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. 1303 _____ Коренев Д.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Научиться выполнять перегрузку операций и логирование путём реализации набора классов, которые отслеживают изменения состояний в программе и классов, которые выводят информацию в разные уровни (консоль/файл) с перегруженным оператором вывода в поток.

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 инициализирован 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)

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.

Для выполнения лабораторной работы созданы классы, отвечающие за сообщение, которое выводится во время логирования, логирование конкретных объектов, и отвечающие за вывод в разные уровни(консоль/файл).

1) Создан интерфейс Logger с ваиртуальным методом output(Message*). Все классы реализующие его — способы представления логов. В данной работе это ConsoleLog — вывод логов в консоль и FileLog — вывод логов в файл. Кроме того это может быть, например, класс реализующий вывод логов в сеть.

2) ConsoleLog реализует интерфейс Logger. Метод output(Message*) определен на вывод аргумента метода в консоль. Так как у класса Message переопределен оператор <<, то это упрощает работу.

3) FileLog реализует интерфейс Logger. Этот класс реализует идиому RAII. Во время создания экземпляра создается и открывается файл, а в деструкторе — закрывается. Метод output(Message*) определен на вывод аргумента метода в файл.

4) Класс Levels имеет в себе перечисление уровней логирования: GameMessage, StatusMessage, ErrorMessage.

5) Класс Message — представляет из себя log. Принимает в качестве аргументов конструктора уровень логирования и текстовое содержание, а в самом конструкторе устанавливает их в поля. Также устанавливает в поле timeMessage время создания экземпляра. У каждого из этих полей есть геттеры. Оператор << переопределен: он принимает два аргумента: out типа ostream и экземпляр Messenger, и по очереди передает в out уровень лога, время и содержание сообщения. Объекты способные работать с экземплярами данного класса, могут выводить его содержимое в определенный поток.

6) Класс LogPool — обработчик экземляров Messenger. Он реализован с помощью паттерна одиночка. Имеет поле instance и метод getInstance, который создает экземпляр данного класса если в случае его отсутствия. Метод setLevels устанавливает какие уровни необходимо логировать, а метод setStream устанавливает куда будут выводиться логи (консоль, файл, консоль и файл, никуда). Метод printLog принимает в качестве аргумента экземпляр Message, проверяет соответствие его уровня требуемому, вызывает для каждого объекта способного выводить логи, хранящиеся в векторе loggers, метод output с аргументом который поступил ему.

Во все классы, в которых необходимо отслеживать изменения, добавлены изменения. Создавалось сообщение с нужным уровнем и содержанием, передавалось в метод printLog у экземпляра LogPool.

Тестирование.

Тестирование программы с считыванием данных для логирования представлено на рисунке 1.

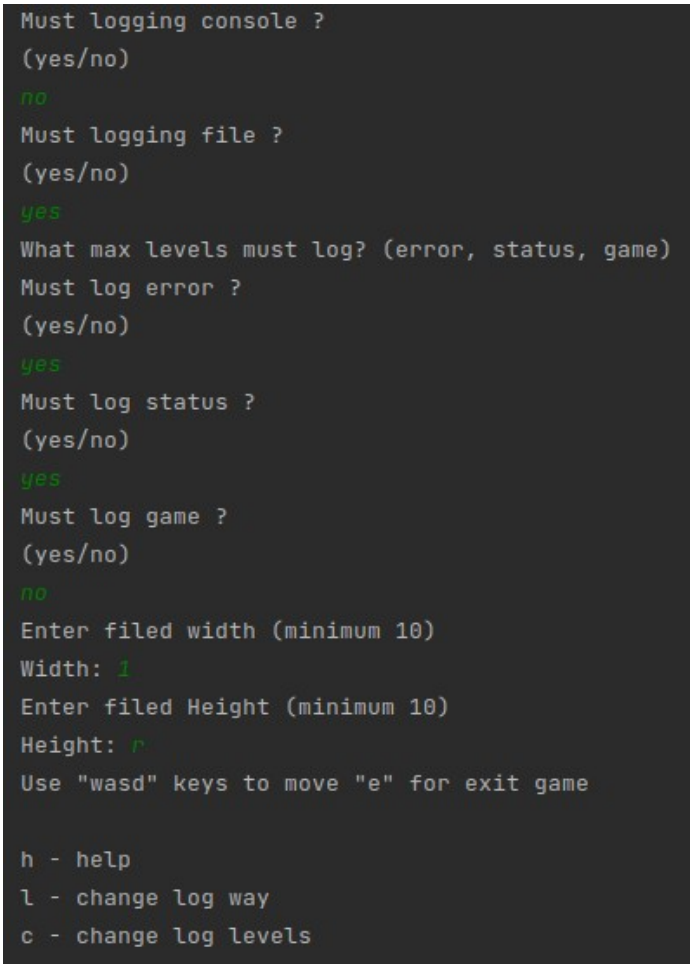

Рисунок 1 - Логирование в файл ошибок и статуса игры

Тестирование программы с выводом в файл представлено на рисунке 2.

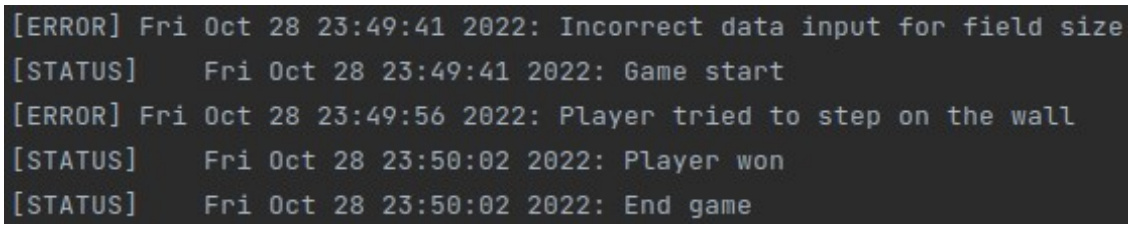

Рисунок 2 - Логи в файле

UML-диаграмма межклассовых отношений.

UML-диаграмма межклассовых отношений, созданных и измененных во время выполнения лабораторной работы, представлена на рисунке 3.

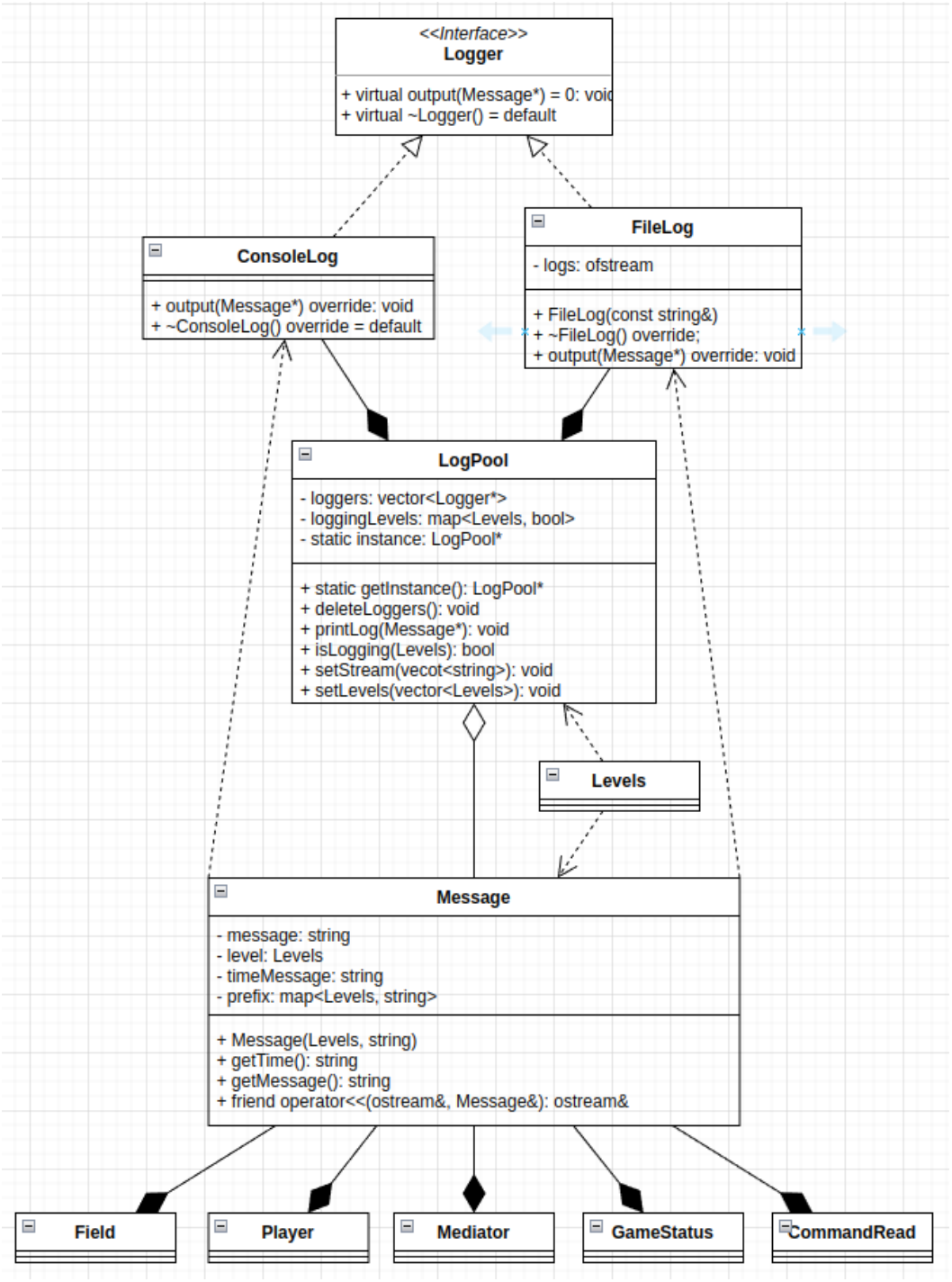

Рисунок 3 - UML диаграмма отношений классов логирования

Выводы.

Изучены основы логирования. В ходе лабораторной работы реализован набор классов, отвечающий за логирование игры, написаны классы, которые выводят информацию в разные потоки, а также создан класс с перегруженными оператором вывода в поток.