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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Объектно-ориентированное программирование»
Тема: Интерфейсы, динамический полиморфизм

Студент(ка) гр. 1381 _____ Денисова О.К.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы

Изучить написание интерфейсов, реализацию интерфейсов, использование виртуальных и чисто виртуальных функций.

Общая формулировка задачи

Требования:

- Разработан интерфейс события с необходимым описанием методов
- Реализовано минимум 2 группы событий (2 абстрактных класса наследников события)
- Для каждой группы реализовано минимум 2 конкретных события (наследники от группы события)
- Реализовано минимум одно условное и безусловное событие (условное - проверяет выполнение условий, безусловное - не проверяет).
- Реализовано минимум одно событие, которое меняет карту (меняет события на клетках или открывает расположение выхода или делает какие-то клетки проходимыми (на них необходимо добавить события) или не непроходимыми
- Игрок в гарантированно имеет возможность дойти до выхода

Выполнение работы

В ходе выполнения лабораторной работы были разработаны следующие классы: интерфейсы mapEvent и enemyEventBuilder, класс enemyEvent, классы событий Merchant, Money, wallCollapse, wolfBuilder, thiefBuilder, Win, а также класс Director.

1) Интерфейс mapEvent и enemyEvent наследуются от интерфейса Event, разработанного в лабораторной работе №1. События класса mapEvent создаются при помощи порождающего паттерна «прототип», события enemyEvent создаются при помощи порождающего паттерна «строитель».

В первом случае использовался паттерн «прототип», чтобы можно было клонировать события и размещать их по карте в разных местах.
Во втором случае использовался «строитель», т.к. враги действуют схожим образом и исключительно на персонажа.

2) Интерфейс enemyEventBuilder и класс Director (а также все конкретные реализации: wolfBuilder, thiefBuilder) реализованы в рамках паттерна «строитель». В enemyEventBuilder описаны шаги создания «события-врага»: устанавливается изменение здоровья, изменение силы и изменение количества монет игрока (три действия, которые враг оказывает на игрока), а Director непосредственно запускает процесс создания: при создании объекта класса Director ему подается в качестве параметра конкретный builder события, далее в методе Director::make у builder'а сначала запускаются три вышеописанных действия, а затем возвращается событие типа enemyEvent *. Паттерн позволяет отделить конструирование объекта от его представления так, что в результате одного и того же процесса конструирования могут получаться разные представления. В результате мы имеем несколько конкретных builder'ов, каждый из которых реализует одну из разновидностей события.

3) Интерфейс mapEvent (а также его наследники wallCollapse, Merchant, Money, Win) реализованы в рамках паттерна «прототип», где mapEvent содержит метод clone(), позволяющий клонировать событие, метод changeField, изменяющий поле, метод trigger, наследованный от Event, а его наследники по-разному их переопределяют: сначала метод changeField меняет поле в зависимости от конкретной реализации, затем метод clone возвращает событие типа enemyEvent *.

Пример работы программы:

1) В данном примере игрок проходит по клетке с событием enemyEvent (вор). H – здоровье, M – монеты. Как видно, у игрока стало на 20 монет меньше, т.к. вор украл их.

```
h: 20, m: 25
h: 20, m: 5
```

2) В данном примере игрок проходит по клетке с событием mapEvent (купец). Видим, что у игрока уменьшилось количество денег, зато увеличилось здоровье (игрок купил у купца ресурсы для повышения здоровья). Если бы у игрока было недостаточно денег, то он не смог бы произвести покупку, т.к. купец – событие с условием.

```
h: 20, m: 25
h: 30, m: 10
```

3) В данном примере игрок наступает на клетку с событием enemyEvent (волк). Волк снижает здоровье персонажа до нуля, и все характеристики персонажа обнуляются, что означает смерть персонажа.

```
h: 5, m: 25
h: 0, m: 0
```

UML-диаграмма:

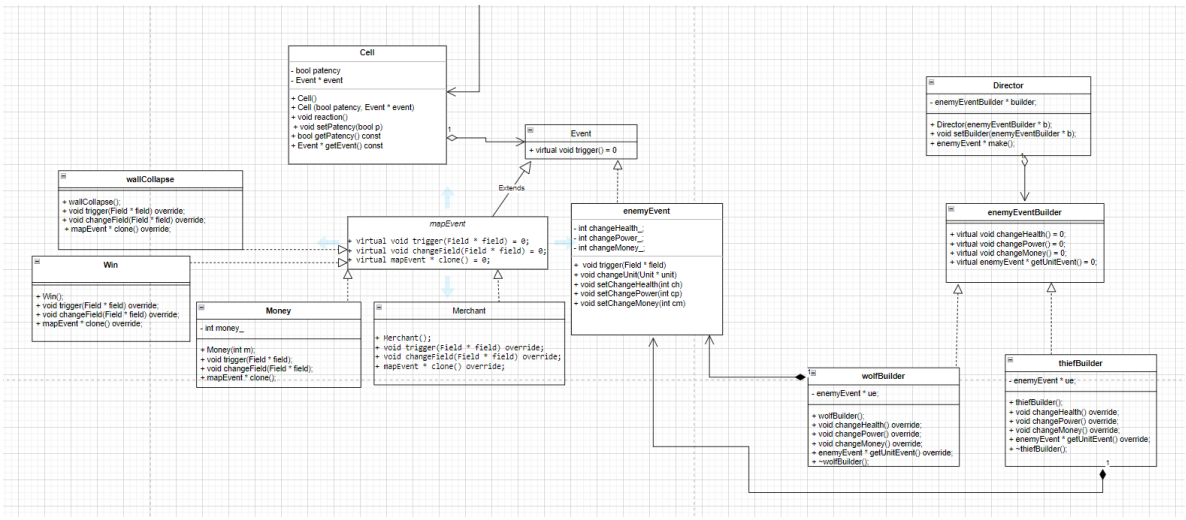

Рисунок 1. UML-диаграмма лабораторной работы №2

Выводы

В ходе выполнения лабораторной работы были приобретены знания об интерфейсах и динамическом полиморфизме и опыт их использования.