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 _____ Злобин А.С.

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 находится 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 SaveAndLoad. У него всего два метода Save(Level* level, SaveType type) и Load(Level* level, SaveType type). Их аргументы – это указатель на объект класса, в котором содержатся указатели

на все нужные нам объекты для сохранения; а также элемент перечисления, которое отвечает за выбор файла чтения / записи; метод Load вызывается в блоке try/catch (для отлавливания ошибки LoadError). Передача команды идет по следующей причине: была сделана возможность сохранить в/загрузить файл с названием, что вводится в консоль, а также возможность быстрого сохранения в/загрузки файл/файла стандартного сохранения (имеет название “save1”); для первого варианта это команды save/load, для второго – save1/load1.

Поговорим подробнее о сохранении. У каждого объекта есть какие-то параметры. Поле, например, состоит из клеток определенного типа; игрок имеет тип, здоровье, координату, максимальное здоровье и наносимый урон; вещи имеют тип, координату и значение эффектов действия на игрока; противники имеют тип, здоровье, координату. Эти данные мы можем записать числами (даже типы, поскольку те являются элементами перечислений).

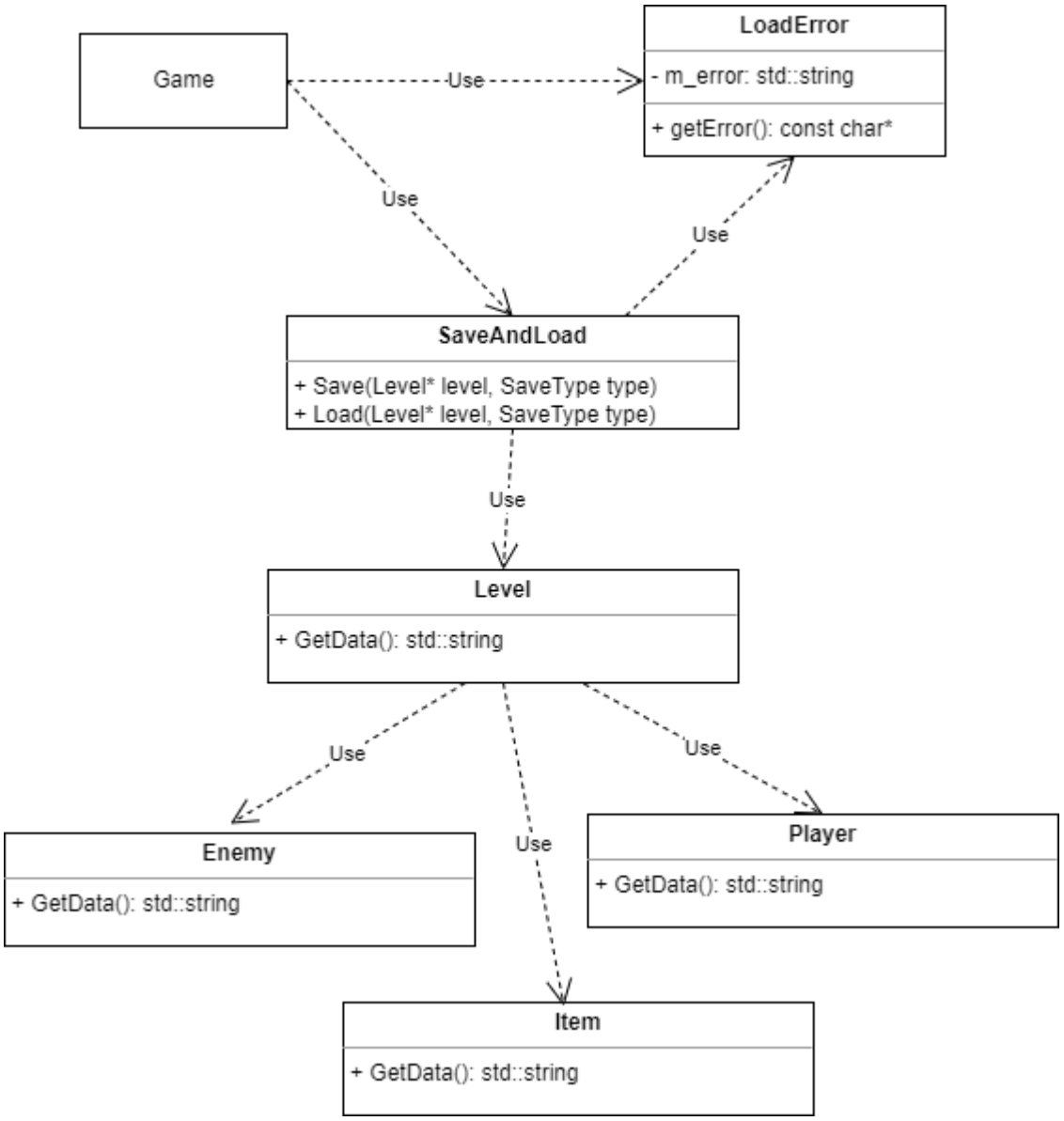
Когда вызывается метод сохранения, то выбирается файл, куда сохранить данные (при команде DEFAULT -> save1.txt; CUSTOM -> просим ввести название в консоль). В файл с указанным названием записываются подряд данные, что получают при помощи метода объекта level (что мы передавали в аргументах) GetData(). Данный метод возвращает нам строку, содержащую нужные нам цифры. Он вернет нам строку, в который записано: а)тип каждого предмета на поле; б) его координаты; в)основные характеристики объекта; Параметры для каждого объекта записываются в строгом порядке с новой строки, что поможет нам отлавливать ошибки при чтении данных для загрузки.

Когда вызывается метод загрузки, то выбирается файл, откуда загрузить данные (при команде DEFAULT -> save1.txt; CUSTOM -> 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 этого мы используем оператор >> (файл

открыт в потоке). Поскольку мы точно знаем размер поля (методы getwidth и getheight класса map), то мы точно знаем количество чисел отвечающих за тип клетки. Сначала создаем новый объект поля new_map, а так же два массива: enemies и items - для хранения массива с врагами и предметами соответсвенно. Затем, начинаем считывать числа из файла сохранения. Первое число каждой строки означает тип элемента и после считывания кастится к перечислению Entity. Затем, в зависимости от полученного типа объета происходят дальнейшие варианты считывания. Для каждого типа объектов существует своё, фиксированное, число необходимых параметров. После чего происходит их считываение, и создаётся объект соответствующего класса со считанными заранее параметрами. Каждый объект содержит обязательные поля, хранящие его координаты. Если координаты выходят за рамки поля, то вызывается исключение throw LoadError(<элемент перечисления objError>, posEr); Для всех объектов были реализованы конструторы, которые принимают все необходимые поля.

UML-диаграмма классов представлена на рис. 1.

Рисунок 1 – UML-диаграмма классов.



Выводы.

В ходе работы было изучено сохранение и восстановление программы.