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

# ОТЧЕТ

**по лабораторной работе №3**
**по дисциплине «Объектно-ориентированное программирование»**
**Тема: Логическое разделение классов**

Студент гр. 8381 _____ Облизанов А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

**Цель работы.**

Разработать и реализовать наборы классов для взаимодействия пользователя с юнитами и базой.

**Задание.**

Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Дополнительные требования:

- Реализован паттерн “Фасад”, через который пользователь управляет программой
- Объекты между собой взаимодействуют через паттерн “Посредника”
- Для передачи команд используется паттерн “Команда”
- Для приема команд от пользователя используется паттерн “Цепочка обязанностей”

**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.

**Основные классы, фасад, команды**

Основные классы, добавленные в программу, и их назначение представлены в табл. 1.

Таблица 1 – Основные добавленные классы

| **Название** | **Назначение** |
|---|---|
| Game<br>(класс игры)<br>(не относится к требованиям) | В классе хранится поле, массив баз, а также информация о юнитах.<br>Класс обладает функционалом: создание баз, нейтральных объектов и помещение их на поле.<br>С помощью посредника GameMediator между игрой и базами ведется учет юнитов: информация о них своевременно добавляется или удаляется. |
| GameCommand,<br>(команда игры),<br>FieldCommand<br>(команда поля),<br>BaseCommand<br>(команда базы) | Реализованы по принципу паттерна «**команда**».<br>Команды принимают на вход тип запроса и массив параметров (int).<br>Команда поля дополнительно принимает на вход указатель на объект, если нужно определить его координаты.<br>Команды возвращают массив выходных значений (int).<br>Команда поля отдельным методом может возвращать указатель на объект, если его нужно найти по координатам. |
| UIFacade<br>(фасад) | Реализован по принципу паттерна «**фасад**».<br>Фасад имеет множество методов, которые умеют работать с командами: передают параметры и обрабатывают выходные значения.<br>Фасад работает непосредственно с виджетами UI, выводя туда результаты команд.<br>Фасад не пользуется методами классов логики игры (напр. класса Game, Field, Base, Unit) – он создает и выполняет команду GameCommand, тем самым запуская цепочку обязанностей, в которой уже реализуется требуемый функционал. |
| FacadeMediator<br>(посредник для команд) | Все запускаемые команды связываются с фасадом посредником.<br>С помощью посредника команда имеет возможность передать фасаду сообщение в виде строки (и типа команды), которое он обрабатывает и выводит на экран. Это сделано для случаев, когда выходные данные неудобно сохранять в массив элементов типа int. |

**UML диаграмма**

Помимо вышеперечисленных классов, в программу было добавлено множество других. UML диаграмма для реализованных классов представлена на рис. 1 (классы MainWindow, Base, Field, IUnit, Unit уже существовали).

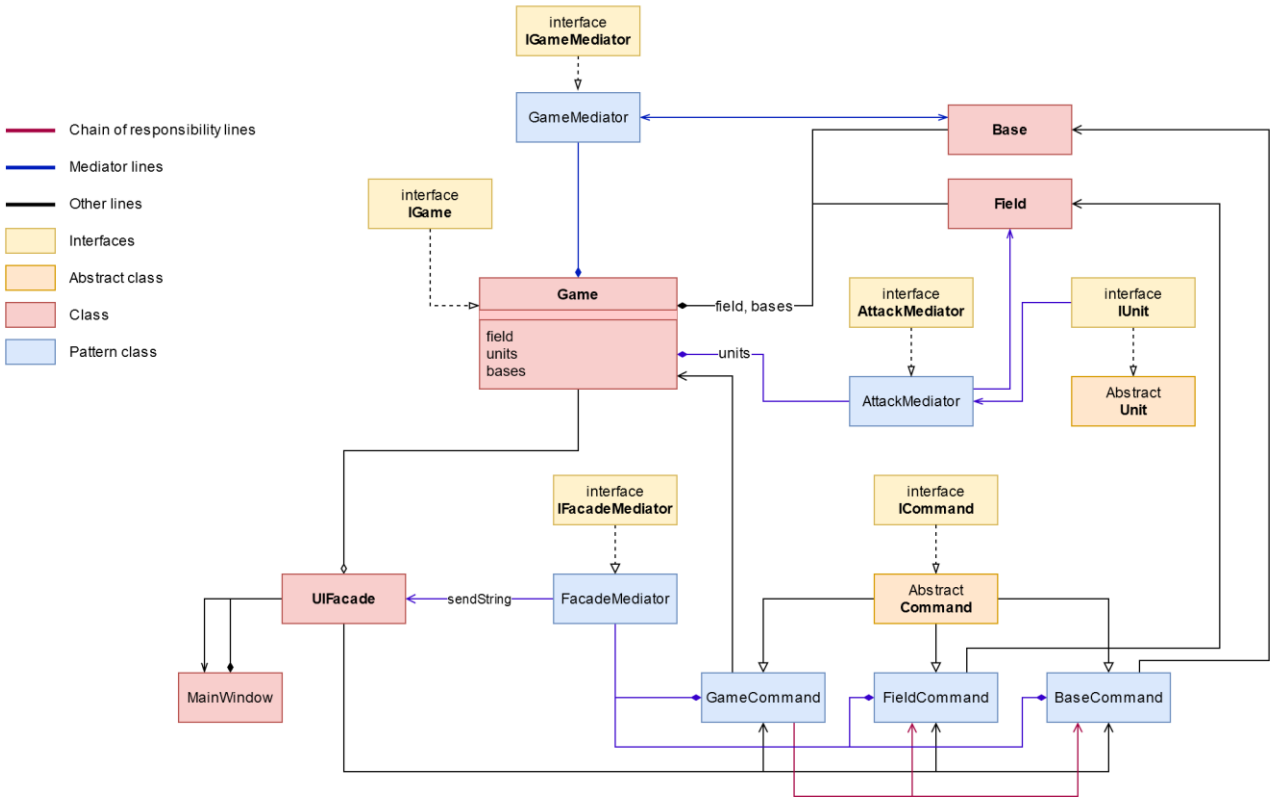


Рисунок 1 – UML диаграмма

**Команды**

Указанные в табл. 1 классы, реализованные по принципу паттерна «команда», имеют следующие свойства

- Каждая команда может пользоваться методами определенных классов, а именно
  - Команда GameCommand пользуется методами класса Game
  - Команда FieldCommand пользуется методами класса Field
  - Команда BaseCommand пользуется методами класса Base

- Входные параметры, указатель на используемый класс, указатель на посредника для команд, тип команды принимаются конструктором команды
- Команда выполняется методом exec(), который возвращает выходные параметры (в случае команды поля есть дополнительный метод findItem для дополнительного функционала)

**Цепочки обязанностей, функционал**

В программе реализован функционал управления базой, юнитами, а также вывод различной информации. Цепочки обязанностей формируются из классов команд и запускаются из фасада:

- Facade
    - GameCommand
        - FieldCommand
        - BaseCommand

Функциональные возможности программы, их описание и цепочка обязанностей представлены в табл. 2.

Таблица 2 – Функционал программы

| **Функция** | **Подробности** | **Цепочка** |
|---|---|---|
| Вывод информации об игре | Вывод размеров поля, числа объектов, числа баз | Facade > GameCommand |
| Вывод ландшафта | Выводится сеткой | Facade > GameCommand > FieldCommand |
| Вывод игрового поля | Выводится сеткой | Facade > GameCommand > FieldCommand |

Окончание таблицы 2

| | | |
|---|---|---|
| Вывод информации об i-й базе | Вывод характеристик, местоположения на поле, состава базы (юнитов)<br>Примечание к цепочке: GameCommand сначала получает от FieldCommand координаты базы, а затем уже вызывается BaseCommand | Facade > GameCommand > FieldCommand > GameCommand > BaseCommand |
| Вывод информации об объекте на поле | Информация берется по координатам (для юнитов - все характеристики, для базы - аналогично пункту выше, для нейтрального объекта - только название)<br>Примечание к цепочке: GameCommand получает от FieldCommand указатель на объект, а далее в зависимости от его типа формирует ответ | Facade > GameCommand > FieldCommand |
| Добавление базы | Есть возможность добавления нескольких баз, и весь функционал учитывает эту возможность | Facade > GameCommand |
| Добавление юнита | Добавление юнита любого типа (есть выбор в UI) по координатам, для юнита выбирается база, которая его создает | Facade > GameCommand > BaseCommand |
| Добавление нейтрального объекта | Добавление нейтрального объекта любого типа (есть выбор в UI) по координатам | Facade > GameCommand |
| Перемещение юнита | Юнит выбирается по координатам, его перемещение определяется смещением по координатам<br>Примечание к цепочке: GameCommand получает от FieldCommand указатель на объект, а затем ищет его в учете юнитов | Facade > GameCommand > FieldCommand |
| Атака юнита | Юнит выбирается по координатам, его атака определяется смещением по координатам<br>Примечание к цепочке: аналогично перемещению юнита | Facade > GameCommand > FieldCommand |

**Посредник для взаимодействия юнитов**

Атака юнитов реализована посредником AttackMediator, который хранит всех юнитов, а также указатель на поле. Доступ к посреднику тоже есть у всех юнитов. Посредник хранится в классе Game. Этапы взаимодействия юнитов представлены в табл. 3.

Таблица 3 – Основные этапы процесса атаки

| |
|---|
| 1. Юнит методом attack() передает посреднику координаты смещения для атаки и указатель на себя |
| 2. Посредник с помощью итератора поля находит объект и проверяет, что объект является юнитом, а также с помощью поля находит объект по координатам атаки |
| 3. Идет проверка, что атакуемый - тоже юнит, и что он принадлежит другой базе |
| 4. Сама атака осуществляется методом юнита receiveAttack(), который возвращает false, если юнит умер. В этом случае посредник обращается к полю для удаления юнита. |

**Демонстрационные примеры**

В программе реализован базовый GUI для взаимодействия пользователя с игрой. В связи с этим, число демонстрационных примеров может быть бесконечным и зависит лишь от последовательности действий пользователя. Момент игры, когда на поле существует 2 базы, несколько юнитов, нейтральных объектов, и выведена информация о первой базе, представлен на рис. 2.

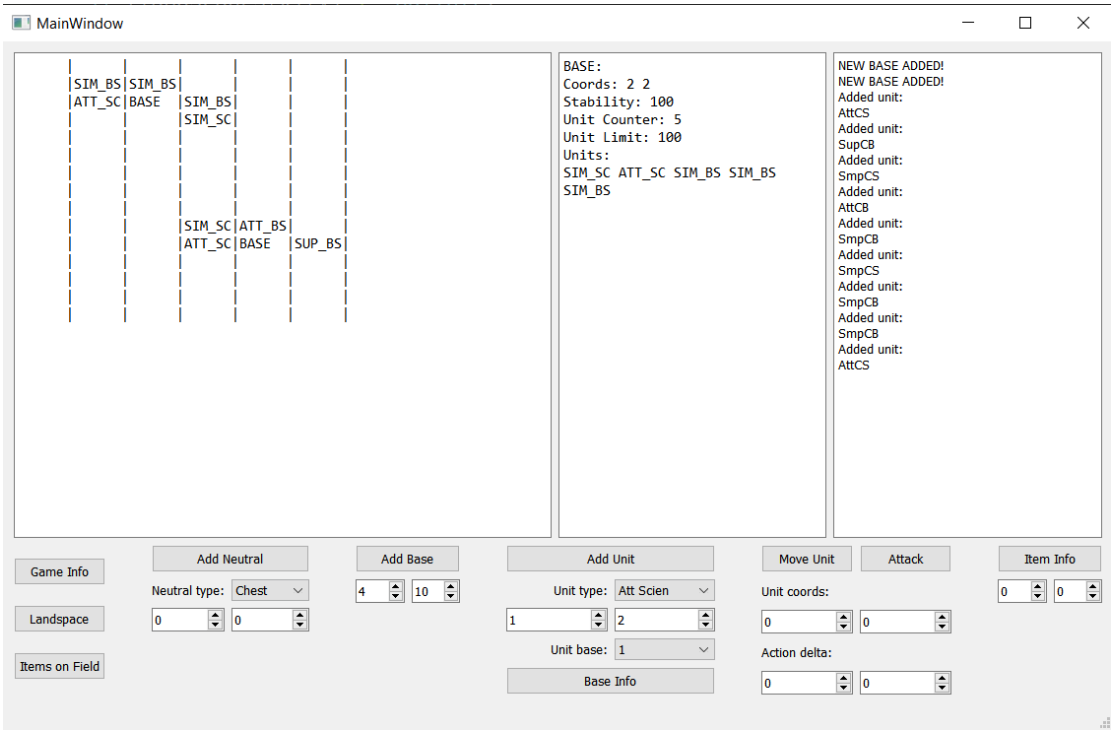


Рисунок 2 – Вид программы во время игры

Тот же момент игры, но с выведенной информацией об одном из юнитов из второй базы, представлен на рис. 3.

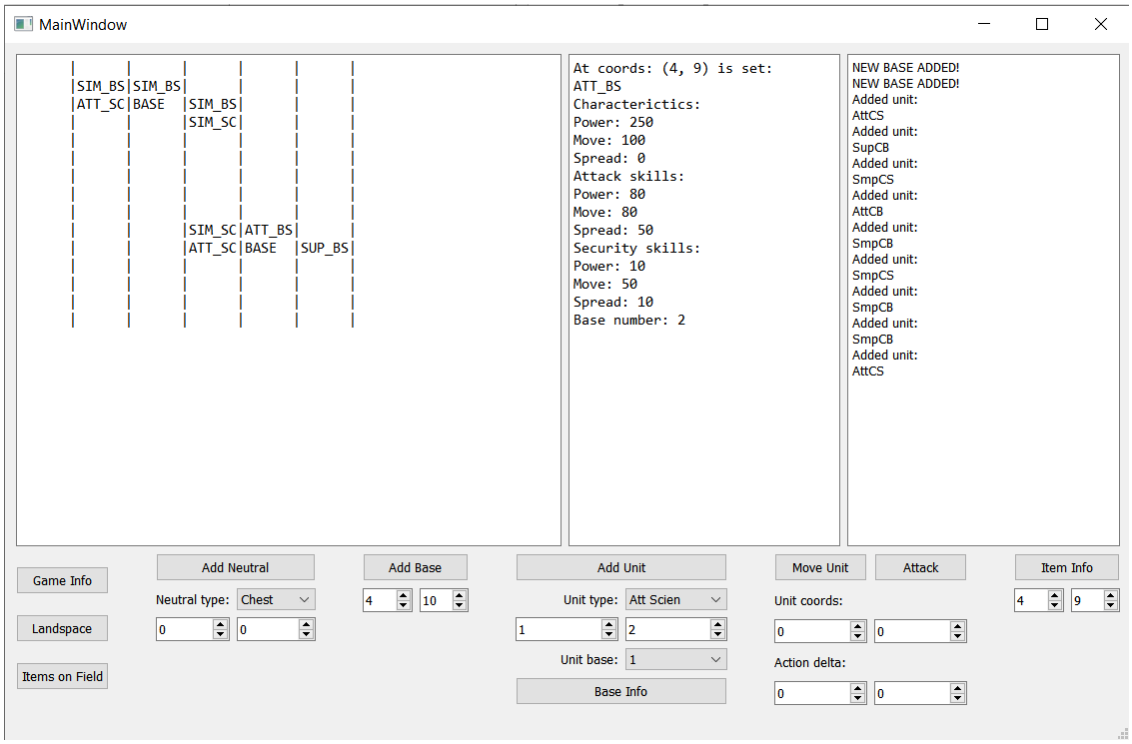


Рисунок 2 – Информация о юните во время игры

**Выводы.**

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для функционала программы и взаимодействия пользователя с программой. Был использован объектно-ориентированный стиль программирования, были изучены и применены его основные положения, а также реализованы некоторые паттерны проектирования.

## 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>



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
};

#endif // LIBRARIES_H
```

## ПРИЛОЖЕНИЕ В

## ИСХОДНЫЙ КОД ПРОГРАММЫ. GAME.H

```
#ifndef GAME_H
#define GAME_H
#include "attackmediator.h"

class Game : public IGame
{
public:
    Game(unsigned x, unsigned y, unsigned limit, LandType basicLand);
    void createBase(std::string name, unsigned limit, unsigned x, unsigned y);
    Field *getField() const;
    std::vector<Base *> getBases() const;
    std::vector<IUnit *> getUnits() const;
    Base *getBaseByNumber(unsigned number);
    void addUnit(IUnit *unit, Base *base);
    void deleteUnit(IUnit *unit, Base *base);
    void createNeutral(unsigned x, unsigned y, NeutralType type);
private:
    Field *field;
    AttackMediator *units;
    std::vector<Base *> bases;
    IGameMediator *baseMediator;
};

class GameMediator : public IGameMediator
{
public:
    GameMediator(Game *game, Base *base)
        :game(game), base(base) {
        base->setGameMediator(this);
    }
    void Notify(IUnit *unit, bool add) {
        if (add)
            game->addUnit(unit, base);
        else
            game->deleteUnit(unit, base);
    }
private:
    Game *game;
    Base *base;
};



#endif // GAME_H
```

## ПРИЛОЖЕНИЕ Г

## ИСХОДНЫЙ КОД ПРОГРАММЫ. GAME.CPP

```
#include "game.h"

Game::Game(unsigned x, unsigned y, unsigned limit, LandType basicLand)
{
    this->field = new Field(x, y, limit, basicLand);
    units = new AttackMediator(field);
}

void Game::createBase(std::string name, unsigned limit, unsigned x, unsigned y)
{
    Base *base = new Base(limit, static_cast<unsigned>(bases.size()), name,
field);
    try {
        field->addItem(x, y, base);
        baseMediator = new GameMediator(this, base);
        base->setGameMediator(baseMediator);
    } catch (std::logic_error a) {
        delete base;
        throw a;
    }
    bases.push_back(base);
}

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

std::vector<Base *> Game::getBases() const
{
    return bases;
}

Base *Game::getBaseByNumber(unsigned number)
{
    for (auto i : bases)
    {
        if (i->getNumber() == number)
            return i;
    }
    return nullptr;
}

void Game::addUnit(IUnit *unit, Base *base)
```

```
{
    bool checkBase = false;
    for (auto i : bases)
        if (i == base)
            checkBase = true;
    if (!checkBase)
        throw std::invalid_argument("Error! Invalid base tries to add unit to
game");
    units->addUnit(unit);
}

void Game::deleteUnit(IUnit *unit, Base *base)
{
    bool checkBase = false;
    for (auto i : bases)
        if (i == base)
            checkBase = true;
    if (!checkBase)
        throw std::invalid_argument("Error! Invalid base tries to add unit to
game");
    units->removeUnit(unit);
}

void Game::createNeutral(unsigned x, unsigned y, NeutralType type)
{
    srand(static_cast<unsigned>(time(nullptr)));
    INeutral *neutral;
    switch (type) {
    case CHEST:
        neutral = new Chest(rand() % 40);
        break;
    case KEYGEN:
        neutral = new Keygen(rand() % 40, rand() % 20);
        break;
    case ANTIVIRUS:
        neutral = new Antivirus(rand() % 40, rand() % 20);
        break;
    case DATA:
        neutral = new Data(rand() % 40);
    }
    try {
        field->addItem(x, y, neutral);
    } catch (std::logic_error a) {
        delete neutral;
        throw a;
    }
}
```

```
std::vector<IUnit *> Game::getUnits() const
{
    return units->getUnits();
}
```

## ПРИЛОЖЕНИЕ Д

## ИСХОДНЫЙ КОД ПРОГРАММЫ. COMMAND.H

```
#ifndef COMMAND_H
#define COMMAND_H

#include "game.h"

class Command : public ICommand
{
    virtual std::vector<int> exec() = 0;
private:
    RequestType type;
    std::vector<int> param;
    IFacadeMediator *facadeMediator;
};

class GameCommand : public Command
{
public:
    GameCommand(UIFacade *facade, Game *game, RequestType type, std::vector<int>
param);
    std::vector<int> exec();
private:
    Game *game;
    RequestType type;
    std::vector<int> param;
    IFacadeMediator *facadeMediator;
};

class BaseCommand : public Command
{
public:
    BaseCommand(IFacadeMediator   *facade,   Base   *base,   RequestType   type,
std::vector<int> param, FieldItem *toFind);
    std::vector<int> exec();
private:
    Base *base;
    RequestType type;
    std::vector<int> param;
    FieldItem *toFind;
    IFacadeMediator *facadeMediator;
};

class FieldCommand : public Command
{
public:
```

```
    FieldCommand(IFacadeMediator *facade, Field *field, RequestType type,
std::vector<int> param, FieldItem *toFind);
    std::vector<int> exec();
    FieldItem *findItem(int x, int y);
private:
    Field *field;
    RequestType type;
    std::vector<int> param;
    FieldItem *toFind;
    IFacadeMediator *facadeMediator;
};

#endif // COMMAND_H
```

## ПРИЛОЖЕНИЕ Е

## ИСХОДНЫЙ КОД ПРОГРАММЫ. COMMAND.CPP

```
#include "command.h"
#include "uifacade.h"

GameCommand::GameCommand(UIFacade *facade, Game *game, RequestType type,
std::vector<int> param)
    : game(game), type(type), param(param) {
    facadeMediator = new FacadeMediator(facade, this);
}

std::vector<int> GameCommand::exec()
{
    std::vector<int> answer;
    if (type == GAME_INFO)
    {
        answer.push_back(static_cast<int>(game->getField()->getWidth()));
        answer.push_back(static_cast<int>(game->getField()->getHeight()));
        answer.push_back(static_cast<int>(game->getField()->getItemCounter()));
        answer.push_back(static_cast<int>(game->getField()->getItemLimit()));
        answer.push_back(static_cast<int>(game->getBases().size()));
    } else if (type == BASE_INFO)
    {
        if (param.front() >= 0)
        {
            unsigned number = static_cast<unsigned>(param.front()) - 1;
            if (number < game->getBases().size())
            {
                std::vector<int> request = {param.front()};
                Base *base = game->getBaseByNumber(number);
                FieldCommand fCom(facadeMediator, game->getField(), FIND, param,
base);
                answer = fCom.exec();
                BaseCommand bCom(facadeMediator, base, type, param, nullptr);
                for (auto i : bCom.exec())
                    answer.push_back(i);
                return answer;
            }
        }
    } else if (type == LAND_INFO || type == ITEMS_INFO)
    {
        FieldCommand fCom(facadeMediator, game->getField(), type, param,
nullptr);
        return fCom.exec();
    } else if (type == GETXY)
    {
```

```
        FieldCommand fCom(facadeMediator, game->getField(), type, param,
nullptr);
        FieldItem *item = fCom.findItem(param[0], param[1]);
        if (item == nullptr)
        {
            answer.push_back(11);
            return answer;
        }
        answer.push_back(item->getType());
        if (item->getType() == BASE)
        {
            for (unsigned i=0; i<game->getBases().size(); i++)
            {
                if (game->getBases()[i] == item)
                {
                    answer.push_back(static_cast<int>(i));
                }
            }
        } else if (item->getType() < BASE)
        {
            for (auto i : game->getUnits())
            {
                if (i == item)
                {
                    answer.push_back(i->getCharacteristics().getPower());
                    answer.push_back(i->getCharacteristics().getMove());
                    answer.push_back(i->getCharacteristics().getSpread());
                    answer.push_back(i->getCharacteristics().getBonus());
                    answer.push_back(i->getAttackSkills().getPower());
                    answer.push_back(i->getAttackSkills().getMove());
                    answer.push_back(i->getAttackSkills().getSpread());
                    answer.push_back(i->getAttackSkills().getBonus());
                    answer.push_back(i->getSecuritySkills().getPower());
                    answer.push_back(i->getSecuritySkills().getMove());
                    answer.push_back(i->getSecuritySkills().getSpread());
                    answer.push_back(i->getSecuritySkills().getBonus());
                    answer.push_back(static_cast<int>(i->getBaseNumber() + 1));
                }
            }
        }
    } else if (type == ATTACK)
    {
        FieldCommand fCom(facadeMediator, game->getField(), type, param,
nullptr);
        FieldItem *item = fCom.findItem(param[0], param[1]);
        if (item == nullptr)
        {
```

```
            facadeMediator->sendString(type, "Error! Field cell is empty");
        }
        for (auto i : game->getUnits())
        {
            if (i == item)
            {
                i->attack(param[2], param[3]);
                return answer;
            }
        }
        facadeMediator->sendString(type, "Attacker is not unit");
    } else if (type == ADD)
    {
        if (param.front() == BASE)
        {
            game->createBase("player",   100,   static_cast<unsigned>(param[1]),
static_cast<unsigned>(param[2]));
            facadeMediator->sendString(type, "NEW BASE ADDED!");
            answer.push_back(static_cast<int>(game->getBases().size()));
        }
        else if (param.front() >= CHEST && param.front() <= DATA)
        {
            unsigned x = static_cast<unsigned>(param[1]);
            unsigned y = static_cast<unsigned>(param[2]);
            game->createNeutral(x, y, static_cast<NeutralType>(param[0]));
        }
        else
        {
            unsigned number = static_cast<unsigned>(param[1]) - 1;
            Base *base = game->getBaseByNumber(number);
            BaseCommand bCom(facadeMediator, base, type, param, nullptr);
            return bCom.exec();
        }
    } else if (type == MOVE)
    {
        FieldCommand   fCom(facadeMediator,   game->getField(),   type,   param,
nullptr);
        FieldItem *item = fCom.findItem(param[0], param[1]);
        for (auto i : game->getUnits())
        {
            if (i == item)
            {
                i->move(param[2], param[3]);
                answer.push_back(1);
                return answer;
            }
        }
```

```
        answer.push_back(0);
    }
    return answer;
}

BaseCommand::BaseCommand(IFacadeMediator *facade, Base *base, RequestType type,
std::vector<int> param, FieldItem* toFind)
    :      base(base),      type(type),      param(param),      toFind(toFind),
facadeMediator(facade) {}

std::vector<int> BaseCommand::exec()
{
    std::vector<int> answer;
    if (type == BASE_INFO)
    {
        answer.push_back(static_cast<int>(base->getStability()));
        answer.push_back(static_cast<int>(base->getUnitCounter()));
        answer.push_back(static_cast<int>(base->getUnitLimit()));
        for (auto i : base->getUnitList())
        {
            answer.push_back(i->getType());
        }
    } else if (type == ADD)
    {
        unsigned x = static_cast<unsigned>(param[2]);
        unsigned y = static_cast<unsigned>(param[3]);
        IUnit *unit = base->createUnit(x, y, static_cast<UnitType>(param[0]));
        facadeMediator->sendString(type, "Added unit:\n" + unit->shortName());
    }
    return answer;
}

FieldCommand::FieldCommand(IFacadeMediator  *facade,  Field  *field,  RequestType
type, std::vector<int> param, FieldItem *toFind)
    :      field(field),      type(type),      param(param),      toFind(toFind),
facadeMediator(facade) {}

std::vector<int> FieldCommand::exec()
{
    std::vector<int> answer;
    if (type == LAND_INFO)
    {
        answer.push_back(static_cast<int>(field->getWidth()));
        answer.push_back(static_cast<int>(field->getHeight()));
        for (unsigned i=0; i<field->getWidth(); i++)
        {
            for (unsigned j=0; j<field->getHeight(); j++)
```

```
            {
                answer.push_back(field->getLandType(i, j));
            }
        }
    } else if (type == ITEMS_INFO)
    {
        answer.push_back(static_cast<int>(field->getWidth()));
        answer.push_back(static_cast<int>(field->getHeight()));
        FieldItem *item;
        for (unsigned i=0; i<field->getWidth(); i++)
        {
            for (unsigned j=0; j<field->getHeight(); j++)
            {
                item = field->getItem(i, j);
                if (item != nullptr)
                    answer.push_back(item->getType());
                else
                    answer.push_back(11);
            }
        }
    }
    else if (type == FIND && toFind != nullptr)
    {
        for (unsigned i=0; i<field->getWidth(); i++)
        {
            for (unsigned j=0; j<field->getHeight(); j++)
            {
                if (field->getItem(i, j) == toFind)
                {
                    answer.push_back(static_cast<int>(i));
                    answer.push_back(static_cast<int>(j));
                }
            }
        }
    }
    return answer;
}

FieldItem *FieldCommand::findItem(int x, int y)
{
    unsigned xU = static_cast<unsigned>(x);
    unsigned yU = static_cast<unsigned>(y);
    return field->getItem(xU, yU);
}
```

## 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);

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

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) {
    // 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()
{
    GameCommand gCom(this, game, GAME_INFO, std::vector<int>());
    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)
{
    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);
    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()
{
    ui->textField->clear();
    GameCommand gCom(this, game, LAND_INFO, std::vector<int>());
    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>());
    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)
{
    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);
    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)
{
    std::vector<int> request = {x, y, xDelta, yDelta};
    GameCommand gCom(this, game, MOVE, request);
    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)
{
    std::vector<int> request = {x, y, xDelta, yDelta};
    GameCommand gCom(this, game, ATTACK, request);
    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)
{
    std::vector<int> request = {BASE, x, y};
    GameCommand gCom(this, game, ADD, request);
    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)
{
    std::vector<int> request = {type, base, x, y};
    GameCommand gCom(this, game, ADD, request);
    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)
{
    std::vector<int> request = {type, x, y};
    GameCommand gCom(this, game, ADD, request);
    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));
}


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