**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. 1384 |  | Усачева Д.В. |
| Преподаватель |  | Жангиров Т.Р. |
|  |  |  |

Санкт-Петербург

2022

**Цель работы.**

Изучить и освоить на практике логирование. Научиться перегружать операторы, а также грамотно расставлять логи для качественного отслеживания ошибок.

**Задание.**

Реализовать класс/набор классов отслеживающих изменения состояний в программе. Отслеживание должно быть 3-х уровней:

1. Изменения состояния игрока и поля, а также срабатывание событий

2. Состояние игры (игра начата, завершена, сохранена, и.т.д.)

3. Отслеживание критических состояний и ошибок (поле инициализировано с отрицательными размерами, игрок попытался перейти на

непроходимую клетку, и.т.д.)

Реализованы классы для вывода информации разных уровней для в консоль и в файл с перегруженным оператором вывода в поток.

# Требования.

* Разработан класс/набор классов отслеживающий изменения разных уровней
* Разработаны классы для вывода в консоль и файл с соблюдением идиомы RAII и перегруженным оператором вывода в поток.
* Разработанные классы спроектированы таким образом, чтобы можно было добавить новый формат вывода без изменения старого кода (например, добавить возможность отправки логов по сети)
* Выбор отслеживаемых уровней логирования должен происходить в runtime
* В runtime должен выбираться способ вывода логов (нет логирования, в консоль, в файл, в консоль и файл)
* или не непроходимыми
* Игрок в гарантированно имеет возможность дойти до выхода

# **Примечания**:

* Отслеживаемые сущности не должны ничего знать о сущностях, которые их логируют.
* Уровни логирования должны быть заданными отдельными классами или перечислением.
* Разные уровни в логах должны помечаться своим префиксом
* Рекомендуется сделать класс сообщения
* Для отслеживания изменений можно использовать наблюдателя
* Для вывода сообщений можно использовать адаптер, прокси и декоратор

**Выполнение работы.**

В проекте для логирования используется паттерн Наблюдатель. В случае логирования изменений поля, игрока и срабатывания событий, наблюдаемыми (наследникам класса Subject) являются классы Field (в методе передвижения игрока), Event (при срабатывании события) и Player (при изменении его характеристик). Для логирования состояния игры наблюдаются классы Controller (при перемещении игрока) и СommandReader(при вводе клавиш пользователем). Для логирования ошибок наблюдается класс Field во время задания размеров поля и перемещения игрока на непроходимую клетку.

При срабатывании оповещения наблюдателей notify(), для них генерируется

соответствующее сообщение (объект Message) и передаётся их метод update(), где создаётся объект логгера (класса Logger), выводящий переданное сообщение в консоль или файл в зависимости от того, какие конфигурации хранятся в объекте класса LogOutInfo, хранящегося в объекте класса сообщения.

1. Определяется класс-интерфейс вывода сообщений Output, от которого наследуются классы FileOut и ConsoleOut, выводящие сообщения в файл и консоль соответственно.

Реализуются виртуальные методы класса с модификатором доступа public:

• void print(Message& message)= 0; – чисто виртуальный метод вывода сообщения.

2. Определяется класс FileOut, записывающий сообщение-лог в файл. Реализуются методы класса с модификатором доступа public:

• void print(Message& message) override – метод вывода сообщения в файл

• FileOut(std::string filename) – конструктор класса, открывающий файл с именем filename.

• ~FileOut() – деструктор класса, закрывающий файл с именем filename

Поля класса с модификатором доступа private:

• std::ofstream file – файл.

3. Определяется класс ConsoleOut, записывающий сообщение-лог в консоль.

Реализуются методы класса с модификатором доступа public.

• void print(Message& message) override – метод вывода сообщения в консоль.

4. Определяется класс LogOutInfo, хранящий уровни логирования и потоки вывода, выбранные пользователем.

Реализуются поля класса с модификатором доступа private:

• std::vector <OUTPUT> outputs – вектор потоков вывода, выбранных пользователем.

• std::vector <LEVEL> levels – вектор уровней логирования, выбранных пользователем.

Реализуются методы конструктор, геттеры и сеттеры вышеприведённых полей класса с модификатором доступа public.

5. Определяется класс Subject, являющийся абстрактным базовым классом для наблюдаемых объектов классов в игре (Field, Event, Player, CommandReader, Controller).

Реализуются методы класса с модификатором доступа public:

• void attach(Observer \*observer)– метод, добавляющий наблюдателя в свой вектор наблюдателей.

• void detach(Observer \*observer) – метод, удаляющий наблюдателя из своего вектора наблюдателей.

• void notify(Message& message) - метод, оповещающий всех наблюдателей наблюдаемого объекта.

Реализуются методы класса с модификатором доступа protected:

• std::vector<Observer \*> observers - вектор указателей на наблюдателей данного класса.

6. Определяется класс-интерфейс Observer, наследниками которого являются наблюдатели.

Реализуются методы класса с модификатором доступа public:

• virtual void update(Message &msg) = 0– метод обновления информации для наблюдателя.

7. Определяются классы, наследуемые от Observer: GameObserver, StatusObserver и ErrorObserver, объектом которых являются наблюдатели, связанные с первым, вторым и третьим уровнем логирования соответственно. Реализуются методы класса с модификатором доступа public:

• void update(Message &msg) override – метод, в который передаётся объект класса сообщения и который, проверяя необходимый уровень логирования данного сообщения, создаёт объект класса Logger, логирующий сообщение.

8. Определяется класс Logger, являющийся логгером сообщений и обёрткой для классов FileOut и ConsoleOut.

Реализуются методы класса с модификатором доступа public:

• Logger(LogOutInfo\*info) – конструктор класса.

• void print(Message &message) – метод вывода сообщения, вызывающий метод вывода сообщения классов FileOut и ConsoleOut в зависимости от выбора пользователя.

Реализуются поля класса с модификатором доступа private:

• std::vector<Output\*>outs – вектор объектов классов, выводящих сообщение в файл либо консоль.

9. Определяется класс Message, объектом которого является событие, отслеживающие победу и поражение в игре соответственно.

Реализуются методы класса с модификатором доступа public:

• friend std::ostream &operator<<(std::ostream &os, Message &message) -

перегрузка оператора вывода в поток для объекта данного класса

• Геттеры и сеттеры всех нижеперечисленных полей

Поля с модификатором доступа private:

• LEVEL type – уровень логирования.

• std::string message – текст сообщения.

• std::string pref– текст префикса с информацией об уровне логирования.

• LogOutInfo \*info - указатель на объект класса c информацией о выборе логирования и способа вывода логов.

**ПРИЛОЖЕНИЕ 1**

# UML-диаграмма межклассовых отношений

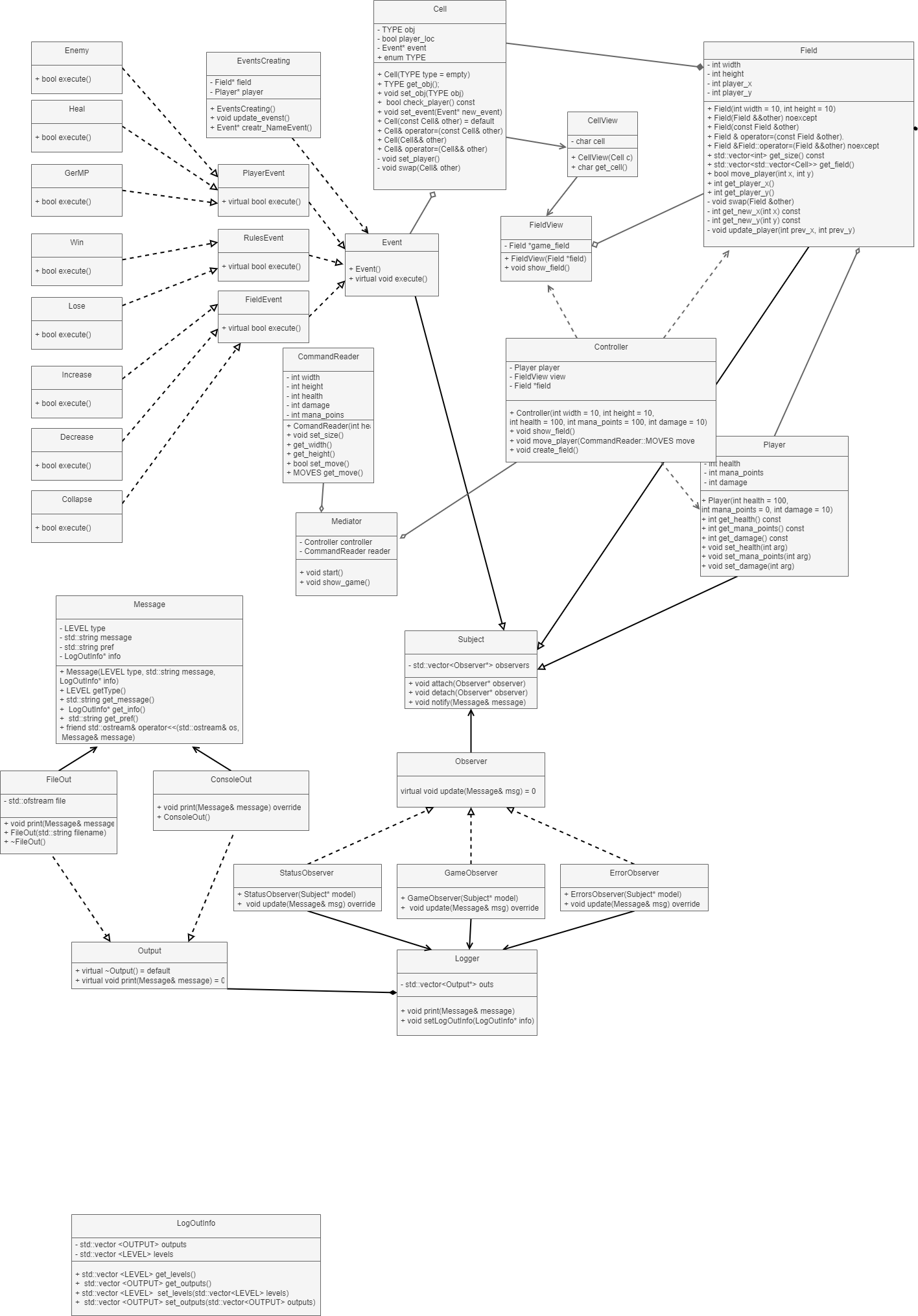


Рис 1. – UML-диаграмма.

**ПРИЛОЖЕНИЕ 2**

Результат работы программы

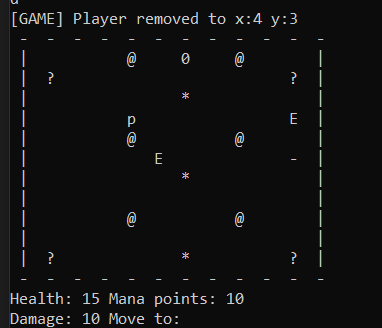


Рис 2. – демонстрация работы программы.

**Выводы.**

Было изучено и освоено на практике логирование. Научились перегружать операторы, а также грамотно расставлять логи для качественного отслеживания ошибок.