

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «ООП»
Тема: Полиморфизм

Студент гр. 8383

Шишкин И.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Приобрести знания по полиморфизму. Реализовать логирование действий в программе.

Постановка задачи.

Реализовать набор классов, для ведения логирования действий и состояний программы. Основные требования:

Логирование действий пользователя

Логирование действий юнитов и базы

Ход работы.

Были реализованы: наборы классов для ведения логирования действий и состояний программы. Для логирования перегружен оператор вывода в поток. Для переключения между разным логированием создан паттерн «Прокси». Для разного формата записи создан паттерн «Адаптер».

Описание дополнений к программе.

Был реализован класс `Logger` (файлы `Logger.h`, `Logger.cpp`), который является интерфейсом для классов, используемых для логирования. Возможны 3 варианта логирования (игрок выбирает их в самом начале игры): `LoggingFile` – логирование в файл “log.txt” (файлы `LoggingFile.h`, `LoggingFile.cpp`); `LoggingTerminal` – логирование в терминал (файлы `LoggingTerminal.h`, `LoggingTerminal.cpp`); `NoLogging` – без логирования (файлы `NoLogging.h`, `NoLogging.cpp`). Также был перегружен оператор вывода в поток.

Чтобы пользователь мог переключать режимы логирования, создан класс `LoggerProхy`, который реализует паттерн «Прокси» (файлы `LoggerProхy.h`, `LoggerProхy.cpp`). С помощью метода `setLogger` можно установить другой режим логирования.

Для разных форматов вывода логов реализован класс `LogFormatter` (файлы `LoggerFormat.h`, `LoggerFormat.cpp`). Созданы классы `FormatterTime` и

FormatterEmpty, наследующиеся от класса LogFormatter. Для вывода отформатированной строки также используется класс LoggerProху, который реализует паттерн «Адаптер». Перед вызовом метода логирования класс LoggerProху предварительно получает отформатированную строку от одного из вышеупомянутых классов (можно задать методом setFormat), а затем передает ее соответствующему логгеру.

Тестирование.

Пример вывода логирования в терминал (логи выводятся со знаком “#” в начале строки).

```
1-й игрок, ваш ход:
Введите персонажа из эльфов или людей:
MS1
Введите направление:
Возможные варианты хода:
1 - вверх
2 - вправо
4 - вниз
5 - не двигаться
4
#GAME: Передвешение юнита "MS1" по направлению 4
Вы можете сделать еще один ход за своего персонажа
MS1
Введите направление:
Возможные варианты хода:
1 - вверх
2 - вправо
4 - вниз
5 - не двигаться
4
#GAME: Передвешение юнита "MS1" по направлению 4
```

Пример вывода логирования в файл “log.txt”.

log.txt – Блокнот

Файл Правка Формат Вид Справка

```
#GAME: Создана игра с размером поля 15; количество орков - 9; количество людей - 3
#GAME: Передвешение юнита "MW2" по направлению 4
#GAME: Добавление юнита " OR"
#UNIT: Юнит "EI0" атакует "OR4"
#GAME: Передвешение юнита "EI0" по направлению 7
#UNIT: Юнит "EI0" атакует "OR4"
#GAME: Передвешение юнита "EI0" по направлению 7
#Base: Создается база орков
#Base: Создается база людей и эльфов
#UNIT: Юнит "OR4" атакует "EI0"
#GAME: Передвешение юнита "OR4" по направлению 8
#GAME: Удаление юнита " EI0"
#GAME: Передвешение юнита "OR5" по направлению 1
#Base: Добавление на базу людей и эльфов юнита "EA"
#GAME: Game over!
```

Выводы.

В ходе выполнения лабораторной работы были реализованы наборы классов для ведения логирования действий и состояний программы.