**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно ориентированное программирование»**

Тема: **Полиморфизм**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8382 |  | Мирончик П.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2020

**Задание**

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

* Логирование действий пользователя
* Логирование действий юнитов и базы

|  |  |
| --- | --- |
| Выполнены основные требования к логированию | 3 балла |
| Реализована возможность записи логов в файл | 3 балла |
| Реализована возможность записи логов в терминал | 3 балла |
| Взаимодействие с файлами должны быть по идиоме RAII | 1 балл |
| *\*Для логирования состояний перегружен оператор вывода в поток* | *2 балла* |
| *\*Переключение между разным логированием (логирование в файл, в терминал, без логирования) реализуется при помощи паттерна “Прокси”* | *4 балла* |
| *\*Реализован разный формат записи при помощи паттерна “Адаптер”* | *4 балла* |
| **Кол-во баллов за основные требования** | **10 баллов** |
| **Максимальное кол-во баллов за лаб. работу** | **20 баллов** |

# Ход работы

**Описание основных классов.**

**Описание из лабораторной работы №2:**

*GameBoard* – корень приложения. Хранит информацию о клетках доски (*Cell*), привязанных к доске объектах (*GameObject*), подписчиках на изменения поля (*BoardListener*). К экземпляру *GameBoard* привязывается *GameController* и *MouseTracker*. GameBoard отвечает за рассылку уведомлений об изменении игрового поля (перемещение/добавление/удаление юнитов), передачу действий пользователя (мышь и клавиатура) игровым объектам, обработку корректного удаления/добавления объектов, отрисовку поля и вызов функций отрисовки у подписанных объектов. Добавление и удаление объектов возможно только через *GameController*.

*Cell* – элемент сетки игры, клетка. Содержит информацию о ландшафте в клетке, положении клетки а также объектах, находящихся в данной клетке.

*GameController* – мост между доской и объектами. Содержит методы для создания объектов поля (юнитов и нейтральных объектов), добавления и удаления элементов с поля (вызывая затем соответствующие методы в *GameBoard*, если вызов корректен: например, при добавлении элемента необходимо убедиться, что в целевой клетке отсутствует объект). При необходимости взаимодействия объектов поля между собой (например нанесение урона) действие также проходит через *GameController*.

*MouseTracker* – как следует из названия, класс предназначен для отслеживания действий пользователя при помощи мыши. На текущий момент единственным классом, использующим *MouseTracker,* является *GameBoard*. Данный класс позволяет отслеживать перемещения мыши в удобном формате, отслеживая смещения мыши относительно последней позиции и нажатия левой клавишей мыши.

*GameObject* – базовый класс для всех объектов поля. Отвечает за хранение своего состояния (привязан ли к доске) и позиции ячейки, в которой он находится в данный момент. *GameObject* предоставляет ряд полезных интерфейсов (*BoardListener*, слушатели состояния привязки) и обязательных к реализации абстрактных методов (отрисовка, обработка нажатий клавиатуры и мыши).

*Unit* – базовый класс для юнитов: объектов, которыми может манипулировать пользователь. Обладает такими характеристиками, как: здоровье, скорость, атака. Может перемещаться по полю.

*Neutral* – базовый класс для нейтральных юнитов. Пользователь не может влиять на нейтральные юниты. Каждый *Neutral* обладает радиусом действия. Если *Unit* попадает в зону действия, на него накладывается определенный эффект, который наследуется от *NeutralEffect*.

*Terrain* – класс ландшафта. Каждой клетке поля (*Cell*) устанавливается определенный тип ландшафта. Terrain обладает следующими возможностями: отрисовка, возможность накладывать эффекты на объекты типа *Unit*.

*Effect* – эффект, который накладывается на объекты типа *Unit*. Имеет возможность изменять любые свойства объекта. По сути эффекты – основной способ взаимодействия с юнитами.

*TerrainEffect* – класс, являющийся наследником *Effect*. По большей части это вспомогательный класс для других эффектов ландшафта. Он отслеживает положение *Unit*-a, к которому привязан, и, если нет нейтральных объектов подходящего типа, в радиус действия которых попадает целевой юнит, то эффект снимается.

**Классы, дополнительно затронутые в лабораторной работе №4:**

*Loggable* – базовый класс для элементов, которые могут быть записаны в лог. Немного похоже на сериализацию – смысл в том, что такие классы имеют метод *fillLogInfo*, где они заносят всю информацию в специальный класс *LogInfo*, причем делают это последовательно (т.е. включая родительские классы). Данный подход используется во фреймворке *Flutter*, откуда и скопирован.

*LogInfo* – класс, собирающий информацию об объектах. Хранит имя (строка), а также список параметров, который выводится в похожем на *json* формате. Переопределяет *operator<<*, что позволяет удобно записывать его в поток вывода.

*LogAdapter* – база для адаптеров, реализующих подключение к потокам вывода.

*Log* – основной класс, с которым работает приложение. Позволяет выводить в лог строки, объекты *Loggable* и объекты *LogInfo*.

# Особенности лабораторной работы

**Логирование действий пользователя.**

Некоторые действия пользователя (а точнее их последствия) выводятся в лог. Сюда входит в основном логирование действий мыши: изменение *Viewport*-а (изменение позиции, размера, масштаба. Более подробно см. в *Viewport.cpp*). Также записываются в лог события клика мыши – тип (up/down) и координаты.

**Логирование действий юнитов и базы.**

Записываются все изменения юнитов: привязка/отвязка, нанесение урона, перемещение (подробнее см. в *GameController.cpp* и *GameBoard.cpp*)

**Выполнены основные требования к логированию.**

Конечно!

**Реализована возможность записи логов в файл.**

Реализован адаптер *FileLogAdapter*, который позволяет задать файл, куда будет производиться логирование.

**Реализована возможность записи логов в терминал.**

Реализован адаптер *ConsoleLogAdapter*.

**Взаимодействие с файлами должны быть по идиоме RAII.**

Создан класс *LogSession* (см. файл *FileLogAdapter.cpp/.hpp*).

**Для логирования состояний перегружен оператор вывода в поток.**

Да, *LogInfo* как раз имеет перегруженный оператор.

**Переключение между разным логированием (логирование в файл, в терминал, без логирования) реализуется при помощи паттерна “Прокси”.**

Фактически класс Log является тем самым прокси.

**Реализован разный формат записи при помощи паттерна “Адаптер”**

Да, базовый абстрактный класс *LogAdapter*, классы-наследники – *FileLogAdapter*, *ConsoleLogAdapter*.

# Пути к классам

**BaseUnitAttackBehaviour** - \include\GAME\engine\behaviour\BaseUnitAttackBehaviour.hpp

**BaseUnitClickBehaviour** - \include\GAME\engine\behaviour\BaseUnitClickBehaviour.hpp

**BaseUnitMoveBehaviour** - \include\GAME\engine\behaviour\BaseUnitMoveBehaviour.hpp

**BlackHole** - \include\GAME\engine\units\BlackHole.hpp

**BlackHoleEffect** - \include\GAME\engine\units\BlackHole.hpp

**BoardListener** - \include\GAME\engine\BoardListener.hpp

**BoardView** - \include\GAME\engine\graphics\BoardView.hpp

**Cell** - \include\GAME\engine\Cell.hpp

**CellClickBehaviour** - \include\GAME\engine\behaviour\CellClickBehaviour.hpp

**CellDrawer** - \include\GAME\engine\graphics\CellDrawer.hpp

**Chancel** - \include\GAME\engine\units\Chancel.hpp

**ChancelEffect** - \include\GAME\engine\units\Chancel.hpp

**ConsoleLogAdapter** - \include\GAME\log\ConsoleLogAdapter.hpp

**Effect** - \include\GAME\engine\Effect.hpp

**EffectsComparator** - \include\GAME\engine\Effect.hpp

**EffectsSet** - \include\GAME\engine\Effect.hpp

**FileLogAdapter** - \include\GAME\log\FileLogAdapter.hpp

**GameBoard** - \include\GAME\engine\GameBoard.hpp

**GameController** - \include\GAME\engine\GameController.hpp

**GameObject** - \include\GAME\engine\GameObject.hpp

**GridDrawer** - \include\GAME\engine\graphics\GridDrawer.hpp

**GroundTerrain** - \include\GAME\engine\terrains\GroundTerrain.hpp

**Heal** - \include\GAME\engine\units\Heal.hpp

**HealthDrawer** - \include\GAME\engine\graphics\HealthDrawer.hpp

**Home** - \include\GAME\engine\units\Home.hpp

**LavaTerrain** - \include\GAME\engine\terrains\LavaTerrain.hpp

**Log** - \include\GAME\log\Log.hpp

**LogAdapter** - \include\GAME\log\LogAdapter.hpp

**Loggable** - \include\GAME\log\Log.hpp

**LogInfo** - \include\GAME\log\LogInfo.hpp

**MouseTracker** - \include\GAME\engine\MouseTracker.hpp

**Neutral** - \include\GAME\engine\Neutral.hpp

**NeutralEffect** - \include\GAME\engine\NeutralEffect.hpp

**SeaTerrain** - \include\GAME\engine\terrains\SeaTerrain.hpp

**ShapeDrawer** - \include\GAME\engine\graphics\ShapeDrawer.hpp

**Stone** - \include\GAME\engine\units\Stone.hpp

**Terrain** - \include\GAME\engine\Terrain.hpp

**Unit** - \include\GAME\engine\Unit.hpp

**UnitAttachBehaviour** - \include\GAME\engine\behaviour\UnitAttachBehaviour.hpp

**UnitMoveBehaviour** - \include\GAME\engine\behaviour\UnitMoveBehaviour.hpp

**Viewport** - \include\GAME\engine\graphics\Viewport.hpp

# Запуск приложения

Проект собирается при помощи VisualStudio2017 и, насколько я знаю, не требует дополнительных разрешений/установки библиотек. Для запуска можно использовать дебажную сборку, находящуюся в ${ProjectRoot}/Debug/SimpleGame.exe. Программа использует дополнительные библиотеки (SFML), однако они находятся внутри проекта, так что приложение должно запуститься корректно.

# Вывод

При выполнении лабораторной работы были изучены различные паттерны проектирования, изучены основные способы работы с потоками вывода, реализованы различные способы записи информации (логирования).