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. 9304 _____ Тиняков С.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Научиться создавать классы на языке программирования 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
- Для обхода по полю используется паттерн Итератор

Выполнение работы.

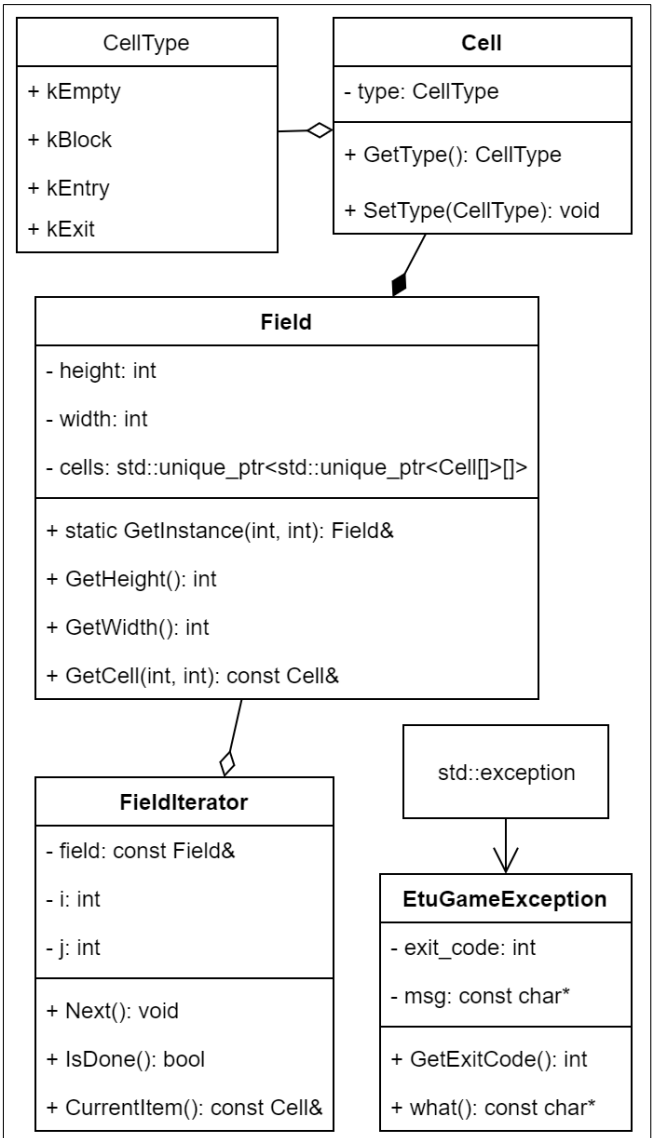

Рисунок 1 — UML-диграмма

Перечисление *CellType* отвечает за состояние клетки. Существует четыре состояния: клетка пуста — *kEmpty*, клетка не проходима — *kBlock*, клетка входа — *kEntry*, клетка выхода — *kExit*.

Класс *Cell* является сруктурной единицей поля. Клетка имеет тип(состояние) — *type*. Данное поле является приватным, поэтому для взаимодействия с ним созданы публичные методы *GetType* и *SetType.*

Класс *Field* реализован при помощи шаблона Синглтон. Поля *width* и *height* хранят размеры поля. Поле состоит из клеток. Для этого был создан двумерный динамический массив с использованием умных указателей. Для

обращения к приватным полям созданны публичные методы *GetWidth* и *GetHeight*, а также *GetCell*, который возвращает ссылку на константный экземпляр класса *Cell*. Обращение происходит по двойному индексу: по высоте и длине. Метод *GetInstance* создаёт единственный экземпляр класса *Field* с заданными размерами и возвращает ссылку на него. При повторном вызове метода возвращает ссылку на уже созданный экземпляр. Также были реализованны приватные конструкторы копирования и перемещения. Если производится попытка создания поля с неправильными размерами(нулевыми или отрицательными), то выбрасывается исключение *EtuGameException*.

Класс *EtuGameException* наследуется от стандартного класса исключений — *std::exception*. Метод *what* возвращает сообщение исключения, метод *GetExitCode* возвращает код исключения.

Для обхода по полю был создан класс *FieldIterator* с использованием шаблона Итератор. В приватных полях *i* и *j* хранятся индексы по высоте и длине текущего элемента. Метод *Next* переводит индексы на следующий элемент. Метод *IsDone* сообщает закончился ли обход. Если вызвать метод *Next*, когда обход закончен, то будет выбрашено исключение *EtuGameException*. Метод *CurrentItem* возвращает ссыслку на текущую константную клетку.

Для проверки работоспособности классов был создан *unittest*. Его запуск и завершение без ошибок показывает проверку классов.

Разработанный программный код см. в приложении А.

Выводы.

Научились создавать классы на языке программирования C++.

Были реализованы классы *Cell, Field, FieldIterator, EtuGameException,* а также перечисление *CellType.* Класс *Field* был создан с использованием шаблона Синглтон. Класс *FieldIterator* был создан с использованием шаблона Итератор. Для проверки реализованных классов был создан *unittest*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: include/etu_game/objects/cell.h

```
#ifndef CELL_CLASS_H
#define CELL_CLASS_H

#include "../types/cell_type.h"

namespace etu_game{
namespace objects{

class Cell{
private:
    types::CellType type;
public:
    Cell();
    Cell(const Cell& cell);
    Cell& operator=(const Cell& cell);
    Cell(types::CellType);
    ~Cell();
    types::CellType GetType() const;
    void SetType(types::CellType);
};

} // namespace objects
} // namespace etu_game

#endif //CELL_CLASS_H.
```

Название файла: include/etu_game/objects/field.h

```
#ifndef FIELD_CLASS_H
#define FIELD_CLASS_H

#include "cell.h"
#include "../types/etu_game_exception.h"

#include <memory>

namespace etu_game {

namespace objects {
class Field{
private:
    int height, width;
    std::unique_ptr<std::unique_ptr<Cell[]>[]> cells;
    Field(int height, int width);
```

```
    //copy
    Field(const Field& field);
    //oper copy
    Field& operator=(const Field& field);
    //move
    Field(Field&& field);
    //oper move
    Field& operator=(Field&& field);

    // TODO: method for changing field
public:

    ~Field();

    static Field& GetInstance(int height, int width);

    int GetWidth() const;

    int GetHeight() const;

    // FIXME: may be do through friend for other classes
    const Cell& GetCell(int h_pos, int w_pos);

    // TODO: loading cells-map from some class called "Map"


    bool CheckInvariant();

    friend class FieldIterator;
};

} // objects
} // etu_game

#endif // FIELD_CLASS_H
```

Название файла: include/etu_game/objects/field_iterator.h

```
#ifndef FIELD_ITERATOR_CLASS_H
#define FIELD_ITERATOR_CLASS_H

#include "field.h"
#include "../types/etu_game_exception.h"

#include <memory>

namespace etu_game {

namespace objects {

class FieldIterator{
private:
    const Field& field;
```

```
    int i, j;
public:
    FieldIterator(const Field& f);

    void Next();

    void operator++();

    void operator++(int);

    bool IsDone();

    bool operator()();

    const Cell& CurrentItem();

    const Cell& operator*();
};

} // objects
} // etu_game

#endif // FIELD_ITERATOR_CLASS_H
```

Название файла: include/etu_game/types/cell_type.h

```
#ifndef CELL_TYPE_H
#define CELL_TYPE_H

namespace etu_game{

namespace types{

enum CellType{
    kEmpty,
    kBlock,
    kEntry,
    kExit,
};

} // types
} // etu_game

#endif // CELL_TYPE_H
```

Название файла: include/etu_game/types/etu_game_exception.h

```
#ifndef ETU_GAME_EXCEPTION_CLASS_H
#define ETU_GAME_EXCEPTION_CLASS_H
```

```
#include<string>

namespace etu_game{

namespace types{

class EtuGameException: std::exception{
private:
    int exit_code;
    const char* msg;
public:
    EtuGameException(int code, const char* message);
    ~EtuGameException();
    int GetExitCode();
    const char* what() const noexcept;
};

/* TODO:
 *  What means exit code:
 *    * exception
 *    * group of exceptions ?
 */

} //types
} //etu_game

#endif // ETU_GAME_EXCEPTION_CLASS_H
```

Название файла: src/etu_game/objects/cell.cc

```
#include "etu_game/objects/cell.h"

namespace etu_game{

namespace objects{

Cell::Cell()
    :type(types::CellType::kEmpty)
{
}

Cell::Cell(const Cell& cell){
    type = cell.type;
}
Cell& Cell::operator=(const Cell& cell){
    if (&cell == this) return *this;
    type = cell.type;
    return *this;
}


Cell::Cell(types::CellType cell_type)
    :type(cell_type)
```

```
{
}

types::CellType Cell::GetType() const {
    return type;
}


void Cell::SetType(types::CellType new_type){
    type = new_type;
}

Cell::~Cell(){}

} // namespace objects
} // namespace etu_game
```

Название файла: src/etu_game/objects/field.cc

```
#include "etu_game/objects/field.h"

namespace etu_game{

namespace objects {

Field::Field(int height, int width){
    if (height <= 0 || width <= 0)
          throw types::EtuGameException(1, "Wrong size values for
Field");
    this->height = height;
    this->width = width;
    cells = std::make_unique<std::unique_ptr<Cell[]>[]>(height);
    for(int i = 0; i<height; i++){
        cells[i] = std::make_unique<Cell[]>(width);
    }
}

Field::Field(const Field& field){
    height = field.width;
    width = field.height;
    cells = std::make_unique<std::unique_ptr<Cell[]>[]>(height);
    for(int i = 0; i<height; i++){
        cells[i] = std::make_unique<Cell[]>(width);
        for(int j = 0; j<width; j++){
            cells[i][j] = field.cells[i][j];
        }
    }
}

Field& Field::operator=(const Field& field){
    if (&field == this) return *this;
    height = field.height;
    width = field.width;
```

```
        cells = std::make_unique<std::unique_ptr<Cell[]>[]>(height);
        for(int i = 0; i<height; i++){
            cells[i] = std::make_unique<Cell[]>(width);
            for(int j = 0; j<width; j++){
                cells[i][j] = field.cells[i][j];
            }
        }
        return *this;
    }

    Field::Field(Field&& field){
        height = field.height;
        width = field.width;
        cells = std::move(field.cells);
                                                field.cells          =
std::unique_ptr<std::unique_ptr<Cell[]>[]>(nullptr);
    }

    Field& Field::operator=(Field&& field){
        if (&field == this) return *this;
        height = field.height;
        width = field.width;
        cells = std::move(field.cells);
                                                field.cells          =
std::unique_ptr<std::unique_ptr<Cell[]>[]>(nullptr);
        return *this;
    }

    bool Field::CheckInvariant(){
        for(int i = 0; i < height; i++){
            for(int j = 0; j < width; j++){
                if (cells[i][j].GetType() == types::CellType::kBlock)
continue;
                   if (i < height - 1 && cells[i+1][j].GetType() !=
types::CellType::kBlock)
                    continue;
                         if (i > 0 && cells[i-1][j].GetType() !=
types::CellType::kBlock)
                    continue;
                         if (j > 0 && cells[i][j-1].GetType() !=
types::CellType::kBlock)
                    continue;
                     if (j < width -1 && cells[i][j+1].GetType() !=
types::CellType::kBlock)
                    continue;
                return false;
            }
        }
        return true;
    }

    Field::~Field(){}

    Field& Field::GetInstance(int height, int width){
        static Field field(height, width);
        return field;
```

```
}

int Field::GetHeight() const {
    return height;
}

int Field::GetWidth() const {
    return width;
}

const Cell& Field::GetCell(int h_pos, int w_pos){
    return cells[h_pos][w_pos];
}

} // objects
} // etu_game
```

Название файла: src/etu_game/objects/field_iterator.cc

```
#include "etu_game/objects/field_iterator.h"

namespace etu_game{

namespace objects {

FieldIterator::FieldIterator(const Field& f)
    :field(f),
    i(0),
    j(0)
{
}

void FieldIterator::Next(){
     if(i == field.height) throw types::EtuGameException(1, "No
next element");
    j++;
    if(j == field.width){
        j = 0;
        i++;
    }
}

void FieldIterator::operator++(){
    Next();
}

void FieldIterator::operator++(int none){
    Next();
}

bool FieldIterator::IsDone(){
    return (i == field.height);
}
```

```
bool FieldIterator::operator()(){
    return (i == field.height);
}

const Cell& FieldIterator::CurrentItem(){
    return field.cells[i][j];
}

const Cell& FieldIterator::operator*(){
    return field.cells[i][j];
}



} // objects
} // etu_game
```

Название файла: src/etu_game/types/etu_game_exception.cc

```
#include "etu_game/types/etu_game_exception.h"

namespace etu_game{

namespace types{

EtuGameException::EtuGameException(int code, const char*
message):
exit_code(code),
msg(message)
{
}

EtuGameException::~EtuGameException(){}

int EtuGameException::GetExitCode(){
    return exit_code;
}

const char* EtuGameException::what() const noexcept{
    return msg;
}

} //types
} //etu_game
```

Название файла: unittests/unittest.h

```
#include <cassert>
```

```
class UnitTest{
public:
    static void Assert();
    static void AssertEqual(int a, int b);
    static void AssertNotEqual(int a, int b);
    static void AssertGreaterEqual(int a, int b);
    static void AssertLessEqual(int a, int b);
    static void AssertGreater(int a, int b);
    static void AssertLess(int a, int b);
};
```

Название файла: unittests/unittest.cc

```
#include "unittest.h"

void UnitTest::Assert(){
    assert(0);
}

void UnitTest::AssertEqual(int a, int b){
    assert(a == b);
}

void UnitTest::AssertNotEqual(int a, int b){
    assert(a != b);
}

void UnitTest::AssertGreaterEqual(int a, int b){
    assert(a >= b);
}

void UnitTest::AssertLessEqual(int a, int b){
    assert(a <= b);
}

void UnitTest::AssertGreater(int a, int b){
    assert(a > b);
}
void UnitTest::AssertLess(int a, int b){
    assert(a < b);
}
```

Название файла: unittests/unittest1.cc

```
#include <iostream>
#include "etu_game/objects/cell.h"
#include "etu_game/objects/field.h"
#include "etu_game/objects/field_iterator.h"
#include "etu_game/types/etu_game_exception.h"
#include "unittest.h"
```

```
using namespace etu_game;
using namespace objects;
using namespace types;
using namespace std;

// FIXME: may be do with abort()?

int main(){
     cout << "UnitTest 1: Testing Cell, EtuGameException, Field,
FieldIterator...\n";
    {
        cout << "   Testing Cell...\n";
        Cell cell;
                                UnitTest::AssertEqual(cell.GetType(),
types::CellType::kEmpty);
        cell.SetType(types::CellType::kBlock);
                                UnitTest::AssertEqual(cell.GetType(),
types::CellType::kBlock);
        Cell cell2 = cell;
                               UnitTest::AssertEqual(cell2.GetType(),
types::CellType::kBlock);
        Cell cell3(types::CellType::kEntry);
                               UnitTest::AssertEqual(cell3.GetType(),
types::CellType::kEntry);
        cout << "   Cell testing done.\n";
    }
    {
          cout << "     Testing EtuGameException and Field
constructor...\n";
        try{
            Field& f = Field::GetInstance(-12,13);
            UnitTest::Assert();
        }catch(EtuGameException& e){}
        try{
            Field& f = Field::GetInstance(12,-67);
            UnitTest::Assert();
        }catch(EtuGameException& e){}
        try{
            Field& f = Field::GetInstance(0,13);
            UnitTest::Assert();
        }catch(EtuGameException& e){}
        try{
            Field& f = Field::GetInstance(48,0);
            UnitTest::Assert();
        }catch(EtuGameException& e){}
        cout << "   EtuGameException testing done.\n";
    }
    {
        cout << "   Testing Field...\n";
        int width = 14, height = 5;
        Field& f =Field::GetInstance(height, width);
        UnitTest::AssertEqual(height, f.GetHeight());
        UnitTest::AssertEqual(width, f.GetWidth());
        Field& f2 =Field::GetInstance(8,8);
        UnitTest::AssertEqual(height, f.GetHeight());
```

```
            UnitTest::AssertEqual(width, f.GetWidth());
             // FIXME: do unittest for CheckInvariant() when it will
be possible to load the map
            if( !f2.CheckInvariant() ) UnitTest::Assert();
            cout << "   Field testing done.\n";
        }
        {
            Field& f2 =Field::GetInstance(8,8);
            cout << "   Testing FieldIterator...\n";
            /* FIXME:
             *  1) prefix increment to postfix increment (completed)
              *  2) do good unittest for when it will be possible to
load the map
             */
            for(FieldIterator iter(f2); iter(); iter++){
                  UnitTest::AssertEqual(iter.CurrentItem().GetType(),
types::CellType::kEmpty);
            }
            try{
                for(FieldIterator iter(f2);;iter++){}
                UnitTest::Assert();
            }catch(EtuGameException& e){}
            cout << "   FieldIterator testing done.\n";
        }
        cout << "UnitTest 1 done.\n";
        return 0;
    }
```