МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Объектно-ориентированное программирование»

ТЕМА: СЕРИАЛИЗАЦИЯ, ИСКЛЮЧЕНИЯ

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать сохранение в определенном виде состояния программы с возможностью последующего его восстановления даже после закрытия программы.

Задание.

Сериализация - это сохранение в определенном виде состоянии программы с возможностью последующего его восстановления даже после закрытия программы. В рамках игры, это сохранения и загрузка игры.

Требования:

- Реализовать сохранения всех необходимых состояний игры в файл
- Реализовать загрузку файла сохранения и восстановления состояния игры
- Должны быть возможность сохранить и загрузить игру в любой момент
- При запуске игры должна быть возможность загрузить нужный файл
- Написать набор исключений, который срабатывают если файл с сохранением некорректный
- Исключения должны сохранять транзакционность. Если не удалось сделать загрузку, то программа должна находиться в том состоянии, которое было до загрузки. То есть, состояние игры не должно загружаться частично

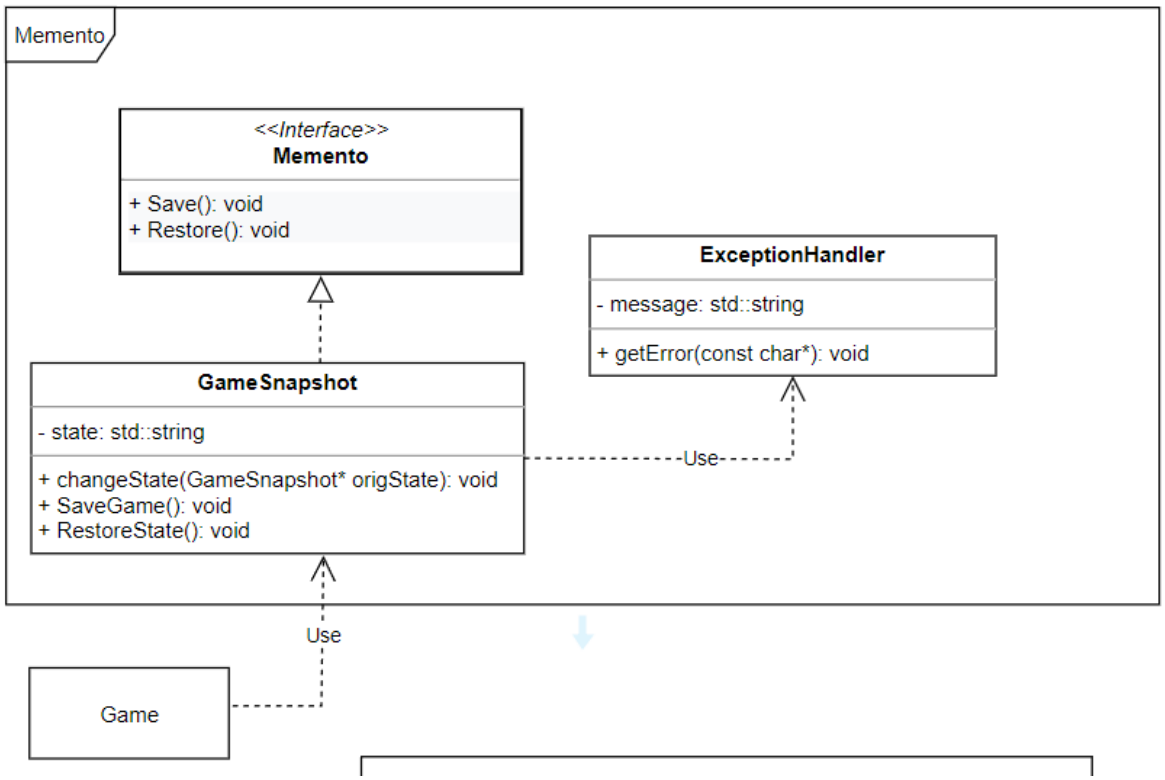
Потенциальные паттерны проектирования, которые можно использовать:

- *Снимок (Memento) - получение и восстановления состояния объектов при сохранении и загрузке*

UML - диаграмма.

UML-диаграмма к данной работе представлена на рисунке 1. Также с диаграммой можно ознакомиться в приложении А.

Рисунок 1 – UML-диаграмма



Выполнение работы.

Краткое описание: при нажатии клавиши Q пользователь делает запрос на сохранение текущего состояния игры. Состояние сохраняется в файл saved_name.txt. При нажатии клавиши E пользователь делает запрос на загрузку сохраненной игры. Она восстанавливается из того же файла saved_name.txt. Сохранение игры и загрузка сохраненной игры будет производиться через класс *GameSnapshot*, наследуемый от интерфейса снимка *Memento*. Для сохранения состояния игры достаточно сохранить всю текущую информацию об игровом поле Map, а именно о содержащихся в нем клетках Cell и их состояниях, об игроке(его координатах, здоровье, наносящем уроне и наличии ключа от выхода), о врагах(их координатах на поле, здоровье и

наносящем уроне), о вещах, лежащих на поле(их координатах). Эти данные мы будем выводить в файл saved_game.txt, посредством файлового ввода через ofstream(реализовано в методе *GameSnapshot::SaveGame()*. А затем извлекать из этого же файла - через ifstream(реализовано в методе *GameSnapshot::RestoreState())*.

Реализуем сериализацию с применением паттерна Снимок.

Для этого сначала создадим интерфейс снимка *Memento*, с помощью которого будем сохранять текущее состояние игры и загружать сохраненное. Методы *SaveGame(), RestoreState()* будут чисто виртуальными, чтобы наследуемые от интерфейса снимки реализовывали данный функционал.

Класс снимка игры *GameSnapshot* будет реализовывать интерфейс *Memento.* Game будет создавать объект этого класса, передавая в конструктор параметры текущего состояния игры. А конструктор будет преобразовывать эту информацию в единую строку-состояние *state* вида: *“size_field/Nstate_cell/Nx/y/Nh_player/Nd_player/Nk_player/Nn_enemies/Nx/Ny/ Ntype_enemy/Nh_enemy/Nd_enemy /N n_items/Nx/Ny/Ntype_item”*. Сформированная строка записывается в поле state класса *GameSnapshot.*

Метод *GameSnapshot::saveGame()* вызывается при нажатии клавиши Q. А так как ранее, уже при создании объекта *GameSnapshot* через конструктор сформировалась и записалась строка-состояние в поле *state.* То внутри метода мы будем обращаться к этому полю. Чтобы записать полученное состояние игры, откроем файл, передадим в поток *ofstream* строку-состояние и закроем файл.

```
std::ofstream saver;
saver.open("saved_game.txt");
saver << state;
saver.close();
```

Метод *GameSnapshot::RestoreState(GameSnapshot* origState)* вызывается при нажатии клавиши E. Он отвечает за загрузку сохраненной игры. Открывается файл с сохраненной игрой, а затем в строгой последовательности

по структуре записанной строки-состояния мы поочередно извлекаем «куски» с необходимой информацией. Сначала размер игрового поля(чтобы знать, сколько клеток считывать во вложенном цикле), затем информацию о каждой клетке(ее состояние:проходимая/непроходимая/..). Далее координаты игрока, его здоровье, урон и наличие ключа от выхода. Затем информацию о врагах, включая их координаты, здоровье и наносимый урон. И последнее – информацию о вещах, лежащих на поле(их координаты). Параллельно с получением необходимого «куска» информации выделяем память под новые параметры игры(н-р, новое поле, новый игрок, враги и т.д). И восстанавливаем соответствующие состояния, согласно полученным данных. Итоговые новые параметры состояния игры «оборачиваем» в структур *GameState*. Вызывем метод *ChangeState(GameState*),* который позволит установить новые значения вместо старых.

Для обработки исключений реализуем класс *ExceptionHandler.* Вызывая конструктор этого класса, будем передавать ему два объекта типа *enum*: *objError* и *Error* с возможными объектами и ошибками (объекты ошибок: cellEr, playerEr, enemyEr, itemEr; сами ошибки: stateEr, coordEr, healthEr, damageEr, keyEr, typeEr). В конструкторе с помощью эти объектов будет формироваться строка *message* '*Not correct data for objEr : Error*'. Если при загрузке состояния *RestoreState()* возникла ошибка, то нам выбрасывает объект этого класса. Если при вызове метода загрузки в *Game* мы отловили ошибку *ExceptionHandler*, то выводим в поток ошибок std::cerr строку с ошибкой, которую получаем с помощью метода getError() из выброшенного методом объекта.

Выводы.

В ходе работы было изучено сохранение и восстановление программы.

ПРИЛОЖЕНИЕ А
UML-ДИАГРАММА