

**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. 9382

Демин В.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

### **Цель работы.**

Создать класс поля и класс клетки. Реализация на языке «чистом» C++, полностью в ООП стиле.

### **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 `std`*

### **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.**

### **class Cell**

Класс Cell представляет собой клеткой поля. Содержит в себе такие поля как:

- 1) x, y – координаты на игровом поле. Будут необходимы для проверок клетки
- 2) type является переменной перечисления TypeCell. Представляет собой тип клетки, необходим для отрисовки и дальнейших взаимодействий в игре.
- 3) также хранит указатели на следующую и предыдущую ячейку.

Имеет следующие методы:

- 1) SetCord – задает координаты клетки
- 2) SetType – задает тип клетки
- 3) GetType – возвращает тип клетки
- 4) GetCordX() и GetCordY() – возвращают соответствующий координаты

### **class Field**

Класс Field является игровым полем, который содержит клетки. К классу применен паттерн Синглтон. Что гарантирует нам что экземпляр класса будет единственный. Содержит в себе такие поля:

- 1) size\_x и size\_y – размеры поля
- 2) cells – двойной массив клеток. Представление самого поля.

Содержит такие методы:

- 1) ~Field – деструктор класса, в котором освобождается выделенная память.
- 2) Конструкторы копирования и перемещения, и их операторы

3)GetInstance – является частью паттерна Синглтон. Возвращает указатель на статический экземпляр класса.

4)GetCells,GetSizeX,GetSizeY – геттеры приватных переменных класса

5) MakeListCells – метод связывающий клетки в список

### **Class Game**

Данный класс является главным инструментом для взаимодействия с игрой. Хранит в себе игровое поле field. Также в классе используется паттерн Синглтон.

Содержит такие методы:

1)~ Game – деструктор класса, в котором освобождается выделенная память.

2)Конструкторы копирования и перемещения, и их операторы удалены

3)GetInstance – является частью паттерна Синглтон. Возвращает указатель на статический экземпляр класса.

4) play – запускает игру, с выбором отрисовки

5) PrintConsole – прорисовка игрового поля в консоли

### **Class CellIterator**

Данный класс является инструментом для доступа к элементам коллекции CellColection, которая содержит в себе клетки игрового поля.

Содержит следующие поля класса:

1) temp – указатель на ячейку, необходим в методах

2)head – указатель на начало списка

Содержит в себе следующие методы:

1) getElem получает элемент temp

Рисунок 1. Uml диаграмма

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

[illegible]

## **Выводы.**

В ходе выполнения лабораторной работы были созданы класс клетки и класс поля, изучен ООП стиль.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include "Game/Game.h"

int main() {
    Game* game=Game::GetInstance();
    return 0;
}
```

Название файла: Cell.h

```
//
// Created by vikto on 13.09.2020.
//
```

```
#ifndef MYGAME_CELL_H
#define MYGAME_CELL_H
enum TypeCell{
    HWALL,
    VWALL,
    CWALL,
    EMPTY,
    ENTRANCE,
    EXIT
};
```

```
class Cell {

    int x;
    int y;
    TypeCell type;
public:
    Cell();

    void SetCord(int x, int y);
    void SetType(TypeCell typeCell);
    TypeCell GetType();
    int GetCordX() const;
    int GetCordY() const;
```



```
Cell *next;  
Cell *prev;
```

```
    //список  
};
```

```
#endif //MYGAME_CELL_H  
Название файла: Cell.cpp  
//  
// Created by vikto on 13.09.2020.  
//
```

```
#include "Cell.h"
```

```
Cell::Cell() {  
    type = EMPTY;  
    next = nullptr;  
    prev = nullptr;  
    x = 0;  
    y = 0;  
}
```

```
void Cell::SetCord(int x, int y) {  
    this->x = x;  
    this->y = y;  
}
```

```
void Cell::SetType(TypeCell typeCell) {  
    this->type = typeCell;  
}
```

```
TypeCell Cell::GetType() {  
    return type;  
}
```

```
int Cell::GetCordY() const {  
    return y;  
}
```

```
int Cell::GetCordX() const {  
    return x;  
}
```

Название файла: Feild.h  
//  
// Created by vikto on 13.09.2020.  
//

*#ifndef MYGAME\_FIELD\_H*  
*#define MYGAME\_FIELD\_H*

*#include "Cell/Cell.h"*

*class Field {*  
    *Cell \*\*cells;*  
    *int size\_x;*  
    *int size\_y;*

*protected:*

*Field();*

*static Field \*field\_;*

*public:*

*~Field();*

*Field(const Field &obj);*

*Field(Field &&obj);*

*//Field(Field &other) = delete;*

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

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

*static Field \*GetInstance();*

*Cell \*\*GetCells() const;*

*int GetSizeX() const;*

*int GetSizeY() const;*

*private:*

*void MakeListCells(Cell \*\*cells\_for\_list) const;*

*};*

*#endif//MYGAME\_FIELD\_H*

Название файла: Feild.cpp

*//*

*// Created by vikto on 13.09.2020.*

*//*

*#include "Field.h"*

*Field \*Field::field\_ = nullptr;*

*Field \*Field::GetInstance() {*

*if (field\_ == nullptr) {*

*field\_ = new Field();*

*}*

*return field\_;*

*}*

*Field::Field() {*

*size\_x = 100;*

*size\_y = 10;*

*//сделать границы поля, вход и выход*

*cells = new Cell \*[size\_y];*

*for (int i = 0; i < size\_y; i++) {*

*cells[i] = new Cell[size\_x];*

*}*

*for (int i = 0; i < size\_y; ++i) {*

*for (int j = 0; j < size\_x; ++j) {*

*cells[i][j].SetCord(j, i);*

*if (i == 0 || i == size\_y - 1) {*

*cells[i][j].SetType(HWALL);*

*}*

*if (j == 0 || j == size\_x - 1) {*

*cells[i][j].SetType(VWALL);*

*}*

*if (i == 0 && j == 0 || i == 0 && j == size\_x - 1 || j == 0 && i ==*

*size\_y - 1 ||*

```

        j == size_x - 1 && i == size_y - 1) {
            cells[i][j].SetType(CWALL);
        }
        if (j == 0 && i == size_y / 2) {
            cells[i][j].SetType(ENTRANCE);
        }

        if (j == size_x - 1 && i == size_y / 2) {
            cells[i][j].SetType(EXIT);
        }
    }
}

MakeListCells(cells);
}

Field::~Field() {
    for (int i = 0; i < size_y; ++i) {
        delete cells[i];
    }
    delete[] cells;
    delete field_;
}

Cell **Field::GetCells() const {
    return cells;
}

int Field::GetSizeX() const {
    return size_x;
};

int Field::GetSizeY() const {
    return size_y;
}

Field::Field(const Field &obj) {
    this->size_x=obj.size_x;
    this->size_y=obj.size_y;
    this->cells = new Cell *[size_y];
    for (int i = 0; i < this->size_y; i++) {
        this->cells[i] = new Cell[size_x];
    }
}

```

```

    MakeListCells(this->cells);

    for (int i = 0; i < size_y; ++i) {
        for (int j = 0; j < this->size_x; ++j) {
            this->cells[i][j].SetType(obj.cells[i][j].GetType());
            this->cells[i][j].SetCord(obj.cells[i][j].GetCordX(),obj.cells[i][j].GetCordY());
        }
    }
}

```

```

Field::Field(Field &&obj) {

    this->size_x=obj.size_x;
    this->size_y=obj.size_y;

    this->cells=obj.cells;
    obj.cells= nullptr;

    obj.cells= nullptr;
    obj.size_y=0;
    obj.size_x=0;
}
Field &Field::operator=(const Field & obj) {
    if(&obj == this)
        return *this;

    this->size_x=obj.size_y;
    this->size_x=obj.size_x;

    for (int i = 0; i < this->size_x; i++){
        delete [] this->cells[i];
    }
    delete [] this->cells;

    this->cells = new Cell *[size_y];
    for (int i = 0; i < this->size_y; i++) {
        this->cells[i] = new Cell[size_x];
    }

    MakeListCells(this->cells);
}

```

```

    for (int i = 0; i < this->size_y; ++i) {
        for (int j = 0; j < this->size_x; ++j) {
            this->cells[i][j].SetType(obj.cells[i][j].GetType());
            this->cells[i][j].SetCord(obj.cells[i][j].GetCordX(),obj.cells[i][j].GetCordY());
        }
    }

```

```

    return *this;
}

```

```

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

```

```

    this->size_x=obj.size_y;
    this->size_y=obj.size_x;

```

```

    for (int i = 0; i < this->size_x; i++){
        delete [] this->cells[i];
    }
    delete [] this->cells;

```

```

    this->cells=obj.cells;
    obj.cells= nullptr;
    obj.size_x=0;
    obj.size_y=0;

```

```

    return *this;
}

```

```

void Field::MakeListCells(Cell **cells_for_list) const {
    Cell *p_temp = nullptr;

```

```

    for (int i = 0; i < size_y; ++i) {
        for (int j = 0; j < size_x; ++j) {
            cells_for_list[i][j].prev = p_temp;
            if (j != size_x - 1) {
                cells_for_list[i][j].next = &cells_for_list[i][j + 1];
            }
            if (j == size_x - 1 && i != size_y - 1) {
                cells_for_list[i][j].next = &cells_for_list[i + 1][0];
            }

```

```

    }
    p_temp = &cells_for_list[i][j];
  }
}
};

```

Название файла: Iterator.h

```

//
// Created by vikto on 13.09.2020.
//

```

```

#ifndef MYGAME_ITERATOR_H
#define MYGAME_ITERATOR_H
#include "../Game/Field/Cell/Cell.h"

```

```

template <class U>
class Iterator {
public:
    virtual U *Next()=0;
    virtual U *begin()=0;
    virtual U *getElem()=0;
    virtual U *Prev()=0;
    virtual bool hasMore()=0;
};

```

```

class CellIterator: public Iterator<Cell> {
//методы для iteratinga
    Cell* temp;
    Cell* head;
public:
    CellIterator(Cell &temp);
    Cell * getElem() override;
    Cell* begin() override;
    Cell* Next() override;
    Cell* Prev() override;
    bool hasMore() override;
};

```

```

#endif //MYGAME_ITERATOR_H

```

Название файла: Iterator.cpp

```

//

```

*// Created by vikto on 14.09.2020.*

*//*

*#include "Iterator.h"*

*CellIterator::CellIterator(Cell &temp):temp(&temp) {*

*head=this->temp;*

*}*

*//ничего не сделано*

*Cell\* CellIterator::Next() {*

*temp=temp->next;*

*return temp;*

*}*

*bool CellIterator::hasMore() {*

*if(temp== nullptr){*

*return false;*

*}else{*

*return true;*

*}*

*}*

*Cell\* CellIterator::Prev() {*

*temp=temp->prev;*

*return temp;*

*}*

*Cell \*CellIterator::begin() {*

*temp=head;*

*return head;*

*}*

*Cell \*CellIterator::getElem() {*

*return temp;*

*}*

Название файла: Game.h

*//*

*// Created by vikto on 13.09.2020.*

*//*

*#include <iostream>*

*#include "Field/Field.h"*

*#ifndef MYGAME\_GAME\_H*

*#define MYGAME\_GAME\_H*



```

class Game {
    ~Game();
protected:
    Game();
    static Game* game_;

public:
    void play();
    void PrintConsole();
    Game(Game &other) = delete;
    void operator=(const Game &) = delete;
    static Game *GetInstance();

private:
    Field* field;
};

```

```

#endif //MYGAME_GAME_H

```

```

#endif //OOP_DISPLAY_H

```

Название файла: Game.cpp

```

//
// Created by vikto on 13.09.2020.
//

```

```

#include "Game.h"
#include "../Tools/MyCollections/collections.h"

```

```

Game *Game::game_ = nullptr;

```

```

Game::Game() {
    field = Field::GetInstance();
    play();
}

```

```

Game *Game::GetInstance() {
    if (game_ == nullptr) {
        game_ = new Game();
    }
    return game_;
}

```

```
}
```

```
void Game::play() {  
    int mode = 0;  
    std::cout << "Select mode: 1-ConsoleMode\n";  
    std::cin >> mode;  
    if (mode == 1) PrintConsole();  
}
```

```
void Game::PrintConsole() {  
    //использование итератора  
    int size_x = field->GetSizeX();  
    int size_y = field->GetSizeY();  
  
    Cell **cells = field->GetCells();  
    //without Iterator  
    /*  
    for (int i = 0; i < size_y; ++i) {  
        for (int j = 0; j < size_x; ++j) {  
            if (cells[i][j].GetType() == EMPTY) {  
                std::cout << " ";  
            }  
            if (cells[i][j].GetType() == HWALL) {  
                std::cout << "_";  
            }  
            if (cells[i][j].GetType() == VWALL || cells[i][j].GetType() == CWALL)  
                std::cout << "|";  
        }  
  
        if (cells[i][j].GetType() == EXIT) {  
            std::cout << "$";  
        }  
        if (cells[i][j].GetType() == ENTRANCE) {  
            std::cout << "#";  
        }  
    }  
    std::cout << "\n";  
    */  
    //with Iterator  
    CellCollection cont(cells);  
  
    Iterator<Cell> *it = cont.CreateIterator();  
    for (it->begin(); it->hasMore(); it->Next()) {
```

```

        if (it->getElem()->GetType() == EMPTY) {
            std::cout << " ";
        }
        if (it->getElem()->GetType() == HWALL) {
            std::cout << "_";
        }
        if (it->getElem()->GetType() == VWALL // it->getElem()->GetType() ==
CWALL) {
            std::cout << "/";
        }

        if (it->getElem()->GetType() == EXIT) {
            std::cout << "$";
        }
        if (it->getElem()->GetType() == ENTRANCE) {
            std::cout << "#";
        }
        if (it->getElem()->GetCordX() == size_x - 1) {
            std::cout << "\n";
        }
    }
}

```

```

Game::~~Game() {
    delete field;
    delete game_;
}

```

Название файла: collections.cpp

```

//
// Created by vikto on 13.09.2020.
//

```

```

#include "collections.h"

```

```

CellCollection::CellCollection(Cell** cells) {
    this->cells=cells;
}

```

```

Iterator<Cell> *CellCollection::CreateIterator() {
    CellIterator* a = new CellIterator(cells[0][0]);
    return a;
}

```

Название файла: collections.h

```
//  
// Created by vikto on 13.09.2020.  
//  
  
#ifndef MYGAME_MYCOLLECTIONS_H  
#define MYGAME_MYCOLLECTIONS_H  
#include "../Game/Field/Cell/Cell.h"  
#include "../Iterator/Iterator.h"  
  
template <class U>  
class Collection {  
    virtual Iterator<U> *CreateIterator()=0;  
};  
  
class CellCollection: public Collection<class Cell>{  
    //хранит коллекцию  
    Cell** cells;  
public:  
    CellCollection(Cell** cells);  
    Iterator<Cell> *CreateIterator() override;  
  
};  
  
#endif //MYGAME_MYCOLLECTIONS_H
```