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. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Изучить пользовательские исключения в языке C++. Реализовать систему сохранения и загрузки игры, покрыть ее пользовательскими исключениями.

Задание.

Реализовать систему классов позволяющих проводить сохранение и загрузку состояния игры. При загрузке должна соблюдаться транзакционность, то есть при неудачной загрузке, состояние игры не должно меняться. Покрыть программу обработкой исключительных состояний.

Требования:

- Реализована загрузка и сохранение состояния игры
- Сохранение и загрузка могут воспроизведены в любой момент работы программы.
- Загрузка может произведена после закрытия и открытия программы.
- Программа покрыта пользовательскими исключениями.
- Пользовательские исключения должны хранить полезную информацию, например значение переменных, при которых произошло исключение, а не просто сообщение об ошибке. Соответственно, сообщение об ошибке должно учитывать эти поля, и выводить информацию с учетом значений полей.
- Исключения при загрузке обеспечивают транзакционность.
- Присутствует проверка на корректность файла сохранения (Файл отсутствует; в файле некорректные данные, которые нарушают логику; Файл был изменен, но данные корректны с точки зрения логики.

Примечания:

- Исключения должны обрабатываться минимум на фрейм выше, где они были возбуждены
- Для реализации сохранения и загрузки можно использовать мементо и посетителя
    - Для проверки файлов можно рассчитывать хэш от данных.

Выполнение работы.

В ходе выполнения работы добавлены новые классы, некоторые старые изменены.

1) Определяется класс *Hashing* для определения и сохранения данных.

Реализуется виртуальный метод класса с модификатором доступа *public:*

- *void make_hash(std::string path);* - сохранение данных. Открывается файл с сохранёнными данными, далее считывается каждый символ из файла, хэшируется и записывается.
- *int hash(char c);* - хэш-функции для символа. Берем ASCI код символа и считаем его остаток от деления на 90.
- *int check_hash(std::string path);* - проверка на изменение файла. Внутри метода реализовано сравнение хэша данных с хэшом из файла.

2) Определяется класс *Save_Load_Field*, который отвечает за сохранение и загрузку данных поля: клетки, события, координаты.

Реализуются методы класса с модификатором доступа *public*:

- *void save__data_field(Field* field);* - Сохранение данных поля. Открывается файл, в который будет происходить запись данных, далее записываем размеры поля, типы клеток (через перечисление *OBJECT*). В конце закрываем файл и записываем хэш.
- *PrintField* load_data_field(Player* player, InfoLog* log_out_info);* - Загрузка данных поля. Открывается файл и происходит проверка на изменения внутри файла, если были, то выбрасывается исключение и сообщение об ошибке. Далее присваивание соответствующим клеткам соответствующие типы, считанные из файла.

Реализуются методы класса с модификатором доступа *private*:

- *std::ofstream file_input;* - файл для записи.

- *std::ifstream file_output;* - файл для считывания.
- *std::string filepath = "Sfield.txt";* - имя файла.

3) Определяется класс *Save_Load_Player,* который отвечает за сохранение и загрузку данных игрока: координат, здоровья, брони, золота и т. д.

Реализуются методы класса с модификатором доступа *public*:

- *void save_data_player(Player* player);* - Сохранение данных игрока. Открывается файл, в который будет происходить запись данных, далее записываем все необходимые данные. В конце закрываем файл и записываем хэш.
- *Player* load_data_player(Player* player);* - Загрузка данных игрока. Открывается файл и происходит проверка на изменения внутри файла, если были, то выбрасывается исключение и сообщение об ошибке. Далее присваивание соответствующим характеристикам игрока данных из файла.

Реализуются методы класса с модификатором доступа *private*:

- *std::ofstream file_input;* - файл для записи.
- *std::ifstream file_output;* - файл для считывания.
- *std::string filepath = "Splayer.txt";* - имя файла.

4) Было создано семейство классов исключений, наследованных от класса *GameExp*, у которого есть чистый виртуальный метод *what*(), конструктор, принимающий строку и поле типа *string*. Его реализуют классы *FileExp*, *LoadExp*, *SaveExp*. В каждом из них в методе *what* возвращается строка, состоящая из префикса, характерного 5 для каждого из класса исключений и постфикса — переменной *message* которая задается из аргумента конструктора.

5) Также был дополнено перечисление *MOVES* элементами *SAVE* и *LOAD*, который предназначены для сохранения и загрузки данных с файлов. В классе *Controller* при обнаружении соответствующего элемента

перечисления запускается процесс по загрузке/сохранению данных.

Архитектура программы.

В классе *Controller* реализовано управление игроком и обработка команд, введенных пользователем. При вводе ‘e’ – подразумевается сохранение текущей игры, ‘l’ – загрузка последнего сохранения. При загрузке данных из файлов все данные извлекаются (через try catch проверяется существование данных и файлов). Перед выгрузкой данных хэш из файла сверяется с хэшем от данных. При их отличие также бросается исключение.

Пользователь может сохранять и загружать игру в любой момент времени, при его желании.

UML-диаграмма представлена на рис. №1.

UML диаграмма.

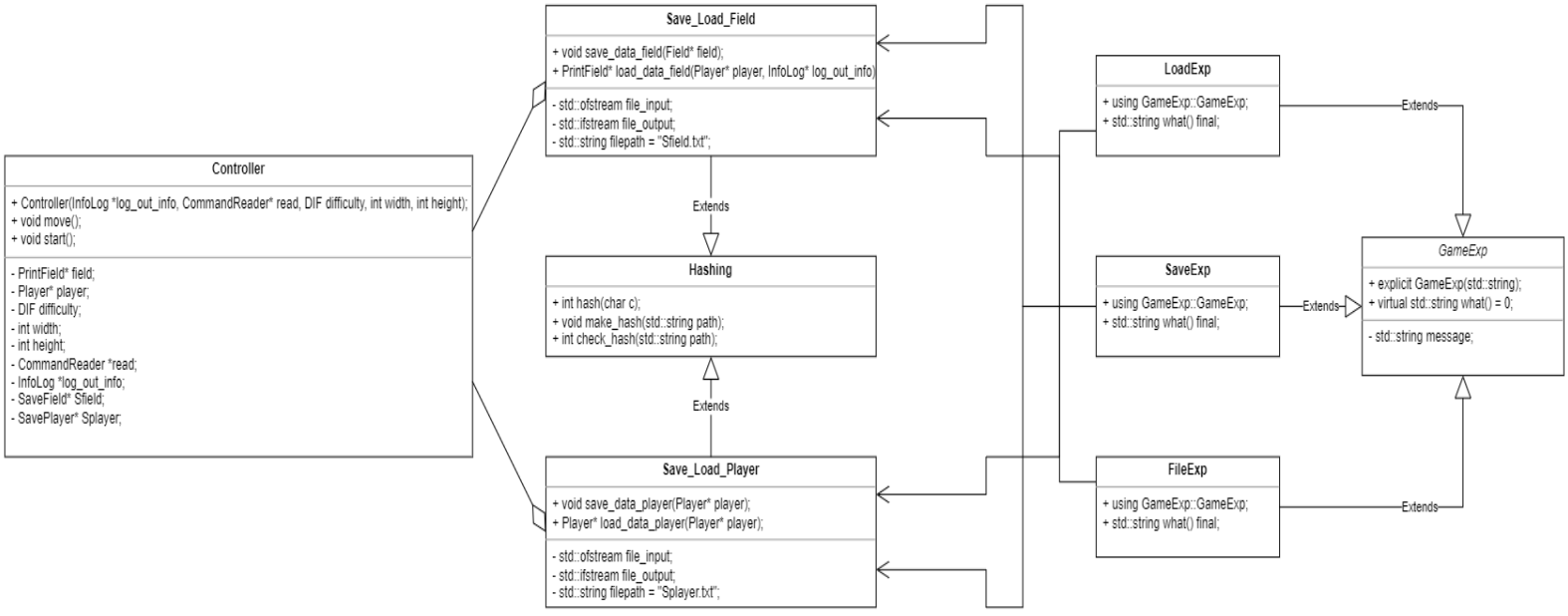

Рис 1 – UML-диаграмма.

Выводы.

В ходе выполнения работы изучена сериализация и создание пользовательских исключений. Реализована система сохранения и загрузки игры, покрытая пользовательскими исключениями.