**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Шаблонные классы, управление

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Калмак Д.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

## **Цель работы.**

Создать классы с набором правил и класс игры, параметризируемый правилами.

## **Задание.**

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и.т.д.). Затем определить класс игры, которое параметризуется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

* Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
* Создан шаблонный класс игры, который параметризуется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и.т.д.

Потенциальные паттерны проектирования, которые можно использовать:

* Компоновщик (Composite) - выстраивание иерарихии правил
* Фасад (Facade) - предоставления единого интерфейса игры для команд управления
* Цепочка обязанностей (Chain  of Responsibility) - обработка поступающих команд управления
* Состояние (State) - отслеживание состояние хода / передача хода от игрока к врагам

Посредник (Mediator) - организация взаимодействия элементов бизнес-логики

## **Выполнение работы.**

Создан класс правил Rules. Это шаблонный класс, который принимает в шаблон количество врагов – поле numberofenemies, которые будут расположены на игровом поле и участвовать в игровом процессе. Шаблон принимает количество предметов – поле numberofitems, которые будут расположены на игровом поле и с которыми будет взаимодействовать игрок.

Создан класс игры Game. Это шаблонный класс, который устанавливает игровое поле, управляет движением врагов и отвечает за игровой процесс. Класс принимает правила игры – поле R<numberofenemies, numberofitems> rules, которые параметризируют игру. Поле Field\* f – указатель на игровое поле, на котором будет происходить игровой процесс.

Метод set\_fieldofgame() – метод устанавливает игровое поле, которое получает на вход.

Метод start() – метод начинает игровой процесс. Созданы два вектора - vector<*Object\_of\_game* \*> items и vector<*Object\_of\_game* \*> enemies, которые хранят предметы и врагов. В зависимости от количества предметов на поле, определяемым правилами, в вектор добавляются предметы, такие как Аптечка, Оружие, Броня, в случае больше трех предметов – остальные Аптечки. В зависимости от количества врагов на поле, определяемым правилами, в вектор добавляются враги, такие как Bolg, Smaug, Nazgul, в случае больше трех врагов – остальные Bolg. На место входа размещается игрок. К игроку добавляется обсервер. С помощью метода set\_objectsofgame() размещаются предметы и враги на свободные места. Запускается цикл, который будет завершаться только когда игрок дойдет до выхода или потеряет все хп. Выводится состояние поля, происходит с помощью метода move\_objectofenemies() движение врагов. Создан класс Coordinates, у которого два поля – x и y. В методе move\_objectofenemies() создан вектор vector<*Coordinates*> Enemy. С помощью цикла находятся враги на поле и вектор записывается их расположения. Для каждого врага в векторе вызывается метод move(), который передвигает врага в соседнюю клетку.

Разработанный код см. в Приложении А.

Разработанную диаграмму классов UML см. в Приложении Б.

## **Тестирование.**

Результаты тестирования представлены в Таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 – Результаты тестирования | | | |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | 10 10 | p 0 0 b 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 m 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  p 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 m 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 b  0 0 0 0 0 0 0 0 0 0 | Все верно. |

|  |
| --- |
|  |

## **Выводы.**

Таким образом, был реализован класс правил и класс игры.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Rules.h:**

#ifndef RULES\_H

#define RULES\_H

template<int *enemy*, int *item*>

class Rules{

    public:

        int numberofenemies = enemy;

        int numberofitems = item;

    };

#endif

**Файл Game.h:**

#ifndef GAME\_H

#define GAME\_H

#include "Rules.h"

#include "vector"

#include "field\_View.h"

#include "Logger/Logger\_console.h"

#include "Logger/Logger\_file.h"

template <template<int, int> class R, int *numberofenemies*, int *numberofitems*>

class Game{

    private:

        R<numberofenemies, numberofitems> rules;

*Field*\* f;

        void move\_objectofenemies(){

            vector<*Coordinates*> Enemy;

            for (int i = 0; i < f->n; i++){

                for (int j = 0; j < f->m; j++){

                    if ((f->arr[i][j].get\_objectofgame()) && ((typeid((\*f->arr[i][j].get\_objectofgame())) == typeid(*Bolg*)) || (typeid((\*f->arr[i][j].get\_objectofgame())) == typeid(*Smaug*)) || (typeid((\*f->arr[i][j].get\_objectofgame())) == typeid(*Nazgul*)))){

                        Enemy.push\_back(*Coordinates*(i, j));

                    }

                }

            }

            for (auto a: Enemy){

                move(a);

            }

        }

        bool move(*Coordinates* *en*){

            int x = *en*.x;

            int y = *en*.y;

            if (y+1 < f->n) {

                if (f->arr[y][x].move\_objectofgame(&(f->arr[y+1][x])))

                    return true;

            }

            if (x+1 < f->m) {

                if (f->arr[y][x].move\_objectofgame(&(f->arr[y][x+1])))

                    return true;

            }

            if (0 <= y-1) {

                if (f->arr[y][x].move\_objectofgame(&(f->arr[y-1][x])))

                    return true;

            }

            if (0 <= x-1) {

                if (f->arr[y][x].move\_objectofgame(&(f->arr[y][x-1])))

                    return true;

            }

            return false;

        }

    public:

        void set\_fieldofgame(*Field*\* *other*){

            f = *other*;

        }

        void start(){

            vector<*Object\_of\_game* \*> items;

            vector<*Object\_of\_game* \*> enemies;

            switch (numberofitems){

                case 1:

                    items.push\_back(**new** *Medicine*(20));

                    break;

                case 2:

                    items.push\_back(**new** *Medicine*(20));

                    items.push\_back(**new** *Weapon*(10));

                    break;

                case 3:

                    items.push\_back(**new** *Medicine*(20));

                    items.push\_back(**new** *Weapon*(10));

                    items.push\_back(**new** *Armor*(20));

                    break;

                default:

                    items.push\_back(**new** *Weapon*(10));

                    items.push\_back(**new** *Armor*(20));

                    for (int i = 3; i < numberofitems; i++)

                        items.push\_back(**new** *Medicine*(20));

            }

            switch (numberofenemies){

                case 1:

                    enemies.push\_back(**new** *Bolg*);

                    break;

                case 2:

                    enemies.push\_back(**new** *Bolg*);

                    enemies.push\_back(**new** *Smaug*);

                    break;

                case 3:

                    enemies.push\_back(**new** *Bolg*);

                    enemies.push\_back(**new** *Smaug*);

                    enemies.push\_back(**new** *Nazgul*);

                    break;

                default:

                    enemies.push\_back(**new** *Bolg*);

                    enemies.push\_back(**new** *Smaug*);

                    for (int i = 3; i < numberofenemies; i++)

                        enemies.push\_back(**new** *Bolg*);

            }

*Player*\* p = **new** *Player*();

            p->add\_observer\_choose("console", "P");

            f->arr[0][0].set\_objectofgame(p);

int x\_exit;

            int y\_exit;

            for (int i = 0; i < f->n; i++){

                for (int j = 0; j < f->m; j++){

                    if (f->arr[i][j].get\_condition() == "e "){

                        x\_exit = j;

                        y\_exit = i;

                    }

                }

            }

*field\_View* field\_for\_view;

            f->set\_objectsofgame(items);

            f->set\_objectsofgame(enemies);

            while (!(xp == x\_exit && yp == y\_exit) && p->get\_hp() > 0

){

                field\_for\_view.print\_F(\*f);

                cout << endl;

                move\_objectofenemies();

            }

        }

};

#endif

# **ПРИЛОЖЕНИЕ Б UML ДИАГРАММА КЛАССОВ**

