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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Объектно-ориентированное программирование»
Тема: Сериализация, исключения

Студент гр. 1304 ___ Байков Е.С.

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).

Описание архитектурных решений и классов.

Реализованы классы, осуществляющие сохранение и загрузку состояния игры, а также при процессе загрузки и сохранения при возникновении ошибки пробрасывается исключение пользовательского типа *SaverException*. Подробное описание классов:

1. *Saver*: класс производит запись информации в файл о текущем состоянии игры. Метод *save* принимает на вход ссылку на *GameInfo*, где происходит открытие файла на запись, а также включена проверка на успешное открытие файла. При неуспешном открытии кидается исключение *SaverException* с сообщением *"File is beaten"*. При успешном открытии в файл записывается информация с *GameInfo* с помощью переопределенного оператора преобразования к строке для данной структуры, а также записывается хэш сообщения для будущих проверок. В методе *load* происходит обратное преобразование из строк в файле происходит декодирование строки в соответствующие структуры и проверка хэша при декодировании. Если новый и старый хэши не совпадают, то выкидывается исключение с сообщением *"File was changed!"* и передается текущее «битое» сохранение в исключение. При успешном прохождении проверки хэша возвращается информация в виде структуры *GameInfo*.
2. *SaverException*: класс наследуется от *std::exception* и переопределяет его метод *what()*, возвращающий сообщение. В конструктор подается сообщение и информация об игре с помощью *std::optional<GameInfo>* по умолчанию *std::nullopt*. Также создано два геттера для полей класса-исключения.
3. *SaverController*: класс является некоторой прослойкой над *Saver* и осуществляет проверку на запись или загрузку состояния игры. В методе *save* по данным из классов *Field* и *Player* создается структура *GameInfo*, которая после в блоке *try-catch* подается *Saver*, при успехе производится

запись информации в файл, иначе логгер выводит сообщение об ошибке и сообщение самой ошибки. В методе *load* в блоке *try-catch* происходит попытка загрузка информации из файла, при выкидывании исключения происходит проверка на исключение с загруженным файлом, который был изменен извне. Затем пользователь решает хочет ли он загрузить сохранение с битым файлом или нет если не хочет, то возвращается nullptr иначе происходит переинициализация поля.

4. *GameController*: класс который реализует сохранение и загрузку по нажатию клавиш *F5* и *F9*. Является наследником класса *IController*. Загрузка и сохранение по клавише происходит с помощью уже имеющегося медиатора.

Демонстрация работы программы и тестирование.

Для демонстрации работы сохранений запустим игру на 3 уровне см. рисунок 1.

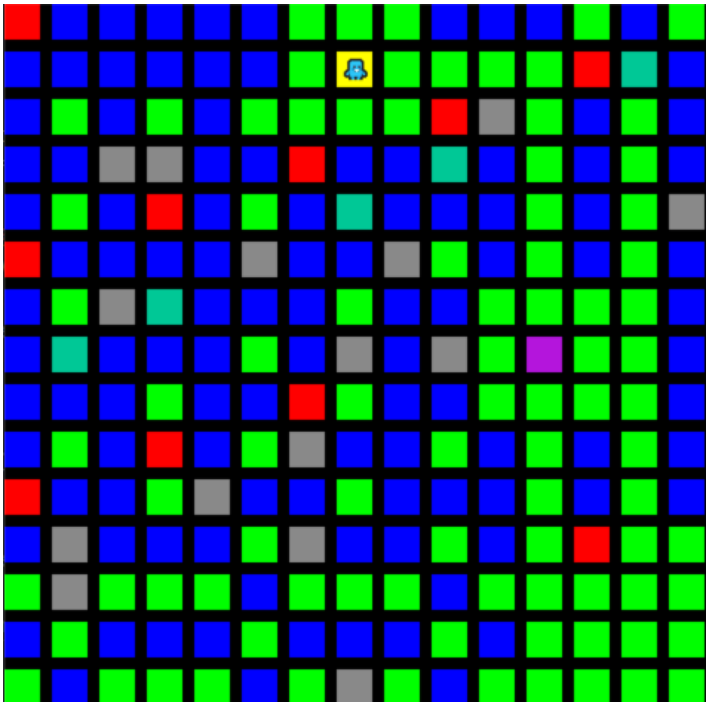

Рисунок 1 — Начало 3 уровня.

Пройдем пару клеток, запуская события, спрятанные в них, и сохраним игру см. рисунок 2.

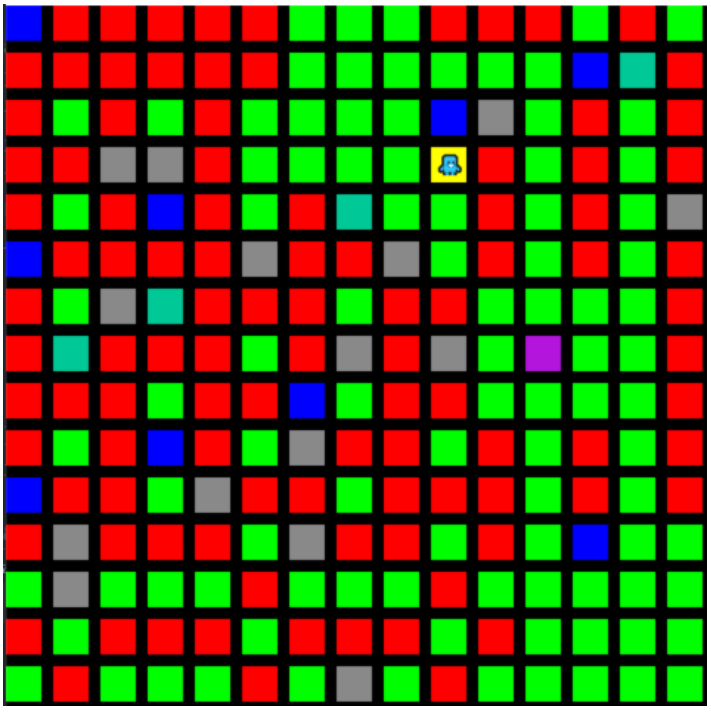

Рисунок 2 – изменение игрового поля посредством действий игрока и сохранение

игры.

Произведем загрузку игры из другой точки см. рисунок 3 и 4.

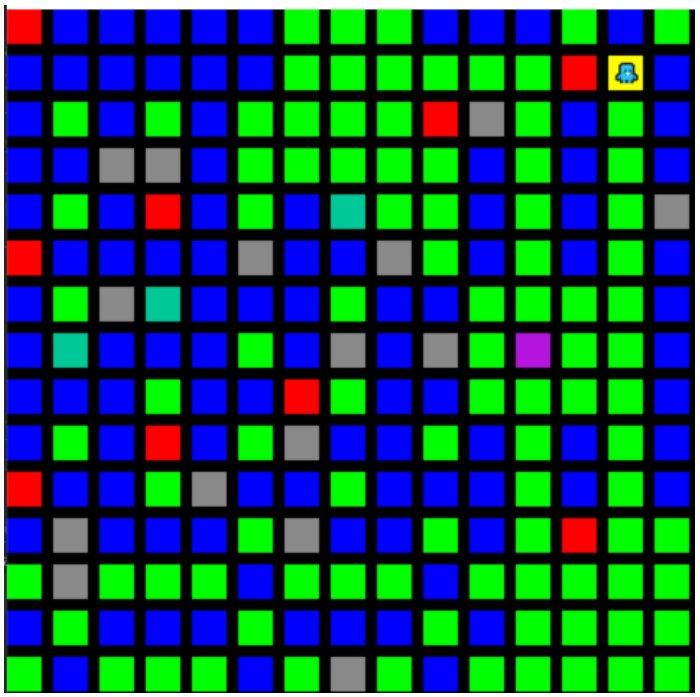

Рисунок 3 – переход на другую клетку.

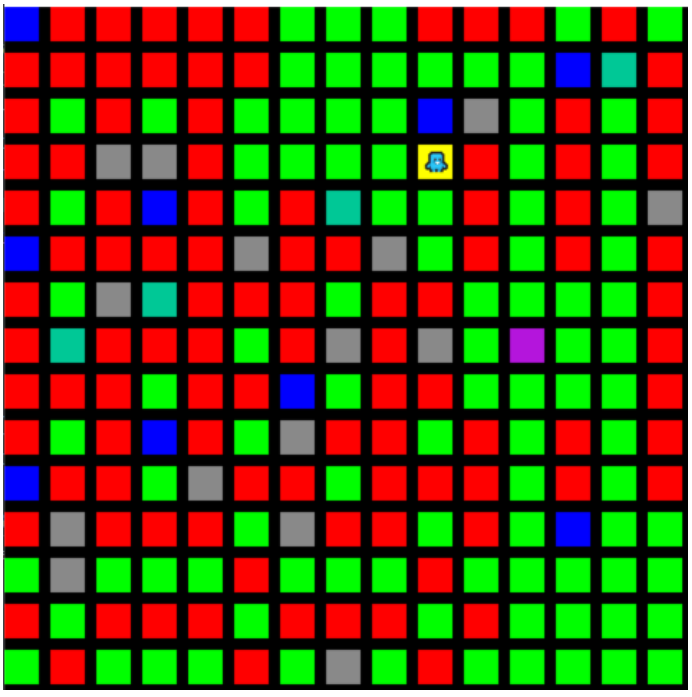

Рисунок 4 – загрузка последнего сохранения после нажатия *F9*.

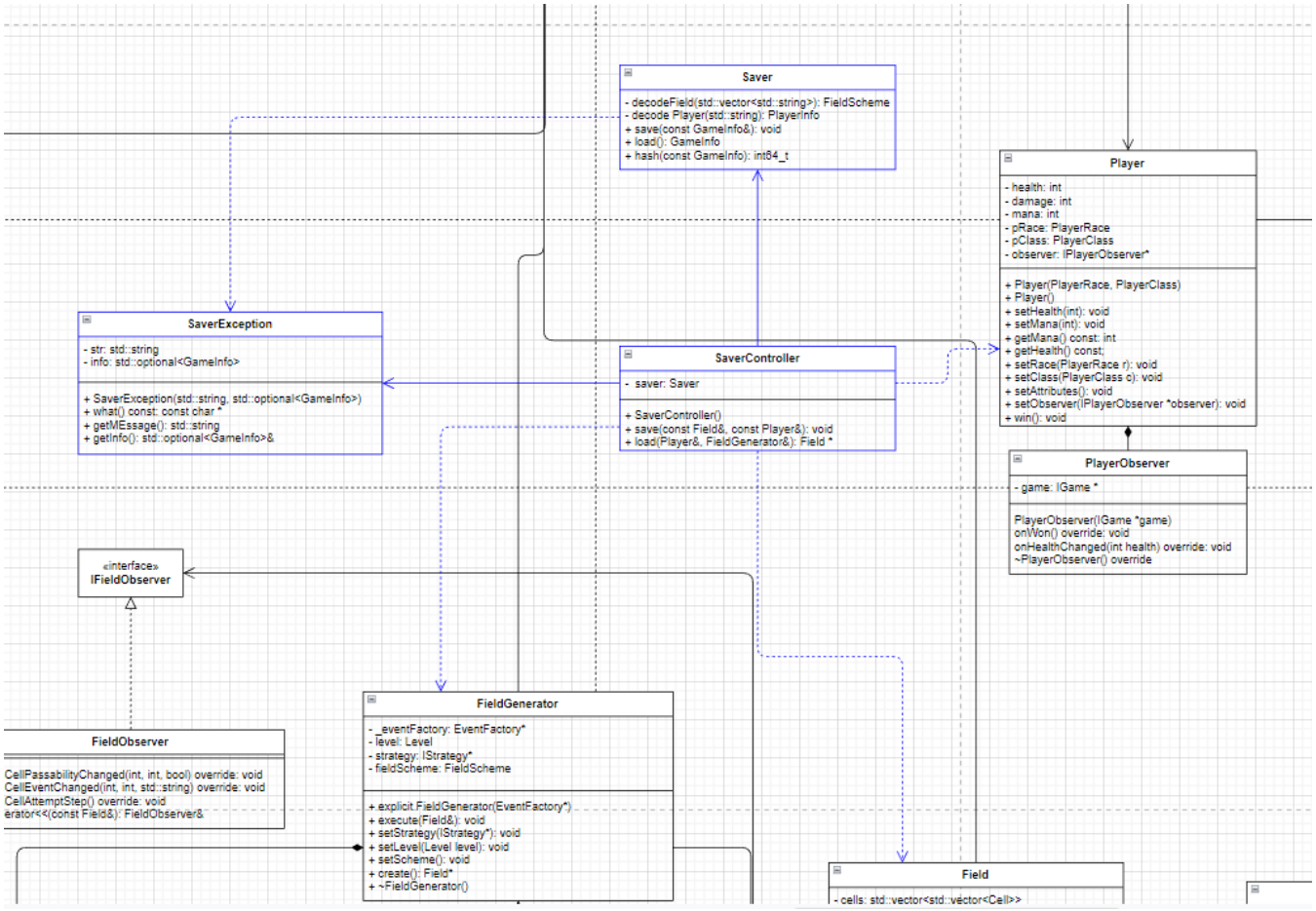

Рисунок 5 — UML-диаграмма классов.

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*.