**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Перегрузка операторов / Логирование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2303 |  | Волков И.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

**Цель работы.**

Создание системы классов, которая реализует логирование, через систему интерфейса и класса, который выполняет запись в нужный поток.

**Задание.**

а)  Реализовать набор классов “сообщений” с общим интерфейсом, который будут срабатывать в определенные моменты и хранить информацию о событии(не путать с классом игрового события из лаб. 3), но не должны хранить сообщение в виде строки. Должны быть реализованы класс для следующих событий:

* Игрок выиграл. Хранится информация о характеристиках игрока
* Игрок проиграл. Хранится информация о координатах клетки на которой событие произошло
* Была запущена новая игра. Хранится информация о размерах поля и стартовой позиции игрока.
* Была введена клавиша и сработала команда. Информация о введенном символе и какая команда сработала.
* Была введена клавиша, но никакая команда не сработала. Информация о введенном символе.

б) Для сообщений перегрузить оператор вывода в поток. Таким образом можно выводить сообщение в различные потоки (cout, файл). При выводе в поток сообщения, должна формироваться строка и подставляться хранимая информация.

в) Разработать систему классов, которые отслеживают сообщения и выводят их в файл и/или консоль. Куда выводить запрашивается у пользователя при запуске программы: никуда, в файл, в консоль, в файл и консоль. Классы, в которых происходит отслеживаемое событие, должны только отправлять сообщение, но не знать куда, то есть только создают сообщение, инициализируя его информацию, и отправляют.

Примечания:

* Система отслеживания должна масштабируема для новых потоков вывода без изменения кода. Для этого вывод в файл и терминал можно обернуть в отдельные классы с общим интерфейсом.
* Для записи в файл придерживайтесь идиомы RAII
* Отслеживаемые сущности не должны знать о том, кто и как их логирует

**Выполнение задания.**

Созданные классы: InputCmdMSG, NewGameMSG, LoseMSG, WinMSG, I\_Message, MessageOutput, lvl.

I\_Message – интерфейс для сообщения, содержит метод PackMSG(), который преобразует нужные данные в строку, которая должна будет пойти в лог.

InputCmdMSG – реализация интерфейса для создания сообщения при исполнении какой-либо команды.

Поля:

* Controlls cmd – команда, которая была введена
* int KeyCode – код клавиши, которая была написана

Методы:

* InputCmdMSG(std::map<int, Controlls>& crls, Controlls cmd) – конструктор класса, принимает словарь со всеми командами и команду
* std::string PackMSG() – упаковывает данные в строку нужного вида, возвращает эту строку

NewGameMSG – реализация интерфейса для создания сообщения при начале новой игры.

Поля:

* Pos fieldShape – структура, где лежат ширина и высота поля
* Pos PlayerPosition – структура с начальной позицией игрока

Методы:

* NewGameMSG(Field& field) – конструктор, принимает игровое поле, выбирает из него нужные данные
* std::string PackMSG() – упаковывает данные в строку нужного вида, возвращает эту строку

LoseMSG – реализация интерфейса для создания сообщения при проигрыше.

Поля:

* Pos position – структура с конечной позицией игрока

Методы:

* LoseMSG(Player& player) – конструктор, принимает объект игрока, выбирает из него нужные данные
* std::string PackMSG() – упаковывает данные в строку нужного вида, возвращает эту строку

WinMSG – реализация интерфейса для создания сообщения при начале новой игры.

Поля:

* std::vector<Charact> characts – список всех характеристик игрока

Методы:

* WinMSG(Player& player) – конструктор, принимает объект игрока, выбирает из него нужные данные
* std::string PackMSG() – упаковывает данные в строку нужного вида, возвращает эту строку

MessageOutput – класс, который хранит поток, куда нужно выводить данные, принимает нужные сообщения, отправляет строки из этих сообщений в нужный поток.

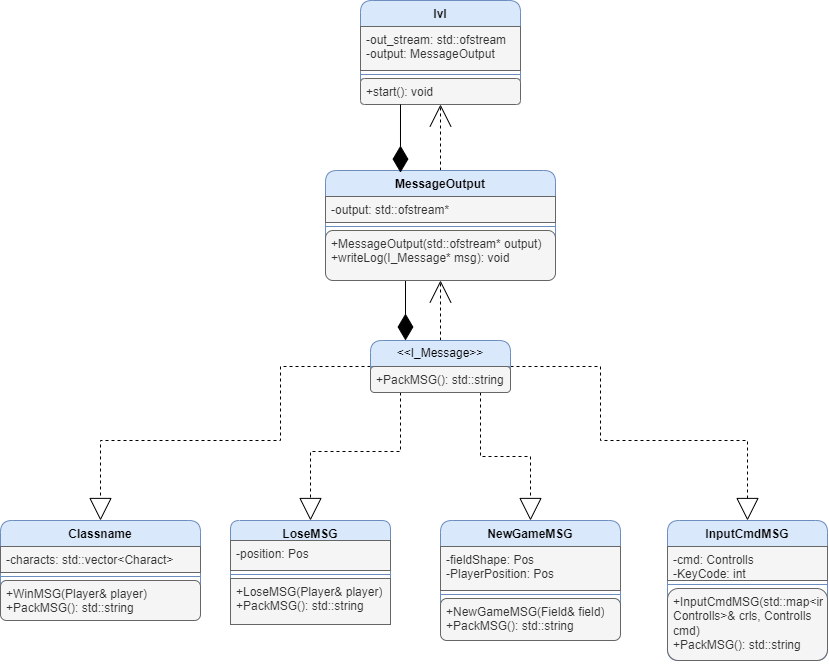
Поля:

* std::ofstream output – указатель на поток, куда нужно будет выводить данные

Методы:

* MessageOutput(std::ofstream output) – конструктор, принимает поток для вывода
* void writeLog(I\_Message\* msg) – принимает нужное сообщение, записывает сообщение в нужный поток

**UML**



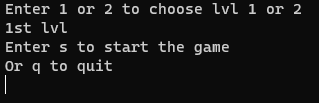
**Выводы.**

Реализованы необходимые классы с конструкторами, деструкторами и функционалом.

**Тестирование.**

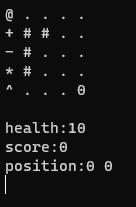
Проверка работы классов:

Здесь запущена программа, ввод – 1, выбран 1-ый уровень.



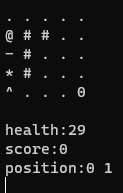
ввод – s

Запущен уровень, он отрисовывается корректно, программа ждёт ввода игрока

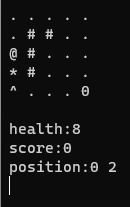


Сразу после ввода перерисовывается игровое поле, для ввода достаточно просто ввести символ, без подтверждения (нажатия enter).

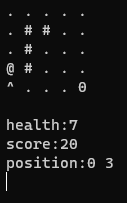
ввод – s



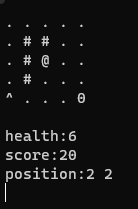
ввод – s



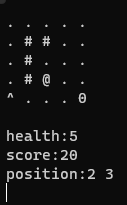
ввод – s



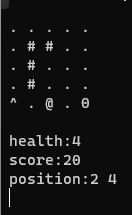
ввод – s



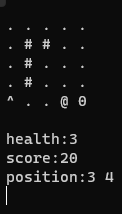
ввод – s



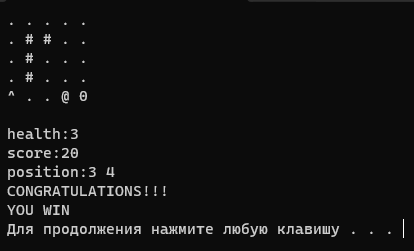
ввод – s



ввод – d



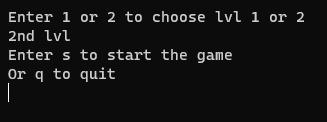
ввод – d



Все функции ввода работают корректно, отрисовка тоже, класс игры выполяет свою задачу.

Игра запущена второй раз, это 2-ой уровень.

ввод – 2



Ввод - s



Правила те же, что и в первом уровне (правила были перечислены на прошлой лабораторной работе). Сам уровень больше, загружается из файла.