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

Студент гр.0382 _____ Тихонов С.В.

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*

Выполнение работы.

Нужно реализовать сохранение и загрузку игры. Чтобы это сделать в рамках данной игры, достаточно сохранять изменяемые параметры объектов игры, т.е. игра, противников, вещей и игрового поля.

Для сохранения и загрузки был написан класс Save. У него всего два метода Savegame(SavePlaeyr *savePlaeyr, SaveOther *saveOther) и Loadgame(SavePlaeyr *savePlaeyr, SaveOther *saveOther). Их аргументы – это указатели на объекты классов, в которых содержатся указатели на все нужные нам объекты для сохранения]; метод Loadgame вызывается в блоке try/catch (для отлавливания ошибки LoadError). Все данные игры сохраняются в файл Save.txt, который можно загрузить даже при повторном запуске программы

У каждого объекта есть определённые параметры. Поле состоит из клеток определенного типа; игрок имеет тип, здоровье, координату, максимальное здоровье, наносимый урон; вещи имеют тип, координату (поле с данными, что изменяют параметры игрока, константны, поэтому их сохранять не нужно); противники имеют тип, здоровье, координату (остальные параметры также константны). Эти данные мы можем записать числами (типы тоже, поскольку те являются элементами перечислений).

Когда вызывается метод сохранения в файл записываются подряд данные, что получают при помощи метода объектов SavePlaeyr и SaveOther - GetData(). Данный метод возвращает нам строку, содержащую нужные нам цифры. SaveOther, вернет нам строку, в который записано: тип каждой клетке, количество предметов и характеристики каждого, количество врагов и их характеристики Метод SaveHero вернет нам: положение игрока, его максимальное здоровье, текущее, урон. Каждый указанный параметр записывается в строгом порядке с новой строки, что поможет нам отлавливать ошибки при чтении данных для загрузки.

При загрузке считываем данные, в идеальном случае без ошибок. По ним создаем объекты. А потом заменяем их везде, где это нужно (в данном случае,

достаточно поменять указатели на них в объектах классов SavePlaeyr и SaveOther).

Когда вызывается метод загрузки, то выбирается файл, откуда загрузить данные (при команде load1 -> save1.txt; load -> просим ввести название в консоль). Если файл не существует, то выбрасываем ошибку LoadError с соответствующими аргументами. Если же файл открыт удачно, то считываем оттуда информацию. Для считывания данных мы используем оператор >> (файл открыт в потоке). Поскольку мы точно знаем размер поля , то мы точно знаем количество чисел отвечающих за тип клетки. Сначала создаем новый объект поля new_field. Затем, начинаем по очереди считывать типы клеток в заранее объявленную переменную t. затем указываем данный тип соответствующей клетке поля методом SetType(). По завершении чтения типов, согласно описанному ранее порядку сохранения, далее идет информация о вещах. Заранее объявляем переменные Things** new_things (указатель на новый массив вещей), int new_THING (количество новых вещей) и int x,y (для координаты объекта). Сначала считывается число новых вещей в переменную new_THING. Создаем новый массив указателей на объекты Things в количестве new_THING. Затем от 0 до new_THING считывает параметры вещей (сначала тип в переменную type, затем координату x, y). Создаем вещь, согласно типу, помещаем его в new_thing[i], устанавливаем его на координату x,y (если она равна (-1;-1), то вызываем метод GetData() для объекта, это поменяет поле is_available на false; координата (-1;-1) в данной игре говорит о недоступности объекта); если координата не (-1;-1) и по ней не стоит уже объект на новом поле – устанавливаем в соответствующую клетку вещь. Как сказанно выше, что даёт нам клетка хранить не нужно, так как это число не меняется и объявляется в конструкторе.

Далее идет информация о противниках. Так же как и с предметами, сначала идёт их количество, потом характеристика каждого, здесь ещё хранится текущие жизни врага, так как сохранение может быть после того, как его

ударили, но не убили. Так же как и с предметами если враг мёртв помещаем его на координаты (-1,-1) и поле is_alive приравниваем к false.

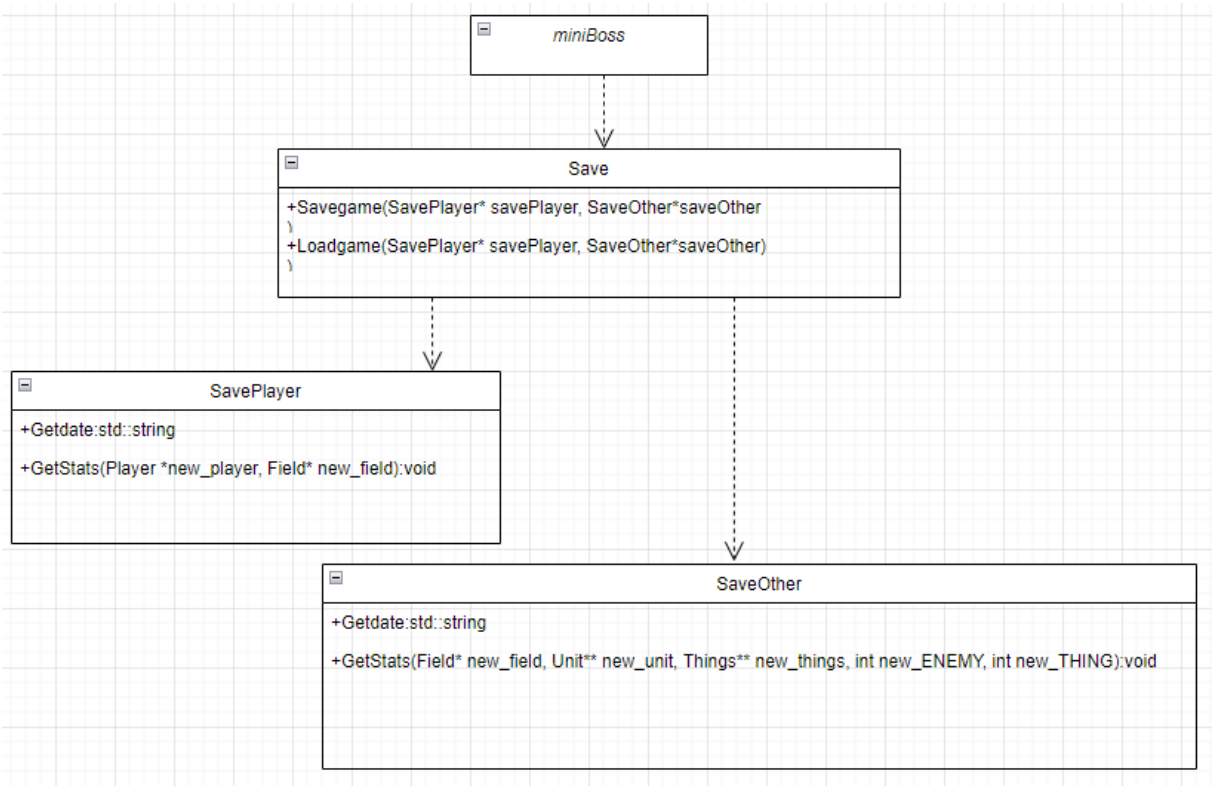
После всего этого остается лишь считать параметры игрока. Объявляем переменные Hero* new_gamer (указатель на объект нового игрока), int demage (для наносимого урона), int max_hp (для максимального здоровья). Затем происходит все аналогично предыдущим описаниям.

Затем новые данные записываются в SavePlayer и SaveOther соответствующими методами.

Для этого были написаны методы GetStats(). Для класса SaveOthers передаются в аргументах new_enemy, new_things, new_ENEMY, new_THING и new_field. В методе мы удаляем старые объекты, а в соответствующие поля (enemy, things, ENEMY, THING, field) устанавливаем новые данные. Также обновляем указатель на массив клеток в объекте draw класса Draw. Для класса HeroPlayer передаются в аргументах new_gamer, new_field. Здесь мы удаляем уже лишь старый объект игрока, затем устанавливаем в соответствующие поля (gamer, field) переданные объекты.

UML-диаграмма классов представлена на рис. 1.

Рисунок 1 – UML-диаграмма классов.



Выводы.

В ходе работы было изучено сохранение данных программы и их загрузка, даже после перезапуска программы.