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, обработка исключений.

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург 2021

Цель работы.

Создать возможность загрузки-сохранения игры.

Задание.

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

Основные теоретические положения.

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.

Выполнение работы.

Класс Snapshot — интерфейс снапшотов (объектов для сериализации-десериализации других объектов), содержит виртуальные методы serialize и deserialize. В качестве выходных данных для сериализации и входных данных для десериализации используется JSON объект, а конкретно QJsonObject (из модуля Core библиотеки QT). Таким образом объект хранится в виде JSON-объекта (который можно вывести в текст или файл).

Класс EntitySnapshot — класс для (де)сериализации игровых сущностей.

Класс CellSnapshot — снапшот для (де)сериализации клетки поля (вместе с клеткой (де)сериализует и сущность на ней). Использует EntitySnapshot.

Класс FieldSnapshot — снапшот для объекта игрового поля. Использует CellSnapshot.

Класс GameSnapshot — класс для (де)сериализации объектов игры. Использует FieldSnapshot . Сохранённая игра задаётся 4-мя объектами: сам объект поля со всеми клетками и существами, объект логики перемещения врагов, объекты правил игры и позиция игрока для удобства (чтобы не перебирать все клетки поля). То есть имея эти объекты можно однозначно восстановить состояние игры. Для удобства класс GameSnapshot дружественный классу Game.

У применённого подхода (как и принципа сериализации в целом) есть один большой минус, при (де)сериализации полиморфных объектов (таких

как клетки и сущности) приходится прописывать множество условий на типы, что делает программу труднорасширяемой.

Класс SerializeException — исключение выбрасываемое при ошибке во время (де)сериализации объектов.

StartGameDialog — диалоговое окно для выбора параметров генерации игры. Наследуется от QDialog. Нужные параметры задаются в виде чисел при помощи виджета QspinBox (диапазоны ограничены адекватными значениями). При указании недостижимых параметров правил будет выброшено исключение *unreachable condition*. Пример диалогового окна представлен на рисунке 4.

Генерация, загрузка и сохранение игры являются слотами главного окна и вызываются соответственно при нажатии на кнопки меню. В каждом из этих методов производится обработка исключений при помощи try-catch конструкции. При возникновении ошибок высвечиваются соответствующие диалоговые окна.

Загрузка/сохранение игры происходит из/в выбранного(ый) пользователем файл(а). Соответственно сам файл с данными об игре содержит просто JSON объект. Также при запущенной игре неудачная загрузка не закончит текущую игру. А при отсутствии запущенной игры при попытке сохранить игру будет соответствующее объявление в диалоговом окне. Примеры обработки исключительных ситуациий представлены на рисунках 1-3.

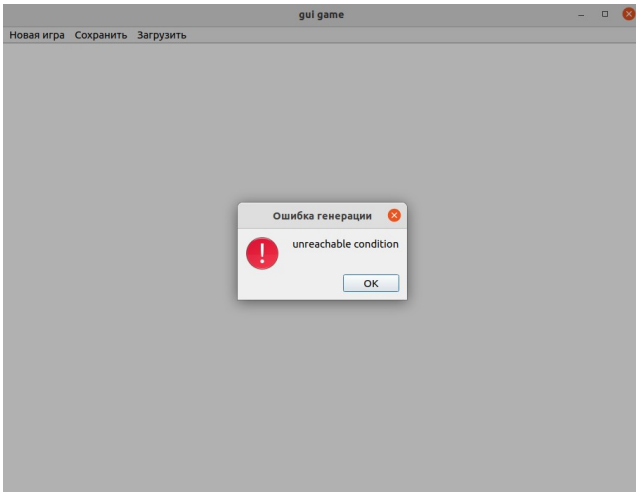

Рисунок 1: Некорректное условие (правило) игры

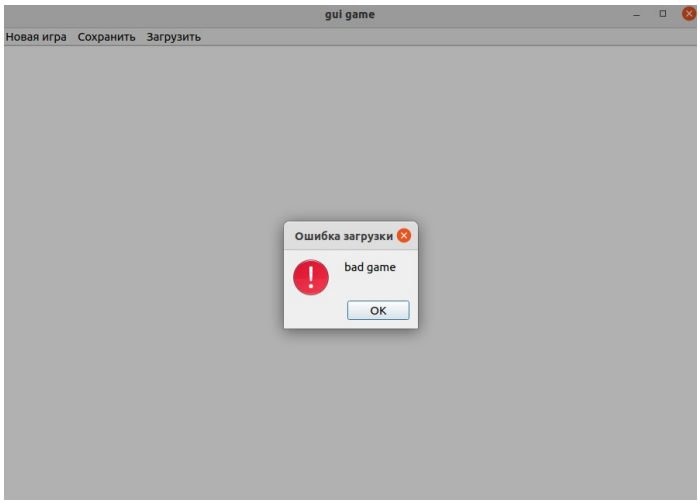

Рисунок 2: Файл не соответствующий JSON объекту

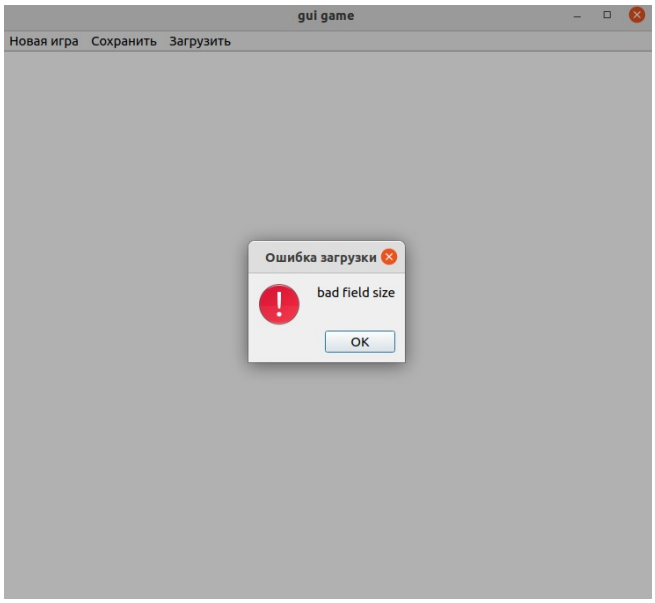

Рисунок 3: В JSON файле были удалены некоторые клетки из поля

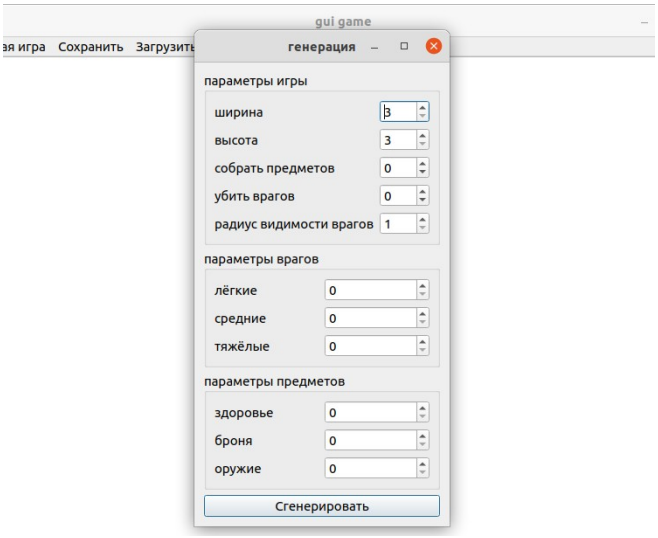

Рисунок 4: Окно параметров генерации

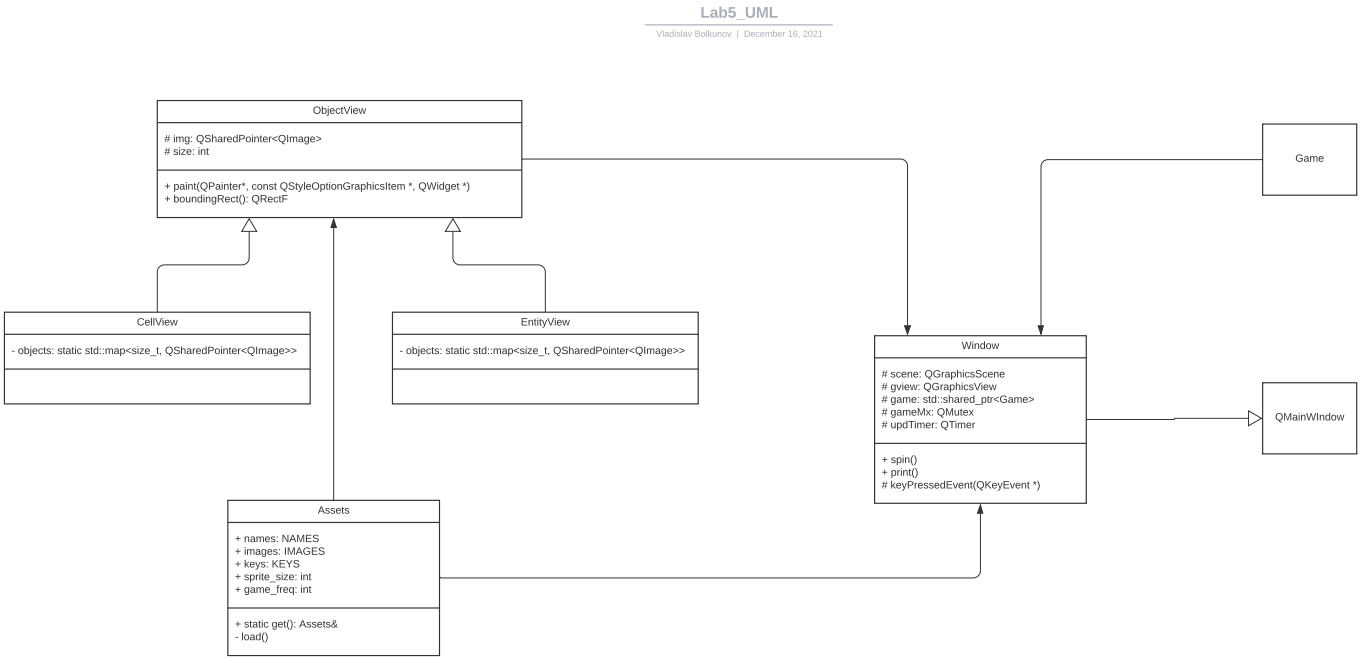

Рисунок 5: UML диаграмма

Разработанный программный код см. в директории */game_lib, /game и /gui*.

Разработанную диаграмму классов UML см. в Lab6_UML.pdf .

Тестирование.

Разработанные тесты см. в директории /tests.

Выводы.

Был реализован набор классов для сохранения и загрузки состояний игры, также реализованы обработки исключительных ситуаций.