**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №3**

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

**Тема: логирование, перегрузка операций**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0382 |  | Андрющенко К.С |
| Преподаватель |  | Жангиров Т. Р |

Санкт-Петербург

2021

**Цель работы.**

Создание класса логгера, который фиксирует процесс игры и реализует различный формат вывода информации о процессе.

**Задание.**

## 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

**Выполнение работы.**

В ходе работы реализуется паттерн проектирования Синглтон (Singleton).

UML-диаграмма для данной лабораторной работы см. Рисунок 1.

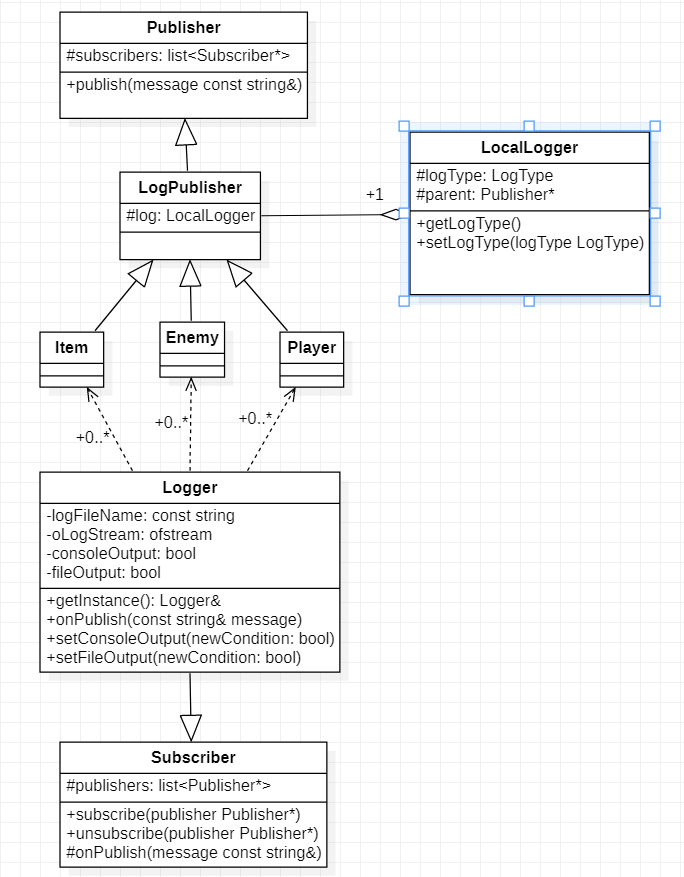


Рисунок 1 – UML-диаграмма лабораторной работы №3

Описание диаграммы.

## Описание диаграммы.

## Классы, созданные и модифицированные в данной лабораторной работе, их методы и свойства.

Класс Logger:

Заголовочный файл см. ПРИЛОЖЕНИЕ A.

Класс, реализующий журналирование данных.

Свойства класса.

*Bool fileOutput* – Флаг вывода логов в файл.

*bool consoleOutput–* Флаг вывода логов в консоль.

*string logFileName* – Имя файла, используемого для вывода логов.

*ofstream oLogStream –* Вывод в файл.

Методы класса.

*static Logger& getInstance() -* Функция получения ссылки на экземпляр класса для работы.

*void onPublish(const std::string& message)* – Вывод лога заданным образом.

*void setConsoleOutput(bool newCondition);*

*void setFileOutput(bool newCondition) –* Установка флагов вывода логов*.*

Класс Subscriber:

От данного класса наследуется класс логгера.

*list<Publisher\*> publishers* – список издателей, на которые подписан логгер.

*virtual void onPublish(const std::string& message) –* виртуальная функция вывода логов, которая переопределена в классе логера.

*void subscribe(Publisher\* publisher);*

*void unsubscribe(Publisher\* publisher) –* удаление и добавление издателя.

Издателями будут являться классы, состояние которых мы фиксируем ( Item, Enemy, Player). Они наследуют логгер.

Заголовочный файл см. ПРИЛОЖЕНИЕ B.

Класс Publisher:

*void publish(const std::string& message) –* обходит лист подписчиков и вызывает у них функцию *onPublish.().*

Заголовочный файл см. ПРИЛОЖЕНИЕ C.

Класс LogPublisher:

Класс, добавляющий в Publisher работу с логами. Можно было бы добавить этот функционал в сам Publisher, но тогда это уже был бы не совсем Publisher, он

был бы уже заточен под работу с логами. Каждый класс должен делать то, что

говорится в его имени. Publisher только быть publisher'ом и не более. Данный класс наследует класс Publisher.

Свойства класса:

*LocalLogger log* – класс, вызывая который классы отправляют сообщения.

Заголовочный файл см. ПРИЛОЖЕНИЕ D.

Класс LocalLogger:

Свойства класса:

*Publisher\* parent* – указатель на издателя.

*Log::Type logType –* иерархия логов.

*Log::Type getLogType()*

*void setLogType(Log::Type logType)* – установка и получение типа лога.

LocalLogger\* operator<<(const std::string& message) – перегрузка оператора <<. Лог, который будет записан в файл будет состоять из заданного сообщения и типа лога. Отношение данного класса с классом LogPublisher – агрегация.

Заголовочный файл см. ПРИЛОЖЕНИЕ E.

## Отношение между классами.

Описание классов.

## Проверка классов.

Приложим скриншоты тестов, находящихся в main(), результат и при необходимости памяти после выполнения тестов. Подписываемся на класс врага, игрока и предмета.

Обработаем случай взаимодействия с врагом, предметом, непроходимой клеткой См. Рисунок 3:

Main():

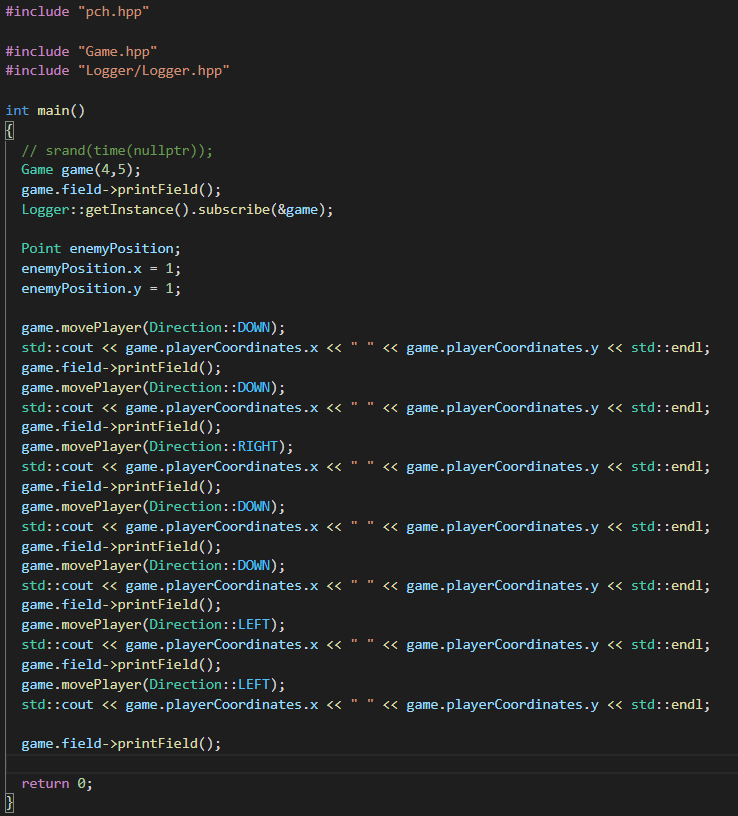


Рисунок 3 - Движение игрока по полю, запись логов.

Output(консоль)/(файл game.log):

2 ent 0 3 0 0

1 0 4 4 0

1 3 0 1 0

0 0 0 0 0 ext

0 1

0 ent 2 0 3 0

1 0 4 4 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

0 2

0 ent 0 2 0 3

1 0 4 4 0

1 0 3 1 0

0 0 0 0 0 ext

MOVEMENT: Player's current health is max

MOVEMENT: Player's current defence: 2

MOVEMENT: Player's current attack: 20

MOVEMENT: Attack: 0

MOVEMENT: Defense: 1

MOVEMENT: Health: 5

Non traversable for enemy

1 2

0 ent 0 0 0 3

1 0 2 4 0

1 0 3 1 0

0 0 0 0 0 ext

MOVEMENT: Player's current health: 100

MOVEMENT: Player's current defence: 0

MOVEMENT: Player's current attack: 25

MOVEMENT: Attack: 5

MOVEMENT: Defense: 0

MOVEMENT: Health: 0

Non traversable for enemy

1 3

0 ent 0 0 0 3

1 0 0 2 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

1 4

0 ent 0 0 0 3

1 0 0 0 2

1 0 3 1 0

0 0 0 0 0 ext

You have been damaged

BATTLE: Damage: 10

MOVEMENT: Player's current health: 90

BATTLE: Emeny is dead.

Non traversable for enemy

Non traversable for enemy

0 4

0 ent 0 0 0 2

1 0 0 0 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

0 4

0 ent 0 0 0 2

1 0 0 0 0

1 0 3 1 0

0 0 0 0 0 ext

**Вывод.**

В ходе работы были написан класс логгера, который фиксирует изменение отслеживаемых объектов и вводит информацию о процессе игры в файл или консоль.

ПРИЛОЖЕНИЕ A

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Logger.hpp**

#pragma once

#include "Subscriber.hpp"

/\*!

 \* \class Singleton

 \* \brief Класс, реализующий журналирование данных. Singleton. Чтобы вызвать функцию

 \* i.e. function() используйте: `Singleton::getInstance().function()`.

 \*/

class Logger : public Subscriber

{

public:

  /\*! \brief Функция получения ссылки на экземпляр класса для работы. \*/

  static Logger& getInstance();

  void onPublish(const std::string& message) override;

  void setConsoleOutput(bool newCondition);

  void setFileOutput(bool newCondition);

private:

  Logger();

  ~Logger();

  Logger(const Logger&) = delete;

  Logger& operator=(const Logger&) = delete;

  Logger(Logger&&) = delete;

  Logger& operator=(Logger&&) = delete;

  const std::string logFileName = "game.log";

  std::ofstream oLogStream;

  bool consoleOutput = true;

  bool fileOutput = true;

};

ПРИЛОЖЕНИЕ B

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Subscriber.hpp**

#pragma once

#include "Publisher.hpp"

class Publisher;

class Subscriber

{

public:

  void subscribe(Publisher\* publisher);

  void unsubscribe(Publisher\* publisher);

protected:

  virtual void onPublish(const std::string& message) = 0;

  std::list<Publisher\*> publishers;

  friend class Publisher;

};

ПРИЛОЖЕНИЕ C

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Publisher.hpp**

#pragma once

#include "Subscriber.hpp"

class Subscriber;

class Publisher;

class Publisher

{

public:

  void publish(const std::string& message);

protected:

  std::list<Subscriber\*> subscribers;

  friend class Subscriber;

};

ПРИЛОЖЕНИЕ D

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл LogPublisher.hpp**

#pragma once

#include "Publisher.hpp"

#include "LocalLogger.hpp"

// Класс, добавляющий в Publisher работу с логами. Можно было бы добавить этот

// функционал в сам Publisher, но тогда это уже был бы не совсем Publisher, он

// был бы уже заточен под работу с логами. Каждый класс должен делать то, что

// говорится в его имени. Publisher только быть publisher'ом и не более.

class LogPublisher : public Publisher

{

protected:

  LocalLogger log = LocalLogger(this);

};

ПРИЛОЖЕНИЕ E

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл LocalLogger.hpp**

#pragma once

#include "Publisher.hpp"

// Класс, "сидящий" в каждом классе, что умеет писать в лог. Именно вызывая

// его классы отправляют сообщения в логгер.

class LocalLogger

{

public:

  LocalLogger(Publisher\* parent) : parent(parent) { };

  ~LocalLogger() { };

  LocalLogger\* operator<<(const std::string& message);

  Log::Type getLogType();

  void setLogType(Log::Type logType);

protected:

  Log::Type logType = Log::Type::GENERAL;

  Publisher\* parent;

};