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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «ООП»

Тема: Логирование, перегрузка операций

Студент гр. 0383 _____ Козлов Т.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

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

Потенциальные паттерны проектирования, которые можно использовать:

- Потенциальные паттерны проектирования, которые можно использовать:
- Адаптер (Adapter) - преобразование данных к нужному формату логирования
- Декоратор (Decorator) - форматирование тВВЦекста для логирования
- Мост (Bridge) - переключение между логированием в файл/консоль
- Наблюдатель (Observer) - отслеживание объектов, которые необходимо логировать
- Синглтон (Singleton) - гарантия логирования в одно место через одну сущность
- Заместитель (Proxy) - подстановка и выбор необходимого логирования

Ход работы:

Для реализации логгера был выбран паттерн «Наблюдатель». Был создан абстрактный класс Logger (наблюдатель) и класс Loggable (наблюдаемый).

По функционалу Logger имеет два метода: AddInLogger(Loggable* loggable) – реализующий добавление наблюдаемого объекта в наблюдатель (данный метод просто добавляет наблюдатель в список наблюдателей у наблюдаемого объекта, подробности далее) и чистый метод Update() – метод, который вызывается у наблюдателя в случае

изменения наблюдаемого объекта – в реализациях конкретных Логгеров данный метод будет логировать наблюдаемый объект.

От класса Logger наследуются классы FileLogger и ConsolLogger – классы, реализующие логирование в файл и в консоль соответственно.

Класс FileLogger имеет два приватных поля: std::ofstream _out – поток вывода и std::string _filename – название файла, в который производится запись. При создании экземпляра данного класса в конструкторе осуществляется открытие файла с заданным названием для записи (в случае, если файл для записи открыть не удалось – бросается исключение), в деструкторе происходит закрытие файла – таким образом соблюдается идиома RAII.

Класс ConsolLogger просто переопределяет функцию Update() – запись в этой функции происходит в стандартный поток вывода в консоль через std::cin.

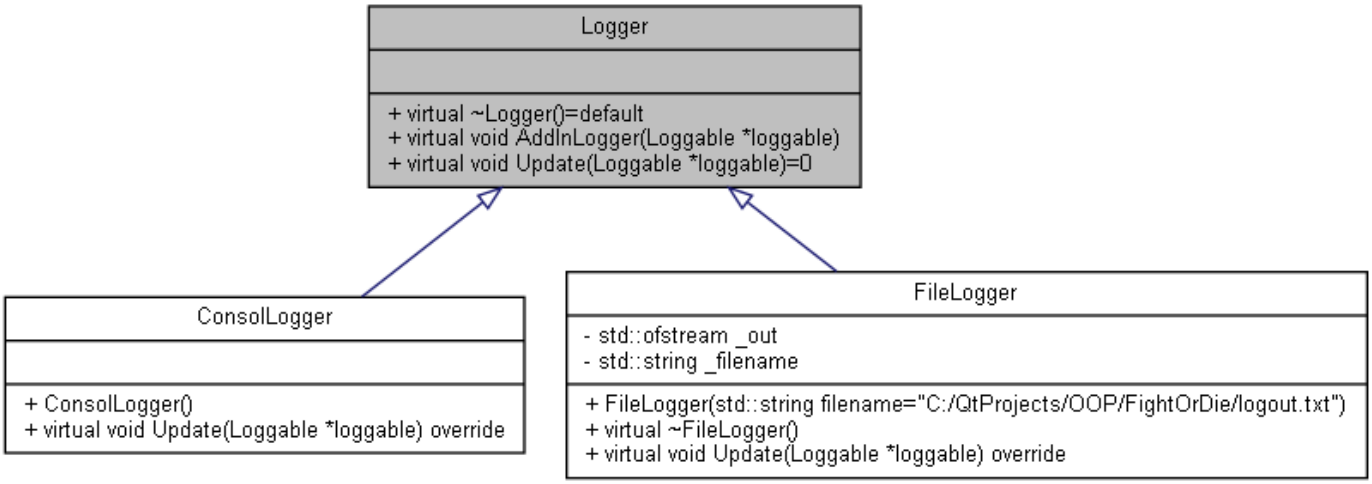

Рис. Logger

Экземпляр класс Loggable представляет собой «наблюдаемый» объект и имеет приватное поле-вектор _loggers – список «наблюдателей» за этим объектом (т.е. у одного наблюдаемого может быть сразу несколько наблюдателей). Данный класс имеет следующий интерфейс: метод AddLogger() – для добавления наблюдателя в список, функция notify() – для уведомления наблюдателя об изменении в объекте (т.е. данный метод просто последовательно вызывает метод Update() у каждого Наблюдателя из списка). Так же определена чистая виртуальная функция LogOut() – возвращающая строку, которую и будет выводить в поток наблюдатель. Так же для данного класса был переопределен бинарный оператор вывода в поток std::ostream& operator<< (std::ostream& out, const Loggable &obj) – который выводит в поток результат метода LogOut().

На момент написания лабораторной работы от класса Loggable наследуются Entity – объекты, находящиеся в клетках. Каждый из них по-своему переопределяет функцию LogOut() (Например игрок Player выводит свои характеристики (здоровье, атаку, защиту) и координаты –

при каждом изменении данных характеристик, а у предметов типа «Инвентарь»(Item) – вывод происходит только при использовании данного предмета)

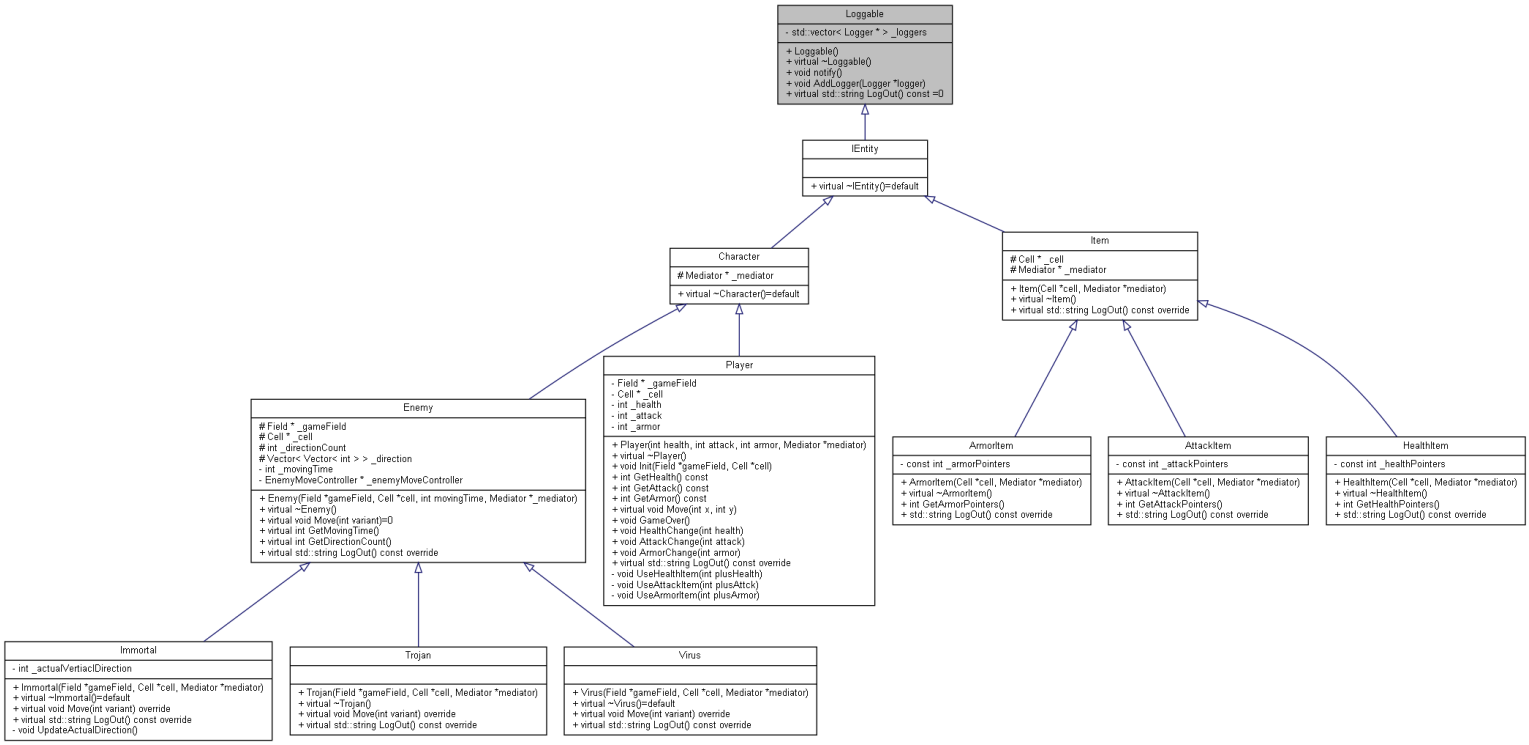

Рис. Loggable

На момент написания лабораторной работы добавление сущностей в Логгер захаркодено в поле, в последующем планируется вывести данную логику в «правило игры» - и передавать его классу-посреднику между командами управления и бизнес-логикой.

Пример работы FileLogger (содержимое файла):

Player Info:
Health: 110 Attack: 100 Armor: 100
Coordinates: column = 0 row = 0
HealthItem: Have used!
Player Info:
Health: 110 Attack: 100 Armor: 100
Coordinates: column = 0 row = 1
Player Info:
Health: 110 Attack: 110 Armor: 100
Coordinates: column = 0 row = 1
AttackItem: Have used!
Player Info:
Health: 110 Attack: 110 Armor: 100
Coordinates: column = 0 row = 2
Player Info:

Health: 110 Attack: 110 Armor: 110
Coordinates: column = 0 row = 2
ArmorItem: Have used!
Player Info:
Health: 110 Attack: 110 Armor: 110
Coordinates: column = 0 row = 3
Player Info:
Health: 110 Attack: 110 Armor: 110
Coordinates: column = 1 row = 3
Player Info:
Health: 110 Attack: 110 Armor: 110
Coordinates: column = 2 row = 3
Trojan info:
Coordinates: column = 0 row = 6
Trojan info:
Coordinates: column = 4 row = 6
Trojan info:
Coordinates: column = 5 row = 3
Trojan info:
Coordinates: column = 6 row = 0
Trojan info:
Coordinates: column = 5 row = 1
Virus info:
Coordinates: column = 2 row = 3
Virus info:
Coordinates: column = 4 row = 0

Выводы:

В ходе выполнения лабораторной работы был изучен и реализован паттерн «Наблюдатель». Была изучено переопределение операторов ввода/вывода в поток. Изучена работа с потоками ввода-вывода. Написаны классы логгеров, соблюдающих идиому RAII. По добавленному функционалу написана UML-диаграмма.