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. 9381 _____ Колованов Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучить парадигму объектно-ориентрированного программирования; реализовать класс игрового поля и класс клетки игрового поля; изучить и реализовать паттерны проектирования *Singleton* и *Iterator*.

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 начала были реализованы классы *Cell* и *Field*. Для них были реализованы конструкторы копирования и перемещения, операторы присваивания копированием и перемещением. В программе используются умные указатели, поэтому очистка памяти в некоторых классах не требуется. Для класса *Field* используются паттерны проектирования *Singleton* (контролирует создание объекта класса в единственном экземпляре) и *Iterator* (предоставляет класс для доступа к элементам коллекции). Для реализации GUI-интерфейса программы был использован фреймворк *Qt*.

Подробное описание классов и перечислений приведено ниже (см. разделы *Описание перечислений* и *Описание классов и структур*).

Разработанный программный код см. в приложении А.

Описание перечислений.

Перечисление CellType.

Хранит тип клетки игрового поля. В зависимости от типа клетка может быть проходима или непроходима. Существуют следующие типы:

- *TYPE_VOID* — недоступная клетка (непроходима). Будет использоваться для заполнения недоступных областей.
- *TYPE_EMPTY* — пустая клетка (проходима).
- *TYPE_WALL* — стена (непроходима).
- *TYPE_ENTRY* — клетка-вход (проходима).
- *TYPE_EXIT* — клетка-выход (проходима).

Перечисление CellTexture.

Хранит идентификатор текстуры клетки игрового поля. В зависимости от значения позволяет выбрать из словаря текстур нужное изображение.

Описание классов и структур.

Структура Coords2D.

Используется для хранения двумерных координат/размеров.

Поля структуры *Coords2D*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
public	*size_t x*	Хранит координату/размер по X.	-
public	*size_t y*	Хранит координату/размер по Y.	-

Методы структуры *Coords2D*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*Coords2D(size_t x, size_t y)*
public	*bool*	*operator==(const Coords2D& other)*
public	*bool*	*operator!=(const Coords2D& other)*

Класс Exception.

Объекты класса используются для выбрасывания информации об ошибки в качестве исключения.

Поля класса *Exception*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*const std::string error_*	Хранит сообщение об ошибке.	-

Методы класса *Exception*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*Exception(const std::string error)*
public	*const std::string&*	*getError()*

Класс GameObject.

Абстрактный класс, используется в качестве суперкласса для других классов различных игровых объектов, которые бы имели общие поля и интерфейс.

Методы класса *GameObject*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*virtual void*	*interact() = 0*

Класс Cell.

Данный класс представляет собой ячейку игрового поля (является элементом коллекции ячеек поля *Field*), которая имеет определенное место (характеризуется координатами) на нем. Содержит информацию о нахождении на ней игрового объекта, персонажа, о свойствах самой клетки (проходимость, тип) и ее внешнего вида (текстура).

Поля класса *Cell*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*bool passable_*	Хранит информацию о том, проходима ли клетка.	*false*

private	*CellType type_*	Хранит информацию о типе клетки.	*TYPE_VOID*
private	*CellTexture texture_*	Хранит информацию о текстуре клетки.	*TEXTURE_VOID*
private	*pGameObject object_*	Хранит адрес игрового объъекта, стоящего на клетке.	*nullptr*
private	*Coords2D coords_*	Хранит координаты клетки в игровом поле.	*Coords2D(0, 0)*

Методы класса *Cell*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*Cell() = default*
public	-	*Cell(Coords2D coords, CellTexture texture = TEXTURE_VOID, CellType type = TYPE_VOID, pGameObject object = nullptr)*
public	-	*Cell(const Cell& other)*
public	-	*Cell(Cell&& other)*
public	-	*~Cell() = default*
public	*Cell&*	*operator=(const Cell& other)*
public	*Cell&*	*operator=(Cell&& other)*
public	*bool*	*isPassable() const*
public	*bool*	*getPassible() const*
public	*pGameObject*	*getObject()*
public	*CellTexture*	*getTexture() const*
public	*CellType*	*getType() const*
public	*Coords2D*	*getCoords() const*
public	*void*	*setObject(pGameObject object)*

public	*void*	*setTexture(CellTexture texture)*
public	*void*	*changeType(CellType type)*

Класс Field.

Данный класс представляет собой игровое поле — прямоугольную матрицу ячеек игрового поля (объектов класса *Cell*), каждый элемент которой имеет уникальные координаты. Класс реализован с использованием паттернов *Singleton* и *Iterator*. Для реализации паттерна *Iterator* в класс было добавлено статическое поле — умный указатель на единственный объект класса *Field*, а также конструктам и операторам присваивания был добавлен модификатор доступа *private*. Для создания/удаления/получения объекта класса Field используются статические методы *initInstance*, *getInstance* и *deleteInstance*. Для получения объекта итератора коллекции используются методы *begin* и *end* (возвращают объект класса *FieldIterator*).

Поля класса *Field*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*Size2D size_*	Хранит размер двумерного массива ячеек игрового поля.	*Size2D(0, 0)*
private	*ppCell cells_*	Указатель на двумерный массив ячеек игрового поля.	*nullptr*
private	*static pField instance_*	Указатель на единственный объект класса *Field*.	*nullptr*

Методы класса *Field*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы

private	-	*Field(const Size2D& size)*
private	-	*Field(const Field& other)*
private	-	*Field(Field&& other)*
private	*Field&*	*operator=(const Field& other)*
private	*Field&*	*operator=(Field&& other)*
public	*static Field&*	*initInstance(const Size2D& size)*
public	*static Field&*	*getInstance()*
public	*static void*	*deleteInstance()*
public	*Cell&*	*getCell(const Coords2D& coords)*
public	*pGameObject*	*getCellObject(const Coords2D& coords)*
public	*Size2D*	*getSize() const*
public	*size_t*	*getSizeX() const*
public	*size_t*	*getSizeY() const*
public	*FieldIterator*	*begin()*
public	*FieldIterator*	*end()*

Класс FieldIterator.

Класс итератора для коллекции *Field*. Используется для обхода каждого элемента (объекта класса *Cell*) игрового поля (объекта класса *Field)*.

Поля класса *FieldIterator*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*Coords2D position_*	Хранит текущее положение итератора.	-

Методы класса *FieldIterator*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*FieldIterator(const Coords2D& coords)*
public	*bool*	*operator==(const FieldIterator& other)*
public	*bool*	*operator!=(const FieldIterator& other)*
public	*FieldIterator&*	*operator++()*
public	*FieldIterator*	*operator++(int)*
	Cell&	*operator*()*

Класс MainWindow.

Класс основного окна приложения (игры). Используется для отображения игрового поля на экран. При помощи вызова метода *display* генерируется карта для игрового поля (устанавливает типы, текстуры, объекты клеткам поля), а после сгенерированная карта отрисовывается в окне *QGraphicsView*.

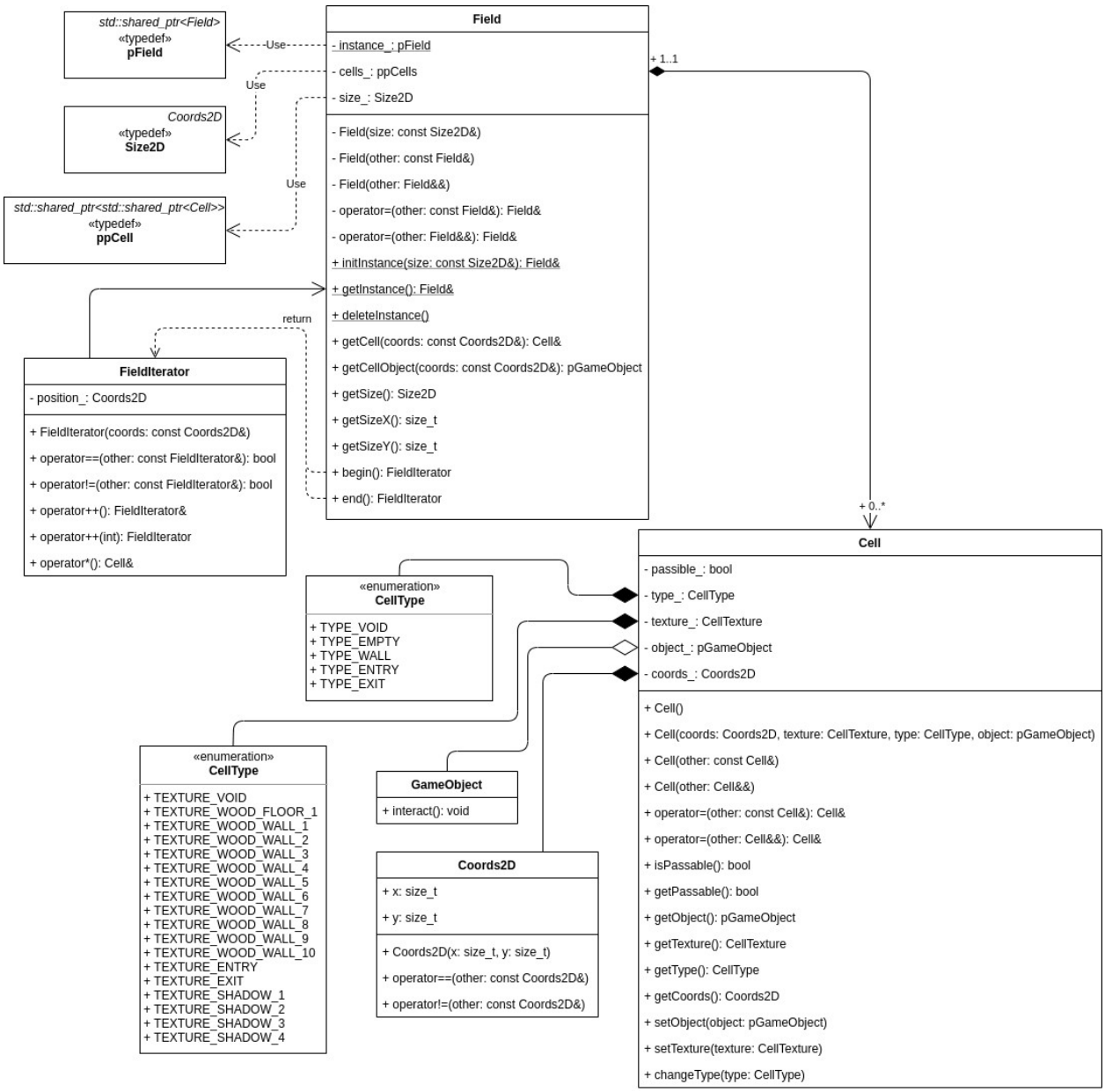
Поля класса *MainWindow*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
public	*ui::MainWindow* ui*	Хранит координату/размер по X.	-

Методы класса *MainWindow*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*MainWindow(QWidget* parent = nullptr)*
public	-	*~MainWindow()*
public	*void*	*display()*

UML-диаграмма.



Тестирование.

Результаты тестирования представлены на рис. 1.

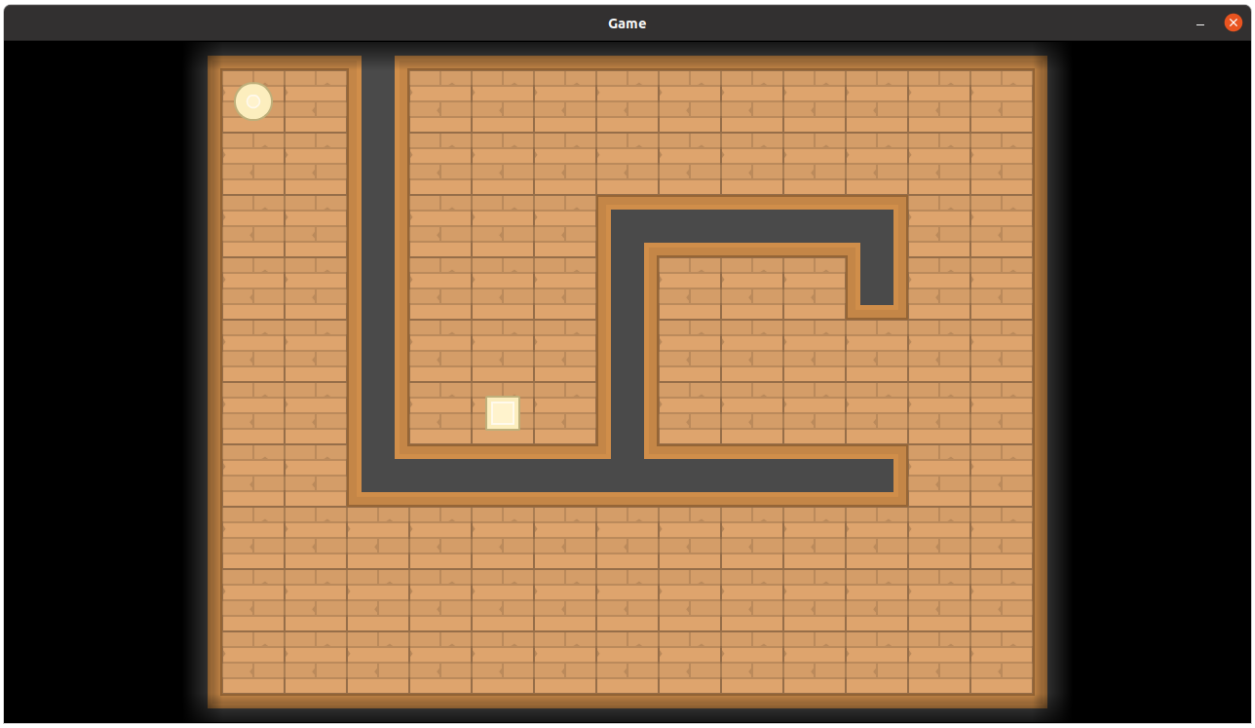


Рисунок 1 — Игровое поле

Выводы.

Была изучена парадигма объектно-ориентрированного программирования. Были реализованны классы игрового поля и клетки игрового поля. Для класса поля написаны конструкторы копирования и перемещения, а также операторы присваивания и перемещения. Были изучены и реализованы паттерны проектирования *Singleton* и *Iterator*. По мимо этого, был реализован GUI-интерфейс игры при помощи фреймворка Qt.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <QApplication>
#include <iostream>
#include "mainwindow.h"
#include "field.h"
#include "exception.h"
#include <QMessageBox>

int main(int argc, char* argv[]) {
    QApplication app(argc, argv);
    MainWindow w;

    try {
        Field::initInstance(Size2D(15, 12));
        w.display();
    } catch (Exception e) {
        std::cerr << e.getError() << "\n";
        return -1;
    }

    w.show();
    return app.exec();
}
```

Название файла: mainwindow.h

```
#ifndef MAIN_WINDOW_H
#define MAIN_WINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui {
    class MainWindow;
}
QT_END_NAMESPACE


class MainWindow: public QMainWindow {
    Q_OBJECT

    Ui::MainWindow* ui;

public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();
    void display();
};


#endif // MAIN_WINDOW_H
```

Название файла: mainwindow.cpp

```
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QMap>
#include <iostream>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "field.h"

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new
Ui::MainWindow) {
    ui->setupUi(this);
}

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::display() {
    Field& field = Field::getInstance();
    QGraphicsView* view = new QGraphicsView(this);
    QGraphicsScene* scene = new QGraphicsScene(this);
    QPixmap map = QPixmap(field.getSizeX() * 64, field.getSizeY() * 64);
    QPainter painter(&map);
    QMap<CellTexture, QImage> textures;

    view-
>setHorizontalScrollBarPolicy(Qt::ScrollBarPolicy::ScrollBarAlwaysOff);
    view-
>setVerticalScrollBarPolicy(Qt::ScrollBarPolicy::ScrollBarAlwaysOff);
    view->setStyleSheet("background-color: black;");

    int textureMap[12][15] = {{6, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
7},
                              {3, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 3, 1, 1, 1, 6, 2, 2, 2, 7, 1, 1,
3},
                              {3, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 10, 1,
1, 3},
                              {3, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 4, 2, 2, 2, 11, 2, 2, 2, 9, 1,
1, 3},
                              {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
3},
                              {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
3},
                              {4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
5}};

    int cellTypeMap[12][15] = {{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2},
```

```
                                    {2, 3, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1,
1, 2},
                                    {2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1,
1, 2},
                                    {2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 2, 1, 4, 1, 2, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
1, 2},
                                    {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2},
                                    {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2}};

          textures[TEXTURE_VOID] = QImage(":/tiles/tile_00.png");
          textures[TEXTURE_WOOD_FLOOR_1] = QImage(":/tiles/tile_100.png");
          textures[TEXTURE_WOOD_WALL_1] = QImage(":/tiles/tile_120.png");
          textures[TEXTURE_WOOD_WALL_2] = QImage(":/tiles/tile_147.png");
          textures[TEXTURE_WOOD_WALL_3] = QImage(":/tiles/tile_145.png");
          textures[TEXTURE_WOOD_WALL_4] = QImage(":/tiles/tile_146.png");
          textures[TEXTURE_WOOD_WALL_5] = QImage(":/tiles/tile_118.png");
          textures[TEXTURE_WOOD_WALL_6] = QImage(":/tiles/tile_119.png");
          textures[TEXTURE_WOOD_WALL_7] = QImage(":/tiles/tile_121.png");
          textures[TEXTURE_WOOD_WALL_8] = QImage(":/tiles/tile_123.png");
          textures[TEXTURE_WOOD_WALL_9] = QImage(":/tiles/tile_124.png");
          textures[TEXTURE_WOOD_WALL_10] = QImage(":/tiles/tile_122.png");
          textures[TEXTURE_ENTRY] = QImage(":/tiles/tile_132.png");
          textures[TEXTURE_EXIT] = QImage(":/tiles/tile_133.png");
          textures[TEXTURE_SHADOW_1] = QImage(":/tiles/shadow_01.png");
          textures[TEXTURE_SHADOW_2] = QImage(":/tiles/shadow_02.png");
          textures[TEXTURE_SHADOW_3] = QImage(":/tiles/shadow_03.png");
          textures[TEXTURE_SHADOW_4] = QImage(":/tiles/shadow_04.png");

          QImage cellTexture(":/tiles/cell.png");

          for (Cell& cell : field) {
              Coords2D coords = cell.getCoords();

              cell.changeType(static_cast<CellType>(cellTypeMap[coords.y]
[coords.x]));
              cell.setTexture(static_cast<CellTexture>(textureMap[coords.y]
[coords.x]));

              painter.drawImage(coords.x * 64, coords.y * 64,
textures[cell.getTexture()]);

              if (cell.getType() == TYPE_ENTRY) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[TEXTURE_ENTRY]);
              } else if (cell.getType() == TYPE_EXIT) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[TEXTURE_EXIT]);
```

```
            }

            if (cell.getPassible()) {
                painter.drawImage(coords.x * 64, coords.y * 64,
cellTexture);
            }
        }

        for (size_t i = 0; i < field.getSizeY(); i++) {
            for (size_t j = 0; j < field.getSizeX(); j++) {
                if (i == 0) {
                    painter.drawImage(j * 64, i * 64,
textures[TEXTURE_SHADOW_2]);
                }
                if (j == 0) {
                    painter.drawImage(j * 64, i * 64,
textures[TEXTURE_SHADOW_1]);
                }
                if (i == field.getSizeY() - 1) {
                    painter.drawImage(j * 64, i * 64,
textures[TEXTURE_SHADOW_4]);
                }
                if (j == field.getSizeX() - 1) {
                    painter.drawImage(j * 64, i * 64,
textures[TEXTURE_SHADOW_3]);
                }
            }
        }

        scene->addPixmap(map);
        view->setScene(scene);
        setCentralWidget(view);
    }
```

Название файла: exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <string>


class Exception {
    const std::string error_;

public:
    Exception(const std::string& error);
    const std::string& getError() const;
};


#endif // EXCEPTION_H
```

Название файла: exception.cpp

```
#include "exception.h"

Exception::Exception(const std::string& error): error_(error) {}
```

```
const std::string &Exception::getError() const {
    return error_;
}
```

Название файла: coords2d.h

```
#ifndef COORDS_2D_H
#define COORDS_2D_H

#include <cstddef>

typedef struct Coords2D Size2D;


struct Coords2D {
    size_t x;
    size_t y;

    Coords2D(size_t x, size_t y);
    bool operator==(const Coords2D& other);
    bool operator!=(const Coords2D& other);
};


#endif // COORDS_2D_H
```

Название файла: coords2d.cpp

```
#include "coords2d.h"

Coords2D::Coords2D(size_t x, size_t y): x(x), y(y) {}

bool Coords2D::operator==(const Coords2D& other) {
    return x == other.x && y ==other.y;
}

bool Coords2D::operator!=(const Coords2D& other) {
    return !operator==(other);
}
```

Название файла: celltype.h

```
#ifndef CELL_TYPE_H
#define CELL_TYPE_H

enum CellType {
    TYPE_VOID,
    TYPE_EMPTY,
    TYPE_WALL,
    TYPE_ENTRY,
    TYPE_EXIT
};

#endif // CELL_TYPE_H
```

Название файла: celltexture.h

```
#ifndef CELL_TEXTURE_H
#define CELL_TEXTURE_H

enum CellTexture {
    TEXTURE_VOID,
    TEXTURE_WOOD_FLOOR_1,
    TEXTURE_WOOD_WALL_1,
    TEXTURE_WOOD_WALL_2,
    TEXTURE_WOOD_WALL_3,
    TEXTURE_WOOD_WALL_4,
    TEXTURE_WOOD_WALL_5,
    TEXTURE_WOOD_WALL_6,
    TEXTURE_WOOD_WALL_7,
    TEXTURE_WOOD_WALL_8,
    TEXTURE_WOOD_WALL_9,
    TEXTURE_WOOD_WALL_10,
    TEXTURE_ENTRY,
    TEXTURE_EXIT,
    TEXTURE_SHADOW_1,
    TEXTURE_SHADOW_2,
    TEXTURE_SHADOW_3,
    TEXTURE_SHADOW_4
};

#endif // CELL_TEXTURE_H
```

Название файла: cell.h

```
#ifndef CELL_H
#define CELL_H

#include <memory>
#include "coords2d.h"
#include "celltype.h"
#include "celltexture.h"
#include "gameobject.h"

typedef std::shared_ptr<class Cell> pCell;
typedef std::shared_ptr<std::shared_ptr<class Cell>> ppCell;


class Cell {
    bool passable_ = false;
    CellType type_ = TYPE_VOID;
    CellTexture texture_ = TEXTURE_VOID;
    pGameObject object_ = nullptr;
    Coords2D coords_ = Coords2D(0, 0);

public:
    Cell() = default;
    explicit Cell(Coords2D coords, CellTexture texture = TEXTURE_VOID,
CellType type = TYPE_VOID, pGameObject object = nullptr);
    Cell(const Cell& other);
    Cell(Cell&& other);
    ~Cell() = default;

    Cell& operator=(const Cell& other);
    Cell& operator=(Cell&& other);
```

```
        bool isPassable() const;
        bool getPassible() const;
        pGameObject getObject();
        CellTexture getTexture() const;
        CellType getType() const;
        Coords2D getCoords() const;
        void setObject(pGameObject object);
        void setTexture(CellTexture texture);
        void changeType(CellType type);
    };


    #endif // CELL_H
```

Название файла: cell.cpp

```
    #include "cell.h"

    Cell::Cell(Coords2D coords, CellTexture texture, CellType type,
pGameObject object) {
        coords_ = coords;
        texture_ = texture;
        object_ = object;
        changeType(type);
    }

    Cell::Cell(const Cell& other) {
        operator=(other);
    }

    Cell::Cell(Cell&& other) {
        coords_ = other.coords_;
        texture_ = other.texture_;
        type_ = other.type_;
        passable_ = other.passable_;
        object_ = other.object_;
        other.object_ = nullptr;
    }

    Cell& Cell::operator=(const Cell& other) {
        if (this != &other) {
            coords_ = other.coords_;
            texture_ = other.texture_;
            type_ = other.type_;
            passable_ = other.passable_;
        }

        return *this;
    }

    Cell& Cell::operator=(Cell&& other) {
        if (this != &other) {
            std::swap(coords_, other.coords_);
            std::swap(texture_, other.texture_);
            std::swap(type_, other.type_);
            std::swap(passable_, other.passable_);
            std::swap(object_, other.object_);
        }
```

```
    return *this;
}

bool Cell::isPassable() const {
    return passable_ && object_ == nullptr;
}

bool Cell::getPassible() const {
    return passable_;
}

pGameObject Cell::getObject() {
    return object_;
}

CellTexture Cell::getTexture() const {
    return texture_;
}

CellType Cell::getType() const {
    return type_;
}

Coords2D Cell::getCoords() const {
    return coords_;
}

void Cell::setObject(pGameObject object) {
    object_ = object;
}

void Cell::setTexture(CellTexture texture) {
    texture_ = texture;
}

void Cell::changeType(CellType type) {
    type_ = type;

    switch (type) {
    case TYPE_EMPTY:
    case TYPE_ENTRY:
    case TYPE_EXIT:
        passable_ = true;
        break;
    case TYPE_VOID:
    case TYPE_WALL:
    default:
        passable_ = false;
        break;
    }
}
```

Название файла: gameobject.h

```
#ifndef GAME_OBJECT_H
#define GAME_OBJECT_H

#include <memory>

typedef std::shared_ptr<class GameObject> pGameObject;
```

```
class GameObject {
public:
    virtual void interact() = 0;
};


#endif // GAME_OBJECT_H
```

Название файла: field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <memory>
#include <cstddef>
#include "cell.h"
#include "coords2d.h"

typedef std::shared_ptr<class Field> pField;
typedef std::shared_ptr<class Cell> pCell;
typedef std::shared_ptr<std::shared_ptr<class Cell>> ppCell;
typedef std::shared_ptr<class GameObject> pGameObject;


class Field {
    static pField instance_;

    Size2D size_ = Size2D(0, 0);
    ppCell cells_ = nullptr;

    Field(const Size2D& size);
    Field(const Field& other);
    Field(Field&& other);

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

    class FieldIterator;

public:
    static Field& initInstance(const Size2D& size);
    static Field& getInstance();
    static void deleteInstance();
    Cell& getCell(const Coords2D& coords);
    pGameObject getCellObject(const Coords2D& coords);
    Size2D getSize() const;
    size_t getSizeX() const;
    size_t getSizeY() const;
    FieldIterator begin();
    FieldIterator end();
};


class Field::FieldIterator {
    Coords2D position_;

public:
    explicit FieldIterator(const Coords2D& coords);
```

```
        bool operator==(const FieldIterator& other);
        bool operator!=(const FieldIterator& other);
        FieldIterator& operator++();
        FieldIterator operator++(int);
        Cell& operator*();
    };


    #endif // FIELD_H
```

Название файла: field.cpp

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

    pField Field::instance_ = nullptr;

    Field::Field(const Size2D& size): size_(size) {
        cells_ = ppCell(new pCell[size.y], std::default_delete<pCell[]>());

        for (size_t y = 0; y < size.y; y++) {
            cells_.get()[y] = pCell(new Cell[size.x],
std::default_delete<Cell[]>());
        }

        for (size_t y = 0; y < size.y; y++) {
            for (size_t x = 0; x < size.x; x++) {
                cells_.get()[y].get()[x] = Cell(Coords2D(x, y));
            }
        }
    }

    Field::Field(const Field& other) {
        operator=(other);
    }

    Field::Field(Field&& other) {
        size_ = other.size_;
        cells_ = other.cells_;
        other.cells_ = nullptr;
    }

    Field& Field::operator=(const Field& other) {
        if (this != &other) {
            size_ = other.size_;

            if (other.cells_ != nullptr) {
                cells_ = ppCell(new pCell[size_.y],
std::default_delete<pCell[]>());

                for (size_t y = 0; y < size_.y; y++) {
                    cells_.get()[y] = pCell(new Cell[size_.x],
std::default_delete<Cell[]>());
                    for (size_t x = 0; x < size_.x; x++) {
                        cells_.get()[y].get()[x] = other.cells_.get()
[y].get()[x];
                    }
                }
```

```
        }
    }

    return *this;
}

Field& Field::operator=(Field&& other) {
    if (this != &other) {
        std::swap(size_, other.size_);
        std::swap(cells_, other.cells_);
    }

    return *this;
}

Field& Field::initInstance(const Size2D& size) {
    if (instance_ == nullptr) {
        instance_ = pField(new Field(size));
    }
    return *instance_;
}

Field& Field::getInstance() {
    if (instance_ == nullptr) {
        instance_ = pField(new Field(Size2D(10, 10)));
    }
    return *instance_;
}

void Field::deleteInstance() {
    Field::instance_ = nullptr;
}

Cell& Field::getCell(const Coords2D& coords) {
    if (coords.x >= size_.x || coords.y >= size_.y) {
        throw Exception("Out of range");
    }
    return cells_.get()[coords.y].get()[coords.x];
}

pGameObject Field::getCellObject(const Coords2D& coords) {
    return getCell(coords).getObject();
}

Size2D Field::getSize() const {
    return size_;
}

size_t Field::getSizeX() const {
    return size_.x;
}

size_t Field::getSizeY() const {
    return size_.y;
}

Field::FieldIterator Field::begin() {
    return FieldIterator(Coords2D(0, 0));
}

Field::FieldIterator Field::end() {
```

```
        return FieldIterator(Coords2D(0, instance_->size_.y));
    }

    Field::FieldIterator::FieldIterator(const Coords2D& coords):
position_(coords) {}

    bool Field::FieldIterator::operator==(const FieldIterator& other) {
        return position_ == other.position_;
    }

    bool Field::FieldIterator::operator!=(const FieldIterator& other) {
        return !operator==(other);
    }

    Field::FieldIterator& Field::FieldIterator::operator++() {
        if (position_.x + 1 >= Field::getInstance().size_.x) {
            position_.y++;
            position_.x = 0;
        } else {
            position_.x++;
        }

        return *this;
    }

    Field::FieldIterator Field::FieldIterator::operator++(int) {
        FieldIterator iterator(*this);
        operator++();
        return iterator;
    }

    Cell& Field::FieldIterator::operator*() {
        return Field::getInstance().getCell(position_);
    }
```