# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №5**

# по дисциплине «Объектно-ориентированное программирование»

**Тема: Управление, разделение на уровни абстракции.**

Студентка гр. 0382 Кривенцова Л.С.

Преподаватель Жангиров Т.Р.

Санкт-Петербург 2021

# Цель работы.

# Изучить принцип разделение на уровни абстракции, организовать управление игрой.

# Задание.

Необходимо организовать управление игрой (номинально через CLI). При управлении игрой с клавиатуры должна считываться нажатая клавиша, после чего происходит перемещение игрок или его взаимодействия с другими элементами поля.

Требования:

* Реализовать управление игрой. Считывание нажатий клавиш не должно происходить в классе игры, а должно происходить в отдельном наборе классов.
* Клавиши управления не должны жестко определяться в коде. Например, это можно определить в отдельном классе.
* Классы управления игрой не должны напрямую взаимодействовать с элементами игры (поле, клетки, элементы на клетках)
* Игру можно запустить и пройти.

# Основные теоретические положения.

Логирование.

Логированием называют запись логов. Оно позволяет ответить на вопросы, что происходило, когда и при каких обстоятельствах. Без логов сложно понять, из-за чего появляется ошибка, если она возникает периодически и только при определенных условиях.

Идиома RAII

# Получение ресурса есть инициализация (англ. Resource Acquisition Is Initialization (RAII)) — программная идиома, смысл которой заключается в том, что с помощью тех или иных программных механизмов получение некоторого ресурса неразрывно совмещается с инициализацией, а освобождение — с уничтожением объекта.

# Типичным (хотя и не единственным) способом реализации является организация получения доступа к ресурсу в конструкторе, а освобождения — в деструкторе соответствующего класса.

# Эта концепция может использоваться для любых разделяемых объектов или ресурсов:

# для выделения памяти,

# для открытия файлов или устройств, и др.

# Выполнение работы.

**Ход решения:**

Используется стандартная библиотека c++ и её заголовочные файлы *iostream*, *cstdlib*, *ctime* (для установки начала последовательности, генерируемой функцией *rand*(), для которой в подключен *cmath*), fstream для работы с файлами и string для использования строк.

Для более удобного логирования использован паттерн Наблюдателя. Для этого были созданы следующие классы:

1. Класс *Logger*. Выступает в роли наблюдателя: принимает сигналы от отслеживаемых объектов и выводит на экран их состояние с помощью переопределенного оператора вывода в поток.

Зависит от класса *IObservable*.

a. Определяются поля:

*std::pair <bool,bool> streams*; - пара логических значений, содержащая информацию о выбранном потоке: *first* – был ли выбран вывод в консоль, *second* – в файл. В конструкторе оба значения инициализируются как *false*. Модификатор доступа поля *private*.

*std::ofstream outfile*; - хранит файл, в который ведётся запись. Открытие файла происходит в конструкторе класса. Модификатор доступа поля *private*.

b. Реализуются методы:

*Logger(IObservable& address*); - Конструктор класса. Принимает на вход в качестве аргумента ссылку на объект класса *IObservable*. В конструкторе инициализируются поля, а принятому на вход объекту устанавливаются значения полей (через сеттеры), сохраняя в них адрес текущего объекта – логгера, который будет вести наблюдение и логическое значение, означаемое что объект находится под наблюдением в данный момент. Вызывает метод класса *LoggerView*() - *SelectStream*(), с помощью которого и оператора *switch* устанавливает выбранные пользователем параметры вывода логирования. Если пользователь ввёл некорректные данные, по умолчанию логирование будет производиться в консоль. Модификатор доступа метода - *public*.

*void File\_Settings()* – метод класса, проверяющий файл записи на безопасность. Если запись в него невозможна – вызывается метод класса *LoggerView*() - *PrintWarning*(), печатающий предупреждение на экран. Поле *streams* корректируется – поток вывода логирования принудительно становится *cout*. Модификатор доступа метода - *public*.

*~Logger();* - Деструктор класса, в нём происходит закрытие текстового файла outfile. Модификатор доступа метода - public.

*void update(std::string namefunc, Player &person);*

*void update(std::string namefunc, Enemy &monster);*

*void update(std::string namefunc, Cellule &cell);*

*void update(std::string namefunc, Field &fieldgame);*

*void update(std::string namefunc, Game &presentgame);*

*void update(std::string namefunc, Heal &food);*

*void update(std::string namefunc, Box &inbox);*

*void update(std::string namefunc, Teleport &teleports);*

Метод *update* перегружается для классов, за которыми может устанавливаться наблюдение. Принимает на вход два аргумента – имя функции, вызвавшей изменение состояния наблюдаемого объекта, и ссылка на этот объект. Метод с помощью *if* проверяет куда нужно выводить результат логирования и производит вывод(запись), вызывая оператор вывода в соответствующий поток. Модификатор доступа метода - *public*.

*friend std::ostream& operator<< (std::ostream &out, const Player &person);*

*friend std::ostream& operator<< (std::ostream &out, const Enemy &monster);*

*friend std::ostream& operator<< (std::ostream &out, const Cellule &cell);*

*friend std::ostream& operator<< (std::ostream &out, const Field &fieldgame);*

*friend std::ostream& operator<< (std::ostream &out, const Game &presentgame);*

*friend std::ostream& operator<< (std::ostream &out, const Heal &food);*

*friend std::ostream& operator<< (std::ostream &out, const Box &inbox);*

*friend std::ostream& operator<< (std::ostream &out, const Teleport &teleports);*

Оператор вывода в поток переопределяется через дружественные функции, так как при перегрузке через метод класса в качестве левого операнда используется текущий объект. В этом случае левым операндом является объект типа *std::ostream. std::ostream* является частью Cтандартной библиотеки C++. std::ostream не может использоваться в качестве левого неявного параметра, на который бы указывал скрытый указатель \**this*, так как указатель \**this* может указывать только на текущий объект текущего класса, члены которого мы можем изменить.

Оператор принимает ссылку на поток вывода и ссылку на объект, поля которые печатаются с поясняющими комментариями. Таким образом состояние изменённого объекта фиксируется в файле или на консоли. Модификатор доступа метода - *private*.

2. Класс *LoggerView*. Ассоциативно связан с классом *Logger*. Класс обеспечивает прослойку между пользователем и логгером – выводит сообщения и даёт возможность сделать выбор с помощью консоли.

a. Реализуются методы класса с модификатором доступа *public*:

*explicit LoggerView();* - Конструктор класса, на вход ничего не принимает. За неимением полей ничего не инициализирует.

*void PrintWarning();* - метод выводит предупредительное сообщение о том, что запись в файл невозможна. Ничего не принимает в качестве аргументов и ничего не возвращает.

*char SelectStream();* - метод выводит сообщение для пользователя, предлагая выбрать предложенные вариант записи логов. Считывает введенный символ, который и возвращает (*char*).

3. Класс *IObservable* предоставляет возможность его наследникам являться отслеживаемыми объектами (субъекты наблюдателя).

Является родителем классов *Characters, Item, Cellule, Field, Game.*

a. Определяются поля класса с модификатором доступа *protected*:

*bool observable;* - логическая переменная, предоставляющая информацию о том, находится объект под наблюдением или нет. Поле инициализируется false в конструкторах наследников класса.

*Logger\* observer*; - адрес объекта класса *Logger*. Хранится чтобы по этому адресу передать сигнал об изменении состояния текущего объекта класса. Поле инициализируется *nullptr* в конструкторах наследников класса.

b. Для доступа к этим полям (для установления их значений) реализуются сеттеры с модификатором доступа public:

*void SetObservable(bool obs);*

*void SetObserver(Logger\* obs);*

Принимают на вход соответствующие аргументы и устанавливают полям актуальные значения.

b. Реализуется метод *notify* с модификатором доступа *public*:

*void notify(std::string namefunc, Enemy &monster);*

*void notify(std::string namefunc, Player &person);*

*void notify(std::string namefunc, Cellule &cell);*

*void notify(std::string namefunc, Field &fieldgame);*

*void notify(std::string namefunc, Game &presentgame);*

*void notify(std::string namefunc, Heal &food);*

*void notify(std::string namefunc, Box &inbox);*

*void notify(std::string namefunc, Teleport &teleports);*

Метод перегружается для возможности принимать адрес объектов наследников класса *IObservable*. Его функция заключается в «уведомлении» наблюдателя – объекта класса *Logger* – об изменении состояния текущего объекта. Поэтому в классах-наследниках она вызывается в местах, где происходят изменения полей (в основном – в сеттерах).

Метод проверяет поля класса *observable* и *observable*, и убедившись, что объект находится под наблюдением – вызывает метод update класса *Logger* (передавая туда имя функции, вызвавшей *notify* и адрес текущего объекта, которые метод принял в качестве аргумента), «обновляя» запись состояния текущего объекта. В противном случае ничего не происходит.

**Результат работы программы:**

Рис 1. – демонстрация работы программы с логированием в терминал Ubuntu.

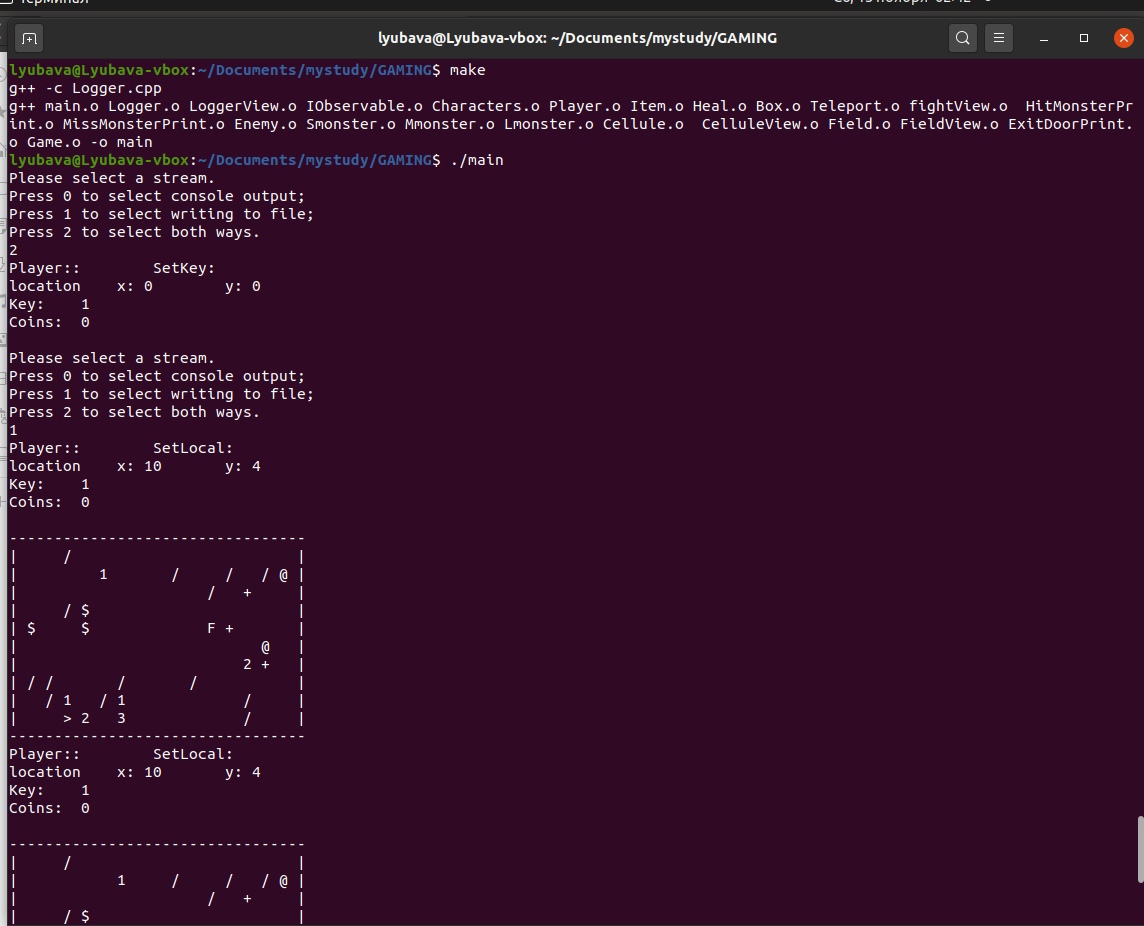
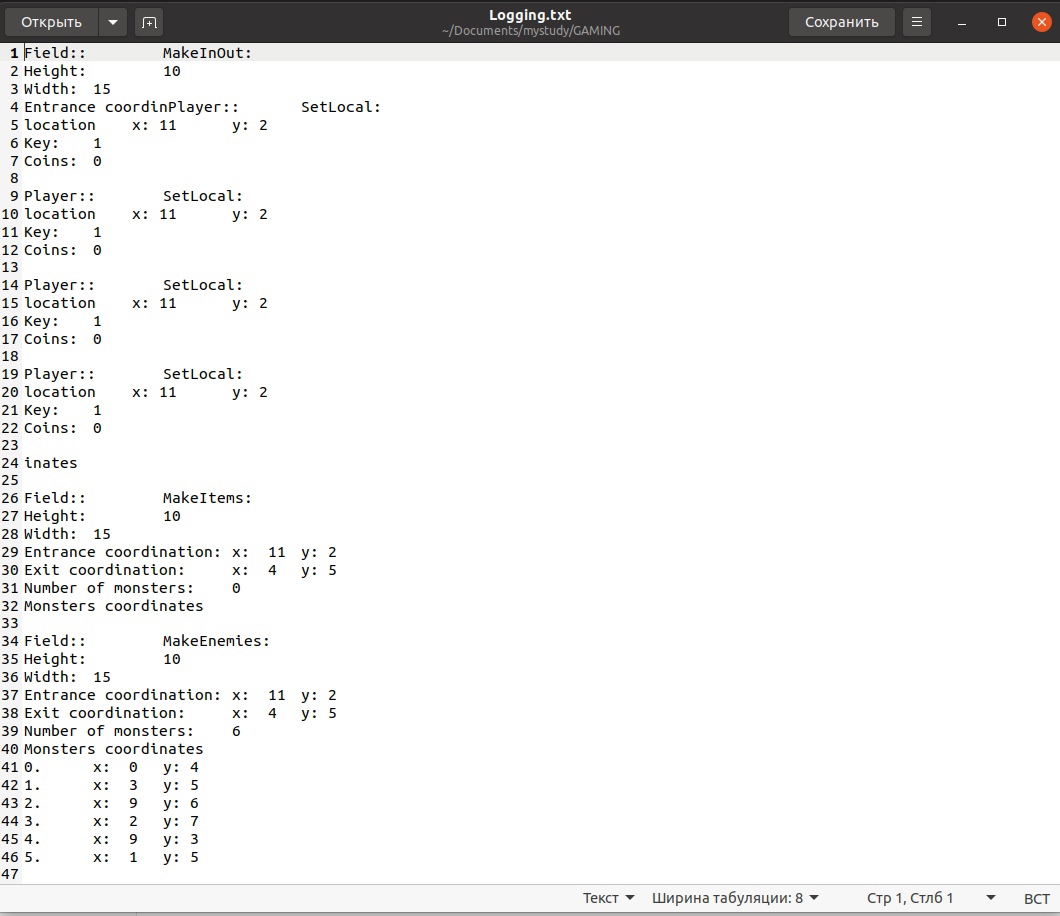
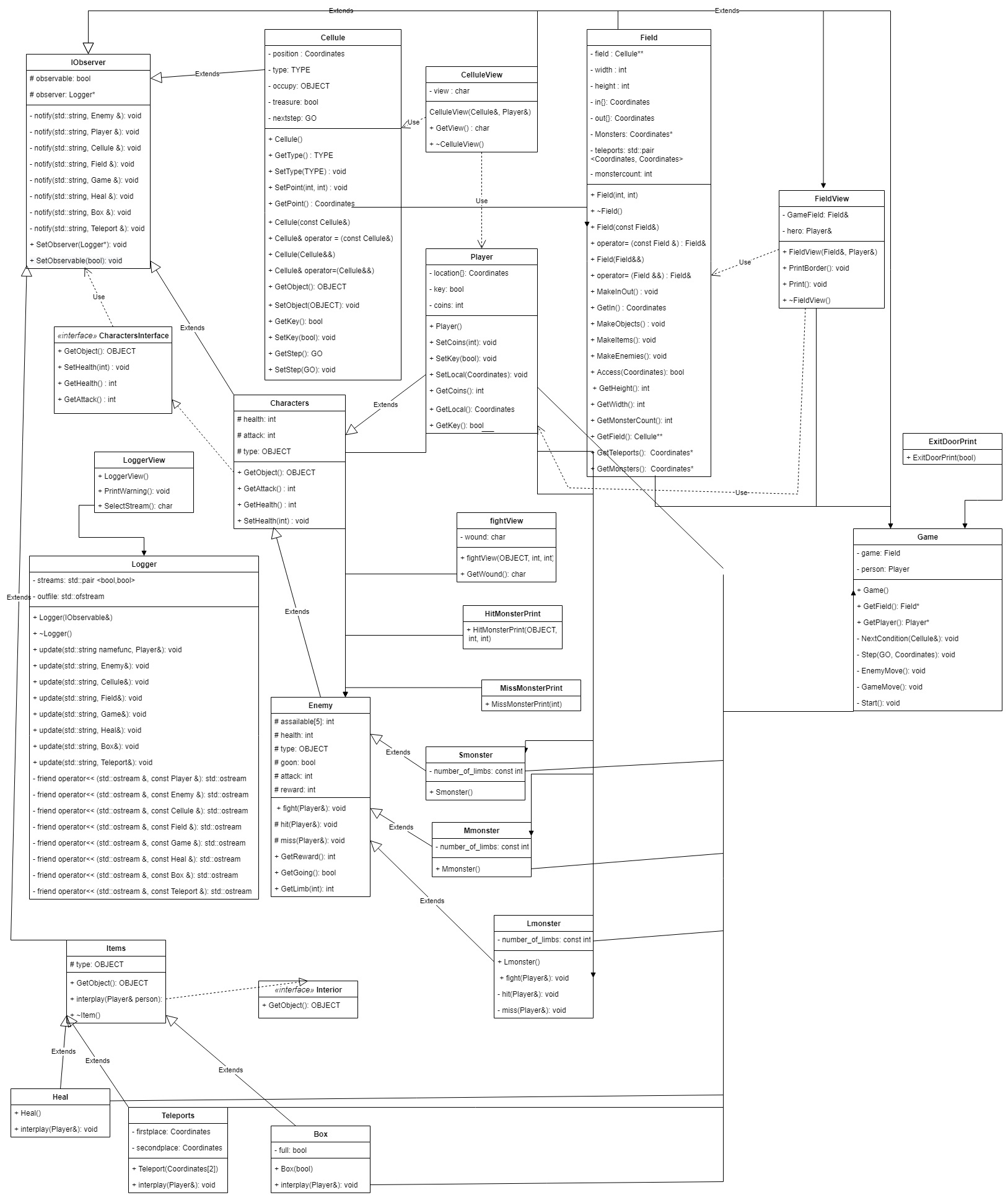


Рис 2. – демонстрация работы программы с логированием в текстовый файл.



**UML-диаграмма межклассовых отношений:**

Рис 3. – UML-диаграмма.



# Выводы.

Были изучены принцип логирования, идиома RAII, получены навыки реализовывать перегрузку операций и осуществлять ввод/вывод в файл.