МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ООП»

Тема: Создание игрового поля

Студент гр. 9383 Моисейченко К.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Научиться работать с классами, с методами, полями и конструкторами классов. Создать игровое поле на базе языка C++.

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

Выполнение работы.

Класс Cell.

Реализация класса клетки. Для класса Cell был реализован enum class CellType, в котором перечислены все возможные типы клеток. Также был создан пустой класс GameObject.

Поля класса:

- CellType cellType - хранит тип объекта клетки

- GameObject* objects - хранит массив игровых объектов, находящихся на клетке

Методы класса:

+ Cell() - конструктор инициализирует клетку, как проходимую

+ ~Cell() - деструктор освобождает выделенную динамически память для массива игровых объектов

+ GameObject* GetObjects() - возвращает указатель на массив игровых объектов

+ CellType GetCellType() - возвращает тип клетки

+ void SetCellType(CellType cellType) - устанавливает переданный тип клетки

Класс Field.

Реализация класса игрового поля.

Поля класса:

- Static Field* instance - статический указатель на объект для Синглтон

- int height - высота поля

- int width - ширина поля

- Cell** field - массив клеток поля

Методы класса:

- Field() - конструктор класса, инициализирует игровое поле, его высоту и ширину

- Field(const Field& other) - перегрузка конструктора присваивания

- Field(Field&& other) - перегрузка конструктора присваивания с перемещением

- Field& operator = (const Field& other) - перегрузка оператора присваивания

- Field& operator = (Field&& other) - перегрузка оператора присваивания с перемещением

- void cleanCells() - очищает динамически выделенную память под массив клеток

+ int GetHeight() - возвращает высоту поля

+ int GetWidth() - возвращает ширину поля

+ Cell** GetField() - возвращает указатель на массив клеток поля

+ void PrintField() - выводит поле на экран

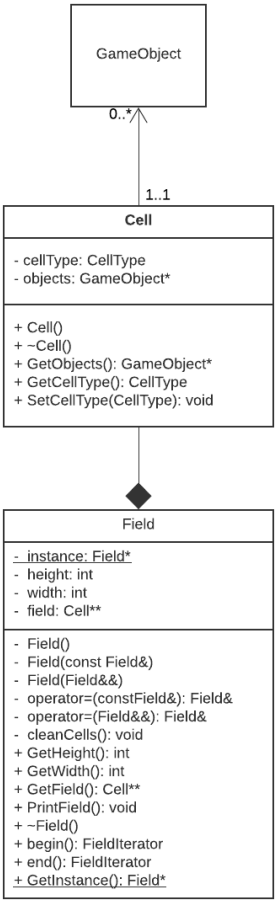
+ ~Field() - очищает память, выделенную под instance. Запускает метод cleanCells()

+ static Field* GetInstance() - при первом вызове создание объекта класса и возврат указателя на объект, при последующих - только возврат указателя на объект (паттерн Синглтон)

Разработанный программный код см. в приложении А.

Рисунок 1 - Результат работы программы

UML-диаграмма.



Выводы.

Были получены знания о работе с классами на языке C++. Был изучен паттерн Синглтон. Создан алгоритм с классами реализации игрового поля.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include "cell.h"
#include "field.h"
#include "gameObject.h"

int main() {
    Field* gameField = Field::GetInstance();
    gameField->PrintField();
    return 0;
}
```

Название файла: cell.h

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

enum class CellType
{
    PATH,
    WALL,
    START,
    END
};

class Cell
{
private:
    CellType cellType;
    GameObject* objects;
public:
    Cell() : cellType(CellType::PATH)
    {
        objects = new GameObject[4];
    }
    ~Cell()
    {
        delete[] objects;
    }
    GameObject* GetObjects();
    CellType GetCellType();
    void SetCellType(CellType cellType);
};
```

Название файла: cell.cpp

```
#include "cell.h"

GameObject* Cell::GetObjects()
{
    return objects;
}

CellType Cell::GetCellType()
```

```
{
    return cellType;
}

void Cell::SetCellType(CellType cellType)
{
    this->cellType = cellType;
}
```

Название файла: field.h

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

class Field
{
private:
    static Field* instance;
    int height;
    int width;
    Cell** field = nullptr;

    Field();
    Field(const Field& other);
    Field(Field&& other);

    Field& operator = (const Field& other);
    Field& operator = (Field&& other);

    void cleanCells();
public:
    int GetHeight();
    int GetWidth();
    Cell** GetField();
    void PrintField();
    ~Field();
    static Field* GetInstance();
};
```

Название файла: field.cpp

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

Field* Field::GetInstance()
{
    if(instance == nullptr)
        instance = new Field;
    return instance;
}

Field::Field()
{
    this->height = 22;
    this->width = 19;
    if (!field)
    {
        field = new Cell * [height];
```

```
            for (int i = 0; i < height; i++)
            {
                field[i] = new Cell[width];
            }

            int                     myField[22][19]                     =
{{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},

{1,0,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,0,1},

{1,0,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,0,1},

{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},

{1,0,1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,0,1},

{1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1},

{1,1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,1},

{1,1,1,1,0,1,0,0,0,0,0,0,0,1,0,1,1,1,1},

{1,1,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,1,1},

{2,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,3},

{1,1,1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,1,1},

{1,1,1,1,0,1,0,0,0,0,0,0,0,1,0,1,1,1,1},

{1,1,1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,1,1},

{1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},

{1,0,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,0,1},

{1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1},

{1,1,0,1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,1},

{1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1},

{1,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1},

{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}};

            for(int i = 0; i < height; i++)
            {
                for(int j = 0; j < width; j++)
                {
                    if(myField[i][j] == 0)
                        field[i][j].SetCellType(CellType::PATH);
                    if(myField[i][j] == 1)
                        field[i][j].SetCellType(CellType::WALL);
```

```
                if(myField[i][j] == 2)
                    field[i][j].SetCellType(CellType::START);
                if(myField[i][j] == 3)
                    field[i][j].SetCellType(CellType::END);
            }
        }
    }
}

int Field::GetHeight()
{
    return height;
}
int Field::GetWidth()
{
    return width;
}

Cell** Field::GetField()
{
    return field;
}

void Field::PrintField()
{
    for(int i = 0; i < height; i++)
    {
        for(int j = 0; j < width; j++)
        {
            if(field[i][j].GetCellType() == CellType::START)
                std::cout << "SS";
            else if(field[i][j].GetCellType() == CellType::END)
                std::cout << "EE";
            else if(field[i][j].GetCellType() == CellType::PATH)
                std::cout << "  ";
            else
                std::cout << "||";
        }
        std::cout << "\n";
    }
}

Field::~Field()
{
    delete[] instance;
    cleanCells();
}

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

```
            this->field[i][j] = other.field[i][j];
        }
    }
}

Field::Field(Field&& other)
{
    this->width = other.width;
    this->height = other.height;
    this->field = other.field;
    other.field = nullptr;
}

Field& Field::operator=(const Field& other)
{
    if (&other == this)
        return *this;
    cleanCells();
    this->width = other.width;
    this->height = other.height;
    this->field = new Cell*[height];
    for (int i = 0; i < height; i++)
    {
        this->field[i] = new Cell[width];
        for (int j = 0; j < width; j++)
        {
            this->field[i][j] = other.field[i][j];
        }
    }
    return *this;
}

Field& Field::operator=(Field&& other)
{
    cleanCells();
    this->width = other.width;
    this->height = other.height;
    this->field = other.field;
    return *this;
}

void Field::cleanCells()
{
    if (field)
    {
        for (int i = 0; i < height; i++)
        {
            delete[] field[i];
        }
        delete[] field;
    }
}

Field* Field::instance = nullptr;
```

Название файла: gameObject.h

```
#pragma once
```

```
class GameObject
{

};
```

Название файла: gameObject.cpp

```
#include "gameObject.h"
```