

**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**

Студентка гр. 8381

Преподаватель

\_\_\_\_\_

\_\_\_\_\_

Ивлева О.А.

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.

## **Сохранение и загрузка состояния программы**

В фасаде реализованы методы для сохранения и загрузки игры, которые вызываются из UI в любой момент времени.

Сохранение и считывание из файла реализованы по идиоме *RAII*, открытие файла происходит в конструкторе, а закрытие в деструкторе. Для этого созданы классы `Memento_Read` и `Memento_Write` в файле `memento.h`. Также реализован контроль корректности файла с сохраненными данными, для этого добавлены исключения.

## **Структуры для сохранения состояния**

Для передачи различных параметров игры в снимок используются структуры, представленные ниже.

Таблица 1 – Поля структуры S\_all

Поле	Назначение
vector<S_bonus> s_bonus;	Вектор для хранения бонусов
vector<S_unit> s_unit;	Вектор для хранения юнитов
S_field s_field;	Хранение параметров поля
S_land s_land;	Хранение параметров ландшафта
S_base s_base;	Хранение параметров базы

Таблица 2 – Поля структуры S\_unit

Поле	Назначение
string name;	Тип юнита
unsigned int x; unsigned int y;	Координаты юнита
int health;	Значение здоровья
int nearAttack;	Значение ближней атаки
int furtherAttack;	Значение дальней атаки
int radius_attack;	Значение радиуса атаки

Таблица 3 – Поля структуры S\_base

Поле	Назначение
unsigned int x; unsigned int y;	Координаты базы
unsigned int max_item;	Максимальное кол-во юнитов
unsigned int now_item;	Текущее кол-во юнитов

int health;	Значение здоровья
-------------	-------------------

Таблица 4 – Поля структуры S\_bonus

Поле	Назначение
unsigned int x; unsigned int y;	Координаты бонуса
int health;	Значение бонусного здоровья
int attack;	Значение бонусной атаки

Таблица 5 – Поля структуры S\_field

Поле	Назначение
unsigned int width;	Ширина поля
unsigned int height;	Высота поля

Таблица 6 – Поля структуры S\_land

Поле	Назначение
vector<string> type;	Вектор типов ландшафта

### Паттерн «Снимок»

Класс Memento (файл memento.h) реализован по принципу *паттерна «Снимок»*. В нем реализована обработка, запись и загрузка из файла. Поиск файлов при запуске программы

## Демонстрационные примеры.

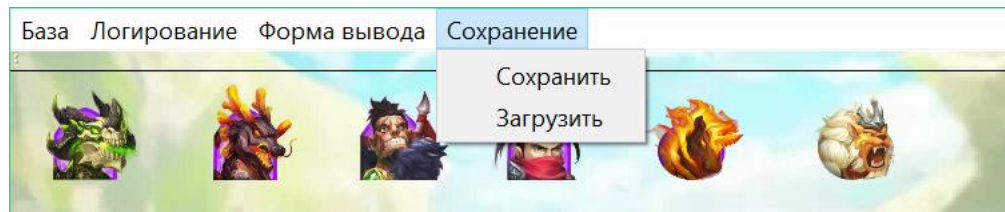


Рисунок 1 – Сохранение и загрузка

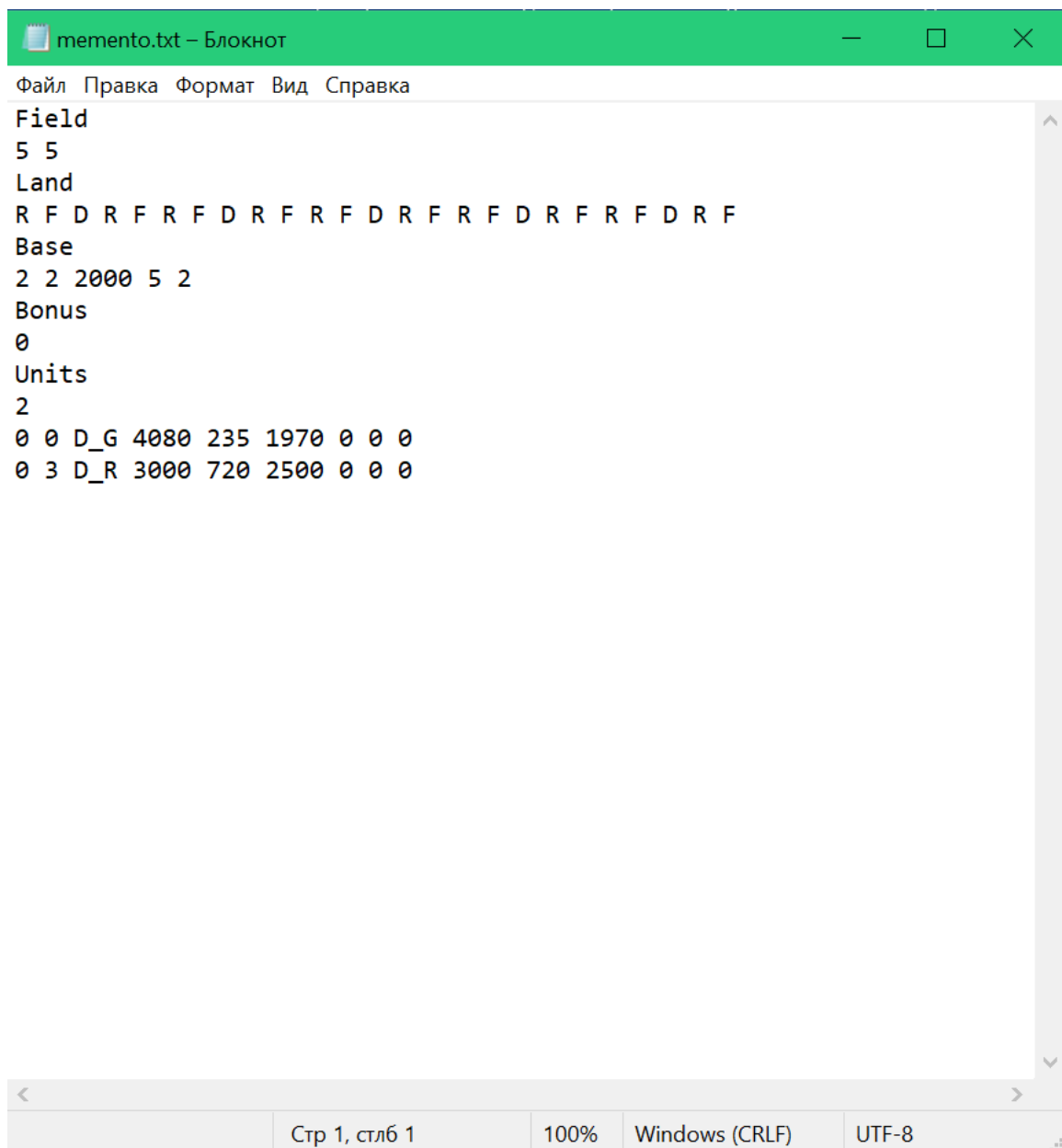


Рисунок 2 – Сохранение в файл

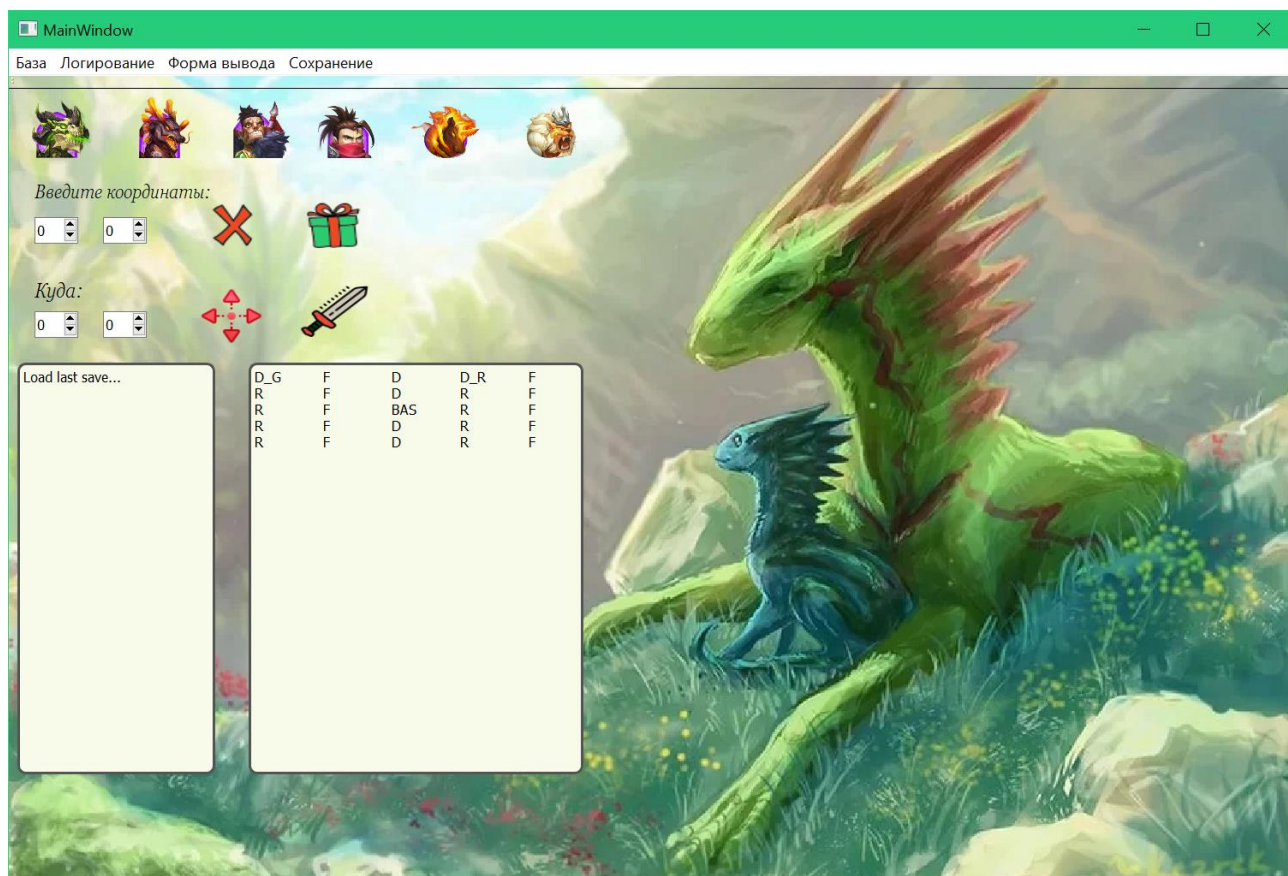


Рисунок 3 – Загрузка из файла

### 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);
    MainWindow w;
    w.show();

    return a.exec();
}
```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ПРОГРАММЫ. BASE.H

```
#include <iostream>
#include <string>
using namespace std;
#pragma once
#include <component.h>

class Base : public IObserver{
    Field *field;
protected:
    ConcreteFactoryDragon concreteFactoryDragon;
    ConcreteFactoryWarrior concreteFactoryWarrior;
    ConcreteFactorySentry concreteFactorySentry;
    Component *units;
    unsigned int width = 5;
    unsigned int height = 5;
    int max_item = 5;
    int now_item = 0;
    int health = 0;
    string who = "BAS";

public:

    Base(Field *obj) : field(obj){
        health = 2000;
        units = new Composite;
        units->Add(new Composite);
        units->Add(new Composite);
        units->Add(new Composite);
        for (unsigned i=0; i<3; i++)
        {
            units->getChildren(i)->Add(new Leaf);
            units->getChildren(i)->Add(new Leaf);
        }
    }

    int getHealth()
    {
        return health;
    }

    void setHealth(int value)
    {
        health = value;
    }
}
```



```

string print_who(){
    return who;
}

string print_units_whoami() {
    return units->getUnits_WhoAmI();
}

Item* copy()
{
    Base* base = new Base(field);
    return base;
}

void add_info(unsigned int x, unsigned int y, string &logs, string s_unit){
    logs += units->getUnits_WhoAmI() + "\n";
    logs += s_unit;
    field->get_unit(x, y, logs);
    now_item++;
}

int getAttack(){}
int get_nearAttack(){}
int get_furtherAttack(){}
virtual void operator+= (Item* bonus){}

void add_green_dr(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* dragon = concreteFactoryDragon.CreateUnitB(this);
        //dragon->setHealth(1000);
        if (field->add_item(x, y, dragon, logs)){
            units->getChildren(0)->getChildren(1)->addUnit(dragon);
            add_info(x, y, logs, "Add green dragon: ");
        }
    }
    else {
        logs += "Maximum number of units!\n";
    }
}

void add_red_dr(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* dragon = concreteFactoryDragon.CreateUnitA(this);
        if (field->add_item(x, y, dragon, logs)){
            units->getChildren(0)->getChildren(0)->addUnit(dragon);
            add_info(x, y, logs, "Add red dragon: ");
        }
    }
}

```

```

    }
    else {
        logs += "Maximum number of units!\n";
    }
}

void add_war_fight(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* fight = concreteFactoryWarrior.CreateUnitA(this);
        //fight->setHealth(1000);
        if (field->add_item(x, y, fight, Logs)){
            units->getChildren(1)->getChildren(0)->addUnit(fight);
            add_info(x, y, Logs, "Add warrior fight: ");
        }
    }
    else {
        logs += "Maximum number of units!\n";
    }
}

void add_war_ronin(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* ronin = concreteFactoryWarrior.CreateUnitB(this);
        //ronin->setHealth(1000);
        if (field->add_item(x, y, ronin, Logs)){
            units->getChildren(1)->getChildren(1)->addUnit(ronin);
            add_info(x, y, Logs, "Add warrior ronin: ");
        }
    }
    else {
        logs += "Maximum number of units!\n";
    }
}

void add_sen_fire(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* fire = concreteFactorySentry.CreateUnitA(this);
        //fire->setHealth(1000);
        if (field->add_item(x, y, fire, Logs)){
            units->getChildren(2)->getChildren(0)->addUnit(fire);
            add_info(x, y, Logs, "Add sentry fire: ");
        }
    }
    else {
        logs += "Maximum number of units!\n";
    }
}

```

```

void add_sen_lion(unsigned int x, unsigned int y, string &logs){
    if (now_item < max_item){
        Item* lion = concreteFactorySentry.CreateUnitB(this);
        if (field->add_item(x, y, lion, logs)){
            units->getChildren(2)->getChildren(1)->addUnit(lion);
            add_info(x, y, logs, "Add sentry lion: ");
        }
    }
    else {
        logs += "Maximum number of units!\n";
    }
}

void delete_unit(Item* obj){
    if (obj->print_who() == "D_R") {
        units->getChildren(0)->getChildren(0)->removeUnit(obj);
    }
    else if (obj->print_who() == "D_G") {
        units->getChildren(0)->getChildren(1)->removeUnit(obj);
    }
    else if (obj->print_who() == "W_F") {
        units->getChildren(1)->getChildren(0)->removeUnit(obj);
    }
    else if (obj->print_who() == "W_R") {
        units->getChildren(1)->getChildren(1)->removeUnit(obj);
    }
    else if (obj->print_who() == "S_F") {
        units->getChildren(2)->getChildren(0)->removeUnit(obj);
    }
    else if (obj->print_who() == "S_L") {
        units->getChildren(2)->getChildren(1)->removeUnit(obj);
    }
    now_item--;
    cout << "Del";
}

};

```

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД ПРОГРАММЫ. ITEM.H

```
#include <string>
#include <iostream>
#include <battle.h>
#include <land.h>
#include <strategy.h>
using namespace std;

class Item{
public:
    virtual ~Item() = default;
    virtual string print_who() = 0;
    virtual int getHealth() = 0;
    virtual int getAttack() = 0;
    virtual int get_nearAttack() = 0;
    virtual int get_furtherAttack() = 0;
    virtual void setHealth(int value) = 0;
    virtual Item* copy() = 0;
    virtual void operator+= (Item* bonus) = 0;
};

class Neutral : public Item{
protected:
    int number = 0;

public:
    virtual void take_obj() = 0;
};

class Bonus : public Neutral{
protected:
    string who = "B";
    int health = 0;
    int attack = 0; //Атака

public:
    int getHealth() {
        return health;
    }
    int getAttack() {
        return attack;
    }
    string print_who() {
        return who;
    }
}
```

```

    int getNumber() {
        return number;
    }
    void setHealth(int value) {}
    int get_nearAttack() {}
    int get_furtherAttack() {}
    void operator+= (Item* bonus) {}
};

class Bonus_health : public Bonus {
public:
    Bonus_health(int num) {
        number = num;
        health = 2000;
    }

    void take_obj() {
        number--;
    }
    Item* copy() {
        Item* bonus = new Bonus_health(number);
        return bonus;
    }
};

class Bonus_attack : public Bonus {
public:
    Bonus_attack(int num) {
        number = num;
        attack = 500;
    }
    void take_obj() {
        number--;
    }
    Item* copy() {
        Item* bonus = new Bonus_health(number);
        return bonus;
    }
};

class Bonus_all : public Bonus {
public:
    Bonus_all(int num) {
        number = num;
        health = 1000;
        attack = 500;
    }
};

```

```

void take_obj(){
    number--;
}
Item* copy(){
    Item* bonus = new Bonus_health(number);
    return bonus;
}
};

```

```

class Fence : public Neutral{
protected:
    string who = "";
    int health = 0;
public:
    Fence(){
        who = "#";
        health = 1000;
    }
    string print_who(){
        return who;
    }
    int getHealth(){
        return health;
    }
    void setHealth(int value){
        health = value;
    }
    void take_obj(){}
    int getAttack(){}
    int get_nearAttack(){}
    int get_furtherAttack(){}
    void operator+= (Item* bonus){}
    Item* copy(){
        Fence* fence = new Fence;
        return fence;
    }
};

```

```

class IObserver : public Item{
public:
    virtual void delete_unit(Item* obj) = 0;
};

```

```

class Unit: public Item{
protected:
    string who = "";

```

```

    int health = 0;
    IObserver *observer;
    Battle near_battle;
    Battle further_battle;

public:

    virtual ~Unit(){
        observer->delete_unit(this);
    };

    virtual void setHealth(int value)
    {
        health = value;
    }

    virtual int getHealth()
    {
        return health;
    }

    virtual string print_who(){
        return who;
    }
    int getAttack(){}
    int get_nearAttack(){
        return near_battle.get_attack();
    }
    int get_furtherAttack(){
        return further_battle.get_attack();
    }
    virtual Item* copy() = 0;

    void operator+= (Item* bonus){
        Strategy* strategy = new Strategy_selector(bonus->getHealth());
        health += strategy->get_bonus(this->print_who());
        near_battle.add_attack(bonus->getAttack());
        further_battle.add_attack(bonus->getAttack());
    }

};

class Dragons : public Unit{

};

class Warrior : public Unit{

```

```

};

class Sentry : public Unit{

};

class Dragons_red : public Dragons{

public:
    Dragons_red(IObserver *observer){
        who = "D_R";
        health = 2000;
        Battle near_battle(10, 220, 217, 1000);
        this->near_battle = near_battle;
        Battle further_battle(5, 2000, 217, 800);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy(){
        Dragons_red* dragon_red = new Dragons_red(observer);
        return dragon_red;
    }

};

class Dragons_green : public Dragons{

public:
    Dragons_green(IObserver *observer){
        who = "D_G";
        health = 4300;
        Battle near_battle(10, 235, 168, 1000);
        this->near_battle = near_battle;
        Battle further_battle(5, 1970, 219, 1200);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy(){
        Dragons_green* dragon_green = new Dragons_green(observer);
        return dragon_green;
    }

};

class Warrior_fight : public Warrior{

```



```

public:
    Warrior_fight(IObserver *observer){
        who = "W_F";
        health = 4300;
        Battle near_battle(10, 235, 168, 1000);
        this->near_battle = near_battle;
        Battle further_battle(25, 1970, 219, 1200);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy(){
        Warrior_fight* warrior_fight = new Warrior_fight(observer);
        return warrior_fight;
    }

};

class Warrior_ronin : public Warrior{

public:
    Warrior_ronin(IObserver *observer){
        who = "W_R";
        health = 2700;
        Battle near_battle(4, 2420, 249, 650);
        this->near_battle = near_battle;
        Battle further_battle(10, 300, 188, 650);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy(){
        Warrior_ronin* warrior_ronin = new Warrior_ronin(observer);
        return warrior_ronin;
    }

};

class Sentry_fire : public Sentry{

public:
    Sentry_fire(IObserver *observer){
        who = "S_F";
        health = 2100;
        Battle near_battle(6, 1820, 249, 1000);
        this->near_battle = near_battle;
    }
};

```

```

        Battle further_battle(10, 5200, 188, 1000);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy() {
        Sentry_fire* sentry_fire = new Sentry_fire(observer);
        return sentry_fire;
    }
};

class Sentry_lion : public Sentry {
public:
    Sentry_lion(IObserver *observer) {
        who = "S_L";
        health = 2700;
        Battle near_battle(8, 2680, 249, 1650);
        this->near_battle = near_battle;
        Battle further_battle(12, 4800, 188, 1650);
        this->further_battle = further_battle;
        this->observer = observer;
    }

    Item* copy() {
        Sentry_lion* sentry_lion = new Sentry_lion(observer);
        return sentry_lion;
    }
};

```

## ПРИЛОЖЕНИЕ Г

### ИСХОДНЫЙ КОД ПРОГРАММЫ. LAND.H

```
#include <string>
#include <iostream>
using namespace std;

class Land{
public:
    virtual bool CheckAccess(string type_unit) const = 0;
    virtual string get_type() = 0;
};

class Desert : public Land{
protected:
    string type = "D";

public:
    string get_type() {
        return type;
    }
    bool CheckAccess(string type_unit) const {
        return true;
    }
};

class Forest : public Land{
protected:
    string type = "F";

public:
    string get_type() {
        return type;
    }
    bool CheckAccess(string type_unit) const {
        return true;
    }
};

class Rocks : public Land{
protected:
    string type = "R";

public:
    string get_type() {
        return type;
    }
}
```

```

        bool CheckAccess(string type_unit) const{
            return true;
        }
};

class Proxy : public Land {
private:
    Land *landsc; //Ландшафт

public:

    Proxy(string type){
        if(type == "D"){
            landsc = new Desert;
        }
        else if (type == "R") {
            landsc = new Rocks;
        }
        else if (type == "F"){
            landsc = new Forest;
        }
    }

    bool CheckAccess(string type_unit) const {
        (((map[x1][y1]->print_who() == "D_R" || map[x1][y1]->print_who() ==
"D_G") && (land[x2][y2]->get_type() == "R"
        //|| land[x2][y2]->get_type() == "D")) ||
((map[x1][y1]->print_who() == "W_F" || map[x1][y1]->print_who() == "W_R") &&
(land[x2][y2]->get_type() == "F"
        //|| land[x2][y2]->get_type() == "D")) ||
((map[x1][y1]->print_who() == "S_F" || map[x1][y1]->print_who() == "S_L") &&
(land[x2][y2]->get_type() == "F"
//|| land[x2][y2]->get_type() == "R"))
        if((type_unit == "D_R" || type_unit == "D_G") && ((landsc->get_type() ==
"R" || landsc->get_type() == "D"))){
            return true;
        }
        if((type_unit == "W_R" || type_unit == "W_F") && ((landsc->get_type() ==
"F" || landsc->get_type() == "D"))){
            return true;
        }
        if((type_unit == "S_F" || type_unit == "S_L") && ((landsc->get_type() ==
"R" || landsc->get_type() == "F"))){
            return true;
        }
        return false;
    }
};

```

```
    }  
  
    string get_type() {  
        return landsc->get_type();  
    }  
};
```

## ПРИЛОЖЕНИЕ Д

### ИСХОДНЫЙ КОД ПРОГРАММЫ.STRATEGY.H

```
#include <iostream>
#include <string>
using namespace std;

class Strategy{
public:
    virtual ~Strategy() {}
    virtual int get_bonus(string type) = 0;
};

class Strategy_dragons : public Strategy{

    int health = 0;
public:
    Strategy_dragons(int health): health(health){}

    int get_bonus(string type){
        return health;
    }
};

class Strategy_warrior : public Strategy{

    int health = 0;
public:
    Strategy_warrior(int health): health(health){}

    int get_bonus(string type){
        return health/2;
    }
};

class Strategy_sentry : public Strategy{

    int health = 0;
public:
    Strategy_sentry(int health): health(health){}

    int get_bonus(string type){
        return health/3;
    }
};

class Strategy_selector : public Strategy{
```

```

    int health = 0;
public:
    Strategy_selector(int health): health(health){}

    int get_bonus(string type){
        if (type[0] == 'D'){
            Strategy_dragons dragon(health);
            return dragon.get_bonus(type);
        }
        else if (type[0] == 'W'){
            Strategy_warrior warrior(health);
            return warrior.get_bonus(type);
        }
        else if (type[0] == 'S'){
            Strategy_sentry sentry(health);
            return sentry.get_bonus(type);
        }
        else {
            return 0;
        }
    }
};

```