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 гр. 1381 _____ Мамин Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Изучить понятия интерфейса, реализовать события и их интерфейс.

Задание.

Реализовать систему событий. Событие - сущность, которая срабатывает при взаимодействии с игроком. Должен быть разработан класс интерфейс общий для всех событий, поддерживающий взаимодействие с игроком. Необходимо создать несколько групп разных событий реализуя унаследованные от интерфейса события (например, враг, который проверяет условие, будет ли воздействовать на игрока или нет; ловушка, которая безусловно воздействует на игрока; событие , которое меняет карту; и.т.д.). Для каждой группы реализовать конкретные события, которые по разному воздействуют на игрока (например, какое-то событие заставляет передвинуться игрока в определенную сторону, а другое меняет характеристики игрока). Также, необходимо предусмотреть событие “Победа/Выход”, которое срабатывает при соблюдении определенного набора условий.

Реализовать ситуацию проигрыша (например, потери всего здоровья игрока) и выигрыша игрока (добрался и активировал событие “Победа/Выход”)

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. Ход решения:

Используется стандартная библиотека c++ и её заголовочные файлы *iostream*, *random*.

1. Определяется класс-интерфейс события *Event*, от которого наследуются классы-группы событий.

Реализуются виртуальные методы класса с модификатором доступа *public*:

- *virtual bool execute(void* obj) = 0;* – метод срабатывания события.

2. Определяется абстрактный класс *PlayerEvent*, от которого наследуются классы событий, связанные с изменением игрока.

Реализуются методы класса с модификатором доступа *public*, аналогичные классу Event.

Поля класса с модифиактором доступа protected:

- *Player* player* – указатель на объект класса игрока.

3. Определяется абстрактный класс *FieldEvent*, от которого наследуются классы событий, связанные с изменением поля.

Реализуются методы класса с модификатором доступа *public*, аналогичные классу Event.

Поля класса с модифиактором доступа protected:

- *Player* player* – указатель на объект класса игрока.
- *Field* field* – указатель на объект класса поля.

4. Определяется класс *Heal*, объектом которого является событие, увеличивающее здоровье игрока.

Реализуются методы класса с модификатором доступа *public*:

- *bool execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который добавляет к здоровью игрока +1.
- *Heal(Player* player)* – конструктор класса.

5. Определяется класс *Heal*, объектом которого является событие увеличивающее количество монет игрока.

Реализуются методы класса с модификатором доступа *public*:

- *bool execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который добавляет к кол-ву монет игрока +1.
- *Heal(Player* player)* – конструктор класса.

6. Определяется класс *Enemy*, объектом которого является событие событие столкновения с врагом.

Реализуются методы класса с модификатором доступа *public*:

- *bool execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который отнимает от здоровья игрока случайную величину от 1 до 5 и прибавляет 5 монет игроку, если урона игрока больше, чем количество жизней врага (число от 1 до 10).
- *Enemy(Player* player)* – конструктор класса.

7. Определяются классы *Increase и Decrease*, объектом которых являются события увеличивающее и уменьшающие размер поля соответственно.

Реализуются методы класса с модификатором доступа *public*:

- *bool execute(void* obj) override* – метод, в который передаётся указатель на объект класса поля и который сохраняет размер увеличения или уменьшения размера поля при следующем его генерации.

8. Определяется класс *Collapse*, объектом которого является событие, заново генерирующее игровое поле (провал игрока на новый уровень).

Реализуются методы класса с модификатором доступа *public*:

- *bool execute(void* obj) override* – метод, в который передаётся указатель на объект класса поля и который вызывает у него новую генерацию.
- *Collapse(Player* player, Field* field)* – конструктор класса.

9. Определяется абстрактный класс RulesEvent, от которого наследуются классы событий, связанные с правилами игры.

Реализуются методы класса с модификатором доступа *public* аналогично классу *Event*:

Поля с модификатором доступа *protected:*

- *Player* player* – указатель на объект класса игрока.
- *Field* field* – указатель на объект класса поля.

10. Определяются классы Win и Lose, объектами которых являются события, отслеживающие победу и поражение в игре соответственно.

Реализуются методы класса с модификатором доступа *public*:

- *bool* execute() override – метод, возвращающий *true* в случае победы или поражения.

11. Определяется класс EventBuilder создающий и обновляющий события на поле.

Реализуются методы класса с модификатором доступа *public*:

- *void update_events()* – обновляет все события в клетках.
- *Event* create_NameEvent()* – создаёт событие Name.

Поля с модификатором доступа *private:*

- *Player* player* – указатель на объект класса игрока.
- *Field* field* – указатель на объект класса поля.

Архитектура программы.

В каждой клетке (объекте класса *Cell*) хранится указатель на событие (объект класса *Event*). При генерации игрового поля (объекта класса *Field*) с помощью метода *create_field(),* для каждой клетки в зависимости от её координат методом *update_events()* класса *EventBuilder* назначается соответствующее событие. Объект класса *EventBuilder* инициализируется внутри метода *move_player() класса Field*. Внутри соответствующи методов класса *EventBuilder* создаются конкретные события и возвращаются в виде указателя на абстрактный

класс *Event*.

При перемещении игрока по полю с помощью метода класса *Field, move_player()* и при его наступании в клетку, тип которой предусматривает в себе наличие события, вызывается метод *execute()* соответствующего ей события.

Так как условие победы или поражения в игре зависит исключительно от характеристик игрока, при каждом срабатывании события класса *PlayerEvent* вызывается метод *execute() событий Win* и *Lose*, наследуемых от класса *RulesEvent*, в случае возвращения которыми *true*, игра завершается.

Результат работы программы:

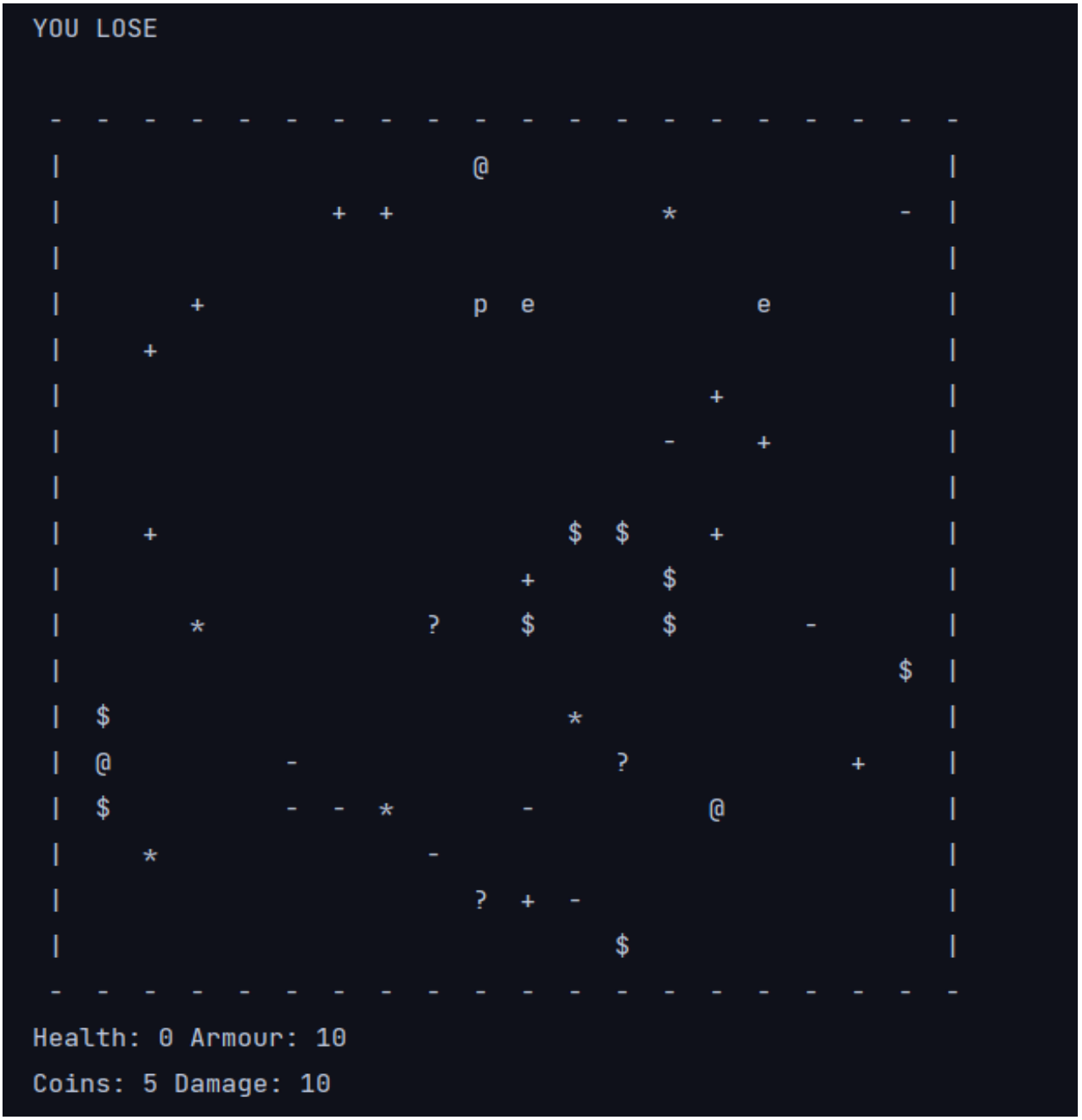

Рис 1. – демонстрация работы программы в терминале Ubuntu. Срабатывание события поражения.

UML-диаграмма межклассовых отношений:

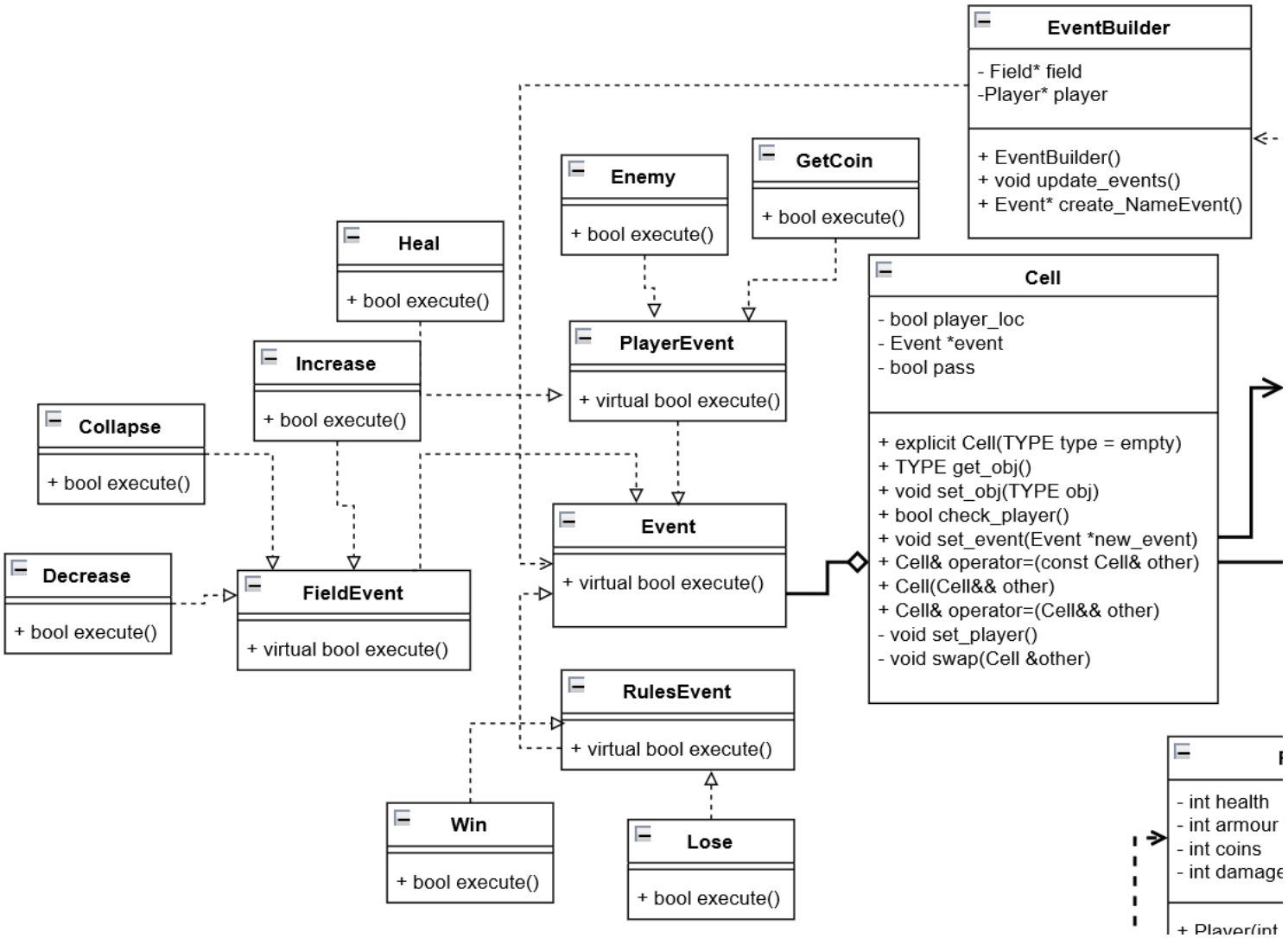

Рис 2. – UML-диаграмма.

Вывод: Был реализован интерфейс события (класс *Event*) и наследуемые от него абстрактные классы-группы событий с конкретными событиями, хранящимися по указателю в каждой клетке игрового поля.