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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программировани»
Тема: Создание классов, конструкторов класса, методов класса, наследование

Студент гр. 8303 _____ Крыжановский К.Е.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы

Научиться создавать классы и их конструкторы, реализовать методы классов и познакомиться с наследованием классов.

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

Выводы

В ходе выполнения работы были созданы классы поля и юнитов и их конструкторы, реализовано наследование классов юнитов и методов классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:

```
Название файла: Peon.h
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_PEON_H
#define LAB1_PEON_H

#include "Alliance.h"

class Peon : public Alliance {

    const static int DEFAULT_SPEED = 3;
    const static int DEFAULT_HEALTH = 210;
    const static int DEFAULT_ARMOR = 10;
    const static int DEFAULT_DAMAGE = 15;

    char getId() override;

public:
    Peon();
};


#endif //LAB1_PEON_H


Название файла: Elf.h
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_ELF_H
#define LAB1_ELF_H


#include "Magicians.h"

class Elf : public Magicians {

    const static int DEFAULT_SPEED = 3;
    const static int DEFAULT_HEALTH = 180;
    const static int DEFAULT_ARMOR = 5;
    const static int DEFAULT_DAMAGE = 18;

    char getId() override;

public:
```

```
        Elf();
    };


    #endif //LAB1_ELF_H


    Название файла: GameField.cpp
    //
    // Created by therealyou on 11.02.2020.
    //
    /**
     * 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
     * */

    #include "GameField.h"

    GameField::GameField() {
        std::cout << "Creating default field" << std::endl;
        setWidth(DEFAULT_X);
        setHeight(DEFAULT_Y);
        setCountUnits();
        setMatrix();
    }

    GameField::GameField(int x, int y) {
        setWidth(x);
        setHeight(y);
        setCountUnits();
        setMatrix();
    }

    GameField::GameField(const GameField* gameField) {
        setWidth(gameField->width);
        setHeight(gameField->height);
        setCountUnits();
        setMatrix(gameField->matrix);
    }

    void GameField::setMatrix() {
        matrix = new baseType*[height];
        for (int i = 0; i < height; i++){
            matrix[i] = new baseType[width];
        }
    //    for (int i = 0; i < height; i++){
    //        for (int j = 0; j < width; j++){
    //            matrix[i][j].valueAsChar = DEFAULT_VALUE;
    //        }
    //    }
```

```
}

void GameField::setMatrix(baseType** matrix) {
    this->matrix = new baseType*[height];
    for (int i = 0; i < height; i++){
        this->matrix[i] = new baseType[width];
    }
    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            this->matrix[i][j] = matrix[i][j];
        }
    }
}

void GameField::setWidth(int width) {
    this->width = width;
}

void GameField::setHeight(int height) {
    this->height = height;
}

void GameField::setCountUnits() {
    this->maxCountUnits = height * width;
}


int GameField::addUnit(Unit* unit, int x, int y) {
    if (x < 0 || x >= width || y < 0 || y >= height){
        std::cerr << "\t\tWrong coords to add" << std::endl;
        return 1;
    }
    if (countUnits == maxCountUnits){
          std::cerr << "\t\tHas reached total count of units" <<
std::endl;
        return 2;
    }
    if (!unit->isCanAdded()){
           std::cerr << "\t\tThis unit has been added early" <<
std::endl;
        return 3;
    }
    if (matrix[y][x].valueAsChar != DEFAULT_VALUE){
        std::cerr << "\t\tThis place is not free" << std::endl;
        return 4;
    }
    unit->addUnit();
    unit->setX(x);
    unit->setY(y);
    unit->setGameField(this);
    matrix[y][x].valueAsChar = unit->getId();
    matrix[y][x].unit = unit;

    countUnits++;
//    std::cout << matrix[y][x] << std::endl;
//    std::cout << matrix[y][x] << std::endl;
```

```
//    this->printField();
    return 0;
}

void GameField::deleteUnit(int x, int y) {
    if (x < 0 || x >= width || y < 0 || y >= height){
        std::cerr << "\t\tWrong coords to delete" << std::endl;
        return;
    }
    if (matrix[y][x].valueAsChar != DEFAULT_VALUE){
        matrix[y][x].valueAsChar = DEFAULT_VALUE;
        countUnits--;
    } else {
        std::cerr << "\t\tThis place is free" << std::endl;
        return;
    }
}

void GameField::printField() {
    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            std::cout << matrix[i][j].valueAsChar << " ";
        }
        std::cout << std::endl;
    }
}

void GameField::move(Unit* unit, int dx, int dy) {
    unit->deleteUnit();
    int res = this->addUnit(unit, unit->getX() + dx, unit->getY()
+ dy);
    if (res != 0){
         std::cerr << "\t\tCan not move unit to another place" <<
std::endl;
        unit->addUnit();
        return;
    }
    this->deleteUnit(unit->getX() - dx, unit->getY() - dy);
}

/**
 * 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
 * */

Название файла: Witch.cpp
//
// Created by therealyou on 11.02.2020.
//
```

```
#include "Witch.h"

Witch::Witch()    :    Magicians(DEFAULT_SPEED,    DEFAULT_ARMOR,
DEFAULT_DAMAGE, DEFAULT_HEALTH){};


char Witch::getId() {
    return 'W';
}


Название файла: Unit.h
//
// Created by therealyou on 11.02.2020.
//
/**
  * 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
  * */
#ifndef LAB1_UNIT_H
#define LAB1_UNIT_H
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <ctime>
#include "GameField.h"

#define SCATTER 0.0 // разброс [0, 1) | элемент случайности

class GameField;

class Unit {
protected:
    GameField* gameField = nullptr;

    // coords
    int x;
    int y;

    // attributes
    int speed;
    double health;
    double armor;
    double damage;

    bool canAdded = true;
```

```
    double deltaAttribute(double attribute);
    double minAttribute(double attribute);

public:
    Unit(int speed, int armor, int damage, int health);

    virtual char getId() = 0;

    void addUnit();
    void deleteUnit();

    bool isCanAdded() const;

    double getSpeed() const;
    double getHealth() const;
    double getArmor() const;
    double getDamage() const;

    void setSpeed(int speed);
    void setHealth(int health);
    void setArmor(int armor);
    void setDamage(int damage);

    GameField *getGameField() const;

    void setGameField(GameField *gameField);

    int getX() const;

    void setX(int x);

    int getY() const;

    void setY(int y);

    void print();

    void move(int dx, int dy);
};


#endif //LAB1_UNIT_H


Название файла: Goblin.h
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_GOBLIN_H
#define LAB1_GOBLIN_H


#include "Orcs.h"
```

```
class Goblin : public Orcs {

    const static int DEFAULT_SPEED = 4;
    const static int DEFAULT_HEALTH = 190;
    const static int DEFAULT_ARMOR = 12;
    const static int DEFAULT_DAMAGE = 16;

    char getId() override;

public:
    Goblin();

};


#endif //LAB1_GOBLIN_H
```

Название файла: Orcs.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_ORCS_H
#define LAB1_ORCS_H


#include "Unit.h"

class Orcs : public Unit {
public:
    Orcs(int speed, int armor, int damage, int health);
};


#endif //LAB1_ORCS_H
```

Название файла: GameField.h

```
//
// Created by therealyou on 11.02.2020.
//
/**
 * 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
 * */

#ifndef LAB1_GAMEFIELD_H
#define LAB1_GAMEFIELD_H
```

```
#define DEFAULT_X 4
#define DEFAULT_Y 5
#define DEFAULT_VALUE '-'

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

struct Cell{
    void *unit;
    char valueAsChar = DEFAULT_VALUE;
};

typedef Cell baseType;

class Unit;

class GameField {
private:
    baseType** matrix;
    int width;
    int height;
    int countUnits = 0;
    int maxCountUnits;

public:

    GameField();

    GameField(int x, int y);

    GameField(const GameField* gameField);

    void setCountUnits();

    void printField();

    void setMatrix();

    void setMatrix(baseType** matrix);

    void setWidth(int width);

    void setHeight(int height);

    int addUnit(Unit* unit, int x, int y);

    void deleteUnit(int x, int y);

    void move(Unit* unit, int dx, int dy);
};


#endif //LAB1_GAMEFIELD_H
```

Название файла: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(lab1)

set(CMAKE_CXX_STANDARD 17)

add_executable(lab1 main.cpp GameField.cpp GameField.h Unit.cpp Unit.h Alliance.cpp Alliance.h Peon.cpp Peon.h Shooter.cpp Shooter.h Orcs.cpp Orcs.h Troll.cpp Troll.h Goblin.cpp Goblin.h Magicians.cpp Magicians.h Elf.cpp Elf.h Witch.cpp Witch.h)
```

Название файла: Shooter.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_SHOOTER_H
#define LAB1_SHOOTER_H

#include "Alliance.h"

class Shooter : public Alliance {

    const static int DEFAULT_SPEED = 4;
    const static int DEFAULT_HEALTH = 280;
    const static int DEFAULT_ARMOR = 7;
    const static int DEFAULT_DAMAGE = 20;

    char getId() override;

public:
    Shooter();
};


#endif //LAB1_SHOOTER_H
```

Название файла: Troll.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_TROLL_H
#define LAB1_TROLL_H


#include "Orcs.h"


class Troll : public Orcs {

    const static int DEFAULT_SPEED = 5;
    const static int DEFAULT_HEALTH = 220;
    const static int DEFAULT_ARMOR = 8;
    const static int DEFAULT_DAMAGE = 13;
```

```
    char getId() override;

public:
    Troll();
};


#endif //LAB1_TROLL_H
```

Название файла: Unit.cpp

```
//
// Created by therealyou on 11.02.2020.
//

#include "Unit.h"

int Unit::getX() const {
    return x;
}

void Unit::setX(int x) {
    this->x = x;
}

int Unit::getY() const {
    return y;
}

void Unit::setY(int y) {
    this->y = y;
}

void Unit::setGameField(GameField *gameField) {
    this->gameField = gameField;
}

double Unit::minAttribute(double attribute) {
    return attribute - deltaAttribute(attribute);
}

double Unit::deltaAttribute(double attribute) {
    return attribute * SCATTER;
}

double Unit::getSpeed() const {
    return speed;
}

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

double Unit::getArmor() const {
    return armor;
}
```

```
double Unit::getDamage() const {
    return damage;
}

GameField *Unit::getGameField() const {
    return gameField;
}

void Unit::move(int dx, int dy) {
    this->canAdded = true;
    if (this->getGameField()->addUnit(this, x + dx, y + dy)){
         std::cerr << "\t\tCan not move unit to another place" <<
std::endl;
        this->canAdded = false;
        return;
    }
    this->getGameField()->deleteUnit(x - dx, y - dy);
//    x += dx;
//    y += dy;
}

void Unit::addUnit() {
    this->canAdded = false;
}

void Unit::deleteUnit() {
    this->canAdded = true;
}

bool Unit::isCanAdded() const {
    return canAdded;
}

void Unit::print() {
    char id = this->getId();
    char *name = (char*) malloc(10 * sizeof(char));
    switch (id){
        case 'P':
            strcpy(name, "Peon");
            break;
        case 'S':
            strcpy(name, "Shooter");
            break;
        case 'E':
            strcpy(name, "Elf");
            break;
        case 'W':
            strcpy(name, "Witch");
            break;
        case 'T':
            strcpy(name, "Troll");
            break;
        case 'G':
            strcpy(name, "Goblin");
            break;
```

```
        }
        std::cout << "[" << name << "]" << std::endl <<
        "Armor: " << this->getArmor() << std::endl <<
        "Speed: " << this->getSpeed() << std::endl <<
        "Damage: " << this->getDamage() << std::endl <<
        "Health: " << this->getHealth() << std::endl <<

        "Current position: " << std::endl <<
        "X: " << this->getX() << std::endl <<
        "Y: " << this->getY() << std::endl <<

        std::endl << std::endl;
    }


    void Unit::setDamage(int damage) {
        std::srand(unsigned(std::time(0)));
        double min = minAttribute(damage);
                    this->damage    =    min    +    static_cast<double
>(std::rand())/RAND_MAX * 2 * deltaAttribute(min);
    }

    void Unit::setArmor(int armor) {
        std::srand(unsigned(std::time(0)));
        double min = minAttribute(armor);
                     this->armor    =    min    +    static_cast<double
>(std::rand())/RAND_MAX * 2 * deltaAttribute(min);
    }

    void Unit::setHealth(int health) {
        std::srand(unsigned(std::time(0)));
        double min = minAttribute(health);
                    this->health    =    min    +    static_cast<double
>(std::rand())/RAND_MAX * 2 * deltaAttribute(min);
    }

    void Unit::setSpeed(int speed) {
        std::srand(unsigned(std::time(0)));
        double min = minAttribute(speed);
                     this->speed    =    min    +    static_cast<double
>(std::rand())/RAND_MAX * 2 * deltaAttribute(min);
    }

    Unit::Unit(int speed, int armor, int damage, int health) {
        setArmor(armor);
        setDamage(damage);
        setHealth(health);
        setSpeed(speed);
    }


    Название файла: Magicians.cpp
    //
    // Created by therealyou on 11.02.2020.
    //
```

```
#include "Magicians.h"

Magicians::Magicians(int speed, int armor, int damage, int health) : Unit(speed, armor, damage, health) {

}
```

Название файла: Peon.cpp

```
//
// Created by therealyou on 11.02.2020.
//

#include "Peon.h"

Peon::Peon() : Alliance(DEFAULT_SPEED, DEFAULT_ARMOR, DEFAULT_DAMAGE, DEFAULT_HEALTH){};

char Peon::getId() {
    return 'P';
}
```

Название файла: Witch.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_WITCH_H
#define LAB1_WITCH_H


#include "Magicians.h"

class Witch : public Magicians{

    const static int DEFAULT_SPEED = 4;
    const static int DEFAULT_HEALTH = 150;
    const static int DEFAULT_ARMOR = 3;
    const static int DEFAULT_DAMAGE = 25;

    char getId() override;

public:
    Witch();

};

#endif //LAB1_WITCH_H
```

Название файла: Makefile

```
CC=g++
```

```
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp Alliance.cpp GameField.cpp Magicians.cpp Orcs.cpp Shooter.cpp Unit.cpp Elf.cpp Goblin.cpp Peon.cpp Troll.cpp Witch.cpp

OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=game.out

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf *.o
```

Название файла: Orcs.cpp

```
//
// Created by therealyou on 11.02.2020.
//

#include "Orcs.h"

Orcs::Orcs(int speed, int armor, int damage, int health) : Unit(speed, armor, damage, health) {}
```

Название файла: result.txt

Название файла: Peon.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_PEON_H
#define LAB1_PEON_H

#include "Alliance.h"

class Peon : public Alliance {

    const static int DEFAULT_SPEED = 3;
    const static int DEFAULT_HEALTH = 210;
    const static int DEFAULT_ARMOR = 10;
    const static int DEFAULT_DAMAGE = 15;

    char getId() override;

public:
    Peon();
};
```

```
#endif //LAB1_PEON_H
```

Название файла: game.out

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include "GameField.h"
#include "Peon.h"
#include "Shooter.h"
#include "Elf.h"
#include "Witch.h"
#include "Goblin.h"
#include "Troll.h"

/**
 * 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
 */


int main() {
    GameField* gameField = new GameField(10, 4);

    Peon* peon1 = new Peon();
    gameField->addUnit(peon1, 1, 1);
    gameField->printField();
    std::cout << std::endl;

    Peon* peon2 = new Peon();
    gameField->addUnit(peon2, 1, 0);
    gameField->printField();
    std::cout << std::endl;
```

```
    Shooter* shooter1 = new Shooter();
    gameField->addUnit(shooter1, 2, 0);
    gameField->printField();
    std::cout << std::endl;

    Elf* elf1 = new Elf();
    gameField->addUnit(elf1, 3, 0);
    gameField->printField();
    std::cout << std::endl;

    Witch* witch1 = new Witch();
    gameField->addUnit(witch1, 4, 0);
    gameField->printField();
    std::cout << std::endl;

    Troll* troll1 = new Troll();
    gameField->addUnit(troll1, 5, 0);
    gameField->printField();
    std::cout << std::endl;

    std::cout << "Add goblin" << std::endl;
    Goblin* goblin1 = new Goblin();
    gameField->addUnit(goblin1, 6, 0);
    gameField->printField();
    std::cout << std::endl;

    goblin1->print();

    std::cout << "Move goblin" << std::endl;

//    gameField->move(goblin1, 1, 1);
    goblin1->move(-1, 1);
    gameField->printField();


    std::cout << std::endl;
    goblin1->print();

//    shooter1->print();
//    peon1->print();
//
//    troll1->print();

//    elf1->print();
//    witch1->print();

    return 0;
}


Название файла: Shooter.cpp
//
// Created by therealyou on 11.02.2020.
//
```

```
#include "Shooter.h"

Shooter::Shooter()   :   Alliance(DEFAULT_SPEED,   DEFAULT_ARMOR,
DEFAULT_DAMAGE, DEFAULT_HEALTH){};

char Shooter::getId() {
    return 'S';
}


Название файла: Alliance.cpp
//
// Created by therealyou on 11.02.2020.
//

#include "Alliance.h"

Alliance::Alliance(int   speed,   int   armor,   int   damage,   int
health) : Unit(speed, armor, damage, health) {

}


Название файла: Elf.cpp
//
// Created by therealyou on 11.02.2020.
//

#include "Elf.h"

Elf::Elf()     :     Magicians(DEFAULT_SPEED,     DEFAULT_ARMOR,
DEFAULT_DAMAGE, DEFAULT_HEALTH){};

char Elf::getId() {
    return 'E';
}


Название файла: Troll.cpp
//
// Created by therealyou on 11.02.2020.
//

#include "Troll.h"

Troll::Troll()     :     Orcs(DEFAULT_SPEED,     DEFAULT_ARMOR,
DEFAULT_DAMAGE, DEFAULT_HEALTH){};


char Troll::getId() {
    return 'T';
}


Название файла: Magicians.h
//
```

```
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_MAGICIANS_H
#define LAB1_MAGICIANS_H


#include "Unit.h"

class Magicians : public Unit {
public:
    Magicians(int speed, int armor, int damage, int health);
};


#endif //LAB1_MAGICIANS_H
```

Название файла: Alliance.h

```
//
// Created by therealyou on 11.02.2020.
//

#ifndef LAB1_ALLIANCE_H
#define LAB1_ALLIANCE_H


#include "Unit.h"

class Alliance : public Unit {
public:
    Alliance(int speed, int armor, int damage, int health);
};


#endif //LAB1_ALLIANCE_H
```

Название файла: Goblin.cpp

```
//
// Created by therealyou on 11.02.2020.
//

#include "Goblin.h"

Goblin::Goblin()     :     Orcs(DEFAULT_SPEED,     DEFAULT_ARMOR,
DEFAULT_DAMAGE, DEFAULT_HEALTH){};


char Goblin::getId() {
    return 'G';
}
```