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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «ООП»
Тема: Сериализация, исключения

Студентка гр. 1383 Седова Э.А

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Реализовать систему классов позволяющих проводить сохранение и загрузку состояния игры.

Задание.

При загрузке должна соблюдаться транзакционность, то есть при неудачной загрузке, состояние игры не должно меняться. Покрыть программу обработкой исключительных состояний.

Теребования:

- 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 значения переменных при которых произошло исключение, а не просто сообщение об ошибке. Соответственно, сообщение об ошибке должно учитывать это 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

Выполнение работы.

В классе *Player* реализованы методы *reloadPlayer()* и *savePlayer(),* сохраняющие и восстанавливающие прошлое сохранение игрока.

В *savePlayer()* текущие поля *healthPoint*, *armor* и *key* записываются в вектор *mas* и в файл “*SavePlayer.txt”*. Для дальнейшего чтения между значениями полей при записи проставляется символ “|”. Также высчитывается хэш записываемых данных в поле *hashPlayer,* с помощью функции стандартной библиотеки *hash().*

В *reloadPlayer()* данные сначала считываются из файла. С помощью метода *del()* они записываются в вектор *mas*. Также высчитывается хэш считанных данных в поле *hashPlayerfield*, с помощью функции стандартной библиотеки *hash().* Далее, с помощью метода *isEqual()*, сравнивается полученный хэш с изначальным. Если они не равны вызывается исключение и выгрузка сохранения игрока не производится.

Позже, если не было вызвано исключение, полям игрока присваиваются соответствующие им значения элементов вектора в методе *reloadCorrectState()*.

В классе *Field* реализованы те же методы, что и в *Player*. Важным отличием является то, что в методе *reloadField()* данные выгружаются в вектор векторов *arr*. Также реализован новый метод *recoverCell()*. Он возвращает изменённым в ходе игры клеткам исходное состояние (состояние во время сохранения).

В классе *Mediator* реализованы два новых метода *saveGame()* и *reloadGame(),* сохраняющие и выгружающие последнее сохранение соответственно. В них вызываются нужные методы у *player* и *field,* отлавливаются исключения из этих методов.

Тестирование программы.

На рисунках 1 - 4 представлено тестирование программы

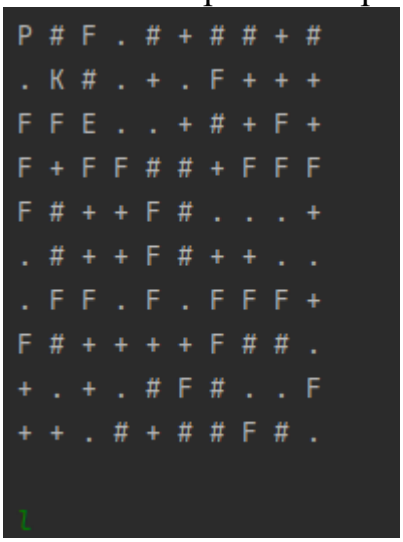

Рисунок 1 – Сохраняем игру

Рисунок 2 – Проходим часть уровня

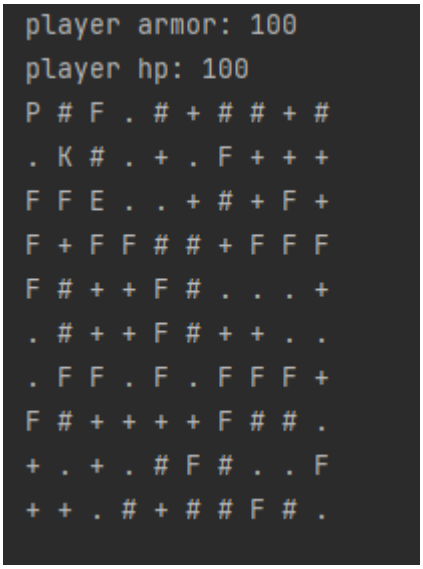

Рисунок 3 – Выгружаем сохранение

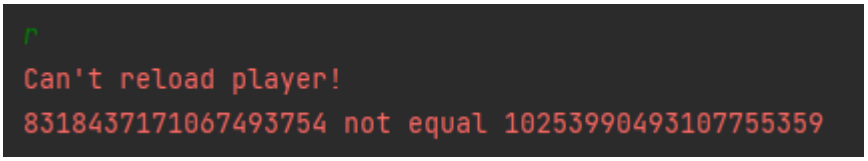

Рисунок 4 – Ошибка, при попытке изменить данные в файле “SavePlayer.txt”

UML-диаграмма межклассовых отношений.

На рисунке 5 изображена UML Диаграмма классов

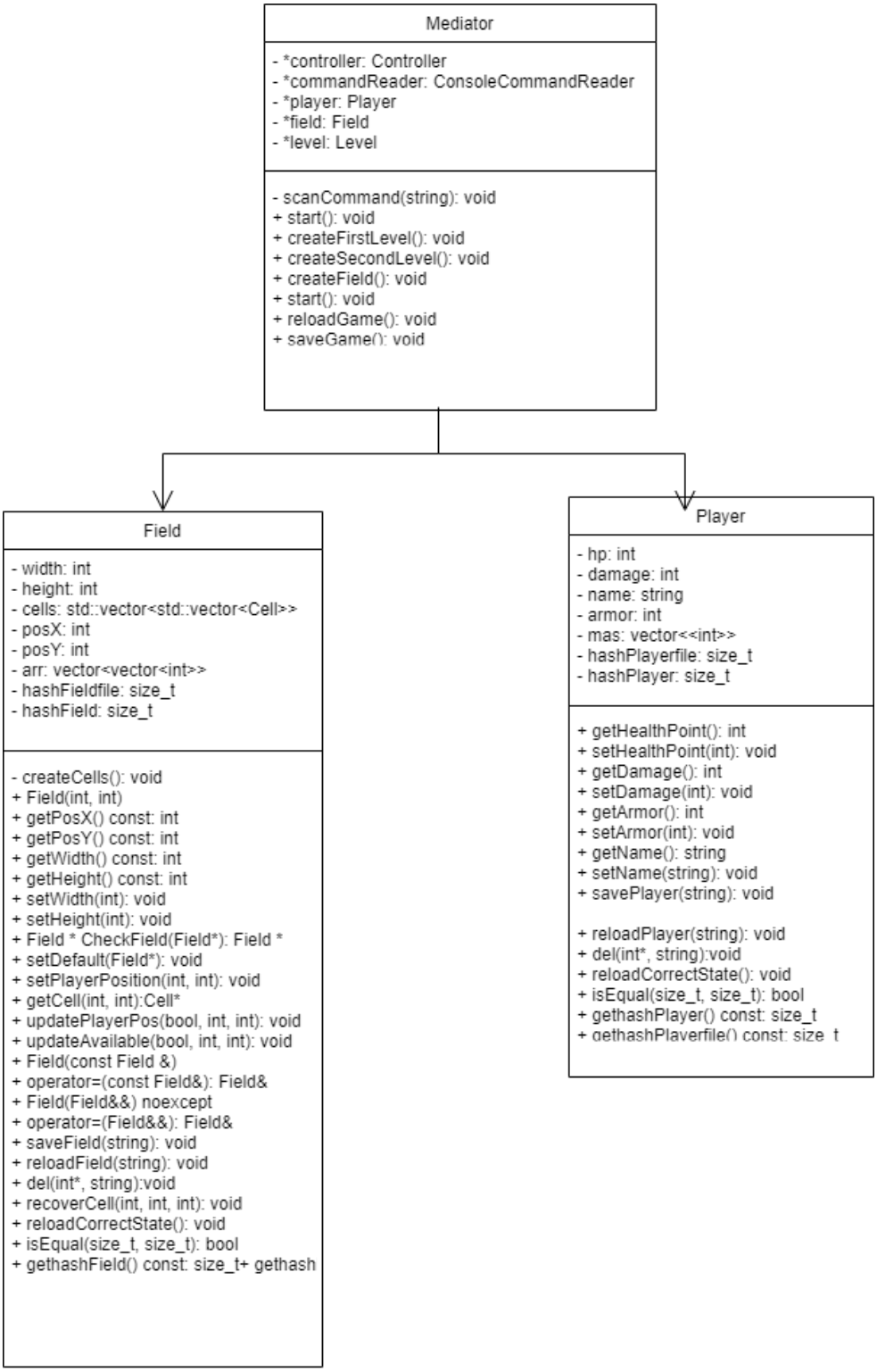

Рисунок 5 – UML Диаграмма классов

Выводы.

В ходе выполнения работы реализована система классов позволяющая проводить сохранение и загрузку состояния игры.