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, сключения

Студент гр. 1303 _____ Коренев Д.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

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

Задание.

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 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

• Для реализации сохранения и загрузки можно использовать мементо и посетителя

• Для проверки файлов можно рассчитывать хэш от данных.

Выполнение работы.

Для выполнения лабораторной работы использован паттерн Memento, сохраняющий состояние объекта, чтобы позднее восстановить его состояние.

Используемые классы:

1) Memento. Имеет метод saveState и restoreState. Первый принимает на вход две строки: содержание, которое надо сохранить и название в файла, в который надо сохранить состояние. Открывается файл, проверяется получилось ли его открыть (если нет — прокидывается исключение OpenFileException с сообщением невозможности открыть файл и его название), записывается состояние и закрывается. Метод restoreState принимает строку — название файла, открывает его, проверяет, получилось ли его открыть (если нет — прокидывается исключение OpenFileException с сообщением невозможности открыть файл и его название), данные считываются и возвращаются.

2) Originator — интерфейс с тремя чистыми виртуальными методами: saveState сохраняет состояние, обращаясь к Memento, restoreState восстанавливает сохраненные данные, restoreCorrectState восстанавливает состояние объекта из восстановленных данных. Классы Player и Field должны реализовать этот интерфейс, так как они являются объектами, чьи состояния необходимо сохранить.

3) Класс Player реализует интерфейс Originator. Метод saveState создает экземпляр Memento, передает ему состояние полученное от метода createSaveState и название файла. Метод CreateSaveState получает хэш от

параметров, необходимых сохранить, записывает в строку, добавляет значения параметров. Метод hash получает переменные, хэш от которых он должен вернуть, получает хэш от каждого, а затем используя xor и битовые смещения применяет для хэшей параметров, возвращает полученный хэш. Метод restoreState получает экземляр Memento запрашивает у него данные (вызвав метод restoreState с аргументом названия файла), передает считанные данные методу restoreData. Метод restoreData принимает данные которые нужно попытаться восстановить. Из данных получает значение хэша и значения параметров, приводит параметры к типу int, вызывает hash от них и, если хэш в файле совпадает с хэшем восстановленных данных, сохраняет их в переменную restoredData, иначе пробрасывает ошибку RestoreStateException с сообщением отличающихся хэшей. Также, если во время извлечения данных файла произошла ошибка, она перехватывается и пробрасывается исключение OpenFileException с сообщением некорректности данных файла, строка файла и ее номер, на которой произошла ошибка. Метод restoreCorrectState восстанавливает корректные данные в поля Player.

4) Класс Field аналогично классу Player имеет методы saveState, createSaveState, hash, restoreState, restoreData, restoreCorrectState, выполняющие аналогичные действия, однако применимые для класса поля, рассмотрим сильно отличающиеся классы. CreateSaveState создает данные с параметрами поля: размер, позиция игрока, позиция финиша, количество монет, поле. Чтобы преобразовать поле в данные созданы словари, первый для клеток, второй для событий. Ключом является хэш код от идентификатора класса, а значением строка, которая и будет помещена в выходные данные. Для этих данных так же нужно вычислять хэш, чтобы избежать внешнего вмешательства в файл. Метод hash получает хэш код параметров поля, а объектов самого игрового поля (клетки и события)

получает хэш, побитово смещает на определенное значение и прибавляет к переменной хранящей промежуточный хэш. Метод restoreData считывает параметры полей, клетки и события, записывает в временную переменную, преобразовывая названия клеток и событий в соответствующие им объекты. Для этого созданы словари с ключем — название клетки/события, а значение — лямбда функция создающая объект. Все ошибки считывания отлавливаются и побрасывается исключение OpenFileException с строкой и ее номером в файле, где была отловлена ошибка. Если данные удалось считать, берется хэш этих данных и сравнивается с хэшем записанного в файл, если они отличаются, побрасывается исключение RestoreStateException с сообщением отличающихся хэшей, иначе данные записываются в временную переменную. Метод restoreCorrectState восстанавливает полученные данные из временной переменной в поле.

5) Класс model был изменен, теперь в нем есть методы saveGame и restoreGame. Метод saveGame сохраняет состояния игрока и поля, отлавливает ошибки и пробрасывает их снова. Метод restoreGame восстанавливает состояние игры, он вызывает метод restoreState у игрока и поля и если ошибки не было вызывает метод restoreCorrectState у обоих. Таким образом если хотя бы один из объектов не получилось восстановить, другой не будет восстановлен. Перехваченные ошибки побрасываются снова.

6) Метод notify в классе Controller был покрыть обработкой исключений, при обработке команд, ошибки логируются.

7) Было создано семейство классов исключений наследованных от класса GameException, у которого есть чистый виртуальный метод what(), конструктор, принимающий строку и поле типа string. Его реализуют классы OpenFileException, RestoreStateException, SaveStateException. В каждом из них в методе what возвращается строка, состоящая из префикса, характерного

для каждого из класса исключений и постфикса — переменной message которая задается из аргумента конструктора.

Тестирование.

Тест 1 — сохранение и восстановление состояния без выхода из игры представлено на рисунке 1.

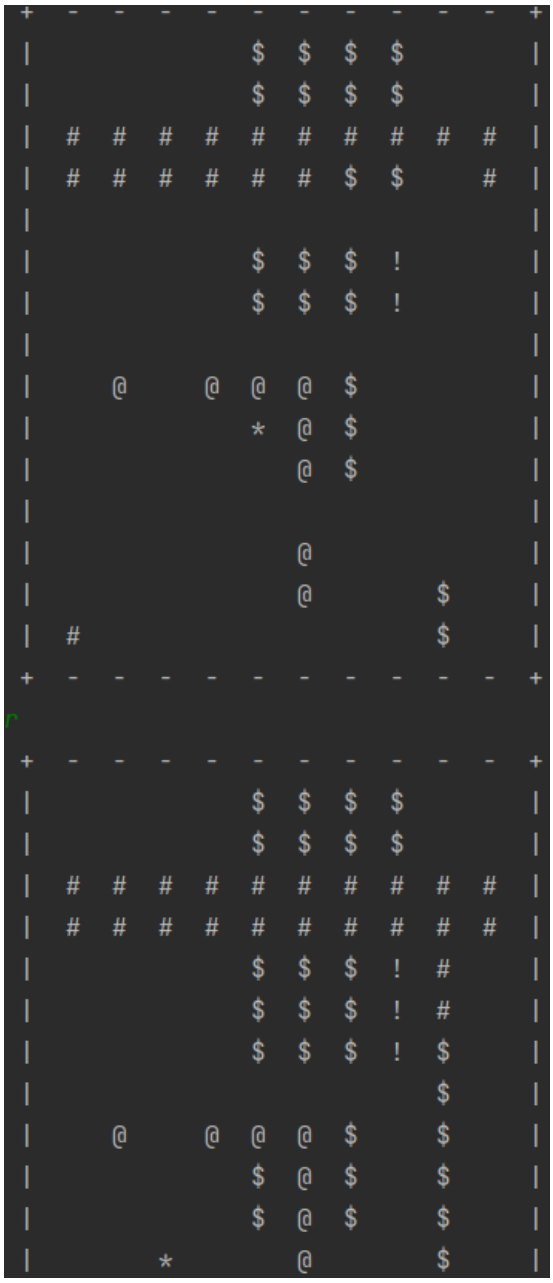

Рисунок 1 восстановление
состояния без выхода

Тест 2 — восстановление данных сразу после запуска игры представлено на рисунке 2.

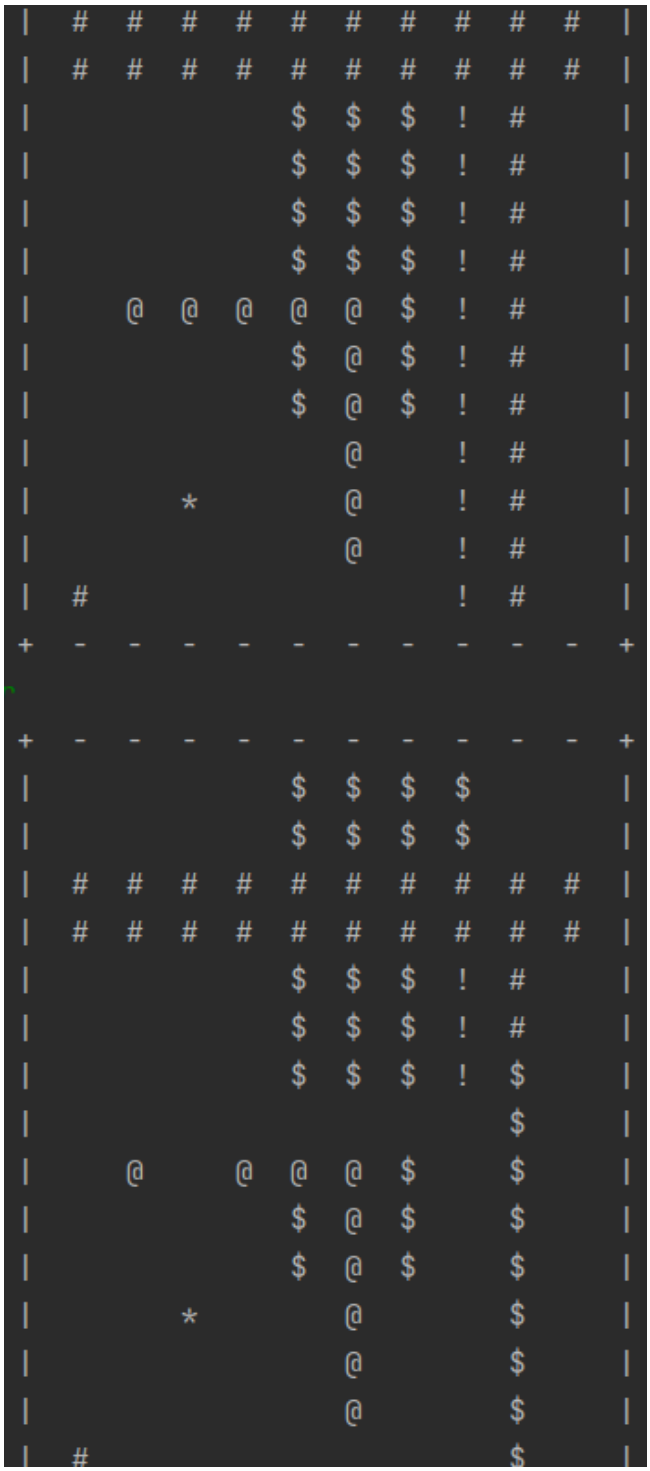

Рисунок 2 восстановление данных после перезапуска программы

Тест 3 — изменение данных файла поля и попытка восстановить такое состояние представлено на рисунке 3.

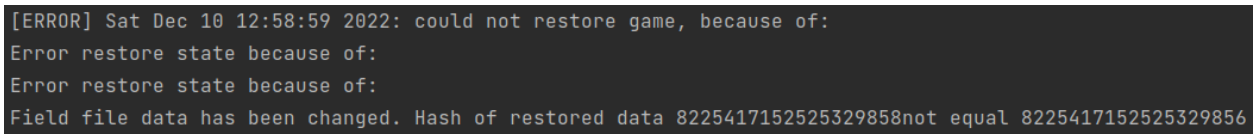

Рисунок 3 попытка восстановить измененный файл

Тест 4 — файл сохранения игрока не существует, однако было вызвано восстановления состояния представлено на рисунке 4.

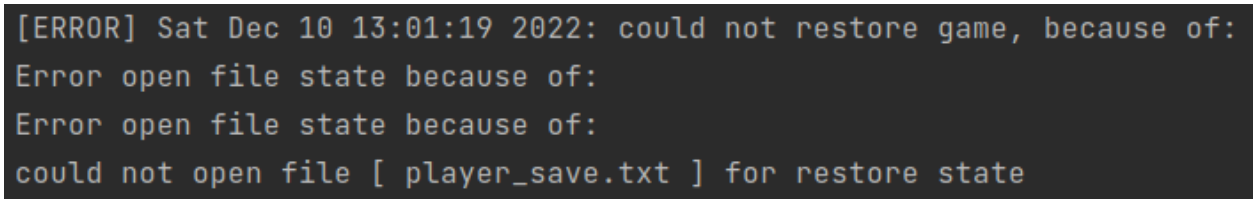

Рисунок 4 попытка восстановить данные из несуществующего файла

UML-диаграмма

UML-диаграмма межклассовых отношений представлена на рисунке 5.

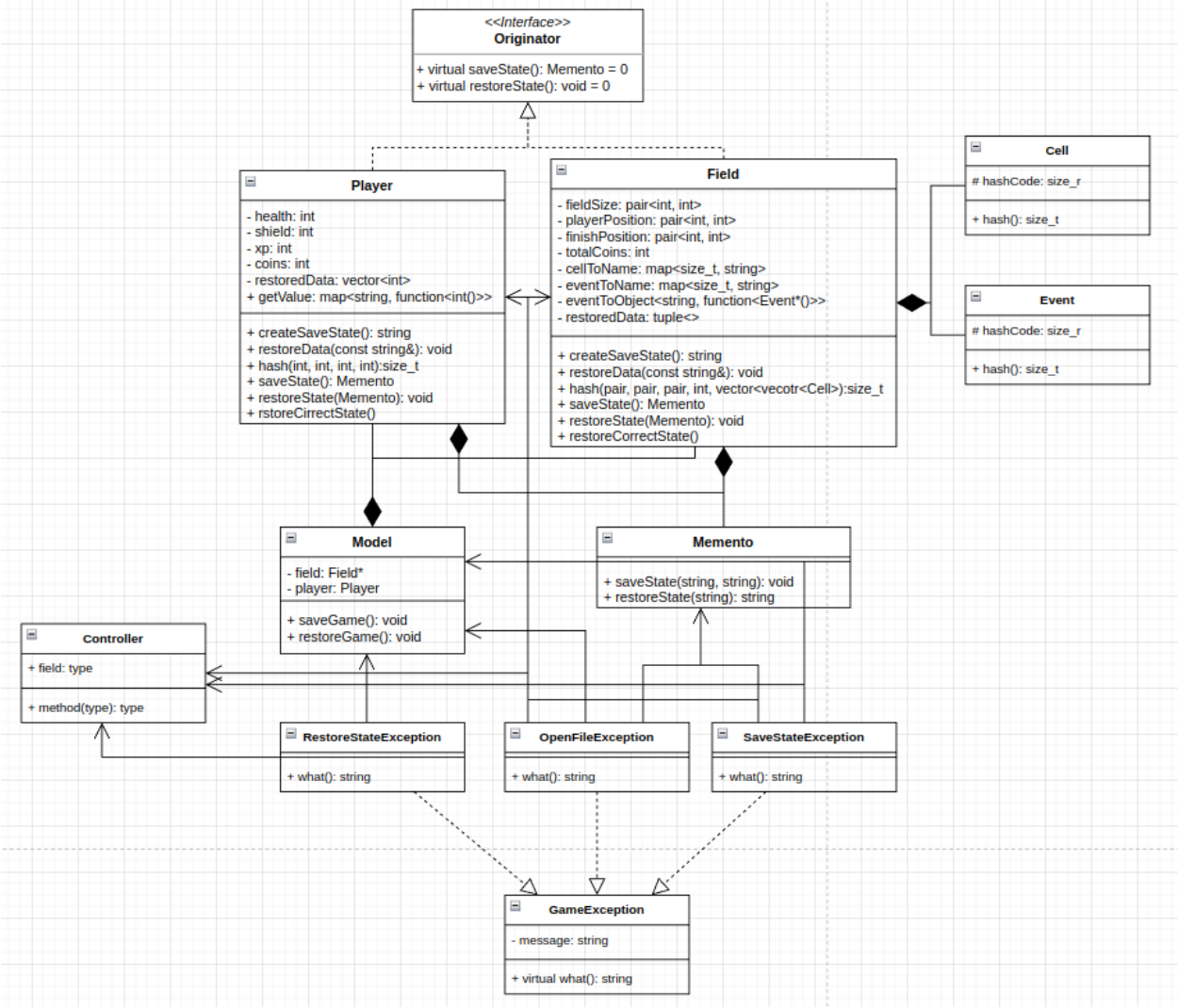

Рисунок 5 диаграмма межклассовых отношений

Выводы.

Реализованы сохранение и загрузка игрового процесса. Программа покрыта исключениями. Изучена работа с классами, паттерны проектирования, основы составления UML-диаграмм.