**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Объектно ориентированное программирование»**

Тема: **Сериализация состояния программы**

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

Санкт-Петербург

2020

**Задание**

Реализация сохранения и загрузки состояния программы. Основные требования:

* Возможность записать состояние программы в файл
* Возможность считать состояние программы из файла

|  |  |
| --- | --- |
| Выполнены основные требования к сохранению и загрузке | 4 баллов |
| Загрузка и сохранение должно выполняться в любой момент программы | 5 баллов |
| Взаимодействие с файлами должны быть по идиоме RAII | 1 балл |
| *\*Сохранение и загрузка реализованы при помощи паттерна “Снимок”* | *5 баллов* |
| *\*Реализован контроль корректности файла с сохраненными данными* | *5 баллов* |
| **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:**

*Serializer* – вспомогательный класс, упрощающий работу с потоками ввода/вывода. Позволяет вести побайтовую запись переменных и некоторых базовых классов.

*InObjectsTable* – таблица объектов, которые были считаны из файла. Содержит объект *map<ObjectInfo, void\*>*, что позволяет получать созданные объекты по их идентификатору.

*OutObjectsTable* – таблица записанных объектов. Хранит информацию о том, какие объекты уже были сериализованы, чтобы избежать их повторной сериализации.

*ObjectInfo* – информация о сериализуемом объекте. Хранит его идентификатор (ссылку на объект) а также поле *needWrite*, которое указывает, находится ли сериализованный экземпляр объекта после сериализованного *ObjectInfo* в файле.

*GameSerializer* – класс, обеспечивающий запись в файл/чтение из файла. По сути это 2 функции read/write, где открывается поток и вызываются функции записи/чтения *GameBoard* и получившийся файл подписывается.

*FileSigner* – класс, предоставляющий возможность подписывать файлы и проверять корректность подписи.

# Особенности лабораторной работы

Сама по себе сериализация реализована максимально просто. *GameBoard* является корневым объектом, который хранит ссылки на все остальные объекты, которые нужно сериализовать. У экземпляра *GameBoard* вызывается метод *writeObject(istream&, InObjectsTable&)*, в котором *GameBoard* сохраняет все свои поля, включая массивы объектов, рекурсивно вызывая аналогичные функции для объектов и записывая элементарные поля (*int*, *float*, структуры и т.д.) при помощи *Serializer*.

При считывании объектов, которые используются из нескольких мест, происходит следующее:

1. Считывается идентификатор *ObjectInfo*.

2. Если поле *needWrite* идентификатора равно *false*, значит сериализованный объект находится в другом месте. В таблицу *InObjectsTable* добавляется слушатель (функция *void(void\*)*), которая вызывается сразу после десериализации объекта. Если же *needWrite* бал равен true, то считывается сериализованный объект и ссылка на него добавляется в таблицу.

Запись объектов ведется аналогичным образом.

**Возможность записать состояние программы в файл**

Да, это происходит по нажатию клавиши *F1*. Файл называется *saved.hehe*, и находится в одной директории с программой.

**Возможность считать состояние программы из файла**

По нажатию клавиши *F3* производится восстановление состояния программы из файла.

**Выполнены основные требования к сохранению и загрузке**

Да.

**Загрузка и сохранение должно выполняться в любой момент программы**

Да, поскольку производится полная запись всех полей.

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

См. класс *GameSerializer*. Сама сессия реализована классом *FileSession*.

**Сохранение и загрузка реализованы при помощи паттерна “Снимок”**

Да

**Реализован контроль корректности файла с сохраненными данными**

Да, контроль корректности производится при помощи класса FileSigner. Фактически он считает сумму 4-х байтовых блоков (не обращая внимания на переполнение) и записывает получившуюся 4-х байтную сумму в конец файла (при этом файл до записи суммы дополняется нулями до кратного 4-м числа байт). При чтении файла проверяется равенство суммы, записанной в последних 4-х байтах, и реальной суммы по файлу.

# 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

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

**FileSigner** - \include\GAME\serialize\FileSigner.hpp

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

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

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

**GameSerializer** - \include\GAME\serialize\GameSerializer.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

**InObjectsTable** - \include\GAME\serialize\InObjetsTable.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

**ObjectInfo** - \include\GAME\serialize\ObjectInfo.hpp

**OutObjectsTable** - \include\GAME\serialize\OutObjetsTable.hpp

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

**Serializer** - \include\GAME\serialize\Serializer.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, способы проверки корректности файла а также сериализация приложения и его восстановление.