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. 0383: Бояркин Н.А.

Преподаватель: Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Научиться работать с шаблонными классами и реализовать шаблонный класс правил игры и класс самой игры, где будут производиться действия с объектами.

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.

Требования:

- 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 иерархии 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:

1) Так как класс *Game* был реализован во второй лабораторной работе, нужно было сделать его шаблонным. *Game* это шаблонный класс с переменным количеством параметров, то есть в данном случае он может принимать на вход 0 или более шаблонных классов правил игры.

2) Для проверки выполнил ли игрок определенные правила игры был создан класс *GameStats*, который собирает статистику игрока, а именно: сколько было убито врагов (хранится в поле *int enemy_killed*), сколько сделал шагов пользователь (хранится в поле *int steps_made*). Также были разработаны функции *void setEnemyKilled(int number)* и *void setStepsMade()*, которые позволяют установить значения для *int enemy_killed* и *int steps_made*, и *int getEnemyKilled()* const и *int getStepsMade() const*, чтобы получать их значения.
3) Класс *Rules*: шаблонный класс правил игры. В данном случае параметр шаблона принимает на вход *int EnemiesDied*, а именно число врагов, которое необходимо убить, чтобы выйти из игры. В этом классе описан метод *bool EnemiesDiedChecker(GameStats &stats)*, в котором происходит проверка, выполняется ли правило. Аналогично был создан класс *Rules_second*, где прописано правило: сделать определенное количество шагов.

4) Был создан метод *void isGamePassed(WinChecker... winChecker)*, в котором происходит распаковка экземпляров шаблонных классов правил игры и к каждому из них применяется метод *bool Check(GameStats &stats)*. Соответственно, если правило выполнено и игрок жив, то вызывается функция *void EndGame()*.
5) Инициализация всех объектов и реализованная логика передачи хода, управление врагами и передача информации куда переместить игрока описана в функции *void StartGame()*. Также здесь создаётся логгер.
6) Также класс Game имеет поле quit типа bool, который по умолчанию равен false, функция *void EndGame()* присваивает значение полю quit true. После созданий экземпляров класса Field, WinChecker, FieldView, LoggerPull происходит цикл while, который завершается когда поле quit равен true, далее происходит управление игроком и врагами, тут же проводится проверка, изменились ли отслеживаемые логгером объекты.

Разработанный программный код см. в приложении А.

Диаграмму с зависимостями классов см. в приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	Результат	Комментарии
1.	Есть возможность многократно ходить, при этом враги тоже двигаются.		Верно
2.	Игру можно пройти, убив необходимое количество врагов и сделав необходимое кол-во шагов дойдя живым до финиша.		Верно
3.	Если герой не убил достаточно врагов и/или не сделал необходимое кол-во шагов, но пришёл на финиш, то игра не заканчивается		Верно
4.	При смерти героя, игра будет окончена.		Верно

Выводы.

В ходе работы были изучены шаблонные классы и шаблонные методы. Было создано два шаблонных класса, класс правил и класс игры. Игру можно полностью пройти, выполнив ряд задач.

Приложение А.

Исходный код программы.

Название файла: Game.h

```
#pragma once
#include "Field.h"
#include "View/FieldView.h"
#include "../Logger/Logger.h"
#include "../Logger/ConsoleLogger.h"
#include "../Logger/LoggerPull.h"
#include "../Rules/Rules.h"
#include "../Rules/Rules_second.h"
#include "GameStats.h"


template<class... WinChecker>
class Game{
private:
    bool quit = false;  // флаг для выхода из игры
    int countEnemiesDied = 0;
    bool isWinner = false;
    GameStats gameStats;
protected:
    void EndGame();
    void isGamePassed(WinChecker... winChecker);
public:
    void StartGame();
};

template<class... WinChecker>
void Game<WinChecker...>::isGamePassed(WinChecker...
winChecker) {
        if ((winChecker.Check(gameStats) && ...) && isWinner){
//  Если умерли все враги, то игра завершается
winChecker.EnemiesDiedChecker()
            EndGame();
        }
}

template<class... WinChecker>
void Game<WinChecker...>::EndGame() {
    this->quit = true;
}

template<class... WinChecker>
void Game<WinChecker...>::StartGame() {
    int width = 2;
    int height = 2;
```

```
        std::cout << "Enter width: \n";
        std::cin >> width;
        std::cout << "Enter height: \n";
        std::cin >> height;
        Field field(width, height);
        //GameStats gameStats;
        //    WinChecker rules;  // 5 врагов должны быть умереть
        int mode = 0;
         std::cout << "Enter mode: 0 - logging only in console, 1
- logging only in file, 2 - logging in console and file.\n";
        std::cin >> mode;
        std::cout << '\n';
        FieldView fieldView(field);
    //    fieldView.printField();
        fieldView.printFieldObj();
        std::cout << "\n";
    //                            MainCharacter      player      =
dynamic_cast<MainCharacter&>(*(field.hero->getEntity()));
               LoggerPull   logger(mode,   field.hero->getEntity(),
field.arr_enemies);
        int iter = 0;
        while (!this->quit){
                                      MainCharacter    player    =
dynamic_cast<MainCharacter&>(*(field.hero->getEntity()));
            char command;
            std::cout << "Enter command: ";
            std::cin >> command;
            std::cout << std::endl;
            switch (command) {
                case 'Q':
                    EndGame();
                    break;
                case 'W':
                    field.moveEntity(*field.hero, 'W', 0);
                    gameStats.setStepsMade();
                                                                    if
(!dynamic_cast<MainCharacter&>(*(field.hero->getEntity())).isAlive
()) //  or field.HeroWin()
                        EndGame();
                    break;
                case 'A':
                    gameStats.setStepsMade();
                    field.moveEntity(*field.hero, 'A', 0);
                                                                    if
(!dynamic_cast<MainCharacter&>(*(field.hero->getEntity())).isAlive
()) // or field.HeroWin()
                        EndGame();
                    break;
                case 'D':
                    gameStats.setStepsMade();
                    field.moveEntity(*field.hero, 'D', 0);
```

```
                                                                    if
(!dynamic_cast<MainCharacter&>(*(field.hero->getEntity()))).isAlive
()) // or field.HeroWin()
                        EndGame();
                    break;
                case 'S':
                    gameStats.setStepsMade();
                    field.moveEntity(*field.hero, 'S', 0);
                                                                    if
(!dynamic_cast<MainCharacter&>(*(field.hero->getEntity()))).isAlive
()) //  or field.HeroWin()
                        EndGame();
                    break;
                case 'I':
                                                          player   =
dynamic_cast<MainCharacter&>(*(field.hero->entity));
                         std::cout << "Info hero: \nPower: " <<
player.getPower() << std::endl;
                     std::cout << "Health: " << player.getHealth()
<< std::endl;
                       std::cout << "Armor: " << player.getArmor()
<< std::endl;
     //                          std::cout << "Health (*_*): " <<
consoleLogger.player[TypeItems::HEALTH] << std::endl;
     //                consoleLogger.writeToConsole();
                                                                    if
(!dynamic_cast<MainCharacter&>(*(field.hero->getEntity()))).isAlive
())  // умерает не сразу
                        EndGame();
                    break;
                default:
                    break;
            }
             std::cout << "steps: " << gameStats.getStepsMade() <<
"\n";
            countEnemiesDied = field.getCountEnemiesDied();
            gameStats.setEnemyKilled(countEnemiesDied);
     //        gameStats.setStepsMade(c); // временно!!!
            isWinner = field.HeroWin();
            Rules<1> rule;  // !!!
            Rules_second<30> rulesSecond;

            isGamePassed(rule, rulesSecond);
     //        flag = (CheckRules(winChecker) & ...);
     //                                          (winChecker     &
...).EnemyKilled(field.getCountEnemiesDied());   // сколько умерло
врагов
     //           if ((winChecker & ...).EnemiesDiedChecker() and
field.HeroWin()){   // Если умерли все враги, то игра завершается
winChecker.EnemiesDiedChecker()
     //            EndGame();
```

```
//         }
```

```
        char ZDir, GDir, MDir;
        if (iter % 12 == 0) {
            ZDir = 'W';
            GDir = 'A';
            MDir = 'D';
        }
        if (iter % 12 == 1) {
            ZDir = 'A';
            GDir = 'A';
            MDir = 'D';
        }
        if (iter % 12 == 2) {
            ZDir = 'S';
            GDir = 'W';
            MDir = 'D';
        }
        if (iter % 12 == 3) {
            ZDir = 'D';
            GDir = 'W';
            MDir = 'W';
        }
        if (iter % 12 == 4) {
            ZDir = 'W';
            GDir = 'D';
            MDir = 'W';
        }
        if (iter % 12 == 5) {
            ZDir = 'A';
            GDir = 'D';
            MDir = 'W';
        }
        if (iter % 12 == 6) {
            ZDir = 'S';
            GDir = 'S';
            MDir = 'A';
        }
        if (iter % 12 == 7) {
            ZDir = 'D';
            GDir = 'S';
            MDir = 'A';
```

```
            }
            if (iter % 12 == 8) {
                ZDir = 'W';
                GDir = 'A';
                MDir = 'A';
            }
            if (iter % 12 == 9) {
                ZDir = 'A';
                GDir = 'A';
                MDir = 'S';
            }
            if (iter % 12 == 10) {
                ZDir = 'S';
                GDir = 'S';
                MDir = 'S';
            }
            if (iter % 12 == 11) {
                ZDir = 'D';
                GDir = 'S';
                MDir = 'S';
            }
            for (int i = 0; i < field.getCountEnemies(); i++) {
                if (field.arr_enemies[i]->getEntity() != nullptr)
{
                                                                    if
(typeid(*field.arr_enemies[i]->entity).name()                       !=
typeid(MainCharacter).name()) {
                        if (i % 3 == 0) {

field.moveEntity(*field.arr_enemies[i], ZDir, i);
                        }
                        if (i % 3 == 1) {

field.moveEntity(*field.arr_enemies[i], GDir, i);
                        }
                        if (i % 3 == 2) {

field.moveEntity(*field.arr_enemies[i], MDir, i);
                        }
                                                auto&  t_enemy  =
dynamic_cast<Enemies&>(*field.arr_enemies[i]->getEntity());
                                          if  (t_enemy.getHealth()  !=
(*logger.mainLogger).enemy_stats[field.arr_enemies[i]->getEntity()
][0]){

logger.writeEnemy(field.arr_enemies[i]->getEntity());

(*logger.mainLogger).save_enemy(field.arr_enemies[i]->getEntity())
;
                        }
                    }
```

```
                }
            }
            iter++;
                                MainCharacter   hero_entity   =
dynamic_cast<MainCharacter&>(*field.hero->getEntity());
                               if   (hero_entity.getHealth()   !=
(*logger.mainLogger).player_stats[HP] or
                                    hero_entity.getPower()   !=
(*logger.mainLogger).player_stats[POWER] or
                                    hero_entity.getArmor()   !=
(*logger.mainLogger).player_stats[AP]){
                logger.writeHero();
                (*logger.mainLogger).save_main();
            }
    //        fieldView.printField();
            fieldView.printFieldObj();
            std::cout << "\n";
        }
    }
```

Название файла: Rules.h

```
#pragma once
#include "../Tools/GameStats.h"

template<int  EnemiesDied>   // 5 врагов должно умереть при
создании поля 10 на 10
class Rules{
public:
    bool Check(GameStats &stats);
};

template<int EnemiesDied>
bool Rules<EnemiesDied>::Check(GameStats &stats) {
    return EnemiesDied <= stats.getEnemyKilled();
}
```

Название файла: Rules.h

```
#pragma once
#include "../Tools/GameStats.h"

template<int Steps>
class Rules_second{
public:
    bool Check(GameStats &stats);
};
```

```
template<int Steps>
bool Rules_second<Steps>::Check(GameStats &stats) {
    return Steps <= stats.getStepsMade();
}
```

Приложение Б

UML-диаграмма классов