**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1303 |  | Андреева Е.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2022

**Цель работы.**

Реализация набора классов отслеживающих изменения состояний в программе.

**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 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)

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.**

В программе были реализованы следующие классы: *ConsoleLogger, FileLogger, ILogger, Level, Log, Message, EventObserver, FieldObserver, GameObserver, PlayerObserver, IEventObserver, IFieldObserver, IGameObserver, IPlayerObserver*.

Класс *GameObserver* был создан для наблюдения за изменениями, которые происходят в игре*.* В классе были реализованы методы для отслеживания начала, конца игры и для отслеживания неправильно введенной высоты, ширины поля. Класс *IGameObserver* является интерфейсом для класса *GameObserver*.

Класс *EventObserver* был создан для наблюдения за срабатыванием событий*.* В классе были реализованы методы для отслеживания срабатывания каждого события (*Exit, Food, Money, Trap, Enemy*). Класс *IEventObserver* является интерфейсом для класса *EventObserver*.

Класс *FieldObserver* был создан для наблюдения за изменением полем. В классе были реализованы метод, который отслеживает открытие выхода, метод, который отслеживает, на какую клетку перешел игрок, и метод, который отслеживает ситуацию, когда игрок попытался перейти на непроходимую клетку. Класс *IFieldObserver* является интерфейсом для класса *FieldObserver*.

Класс *PlayerObserver* был создан для наблюдения за изменением характеристик игрока. В классе были реализованы методы, которые отслеживают ситуации, когда игрок выиграл, проиграл, и когда у игрока увеличились или уменьшились здоровье, защита, очки. Класс *IPlayerObserver* является интерфейсом для класса *PlayerObserver*.

В каждом методе классов-наблюдателей формируется сообщение (объект класса *Message*) с соответствующей строкой-сообщением и уровнем логирования и вызывается метод *viewMessage()* класса *Log* для дальнейшего вывода сообщения.

Для удобства представления уровней логирования был создан *enum* класс *Level*.

Класс *Message* создан для удобства передачи сообщения и уровня между классом *Log* и классами-наблюдателями. У класса Message были созданы приватные поля *message*(хранит текст сообщения) типа *std::string* и *level*(хранит уровень для сообщения) типа *Level*. Также были реализованы методы для получения и изменения значений этих полей и конструкторы (конструктор без аргументов, конструктор с двумя аргументами).

Класс *Log* был создан для вывода полученного сообщения в консоль либо в файл. У класса были созданы приватные поля *loggers\_array*(для хранения ссылок на объекты, с помощью которых будет выводится сообщение либо в файл, либо в консоль) типа *ILogger \*\**, *level*(для хранения уровня логирования, выбранного пользователем) типа Level и *number\_of\_loggers*(количсетво элементов массива *logger\_array*) типа *int*. В классе был реализован метод *viewMessage*, в котором осуществляется вывод сообщения в консоль, либо в файл, либо и в консоль и в файл. Также был реализован конструктор, который, в зависимости от того, как выбрал пользователь, создает масссив *logges\_array* и устанавливает уровень логирования также по выбору пользователя.

В классе-интерфейсе *Ilogger* был объявлен оператор <<, который принимает объект класса Message. Далее оператор будет перегружен в классах *ConsoleLogger* и *FileLogger*.

Класс *FileLogger* был создан для вывода сообщения в файл. В классе было создано приватное поле *file*(для хранения файла, в который будет делаться вывод сообщений) типа *std::ofstream*. В классе перегружен оператор вывода <<, который принимает объект класса *Message* и выводит текст сообщения в файл.

Класс *ConsoleLogger* был создан для вывода сообщений в консоль. В классе перегружен оператор вывода <<, который принимает объект класса Message и выводит текст сообщения в консоль.

**Проверка работоспособности программы.**

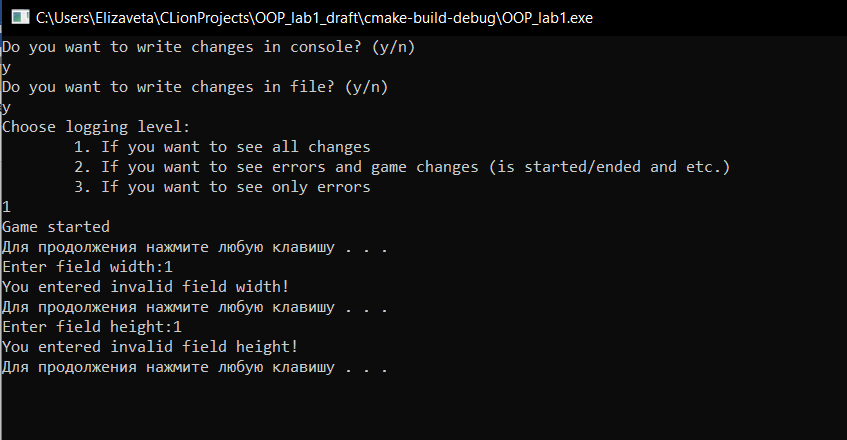
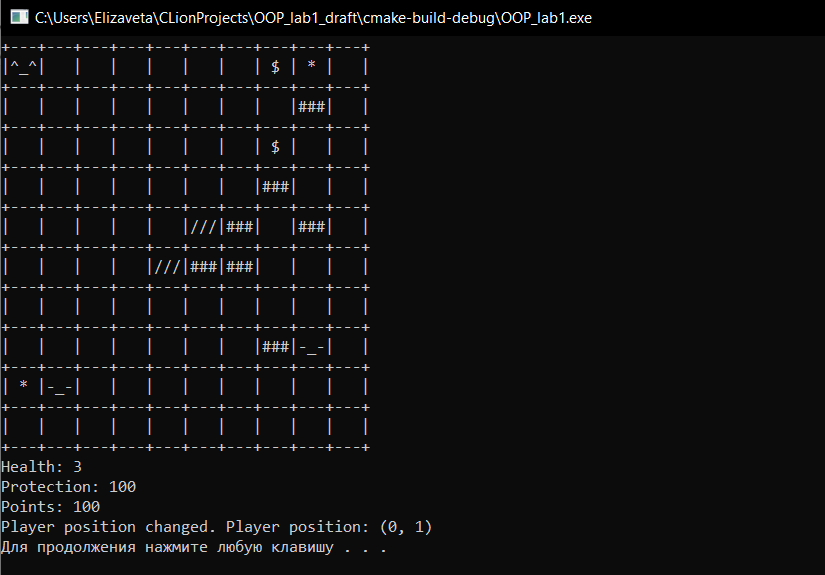
Рис. 1 Выбор уровня логирования и способа записи сообщений. 

Рис.2 Переход на клетку, которая находится снизу от игрока.

Содержимое файла FileLogger.txt после прохождения игры:

Game started

You entered invalid field width!

You entered invalid field height!

Player position changed. Player position: (0, 1)

Player position changed. Player position: (0, 2)

Player position changed. Player position: (0, 3)

Player position changed. Player position: (0, 4)

Player position changed. Player position: (0, 5)

Player position changed. Player position: (0, 6)

Player position changed. Player position: (0, 7)

Player position changed. Player position: (0, 8)

Food event is active

Player's health changed. Now there are 4 health.

Player's points changed. Now there are 110 points.

Player's points changed. Now there are 115 points.

Player position changed. Player position: (0, 9)

Player position changed. Player position: (1, 9)

Player position changed. Player position: (2, 9)

Player position changed. Player position: (3, 9)

Player position changed. Player position: (4, 9)

Player position changed. Player position: (5, 9)

Player position changed. Player position: (6, 9)

Player position changed. Player position: (7, 9)

Player position changed. Player position: (7, 0)

Money event is active

Player's points changed. Now there are 130 points.

Player's points changed. Now there are 135 points.

Exit is open now.

Player position changed. Player position: (7, 1)

Player position changed. Player position: (7, 2)

Money event is active

Player's points changed. Now there are 150 points.

Player's points changed. Now there are 155 points.

Player position changed. Player position: (8, 2)

Player position changed. Player position: (9, 2)

Player position changed. Player position: (9, 3)

Player position changed. Player position: (9, 4)

Player position changed. Player position: (9, 5)

Player position changed. Player position: (9, 6)

Player position changed. Player position: (9, 7)

Player position changed. Player position: (9, 8)

Player position changed. Player position: (9, 9)

Exit event is active

Player won!

Game ended

**UML-диаграмма межклассовых отношений:**

