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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Логирование, перегрузка операций

Студент гр. 1304 ___ Байков Е.С.

Преподаватель ___ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

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)

Описание архитектурных решений и классов.

Реализован класс *Logger*, который осуществляет логирование трех уровней – *Acts* – производится запись действий, совершенных игроком, а также других действий, *Info* – выводит информацию об изменении поля, *Error* – вывод действий, которые являются ошибочными для игры (переход на заблокированную клетку или введение некорректных размеров поля). Класс является примером паттерна синглтон, чтобы облегчить работу с выводом в поток. *Logger* хранит в своих полях логгеры в несортированном множестве и массив используемых уровней. Для *Logger* перегружен оператор вывода в поток, который проходится по множеству логгеров и вызывает их оператор вывода в поток.

В *Logger* также реализован метод, который определяет префикс для определенного уровня сообщения, в виде названия уровня в квадратных скобках.

Создан интерфейс *ILogger*, который переопределяет оператор вывода в поток. Также реализовано два класса – *FieldLogger* и *ConsoleLogger*, которые реализуют интерфейс *ILogger*. *FieldLogger* с помощью переопределенного оператора вывода в поток выводит строку в файл, деструктор данного класса закрывает файл. *ConsoleLog* производит вывод в консоль.

Для событий созданы интерфейсы, которые имеют общего наследника *EventObserver*, переопределяющий методы каждого интерфейса. Внутри методов *EventObserver* производится логирование, по определенному правилу, а внутри событий вызывается определенный *observer* и производит специальный метод для логирования.

Для поля был создан интерфейс *IFieldObserver*, который наследуется *FieldObserver*, логирующий изменения на поле с помощью методов, просматривающих поле, также переопределен оператор вывода в поток, принимающий ссылку на *Field*, выводящий с помощью логера определенную строку вывода. Для игрока также был создан интерфейс *IPlayerObserver*, который

также просматривает изменения в игроке и выводит информацию с помощью соответствующего логгера, также переопределен оператор вывода в поток, принимающий ссылку на *Player*, выводящий с помощью логера определенную строку вывода.

Для того чтобы сделать *Logger* работоспособным был создан класс LogInitializer, который реализует выбор способа логирования информации и уровни логирования. Также создаются определенные логгеры для того, чтобы выводить информацию в соответствующий поток – файл, консоль, файл и консоль или ни в какой.

Демонстрация работы программы и тестирование.

Вывод в поток представлен на рисунке 1:

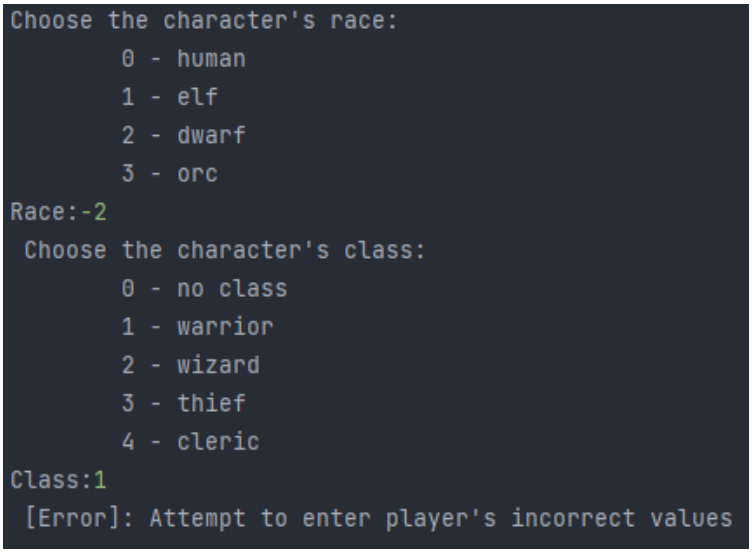

Рисунок 1 – Некорректные данные для создания персонажа логируются с префиксом [Error]

Вывод в консоль информации о действиях таких, как нанесение урона персонажу и разбиение блокированных клеток на рисунке 2:

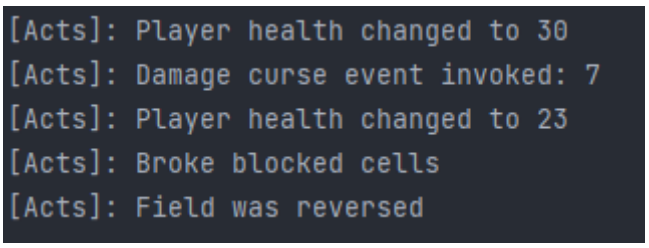

Рисунок 2 – действия, происходящие с полем и персонажем с префиксом [Acts]

Также представлен вывод в файл на рисунке 3:

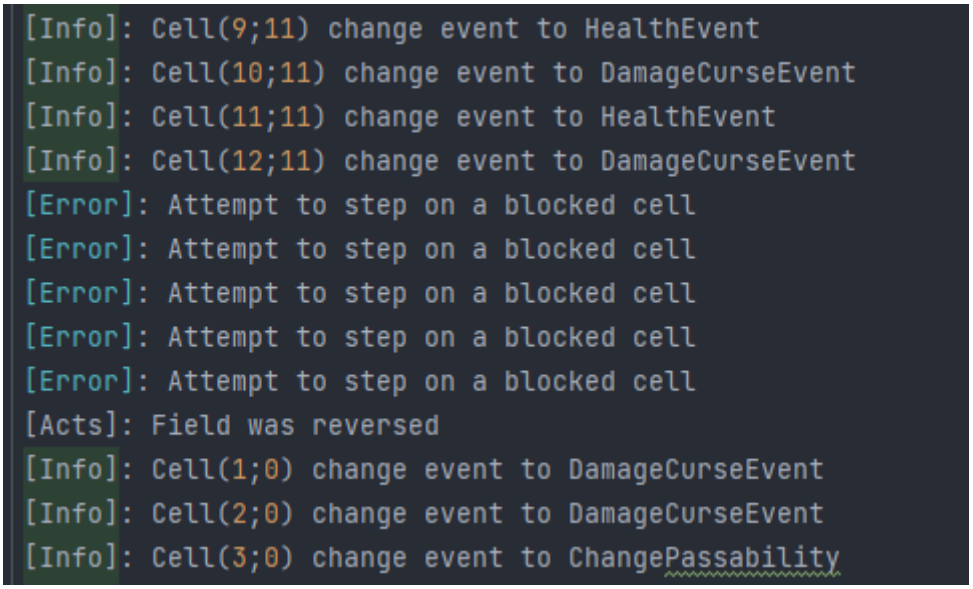

Рисунок 3 – логи внутри файла data.log

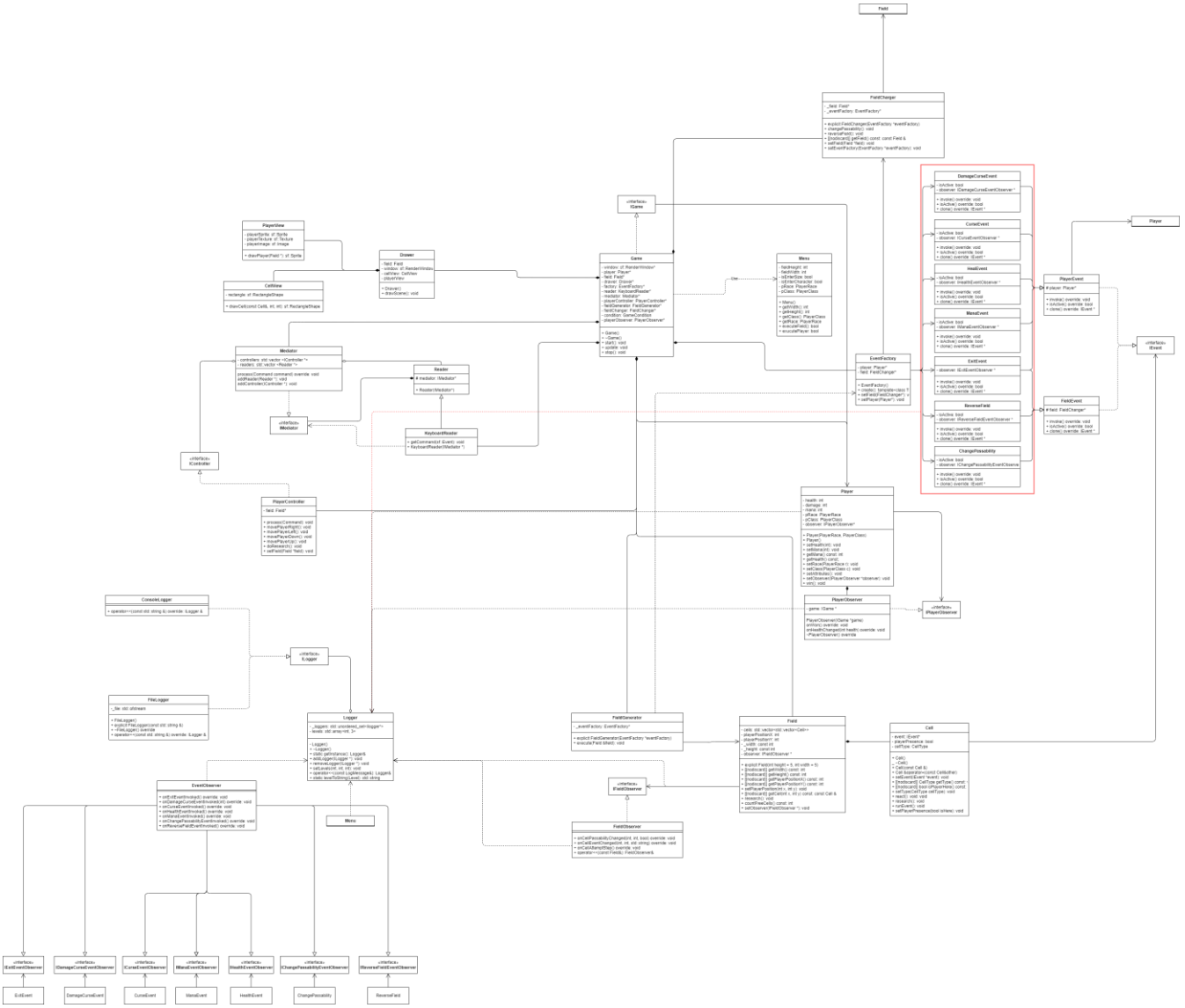

Рисунок 4 — UML-диаграмма классов.

Вывод.

Реализован классы, производящие логирование информации в соответствии с выбором пользователя.