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:

КЛЕТКИ: был создан интерфейс *ICell*, от которого будут наследоваться все типы клеток (*CellBase*, *CellBuff*, *CellCoin*, *CellMove*, *CellTrap*, *CellWall*). Интерфейс содержит виртуальный метод *get_event*, который возвращает указатель на событие. В случае, когда наследуемый класс не содержит внутри себя событие возвращается *nullptr*.

СОБЫТИЯ: был создан интерфейс Event, от которого будут наследуются все абстрактные классы групп событий (*PlayerEvents*, *FieldEvents*).

Интерфейс *PlayerEvents* содержит в себе в себе метод *execute(Player& 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(Field& field)*, который, исходя из конкретного события изменяет игровое поле *Field*. От него наследуется классы *FieldRegen* и *CellMove*, при вызове метода *execute*, соответственно изменяют поле очищает игровое поле от клеток и заново его генерирует, игрок делает шаг в случайном направлении.

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

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'y*.

Изменение в классах:

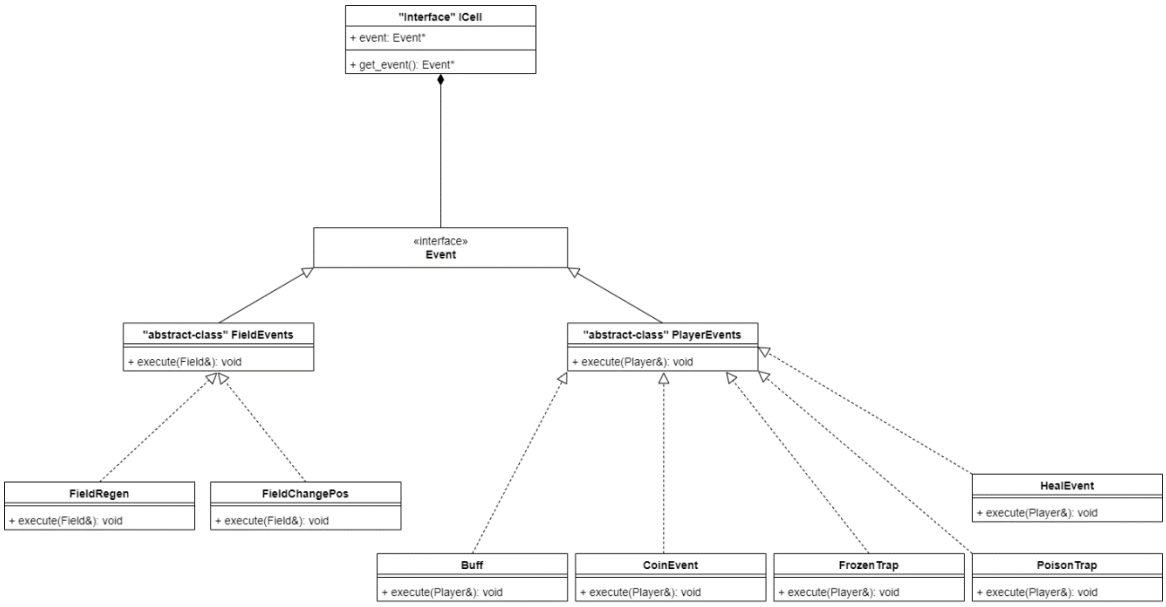
1) В классе *Field* был переписан метод *generate_field* под новую реализацию клеток. Этот метод теперь полностью инициализирует двумерные массив из указателей на клетки. Он же вызывается в конструкторе. Так было сделано для того, чтобы использовать этот метод в событии *FieldRegen*. Также был написан метод deconstruct, который чистит память. Он же вызывается в деструкторе поля. Метод *change_player_position* теперь возвращает указатель на событие, которое находится на позиции игрока. Это нужно чтобы поднять событие на уровень класса *Game*.

2) В класс *Controller* были добавлены методы *is_end(),* который проверяет игрока(умер ли он, потерял ли все силы, или набрал нужное количество монет). Был добавлен метод *event_handler*, который проверяет к какой группе событий (*PlayerEvents* или *FieldEvents*) относится событие.

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.