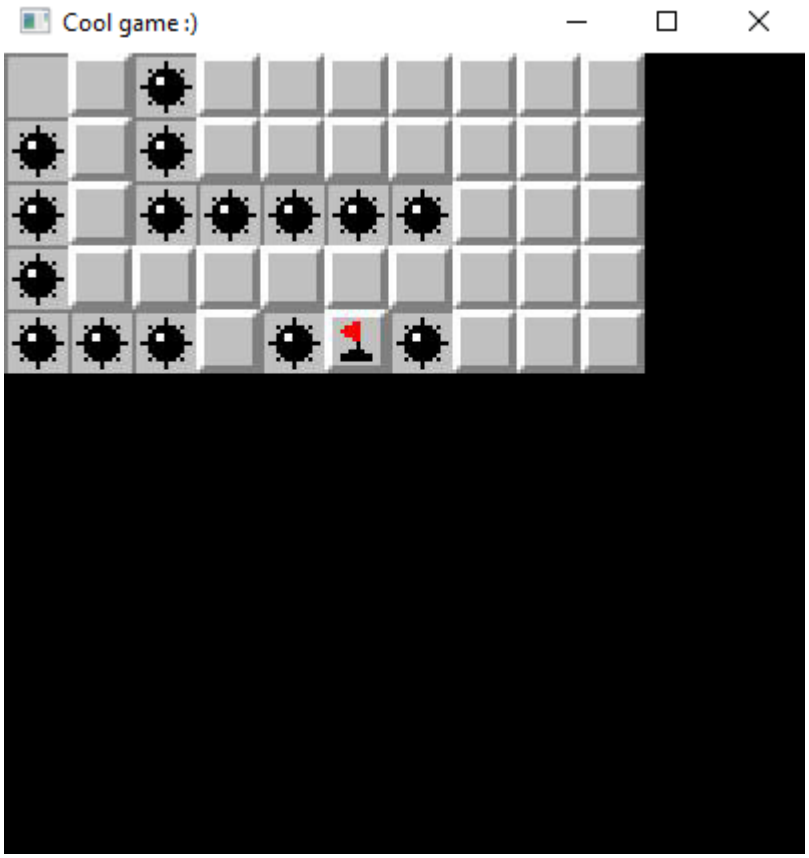

Рисунок 1.

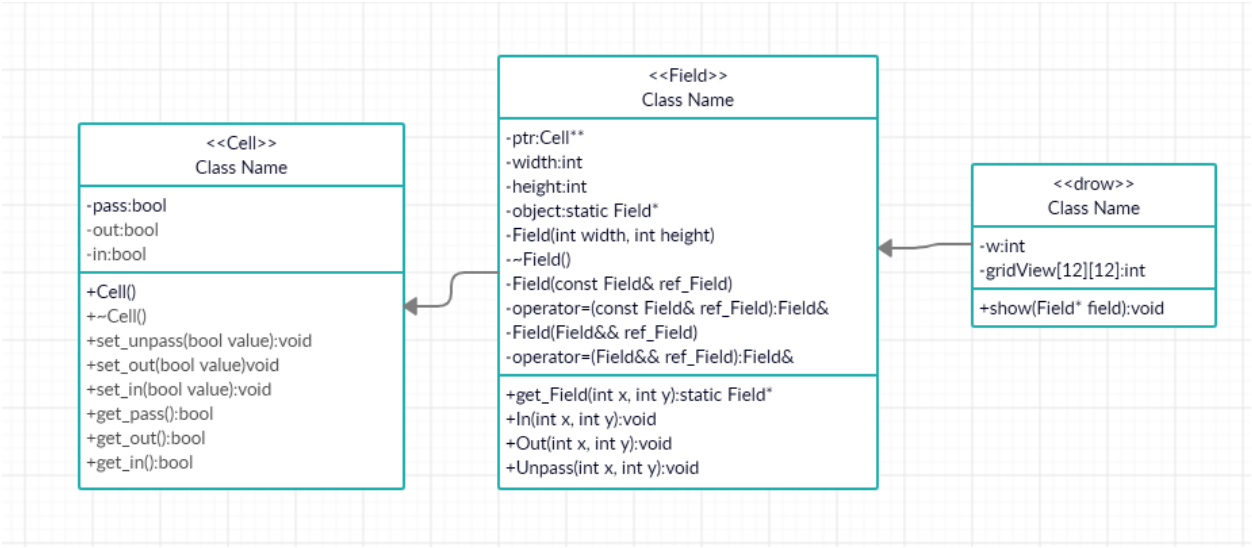

Рисунок 2.

На рисунке 2 изображена UML-диаграмма классов, реализованных в данной работе.

Выводы.

Написана программа в ООП стиле согласно заданию. Углубилины знания об ООП, реализованы некоторые из паттернов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include "Cell.h"
#include "Field.h"
#include "drow.cpp"

using namespace std;

int main()
{
 Field* field = Field::get_Field(10, 5);
field->In(0, 0);
field->Out(5, 4);
field->Unpass(0,1);
field->Unpass(0,2);
field->Unpass(0,3);
field->Unpass(0,4);
field->Unpass(1,4);
field->Unpass(2,4);
field->Unpass(4,4);
field->Unpass(2,0);
field->Unpass(2,1);
field->Unpass(2,2);
field->Unpass(3,2);
field->Unpass(4,2);
field->Unpass(5,2);
field->Unpass(6,2);
```

```
field->Unpass(6,4);
drow go;
go.show(field);
   return 0;
}
```

Файл Cell.h

```
#pragma once

class Cell
{
private:
   bool pass;
   bool out;
   bool in;

public:
   Cell();
   ~Cell();
   void set_unpass(bool value);
   void set_out(bool value);
   void set_in(bool value);
   bool get_pass();
   bool get_out();
   bool get_in();
};
```

Файл Cell.cpp

```
#include "Cell.h"

Cell::Cell()
```

```
{
    this->pass = true;
    this->in = false;
    this->out = false;

}

Cell::~Cell(){};

void Cell::set_unpass(bool val)
{
    this->pass = false;
    this->in = false;
    this->out = false;
}

void Cell::set_in(bool val)
{
    this->pass = true;
    this->in = true;
    this->out = false;
}

void Cell::set_out(bool val)
{
    this->pass = true;
    this->in = false;
    this->out = true;
}
```

```
bool Cell::get_pass()
{
   return this->pass;
}

bool Cell::get_in()
{
   return this->in;
}

bool Cell::get_out()
{
   return this->out;
}
```

Файл Field.h

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

class Field
{
private:
   Cell** ptr = nullptr;
   int width, height;
   static Field* object;

   Field(int width, int height);
   ~Field();
   Field(const Field& ref_Field);
```

```
    Field& operator=(const Field& ref_Field);
    Field(Field&& ref_Field);
    Field& operator=(Field&& ref_Field);

public:
    static Field* get_Field(int x, int y);
    void In(int x, int y);
    void Out(int x, int y);
    void Unpass(int x, int y);
friend class drow;
};
```

Файл Field.cpp

```
#include "Field.h"

Field* Field::object = nullptr;

Field::Field(int x, int y) : width(x), height(y)
{
    this->ptr = new Cell* [this->width];
    for (int i = 0; i < this->width; i++)
    {
        this->ptr[i] = new Cell [this->height];
    }
}

Field::~Field()
{
    for (int i = 0; i < this->width; i++)
    {
```

```
        delete[] this->ptr[i];
    }
    delete[] this->ptr;
}

Field* Field::get_Field(int x, int y)
{
    object = new Field(x, y);
    return object;
}

void Field::In(int x, int y)
{
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)
    {
        this->ptr[x][y].set_in(1);
    }
}

void Field::Out(int x, int y)
{
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)
    {
        this->ptr[x][y].set_out(1);
    }
}

void Field::Unpass(int x, int y)
{
```

```
    if (x >= 0 && x < this->width && y >= 0 && y < this->height)
    {
        this->ptr[x][y].set_unpass(0);
    }
}

Field::Field(const Field& ref_Field)
{
   this->width = ref_Field.width;
   this->height = ref_Field.height;
   this->ptr = new Cell* [ref_Field.width];
   for (int i = 0; i < ref_Field.width; i++)
   {
      this->ptr[i] = new Cell[ref_Field.height];
      for (int j = 0; j < ref_Field.height; j++)
      {
         this->ptr[i][j] = ref_Field.ptr[i][j];
      }
   }
}

Field& Field::operator=(const Field& ref_Field)
{
   if (&ref_Field == this)
   {
      return *this;
   }

   if (this != &ref_Field){
```

```
        for (int i = 0; i < this->width; i++)
        {
            delete[] this->ptr[i];
        }
        delete[] this->ptr;
    }

    this->width = ref_Field.width;
    this->height = ref_Field.height;
    this->ptr = new Cell* [ref_Field.width];
    for (int i = 0; i < ref_Field.width; i++)
    {
        this->ptr[i] = new Cell[ref_Field.height];
        for (int j = 0; j < ref_Field.height; j++)
        {
            this->ptr[i][j] = ref_Field.ptr[i][j];
        }
    }
    return *this;
}

Field::Field(Field&& ref_Field)
{
    this->ptr = ref_Field.ptr;
    this->width = ref_Field.width;
    this->height = ref_Field.height;
    ref_Field.ptr = nullptr;
    ref_Field.width = 0;
    ref_Field.height = 0;
```

```
}

Field& Field::operator=(Field&& ref_Field)
{
   if (&ref_Field == this)
   {
      return *this;
   }

   if (this != &ref_Field){
      for (int i = 0; i < this->width; i++)
      {
         delete[] this->ptr[i];
      }
      delete[] this->ptr;
   }

   this->ptr = ref_Field.ptr;
   this->width = ref_Field.width;
   this->height = ref_Field.height;
   ref_Field.ptr = nullptr;
   ref_Field.width = 0;
   ref_Field.height = 0;
   return *this;
}
```

Файл drow.cpp

```
#pragma once
#include "Field.h"
#include <SFML/Graphics.hpp>
```

```
using namespace sf;

class drow
{
private:
   int w = 32;
   int gridView[12][12];

public:
   void show(Field* field)
   {
      RenderWindow app(VideoMode(400, 400), "Cool game :)");
      for (int i = 0; i < 12; i++)
         for (int j = 0; j < 12 ; j++)
      {
         gridView[i][j] = 10;
      }

      Texture t;
      t.loadFromFile("C:/Users/Eldorado/Documents/qwe/oop/govno/titles.jp
g");
      Sprite s(t);

      for (int i = 0; i < field->width; i++)
      {
        for (int j = 0; j < field->height; j++)
        {
```

```
                    if (!field->ptr[i][j].get_in() && !field->ptr[i][j].get_out() &&
field->ptr[i][j].get_pass())
                {
                    s.setTextureRect(IntRect(gridView[i][j] * w, 0, w, w));
                    s.setPosition(i*w, j*w);
                    app.draw(s);
                    //проходимая
                }
                        if (field->ptr[i][j].get_in())
                {
                    s.setTextureRect(IntRect(0 * w, 0, w, w));
                    s.setPosition(i*w, j*w);
                    app.draw(s);
                    //вход
                }
                        if (field->ptr[i][j].get_out())
                {
                    s.setTextureRect(IntRect(11 * w, 0, w, w));
                    s.setPosition(i*w, j*w);
                    app.draw(s);
                    //выход
                }
                        if (!field->ptr[i][j].get_pass())
                {
                    s.setTextureRect(IntRect(9 * w, 0, w, w));
                    s.setPosition(i*w, j*w);
                    app.draw(s);
                    //непроходимая
                }
```

```
        }
    }
    app.display();
    while (app.isOpen())
    {
        Event e;

        while(app.pollEvent(e))
        {
            if (e.type == Event::Closed)
                app.close();
        }
    }

  }


};
```