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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объенктно-ориентированное программирование»

Тема: Создание классов, конструкторов классов, методов классов; наследование.

Студент гр. 8382 _____ Терехов А.Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Создать наборы классов для поля и юнитов; для классов создать конструкторы и необходимые методы; использовать наследование.

Задание.

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера
- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)
- Для хранения запрещается использовать контейнеры из stl

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа(например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Ход работы.

Для игры был реализован соответствующий класс Game:

```
class Game {
    char answer;
    World *world;
    char playerName;
    int playerX;
```

```
    int playerY;
    int objectCount = 0;
    int unitCount = 0;
    int maxObjCount = 0;
    int maxUnitCount = 0;
    std::vector<int> unitsID;
    void addWalls();
    void addUnits();
    void delWall();
    void delUnit();
    void go(Side side, int& x, int& y);
    void unitRandomWalk();
public:
    Game();
    void run();
    void printWorld();
    void saveWorld(const char *fileName);
    std::pair<int, int> findUnit(int id);
    ~Game();
};
```

Класс хранит счетчики, массив ID юнитов, карту, и информацию о главном игроке. В конструкторе можно как считать карту из файла, так и сгенерировать случайную. В методе run реализована основная логика игры.

```
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);
    World(const World& w);
    World& operator= (const World& w);
    int getHeight() const;
    int getWidth() const;
    Cell &getCell(int x, int y);
    void switchUnit(int x, int y, int choose());
    ~World();
};
```

Класс поля хранит в себе информацию о размере поля и массив клеток поля.

Для клеток поля был реализован класс Cell.

```
class Cell {
    bool isUnit;
    bool isWall;
    AbstractCell* cell;
public:
    bool isEmpty() const;
```

```
        ClosedCells *getClosedCell() const;
        Unit *getUnit() const;
        friend std::ostream &operator<<(std::ostream &out, const
Cell &cc);
        explicit Cell(bool isUnit = false, bool isWall = false);
        template<class ClosedClass>
        Cell &setWall();
        template<class UnitClass>
        Cell &setUnit();
        Cell &setPlayer(char playerName);
        Cell &delWall();
        Cell &delUnit();
        bool getIsWall() const;
        bool getIsUnit() const;
        void move(Cell& from, Cell& to);
    };
```

В классе хранится информация о свойствах клетки (стена, юнит или пусто), а также указатель на абстрактную непустую клетку class AbstractCell{};.

Для стен реализовано семейство классов.

```
class ClosedCells : public AbstractCell {
protected:
    char pict;
public:
    explicit ClosedCells(char pict);

    char getPict();
};

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

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

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

Для юнитов были написаны следующие классы.

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

```
class Unit : public AbstractCell {
protected:
    int health = 50;
    int damage;
    int armore;
    int exp = 0;
    int level = 1;
    int id;
    char pict;
public:
    explicit Unit(char pict);
    int getHealth() const;
    int getDamage() const;
    int getArmore() const;
    int getExp() const;
    int getLevel() const;
    char getName() const;
    int getID();
};
class Knight : public Unit {
public:
    Knight(char pict);
};
class Ranger : public Unit {
public:
    Ranger(char pict);
};
class Wizard : public Unit {
public:
    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:
    Player(char digit);
};
```

Как и предложено в задании есть 3 типа, и для каждого типа 2 вида. Также есть класс игрока, унаследованный от класса абстрактного юнита.

Вывод.

В ходе работы были получены навыки реализации классов с использованием наследования.