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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Объектно-ориентированное программирование»

Тема: Паттерны проектирования

Студент гр. 7304 _____ Шарапенков И.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Изучить паттерны проектирования. Написать программу, использующую как минимум 3 паттерна каждого типа.

Задача.

Обязательные пункты для всех вариантов:

1. Использование как минимум один паттерна каждого типа для решения задачи (*порождающие, структурные, поведенческие*). Чем больше правильно реализованных паттернов, тем выше оценивается л/р.
2. Сохранение логов хода работы программы в файл. Необходимо реализовать 3 вида сохранения (*рекомендуется использовать паттерны Proxy/Заместитель и Singleton/Одиночка*)
   - Без сохранения логов
   - Моментальное сохранение (*информацию о событии сразу записывается в файл*)
   - Сохранение с кэшированием (*информация сохраняется в кэш, при заполнении кэша происходит запись в файл и очистка кэша*)
3. UML-диаграммы классов. Не обязательно все используемые классы помещать на одну диаграмму. Рекомендуется создавать отдельные диаграммы под отдельные логические компоненты программы.
4. Описание в отчете всех используемых паттернов. Необходимо указать, для чего и как используется тот или иной паттерн.
5. Реализация своего класса исключений их обработка.

Опциональные пункты (*для тех, кто претендует на автомат, необходимо выполнить как минимум один пункт*):

1. Сохранение и загрузка состояния программы в файл. Реализовать как минимум для 2 расширений файлов (*txt, dat, xml, json, бинарный файл*). Можно предложить свои расширения, но с разной структурой файлов.
2. Реализация GUI. При создании GUI необходимо разделять логику интерфейса и бизнес-логику.
3. Выполнение доп. задания

Дополнительные критерии оценки:

1. Разбиение проекта на файлы
2. Использование std::shared_ptr и std::unique_ptr
3. Отсутствие дублирования кода

При защите л/р будет задание по расширению программы (*добавление нового функционала или новых элементов*). Будет оцениваться то, насколько гибкий код и как быстро он был расширен, а также сколько изменений пришлось вносить в уже существующий код.

Вариант 1. Симуляция игрового поля.

Реализовать программу симуляции битвы двух армий на поле боя. Поле боя представляется квадратной сеткой, на которой располагаются юниты и препятствия. Создание поля можно сделать считыванием из файла или генерировать случайно.

Препятствия, могут быть статическими и динамическими. Статические препятствия находятся на поле все время на одном месте (*например, гора, лес или река*). Динамические препятствия могут появляться в какой-то момент времени или/и могут изменять место на поле (*например, песчаная буря или снегопад*). Препятствия должны по-разному влиять на юнитов, в положительную или отрицательную сторону. Препятствия могут запрещать передвижение через них, запрещать дальнюю атаку через них или добавлять дополнительную защиту юниту.

Юнит – единица, управляемая игроком. Каждый юнит может характеризоваться рядом параметров таких как: сила атаки, тип атаки, кол-во здоровья, тип передвижения, дальность передвижения, событие при уничтожении и.т.д.. Например, могут следующие юниты: башня - не может двигаться и имеет сильную дальнюю атаку; лучник – может далеко двигаться и имеет дальнюю атаку средней силы; тяжелая пехота – не может далеко двигаться и имеет сильную атаку в ближнем бою.

Формирование армии для каждого игрока должно происходить после запуска программы.

Должны быть сформированы правила битвы: условия победы, как происходит чередование хода, что можно сделать за один ход.

Минимальные требования к заданию:

1. 6 типов юнитов
2. 4 типа статических препятствий
3. 3 типа динамических препятствий
4. Задание размера поля

Дополнительное задание (выбрать минимум одно):

1. Реализация простого ИИ
2. Добавить базы, на которых можно создавать новых юнитов и любое кол-во игроков
3. Добавить возможность создания новых типов юнитов в ходе работы программы

Описание результатов.

1. Был реализован основной класс Game, являющийся Singleton. Цель класса хранить всю информацию о текущем состоянии игры, управлять ее состоянием. Все взаимодействие с игрой осуществляется через этот класс. UML диаграмма класса Game представлена на рисунке 1.

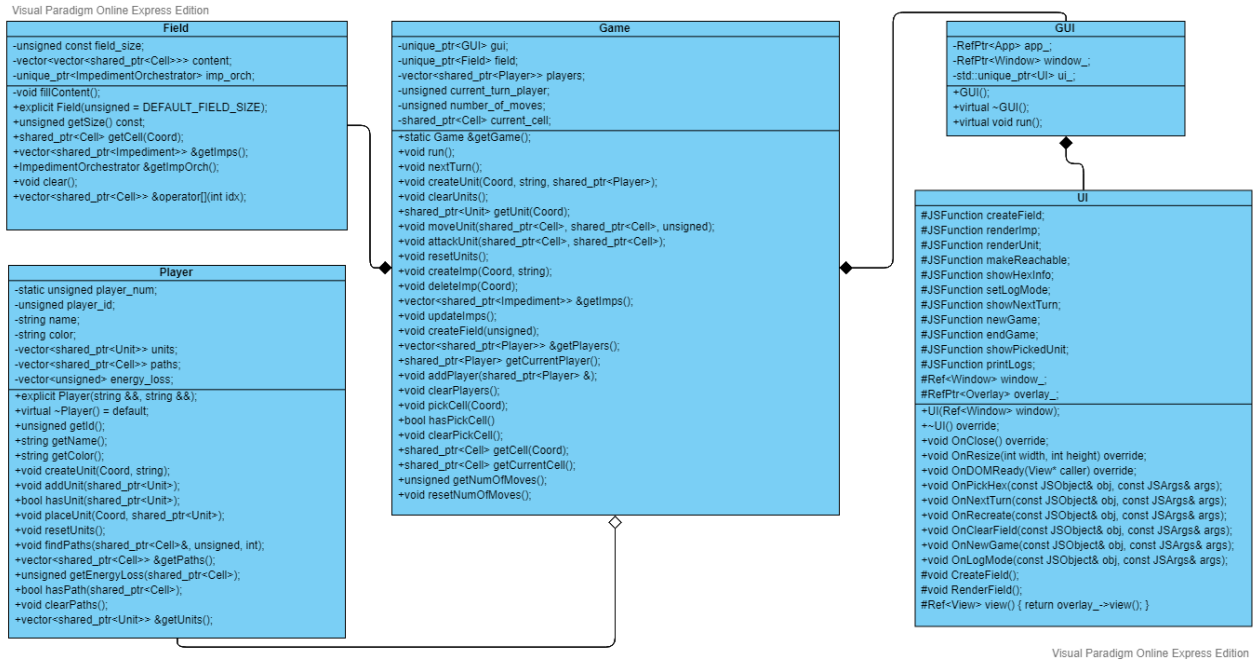

Рисунок 1. UML диаграмма класса Game.

2. Класс Field представляет игровое поле и содержит все основные операции, выполняемые над этим полем. Поле представляет собой массив объектов типа Cell. Так как препятствия – сущность, относящаяся к полю, то и обязанность по управлению препятствиями лежит на этом классе. Поэтому класс Field содержит объект класса ImpedimentOrchestrator, который представляет собой посредника для управления препятствиями на игровом поле. Препятствия представлены базовым классом Impediment. Так как препятствия делятся на динамические и статические, от базового класса были наследованы классы StaticImpediment и DynamicImpediment. В свою очередь от этих классов были наследованы конкретные препятствия. Вся логика, относящаяся к генерации

препятствий находится в классе ImpedimentOrchestrator. UML диаграмма классов, относящиеся к препятствиям представлена на рисунке 2.

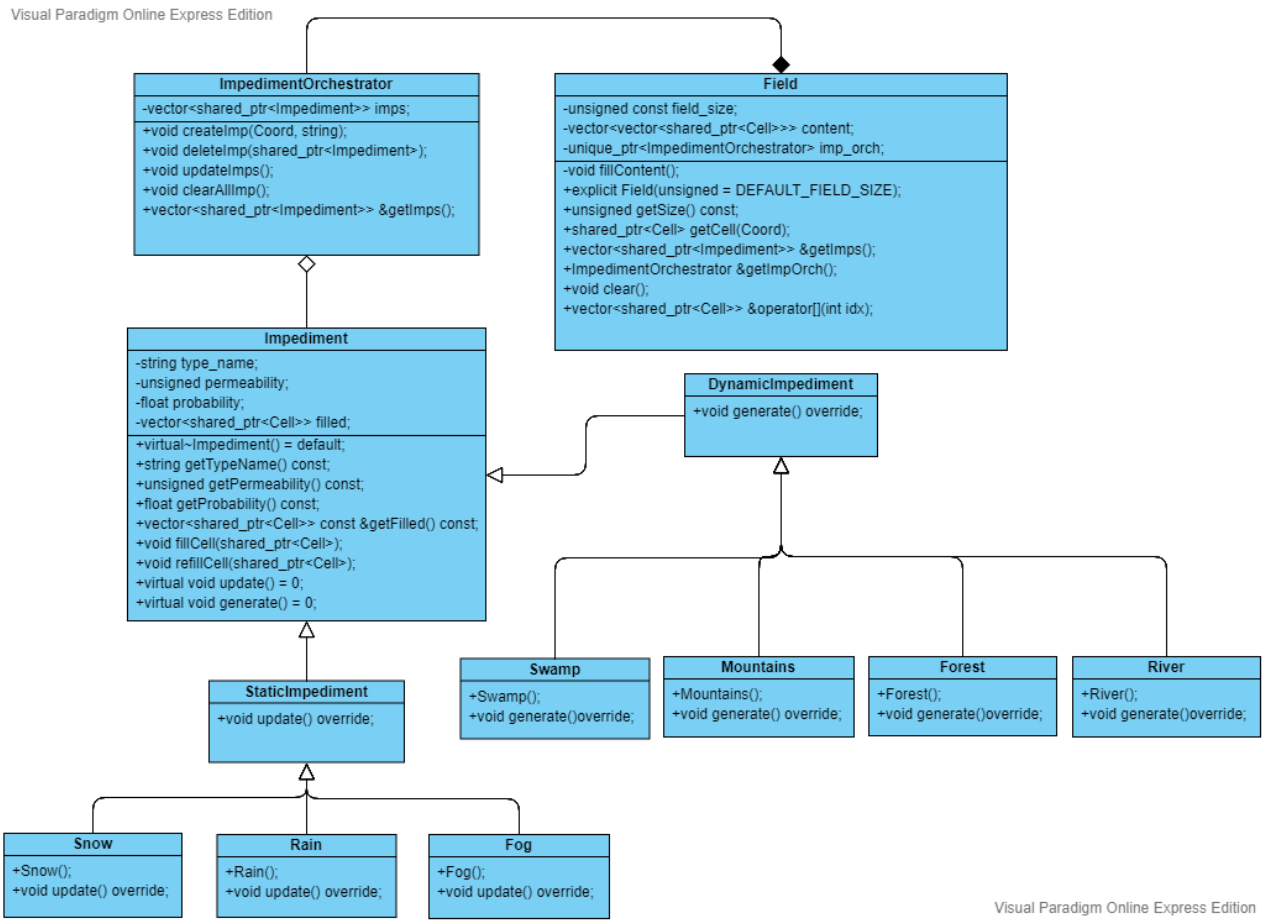

Рисунок 2. UML диаграмма класса ImpedimentOrchestrator

3. Класс Player представляет собой абстракцию игрока. Он содержит в себе методы управлению юнитами на игровом поле, а также данные для идентификации игрока и его юнитов: цвет, никнейм. Игрок выступает посредником для взаимодействия с юнитами. Игрок может создавать, удалять и перемещать юнитов. Юниты представлены базовым абстрактным классом Unit. Также были написаны функции для нахождения пути для текущего юнита. UML диаграмма классов, относящиеся к Player и Unit представлена на рисунке 3.

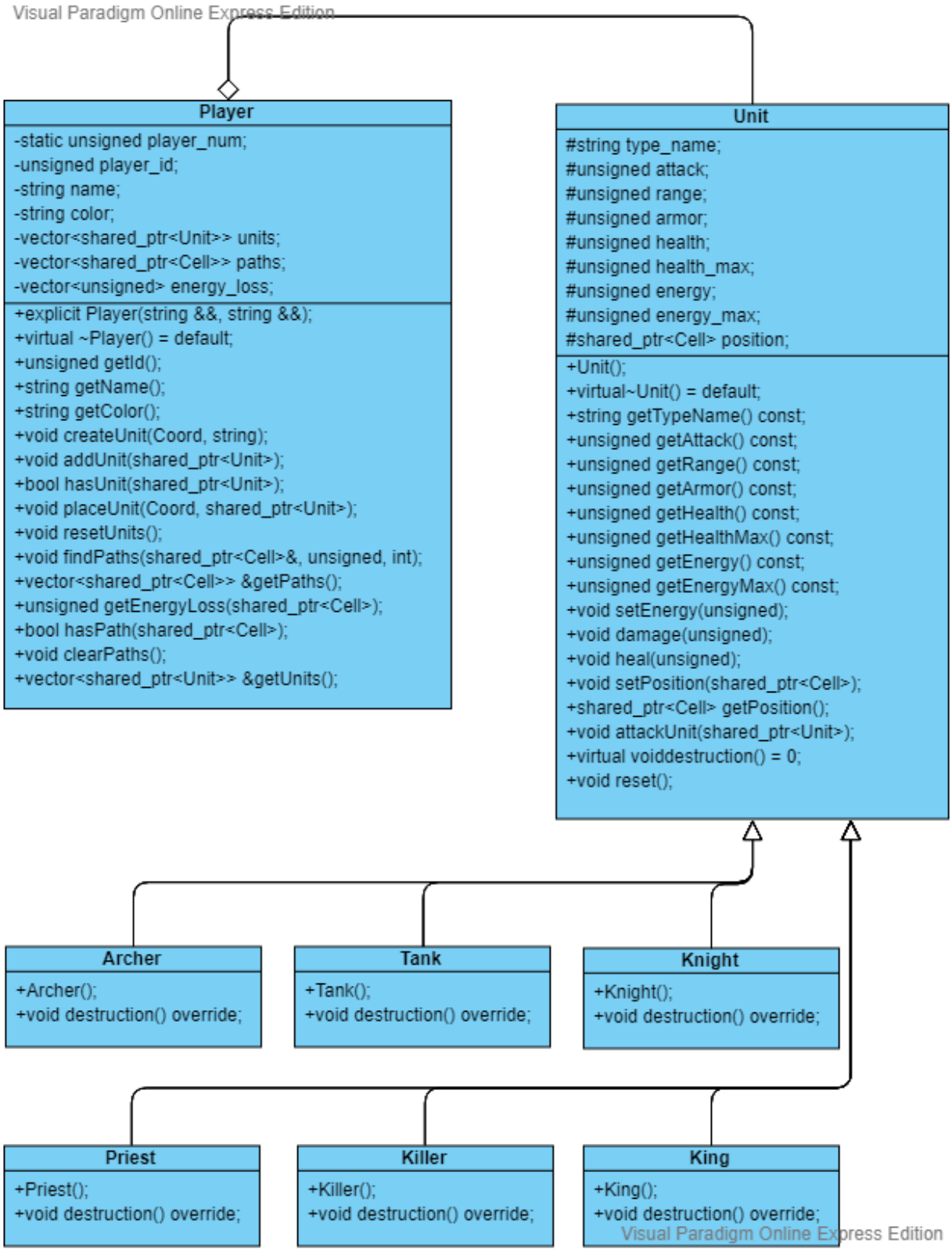

Рисунок 3. UML диаграмма класса Player

4. Был написан вспомогательный класс Coord для абстрагирования координат ячеек на игровом поле. А также класс исключений Exaption. Для логгирования был создан singleton класс-proxy Logger, интерфейс для него LoggerInterface и конкретная реализации логгера FileLogger, позволяющий записывать в лог-файл. UML диаграмма описанных классов представлена на рисунке 4.

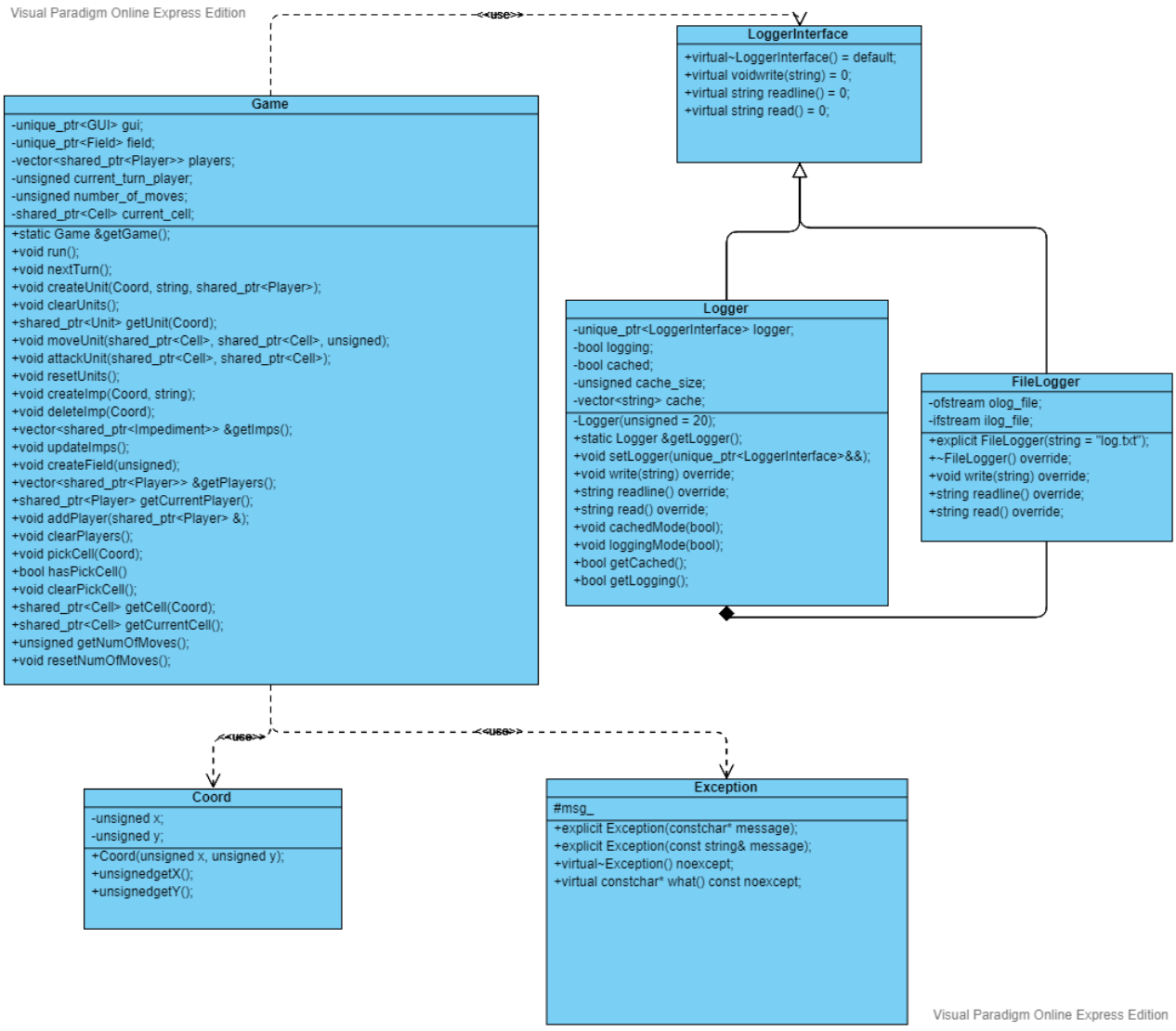

Рисунок 4. UML диаграмма логгера, класса исключений и координат.

Логгер позволяет использовать несколько режимов логгирования: с кэшированием, без кэширования. Режим с кэшированием позволяет откладывать запись в файл до тех пор, пока в кэше не накопится достаточное количество данных. Такой способ ведения логов уменьшает нагрузку на файловую систему.

5. Таким образом, в лабораторной работе было использованы следующие типы паттернов:

   Singleton – это класс Game и класс Logger. Причина использования: класс Game, как и Logger, в процессе выполнения программы используются только в одном экземпляре во многих частях кода. Поэтому необходимо обеспечить удобный и безопасный доступ к объекту этого класса. Для этого идеально подходит паттерн Singleton. Благодаря использованию этого паттерна, легко можно получить доступ к экземпляру данного класса из любой функции.

   Proxy – это класс Logger. Чтобы реализовать кэширование, надо каким-то образом перехватить операцию записи в лог-файл и переместить вместо этого данные в кэш. Для этого подходит паттерн Proxy. Proxy объект Logger перехватывает обращение к р FileLogger и кэширует информацию в случае необходимости. Аналогичным образом, в случае отключения логгера, proxy объект просто блокирует вызов функций для записи в файл.

   Mediator – класс Player и ImpedimentOrchestrator. Юниты постоянно взаимодействуют с игровым полем, и чтобы была возможность ими удобно управлять и получать различную информацию о них, был создан класс-посредник Player. Теперь, для осуществления каких-либо операций над объектами Unit, можно обращаться к классу Player, что значительно упрощает читаемость кода.Аналогично, класс ImpedimentOrchestrator используется для управления объектами типа Impediment.

6. Для реализации интерфейса использовалась библиотека Ultralight, позволяющая отображать JavaScript+HTML+CSS страницы прямо из C++ программы. Интерфейс программы представлен на рисунке 5.

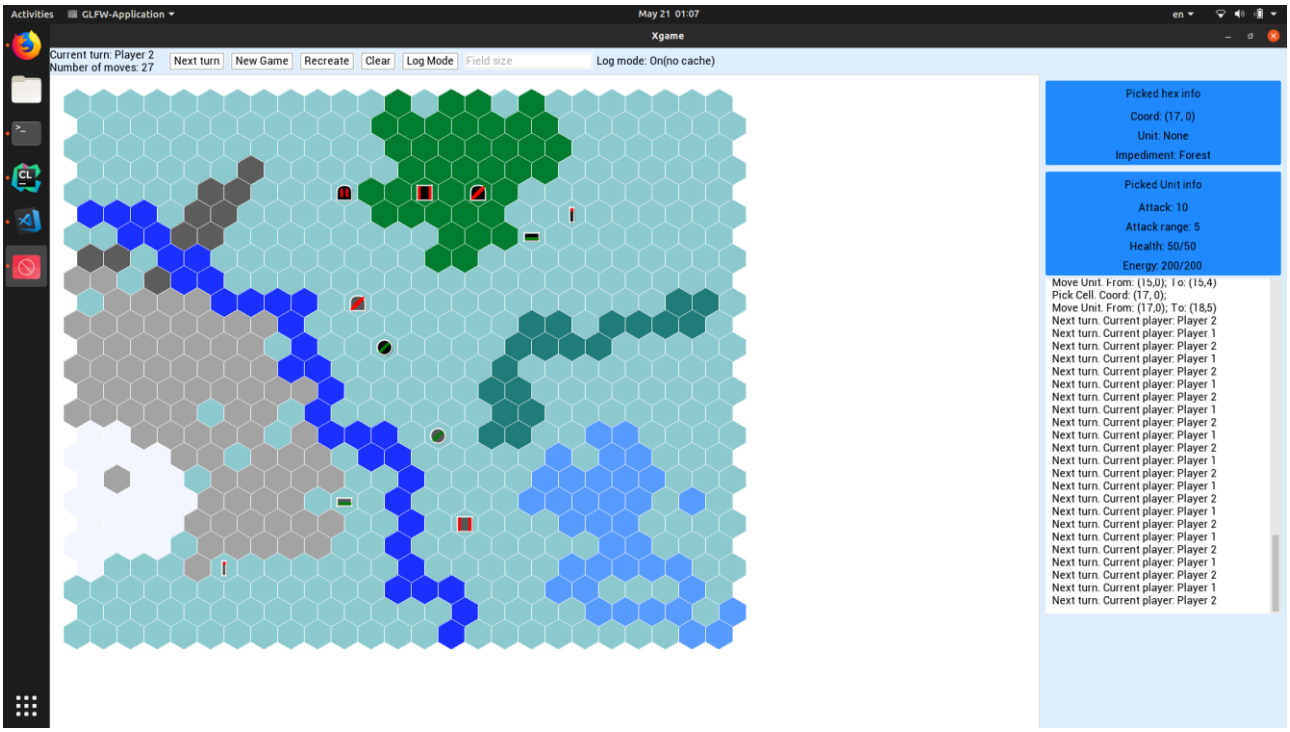

Рисунок 5. Интерфейс программы.

Описание кнопок:

New turn – завершение хода, переход к следующему игроку.

New Game – новая игра, при этом все юниты будут созданы заново на исходных позициях.

Recreate – генерация нового расположения препятствий, игра будет прекращена.

Clear – полная отчистка карты.

Log Mode – переключение режима ведения логов.

Field Size – размер поля. (20 > и < 35).

Справа отображается информация о составе выбранной ячейки, а также информация о юните, расположенном в текущей ячейке, если он есть.

7. В начале игры случайным образом генерируются препятствия на игровом поле. Создаются два игрока и по шесть юнитов каждого типа для игроков. Расположение юнитов сверху и снизу карты соответственно. Игроку дается неограниченное число попыток перемещения юнитов, однако энергия юнитов ограничена, поэтому, рано или поздно, игроку некуда будет ходить. После

того как игрок завершил ход, он может нажать на кнопку Next turn и завершить ход, передав управление другому игроку. Чтобы атаковать, игрок должен подойти на достаточное для атаки расстояние и, выбрав юнита, нажать на ячейку с вражеским юнитом. Игра заканчивается тогда, когда все вражеские юниты будут уничтожены. Игровой процесс представлен на рисунке 6.

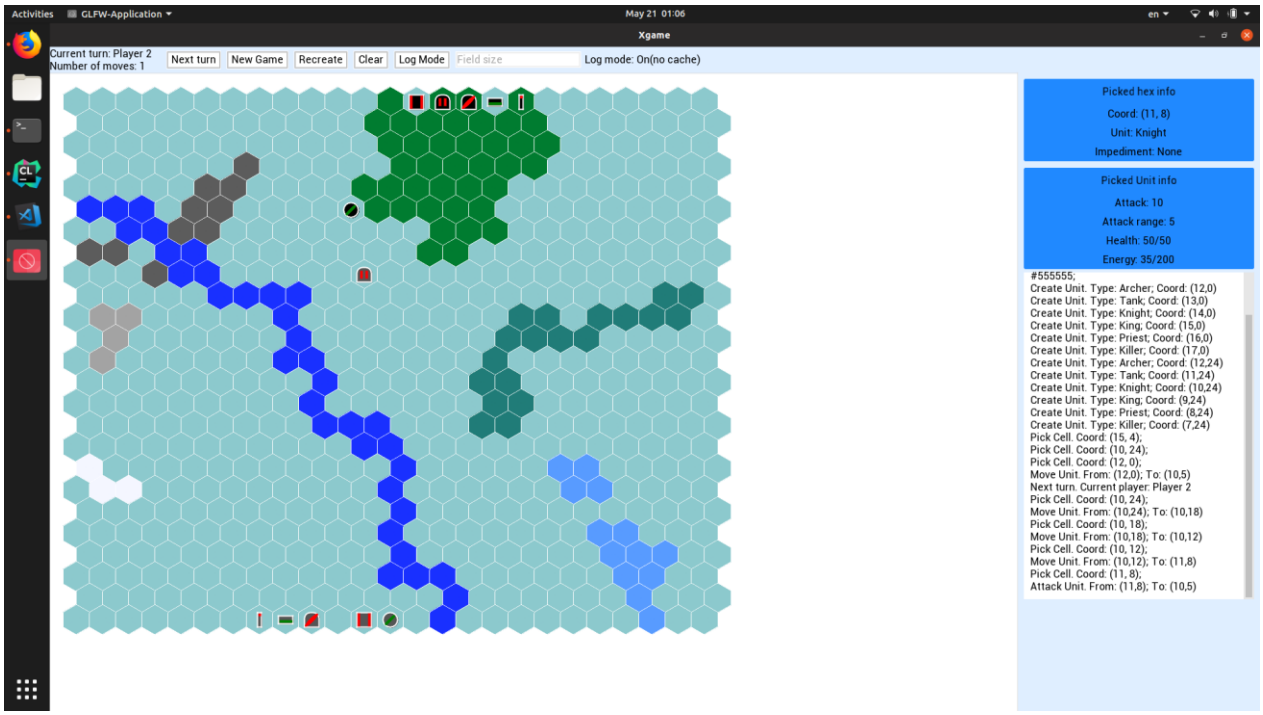

Рисунок 6. Игровой процесс

Вывод.

В ходе выполнения данной лабораторной работы были изучены паттерны проектирования. Была написана программа, симулирующая битву двух игроков на поле, состоящим из шестиугольных ячеек. В реализации программы были использованы такие паттерны проектирования, как Singleton (классы Logger и Game), Proxy (класс Logger), Mediator (класс ImpedimentOrchestrator и Player). Singleton - это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. Proxy - это структурный паттерн проектирования, который позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то *до* или *после* передачи вызова оригиналу. Mediator - это поведенческий паттерн проектирования, который позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник. Также был реализован графический интерфейс пользователя на основе библиотеки Ultralight.

Исходный код

Exception.h

```
#ifndef XGAME_EXCEPTION_H
#define XGAME_EXCEPTION_H

#include <exception>
#include <string>

using namespace std;

class Exception: public std::exception
{
public:
    explicit Exception(const char* message):
            msg_(message)
    {
    }

    explicit Exception(const string& message):
            msg_(message)
    {}

    virtual ~Exception() noexcept {}

    virtual const char* what() const noexcept {
        return msg_.c_str();
    }

protected:
    std::string msg_;
};


#endif //XGAME_EXCEPTION_H
```

Cell.cpp

```
#include <iostream>
#include "Cell.h"
#include "../gui/UI.h"
#include "../runtime/Game.h"

Cell::Cell(Coord c) : coord(c) {
    unit = shared_ptr<Unit>(nullptr);
    imp = shared_ptr<Impediment>(nullptr);
}

unsigned Cell::getX() { return coord.getX(); }
unsigned Cell::getY() { return coord.getY(); }

shared_ptr<Unit> Cell::getUnit() { return unit; }
void Cell::setUnit(shared_ptr<Unit> new_unit) { unit.swap(new_unit); }
void Cell::deleteUnit() { unit.reset(); }
bool Cell::hasUnit() { return !!unit; }

shared_ptr<Impediment> Cell::getImp() { return imp; }
void Cell::setImp(shared_ptr<Impediment> new_imp) { imp.swap(new_imp); }
void Cell::deleteImp() { imp.reset(); }
bool Cell::hasImp() { return !!imp; }
```

```
vector<shared_ptr<Cell>> &Cell::getReachable() { return reachable; }
```

Cell.h

```
#ifndef XGAME_CELL_H
#define XGAME_CELL_H

#include <memory>
#include <vector>
#include "../utility/Coord.h"

using namespace std;

class Impediment;
class Unit;

class Cell {
    Coord coord;
    shared_ptr<Unit> unit;
    shared_ptr<Impediment> imp;

    vector<shared_ptr<Cell>> reachable;

public:
    explicit Cell(Coord);


    unsigned getX();
    unsigned getY();

    shared_ptr<Unit> getUnit();
    void setUnit(shared_ptr<Unit>);
    void deleteUnit();
    bool hasUnit();

    shared_ptr<Impediment> getImp();
    void setImp(shared_ptr<Impediment>);
    void deleteImp();
    bool hasImp();

    vector<shared_ptr<Cell>> &getReachable();
};


#endif //XGAME_CELL_H
```

Field.cpp

```
using namespace std;

Field::Field(unsigned s) : field_size(s) {
    imp_orch = make_unique<ImpedimentOrchestrator>();
    fillContent();
}

unsigned Field::getSize() const { return field_size; }

void Field::fillContent() {

    for (auto i = 0; i < field_size; i++) {
```

```
        content.emplace_back();
        for (auto j = 0; j < field_size; j++)
            content[i].push_back(move(make_shared<Cell>(Coord(j, i))));
    }

    for (auto i = 0; i < field_size; i++) {
        for (auto j = 0; j < field_size; j++) {
            vector<shared_ptr<Cell>> &reachable = content[i][j]->getReachable();
            if(i % 2 == 0) {
                if(i - 1 >= 0 && i + 1 < field_size) {
                    if(j - 1 >= 0) {
                        reachable.push_back(content[i - 1][j - 1]);
                        reachable.push_back(content[i + 1][j - 1]);
                        reachable.push_back(content[i][j - 1]);
                    }
                    reachable.push_back(content[i - 1][j]);
                    reachable.push_back(content[i + 1][j]);
                    if(j + 1 < field_size)
                        reachable.push_back(content[i][j + 1]);
                    continue;
                } else if(i - 1 < 0) {
                    if(j - 1 >= 0) {
                        reachable.push_back(content[i + 1][j - 1]);
                        reachable.push_back(content[i][j - 1]);
                    }
                    reachable.push_back(content[i + 1][j]);
                    if(j + 1 < field_size)
                        reachable.push_back(content[i][j + 1]);
                    continue;
                } else if(i + 1 == field_size) {
                    if(j - 1 >= 0) {
                        reachable.push_back(content[i - 1][j - 1]);
                        reachable.push_back(content[i][j - 1]);
                    }
                    reachable.push_back(content[i - 1][j]);
                    if(j + 1 < field_size)
                        reachable.push_back(content[i][j + 1]);
                    continue;
                }
            } else {
                if(i - 1 >= 0 && i + 1 < field_size) {
                    if(j + 1 < field_size) {
                        reachable.push_back(content[i - 1][j + 1]);
                        reachable.push_back(content[i + 1][j + 1]);
                    }
                    if(j - 1 >= 0)
                        reachable.push_back(content[i][j - 1]);
                    reachable.push_back(content[i - 1][j]);
                    reachable.push_back(content[i + 1][j]);
                    if(j + 1 < field_size)
                        reachable.push_back(content[i][j + 1]);
                    continue;
                } else if(i + 1 == field_size) {
                    if(j + 1 < field_size)
                        reachable.push_back(content[i - 1][j + 1]);
                    if(j - 1 >= 0)
                        reachable.push_back(content[i][j - 1]);
                    reachable.push_back(content[i - 1][j]);
                    if(j + 1 < field_size)
                        reachable.push_back(content[i][j + 1]);
                    continue;
                }
            }
```

```
        }
    }

}

shared_ptr<Cell> Field::getCell(Coord c) { return content[c.getY()][c.getX()]; }

vector<shared_ptr<Impediment>>& Field::getImps() { return imp_orch->getImps(); }

ImpedimentOrchestrator& Field::getImpOrch() { return *imp_orch; }

void Field::clear() {
    imp_orch->clearAllImp();
}

vector<shared_ptr<Cell>> &Field::operator[](int idx) { return content[idx]; }
```

Field.h

```
#ifndef XGAME_FIELD_H
#define XGAME_FIELD_H

#include <vector>
#include "../utility/Coord.h"
#include "../field/Cell.h"
#include "../impediment/ImpedimentOrchestrator.h"

#define DEFAULT_FIELD_SIZE 20

using namespace std;

class Field {
    unsigned const field_size;
    vector<vector<shared_ptr<Cell>>> content;

    unique_ptr<ImpedimentOrchestrator> imp_orch;

    void fillContent();

public:
    explicit Field(unsigned = DEFAULT_FIELD_SIZE);

    unsigned getSize() const;

    shared_ptr<Cell> getCell(Coord);

    vector<shared_ptr<Impediment>> &getImps();

    ImpedimentOrchestrator &getImpOrch();

    void clear();

    vector<shared_ptr<Cell>> &operator[](int idx);

};


#endif //XGAME_FIELD_H
```

Impediment.cpp

```cpp
#include "Impediment.h"

unsigned Impediment::getPermeability() const { return permeability; }

string Impediment::getTypeName() const { return type_name; }

float Impediment::getProbability() const { return probability; }

vector<shared_ptr<Cell>> const& Impediment::getFilled() const { return filled; }

void Impediment::fillCell(shared_ptr<Cell> cell) { filled.push_back(cell); }

void Impediment::refillCell(shared_ptr<Cell> cell) {
    filled.erase(remove(filled.begin(), filled.end(), cell), filled.end());
}
```

Impediment.h

```cpp
#ifndef XGAME_IMPEDIMENT_H
#define XGAME_IMPEDIMENT_H

#include <vector>
#include <algorithm>
#include "../field/Cell.h"

using namespace std;

class Impediment {

protected:
    string type_name;
    unsigned permeability;
    float probability;
    vector<shared_ptr<Cell>> filled;

public:
    virtual ~Impediment() = default;

    string getTypeName() const;
    unsigned getPermeability() const;
    float getProbability() const;

    vector<shared_ptr<Cell>> const &getFilled() const;
    void fillCell(shared_ptr<Cell>);
    void refillCell(shared_ptr<Cell>);

    virtual void update() = 0;
    virtual void generate() = 0;

};


#endif //XGAME_IMPEDIMENT_H
```

Logger.cpp

```cpp
#include "Logger.h"


Logger::Logger(unsigned cs) {
    logger = unique_ptr<LoggerInterface>(nullptr);
```

```
    cached = 0;
    logging = 0;
    cache_size = cs;
}

Logger& Logger::getLogger() {
    static Logger instance;

    return instance;
}

void Logger::setLogger(unique_ptr<LoggerInterface> &&l) {
    logger.swap(l);
}

void Logger::write(string str) {
    if(!logging) return;

    if(cached && cache.size() < cache_size) {
        cache.push_back(str);
    } else if(cached) {
        for(auto &s : cache)
            logger->write(s);
        cache.clear();
        cache.push_back(str);
    }

    if(!cached)
        logger->write(str);
}

string Logger::readline() {
    return logger->readline();
}

string Logger::read() {
    return logger->read();
}

void Logger::cachedMode(bool) {
    cached = !cached;
}

void Logger::loggingMode(bool) {
    logging = !logging;
}

bool Logger::getCached() {
    return cached;
}

bool Logger::getLogging() {
    return logging;
}


FileLogger::FileLogger(string file_name) {
    ilog_file.open(file_name);
    olog_file.open(file_name);
}

FileLogger::~FileLogger() {
```

```
    ilog_file.close();
    olog_file.close();
}

void FileLogger::write(string str) {
    olog_file << str << endl;
    olog_file.flush();
}

string FileLogger::readline() {
    string line;

    getline(ilog_file, line);

    return line;
}

string FileLogger::read() {
    string line;
    stringstream result;

    ilog_file.clear();
    ilog_file.seekg(0, ios::beg);

    while(getline(ilog_file, line))
        result << line << "<br>";

    return result.str();
}
```

Logger.h

```
#ifndef XGAME_LOGGER_H
#define XGAME_LOGGER_H

#include <fstream>
#include <string>
#include <memory>
#include <vector>
#include <sstream>
#include <iostream>

using namespace std;


class LoggerInterface {

public:
    virtual ~LoggerInterface() = default;

    virtual void write(string) = 0;
    virtual string readline() = 0;
    virtual string read() = 0;
};


class Logger : public LoggerInterface {
    Logger(unsigned = 20);

    unique_ptr<LoggerInterface> logger;

    bool logging;
```

```
    bool cached;
    unsigned cache_size;
    vector<string> cache;

public:
    Logger(Logger const&)           = delete;
    void operator=(Logger const&)   = delete;

    static Logger &getLogger();

    void setLogger(unique_ptr<LoggerInterface>&&);

    void write(string) override;
    string readline() override;
    string read() override;

    void cachedMode(bool);
    void loggingMode(bool);

    bool getCached();
    bool getLogging();
};


class FileLogger : public LoggerInterface {
    ofstream olog_file;
    ifstream ilog_file;

public:
    explicit FileLogger(string = "log.txt");
    ~FileLogger() override;

    void write(string) override;
    string readline() override;
    string read() override;

};
#endif //XGAME_LOGGER_H
```

Player.cpp

```
#include <algorithm>
#include <iostream>
#include "../runtime/Game.h"
#include "Player.h"
#include "../exception/Exception.h"

unsigned Player::player_num = 0;

Player::Player(string &&n, string &&c) : name(n), color(c) {
    player_id = player_num++;

    Game &game = Game::getGame();
    Field &field = game.getField();
}

unsigned Player::getId() { return player_id; }

string Player::getName() { return name; }

string Player::getColor() { return color; }
```

```
void Player::createUnit(Coord c, string unit_type) {
    shared_ptr<Unit> unit;
    if(unit_type == "Archer")
        unit = make_shared<Archer>();
    else if(unit_type == "Tank")
        unit = make_shared<Tank>();
    else if(unit_type == "Knight")
        unit = make_shared<Knight>();
    else if(unit_type == "King")
        unit = make_shared<King>();
    else if(unit_type == "Priest")
        unit = make_shared<Priest>();
    else if(unit_type == "Killer")
        unit = make_shared<Killer>();
    else
        throw Exception("Unknown unit type: " + unit_type);

    addUnit(unit);
    placeUnit(c, unit);
}
void Player::addUnit(shared_ptr<Unit> unit) { units.push_back(unit); }
bool Player::hasUnit(shared_ptr<Unit> unit) { return find(units.begin(),
units.end(), unit) != units.end(); }
void Player::placeUnit(Coord c, shared_ptr<Unit> unit) {
    Game &game = Game::getGame();
    Field &field = game.getField();

    shared_ptr<Cell> cell = field.getCell(c);
    cell->setUnit(unit);
    unit->setPosition(cell);
}
void Player::resetUnits() {
    for(auto &unit : units) {
        unit->reset();
    }
}

void Player::findPaths(shared_ptr<Cell> &cell, unsigned energy, int deep) {
    unsigned permeability = 5;

    if(cell->hasImp()) permeability = cell->getImp()->getPermeability();

    if(find(paths.begin(), paths.end(), cell) != paths.end() || energy <
permeability || deep < 0) return;

    vector<shared_ptr<Cell>> reachable = cell->getReachable();

    paths.push_back(cell);
    energy_loss.push_back(energy - permeability);
    for(auto &next_cell : reachable)
        findPaths(next_cell, energy - permeability, deep - 1);
}

vector<shared_ptr<Cell>>& Player::getPaths() { return paths; }

unsigned Player::getEnergyLoss(shared_ptr<Cell> c) {
    if(hasPath(c)) {
        auto it = find(paths.begin(), paths.end(), c);
        return energy_loss[distance(paths.begin(), it)];
    } else {
        return 0;
    }
}
```

```
bool Player::hasPath(shared_ptr<Cell> c) { return find(paths.begin(),
paths.end(), c) != paths.end(); }

void Player::clearPaths() {
    paths.clear();
    energy_loss.clear();
}

vector<shared_ptr<Unit>>& Player::getUnits() { return units; }
```

Player.h

```
#ifndef XGAME_PLAYER_H
#define XGAME_PLAYER_H

#include <memory>
#include <string>
#include <vector>
#include "../field/Cell.h"

using namespace std;

class Player {
    static unsigned player_num;
    unsigned player_id;
    string name;
    string color;

    vector<shared_ptr<Unit>> units;

    vector<shared_ptr<Cell>> paths;
    vector<unsigned> energy_loss;

public:
    explicit Player(string &&, string &&);
    virtual ~Player() = default;

    unsigned getId();
    string getName();
    string getColor();

    void createUnit(Coord, string);
    void addUnit(shared_ptr<Unit>);
    bool hasUnit(shared_ptr<Unit>);
    void placeUnit(Coord, shared_ptr<Unit>);
    void resetUnits();

    void findPaths(shared_ptr<Cell>&, unsigned, int);
    vector<shared_ptr<Cell>> &getPaths();
    unsigned getEnergyLoss(shared_ptr<Cell>);
    bool hasPath(shared_ptr<Cell>);
    void clearPaths();

    vector<shared_ptr<Unit>> &getUnits();
};


#endif //XGAME_PLAYER_H
```

Game.cpp

```
#include <iostream>
#include "Game.h"
#include "../logger/Logger.h"
#include "../exception/Exception.h"

Game::Game() {
    gui = make_unique<GUI>();
    field = unique_ptr<Field>(nullptr);

    current_turn_player = 0;

    number_of_moves = 0;

    current_cell = shared_ptr<Cell>(nullptr);

    Logger &logger = Logger::getLogger();
    logger.setLogger(make_unique<FileLogger>());
    logger.loggingMode(true);
}

Game &Game::getGame() {
    static Game instance;

    return instance;
}

void Game::run() { gui->run(); }

void Game::nextTurn() {

    unsigned num_of_players = players.size();

    if(num_of_players == current_turn_player + 1 || !num_of_players)
current_turn_player = 0;
    else current_turn_player++;

    updateImps();
    clearPickCell();
    resetUnits();

    number_of_moves++;

    Logger &logger = Logger::getLogger();
    logger.write("Next turn. Current player: " + players[current_turn_player]-
>getName());
}

void Game::createUnit(Coord c, string unit_type, shared_ptr<Player> player) {

    try {
        player->createUnit(c, unit_type);

        Logger &logger = Logger::getLogger();
        logger.write(
                "Create Unit. Type: " + unit_type + "; Coord: " + "(" +
to_string(c.getX()) + "," + to_string(c.getY()) +
                ")");
    } catch (Exception &err) {
        Logger &logger = Logger::getLogger();
        logger.write(err.what());
    }
```

```
}

void Game::clearUnits() {
    for (auto &player : players) {
        auto units = player->getUnits();
        for (auto &unit : units) {
            if (!unit->getPosition()->hasUnit())
                player->getUnits().erase(std::remove(player->getUnits().begin(),
player->getUnits().end(), unit),
                                         player->getUnits().end());
        }
    }
}

shared_ptr<Unit> Game::getUnit(Coord) {

}

void Game::moveUnit(shared_ptr<Cell> from, shared_ptr<Cell> to, unsigned
energy_loss) {
    to->setUnit(from->getUnit());
    from->deleteUnit();
    to->getUnit()->setPosition(to);
    to->getUnit()->setEnergy(energy_loss);

    Logger &logger = Logger::getLogger();
    logger.write(
            "Move Unit. From: (" + to_string(from->getX()) + "," +
to_string(from->getY()) +
            "); To: (" + to_string(to->getX()) + "," + to_string(to->getY()) +
")");
}

void Game::attackUnit(shared_ptr<Cell> from, shared_ptr<Cell> to) {
    shared_ptr<Unit >unit1 = from->getUnit();
    shared_ptr<Unit >unit2 = to->getUnit();

    unit1->attackUnit(unit2);


    Logger &logger = Logger::getLogger();
    logger.write(
            "Attack Unit. From: (" + to_string(from->getX()) + "," +
to_string(from->getY()) +
            "); To: (" + to_string(to->getX()) + "," + to_string(to->getY()) +
")");

}

void Game::resetUnits() {
    for(auto &player : players)
        player->resetUnits();
}

void Game::createImp(Coord c, string imp_type) {

    try {
        ImpedimentOrchestrator &imp_orch = field->getImpOrch();
        imp_orch.createImp(c, imp_type);
    } catch (Exception &err) {
        Logger &logger = Logger::getLogger();
        logger.write(err.what());
```

```
    }

}

void Game::deleteImp(Coord c) {
    ImpedimentOrchestrator &imp_orch = field->getImpOrch();

    imp_orch.deleteImp(field->getCell(c)->getImp());
}

vector<shared_ptr<Impediment>> &Game::getImps() { return field->getImps(); }

void Game::updateImps() {
    ImpedimentOrchestrator &imps_orch = field->getImpOrch();

    imps_orch.updateImps();
}

void Game::createField(unsigned field_size) { field =
make_unique<Field>(field_size); }

Field &Game::getField() { return *field; }

vector<shared_ptr<Player>>& Game::getPlayers() { return players; }

shared_ptr<Player> Game::getCurrentPlayer() {
    return players.size() ? players[current_turn_player] :
shared_ptr<Player>(nullptr);
}

void Game::addPlayer(shared_ptr<Player> &player) {

    players.push_back(player);

    Logger &logger = Logger::getLogger();
    logger.write("Create Player. Name: " + player->getName() + "; Color: " +
player->getColor() + ";");
}

void Game::clearPlayers() {
    players.clear();
    current_cell.reset();
    current_turn_player = 0;
}

void Game::pickCell(Coord c) {

    if (!!getCurrentPlayer() && hasPickCell() && getCurrentPlayer()-
>hasUnit(getCurrentCell()->getUnit()) &&
        !getCell(c)->hasUnit()) {
        if (getCurrentPlayer()->hasPath(getCell(c))) {

            moveUnit(getCurrentCell(), getCell(c),
                     getCurrentPlayer()->getEnergyLoss(getCell(c)));

            getCurrentPlayer()->clearPaths();

            return;
        }
    } else if (!!getCurrentPlayer() && hasPickCell() && getCell(c)->hasUnit() &&
               !getCurrentPlayer()->hasUnit(getCell(c)->getUnit()) &&
               getCurrentPlayer()->hasPath(getCell(c))) {
```

```
        attackUnit(getCurrentCell(), getCell(c));

        clearUnits();
        getCurrentPlayer()->clearPaths();

        for(auto &player : players) {
            if(player->getUnits().size() == 0) {
                Logger &logger = Logger::getLogger();
                logger.write(
                        "Player " + getCurrentPlayer()->getName() + " win!");
            }
        }

        return;
    }

    current_cell = field->getCell(c);

    if(!!getCurrentPlayer())
        getCurrentPlayer()->clearPaths();

    if(!!getCurrentPlayer() && current_cell->hasUnit() && getCurrentPlayer()-
>hasUnit(current_cell->getUnit()))
        getCurrentPlayer()->findPaths(current_cell, current_cell->getUnit()-
>getEnergy(), 6);

    if(current_cell->hasImp()) {

    }

    Logger &logger = Logger::getLogger();
    logger.write("Pick Cell. Coord: (" + to_string(current_cell->getX()) + ", "
+ to_string(current_cell->getY()) + ");");
}

bool Game::hasPickCell() { return !!current_cell; }

void Game::clearPickCell() { current_cell.reset(); }

shared_ptr<Cell> Game::getCell(Coord c) { return field->getCell(c); }

shared_ptr<Cell> Game::getCurrentCell() { return current_cell; }

unsigned Game::getNumOfMoves() { return number_of_moves; }

void Game::resetNumOfMoves() { number_of_moves = 0; }
```

Game.h

```
#ifndef XGAME_GAME_H
#define XGAME_GAME_H

#include <vector>
#include "../field/Field.h"
#include "../utility/Coord.h"
#include "../unit/Unit.h"
#include "../impediment/Impediment.h"
#include "../player/Player.h"
#include "../gui/GUI.h"

using namespace std;
```

```
class Game {
    Game();

    unique_ptr<GUI> gui;
    unique_ptr<Field> field;

    vector<shared_ptr<Player>> players;
    unsigned current_turn_player;
    unsigned number_of_moves;

    shared_ptr<Cell> current_cell;

public:
    Game(Game const&)             = delete;
    void operator=(Game const&)   = delete;

    static Game &getGame();

    void run();

    void nextTurn();

    void createUnit(Coord, string, shared_ptr<Player>);
    void clearUnits();
    shared_ptr<Unit> getUnit(Coord);
    void moveUnit(shared_ptr<Cell>, shared_ptr<Cell>, unsigned);
    void attackUnit(shared_ptr<Cell>, shared_ptr<Cell>);
    void resetUnits();

    void createImp(Coord, string);
    void deleteImp(Coord);
    vector<shared_ptr<Impediment>> &getImps();
    void updateImps();

    void createField(unsigned);
    Field &getField();

    vector<shared_ptr<Player>> &getPlayers();
    shared_ptr<Player> getCurrentPlayer();
    void addPlayer(shared_ptr<Player> &);
    void clearPlayers();

    void pickCell(Coord);
    bool hasPickCell();
    void clearPickCell();
    shared_ptr<Cell> getCell(Coord);
    shared_ptr<Cell> getCurrentCell();

    unsigned getNumOfMoves();
    void resetNumOfMoves();

};


#endif //XGAME_GAME_H
```

main.cpp

```
#include <iostream>
#include <string>
#include <Ultralight/Ultralight.h>
```

```cpp
#include <AppCore/AppCore.h>
#include "runtime/Game.h"
#include "field/Field.h"
#include "player/Player.h"

using namespace std;
using namespace ultralight;

int main() {

    Game &game = Game::getGame();

    game.createField(25);

    game.run();

    return 0;
}
```