

**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

Смирнов Д. Ю.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2022

### **Цель работы.**

Изучить понятие принципы полиморфизма, научиться реализовывать классы, которые в иногда являются интерфейсом, и осуществлять межклассовые отношения соблюдая полиморфизм.

### **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

### Примечания:

- Классы событий не должны хранить никакой информации о типе события (никаких переменных и функций дающие информации о типе события)
- Для создания события можно применять абстрактную фабрику/прототип/строитель

### Выполнение работы.

Для выполнения лабораторной работы были созданы классы, отвечающие за игрока, создание клетки поля, создания поля, их вывод и взаимодействие пользователя с программой.

*Новые классы:*

**СОБЫТИЯ:** был создан интерфейс *Event* с виртуальным методом *execute()*, от данного интерфейса наследуются все абстрактные классы групп событий (*PlayerEvents*, *FieldEvents*).

Абстрактный класс *PlayerEvents* содержит в себе метод *execute()* и поле указатель на *Player*, который, исходя из конкретного события, изменяет поля игрока. От него наследуются классы *Buff*, *CoinEvent*, *PoisonTrap*, *FrozenTrap*, *HealEvent*, которые соответственно увеличивает поле *max\_stat*, увеличивает поле *coins* игрока на случайное количество монет в пределах от 1 до 4, уменьшает поле *hearts*, восстанавливает поля *hearts* и *power* игрока до *max\_stat*, если они ниже данного значения.

Интерфейс *FieldEvents* содержит в себе в себе метод *execute()* и у казатель на *Field*, который, исходя из конкретного события изменяет игровое поле. От него наследуется классы *FieldNoWalls* и *FieldChangePos*, при вызове метода *execute*,

соответственно изменяют поле очищает игровое поле от стенок, игрок делает шаг в случайном направлении.

**ОТОБРАЖЕНИЕ:** был написан класс *PlayerView*, который выводит все характеристики игрока.

**ГЕНЕРАТОР СОБЫТИЙ:** Класс *EventGenerator* хранит в полях указатель на игрока и игровое поле. Имеет метод, создающий указатель на одно из вышеперечисленных событий случайным образом.

**ФАБРИКА КЛЕТОК:** Класс *CellFactory*, конструктор данного класса должен получить в качестве аргумента объект класса *EventGenerator*. Класс имеет три метода: *base\_cell()* - создание проходимой клетки без события, *wall\_cell()* – создание не проходимой клетки, *event\_cell()* – создание клетки с событием полученным из *EventGenerator'a*.

**ГЕНЕРАТОР ИГРОВОГО ПОЛЯ:** Класс *FieldGenerator*, конструктор класса должен получить в качестве аргумента объект класса *CellFactory*, конструктор добавляет пары ключ:значение(название клетки : лямбда-выражение обернутое в *std::function*, которое вызывает метода у объекта *CellFactory*) в *map*. У класса есть метод *set\_cell()* добавляющий в вектор клетку случайным образом. Метод *field\_generate()* в качестве аргумента принимает объект игрового поля, заполняют вектор клеток, используя *set\_cell()*, и устанавливает вектор клеток в объект класса *Field*.

## **ПАТТЕРНЫ:**

### **Наблюдатель:**

Были написан интерфейс *IObserver*, который содержит виртуальный метод *update*, Интерфейс реализуется в классах *PlayerView* и *FieldView*. Их реализованный метод *update* выводит статистику игрока и поле, соответственно.

Создан класс *ISubject*, который хранит в себе все наблюдаемые объекты в виде вектора указателей на *IObserver*. Есть методы *attach*, *detach*, *notify*, которые добавляет в вектор наблюдаемых объектов, удаляет объект из этого вектора и вызывает *update* у всех подписанных *IObserver'ов*.

### Посредник:

Был написан абстрактный класс *MediatorObject*, который имеет поле *mediator* указывающее на *IMediator*. Конструктор класса заносит указатель на *IMediator* в поле *mediator* если он был передан, иначе *nullptr*. Класс также имеет метод *set\_mediator()* для переприсваивания поля *mediator*.

Были написан интерфейс *IMediator*, который содержит виртуальный метод *notify*. Данный метод принимает указатель на *MediatorObject* и команду, которую *MediatorObject* хочет произвести, *Mediator* в свою очередь анализируя кто запросил и что, дает распоряжению другому *MediatorObject*'у.

#### Изменение в классах:

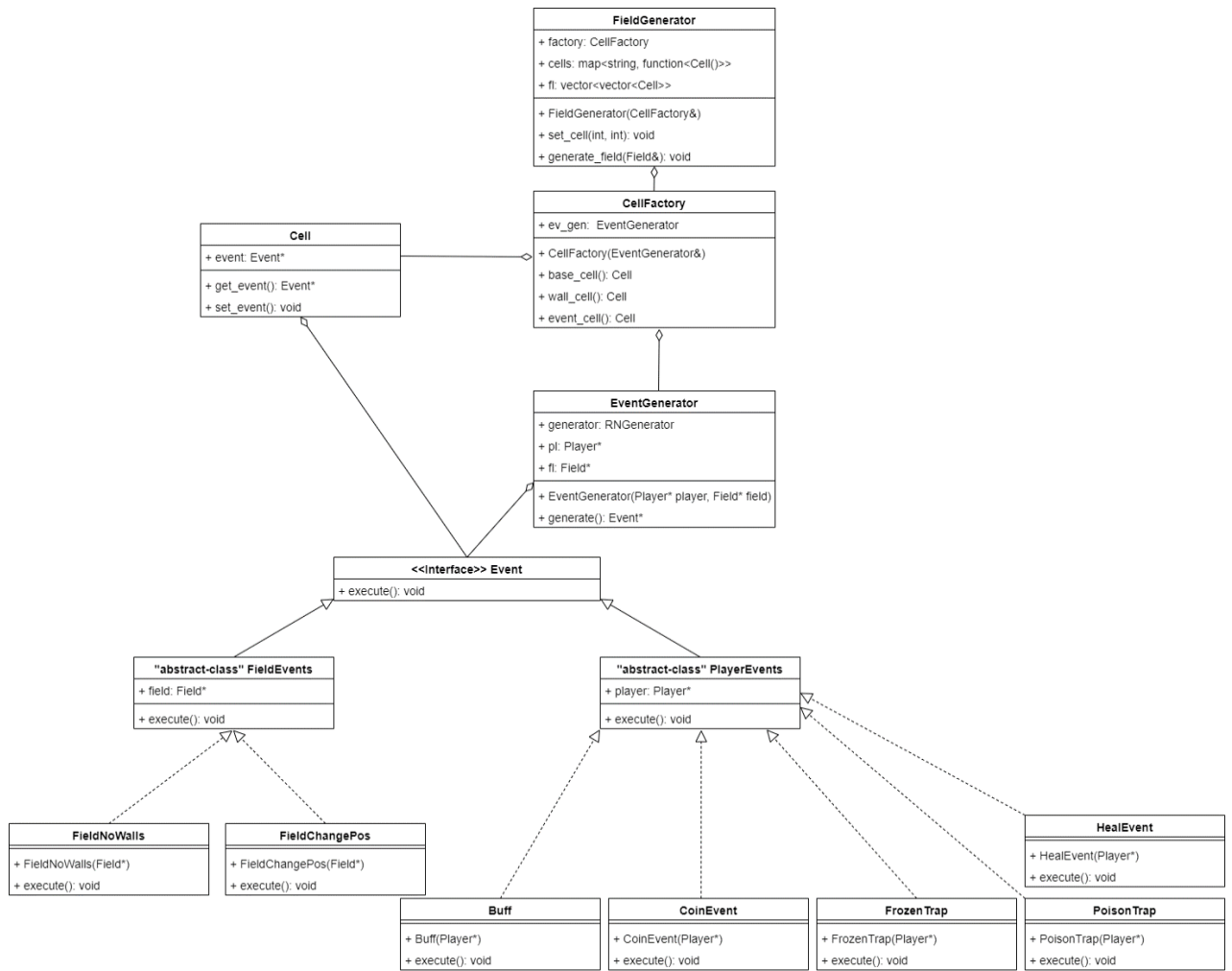
1) В классе *Field* был убран метод *generate\_field*, теперь есть метод *set\_field* который позволяет установить вектор клеток. Метод *change\_player\_position* после того как игрок переместился на клетку вызывает метод *execute()* у события, если оно есть у клетки.

2) В класс *Controller* были добавлены методы *is\_end()*, который проверяет игрока(умер ли он, потерял ли все силы, или набрал нужное количество монет).

3) В класс *IOCommander* были добавлены методы *Victory()* и *Defeat()*, которые печатают, сообщение об победе и поражении.

## UML-диаграмма межклассовых отношений.

Рис. 1 UML-диаграмма.



**Выводы.**

Я научился реализовывать классы-интерфейсы, и осуществлять межклассовые отношения, соблюдая динамический полиморфизм.