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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Логирование, перегрузка операций

Студент гр. 0383 _____ Бояркин Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить механизм работы логирования, перегрузки операторов.

Задание.

Необходимо проводить логирование того, что происходит во время игры.

Требования:

Реализован класс логгера, который будет получать объект, который необходимо отслеживать, и при изменении его состоянии записывать данную информацию.

Должна быть возможность записывания логов в файл, в консоль или одновременно в файл и консоль.

Должна быть возможность выбрать типа вывода логов

Все объекты должны логироваться через перегруженный оператор вывода в поток.

Должна соблюдаться идиома RAII

Выполнение работы.

1. Реализован класс Logger, который получает отслеживаемые объекты, а именно героя и врагов (тип *CageEntity** и *CageEntity*** соответственно). Далее он сохраняет исходные характеристики героя (реализовано это в *void save_main()*) и врагов (реализовано это в *void save_enemy(CageEntity* enemy_track)*

   Для сохранения характеристик героя и врагов были созданы поля *player_stats* и *enemy_stats* с помощью контейнера map. У *player_stats* в качестве ключа принимает перечисления *characteristics(HP, POWER, AP)*, *enemy_stats* в качестве ключа принимает указатель на то, что находится на клетки.
2. Далее в классе *Game* реализована проверка изменения состояния объекта (отдельно для героя и отдельно для врагов). И если объект изменился, то перезаписываем характеристики объекта и выводим.
3. Для обеспечения возможности записывания логов в файл, в консоль или одновременно в файл и консоль были реализованы два класса: ConsoleLogger и FileLogger, которые наследуются от Logger. Решение с наследованием позволяет сохранять возможность для добавления новых путей вывода логов (например, если захотим выводить логи на сервер). Они принимают указатель на Logger, чтобы иметь доступ к данным, которые отслеживаются в логгере. С помощью функции *void writeHero() const* выводится информация об изменении характеристик героя в консоль (у класса *ConsoleLogger*) и в файл (у класса *FileLogger*). (Формат вывода: Hero was: …, Hero now: …) Аналогично работает функция *void writeEnemy(CageEntity* enemy) const*, но в качестве аргумента она принимает указатель на *CageEntity*, который кастуется в *Enemies*. (Формат вывода: Enemy was: …, Enemy now: …). Также для соблюдения идиомы RAII

открытие файла происходит в конструкторе, а закрытие в деструкторе класса *FileLogger*.

4. У класса *MainCharacter* и *Enemies* перегружен оператор вывода в поток.
5. Чтобы не создавать несколько экземпляров логгеров для вывода сразу в несколько потоков было решено объединить их в классе *LoggerPull*, который хранит в себе экземпляры логгеров. Класс *LoggerPull* имеет поля *mainLogger*, чтобы иметь доступ к характеристикам объекта. В конструкторе создается экземпляр класса *Logger*, далее создаются экземпляры класса *ConsoleLogger* и *FileLogger* в зависимости от переменной *mode*, которую передали в качестве аргумента в конструктор *LoggerPull* (значение переменной *mode* определяется в классе Game). Также реализованы функции *void writeHero() const* и *void writeEnemy(CageEntity* enemy) const*, которые в зависимости от выбора записи логов записывает изменения в консоль или в файл. Также был реализован деструктор, который удаляет экземпляры классов ConsoleLogger и FileLogger.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Герой поднял предмет, увеличивающий здоровье на 100	Hero: Health was: 100 Damage was: 10 Armor was: 0 Hero now: Player info: Health: 200 Damage: 10 Armor: 0	Верно
2.	Герой ударил врага типа Monster	Enemy: Health was: 1000 Damage was: 50 Enemy now: Enemy info: Health: 990 Damage: 50 Hero: Health was: 100 Damage was: 10 Armor was: 0 Hero now: Player info: Health: 50 Damage: 10 Armor: 0	Верно

3.	Герой взял предмет, увеличивающий защиту на 30.	Hero: Health was: 100 Damage was: 10 Armor was: 0 Hero now: Player info: Health: 100 Damage: 10 Armor: 30	Верно

Выводы.

Был изучен механизм работы логирования, перегрузки операторов.

Приложение А

Исходный код программы

Название файла: Logger.h

```
#pragma once

#include <fstream>
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include "../Entity/CageEntity.h"
#include "../Entity/MovableCharacters/MainCharacter.h"
#include "../Entity/MovableCharacters/Enemies/Enemies.h"
#include "../Entity/MovableCharacters/Enemies/Zombie.h"
#include "../Entity/MovableCharacters/Enemies/Ghost.h"
#include "../Entity/MovableCharacters/Enemies/Monster.h"

enum characteristics{
    HP,
    POWER,
    AP
};

class Logger{
public:
    Logger() = default;
    Logger(CageEntity* hero, Cage** array_enemies);
    CageEntity* subscriber_hero = nullptr;
    std::map<characteristics, int> player_stats;
//    std::map<characteristics, int> enemy;
    std::map<CageEntity*, std::vector<int>> enemy_stats;
    //    explicit Logger(CageEntity* cage_entity);
//    Logger(CageEntity** pCageEntities);
    void save_main();
    void save_enemy(CageEntity* enemy_track);
//    virtual void write(const std::string &info);

};
```

Название файла: Logger.cpp

```
#include "Logger.h"

void Logger::save_main() {
    if (subscriber_hero){
                                        this->player_stats[HP]     =
(dynamic_cast<MainCharacter&>(*subscriber_hero)).getHealth();
                                        this->player_stats[AP]     =
(dynamic_cast<MainCharacter&>(*subscriber_hero)).getArmor();
                                    this->player_stats[POWER]     =
(dynamic_cast<MainCharacter&>(*subscriber_hero)).getPower();
    }
```

```
    }

    Logger::Logger(CageEntity *hero, Cage **array_enemies) {
        subscriber_hero = hero;
        save_main();
        for (int i = 0; array_enemies[i] != nullptr; ++i) {  // ?
                            enemy_stats.insert(std::pair<CageEntity*,
std::vector<int>>(array_enemies[i]->getEntity(), {0,0}));
            save_enemy(array_enemies[i]->getEntity());
        }
    }

    void Logger::save_enemy(CageEntity *enemy_track) {
        if (enemy_track){
                                    this->enemy_stats[enemy_track][0]    =
(dynamic_cast<Enemies&>(*enemy_track)).getHealth();
                                    this->enemy_stats[enemy_track][1]    =
(dynamic_cast<Enemies&>(*enemy_track)).getDamage();
    //                              this->enemy_stats[enemy_track][2]    =
(dynamic_cast<Enemies&>(*enemy_track)).getArmor();
        }
    }

    //Logger::Logger(CageEntity *cage_entity) {
    //                      if     (typeid(*cage_entity).name()     ==
typeid(MainCharacter).name()){
    //        subscriber = cage_entity;
    //
save_characteristics(dynamic_cast<MainCharacter*>(this->subscriber));
    //    }
    //}
```

Название файла: ConsoleLogger.h

```
#pragma once
#include "Logger.h"

class ConsoleLogger: public Logger{
public:
    Logger* logger;
    ConsoleLogger(Logger* logger);
    void writeHero() const;
    void writeEnemy(CageEntity* enemy) const;
};
```

Название файла: ConsoleLogger.cpp

```
#include "ConsoleLogger.h"
//
//ConsoleLogger::ConsoleLogger(CageEntity*   cage_entity,   Cage**
enemies){
//    subscriber = cage_entity;  // hero
//
save_characteristics(dynamic_cast<MainCharacter*>(this->subscriber));
```

```
//
//};
//
//void ConsoleLogger::writeToConsole() {
////    std::cout << "Old info about Player:\n";
////    std::cout << "Health: ";
////    std::cout << player[TypeItems::HEALTH] << "\n";
////    std::cout << "Armor: ";
////    std::cout << player[TypeItems::ARMOR] << "\n";
////    std::cout << "Damage: ";
////    std::cout << player[TypeItems::WEAPON] << "\n";
//                                                std::cout          <<
dynamic_cast<MainCharacter&>(*this->subscriber);  // реализовано только
для героя
//}
ConsoleLogger::ConsoleLogger(Logger *logger) {
    this->logger = logger;
}

void ConsoleLogger::writeHero() const {
    std::cout << "Hero: \n";
     std::cout << "Health was: " << (*logger).player_stats[HP] <<
"\nDamage was: " << (*logger).player_stats[POWER]
      << "\nArmor was: " << (*logger).player_stats[AP] << "\nHero
now: "
       << dynamic_cast<MainCharacter&>(*(*logger).subscriber_hero)
<< std::endl;
}

void ConsoleLogger::writeEnemy(CageEntity *enemy) const {
//    bool what_enemy = false;
//    Enemies enemy_t = dynamic_cast<Enemies&>(*enemy);
//          if   (typeid(dynamic_cast<Ghost&>(*enemy)).name()   ==
typeid(Ghost).name()){
//        auto t_enemy = (dynamic_cast<Ghost&>(*enemy));
//     } else if (typeid(dynamic_cast<Zombie&>(*enemy)).name() ==
typeid(Zombie).name()){
//        auto t_enemy = (dynamic_cast<Zombie&>(*enemy));
//    } else{
//        auto t_enemy = (dynamic_cast<Monster&>(*enemy));
//    }
    std::cout << "Enemy: \n";
                    std::cout    <<    "Health    was:    "    <<
(*logger).enemy_stats[enemy][HP]    <<    "\nDamage    was:    "    <<
(*logger).enemy_stats[enemy][POWER]
                                          <<   "\nEnemy   now:   "   <<
(dynamic_cast<Enemies&>(*enemy)) << std::endl;  // << enemy_t <<
}
```

Название файла: FielLogger.h

```
#pragma once
#include "Logger.h"

class FileLogger: public Logger{
```

```
private:
    std::ofstream file;
public:
    Logger* logger;
    FileLogger(Logger* logger);
    void writeHero();
    void writeEnemy(CageEntity* enemy);
    ~FileLogger();
//    explicit FileLogger(const std::string& filePath);
//    void write(const std::string &info);
//    ~FileLogger();
};
```

Название файла: FileLogger.cpp

```
#include "FileLogger.h"

FileLogger::FileLogger(Logger *logger) {
    this->logger = logger;
    file.open("log.txt");
}

void FileLogger::writeHero(){
    file << "Hero: \n";
       file << "Health was: " << (*logger).player_stats[HP] <<
"\nDamage was: " << (*logger).player_stats[POWER]
                  << "\nArmor was: " << (*logger).player_stats[AP] <<
"\nHero now: "
                                                                  <<
dynamic_cast<MainCharacter&>(*(*logger).subscriber_hero) << std::endl;
}

void FileLogger::writeEnemy(CageEntity* enemy){
    file << "Enemy: \n";
     file << "Health was: " << (*logger).enemy_stats[enemy][HP] <<
"\nDamage was: " << (*logger).enemy_stats[enemy][POWER]
                                          <<  "\nEnemy  now:  "  <<
(dynamic_cast<Enemies&>(*enemy)) << std::endl;  // << enemy_t <<
}

FileLogger::~FileLogger() {
    file.close();
}
```

Название файла: LoggerPull.h

```
#pragma once
#include "ConsoleLogger.h"
#include "FileLogger.h"

class LoggerPull{
public:
    int mode;  // не знаю как сделать иначе
    Logger* mainLogger;
    ConsoleLogger* consoleLogger;
```

```
    FileLogger* fileLogger;
    LoggerPull() = default;
    LoggerPull(int mode, CageEntity* hero, Cage** enemies);
    void writeHero() const;
    void writeEnemy(CageEntity* enemy) const;
    ~LoggerPull();
};
```

Название файла: LoggerPull.cpp

```
#include "LoggerPull.h"

LoggerPull::LoggerPull(int mode, CageEntity *hero, Cage **enemies)
{
    this->mode = mode;
    mainLogger = new Logger(hero, enemies);
    if (mode == 0 or mode == 2){
        consoleLogger = new ConsoleLogger(mainLogger);
    }
    if (mode == 1 or mode == 2){
        fileLogger = new FileLogger(mainLogger);
    }
}

void LoggerPull::writeHero() const {
    if (mode == 0 || mode == 2) {
        consoleLogger->writeHero();
    }
    if (mode == 1 || mode == 2) {
        fileLogger->writeHero();
    }
}


void LoggerPull::writeEnemy(CageEntity* enemy) const {
    if (mode == 0 || mode == 2) {
        consoleLogger->writeEnemy(enemy);
    }
    if (mode == 1 || mode == 2) {
        fileLogger->writeEnemy(enemy);
    }
}

LoggerPull::~LoggerPull() {
    delete mainLogger;
    if (mode == 0 || mode == 2) {
        delete consoleLogger;
    }
    if (mode == 1 || mode == 2) {
        delete fileLogger;
    }
}
```

Приложение Б

UML-Диаграмма