**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра математического обеспечения и применения ЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Полиморфизм**

Студент гр. 8381 _____ Облизанов А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

**Цель работы.**

Реализовать набор классов, для ведения логирования действий и состояний программы.

**Задание.**

Основные требования:

- Логирование действий пользователя
- Логирование действий юнитов и базы
- Реализована возможность записи логов в файл и в терминал
- Взаимодействие с файлами должны быть по идиоме RAII
- Для логирования состояний перегружен оператор вывода в поток
- Переключение между разным логированием (логирование в файл, в терминал, без логирования) реализуется при помощи паттерна “Прокси”
- Реализован разный формат записи при помощи паттерна “Адаптер”

**Выполнение работы.**

Написание работы производилось на базе операционной системы Windows 10 в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator, запуск программы осуществлялся через командную строку. Исходные коды файлов программы представлены в приложениях А-И.

**Система логирования**

Система логирования в данной работе следующая:

- Классы логирования в файл и терминал (а также прокси класс) обладают возможностью вывода входной строки и текущего времени в поток

- Класс адаптера обладает возможностью преобразовывать входные параметры в строки разного вида для последующего вывода через классы логирования в файл и терминал.
- Фасад и команды (файлы facade.h/cpp, command.h/cpp) логируют действия пользователя и изменения в игре через класс адаптера, отправляя ему необходимые данные.

Далее рассмотрим каждый элемент системы подробнее.

**Классы логирования в файл, в терминал**

Классы для логирования в файл и терминал представлены в табл. 1.

Таблица 1 – Классы для логирования в файл и терминал

| Класс | Назначение и особенности |
|---|---|
| Time | Описан в файлах logger.h и logger.cpp. Предназначен для вывода в поток текущего времени.<br>*Используется* ***перегрузка оператора вывода в поток .*** |
| ILogger | Описан в файле basicinterfaces.h. Интерфейс класса для логирования в поток (далее логгера).<br>• Метод printLog(std::string info) выводит в поток время и переданную строку.<br>• Наследуемый от интерфейса класс может хранить в себе строку для вывода. Для установки строки используется метод setInfo(), для вывода сохраненной строки в поток используется print().<br>• Класс может вывести в поток только время – printTime(). |
| TerminalLogger (наследован от ILogger) | Описан в файлах logger.h и logger.cpp. Предназначен для вывода в терминал логов.<br>• В деструкторе логгера выводится лог о завершении его работы.<br>• В качестве потока используется std::cout.<br>*Используется* ***перегрузка оператора вывода в поток.*** |

Окончание таблицы 1

| | |
|---|---|
| FileLogger (наследован от ILogger) | Описан в файлах logger.h и logger.cpp. Предназначен для вывода в терминал логов.<br>• В конструкторе файл открывается на потоковый вывод, в случае ошибки бросается исключение. В деструкторе выводится лог о завершении и файл закрывается.<br>• В качестве потока используется std::ofstream.<br>*Используется **перегрузка оператора вывода в поток** (та же перегрузка, что и в TerminalLogger).* |

**Переключение между разным логированием**

Переключение между разным логированием (логирование в файл, в терминал, без логирования) по принципу паттерна «Прокси» реализуется в классе ProxyLogger (файлы logger.h/cpp). Для удобства было введено перечисление enum LoggerType, содержащее в себе значения NO_LOGGER, TO_TERMINAL, TO_FILE.

- Класс унаследован от интерфейса ILogger и не содержит новых методов.
- В конструктор передается желаемый тип логгера LoggerType, и, если значение не равно NO_LOGGER, создается соответствующий логгер, указатель на который хранится в приватном поле класса. Если логирование отключено, указателю задается значение nullptr.
- В методах класса есть проверка на то, включено ли логирование, логгер вызывается только если оно включено.
- В деструкторе класса, если логгер был создан, логгер удаляется.

Для полного описание процесса переключения логирования необходимо рассмотреть остальные элементы системы.

**Адаптер**

Класс LogAdapter реализован по принципу паттерна «адаптер» (файлы logadater.h/cpp). В классе реализована обработка информации (параметров) для

формирования строки и вывод ее логгером. Логгер хранится в классе в приватном поле.

Для реализации логирования было создано перечисление LogEvent, элементы которого можно разбить на две части:

- USER_* – Сообщения о действиях пользователя
- Все остальное – сообщения об изменениях в игровом процессе

Также было создано перечисление LogType для ***двух форматов вывода***: SHORT, FULL. Все перечисления описаны в файле libraries.h.

Основные методы класса и примеры их вывода приведены в табл. 2.

Таблица 2 – Методы класса LogAdapter

| Название | Назначение, особенности |
|---|---|
| setLogger | Метод для установки метода логирования |
| setType | Метод для установки формата вывода (SHORT – краткий, FULL – полный) |
| pushLog | Метод вывода лога. Передаются параметры:<br>• Тип события (LogEvent)<br>• Массив параметров std::vector<int> param (задано значение пустого по умолчанию для случаев, когда он не нужен)<br>В методе pushLog() осуществляется выбор одного из методов make* для формирования строки и вывод ее через логгер его методом printLog(). Если логгер не установлен, вывод не происходит. |
| makeUserActionLog | • Формирование строки лога действия пользователя<br>• Принимает на вход тип события LogEvent (для нажатия разных кнопок в UI свое событие)<br>• Пример сформированной строки (тип FULL): |
| `11:7:8 >> USER \| Request: create new base` | |

| | |
|---|---|
| makeGameCreateLog | • Формирование строки лога создания игры<br>• Принимает и выводит на вход параметры: размер поля, максимальное число объектов на поле<br>• Пример сформированной строки (тип FULL): |
| `20:25:7 >> Created: GAME with FIELD \| size: width=6 height=15 \| item limit: 100` | |
| makeBaseAddLog | • Формирование строки лога создания базы<br>• Принимает на вход координаты базы, ее номер, число объектов на поле<br>• Пример сформированной строки (тип FULL): |
| `20:25:19 >> Created: BASE at coords: X=0 Y=0 \| game bases number: 1 \| Update: field item counter: 2` | |
| makeNeutralAddLog | • Формирование строки лога создания нейтрального объекта<br>• Принимает на вход тип объекта (enum NeutralType), координаты, число объектов на поле<br>• Пример сформированной строки (тип FULL): |
| `20:25:18 >> Created: neutral CHEST \| coords: X=1 Y=2 \| Update: field item counter: 1` | |
| makeUnitAddLog | • Формирование строки лога создания юнита<br>• Принимает на вход тип юнита (enum UnitType), координаты, номер базы<br>• Пример сформированной строки (тип FULL): |
| `20:25:22 >> Created: unit ATTACK SCIENTIST \| coords: X=1 Y=1 \| base: number=1` | |
| makeBaseUpdateLog | • Формирование строки лога состояния базы<br>• Принимает на вход координаты, число юнитов, максимальное число юнитов, а также типы всех юнитов базы (enum UnitType)<br>• Пример сформированной строки (тип FULL): |
| `20:25:22 >> Update: BASE \| coords: X=0 Y=0 \| unit counter: 1 \| unit limit: 100 \| units: ATT_SC` | |

| | |
|---|---|
| makeAttackLog | • Формирование строки лога атаки юнита<br>• Принимает на вход координаты атакующего, его тип, координаты смещения атаки, флаг убийства<br>• Пример сформированной строки (тип FULL): |
| `20:36:5 >> Attacked: FROM unit ATT_SC \| coords: X=1 Y=1 \| TO coords: X=2 Y=2` | |
| makeCounterUpdate | • Формирование строки лога числа объектов на поле<br>• Принимает на вход число объектов на поле<br>• Пример сформированной строки (тип FULL): |
| `\| Update: field item counter: 2` | |
| makeMoveLog | • Формирование строки лога передвижения юнита<br>• Принимает на вход координаты юнита, его тип, координаты смещения передвижения<br>• Пример сформированной строки (тип FULL): |
| `20:39:5 >> Moved: unit ATT_SC \| FROM: X=1 Y= 1 \| BY: X=3 Y= 4` | |

**Сравнение форматов записи FULL и SHORT**

Сравнение форматов приведено в табл. 3.

Таблица 3 – Сравнение форматов записи

| **Тип** | **Вывод (добавление юнита)** |
|---|---|
| SHORT | Created: unit ATT_SC \| base: N=1 |
| FULL | Created: unit ATTACK SCIENTIST \| coords: X=1 Y=2 \| base: number=1 |
| **Тип** | **Вывод (состояние базы)** |
| SHORT | Update: BASE \| coords: X=1 Y=1 \| uc: 3 \| ul: 100 |
| FULL | Update: BASE \| coords: X=1 Y=1 \| unit counter: 4 \| unit limit: 100<br>\| units: SIM_SC ATT_SC ATT_SC SIM_SC |

**Логирование фасадом и командами**

Фасад и команды расположены в файлах facade.h/cpp, command.h/cpp.

Настройка логирования происходит по следующим правилам:

- Создает и хранит в себе адаптера фасад (UIFacade)
- У фасада есть методы, реагирующие на действия в UI, для переключения логгеров. Новый логгер (его «прокси») устанавливается в адаптер методом setLogger().
- Каждая из следующих команд также хранит указатель на адаптер, который получает от фасада
    - GameCommand
    - FieldCommand
    - BaseCommand

Логирование событий происходит по следующим правилам:

- Фасад логирует все действия пользователя, то есть нажатия кнопок. Для этого он обращается к LogAdapter, передавая ему только тип события, например, следующим вызовом выводится лог нажатия пользователем кнопки вывода основной информации об игре:
    - logger->pushLog(USER_GAME_INFO);
- Команды логируют следующие изменения в игре:
    - Создание игры (размеры поля, максимальное число объектов)
    - Добавление базы (координаты, номер базы, число объектов на поле)
    - Добавление нейтрального объекта (тип объекта, координаты, число объектов на поле)
    - Добавление юнита (тип юнита, координаты, номер базы, число объектов на поле)
    - Обновление информации о базе (параметры, список юнитов)
    - Передвижение юнита (тип юнита, местоположение, координаты перемещения)
    - Атака юнита (тип юнита, местоположение, координаты атаки, убийство)

- Команды во время работы формируют массив выходных параметров типа int и возвращают его фасаду. Для оптимизации и экономии памяти почти во всех случаях адаптер принимает на вход именно этот массив. Таким образом, один массив используется и для вывода информации на экран в фасаде, и для логирования.

**UML-диаграмма**

UML диаграмма представлена на рис. 1.

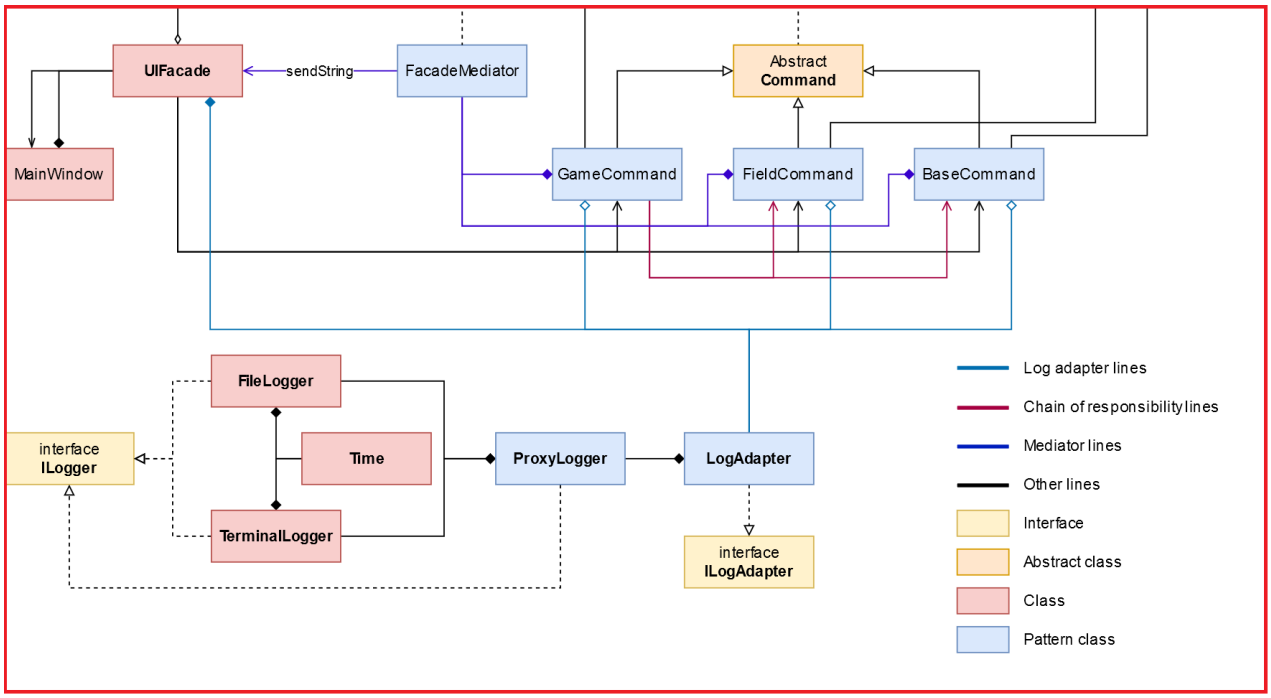


Рисунок 1 – UML-диаграмма

**Демонстрационные примеры**

В пользовательском интерфейсе есть кнопки для переключения записи логов в файл, в терминал, либо без записи. Также есть кнопки для переключения форматов логов: короткие или полные. Кнопки отмечены красным прямоугольником на рис. 2.

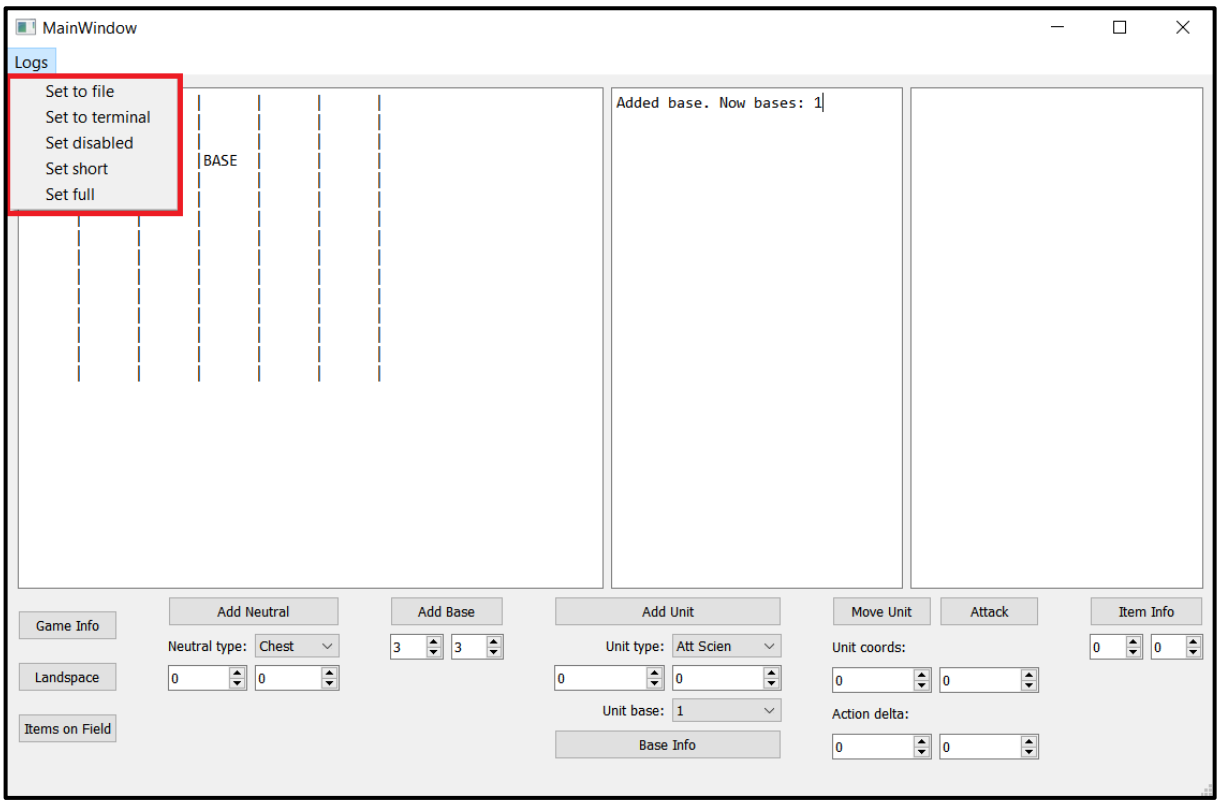


Рисунок 2 – Кнопки настройки логов

Демонстрационный вид файла логов после завершения игры представлен на рис. 3.

```
22:28:30 >> Created: GAME with FIELD | size: width=6 height=15 | item limit: 100
22:28:39 >> USER | Request: create new base
22:28:39 >> Created: BASE at coords: X=3 Y=3 | game bases number: 1
         | Update: field item counter: 1
22:30:18 >> USER | Request: create neutral
22:30:18 >> Created: neutral CHEST | coords: X=1 Y=5
         | Update: field item counter: 2
22:30:21 >> USER | Request: create neutral
22:30:21 >> Created: neutral ANTIVIRUS | coords: X=3 Y=2
         | Update: field item counter: 3
22:30:26 >> USER | Request: create unit
22:30:26 >> Created: unit SUPPORT BERSERK | coords: X=3 Y=4 | base: number=1
22:30:26 >> USER | Request: base info
22:30:26 >> Update: BASE | coords: X=3 Y=3 | unit counter: 1 | unit limit: 100
         | units: SUP_BS
22:30:33 >> USER | Request: create unit
22:30:33 >> Created: unit ATTACK BERSERK | coords: X=2 Y=4 | base: number=1
22:30:33 >> USER | Request: base info
22:30:33 >> Update: BASE | coords: X=3 Y=3 | unit counter: 2 | unit limit: 100
         | units: ATT_BS SUP_BS
22:30:43 >> USER | Request: move unit
22:30:43 >> Moved: unit ATT_BS | FROM: X=2 Y= 4 | BY: X=-2 Y= -4
22:30:47 >> USER | Request: item by XY info
22:30:50 >> USER | Request: land info
22:30:51 >> USER | Request: game info
22:30:58 >> USER | Request: base info
22:30:58 >> Update: BASE | coords: X=3 Y=3 | unit counter: 2 | unit limit: 100
         | units: ATT_BS SUP_BS
22:31:0 << Logger stopped
```

Рисунок 3 – Демонстрационный вид файла логов

**Выводы.**

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для логирования состояний программы. Были использованы паттерны проектирования, а также принципы объектно-ориентированного программирования.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Game *game = new Game(6, 15, 100, OPEN);
    MainWindow w(nullptr, game);
    w.show();
    return a.exec();
}
```

## ПРИЛОЖЕНИЕ Б

## ИСХОДНЫЙ КОД ПРОГРАММЫ. LIBRARIES.H

```
#ifndef LIBRARIES_H
#define LIBRARIES_H

#include <ctime>
#include <iostream>
#include <string>
#include <vector>
#include <list>
#include <time.h>
#include <fstream>

#define LOGFILE "log.txt"

enum LoggerType {
    NO_LOGGER,
    TO_TERMINAL,
    TO_FILE
};

enum UnitType {
    ATT_SC,
    ATT_BS,
    SUP_SC,
    SUP_BS,
    SIM_SC,
    SIM_BS
};

enum BaseType {
    BASE = 6
};

enum LandType {
    OPEN,
    OPEN_B,
    VPN,
    VPN_B,
    FAST,
    FAST_B
};

enum NeutralType{
    CHEST = 7,
    KEYGEN,
```

```
    ANTIVIRUS,
    DATA
};

enum RequestType{
    GAME_INFO = 100,
    BASE_INFO,
    LAND_INFO,
    ITEMS_INFO,
    ADD,
    FIND,
    MOVE,
    GETXY,
    ATTACK
};

enum LogEvent{
    USER_GAME_INFO = 200,
    USER_BASE_INFO,
    USER_LAND_INFO,
    USER_ITEMS_INFO,
    USER_ITEM_INFO,
    USER_BASE_ADD,
    USER_UNIT_ADD,
    USER_NEUTRAL_ADD,
    USER_MOVE,
    USER_ATTACK,

    GAME_CREATE,
    BASE_ADD,
    NEUTRAL_ADD,
    UNIT_ADD,
    BASE_UPDATE,
    COUNTER_UPDATE,
    UNIT_ATTACK,
    UNIT_MOVE
};

enum LogType{
    SHORT = 300,
    FULL
};

#endif // LIBRARIES_H
```

## ПРИЛОЖЕНИЕ В

## ИСХОДНЫЙ КОД ПРОГРАММЫ. LOGGER.H

```
#ifndef LOGGER_H
#define LOGGER_H
#include "basicinterfaces.h"
#include <fstream>

class Time
{
public:
    Time() {
        time_t timer;
        time(&timer);
        lastTime = localtime(&timer);
    }

    void update() {
        time_t timer;
        time(&timer);
        lastTime = localtime(&timer);
    }

    friend std::ostream& operator<<(std::ostream& os, Time& tm);
private:
    tm *lastTime;
};

class TerminalLogger : public ILogger
{
public:
    TerminalLogger();

    void printTime();
    void printLog(std::string info);
    void print();
    bool isActive();

    Time *getLocalTime() const;
    std::string getInfo() const;
    void setInfo(const std::string &value);

    ~TerminalLogger();
    friend std::ostream& operator<<(std::ostream& os, ILogger& tl);

private:
    Time *localTime;
```

```
    std::string info;
};

class FileLogger : public ILogger
{
public:
    FileLogger(std::string fileName);

    void printTime();
    void printLog(std::string info);
    void print();
    bool isActive();

    std::string getInfo() const;
    void setInfo(const std::string &value);
    Time *getLocalTime() const;

    friend std::ostream& operator<<(std::ostream& os, ILogger& tl);
    ~FileLogger();

private:
    Time *localTime;
    std::string info;
    std::ofstream fout;
};

class ProxyLogger : public ILogger
{
public:
    ProxyLogger(LoggerType type);
    void print();
    void printTime();
    void printLog(std::string output);
    bool isActive();

    std::string getInfo() const;
    void setInfo(const std::string &value);
    Time *getLocalTime() const;

    ~ProxyLogger();
private:
    ILogger *logger;
};

#endif // LOGGER_H
```

## ПРИЛОЖЕНИЕ Г

## ИСХОДНЫЙ КОД ПРОГРАММЫ. LOGGER.CPP

```
#include "logger.h"

std::ostream& operator<<(std::ostream& os, Time& tm)
{
    tm.update();
    os  <<  tm.lastTime->tm_hour  <<  ":"  <<  tm.lastTime->tm_min  <<  ":"  <<
tm.lastTime->tm_sec;
    return os;
}

std::ostream &operator<<(std::ostream &os, ILogger &il)
{
    os << *(il.getLocalTime()) << " >> " << il.getInfo();
    return os;
}

//TERMINAL LOGGER

TerminalLogger::TerminalLogger()
{
    localTime = new Time;
}

void TerminalLogger::printTime()
{
    std::cout << *localTime << std::endl;
}

void TerminalLogger::printLog(std::string info)
{
    this->info = info;
    print();
}

void TerminalLogger::print()
{
    std::cout << *this << std::endl;
}

TerminalLogger::~TerminalLogger()
{
    std::cout << *localTime << " << Logger stopped\n";
    delete localTime;
}
```

```
Time *TerminalLogger::getLocalTime() const
{
    return localTime;
}

std::string TerminalLogger::getInfo() const
{
    return info;
}

void TerminalLogger::setInfo(const std::string &value)
{
    info = value;
}

//FILE LOGGER

FileLogger::FileLogger(std::string fileName)
{
    fout.open(fileName);
    if (!fout.is_open())
        throw std::runtime_error("File open failure");
    localTime = new Time;
}

void FileLogger::printTime()
{
    fout << *localTime;
}

void FileLogger::printLog(std::string info)
{
    this->info = info;
    print();
}

void FileLogger::print()
{
    fout << *this << "\n";
}

FileLogger::~FileLogger()
{
    fout << *localTime << " << Logger stopped\n";
    fout.close();
    delete localTime;
```

```
}


std::string FileLogger::getInfo() const
{
    return info;
}

void FileLogger::setInfo(const std::string &value)
{
    info = value;
}

Time *FileLogger::getLocalTime() const
{
    return localTime;
}

//PROXY LOGGER

ProxyLogger::ProxyLogger(LoggerType type)
{
    if (type == NO_LOGGER)
        logger = nullptr;
    else if (type == TO_TERMINAL)
        logger = new TerminalLogger;
    else if (type == TO_FILE)
        logger = new FileLogger(LOGFILE);
}

void ProxyLogger::print()
{
    if (logger)
        logger->print();
}

void ProxyLogger::printTime()
{
    if (logger)
        logger->printTime();
}

void ProxyLogger::printLog(std::string output)
{
    if (logger)
        logger->printLog(output);
}
```

```
std::string ProxyLogger::getInfo() const
{
    if (logger)
        return logger->getInfo();
    return "No logger";
}

void ProxyLogger::setInfo(const std::string &value)
{
    if (logger)
        logger->setInfo(value);
}

Time *ProxyLogger::getLocalTime() const
{
    if (logger)
        return logger->getLocalTime();
    return nullptr;
}

ProxyLogger::~ProxyLogger()
{
    if (logger)
        delete logger;
}
```

## ПРИЛОЖЕНИЕ Д

## ИСХОДНЫЙ КОД ПРОГРАММЫ. LOGADAPTER.H

```
#ifndef LOGADAPTER_H
#define LOGADAPTER_H

#include "logger.h"

class LogAdapter : public ILogAdapter
{
public:
    LogAdapter(ILogger *logger);
    void setLogger(ILogger *logger);
    void pushLog(LogEvent event, std::vector<int> param = {});
    LogType getType() const;
    void setType(const LogType &value);
    ~LogAdapter();
private:
    std::string makeUserActionLog(LogEvent event, std::vector<int> param = {});
    std::string makeGameCreateLog(std::vector<int> param = {});
    std::string makeBaseAddLog(std::vector<int> param = {});
    std::string makeNeutralAddLog(std::vector<int> param = {});
    std::string makeUnitAddLog(std::vector<int> param = {});
    std::string makeBaseUpdateLog(std::vector<int> param = {});
    std::string makeAttackLog(std::vector<int> param = {});
    std::string makeCounterUpdate(std::vector<int> param = {});
    std::string makeMoveLog(std::vector<int> param = {});
    ILogger *logger;
    LogType type;
};

#endif // LOGADAPTER_H
```

## ПРИЛОЖЕНИЕ Е

## ИСХОДНЫЙ КОД ПРОГРАММЫ. LOGADAPTER.CPP

```
#include "logadapter.h"

LogAdapter::LogAdapter(ILogger *logger)
    : logger(logger) {
    type = SHORT;
}

void LogAdapter::setLogger(ILogger *logger)
{
    if (this->logger != nullptr)
        delete this->logger;
    this->logger = logger;
}

void LogAdapter::pushLog(LogEvent event, std::vector<int> param)
{
    std::string output;
    if (event == GAME_CREATE)
        logger->printLog(makeGameCreateLog(param));
    else if (event == BASE_ADD)
        logger->printLog(makeBaseAddLog(param));
    else if (event == NEUTRAL_ADD)
        logger->printLog(makeNeutralAddLog(param));
    else if (event == UNIT_ADD)
        logger->printLog(makeUnitAddLog(param));
    else if (event == BASE_UPDATE)
        logger->printLog(makeBaseUpdateLog(param));
    else if (event == UNIT_ATTACK)
        logger->printLog(makeAttackLog(param));
    else if (event == UNIT_MOVE)
        logger->printLog(makeMoveLog(param));
    else if (event >= USER_GAME_INFO && event <= USER_ATTACK)
        logger->printLog(makeUserActionLog(event, param));
}

std::string LogAdapter::makeBaseAddLog(std::vector<int> param)
{
    if (param.size() < 4)
        throw std::out_of_range("Error! Wrong parameters for base add log");
    std::string output;
    output += "Created: BASE at coords: X=" + std::to_string(param[1]) + " Y=" +
std::to_string(param[2]);
    output += " | game bases number: " + std::to_string(param[0]);
    output += "\n\t\t | Update: field item counter: " + std::to_string(param[3]);
```

```
    return output;
}

std::string LogAdapter::makeNeutralAddLog(std::vector<int> param)
{
    if (param.size() < 4)
        throw std::out_of_range("Error! Wrong parameters for neutral add log");
    std::string output;
    if (type == SHORT)
    {
        std::vector<std::string> shortNames = {"CHEST", "KEYG", "ANTIV", "DATA"};
        output         +=         "Created:         neutral         "         +
shortNames[static_cast<unsigned>(param[0]-CHEST)];
        param = {param[3]};
        output += makeCounterUpdate(param);
    }
    else
    {
        std::vector<std::string> names = {"CHEST", "KEYGEN", "ANTIVIRUS", "DATA"};
        output  +=  "Created:  neutral  "  +  names[static_cast<unsigned>(param[0]-
CHEST)];
        output  +=  "  |  coords:  X="  +  std::to_string(param[1])  +  "  Y="  +
std::to_string(param[2]);
        param = {param[3]};
        output += "\n\t\t | " + makeCounterUpdate(param);
    }
    return output;
}

std::string LogAdapter::makeUnitAddLog(std::vector<int> param)
{
    if (param.size() < 4)
        throw std::out_of_range("Error! Wrong parameters for unit add log");
    std::string output;
    if (type == SHORT)
    {
         std::vector<std::string>  shortNames  =  {"ATT_SC",  "ATT_BS",  "SUP_SC",
"SUP_BS", "SIM_SC", "SIM_BS"};
         output += "Created: unit " + shortNames[static_cast<unsigned>(param[0])];
         output += " | base: N=" + std::to_string(param[1]);
    }
    else
    {
        std::vector<std::string> names = {"ATTACK SCIENTIST", "ATTACK BERSERK",
"SUPPORT SCIENTIST",
                                          "SUPPORT BERSERK", "SIMPLE SCIENTIST",
"SIMPLE BERSERK"};
```

```
        output += "Created: unit " + names[static_cast<unsigned>(param[0])];
        output += " | coords: X=" + std::to_string(param[2]) + " Y=" +
std::to_string(param[3]);
        output += " | base: number=" + std::to_string(param[1]);
    }
    return output;
}

std::string LogAdapter::makeBaseUpdateLog(std::vector<int> param)
{
    if (param.size() < 5)
        throw std::out_of_range("Error! Wrong parameters for base update log");
    std::string output;
    if (type == SHORT)
    {
        output = "Update: BASE | coords: X=" + std::to_string(param[0]) + " Y="
+ std::to_string(param[1]);
        output += " | uc: " + std::to_string(param[3]) + " | ul: " +
std::to_string(param[4]);
    }
    else
    {
        std::vector<std::string> shortNames = {"ATT_SC", "ATT_BS", "SUP_SC",
"SUP_BS", "SIM_SC", "SIM_BS"};
        output = "Update: BASE | coords: X=" + std::to_string(param[0]) + " Y="
+ std::to_string(param[1]);
        output += " | unit counter: " + std::to_string(param[3]) + " | unit limit:
" + std::to_string(param[4]);
        output += "\n\t\t | units: ";
        for (unsigned i=5; i<param.size(); i++)
        {
            output += shortNames[static_cast<unsigned>(param[i])] + " ";
        }
    }
    return output;
}

std::string LogAdapter::makeAttackLog(std::vector<int> param)
{
    if (param.size() < 6)
        throw std::out_of_range("Error! Wrong parameters for attack log");
    std::vector<std::string> shortNames = {"ATT_SC", "ATT_BS", "SUP_SC", "SUP_BS",
"SIM_SC", "SIM_BS"};
    std::string output;
    output += "Attacked: FROM unit " + shortNames[static_cast<unsigned>(param[5])]
+ " | coords: X=" + std::to_string(param[0]) + " Y=" + std::to_string(param[1]);
```

```
    output += " | TO coords: X=" + std::to_string(param[2] + param[0]) + " Y=" +
std::to_string(param[3] + param[1]);
    if (!param[4])
    {
        output += " | KILLED";
        param = {param[6]};
        output += "\n\t\t | " + makeCounterUpdate(param);
    }
    return output;
}

std::string LogAdapter::makeCounterUpdate(std::vector<int> param)
{
    std::string output;
    output += "Update: field item counter: " + std::to_string(param[0]);
    return output;
}

std::string LogAdapter::makeMoveLog(std::vector<int> param)
{
    if (param.size() < 5)
        throw std::out_of_range("Error! Wrong parameters for create game log");
    std::vector<std::string> shortNames = {"ATT_SC", "ATT_BS", "SUP_SC", "SUP_BS",
"SIM_SC", "SIM_BS"};
    std::string output;
    output += "Moved: unit " + shortNames[static_cast<unsigned>(param[4])];
    if (type == FULL)
    {
        output += " | FROM: X=" + std::to_string(param[0]) + " Y= " +
std::to_string(param[1]);
        output += " | BY: X=" + std::to_string(param[2]) + " Y= " +
std::to_string(param[3]);
    }
    return output;
}

std::string LogAdapter::makeGameCreateLog(std::vector<int> param)
{
    if (param.size() < 4)
        throw std::out_of_range("Error! Wrong parameters for create game log");
    std::string output;
    if (type == SHORT)
    {
        output += "Created: GAME with FIELD | size: W=" + std::to_string(param[0])
+ " H=" + std::to_string(param[1]);
        output += " | il: " + std::to_string(param[3]);
    }
```

```
    else
    {
        output += "Created: GAME with FIELD | size: width=" +
std::to_string(param[0]) + " height=" + std::to_string(param[1]);
        output += " | item limit: " + std::to_string(param[3]);
    }
    return output;
}

std::string LogAdapter::makeUserActionLog(LogEvent event, std::vector<int> param)
{
    std::string output;
    switch (event) {
    case USER_BASE_INFO:
        output += "USER | Request: base info";
        break;
    case USER_GAME_INFO:
        output += "USER | Request: game info";
        break;
    case USER_LAND_INFO:
        output += "USER | Request: land info";
        break;
    case USER_ITEMS_INFO:
        output += "USER | Request: items info";
        break;
    case USER_ITEM_INFO:
        output += "USER | Request: item by XY info";
        break;
    case USER_BASE_ADD:
        output += "USER | Request: create new base";
        break;
    case USER_UNIT_ADD:
        output += "USER | Request: create unit";
        break;
    case USER_NEUTRAL_ADD:
        output += "USER | Request: create neutral";
        break;
    case USER_MOVE:
        output += "USER | Request: move unit";
        break;
    case USER_ATTACK:
        output += "USER | Request: attack unit";
        break;
    }
    return output;
}
```

```
LogType LogAdapter::getType() const
{
    return type;
}

void LogAdapter::setType(const LogType &value)
{
    type = value;
}

LogAdapter::~LogAdapter()
{
    delete logger;
}
```

## ПРИЛОЖЕНИЕ Ж

## ИСХОДНЫЙ КОД ПРОГРАММЫ. UIFACADE.H

```
#ifndef UIFACADE_H
#define UIFACADE_H
#include "command.h"
#include "ui_mainwindow.h"

class FacadeMediator;

class UIFacade
{
public:
    UIFacade(Ui::MainWindow *ui, Game *game);
    bool getGameInfo();
    bool getBaseInfo(int number);
    bool getLandspaceInfo();
    bool getItemsInfo();
    bool getItemInfo(int x, int y);
    bool moveItem(int x, int y, int xDelta, int yDelta);
    bool attackUnit(int x, int y, int xDelta, int yDelta);
    bool addBase(int x, int y);
    bool addUnit(int x, int y, int base, int type);
    bool addNeutral(int x, int y, int type);
    void receiveStrAnswer(RequestType type, std::string answer);
    void setLogger(LoggerType type);
    LogAdapter *getLogger() const;
    ~UIFacade();

private:
    Ui::MainWindow *ui;
    Game *game;
    FacadeMediator *gameConnect;
    LogAdapter *logger;
};

class FacadeMediator : public IFacadeMediator {
public:
    FacadeMediator(UIFacade *facade, Command *command)
        : facade(facade), command(command) {}
    void sendString(RequestType type, std::string answer) {
        facade->receiveStrAnswer(type, answer);
    }
    UIFacade *getFacade() const;

private:
    UIFacade *facade;
```

```
    Command *command;
};


#endif // UIFACADE_H
```

## ПРИЛОЖЕНИЕ И

## ИСХОДНЫЙ КОД ПРОГРАММЫ. UIFACADE.CPP

```
#include "uifacade.h"

UIFacade::UIFacade(Ui::MainWindow *ui, Game *game)
    : ui(ui), game(game) {
    logger = new LogAdapter(new ProxyLogger(TO_FILE));
    logger->setType(FULL);
    GameCommand gCom(this, game, GAME_INFO, std::vector<int>(), logger);
    logger->pushLog(GAME_CREATE, gCom.exec());
    // RANDOM LAND
    unsigned width = game->getField()->getWidth();
    unsigned height = game->getField()->getHeight();
    for (unsigned i=0; i<width; i++)
    {
        for (unsigned j=0; j<height; j++)
        {
            if ((i+j)%3 != 0)
                game->getField()->addLandscape(i,             j,             new
ProxyLandscape(static_cast<LandType>((i*2+j*3)%5)));
        }
    }
}

bool UIFacade::getGameInfo()
{
    logger->pushLog(USER_GAME_INFO);
    GameCommand gCom(this, game, GAME_INFO, std::vector<int>(), logger);
    std::vector<int> answer = gCom.exec();
    std::string output;
    output += "Field:\nWidth: " + std::to_string(answer[0]) + "\nHeight: " +
std::to_string(answer[1]);
    output += "\nItem counter: " + std::to_string(answer[2]) + "\nItem limit: "
+ std::to_string(answer[3]);
    output += "\nBases number: " + std::to_string(answer[4]);
    ui->textOutput->append(QString::fromStdString(output));
    return true;
}

bool UIFacade::getBaseInfo(int number)
{
    logger->pushLog(USER_BASE_INFO);
    std::vector<std::string> typesName = {"ATT_SC", "ATT_BS", "SUP_SC", "SUP_BS",
"SIM_SC", "SIM_BS"};
    std::vector<int> request = {number};
```

```
    GameCommand gCom(this, game, BASE_INFO, request, logger);
    std::vector<int> answer = gCom.exec();
    std::string output = "BASE:\nCoords: " + std::to_string(answer[0]) + " " +
std::to_string(answer[1]) + "\nStability: " + std::to_string(answer[2]);
    output += "\nUnit Counter: " + std::to_string(answer[3]) + "\nUnit Limit: "
+ std::to_string(answer[4]);
    output += "\nUnits:\n";
    for (unsigned i=5; i<answer.size(); i++)
    {
        output += typesName[static_cast<unsigned>(answer[i])] + " ";
    }
    ui->textOutput->append(QString::fromStdString(output));
    return true;
}

bool UIFacade::getLandspaceInfo()
{
    logger->pushLog(USER_LAND_INFO);
    ui->textField->clear();
    GameCommand gCom(this, game, LAND_INFO, std::vector<int>(), logger);
    std::vector<int> answer = gCom.exec();
    std::vector<std::string> landsName = {"OPEN  |", "OPEN_B|", "VPN   |", "VPN_B
|", "FAST  |", "FAST_B|"};
    unsigned width = static_cast<unsigned>(answer[0]);
    unsigned height = static_cast<unsigned>(answer[1]);
    std::string output;
    for (unsigned i=0; i<height; i++)
    {
        for (unsigned j=0; j<width; j++)
        {
            output += landsName[static_cast<unsigned>(answer[2+(j*height)+i])];
        }
        output += "\n";
    }
    ui->textField->append(QString::fromStdString(output));
    return true;
}

bool UIFacade::getItemsInfo()
{
    GameCommand gCom(this, game, ITEMS_INFO, std::vector<int>(), logger);
    std::vector<std::string>  itemsName  =  {"ATT_SC|",  "ATT_BS|",  "SUP_SC|",
"SUP_BS|", "SIM_SC|", "SIM_BS|", "BASE   |", "CHEST |", "KEYGEN|", "ANTIVR|",
"DATA  |", "      |"};
    std::vector<int> answer = gCom.exec();
    unsigned width = static_cast<unsigned>(answer[0]);
    unsigned height = static_cast<unsigned>(answer[1]);
```

```
    std::string output;
    for (unsigned i=0; i<height; i++)
    {
        for (unsigned j=0; j<width; j++)
        {
            output += itemsName[static_cast<unsigned>(answer[2+(j*height)+i])];
        }
        output += "\n";
    }
    ui->textField->clear();
    ui->textField->append(QString::fromStdString(output));
    return true;
}

bool UIFacade::getItemInfo(int x, int y)
{
    logger->pushLog(USER_ITEM_INFO);
    std::vector<int> request = {x, y};
    std::vector<std::string> itemsName = {"ATT_SC", "ATT_BS", "SUP_SC", "SUP_BS",
"SIM_SC", "SIM_BS", "BASE", "CHEST", "KEYGEN", "ANTIVR", "DATA", "NOTHING"};
    GameCommand gCom(this, game, GETXY, request, logger);
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while getting item info: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
        return false;
    }
    output += "At coords: (" + std::to_string(x) + ", " + std::to_string(y) + ")
is set:\n";
    output += itemsName[static_cast<unsigned>(answer[0])] + "\n";
    if (answer[0] == BASE)
    {
        ui->textOutput->append(QString::fromStdString(output));
        getBaseInfo(answer[1]+1);
    }
    else if (answer[0] < BASE)
    {
        output += "Characterictics:\n";
        output += "Power: " + std::to_string(answer[1] + answer[4]) + "\n";
        output += "Move: " + std::to_string(answer[2] + answer[4]) + "\n";
        output += "Spread: " + std::to_string(answer[3] + answer[4]) + "\n";
        output += "Attack skills:\n";
        output += "Power: " + std::to_string(answer[5] + answer[8]) + "\n";
```

```
        output += "Move: " + std::to_string(answer[6] + answer[8]) + "\n";
        output += "Spread: " + std::to_string(answer[7] + answer[8]) + "\n";
        output += "Security skills:\n";
        output += "Power: " + std::to_string(answer[9] + answer[12]) + "\n";
        output += "Move: " + std::to_string(answer[10] + answer[12]) + "\n";
        output += "Spread: " + std::to_string(answer[11] + answer[12]) + "\n";
        output += "Base number: " + std::to_string(answer[13]) + "\n";
        ui->textOutput->append(QString::fromStdString(output));
    }
    else
        ui->textOutput->append(QString::fromStdString(output));
    return true;
}

bool UIFacade::moveItem(int x, int y, int xDelta, int yDelta)
{
    logger->pushLog(USER_MOVE);
    std::vector<int> request = {x, y, xDelta, yDelta};
    GameCommand gCom(this, game, MOVE, request, logger);
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while moving item: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
        return false;
    }
    if (answer.front() == 0)
    {
        output += "Item is not unit (is not movable)";
        return false;
    }
    else
        output += "Item was moved";
    ui->textOutput->append(QString::fromStdString(output));
    ui->setActX->setValue(x + xDelta);
    ui->setActY->setValue(y + yDelta);
    return true;
}

bool UIFacade::attackUnit(int x, int y, int xDelta, int yDelta)
{
    logger->pushLog(USER_ATTACK);
    std::vector<int> request = {x, y, xDelta, yDelta};
    GameCommand gCom(this, game, ATTACK, request, logger);
```

```
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while attacking: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
        return false;
    }
    return true;
}

bool UIFacade::addBase(int x, int y)
{
    logger->pushLog(USER_BASE_ADD);
    std::vector<int> request = {BASE, x, y};
    GameCommand gCom(this, game, ADD, request, logger);
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while adding base: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
        return false;
    }
    output += "Added base. Now bases: " + std::to_string(answer[0]);
    ui->textOutput->append(QString::fromStdString(output));
    ui->comboBases->addItem(QString::number(answer[0]));
    return true;
}

bool UIFacade::addUnit(int x, int y, int base, int type)
{
    logger->pushLog(USER_UNIT_ADD);
    std::vector<int> request = {type, base, x, y};
    GameCommand gCom(this, game, ADD, request, logger);
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while adding unit: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
```

```
        return false;
    }
    return true;
}

bool UIFacade::addNeutral(int x, int y, int type)
{
    logger->pushLog(USER_NEUTRAL_ADD);
    std::vector<int> request = {type, x, y};
    GameCommand gCom(this, game, ADD, request, logger);
    std::string output;
    std::vector<int> answer;
    try {
        answer = gCom.exec();
    } catch (std::logic_error a) {
        output += "Error while adding neutral: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
        return false;
    }
    return true;
}

void UIFacade::receiveStrAnswer(RequestType type, std::string answer)
{
    ui->debugOutput->append(QString::fromStdString(answer));
}

void UIFacade::setLogger(LoggerType type)
{
    logger->setLogger(new ProxyLogger(type));
}

UIFacade::~UIFacade()
{
    delete logger;
}

LogAdapter *UIFacade::getLogger() const
{
    return logger;
}


UIFacade *FacadeMediator::getFacade() const
{
    return facade;
```

```
}
```