# 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. 1303 |  | Куклина Ю.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург 2022

# Цель работы.

Реализовать систему классов, позволяющих проводить сохранение и загрузку состояния игры.

# 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.
* Исключения при загрузке обеспечивают транзакционность.
* Присутствует проверка на корректность файла сохранения

# Выполнение работы.

1. **Class GameInfo()**

В данном классе хранится две структуры: PlayerInfo (хранит основную информацию об игроке) и GameInfo (которая в свою очередь хранит объекты стуктур PlayerInfo и FieldScheme). Для каждой из этих структур перегружен оператор std::string для записи в файл.

# Class SaveGame()

Данный класс содержит методы load() и save(), которые восстанавливает игру, основывая на сохранении и сохраняют соответственно. Save принимает в качестве аргументов структуру GameInfo, из которой считывает состояние игры и записывает в файл, кодируя хеш-функцией, чтобы можно было отследить изменения файла. 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. Также здесь обрабатываются и выбрасываются исключения SaveException.

# explicit SaveGame(std::string filename = "Save.txt");

* **void save(const GameInfo& game\_info);**

# GameInfo load();

* **static int64\_t getHash(const std::string& data);**

# static std::optional<FieldScheme> doField(const std::vector<std::string>& arr);

* **static std::optional<PlayerInfo> doPlayer(const std::string& arr);**

# Class SaveException()

Данный класс – посредник исключений. Класс-потомок от std::exception.

# explicit SaveException(std::string message,std::optional<GameInfo> info = std::nullopt);

* **const char \* what() const noexcept override;**

# const std::optional<GameInfo>& getInfo() const noexcept;

1. **Class 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).

# void save(const Field& field, const Player& player,const PlayerController& controller);

* **std::pair<FieldFacade\*, PlayerFacade\*>load();**

# Сlass 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.

# GameController(Applications& applications);

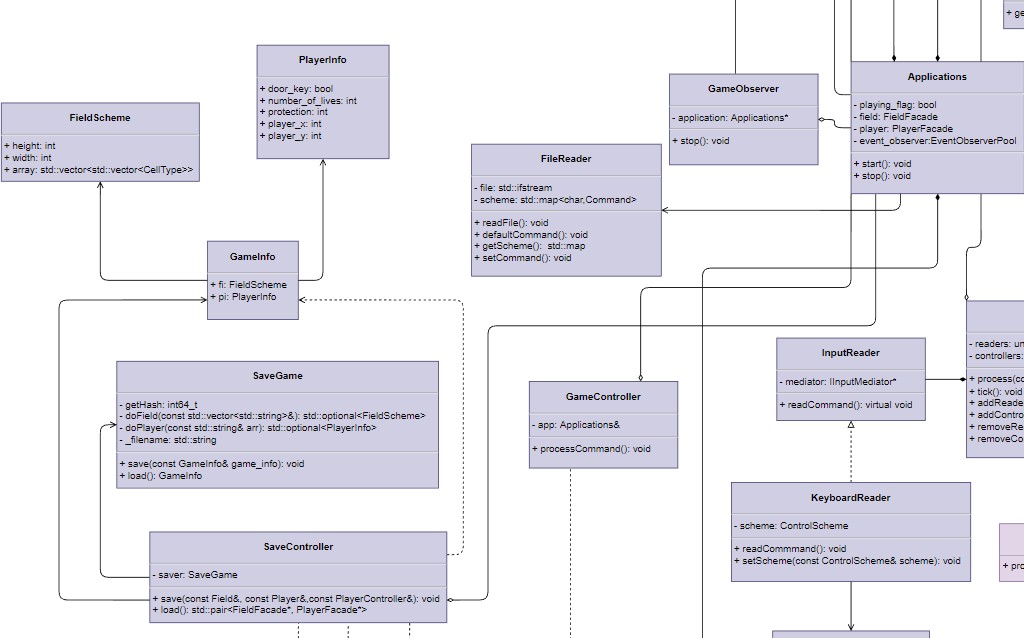
* **void processCommand(const InputMessage& message);**

# Class FieldRegister()

Класс – преобразующий поле field в FieldScheme(). В методе getScheme() создается std::vector<std::vector<CellType>> map по размерам поля. И в каждую клетку записывается значение, возвращаемое методом getCellType у CellRegister.

# static FieldScheme getScheme(const Field& field);

**UML диаграмма классов.**



# Выводы.

Успешно реализована система классов позволяющих проводить сохранение и загрузку состояния игры.