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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Объектно-ориентированное программирование»
Тема: Сериализация состояния программы

Студент гр. 8381 _____ Облизанов А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Реализация сохранения и загрузки состояния программы.

Задание.

Основные требования:

- Возможность записать состояние программы в файл
- Возможность считать состояние программы из файла
- Загрузка и сохранение должно выполняться в любой момент программы
- Взаимодействие с файлами должны быть по идиоме RAII
- Сохранение и загрузка реализованы при помощи паттерна “Снимок”
- Реализован контроль корректности файла с сохраненными данными

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.

Реализация сохранения и загрузки состояния программы

Систему сохранения и загрузки состояния игры можно разбить на следующие элементы:

- Класс фасада Facade имеет методы для *сохранения и загрузки* игры, которые вызываются из UI *в любой момент времени*
- Загрузка и сохранение реализованы по принципу паттерна «*Снимок*». Создателем является класс Game, снимком – класс GameMemento. Для передачи информации об игре в снимок используются структуры.

- Для сохранения и загрузки в файл *по принципу RAII* созданы классы MementoReader и MemenroWriter, которые также реализуют *контроль корректности файла с сохраненными данными*
- Файлы с сохранениями имеют нестандартное расширение «.cyb». В UI игры пользователь может выбирать файлы для загрузки и создавать новые для сохранения.

Далее рассмотрим каждый элемент системы подробнее.

Структуры для сохранения состояния

Для передачи различных параметров игры в снимок используются структуры (файл structures.h), основная из который – GameParameters, а другие (BaseParameters, UnitParameters, NeutralParameters) являются ее составляющими. Поля структуры GameParameters представлены в табл. 1.

Таблица 1 – Поля структуры GameParameters

Поле	**Назначение**
unsigned width;	Значение ширины поля
unsigned height;	Значение высоты поля
unsigned itemLimit;	Значение максимального числа объектов
unsigned itemCounter;	Значение текущего числа объектов
std::vector<LandType> landscape;	Массив ландшафта поля (все клетки)
size_t baseCount;	Значение количества баз
std::vector<BaseParameters> bases;	Массив информации о базах
std::vector<NeutralParameters> neutrals;	Массив информации о нейтральных объектах

Как видно, в структуре параметров игры хранятся массивы параметров баз и нейтральных объектов. Поля структуры NeutralParameters представлены в табл. 2.

Таблица 2 – Поля структуры NeutralParameters

Поле	**Назначение**
NeutralType type;	Тип объекта (перечисление NeutralType)
unsigned x;	Значение координаты X объекта на поле
unsigned y;	Значение координаты Y объекта на поле

Поля структуры BaseParameters представлены в табл. 3.

Таблица 3 – Поля структуры BaseParameters

Поле	**Назначение**
unsigned number;	Значение номера базы
unsigned stability;	Значение стабильности базы (здоровье)
unsigned limit;	Максимальное число юнитов
unsigned x;	Значение координаты X базы на поле
unsigned y;	Значение координаты Y базы на поле
std::vector<UnitParameters> units;	Массив информации о юнитах

Поля структуры UnitsParameters представлены в табл. 4.

Таблица 4 – Поля структуры UnitsParameters

Поле	**Назначение**
UnitType type;	Тип объекта (перечисление UnitType)
unsigned x; unsigned y;	Координаты X, Y юнита на поле
int move; int power; int spread;	Значения характеристик юнита
int charactBonus; int attackBonus; int securityBonus;	Значения бонусов к характеристикам, атаке и защите юнита

Для удобства, у структур GameParameters, NeutralParameters созданы конструкторы, в которых сразу определяются значения некоторых полей.

Классы взаимодействия с файлами

Для взаимодействия с файлами по *идиоме RAII* были созданы классы MementoWriter – для записи в файл, MementoReader – для чтения из файла (файлы gamememento.h/cpp), который также реализует *контроль корректности файла с сохраненными данными.*

- В классе MementoWriter для чтения из файла используется класс std::ifstream, в MementoReader для записи – std::ofstream.
- Классы в конструкторе принимают имя файла. Создаваемый или считываемый файл имеет расширение «.cyb».
- В обоих классах файл открывается в конструкторе (при ошибке бросается соответствующее исключение), а закрывается в деструкторе

Методы классов представлены в табл. 5.

Таблица 5 – Методы классов взаимодействия с файлами

Метод	**Назначение**
void MementoWriter::write (GameParameters param)	Метод записи в файл, принимает на вход структуру параметров игры, записывает в определенном формате все параметры
GameParameters MementoReader::read()	Метод считывания из файла, возвращает структуру параметров игры со считанными данными

Контроль корректности файла реализован такими способами:

- Первой строкой в файле обязано быть его название (то есть название сохранения)
- Разделы в файле должны иметь заголовки: например, перед разделом информации о поле должен быть заголовок “FIELD”, перед разделом

с нейтральными объектами – “NEUTRALS”. Проверка осуществляется в методе read() класса MementoReader.

- В том же методе имеется контроль корректности численных значений параметров. В случае ошибки (недопустимые символы, числа) бросается исключение, загрузка игры в конечном счете не осуществляется.

Пример файла с сохранением игры и пояснениями представлен на рис. 1.

```
1  save2 Название сохранения
2  FIELD
3  6 15 100 11 Параметры поля
4  LAND
5  0 3 1 0 2 0 0 1 4 0 0 3 0 4 2 2 0 0 1 4 0 0 3 0 4 2 0 3 1 0 4 0 0 3 0 4 2 0 3 1 0 2 0 0 1 0 4 2 0 3
   1 0 2 0 0 1 4 0 0 3 3 1 0 2 0 0 1 4 0 0 3 0 4 2 0 0 0 1 4 0 0 3 0 4 2 0 3 1 0 2
6  BASES                                          Массив типов ландшафта
7  3 Число баз
8  0 2 3 100 100 Параметры базы
9  3 Число юнитов на базе
10 3 3 0 150 200 0 0 0 0 ┐
11 1 3 3 150 200 0 0 0 0 ├ Параметры юнитов
12 2 2 4 150 100 0 0 0 0 ┘
13 1 4 6 100 100 Параметры базы
14 1 Число юнитов на базе
15 3 5 0 150 200 0 0 6 -10 Параметры юнита
16 2 4 7 100 100
17 0              Бонусы юнита
18 NEUTRALS
19 4 Число нейтральных объектов
20 1 7 7 Параметры нейтрального объекта
21 4 8 8
22 4 9 9
23 5 7 7
```

Рисунок 1 – Файл сохранения игры save2.cyb

Снимок

Класс GameMemento (файлы gamememento.h/cpp) реализован по принципу *паттерна «Снимок»*. Однако основной задачей класса является скорее не хранение состояния программы (что, впрочем, реализовано), а его обработка, запись и загрузка из файла. У снимка есть имя, соответственно, он будет работать с одноименным файлом. Основные методы и конструкторы класса приведены в табл. 6.

Таблица 6 – Основные методы класса GameMemento

Метод	**Назначение**
GameMemento(std::string name, GameParameters param)	Конструктор для сохранения состояния программы. С помощью класса MementoWriter в файл с именем name записывается состояние, определенное param
GameMemento(std::string name)	Конструктор для последующей загрузки из файла. В теле конструктора загрузка не выполняется
GameParameters loadFromFile();	Метод загрузки состояния из файла. Имя снимка (и файла) определяется в конструкторе, информация считывается с помощью класса MementoReader

Паттерн «Снимок» также предполагает наличие «Создателя», которым является класс Game (файлы game.h/cpp), в котором были добавлены методы для сохранения и загрузки игры. Особенности методов описаны в табл. 7.

Таблица 7 – Методы загрузки и сохранения класса Game

Метод	**Назначение**
GameMemento *Game:: createMemento(std::string name)	Метод создает объект снимка и заполняет его данными из поля (Field) и баз (Base). Практически все параметры уже имеют необходимый тип, преобразования требуются только для перечислений
void Game::restoreMemento (IMemento *memento)	Метод по объекту снимка последовательно восстанавливает состояние игры в следующем порядке: 1. Удаляются старые данные об игре 2. Создается новое поле и посредник для хранения и взаимодействия юнитов 3. Поле заполняется ландшафтом 4. На поле помещаются нейтральные объекты 5. На поле последовательно добавляются базы и их юниты (то есть база, ее юниты, потом следующая база, ее юниты, и так далее). После создания, юнитам присваиваются сохраненные характеристики

Поиск файлов при запуске программы

В классе фасада Facade (файлы facade.h/cpp) в конструкторе был добавлен поиск существующих файлов с расширением «.cyb» в папке программы. Поиск осуществляется с применением библиотеки QDir фреймворка Qt. Таким образом, сразу после запуска игры в UI программы появляется список с названиями существующих сохранений (которые идентичны названиям файлов).

UML-диаграмма

UML диаграмма представлена на рис. 1.

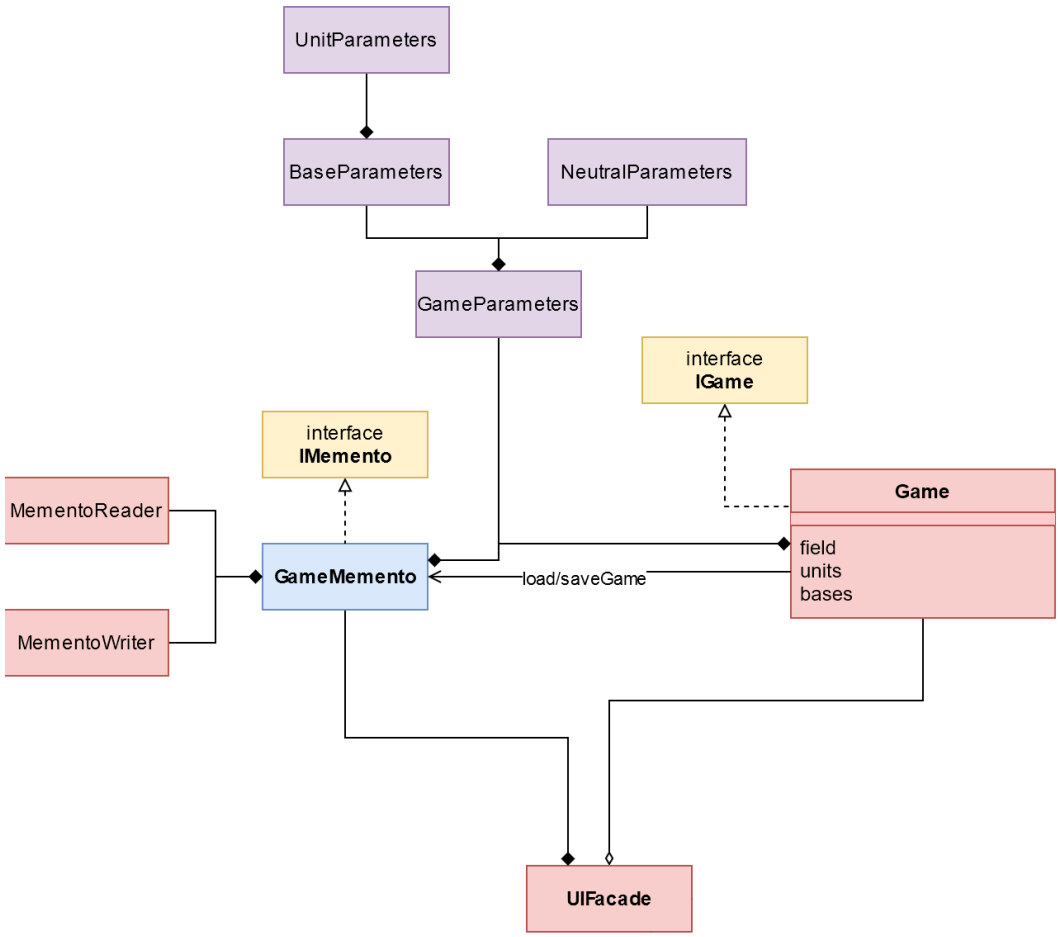

Рисунок 1 – UML-диаграмма

Демонстрационные примеры.

Список сохранений в программе и кнопка загрузки выделены красным прямоугольником на рис. 2.

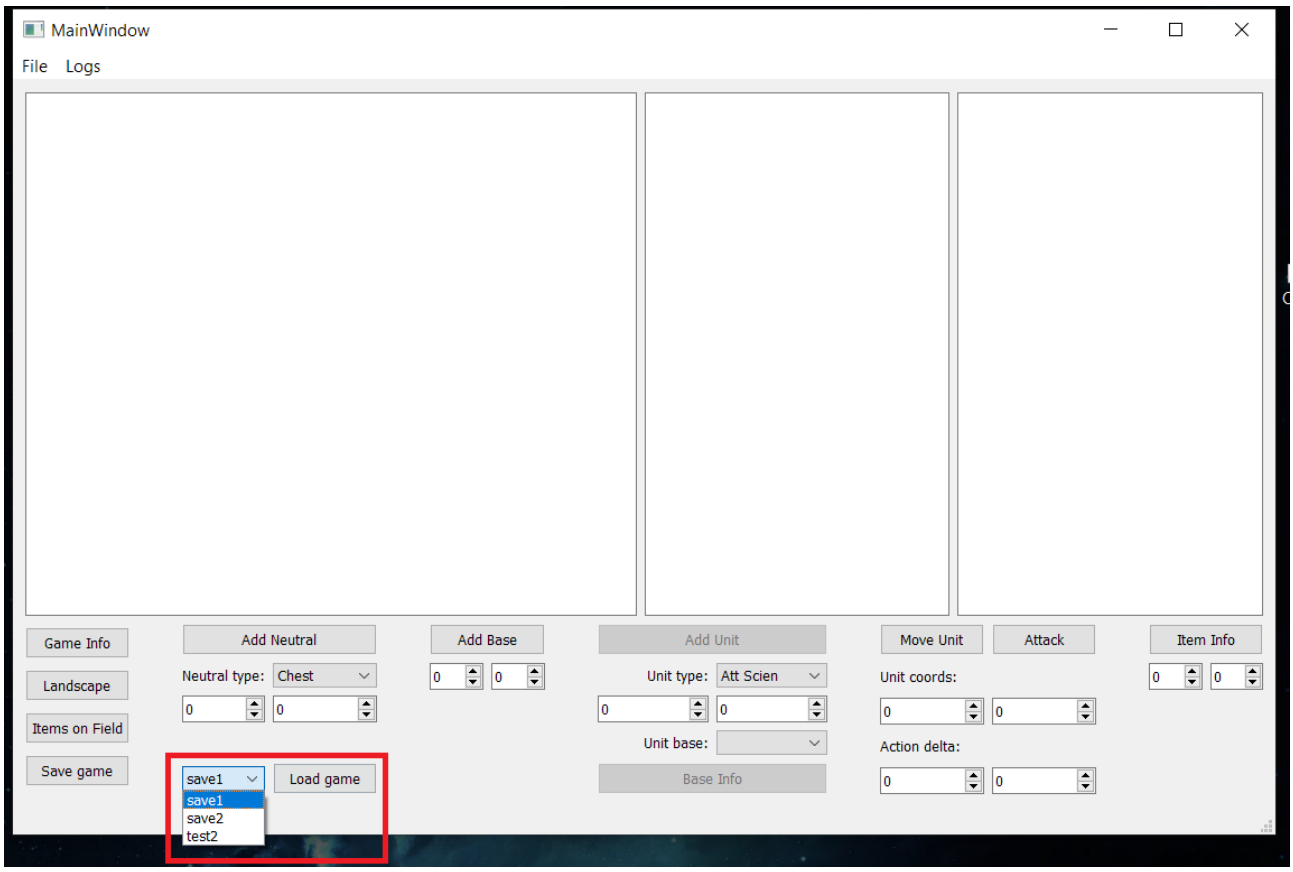

Рисунок 2 – Список сохранений в программе

Текст файла сохранения save1.cyb представлен на рис. 3.

```
save1
FIELD
6 15 100 7
LAND
0 3 1 0 2 0 0 1 4 0 0 3 0 4 2 2 0 0 1 4 0 0 3 0 4 2 0 3 1 0 4 0 0 3 0 4 2 0 3 1 0 2 0 0 1 0 4 2 0 3 1 0 2 0 0 1 4 0 0 3
3 1 0 2 0 0 1 4 0 0 3 0 4 2 0 0 0 1 4 0 0 3 0 4 2 0 3 1 0 2
BASES
2
0 2 3 100 100
3
3 3 0 150 200 0 0 0 0
1 3 3 150 200 0 0 0 0
2 2 4 150 100 0 0 0 0
1 4 6 100 100
1
3 5 0 150 200 0 0 6 -10
NEUTRALS
3
4 8 8
4 9 9
1 7 7
```

Файл сохранения save1.cyb

Вид программы после загрузки сохранения save1 представлен на рис. 4. На рисунке также можно увидеть восстановленные нестандартные параметры юнита по координатам (3,5).

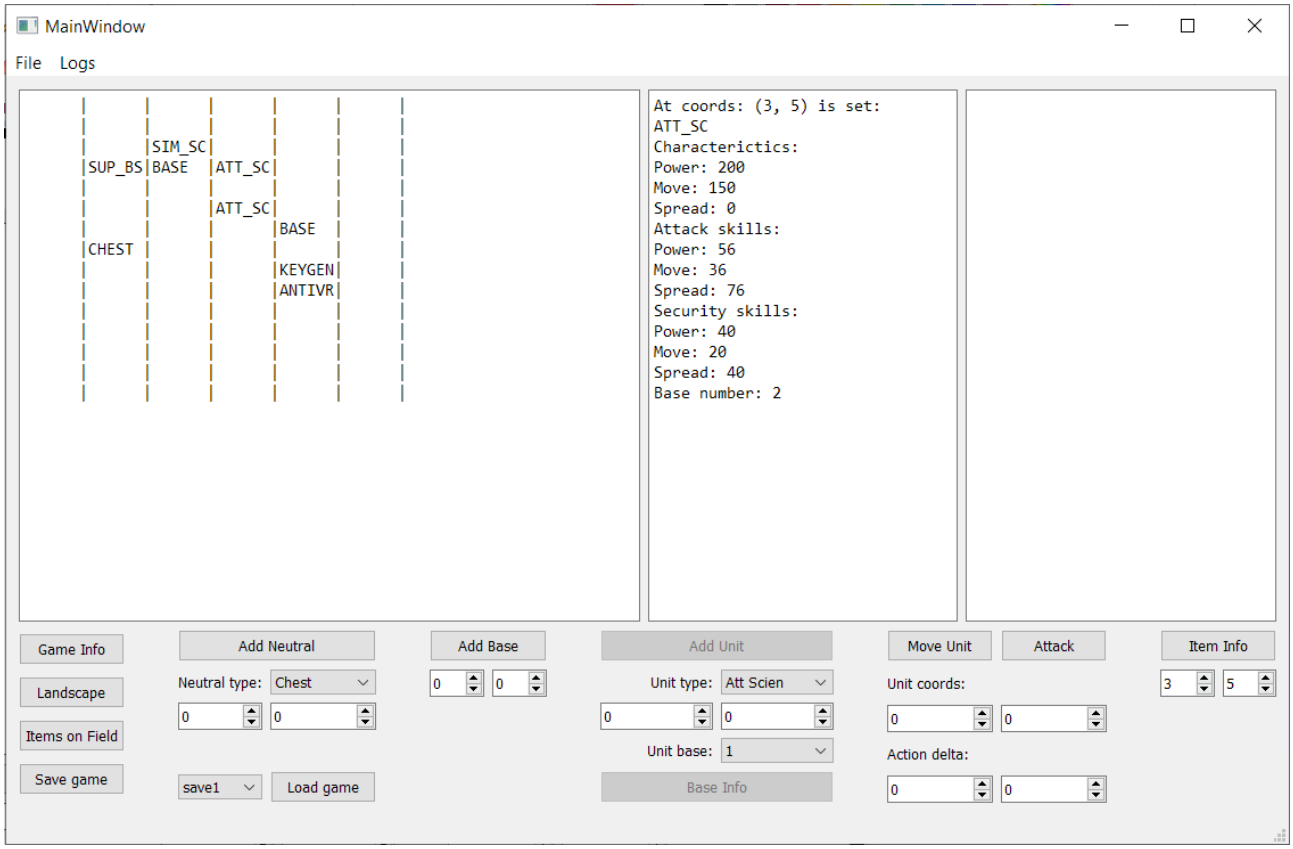

Рисунок 4 – Вид программы после загрузки save1

Текст файла с некорректными данными сохранения представлен на рис. 5.

```
1  test2
2  FIELD
3  6 15 100 4
4  LAND
5  0 3 1 0 2 0 0 1 4 0 0 3 0 4 2 2 0 0 1 4 0 0 3 0 4 2 0 3 1 0 4 0 0 3 0 4 2 0 3 1 0 2 0 0 1 0 4 2 0 3 1 0 2 0 0 1 4 0 0 3
   3 1 0 2 0 0 1 4 0 0 3 0 4 2 0 0 0 1 4 0 0 3 0 4 2 0 3 1 0 2
6  BASES
7  1
8  0 2 0 100 100
9  1
10 2 1 0 150 200 0 0 0 0
11 NEUTRALS
12 2
13 4 7 7
14 4
```

Рисунок 5 – Текст некорректного файла

Вывод ошибки загрузки сохранения из-за некорректного формата файла представлен на рис. 6.

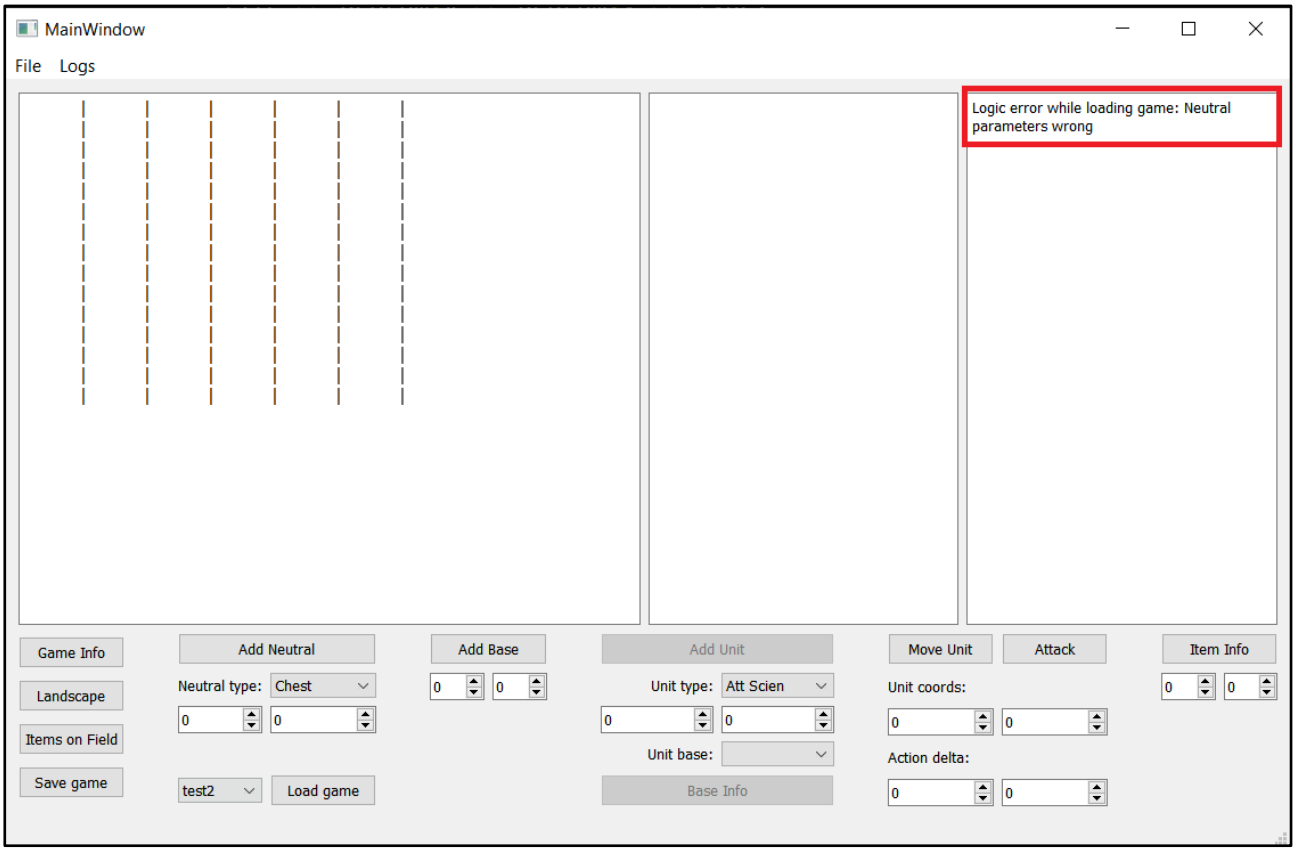

Рисунок 6 – Вывод ошибки загрузки

Выводы.

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для сохранения и загрузки состояния 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. STRUCTURES.H

```
#ifndef STRUCTURES_H
#define STRUCTURES_H

#include "libraries.h"
#include "skills.h"

struct UnitParameters
{
    UnitType type;
    unsigned x;
    unsigned y;
    int move;
    int power;
    int spread;
    int charactBonus;
    int attackBonus;
    int securityBonus;
};

struct NeutralParameters
{
    NeutralParameters(NeutralType type, unsigned x, unsigned y)
        : type(type), x(x), y(y) {}
    NeutralParameters() {}
    NeutralType type;
    unsigned x;
    unsigned y;
};

struct BaseParameters
{
    unsigned number;
    unsigned stability;
    unsigned limit;
    unsigned x;
    unsigned y;
    std::vector<UnitParameters> units;
};

struct GameParameters
{
    GameParameters(unsigned width, unsigned height, unsigned itemLimit, unsigned
itemCounter, size_t baseCount)
```

```
        :       width(width),       height(height),       itemLimit(itemLimit),
itemCounter(itemCounter), baseCount(baseCount) {}
    GameParameters() {}
    unsigned width;
    unsigned height;
    unsigned itemLimit;
    unsigned itemCounter;
    std::vector<LandType> landscape;
    size_t baseCount;
    std::vector<BaseParameters> bases;
    std::vector<NeutralParameters> neutrals;
};

#endif // STRUCTURES_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. GAME.H

```
#ifndef GAME_H
#define GAME_H
#include "gamememento.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);
    GameMemento *createMemento(std::string name);
    void restoreMemento(IMemento *memento);
    ~Game();
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;
    }
}
```

```
GameMemento *Game::createMemento(std::string name)
{
    GameParameters          param(field->getWidth(),          field->getHeight(),
field->getItemLimit(), field->getItemCounter(), bases.size());
    //making landscape info [getting from Field]
    for (unsigned i=0; i<field->getWidth(); i++)
    {
        for (unsigned j=0; j<field->getHeight(); j++)
        {
            param.landscape.push_back(field->getLandType(i, j));
        }
    }
    for (auto i : bases)
    {
        std::pair<unsigned, unsigned> coords = field->getItemCoords(i);
        BaseParameters bParam;
        //making base info [getting from bases vector in Game]
        bParam.x = coords.first;
        bParam.y = coords.second;
        bParam.number = i->getNumber();
        bParam.limit = i->getUnitLimit();
        bParam.stability = i->getStability();
        for (auto j : units->getUnits())
        {
            if (j->getBaseNumber() == i->getNumber())
            {
                //making unit info [getting from units vector in Game]
                UnitParameters uParam;
                coords = field->getItemCoords(j);
                uParam.x = coords.first;
                uParam.y = coords.second;
                uParam.type = static_cast<UnitType>(j->getType());
                uParam.move = j->getCharacteristics().getMove();
                uParam.power = j->getCharacteristics().getPower();
                uParam.spread = j->getCharacteristics().getSpread();
                uParam.charactBonus = j->getCharacteristics().getBonus();
                uParam.attackBonus = j->getAttackSkills().getBonus();
                uParam.securityBonus = j->getSecuritySkills().getBonus();
                bParam.units.push_back(uParam);
            }
        }
        param.bases.push_back(bParam);
    }
    //making neutrals info [getting from Field]
    int type;
    for (unsigned i=0; i<field->getWidth(); i++)
```

```
    {
        for (unsigned j=0; j<field->getHeight(); j++)
        {
            if (field->getItem(i, j) != nullptr)
            {
                type = field->getItem(i, j)->getType();
                if (type >= CHEST && type <= DATA)
                {
                    NeutralParameters nParam(static_cast<NeutralType>(type), i,
j);
                    param.neutrals.push_back(nParam);
                }
            }
        }
    }
    return new GameMemento(name, param);
}

void Game::restoreMemento(IMemento *memento)
{
    GameParameters param = memento->loadFromFile();
    delete field;
    for (auto i : bases)
        delete i;
    bases.clear();
    delete units;
    field = new Field(param.width, param.height, param.itemLimit, OPEN);
    units = new AttackMediator(field);
    unsigned index = 0;
    //recreating landscape
    for (unsigned i=0; i<field->getWidth(); i++)
    {
        for (unsigned j=0; j<field->getHeight(); j++)
        {
            field->addLandscape(i,                    j,                    new
ProxyLandscape(static_cast<LandType>(param.landscape[index++])));
        }
    }
    //recreating neutrals
    for (auto i : param.neutrals)
    {
        createNeutral(i.x, i.y, i.type);
    }
    //recreating bases and units
    for (auto i : param.bases)
    {
        createBase(std::to_string(i.number), i.limit, i.x, i.y);
```

```
        Base *base = getBaseByNumber(i.number);
        for (auto j : i.units)
        {
            Skills skills;
            skills.setAll(j.move, j.power, j.spread, j.charactBonus);
            IUnit *unit = base->createUnit(j.x, j.y, j.type);
            unit->setCharacteristics(skills);
            unit->setAttackBonus(j.attackBonus);
            unit->setSecurityBonus(j.securityBonus);
        }
    }
}

Game::~Game()
{
    delete field;
    delete units;
    for (auto i : bases)
        delete i;
}

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

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. GAMEMEMENTO.H

```
#ifndef GAMEMEMENTO_H
#define GAMEMEMENTO_H

#include "attackmediator.h"

class MementoReader
{
public:
    MementoReader(std::string name);
    GameParameters read();
    ~MementoReader();
private:
    std::ifstream file;
    std::string name;
};

class MementoWriter
{
public:
    MementoWriter(std::string name);
    void write(GameParameters param);
    ~MementoWriter();
private:
    std::ofstream file;
    std::string name;
};

class GameMemento : public IMemento
{
public:
    GameMemento(std::string name, GameParameters param);
    GameMemento(std::string name);
    GameParameters loadFromFile();
private:
    std::string name;
    GameParameters param;
};

#endif // GAMEMEMENTO_H
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. GAMEMEMENTO.CPP

```
#include "gamememento.h"


GameMemento::GameMemento(std::string name, GameParameters param)
    : name(name), param(param)
{
    MementoWriter fout(name);
    fout.write(param);
}

GameMemento::GameMemento(std::string name)
    : name(name) {}

GameParameters GameMemento::loadFromFile()
{
    MementoReader fin(name);
    return fin.read();
}

MementoReader::MementoReader(std::string name)
    : name(name)
{
    file.open(name + ".cyb");
    if (!file.is_open())
        throw std::runtime_error("File open for read failure");
}

MementoWriter::MementoWriter(std::string name)
    : name(name)
{
    file.open(name + ".cyb");
    if (!file.is_open())
        throw std::runtime_error("File open for write failure");
}

void MementoWriter::write(GameParameters param)
{
    //field info write
    file << name << "\nFIELD\n";
    file << param.width << " " << param.height << " " << param.itemLimit << " "
<< param.itemCounter;
    unsigned fieldSize = param.height*param.width;
    //landscape info write
    file << "\nLAND\n";
```

```
    for (unsigned i=0; i<fieldSize; i++)
    {
        file << param.landscape[i] << " ";
    }
    //bases and units info write
    file << "\nBASES\n";
    file << param.baseCount << "\n";
    for (auto i : param.bases)
    {
        file << i.number << " " << i.x << " " << i.y << " " << i.limit << " " <<
i.stability << "\n";
        file << i.units.size() << "\n";
        for (auto j : i.units)
        {
            file << j.x << " " << j.y << " " << j.type << " ";
            file << j.move << " " << j.power << " " << j.spread << " ";
            file  <<  j.charactBonus  <<  "  "  <<  j.attackBonus  <<  "  "  <<
j.securityBonus << "\n";
        }
    }
    //neutrals info write
    file << "NEUTRALS\n";
    file << param.neutrals.size() << "\n";
    for (auto i : param.neutrals)
    {
        file << i.x << " " << i.y << " " << i.type << "\n";
    }
}

GameParameters MementoReader::read()
{
    GameParameters param;
    unsigned tempInt = 0;
    std::string tempStr;

    //getting field info
    file >> tempStr;
    if (tempStr != name)
        throw std::invalid_argument("File heading wrong");
    file >> tempStr;
    if (tempStr != "FIELD")
        throw std::invalid_argument("Field header wrong");
    file >> param.width >> param.height;
    file >> param.itemLimit >> param.itemCounter;

    //getting landscape info
    file >> tempStr;
```

```
    if (tempStr != "LAND")
        throw std::invalid_argument("Landscape header wrong");
    for (unsigned i=0; i<param.width*param.height; i++)
    {
        if (!(file >> tempInt))
            throw std::invalid_argument("Landscape parameters wrong");
        param.landscape.push_back(static_cast<LandType>(tempInt));
    }
    file >> tempStr;

    //getting bases and units info
    if (tempStr != "BASES")
        throw std::invalid_argument("Bases header wrong");
    file >> param.baseCount;
    for (unsigned i=0; i<param.baseCount; i++)
    {
        BaseParameters bParam;
        file >> bParam.number >> bParam.x >> bParam.y >> bParam.limit >>
bParam.stability;
        file >> tempInt;
        for (unsigned i=0; i<tempInt; i++)
        {
            unsigned type;
            UnitParameters uParam;
            file >> uParam.x >> uParam.y >> type;
            file >> uParam.move >> uParam.power >> uParam.spread;
            file >> uParam.charactBonus >> uParam.attackBonus >>
uParam.securityBonus;
            uParam.type = static_cast<UnitType>(type);
            bParam.units.push_back(uParam);
        }
        param.bases.push_back(bParam);
    }

    //getting neutrals info
    file >> tempStr;
    if (tempStr != "NEUTRALS")
        throw std::invalid_argument("Neutrals header wrong");
    file >> tempInt;
    for (unsigned i=0; i<tempInt; i++)
    {
        unsigned type;
        NeutralParameters nParam;
        if (!(file >> nParam.x >> nParam.y >> type))
            throw std::invalid_argument("Neutral parameters wrong");
        nParam.type = static_cast<NeutralType>(type);
        param.neutrals.push_back(nParam);
```

```
    }
    //to file beginning
    file.clear();
    file.seekg(0);
    return param;
}

MementoReader::~MementoReader()
{
    if (file.is_open())
        file.close();
}

MementoWriter::~MementoWriter()
{
    if (file.is_open())
        file.close();
}
```

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);
    void saveGame(std::string name);
    void loadGame(std::string name);
    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"
#include <QDir>

UIFacade::UIFacade(Ui::MainWindow *ui, Game *game)
    : ui(ui), game(game) {
    QDir dir;
    QFileInfoList list = dir.entryInfoList();
    for (auto i : list)
    {
        if (i.suffix() == "cyb")
        {
            ui->comboLoad->addItem(i.baseName());
        }
    }
    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)));
        }
    }
}

void UIFacade::saveGame(std::string name)
{
    try {
        game->createMemento(name);
        ui->comboLoad->addItem(QString::fromStdString(name));
    } catch (std::runtime_error a) {
        std::string output;
        output += "Error while loading game: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
    }
}
```

```
void UIFacade::loadGame(std::string name)
{
    try {
        GameMemento *memento = new GameMemento(name);
        game->restoreMemento(memento);
        ui->comboBases->clear();
        //bases selector restore
        GameCommand gCom(this, game, GAME_INFO, std::vector<int>(), logger);
        std::vector<int> answer = gCom.exec();
        for (int i=0; i<answer[4]; i++)
        {
            ui->comboBases->addItem(QString::number(i+1));
        }
    } catch (std::runtime_error a) {
        std::string output;
        output += "Runtime error while loading game: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
    }
    catch (std::logic_error a) {
        std::string output;
        output += "Logic error while loading game: ";
        output += a.what();
        ui->debugOutput->append(QString::fromStdString(output));
    }
}

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;
}
```