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

# ОТЧЕТ

**по лабораторной работе №6**

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

**Тема: Сериализация, исключения**

Студент гр. 0381 _____ Прохоров Б. В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

**Цель работы.**

Изучить основы объектно-ориентированного программирования на языке C++.

**Задание.**

Сериализация - это сохранение в определенном виде состоянии программы с возможностью последующего его восстановления даже после закрытия программы. В рамках игры, это сохранения и загрузка игры.

Требования:

- Реализовать сохранения всех необходимых состояний игры в файл
- Реализовать загрузку файла сохранения и восстановления состояния игры
- Должны быть возможность сохранить и загрузить игру в любой момент
- При запуске игры должна быть возможность загрузить нужный файл
- Написать набор исключений, который срабатывают если файл с сохранением некорректный
- Исключения должны сохранять транзакционность. Если не удалось сделать загрузку, то программа должна находится в том состоянии, которое было до загрузки. То есть, состояние игры не должно загружаться частично.

**Выполнение работы.**

Чтобы выполнить задание, нужно было подключить библиотеку *nlohmann* для работы с *json* файлами, реализовать 2 класса: *SaveButton, LoadButton* и немного модифицировать классы *View, Model, Field* и перечисление *Direction*. Также необходимо было реализовать набор исключений, срабатывающий, если файл сохранения некорректный.

Были добавлены классы новых кнопок: *SaveButton* и *LoadButton*. Они имеют общий интерфейс Button, в каждой переопределён метод запуска. Обе кнопки имеют поле для хранения указателя на объект класса *Model* для того,

чтобы у неё была возможность обращаться к бизнес-логике. Значение поля кнопки получат через конструктор.

Было расширено перечисление *Direction*: появились кнопки сохранения и загрузки сохранения.

Классу *Model*, представляющему собой класс бизнес-логики, необходимо было добавить 2 метода: *get_json_copy()* и *set_json_copy* для того, чтобы была возможность получить, загрузить сохранение и кнопки не делали всю работу самостоятельно, а лишь перенаправляли запрос бизнес-логике.

Классу *Field*, представляющему собой класс поля, необходимо было добавить конструктор, воссоздающий поле из сохранения.

Реализован паттерн Singleton – см. класс *View* и приложение А. Он нужен для гарантии того, что у нас будет существовать только один проектор.

Был реализован набор исключений, срабатывающий, если файл некорректный. Исключение будет вызвано, если:

- в файле будут записаны некорректные данные (характеристики игрока, высота и ширина поля)
- файл будет пуст
- количество врагов на поле не соответствует задуманному
- на поле больше одного игрока, входа или выхода
- на поле есть что-то кроме врагов, предметов, игрока, входа, выхода и стен
- количество записей в файле будет некорректно

Исключения сохраняют транзакционность. Если не удастся совершить загрузку сохранения, то программа останется в том состоянии, которое было до загрузки, однако выведется сообщение о том, что файл сохранения некорректен.

**Выводы.**

Были получены навыки внедрения в приложение сериализации и работы со сторонней библиотекой. Был изучен механизм вызова и обработки исключений.

# ПРИЛОЖЕНИЕ А
# UML ДИАГРАММА

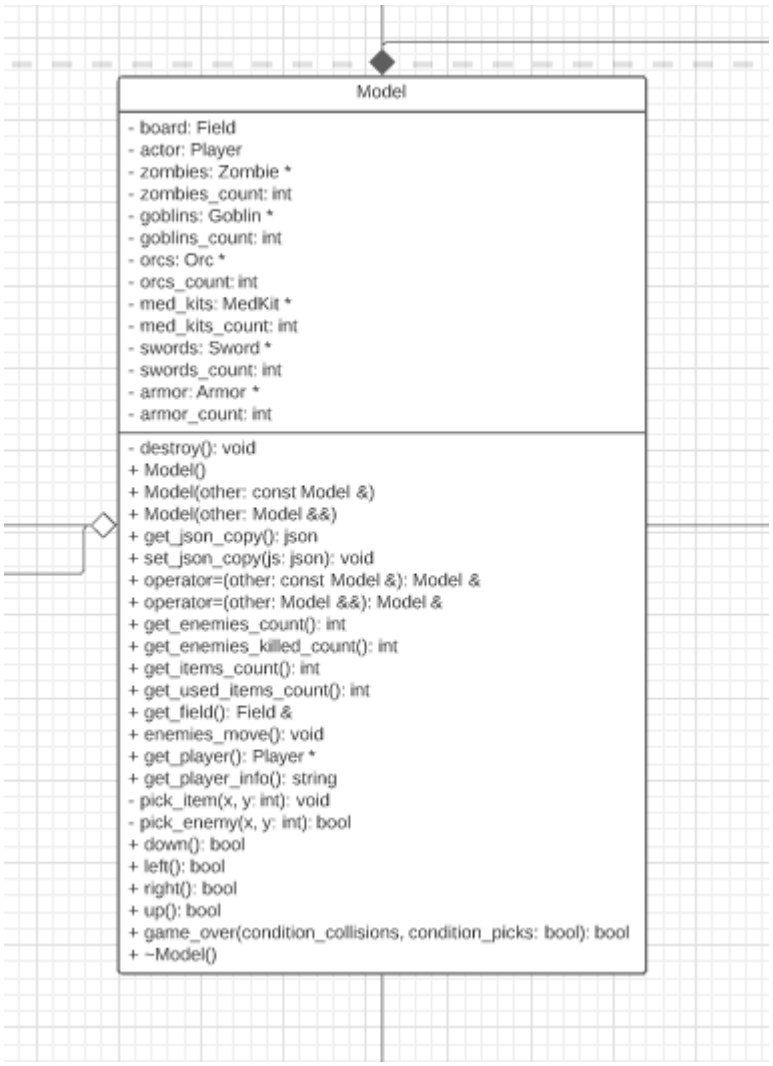


Рисунок 1 – Устройство класса бизнес-логики


Рисунок 2 – Устройство класса поля

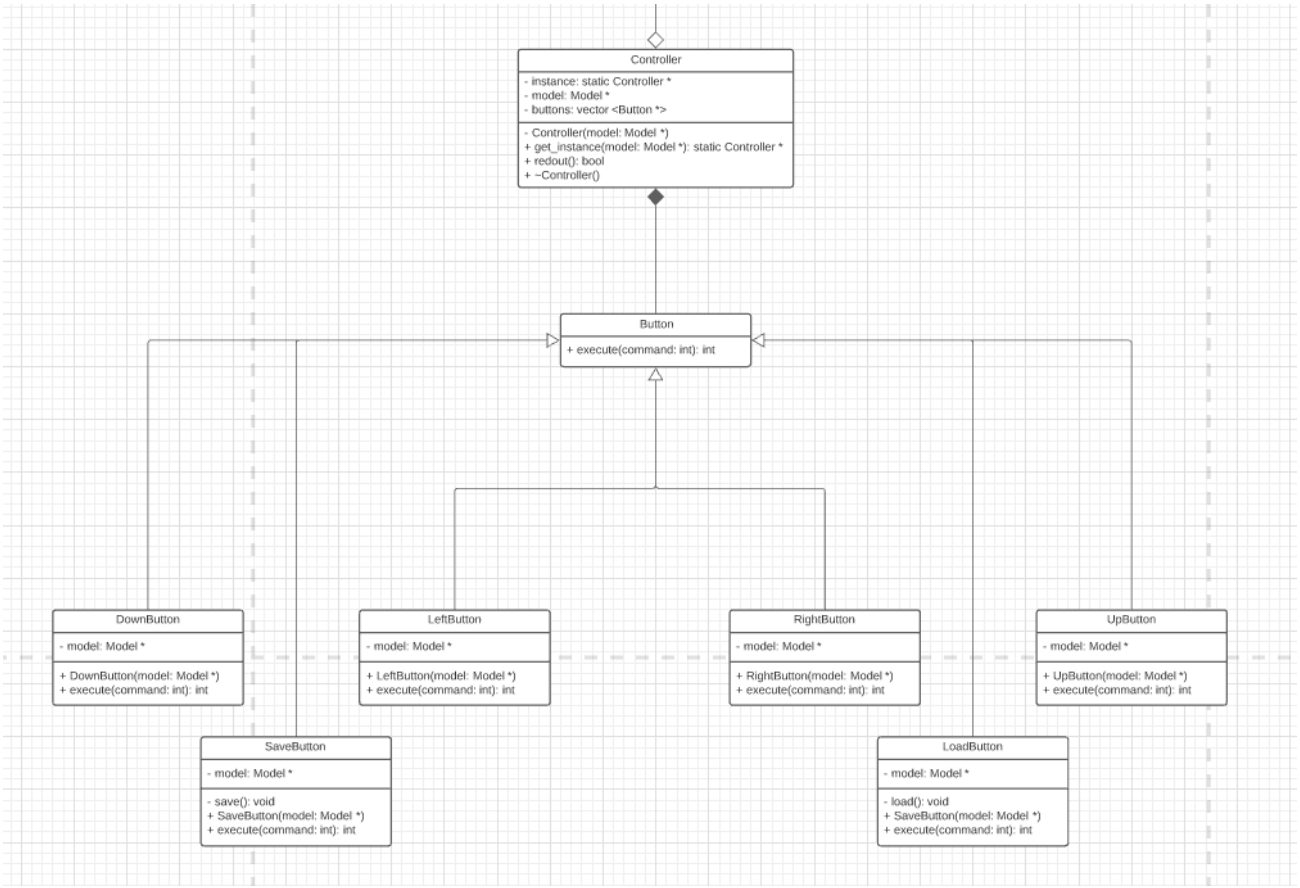


Рисунок 3 – Устройство классов, отвечающих за управление

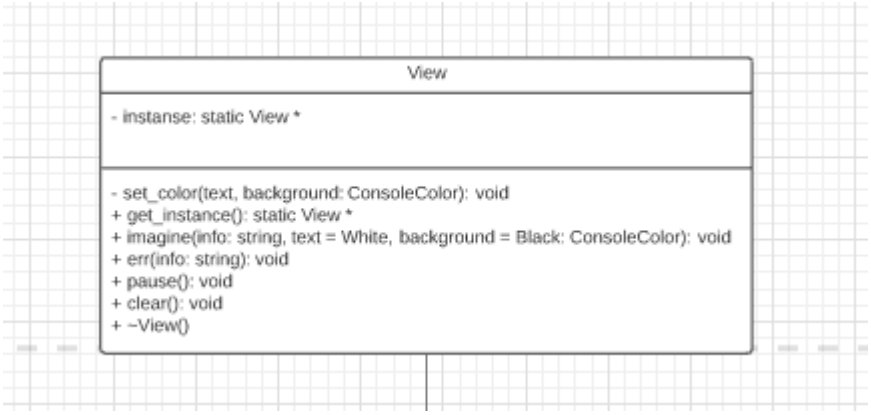


Рисунок 4 – Реализованный паттерн Singleton в классе проектора

Ссылка на полную UML диаграмму: http://surl.li/audlr