**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:

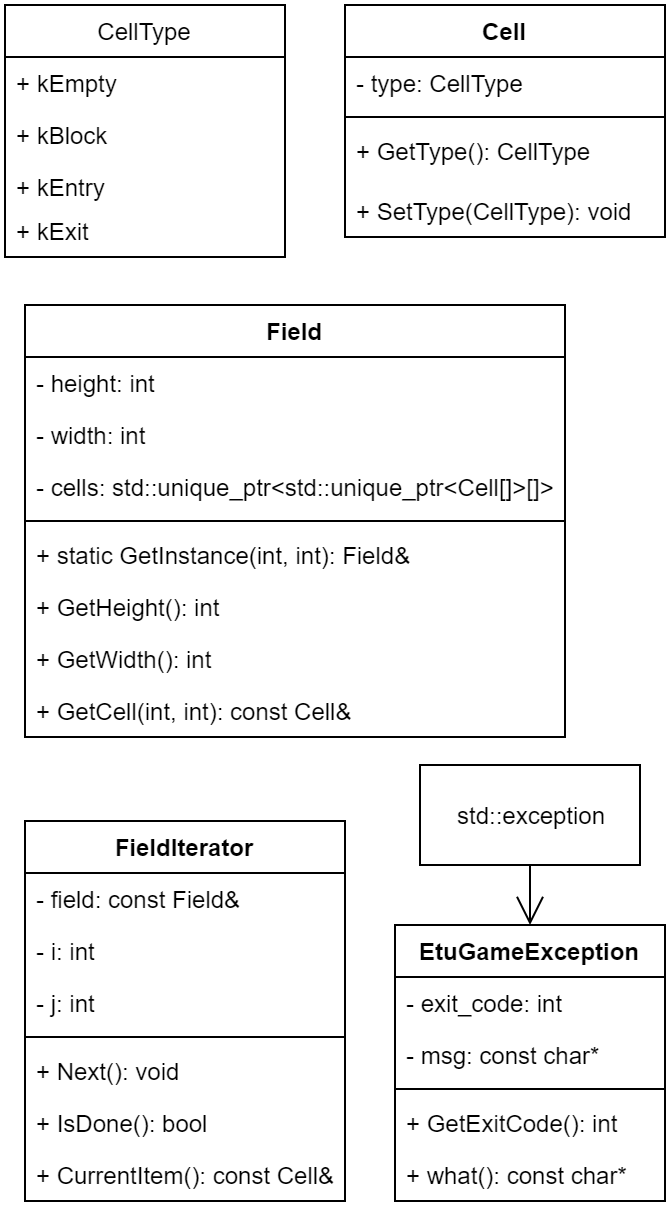
* Реализован класс поля
* Реализован класс клетки
* Для класса поля написаны конструкторы копирования и перемещения, а также операторы присваивания и перемещения
* Поле сохраняет инвариант - из любой клетки можно провести путь до любой другой
* Гарантированно отсутствует утечки памяти

Дополнительные требования:

* Поле создается с использованием паттерна Синглтон
* Для обхода по полю используется паттерн Итератор

## Выполнение работы.

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

Рисунок 1 — UML-диграмма

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;

}