

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. 8382

Терехов А.Е.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Реализация сохранения и загрузки состояния программы.

Задание.

Реализация сохранения и загрузки состояния программы. Основные требования:

- Возможность записать состояние программы в файл
- Возможность считать состояние программы из файла

Ход работы.

При запуске теперь можно загрузить игру. Для этого в самом начале необходимо ответить "L" и написать имя сохранения.

В противном случае, если ответить "C", то все происходит как прежде. От пользователя требуются два числа – высота и ширина случайно генерируемого мира (для достижения пропорционального мира рекомендуются значения 10 20). После этого пользователь может модифицировать набор стен и юнитов, следуя выводимым указаниям и отвечая символами, которые заключены в круглые скобки. Затем создается главный герой, необходимо выбрать его отображение – одна из 10 арабских цифр, и его начальное расположение – любая пустая клетка. После этого игрок может передвигаться, используя клавиши wasd. Для сборки на разных платформах пришлось отказаться от функции getch, предоставленной библиотекой conio.h, и на cout, что повлекло за собой подтверждение нажатий. Другими словами, после нажатия на одну из клавиш wasd необходимо нажимать ENTER. Но зато теперь можно перемещаться из одной точки в другую, вводя целую последовательность шагов. Игровая сессия заканчивается в 3 случаях:

1. Игрок был убит другими юнитами.
2. Игрок разбил находящуюся в правом нижнем углу базу.
3. Была нажата клавиша "q".

Для сборки проекта существует файл CMakeLists.txt, генерирующий Makefile, с помощью утилиты cmake.

Изменения, произошедшие в данной лабораторной работе, следуют ниже.

Как было сказано выше была добавлена возможность загрузки мира с самого начала игры. Был изменен конструктор Game.

Листинг 1 – изменения в конструкторе Game.

```
if (answer == 'L') {
    while (!facade.load(*this));
    facade.setBase(world->getCell(world->getWidth() - 2, world-
>getHeight() - 2).getBase());
    mainPlay();
}
```

Игру можно сохранять и загружать прямо во время передвижения. Для этого необходимо вызвать меню клавишей "m", и выбрать сохранение "s" или загрузку "l".

Также в класс MenuFacade был добавлен метод load и в класс World добавлен еще один конструктор, читающий карту из файла.

Листинг 2 – Реализация метода load.

```
bool MenuFacade::load(Game &g) {
    std::string fname;
    std::ifstream file;
    cout << "Enter load name: ";
    cin >> fname;
    std::string path = fname + ".save";
    file.open(path);
    if (!file.is_open()) {
        g.logger << "ERROR: File is not exist!";
        return false;
    }
    delete g.world;
    int h, w;
    char r_char;
    file >> h;
    if (h < 10)
        return false;
    file >> w;
    if (w < 10)
        return false;
    file >> r_char;
    if (r_char > '9' || r_char < '0')
        return false;
    g.playerName = r_char;
    g.world = new World(file, h, w);
    g.coordPlayer = g.findUnit(0);
    file.close();
    return true;
}
```

Листинг 3 – Реализация конструктора для чтения карты из файла.

```
World::World(std::ifstream &file, int h, int w) : height(h), width(w) {
    cout << height << " " << width << endl;
    std::vector<Unit *> units;
```

```

cells = new Cell *[height];
for (int i = 0; i < height; ++i)
    cells[i] = new Cell[width];
char c = 0;
int r_id, r_hp, r_dam, r_arm;
for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
        file >> c;
        switch (c) {
            case '#':
                cells[i][j].setWall<Wall>();
                break;
            case '*':
                cells[i][j].setWall<Rock>();
                break;
            case '^':
                cells[i][j].setWall<Tree>();
                break;
            case 'C':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<Cavalry>(r_id, r_hp, r_dam,
r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case 'I':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<Infantry>(r_id, r_hp, r_dam,
r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case 'S':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<Sniper>(r_id, r_hp, r_dam,
r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case 'R':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<Rifleman>(r_id, r_hp, r_dam,
r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case 'Y':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<YellowWizard>(r_id, r_hp,
r_dam, r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case 'G':
                if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                    cells[i][j].setUnit<GreenWizard>(r_id, r_hp,
r_dam, r_arm);

                    units.push_back(cells[i][j].getUnit());
                }
                break;
            case '$':
                file >> r_hp;
                if (r_hp < 1 || r_hp > 555)

```

```

        r_hp = 555;
        setBase(units, h * w / 20, r_hp);
        break;
    default:
        if (c > '0' && c < '9') {
            if (readUnitParam(file, r_id, r_hp, r_dam, r_arm))
                cells[i][j].setPlayer(c, r_id, r_hp, r_dam,
r_arm);
        }
    }
    break;
}
}
}
}
}

```

По листингу 2 видно, что идиома RAII соблюдена, все отрытые файлы закрываются. При чтении игнорируются нейтральные объекты, что не критично для пользователя.

Были изменены конструкторы юнитов, добавлены параметры по умолчанию, если конструктор вызван без добавленных в этой лабораторной работе, то объекты создаются как прежде, но если были переданы характеристики, то соответствующим полям присваиваются переданные значения. Также присутствует контроль получаемых значений, если характеристики юнита не выглядят правдиво, то юнит не будет создан. Контроль реализован в функции readUnitParam. Код представлен в листинге 4.

Листинг 4 – Чтение параметров юнитов.

```

bool World::readUnitParam(std::ifstream &file, int &id, int &hp, int &dam,
int &arm) {
    file >> id >> hp >> dam >> arm;
    if (id < 0)
        return false;
    if (hp < 1 || hp > 200)
        return false;
    if (dam < 0 || dam > 11)
        return false;
    if (arm < 0 || arm > 11)
        return false;
    return true;
}

```

Сохранение так же предоставляется классом MenuFacade.

Листинг 5 – Реализация метода save.

```

MenuFacade &MenuFacade::save(Game &g) {
    std::string fname;
    cout << "Enter save name: ";
}

```

```

        cin >> fname;
        std::string path = fname + ".save";
        std::ofstream file;
        file.open(path);
        if (!file.is_open()) {
            g.logger << "ERROR: Game not saved";
            return *this;
        }
        file << g.world->getHeight() << " " << g.world->getWidth() << " " <<
g.playerName << endl;
        for (int i = 0; i < g.world->getHeight(); ++i) {
            for (int j = 0; j < g.world->getWidth(); ++j) {
                file << g.world->getCell(j, i);
                if (g.world->getCell(j, i).getIsUnit()) {
                    if (g.world->getCell(j, i).getIsWall())
                        file << " " << g.world->getCell(j, i).getBase()-
>getHealth();
                    else
                        file << " " << g.world->getCell(j, i).getUnit()-
>getID() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getHealth() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getDamage() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getArmor() << " ";
                }
            }
            file << endl;
        }
        file.close();
        return *this;
    }
}

```

На выходе имеем файл с расширением .save.

Его структура:

1. Размеры файла.

2. Имя главного игрока.

3. Карта, состоящая из символьных отображений стен, дорог, юнитов и базы. Также для юнитов сразу за их отображением следуют их характеристики, а для базы – ее здоровье.

Вывод.

В ходе работы были модифицированы ранее написанные классы. В игру была добавлена сериализация. Идиома RAII соблюдена как для сохранения, так и для загрузки. Паттерн "Снимок" не использовался. Значения характеристик юнитов при чтении из файла подвергаются проверке.

ПРИЛОЖЕНИЕ А

Заголовочные файлы.

Logger.h

```
#ifndef OOP_LOGGER_H
#define OOP_LOGGER_H

#include <fstream>
#include <iostream>
#include <chrono>

// интерфейс с рабочей функцией возврата времени
class CurrentTime{
public:
    std::string show();
};

// адаптер для вывода времени в консоль
class CurrentTimeCon{    // Есть класс CurrentTime и нам нужно чтоб
    CurrentTime ct;      // он реализовывал еще вывод времени для
public:                  // консоли, но менять его нельзя.
    std::string showC(); // Молчит.
    std::string show();
};

// класс реализующий вывод логов без вывода логов
class LazyLogger{
public:
    LazyLogger(){};
    LazyLogger& operator<< (const std::string){return *this;};
    ~LazyLogger(){};
};

// класс для вывода логов в файл
// заместитель ленивого логгера
class LoggerF {
    LazyLogger lg; // храним того кого замещаем, несмотря на то что он
ничего не делает
    std::ofstream file;
    CurrentTime time; // использование интерфейса времени
public:
    LoggerF();
    LoggerF& operator<< (const std::string);
    ~LoggerF();
};

// класс заместитель для вывода логов в консоль
class LoggerC {
    LazyLogger lg;      // храним того кого замещаем
    CurrentTimeCon time; // использование переходника для времени
public:
    LoggerC();
    LoggerC& operator<< (const std::string);
};
#endif
```

AbstractObject.h

```
#ifndef OOP_ABSTRACTOBJECT_H
#define OOP_ABSTRACTOBJECT_H
```

```

class AbstractObject {
protected:
    char pict;
public:
    AbstractObject(char pict):pict{pict}{}
    char getPict() const { return pict; }
};

```

```

#endif //OOP_ABSTRACTOBJECT_H

```

Game.h

```

#ifndef OOP_GAME_H

```

```

#define OOP_GAME_H

```

```

#include <iostream>
#include <algorithm>
#include <unistd.h>
#include <sstream>
#include "Logger.h"
#include "World.h"

```

```

class Game;

```

```

class MenuFacade{
    Base* base;
public:
    MenuFacade& setBase(Base* b);
    Base& getBase();
    bool isUnitLimit();
    MenuFacade& addUnit(Game& g);
    MenuFacade& delUnit(Game& g);
    MenuFacade& printInfo(Unit* u = nullptr);
    MenuFacade& printBase();
    MenuFacade& save(Game& g);
    bool load(Game& g);
};

```

```

class Game {
    MenuFacade facade;
    LoggerC logger;
    std::stringstream log;
    char answer;
    World *world;
    char playerName = 0;

```



```

    int objectCount = 0;
    int maxObjCount = 0;
    std::pair<int, int> coordPlayer;
    void addWalls();
    void addUnits();
    void delWall();
    void delUnit(int x = 0, int y = 0);
    std::pair<int, int> getUnitCoord();
    void goTo(std::pair<int, int>& from, std::pair<int, int> to);
    void attack(Cell& attacker, Cell& defender);
    void unitRandomWalk();
    std::pair<int, int> findUnit(Unit* u);
    std::pair<int, int> findUnit(int id);
    void mainPlay();
    void menu();
    void goFor(std::pair<int,int> &coordUnit);
    void createPlayerSession();
public:
    Game();
    void printWorld();
    ~Game();
    friend class MenuFacade;
};

#endif //OOP_GAME_H

```

World.h

```

#ifndef OOP_WORLD_H
#define OOP_WORLD_H
#include <random>
#include <iostream>
#include <fstream>
#include <ctime>
#include "AbstractObject.h"
#include "Unit.h"

class ClosedCells : public AbstractObject {
public:
    ClosedCells(char pict);
};

class Tree : public ClosedCells {
public:
    Tree();
};

class Rock : public ClosedCells {
public:
    Rock();
};

```

```

class Wall : public ClosedCells {
public:
    Wall();
};

class Road : public AbstractObject {
public:
    Road(char pict = '_');
};

class Cell {
protected:
    bool isUnit = false;
    bool isWall = false;
    bool isLoot = false;
    AbstractObject *object;
public:
    explicit Cell(bool isUnit = false, bool isWall = false);
    char getLoot();
    bool getIsLoot() const;
    Unit *getUnit() const;
    bool getIsWall() const;
    bool getIsUnit() const;
    bool isEmpty() const;
    template<class ClosedCellsClass>
    Cell &setWall();
    template<class UnitClass>
    Cell &setUnit(int id = -1, int health = 0, int damage = 0, int armor =
0);

    template<class NeutralClass>
    Cell &setNeutral();
    Cell &setPlayer(char playerName, int t_id = -1, int t_health = 0, int
t_damage = 0, int t_armor = 0);
    Cell &delWall();
    Cell &delUnit();
    Cell &setBase(std::vector<Unit*> units, int maxUnit, int hp = 0);
    Base *getBase();
    Cell &operator=(Cell &from); // перемещение юнитов
    Cell &operator<<(Cell &dest); //копирование клетки
    friend std::ostream &operator<<(std::ostream &out, const Cell &cc);
};

class World {
    int height = 10;
    int width = 10;
protected:
    Cell **cells;
public:
    explicit World(int h, int w, int maxObj, int maxUnit);
    explicit World(std::ifstream &file, int h, int w);
    bool readUnitParam(std::ifstream &file, int &id, int &hp, int &dam,
int &arm);
    World(const World &w);
    void dropLoot();
    World &operator=(const World &w);
    int getHeight() const;
    int getWidth() const;
    Cell &getCell(int x, int y);
    Cell &getCell(std::pair<int, int>coord);
    void switchUnit(int x, int y, int choose());
    void setBase(std::vector<Unit*> units, int maxUnit, int hp = 0);

```

```

        void assistBase(Base& b);
        ~World();
};

```

```

#endif // OOP_WORLD_H

```

Unit.h

```

#ifndef OOP_UNIT_H
#define OOP_UNIT_H

#include <ostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include "NeutralObject.h"
#include "Logger.h"

class IDGenerator {
private:
    static int s_nextID;
public:
    static int getNextID();
};

class Unit : public AbstractObject {
protected:
    int health = 50;
    int damage;
    int armor;
    int id;
    bool isEnemy = true;
public:
    int getDamage() const;

    int getArmor() const;

    Unit(char pict, int t_id = -1, int t_health = 0, int t_damage = 0, int
t_armor = 0);
    bool getIsEnemy() const;
    int getHealth() const;
    int getID() const;
    int giveDamage() const;
    int takeDamage(int dam);
    friend std::ostream &operator<<(std::ostream &out, const Unit &u);
    Unit &operator+=(char n);
};

class Knight : public Unit {
protected:
    explicit Knight(char pict, int t_id = -1, int t_health = 0, int t_damage
= 0, int t_armor = 0);
};

class Ranger : public Unit {
protected:
    explicit Ranger(char pict, int t_id = -1, int t_health = 0, int t_damage
= 0, int t_armor = 0);
};

```

```

class Wizard : public Unit {
protected:
    explicit Wizard(char pict,int t_id = -1, int t_health = 0, int t_damage
= 0, int t_armor = 0);
};

class Cavalry : public Knight {
public:
    Cavalry(int t_id = -1, int t_health = 0, int t_damage = 0, int t_armor
= 0);
};

class Infantry : public Knight {
public:
    Infantry(int t_id = -1, int t_health = 0, int t_damage = 0, int t_armor
= 0);
};

class Sniper : public Ranger {
public:
    Sniper(int t_id = -1, int t_health = 0, int t_damage = 0, int t_armor =
0);
};

class Rifleman : public Ranger {
public:
    Rifleman(int t_id = -1, int t_health = 0, int t_damage = 0, int t_armor
= 0);
};

class YellowWizard : public Wizard {
public:
    YellowWizard(int t_id = -1, int t_health = 0, int t_damage = 0, int
t_armor = 0);
};

class GreenWizard : public Wizard {
public:
    GreenWizard(int t_id = -1, int t_health = 0, int t_damage = 0, int
t_armor = 0);
};

class Player : public Unit {
public:
    explicit Player(char digit ,int t_id = -1, int t_health = 0, int t_damage
= 0, int t_armor = 0);
};

class Base : public AbstractObject {
    std::stringstream log;
    LoggerC logger;
    int maxUnitCount = 0;
    int health;
    std::vector<Unit *> units;
public:
    bool isUnitLimit();
    int getHealth() const;
    explicit Base(std::vector<Unit *> units, int maxUnit, int t_health =
0);

    int takeDamage(int dam);
    void addEnemy(Unit *u);
    void killEnemy(Unit *u);

```

```

        std::vector<Unit *> getUnits();
        void printUnitsInfo();
        void printUnitsInfo(Unit *u);
        void printBase();
        ~Base();
};

#endif //OOP_UNIT_H
NeutralObject.h

#ifndef OOP_NEUTRALOBJECT_H
#define OOP_NEUTRALOBJECT_H

#include "AbstractObject.h"
class NeutralObject : public AbstractObject{
public:
    explicit NeutralObject(char pict);
};

class HealthBox : public NeutralObject{
public:
    HealthBox();
};

class ArmorBox : public NeutralObject{
public:
    ArmorBox();
};

class RandomBox : public NeutralObject{
public:
    RandomBox();
};

class RareBox : public NeutralObject{
public:
    RareBox();
};
#endif //OOP_NEUTRALOBJECT_H

```

ПРИЛОЖЕНИЕ Б

Файлы исходники.

main.cpp

```
#include "Game.h"

int main() {
    std::cout << "Game!" << std::endl;
    Game g;
    g.createPlayerSession();
    return 0;
}
```

Logger.cpp

```
#include "Logger.h"

LoggerF::LoggerF() {
    file.open("LOG.txt", std::ios_base::out);
    if (!file.is_open()) {
        exit(1);
    }
    file << time.show() << "  Start!\n";
}

LoggerF::~~LoggerF() {
    file.close();
}

LoggerF& LoggerF::operator<<(std::string log) {
    file << time.show() << "  " << log << "\n";
    lg << "Очень важно";
    return *this;
}

LoggerC::LoggerC(){
    std::cout << time.show() << "Start!\n";
}

LoggerC &LoggerC::operator<<(std::string log) {
    std::cout << time.show() << log << "\n";
    lg << "Очень важно";
    return *this;
}

std::string CurrentTime::show() {
    auto time =
std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
    std::string str = ctime(&time);
    str.erase(str.end() - 1);
    return str;
}

std::string CurrentTimeCon::showC() {
    return "";
}

std::string CurrentTimeCon::show() {
    return showC();
}
```

World.cpp

```
#include "World.h"

using std::endl;
using std::cin;
using std::cout;
using std::pair;
using std::vector;
using std::rand;

World::World(int h, int w, int maxObj, int maxUnit) : height{h}, width{w}
{
    if (height < 10 || width < 10)
        height = width = 10;
    std::srand(std::time(0));
    cells = new Cell *[height];
    for (int i = 0; i < height; ++i)
        cells[i] = new Cell[width];
    // frame
    for (int i = 0; i < height; ++i) {
        cells[i][0].setWall<Wall>();
        cells[i][width - 1].setWall<Wall>();
    }
    for (int i = 0; i < width; ++i) {
        cells[0][i].setWall<Wall>();
        cells[height - 1][i].setWall<Wall>();
    }
    // create landscape
    for (int i = 0; i < maxObj; ++i) {
        int randi = rand() % height;
        int randj = rand() % width;
        //если нет стены, ставим стену
        if (cells[randi][randj].isEmpty()) {
            if (rand() % 2)
                cells[randi][randj].setWall<Tree>();
            else
                cells[randi][randj].setWall<Rock>();
        } else { --i; }
    }

    // create units
    for (int i = 0; i < maxUnit; ++i) {
        int randi = rand() % height;
        int randj = rand() % width;
        if (cells[randi][randj].isEmpty()) {
            switchUnit(randj, randi, []() { return rand() % 6; });
        } else { --i; }
    }
}

void World::switchUnit(int x, int y, int choose()) {
    switch (choose()) {
        case 0:
            cells[y][x].setUnit<Cavalry>();
            break;
        case 1:
            cells[y][x].setUnit<Infantry>();
            break;
        case 2:
            cells[y][x].setUnit<Sniper>();
            break;
        case 3:
```

```

        cells[y][x].setUnit<Rifleman>();
        break;
    case 4:
        cells[y][x].setUnit<YellowWizard>();
        break;
    case 5:
        cells[y][x].setUnit<GreenWizard>();
        break;
    default:
        break;
    }
}

World::~~World() {
    for (int i = 0; i < height; ++i) {
        delete cells[i];
    }
    delete cells;
}

World::World(std::ifstream &file, int h, int w) : height(h), width(w) {
    cout << height << " " << width << endl;
    std::vector<Unit *> units;
    cells = new Cell *[height];
    for (int i = 0; i < height; ++i)
        cells[i] = new Cell[width];
    char c = 0;
    int r_id, r_hp, r_dam, r_arm;
    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            file >> c;
            switch (c) {
                case '#':
                    cells[i][j].setWall<Wall>();
                    break;
                case '*':
                    cells[i][j].setWall<Rock>();
                    break;
                case '^':
                    cells[i][j].setWall<Tree>();
                    break;
                case 'C':
                    if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                        cells[i][j].setUnit<Cavalry>(r_id, r_hp, r_dam,
r_arm);
                        units.push_back(cells[i][j].getUnit());
                    }
                    break;
                case 'I':
                    if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                        cells[i][j].setUnit<Infantry>(r_id, r_hp, r_dam,
r_arm);
                        units.push_back(cells[i][j].getUnit());
                    }
                    break;
                case 'S':
                    if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
                        cells[i][j].setUnit<Sniper>(r_id, r_hp, r_dam,
r_arm);
                        units.push_back(cells[i][j].getUnit());
                    }
                    break;
                case 'R':

```



```

        if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
            cells[i][j].setUnit<Rifleman>(r_id, r_hp, r_dam,
r_arm);

            units.push_back(cells[i][j].getUnit());
        }
        break;
    case 'Y':
        if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
            cells[i][j].setUnit<YellowWizard>(r_id, r_hp,
r_dam, r_arm);

            units.push_back(cells[i][j].getUnit());
        }
        break;
    case 'G':
        if (readUnitParam(file, r_id, r_hp, r_dam, r_arm)) {
            cells[i][j].setUnit<GreenWizard>(r_id, r_hp,
r_dam, r_arm);

            units.push_back(cells[i][j].getUnit());
        }
        break;
    case '$':
        file >> r_hp;
        if (r_hp < 1 || r_hp > 555)
            r_hp = 555;
        setBase(units, h * w / 20, r_hp);
        break;
    default:
        if (c > '0' && c < '9') {
            if (readUnitParam(file, r_id, r_hp, r_dam, r_arm))
            {
                cells[i][j].setPlayer(c, r_id, r_hp, r_dam,
r_arm);
            }
        }
        break;
    }
}

}

}

bool World::readUnitParam(std::ifstream &file, int &id, int &hp, int &dam,
int &arm) {
    file >> id >> hp >> dam >> arm;
    if (id < 0)
        return false;
    if (hp < 1 || hp > 200)
        return false;
    if (dam < 0 || dam > 11)
        return false;
    if (arm < 0 || arm > 11)
        return false;
    return true;
}

Cell &World::getCell(pair<int, int> coord) {
    return getCell(coord.first, coord.second);
}

Cell &World::getCell(int x, int y) {
    if (y >= 0 && y < height && x >= 0 && x < width)
        return cells[y][x];
    else
        return cells[0][0];
}

```

```

}

int World::getHeight() const {
    return height;
}

int World::getWidth() const {
    return width;
}

World::World(const World &w) : height{w.height}, width{w.width} {
    cells = new Cell *[height];
    for (int i = 0; i < height; ++i)
        cells[i] = new Cell[width];
    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            cells[i][j] = w.cells[i][j];
        }
    }
}

World &World::operator=(const World &w) {
    if (this == &w) {
        return *this;
    }
    height = w.height;
    width = w.width;
    cells = new Cell *[height];
    for (int i = 0; i < height; ++i)
        cells[i] = new Cell[width];
    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            cells[i][j] = w.cells[i][j];
        }
    }
    return *this;
}

void World::setBase(vector<Unit *> units, int maxUnit, int hp) {
    cells[height - 2][width -
2].delUnit().delWall().setBase(std::move(units), maxUnit, hp);
    cells[height - 3][width - 2].delUnit().delWall();
    cells[height - 2][width - 3].delUnit().delWall();
    cells[height - 3][width - 3].delUnit().delWall();
}

void World::dropLoot() {
    while (true) {
        int x = rand() % width;
        int y = rand() % height;
        if (cells[y][x].isEmpty()) {
            if (rand() % 50 == 0) {
                cells[y][x].setNeutral<RareBox>();
                return;
            }
            switch (rand() % 3) {
                case 0:
                    cells[y][x].setNeutral<HealthBox>();
                    break;
                case 1:
                    cells[y][x].setNeutral<ArmorBox>();
                    break;
                case 2:

```

```

        cells[y][x].setNeutral<RandomBox>();
        break;
    }
    break;
}
}

void World::assistBase(Base &b) {
    cells[height - 3][width - 2].setUnit<Cavalry>();
    if (cells[height - 3][width - 2].getUnit()->getID() != 0)
        b.addEnemy(cells[height - 3][width - 2].getUnit());
    cells[height - 2][width - 3].setUnit<Cavalry>();
    if (cells[height - 2][width - 3].getUnit()->getID() != 0)
        b.addEnemy(cells[height - 2][width - 3].getUnit());
    cells[height - 3][width - 3].setUnit<Cavalry>();
    if (cells[height - 3][width - 3].getUnit()->getID() != 0)
        b.addEnemy(cells[height - 3][width - 3].getUnit());
}

Cell::Cell(bool isUnit, bool isWall) : isUnit(isUnit), isWall(isWall),
object(new Road) {}

template<class ClosedCellsClass>
Cell &Cell::setWall() {
    if (isEmpty()) {
        delete object;
        isWall = true;
        object = new ClosedCellsClass;
    }
    return *this;
}

bool Cell::getIsWall() const {
    return isWall;
}

Cell &Cell::setPlayer(char playerName, int t_id, int t_health, int
t_damage, int t_armor) {
    if (isEmpty()) {
        isUnit = true;
        if (t_id != -1)
            object = new Player(playerName, t_id, t_health, t_damage,
t_armor);
        else
            object = new Player(playerName);
    }
    return *this;
}

template<class UnitClass>
Cell &Cell::setUnit(int id, int health, int damage, int armor) {
    if (isEmpty()) {
        delete object;
        isWall = isLoot = false;
        isUnit = true;
        if (id != -1)
            object = new UnitClass(id, health, damage, armor);
        else
            object = new UnitClass;
    }
    return *this;
}

```

```

}

bool Cell::getIsUnit() const {
    return isUnit;
}

Unit *Cell::getUnit() const {
    return getIsUnit() ? static_cast<Unit *>(object) : nullptr;
}

Tree::Tree() : ClosedCells('^') {}

Rock::Rock() : ClosedCells('*') {}

Wall::Wall() : ClosedCells('#') {}

std::ostream &operator<<(std::ostream &out, const Cell &cc) {
    out << cc.object->getPict();
    return out;
}

Cell &Cell::delWall() {
    if (isWall && !isUnit) {
        delete object;
        isWall = isUnit = false;
        object = new Road;
    }
    return *this;
}

Cell &Cell::delUnit() {
    if (isUnit && !isWall) {
        delete object;
        isUnit = isWall = isLoot = false;
        object = new Road;
    }
    return *this;
}

bool Cell::isEmpty() const {
    return !(isWall || isUnit || isLoot);
}

Cell &Cell::operator=(Cell &from) {
    isUnit = true;
    object = from.object;
    from.isUnit = false;
    from.object = new Road();
    return *this;
}

Cell &Cell::setBase(vector<Unit *> units, int maxUnit, int hp) {
    isUnit = isWall = true;
    object = new Base(units, maxUnit, hp);
    return *this;
}

ClosedCells::ClosedCells(char pict) : AbstractObject(pict) {}

Road::Road(char pict) : AbstractObject(pict) {}

```

```

Base *Cell::getBase() {
    if (isUnit && isWall)
        return static_cast<Base *>(object);
    return NULL;
}

template<class NeutralClass>
Cell &Cell::setNeutral() {
    if (isEmpty()) {
        isUnit = isWall = false;
        isLoot = true;
        object = new NeutralClass;
    }
    return *this;;
}

bool Cell::getIsLoot() const {
    return isLoot;
}

char Cell::getLoot() {
    char pic = object->getPict();
    if (isLoot) {
        delete object;
        isLoot = false;
        object = new Road;
    }
    return pic;
}

Cell &Cell::operator<<(Cell &dest) {
    this->isLoot = dest.isLoot;
    this->isUnit = dest.isUnit;
    this->isWall = dest.isWall;
    this->object = dest.object;
    return *this;
}

```

Unit.cpp

```

#include "Unit.h"
#include <iostream>
#include <utility>

using std::cout;
using std::endl;
using std::cin;
using std::pair;
using std::vector;
using std::rand;

Unit::Unit(char pict, int t_id, int t_health, int t_damage, int t_armor) :
AbstractObject(pict),

id{IDGenerator::getNextID()} {
    if (t_id != -1) {
        id = t_id;
        health = t_health;
        damage = t_damage;
        armor = t_armor;
    }
}

```

```

int Unit::getHealth() const {
    return health;
}

int Unit::getID() const {
    return id;
}

int Unit::takeDamage(int dam) {
    int takedDam = (dam - (armor + rand() % 2));
    health = health - takedDam;
    return takedDam;
}

int Unit::giveDamage() const {
    return damage + rand() % 3 + 10;
}

std::ostream &operator<<(std::ostream &out, const Unit &u) {
    out << "ID:\t" << u.id << "\nName:\t" << u.pict << "\nHP:\t" <<
u.health << "\nDAM:\t" << u.damage << "\nARM:\t"
        << u.armor;
    return out;
}

bool Unit::getIsEnemy() const {
    return isEnemy;
}

Unit &Unit::operator+=(char n) {
    switch (n) {
        case '+':
            this->health += rand() % 10 + 10;
            break;
        case 'o':
            this->armor += 1;
            break;
        case 'X':
            this->armor += 5;
            this->damage += 5;
            break;
        case '?':
            switch (rand() % 2) {
                case 0:
                    this->takeDamage(50);
                    break;
                case 1:
                    this->damage += 1;
                    break;
                default:
                    break;
            }
            break;
        default:
            break;
    }
    return *this;
}

int Unit::getDamage() const {
    return damage;
}

```

```

int Unit::getArmor() const {
    return armor;
}

int IDGenerator::s_nextID = 1;

int IDGenerator::getNextID() { return s_nextID++; }

Knight::Knight(char pict, int t_id, int t_health, int t_damage, int
t_armor) : Unit(pict, t_id, t_health, t_damage,
t_armor) {}

Cavalry::Cavalry(int t_id, int t_health, int t_damage, int t_armor) :
Knight('C', t_id, t_health, t_damage, t_armor) {
    if (t_id == -1) {
        armor = 4 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }
}

Infantry::Infantry(int t_id, int t_health, int t_damage, int t_armor) :
Knight('I', t_id, t_health, t_damage, t_armor) {
    if (t_id == -1) {
        armor = 2 + rand() % 3 - 1;
        damage = 4 + rand() % 3 - 1;
    }
}

Ranger::Ranger(char pict, int t_id, int t_health, int t_damage, int
t_armor) : Unit(pict, t_id, t_health, t_damage,
t_armor) {}

Sniper::Sniper(int t_id, int t_health, int t_damage, int t_armor) :
Ranger('S', t_id, t_health, t_damage, t_armor) {
    if (t_id == -1) {
        armor = 1 + rand() % 3 - 1;
        damage = 5 + rand() % 3 - 1;
    }
}

Rifleman::Rifleman(int t_id, int t_health, int t_damage, int t_armor) :
Ranger('R', t_id, t_health, t_damage, t_armor) {
    if (t_id == -1) {
        armor = 3 + rand() % 3 - 1;
        damage = 3 + rand() % 3 - 1;
    }
}

Wizard::Wizard(char pict, int t_id, int t_health, int t_damage, int
t_armor) : Unit(pict, t_id, t_health, t_damage,
t_armor) {}

YellowWizard::YellowWizard(int t_id, int t_health, int t_damage, int
t_armor) : Wizard('Y', t_id, t_health, t_damage,
t_armor) {
    if (t_id == -1) {

```

```

        armor = 1 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }
}

GreenWizard::GreenWizard(int t_id, int t_health, int t_damage, int
t_armor) : Wizard('G', t_id, t_health, t_damage,

t_armor) {
    if (t_id == -1) {
        armor = 3 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }
}

Player::Player(char digit, int t_id, int t_health, int t_damage, int
t_armor) : Unit(digit, t_id, t_health, t_damage,

t_armor) {
    if (t_id == -1) {
        id = 0;
        armor = 6;
        damage = 6;
        health = 100;
    }
    isEnemy = false;
}

Base::Base(vector<Unit *> units, int maxUnit, int t_health)
    : AbstractObject('$'), units(std::move(units)),
    maxUnitCount(maxUnit) {
    health = (t_health != 0) ? t_health : 555;
}

int Base::takeDamage(int dam) {
    log.str("");
    log << "Base: Player fight with Base! (HP: " << health << ")" << endl;
    logger << log.str();
    cout << "You fight with EnemyBase! (HP: " << health << ")" << endl;
    int takedDam = (dam - (10 + rand() % 2));
    health = health - takedDam;
    return takedDam;
}

int Base::getHealth() const {
    return health;
}

void Base::addEnemy(Unit *u) {
    if (isUnitLimit()) {
        logger << "Base: So many units!";
        return;
    }
    auto it = std::find(units.begin(), units.end(), u);
    if (it == units.end()) {
        units.push_back(u);
    }
}

void Base::printUnitsInfo() {
    for (auto u:units) {
        cout << *u << endl << endl;
    }
}

```



```

    }
}

void Base::printUnitsInfo(Unit *u) {
    cout << *u << endl;
}

void Base::killEnemy(Unit *u) {
    logger << "Base: Oh no, my unit died.";
    auto it = std::find(units.begin(), units.end(), u);
    if (it != units.end()) {
        units.erase(it);
    }
}

vector<Unit *> Base::getUnits() {
    return units;
}

void Base::printBase() {
    cout << "Base HP: " << health << endl;
}

bool Base::isUnitLimit() {
    return units.size() >= maxUnitCount;
}

Base::~Base() {
    units.clear();
}

```

Game.cpp

```

#include "Game.h"

#define WINNER 1
#define DEAD 0
using std::cout;
using std::endl;
using std::cin;
using std::pair;
using std::vector;
using std::rand;

Game::Game() {
    vector<Unit *> units;
    answer = '\0';
    world = nullptr;
    int h = 0;
    int w = 0;
    while (answer != 'C' && answer != 'L') {
        cout << "(C)reate or (L)oad?" << endl;
        cin >> answer;
    }
    if (answer == 'C') {
        while (h < 10 || w < 10) {
            cout << "Write height and width: ";
            cin >> h >> w;
        }
        int maxUnitCount = h * w / 20;
        maxObjCount = h * w / 7;
    }
}

```

```

world = new World(h, w, maxObjCount, maxUnitCount);
objectCount = maxObjCount;
int unitCount = maxUnitCount;
for (int i = 0; i < world->getHeight(); ++i) {
    for (int j = 0; j < world->getWidth(); ++j) {
        if (world->getCell(j, i).getIsUnit()) {
            units.push_back(world->getCell(j, i).getUnit());
        }
    }
}
log.str("");
log << "Main: World create with size " << h << " " << w << ".";
logger << log.str();
world->setBase(units, maxUnitCount);
facade.setBase(world->getCell(world->getWidth() - 2, world-
>getHeight() - 2).getBase());
logger << "Main: Base setted.";

printWorld();
while (answer != 'n') {
    cout << "Do you want to do something else with (w)alls or
(u)nits? (n) to start game." << endl;
    cin >> answer;
    if (answer == 'w') {
        cout << "(a)dd or (d)elele?" << endl;
        cin >> answer;
        if (answer == 'a') {
            addWalls();
        } else if (answer == 'd') {
            delWall();
        }
    } else if (answer == 'u') {
        cout << "(a)dd or (d)elele?" << endl;
        cin >> answer;
        if (answer == 'a') {
            facade.addUnit(*this);
        } else if (answer == 'd') {
            facade.delUnit(*this);
        }
    }
}
createPlayerSession();
} else if (answer == 'L') {
    while (!facade.load(*this));
    facade.setBase(world->getCell(world->getWidth() - 2, world-
>getHeight() - 2).getBase());
    mainPlay();
}
}

Game::~Game() {
    logger << "Main: Game ends.";
    delete world;
}

void Game::createPlayerSession() {
    while (playerName > '9' || playerName < '0') {
        cout << "Select the number you want to play for: ";
        cin >> playerName;
    }
    coordPlayer = {0, 0};
    while (world->getCell(coordPlayer).getIsUnit() ||
        world->getCell(coordPlayer).getIsWall()) {

```

```

        cout << "Where? ";
        cin >> coordPlayer.first >> coordPlayer.second;
    }
    world->getCell(coordPlayer).setPlayer(playerName);
    printWorld();
    log.str("");
    log << "Main: Player " << playerName << " created at " <<
coordPlayer.first << " " << coordPlayer.second;
    logger << log.str();
    log.str("");
    mainPlay();
}

```

```

void Game::mainPlay() {
    while (answer != 'q') {
        if (!(rand() % 10)) {
            world->dropLoot();
            logger << "Main: LOOT!";
        }
        if (world->getCell(findUnit(0)).getUnit()->getHealth() <= 0) {
            cout << "Game over!" << endl;
            logger << "Main: Player is dead";
            return;
        }
        if (playerName == WINNER) {
            cout << "You win!!!" << endl;
            logger << "Main: Base is dead";
            return;
        }
        if (answer == 'm') {
            menu();
        }
        goFor(coordPlayer);
        unitRandomWalk();
    }
}

```

```

void Game::goFor(pair<int, int> &coordUnit) {
    printWorld();
    cin >> answer;
    switch (answer) {
        case 'a':
            goTo(coordUnit, pair{coordUnit.first - 1, coordUnit.second});
            break;
        case 'd':
            goTo(coordUnit, pair{coordUnit.first + 1, coordUnit.second});
            break;
        case 'w':
            goTo(coordUnit, pair{coordUnit.first, coordUnit.second - 1});
            break;
        case 's':
            goTo(coordUnit, pair{coordUnit.first, coordUnit.second + 1});
            break;
        default:
            break;
    }
}

```

```

void Game::menu() {
    while (answer != 'b' && answer != 'u' && answer != 'n' && answer !=
's' && answer != 'l') {
        cout << "Do you want doing something with (b)ase, (u)nits, (s)ave,

```

```

(l)oad or (n)ot?" << endl;
    cin >> answer;
}
switch (answer) {
    case 'l': {
        facade.load(*this);
        facade.setBase(world->getCell(world->getWidth() - 2, world-
>getHeight() - 2).getBase());
        break;
    }
    case 's': {
        facade.save(*this);
        break;
    }
    case 'n':
        return;
    case 'b':
        cout << "1. Add Unit" << endl;
        cout << "2. Delete Unit" << endl;
        cout << "3. Get information about units" << endl;
        cout << "4. Get information about base" << endl;

        do {
            cin >> answer;
        } while (answer < '1' || answer > '4');
        switch (answer) {
            case '1':
                facade.addUnit(*this);
                break;
            case '2':
                facade.delUnit(*this);
                break;
            case '3':
                facade.printInfo();
                break;
            case '4':
                facade.printBase();
                break;
            default:
                break;
        }
        break;
    case 'u':
        pair<int, int> coord = getUnitCoord();
        cout << "1. Show information about this Unit." << endl;
        cout << "2. Take a step for him." << endl;
        do {
            cin >> answer;
        } while (answer < '1' || answer > '2');
        switch (answer) {
            case '1':
                facade.printInfo(world->getCell(coord).getUnit());
                break;
            case '2':
                goFor(coord);
                break;
            default:
                break;
        }
        break;
    }
}

```

```

void Game::goTo(pair<int, int> &from, pair<int, int> to) {
    Unit *u = world->getCell(from).getUnit();
    if (world->getCell(to).getIsLoot()) {
        log.str("");
        log << "Main: Unit " << u->getPict() << " with ID: " << u->getID()
<< " take loot at " << to.first << " "
        << to.second << ".";
        logger << log.str();
        log.str("");
        *u += world->getCell(to).getLoot();
        if (u->getHealth() < 0) {
            pair coord = findUnit(u->getID());
            log.str("");
            log << "Main: " << u->getPict() << " is dead.";
            logger << log.str();
            if (u->getPict() == playerName) {
                playerName = DEAD;
                return;
            }
            delUnit(coord.first, coord.second);
            return;
        }
    }
    if (world->getCell(to).isEmpty()) {
        log.str("");
        log << "Main: Unit " << u->getPict() << " with ID: " << u->getID()
<< " go to " << to.first << " " << to.second
        << ".";
        logger << log.str();
        log.str("");
        world->getCell(to) = world->getCell(from);
        from = to;
    } else if (world->getCell(to).getIsUnit())
        attack(world->getCell(from), world->getCell(to));
}

pair<int, int> Game::getUnitCoord() {
    pair<int, int> coord = {0, 0};
    while (!world->getCell(coord).getIsUnit() || world-
>getCell(coord).getIsWall()) {
        cout << "Where?" << endl;
        cin >> coord.first >> coord.second;
    }
    return coord;
}

void Game::addWalls() {
    if (objectCount >= maxObjCount) {
        logger << "Main: Fail try add wall, limit.";
        cout << "So many walls." << endl;
        return;
    }
    int x, y;
    do {
        cout << "Where? (x y): ";
        cin >> x >> y;
        log.str("");
        log << "Main: Fail try add wall, not empty cell. (" << x << " " <<
y << ")";
        logger << log.str();
    } while (!world->getCell(x, y).isEmpty());
    log.str("");
    log << "Main: Create wall in " << x << " " << y;

```

```

        logger << log.str();
        world->getCell(x, y).setWall<Rock>();
        objectCount++;
        printWorld();
    }

    void Game::addUnits() {
        if (facade.isUnitLimit()) {
            logger << "Main: Fail try add unit, limit.";
            cout << "So many units." << endl;
            return;
        }
        int x, y;
        do {
            cout << "Where? (x y): ";
            cin >> x >> y;
            log.str("");
            log << "Main: Fail try add unit, not empty cell. (" << x << " " <<
y << ")";
            logger << log.str();
        } while (!world->getCell(x, y).isEmpty());
        while (true) {
            cout << "What unit do you want to create?" << endl;
            cin >> answer;
            log.str("");
            log << "Main: Add unit ";
            if (answer == 'C') {
                world->getCell(x, y).setUnit<Cavalry>();
                log << "Cavalry ";
                break;
            }
            if (answer == 'I') {
                world->getCell(x, y).setUnit<Infantry>();
                log << "Infantry ";
                break;
            }
            if (answer == 'S') {
                world->getCell(x, y).setUnit<Sniper>();
                log << "Sniper ";
                break;
            }
            if (answer == 'R') {
                world->getCell(x, y).setUnit<Rifleman>();
                log << "Rifleman ";
                break;
            }
            if (answer == 'Y') {
                world->getCell(x, y).setUnit<YellowWizard>();
                log << "YellowWizard ";
                break;
            }
            if (answer == 'G') {
                world->getCell(x, y).setUnit<GreenWizard>();
                log << "GreenWizard ";
                break;
            }
        }
        facade.getBase().addEnemy(world->getCell(x, y).getUnit());
        log << "on " << x << " " << y << ".";
        logger << log.str();
        printWorld();
    }
}

```

```

void Game::delUnit(int x, int y) {
    if (facade.getBase().getUnits().empty()) {
        logger << "Main: Fail try delete unit, no units.";
        return;
    }
    while (!world->getCell(x, y).getIsUnit() || world->getCell(x,
y).getIsWall()) {
        cout << "Where? (x y): ";
        cin >> x >> y;
        log.str("");
        log << "Main: " << "Fail try delete unit, cell without unit. (" <<
x << " " << y << ")";
        logger << log.str();
    }
    log.str("");
    log << "Main: " << "Delete unit " << world->getCell(x, y).getUnit()-
>getPict() << " with ID: "
    << world->getCell(x, y).getUnit()->getID() << " in " << x << " "
<< y << ".";
    logger << log.str();
    facade.getBase().killEnemy(world->getCell(x, y).getUnit());
    world->getCell(x, y).delUnit();
    printWorld();
}

void Game::delWall() {
    if (objectCount <= 0) {
        logger << "Main: Fail try delete wall, no walls.";
        return;
    }
    int x = 0;
    int y = 0;
    do {
        cout << "Where? (x y): ";
        cin >> x >> y;
        log.str("");
        log << "Main: Fail try delete wall, cell without wall. (" << x <<
" " << y << ")";
        logger << log.str();
    } while (!world->getCell(x, y).getIsWall() || world->getCell(x,
y).getIsUnit());
    world->getCell(x, y).delWall();
    log.str("");
    log << "Main: Delete wall in " << x << " " << y << ".";
    logger << log.str();
    objectCount--;
    printWorld();
}

void Game::printWorld() {
    for (int i = 0; i < world->getHeight(); ++i) {
        for (int j = 0; j < world->getWidth(); ++j)
            cout << world->getCell(j, i);
        cout << endl;
    }
    if (playerName >= '0' && playerName <= '9')
        cout << *(world->getCell(findUnit(0)).getUnit()) << endl;
}

void Game::attack(Cell &attacker, Cell &defender) {
    log << "Main: Attack from " << attacker.getUnit()->getIsEnemy() << "
to " << defender.getUnit()->getIsEnemy();
    if (attacker.getUnit()->getIsEnemy() && defender.getUnit()-

```

```

>getIsEnemy()) {
    return;
}
if (attacker.getUnit()->getIsEnemy() && defender.getIsWall())
    return;
int dam = attacker.getUnit()->giveDamage();
int takedDam = 0;
if (!attacker.getUnit()->getIsEnemy() && defender.getIsUnit() &&
defender.getIsWall()) {
    defender.getBase()->takeDamage(dam);
    if (defender.getBase()->getHealth() < 200 && defender.getBase()-
>getHealth() % 5 == 0) {
        logger << "Main: Base take help.";
        world->assistBase(facade.getBase());
    }
    if (defender.getBase()->getHealth() < 0) {
        playerName = WINNER;
    }
} else {
    takedDam = defender.getUnit()->takeDamage(dam);
    log.str("");
    log << "Main: Attack from " << attacker.getUnit()->getPict() << "
to " << defender.getUnit()->getPict()
        << " for " << takedDam << ".";
    logger << log.str();
    log.str("");
    if (defender.getUnit()->getHealth() < 0) {
        pair coord = findUnit(defender.getUnit()->getID());
        log.str("");
        log << "Main: " << defender.getUnit()->getPict() << " is
dead.";

        logger << log.str();
        if (defender.getUnit()->getPict() == playerName) {
            playerName = DEAD;
            return;
        }
        delUnit(coord.first, coord.second);
        return;
    }
}

}

void Game::unitRandomWalk() {
    for (auto u : facade.getBase().getUnits()) {
        pair<int, int> coord = findUnit(u);
        if (world->getCell(coord).getUnit()->getPict() != playerName) {
            pair<int, int> to = coord;
            if (rand() % 2)
                to.first = coord.first + rand() % 3 - 1;
            else
                to.second = coord.second + rand() % 3 - 1;
            goTo(coord, to);
        }
    }
}

pair<int, int> Game::findUnit(Unit *u) {
    for (int i = 0; i < world->getHeight(); ++i)
        for (int j = 0; j < world->getWidth(); ++j)
            if (u == world->getCell(j, i).getUnit())
                return pair(j, i);
    return pair(0, 0);
}

```



```

    }

    pair<int, int> Game::findUnit(int id) {
        for (int i = 0; i < world->getHeight(); ++i)
            for (int j = 0; j < world->getWidth(); ++j)
                if (world->getCell(j, i).getIsUnit() && id == world->getCell(j, i).getUnit()->getID())
                    return pair(j, i);
        return pair(0, 0);
    }

    MenuFacade &MenuFacade::save(Game &g) {
        std::string fname;
        cout << "Enter save name: ";
        cin >> fname;
        std::string path = fname + ".save";
        std::ofstream file;
        file.open(path);
        if (!file.is_open()) {
            g.logger << "ERROR: Game not saved";
            return *this;
        }
        file << g.world->getHeight() << " " << g.world->getWidth() << " " <<
g.playerName << endl;
        for (int i = 0; i < g.world->getHeight(); ++i) {
            for (int j = 0; j < g.world->getWidth(); ++j) {
                file << g.world->getCell(j, i);
                if (g.world->getCell(j, i).getIsUnit()) {
                    if (g.world->getCell(j, i).getIsWall())
                        file << " " << g.world->getCell(j, i).getBase()-
>getHealth();
                    else
                        file << " " << g.world->getCell(j, i).getUnit()-
>getID() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getHealth() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getDamage() <<
                        " " << g.world->getCell(j, i).getUnit()-
>getArmor() << " ";
                }
            }
            file << endl;
        }
        file.close();
        return *this;
    }

    bool MenuFacade::load(Game &g) {
        std::string fname;
        std::ifstream file;
        cout << "Enter load name: ";
        cin >> fname;
        std::string path = fname + ".save";
        file.open(path);
        if (!file.is_open()) {
            g.logger << "ERROR: File is not exist!";
            return false;
        }
        delete g.world;
        int h, w;
        char r_char;
        file >> h;

```

```

        if (h < 10)
            return false;
        file >> w;
        if (w < 10)
            return false;
        file >> r_char;
        if (r_char > '9' || r_char < '0')
            return false;
        g.playerName = r_char;
        g.world = new World(file, h, w);
        g.coordPlayer = g.findUnit(0);
        file.close();
        return true;
    }

Base &MenuFacade::getBase() {
    return *(base);
}

MenuFacade &MenuFacade::addUnit(Game &g) {
    g.addUnits();
    return *this;
}

MenuFacade &MenuFacade::setBase(Base *b) {
    base = b;
    return *this;
}

bool MenuFacade::isUnitLimit() {
    return (base->isUnitLimit());
}

MenuFacade &MenuFacade::delUnit(Game &g) {
    g.delUnit();
    return *this;
}

MenuFacade &MenuFacade::printInfo(Unit *u) {
    if (u == NULL)
        base->printUnitsInfo();
    else
        base->printUnitsInfo(u);
    return *this;
}

MenuFacade &MenuFacade::printBase() {
    base->printBase();
    return *this;
}

```

NeutralObject.cpp

```

#include "NeutralObject.h"

NeutralObject::NeutralObject(char pict) : AbstractObject(pict) {}

HealthBox::HealthBox() : NeutralObject('+') {}

ArmorBox::ArmorBox() : NeutralObject('o') {}

RandomBox::RandomBox() : NeutralObject('?') {}

```

```
RareBox::RareBox() : NeutralObject('X') {}
```

ПРИЛОЖЕНИЕ В

```
10 20 1
#####
#_____o_**_____o_____#
#_*______*_*_^^_____ *#
#_____^_____G 6 17 40 3 C 10 50 1 4 _____#
#_____^_____#
#1 0 87 6 6 _____^^^_*_____**_#
#_____ * _____ * _____#
#^_____ * _* _____ * _____#
#o*_____ * _____o_____ * _____$ 555#
#####
```