

**МИНОБРНАУКИ РОССИИ**  
**Санкт-Петербургский государственный**  
**электротехнический университет**  
**«ЛЭТИ» им. В.И. Ульянова (Ленина)**  
**Кафедра математического обеспечения и применения ЭВМ**

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

Студент гр. 8382

Терехов А.Е.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

### **Цель работы.**

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

### **Задание.**

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

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

### **Ход работы.**

В предыдущих лабораторных работах был реализован интерфейс, который не помешало бы напомнить.

При запуске программы первым делом 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 пока что остановиться на функции getchar, 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.

В классе Game метод run был разбит на логические части. Метод createPlayerSession теперь создает игровую сессию с конкретным игроком, за которого пользователю предстоит играть. В методе реализовано чтение представления игрока, его координат, и размещение его на поле.

#### Листинг 1 – Реализация создания игровой сессии.

```
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();
    mainPlay();
}
```

Следом вызывается метод mainPlay, представляющий из себя цикл, в котором игрок либо передвигается по полю, либо вызывает меню, нажав в процессе игры клавишу "m", в котором может управлять базой или остальными юнитами. Также в данном цикле реализовано разбрасывание нейтральных предметов из предыдущей лабораторной.

#### Листинг 2 – Реализация основного цикла игры.

```
void Game::mainPlay() {
    while (answer != 'q') {
        if (!(rand() % 10)) {
            world->dropLoot();
            cout << "LOOT!" << endl;
        }
        if (world->getCell(coordPlayer).getUnit()->getHealth() <= 0) {
            cout << "Game over!" << endl;
            return;
        }
        if (playerName == WINNER) {
            cout << "You win!!!" << endl;
            return;
        }
        if (answer == 'm')
            menu();
        pair p = coordPlayer;
        goFor(coordPlayer);
    }
}
```

```

        if (p.first - coordPlayer.first != 0 || p.second -
coordPlayer.second != 0)
            unitRandomWalk();
    }}

```

Метод `goFor`, позволяет передвигаться за какого-либо юнита, по умолчанию за главного героя, но при желании, вызвав меню, можно сходить за вражеского юнита. Вызываемый метод `goTo` позволяет юнитам не только передвигаться, но и собирать коробки и сражаться.

### Листинг 3 – Реализация передвижений юнитов.

```

void Game::goFor(pair<int, int> &coordUnit) {
    printWorld();
    answer = getchar();
    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::goTo(pair<int, int> &from, pair<int, int> to) {
    if (world->getCell(to).getIsLoot())
        *(world->getCell(from).getUnit()) += world->getCell(to).getLoot();
    if (world->getCell(to).isEmpty()) {
        world->getCell(to) = world->getCell(from);
        from = to;
    } else if (world->getCell(to).getIsUnit())
        attack(world->getCell(from), world->getCell(to));
}

```

Как было сказано выше в процессе игры пользователь может вызвать меню нажав "m". В этом меню он может выбрать с чем он хочет что-то сделать. С базой представлено 4 варианта: добавить или удалить юнита, принадлежащего базе, посмотреть информацию обо всех юнитах, принадлежащих базе, или посмотреть текущее состояние самой базы. Прежде чем производить какие-либо действия с конкретным юнитом, необходимо сообщить его координаты. После этого будет возможность либо узнать информацию о нем, либо сделать за него шаг. Шагать за другого юнита можно неограниченное количество раз, и при этом юниты не

будут случайно передвигаться, в отличие от ситуации, когда передвигается главный персонаж.

#### Листинг 4 – Реализация меню (до применения паттерна Фасад).

```
void Game::menu() {
    while (answer != 'b' && answer != 'u' && answer != 'n') {
        cout << "Do you want doing something with (b)ase or (u)nits or
(n)ot?" << endl;
        cin >> answer;
    }
    if (answer == 'n') {
        return;
    }
    if (answer == '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':
                addUnits();
                break;
            case '2':
                delUnit();
                break;
            case '3':
                world->getCell(baseCoord).getBase()->printUnitsInfo();
                break;
            case '4':
                world->getCell(baseCoord).getBase()->printBase();
            default:
                break;
        }
    } else if (answer == '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':
                world->getCell(baseCoord).getBase()->printUnitsInfo(world-
>getCell(coord).getUnit());
                break;
            case '2':
                goFor(coord);
                break;
            default:
                break;
        }
    }
}
```

Но меню как показалось имеет странную реализацию – в зависимости от выбранного пункта вызываются совершенно не связанные методы. Поэтому

было принято решение использовать паттерн Фасад.

#### Листинг 5 – объявление класса MenuFacade.

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

Определения методов очень просты и зачастую не занимают более одной строки.

#### Листинг 6 – определение методов класса MenuFacade.

```
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;
}
```

Может показаться бесполезным данное использование паттерна, но можно убедиться в обратном, если обратить внимание на то, как преобразовалось меню базы.

## Листинг 7 – Меню для базы (после применения паттерна Фасад).

```
if (answer == '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();
        default:
            break;
    }
}
```

Теперь стала очевидна взаимосвязь между пунктами меню.

Более подробно с кодом можно ознакомиться в приложениях А и Б.

### **Вывод.**

В ходе работы были модифицированы ранее написанные функции. В игру были добавлены новые возможности, такие как взаимодействие с базой и юнитами посредством меню. Для удобной реализации меню был использован паттерн фасад. Основные требования выполнены.

## ПРИЛОЖЕНИЕ А

Заголовочные файлы.

### 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 "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 = NULL);
    MenuFacade& printBase();
};

class Game {
    MenuFacade facade;
    char answer;
    World *world;
    char playerName = 0;
    int objectCount = 0;
    int maxObjCount = 0;
    std::pair<int, int> coordPlayer;
    void addWalls();
    void addUnits();
    void delWall();
    std::pair<int, int> getUnitCoord();
    void delUnit(int x = 0, int y = 0);
    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);
public:
        void createPlayerSession();
        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 <conio.h>
#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:
    char getLoot();
    Cell &setRoad();
    Cell &setBase(std::vector<Unit*> units, int unitCount, int maxUnit);
    bool getIsLoot() const;
    Base *getBase();
    bool isEmpty() const;
};

```

```

Unit *getUnit() const;
friend std::ostream &operator<<(std::ostream &out, const Cell &cc);
explicit Cell(bool isUnit = false, bool isWall = false);
template<class ClosedCellsClass>
Cell &setWall();
template<class UnitClass>
Cell &setUnit();
template<class NeutralClass>
Cell &setNeutral();
Cell &setPlayer(char playerName);
Cell &delWall();
Cell &delUnit();
bool getIsWall() const;
bool getIsUnit() const;
Cell &operator=(Cell &from);
};

class World {
    int height = 10;
    int width = 10;
protected:
    Cell **cells;
public:
    void assistBase(Base& b);
    void dropLoot();
    explicit World(int h, int w, int maxObj, int maxUnit);
    explicit World(std::ifstream &file);
    World(const World &w);
    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 unitCount, int maxUnit);
    ~World();
};

#endif //OOP_WORLD_H

```

## Unit.h

```

#ifndef OOP_UNIT_H
#define OOP_UNIT_H

#include <ostream>
#include <vector>
#include <algorithm>
#include "NeutralObject.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:
    Unit(char pict);
    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);
};

class Ranger : public Unit {
protected:
    explicit Ranger(char pict);
};

class Wizard : public Unit {
protected:
    explicit Wizard(char pict);
};

class Cavalry : public Knight {
public:
    Cavalry();
};

class Infantry : public Knight {
public:
    Infantry();
};

class Sniper : public Ranger {
public:
    Sniper();
};

class Rifleman : public Ranger {
public:
    Rifleman();
};

class YellowWizard : public Wizard {
public:
    YellowWizard();
};

class GreenWizard : public Wizard {
public:
    GreenWizard();
};

class Player : public Unit {
public:
    explicit Player(char digit);
};

```

```

class Base : public AbstractObject {
    int unitCount = 0;
    int maxUnitCount = 0;
    int health;
    std::vector<Unit *> units;
public:
    bool isUnitLimit();
    int getHealth() const;
    explicit Base(std::vector<Unit *> units, int unitCount, int maxUnit);
    int takeDamage(int dam);
    void addEnemy(Unit *u);
    void killEnemy(Unit *u);
    std::vector<Unit *> getUnits();
    void printUnitsInfo();
    void printUnitsInfo(Unit *u);
    void printBase();
};

#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;
}

```

## World.cpp

```

#include "World.h"
using std::cout;
using std::endl;
using std::cin;
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) {
    file >> height >> width;
    cells = new Cell *[height];
    for (int i = 0; i < height; ++i)
        cells[i] = new Cell[width];
    char c = 0;
    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':
                    cells[i][j].setUnit<Cavalry>();
                    break;
                case 'I':
                    cells[i][j].setUnit<Infantry>();
                    break;
                case 'S':
                    cells[i][j].setUnit<Sniper>();
                    break;
                case 'R':
                    cells[i][j].setUnit<Rifleman>();
                    break;
                case 'Y':
                    cells[i][j].setUnit<YellowWizard>();
                    break;
                case 'G':
                    cells[i][j].setUnit<GreenWizard>();
                    break;
            }
        }
    }
}

```

```

        }
    }
}
file.close();
}

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 < height; ++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 < height; ++j) {
            cells[i][j] = w.cells[i][j];
        }
    }
    return *this;
}

void World::setBase(vector<Unit *> units, int unitCount, int maxUnit) {
    cells[height - 2][width -
2].delUnit().delWall().setBase(std::move(units), unitCount, maxUnit);
    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) {
    if (isEmpty()) {
        isUnit = true;
        object = new Player(playerName);
    }
    return *this;
}

template<class UnitClass>
Cell &Cell::setUnit() {
    if (isEmpty()) {

```



```

        delete object;
        isWall = isLoot = false;
        isUnit = true;
        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 unitCount, int maxUnit) {
    isUnit = isWall = true;
    object = new Base(units, unitCount, maxUnit);
    return *this;
}

```

```

Cell &Cell::setRoad() {
    delete object;
    isUnit = isWall = isLoot = false;
    object = new Road;
    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;
}

```

## 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) : AbstractObject(pict), id{IDGenerator::getNextID()}
{}

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) {
        cout << n << endl;
        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 IDGenerator::s_nextID = 1;

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

    Knight::Knight(char pict) : Unit(pict) {}

    Cavalry::Cavalry() : Knight('C') {

```

```

        armor = 4 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }

    Infantry::Infantry() : Knight('I') {
        armor = 2 + rand() % 3 - 1;
        damage = 4 + rand() % 3 - 1;
    }

    Ranger::Ranger(char pict) : Unit(pict) {}

    Sniper::Sniper() : Ranger('S') {
        armor = 1 + rand() % 3 - 1;
        damage = 5 + rand() % 3 - 1;
    }

    Rifleman::Rifleman() : Ranger('R') {
        armor = 3 + rand() % 3 - 1;
        damage = 3 + rand() % 3 - 1;
    }

    Wizard::Wizard(char pict) : Unit(pict) {}

    YellowWizard::YellowWizard() : Wizard('Y') {
        armor = 1 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }

    GreenWizard::GreenWizard() : Wizard('G') {
        armor = 3 + rand() % 3 - 1;
        damage = 2 + rand() % 3 - 1;
    }

    Player::Player(char digit) : Unit(digit) {
        id = 0;
        armor = 6;
        damage = 6;
        health = 100;
        isEnemy = false;
    }

    Base::Base(vector<Unit *> units, int unitCount, int maxUnit)
        : AbstractObject('$'), health(555), units(std::move(units)),
          unitCount(unitCount), maxUnitCount(maxUnit) {}

    int Base::takeDamage(int dam) {
        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()){
            cout << "Base: So many units!" << endl;
            return;
        }
    }

```

```

    }
    auto it = std::find(units.begin(), units.end(), u);
    if (it == units.end()) {
        units.push_back(u);
        unitCount++;
    }
    printUnitsInfo(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) {
    cout << "Base: Oh no, my unit died("
    auto it = std::find(units.begin(), units.end(), u);
    if (it != units.end()) {
        units.erase(it);
        unitCount--;
    }
}

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

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

bool Base::isUnitLimit() {
    return unitCount >= maxUnitCount;
}

```

## Game.cpp

```

#include "Game.h"

#define WINNER 1
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 (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());
            }
        }
    }
    world->setBase(units, unitCount, maxUnitCount);
    facade.setBase(world->getCell(world->getWidth() - 2, world->getHeight() - 2).getBase());
    // baseCoord.first = world->getWidth() - 2;
    // baseCoord.second = world->getHeight() - 2;
    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);
            }
        }
    }
}

Game::~~Game() {
    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();
    mainPlay();
}

```

```

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

void Game::goFor(pair<int, int> &coordUnit) {
    printWorld();
    answer = getchar();
    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') {
        cout << "Do you want doing something with (b)ase or (u)nits or (n)ot?" << endl;
        cin >> answer;
    }
    if (answer == 'n') {
        return;
    }
    if (answer == '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();
    default:
        break;
    }
} else if (answer == '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;
    }
}
}

void Game::goTo(pair<int, int> &from, pair<int, int> to) {
    if (world->getCell(to).getIsLoot())
        *(world->getCell(from).getUnit()) += world->getCell(to).getLoot();
    if (world->getCell(to).isEmpty()) {
        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) {
        cout << "So many walls." << endl;
        return;
    }
    int x, y;
    do {
        cout << "Where? (x y): ";
        cin >> x >> y;
    } while (!world->getCell(x, y).isEmpty());
}

```



```

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

void Game::addUnits() {
    if (facade.isUnitLimit()) {
        cout << "So many units." << endl;
        return;
    }
    int x, y;
    do {
        cout << "Where? (x y): ";
        cin >> x >> y;
    } while (!world->getCell(x, y).isEmpty());
    while (true) {
        cout << "What unit do you want to create?" << endl;
        cin >> answer;
        if (answer == 'C') {
            world->getCell(x, y).setUnit<Cavalry>();
            break;
        }
        if (answer == 'I') {
            world->getCell(x, y).setUnit<Infantry>();
            break;
        }
        if (answer == 'S') {
            world->getCell(x, y).setUnit<Sniper>();
            break;
        }
        if (answer == 'R') {
            world->getCell(x, y).setUnit<Rifleman>();
            break;
        }
        if (answer == 'Y') {
            world->getCell(x, y).setUnit<YellowWizard>();
            break;
        }
        if (answer == 'G') {
            world->getCell(x, y).setUnit<GreenWizard>();
            break;
        }
    }
    facade.getBase().addEnemy(world->getCell(x, y).getUnit());
    printWorld();
}

void Game::delUnit(int x, int y) {
    while (!world->getCell(x, y).getIsUnit() || world->getCell(x,
y).getIsWall()) {
        cout << "Where? (x y): ";
        cin >> x >> y;
    }
    cout << "Unit " << world->getCell(x, y).getUnit()->getPict() << " with
ID: "
        << world->getCell(x, y).getUnit()->getID() << " is deleted." <<
endl;
    facade.getBase().killEnemy(world->getCell(x, y).getUnit());
    world->getCell(x, y).delUnit();
    printWorld();
}

void Game::delWall() {

```

```

        if (objectCount <= 0)
            return;
        int x = 0;
        int y = 0;
        do {
            cout << "Where? (x y): ";
            cin >> x >> y;
        } while (!world->getCell(x, y).getIsWall() || world->getCell(x,
y).getIsUnit());
        world->getCell(x, y).delWall();
        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) {
        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) {
                world->assistBase(facade.getBase());
            }
            if (defender.getBase()->getHealth() < 0) {
                playerName = WINNER;
            }
        } else {
            takedDam = defender.getUnit()->takeDamage(dam);
            cout << "Attack from " << attacker.getUnit()->getPict() << " to "
<< defender.getUnit()->getPict()
                << " for "
                << takedDam << endl;
            if (defender.getUnit()->getHealth() < 0) {
                pair coord = findUnit(defender.getUnit()->getID());
                cout << defender.getUnit()->getPict() << " is dead." << endl;
                if (defender.getUnit()->getPict() == playerName) {
                    playerName = 0;
                    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);
}

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') {}
```