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. 1304 _____ Чернякова В.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.

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.

Описание архитектурных решений и классов.

Были реализованы классы, осуществляющие сохранение и загрузку состояние игры. При возникновении ошибки пробрасывается исключение пользовательского типа.

Класс *GameInfo*: класс, хранящий основную информацию об игре. Данный класс хранит две структуры: *PlayerInfo* – основная информация об игроке, *GameInfo* – хранит объекты структур *PlayerInfo* и *FieldScheme*. Для каждой из структур перегружен оператор *std::string* для вывода информации в файл сохранения.

Класс *SaveGame*: класс сохранения игры. Данный класс содержит методы *load()* и *save(),* которые восстанавливает игру, основываясь на сохранении, и сохраняют соответственно.

Save() принимает в качестве аргументов структуру *GameInfo*. В начале происходит проверка на корректность файла *if (!f.is_open())*. Если файл некорректен, то выбрасывается пользовательское исключение, иначе происходит сохранение игры. Из структуры *GameInfo* считывается состояние игры и записывает в файл. Состояние игры также кодируется хеш-функцией, чтобы можно было отследить изменения файла. Значение хэша записывается в файл.

Load() сначала считывает из файла строку – поле и переносит его на вектор, затем отдельно считывает строку с информацией об игроке. Затем собирает строку с информацией о поле и игроке, чтобы сопоставить по хэш-функции и убедиться, что файл не был изменен.

Методы *doField()* и *doPlayer()* возвращают из строки объекты структур *FieldScheme* и *PlayerInfo* соответственно.

Класс *SaveException*. Данный класс – посредник исключений. Класс-потомок от *std::exception*. Переопределяет его метод *what(),* возвращающий сообщение. В конструктор подается сообщение и информация об игре с помощью *std::optional<GameInfo>* по умолчанию *std::nullopt.*

Класс *SaveController.* Данный класс содержит методы *load* и *save*, которые переносят абстракции на игровые объекты и наоборот. *Save* – формирует структуры *FieldScheme*, *PlayerInfo* и *GameInfo* из первых двух. Затем при помощи конструкции *try-catch* у *saver(a*) типа *SaveGame* вызывается

метод *save(),* который всю информацию записывает в файл, но если в нем срабатывает какое-либо исключение, то оно логгируется.

Load – возвращает *pair<FieldFacade*, PlayerFacade*>,* восстановленные с «паузы». Так же с помощью конструкции *try-catch* вызывает метод *load()* у *saver(a).*

Класс *GameController().* Класс – наследник от *IController*. Осуществляет срабатывание в игре методов *load()* и *save(),* вызывая их у *application*, который в свою очередь вызывает их у вышеперечисленных классов.

Вывод.

Реализовано несколько классов для сохранений состояний игры, а также их восстановлений во время игры. Приобретено умение создавать пользовательские исключения, пробрасывать их и отлавливать с помощью блоков try-catch.

ПРИЛОЖЕНИЕ.

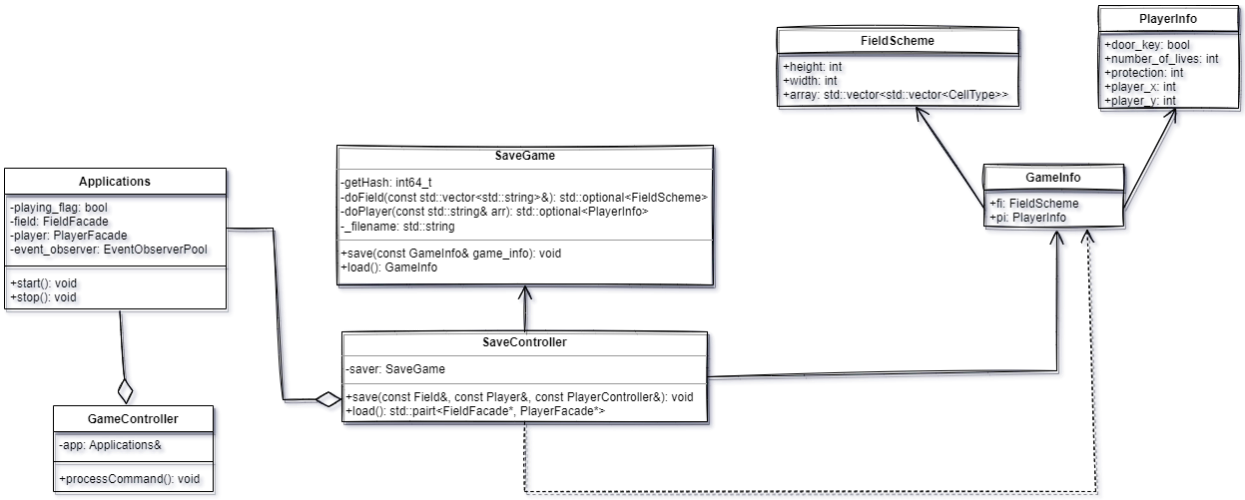

Рисунок 1 – UML-диаграмма межклассовых отношений.