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.

Студентка гр. 0382 _____ Охотникова Г.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать сохранение программы в определенном состоянии в файл, загрузку файла и восстановление состояния игры, обработать исключения.

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.

При выполнении данной лабораторной работы были реализованы классы: *class MyExceptiom* и модифицирован *class Game.*

Class MyException:

В данном классе происходит обработка исключительных ситуаций двух типов: ошибка, связанная с некорректным открытием файла и ошибка, связанная с некорректными данными, полученными из файла. В зависимости от того, какая ошибка произошла, выводится соответствующее сообщение об этом с помощью метода *Message().*

Был также модифицирован класс Game, в который были добавлены два новых метода для работы с сохранением данных в файл и выгрузкой из него.

Методы:

- *void save()* — данный метод ничего не принимает на вход и ничего не возвращает. Он осуществляет сохранение текущего состояния игры в файл. Сначала пользователю предлагается ввести название файла, в который будет сохранено состояние игры, затем в цикле осуществляется проход по всему полю. Каждая клетка проверяется на наличие игрока, врага или вещи. Если что-то из этого нашлось, то сохраняются координаты в заранее объявленные объекты структуры *Pair*. Также на каждом шаге происходит запись в файл координат каждой клетки и ее типа.

  Затем проверяется, на каких клетках находятся враги и игрок, и в файл записывается такая информация, как: количество жизней, здоровья, баллов, набранных за игру, есть ли оружие (последние два — только для игрока).

- *Pair upload()* — данный метод ничего не принимает на вход, но возвращает объект структуры *Pair* — сохраненные координаты игрока. Этот метод предназначен для того, чтобы считать данные из файла и восстановить состояние игры. Открывается файл для того, чтобы считать из него данные. Если этого сделать не удалось, выбрасывается исключение из класса *MyException*. Если открытие файла прошло удачно, то считываем размеры поля и создаем новый объект класса *Field*, передавая ему в конструктор полученные значения. Затем проверяем размеры поля и количество типов клетки на корректность. Если полученные данные ошибочны, то выбрасывается исключение из класса *MyException*.

  В цикле каждой клетке присваивается ее тип, считанный из файла, затем считывается информация об игроке и создается новый объект класса игрока, в конструктор передаются полученные

значения: количество здоровья, жизней, есть ли у него оружие, сколько набрано баллов за игру. Его координаты запоминаются и возвращаются в конце метода, чтобы начать передвижение с соответствующего игроку места.

Аналогично обрабатывается информация о злодеях и вещах. Затем в конце файл закрывается и создается объект класса *CreateField* на основе считанных данных. Можно продолжить игру.

UML-диаграмму см. в приложении А.

Выводы.

Было реализовано сохранение состояния игры и его восстановление, а также обработка исключений.

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА