**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объекто-ориентированное программирование»**

Тема: Полиморфизм

|  |  |  |
| --- | --- | --- |
| Студент гр. 2382 |  | Ваньков Я.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

## Цель работы

Изучить методы работы с классами. Написать простую программу в стиле ООП. Изучить работу конструктора и деструктора в С++. Изучить конструкторы перемещения и конструкторы копирования.

## **Задание**

а) Создать интерфейс игрового события. Интерфейс должен обеспечивать срабатывание события когда игрок наступает на клетку.

б) Реализовать интерфейс игрового события тремя конкретными событиями. Одно событие должно положительно влиять на характеристики игрока, второе должно негативно влиять на характеристики игрока, третье изменять координаты игрока на поле. При желании можно реализовать больше событий и/или события меняющие само поле (например, делать из непроходимой клетки проходимую).

в) В классе управления игроком добавить проверку на наличие события на клетке, если событие присутствует, то оно должно сработать. Срабатывание должно происходить через интерфейс события, и не должно быть никаких проверок на тип события (реализация через динамический полиморфизм)

г) Создать класс создающий поле. Предусмотреть возможность создания 2 разных уровней. По желанию можно сделать случайную генерацию уровней. Должно гарантироваться, что игрок может дойти от входа до выхода.

Примечания:

События должны быть такими, чтобы был сценарий проигрыша игрока.

В событиях и клетках не должно быть полей сообщающих информацию о типе события

## Выполнение работы

**Player — класс игрока**

Класс игрока. В публичных полях находятся ссылки на все характеристики игрока. Не имеет методов.

**Characterisric — класс характеристики**

Главный класс для характеристик: от него наследуются все характеристики. В конструктор класса передается минимальное, максимальное и установленное значение.

Определено три метода:

getValue — возвращает значение

setValue — меняет значение, учитывая ограничение класса.

Check — проеверяет измененное значение, согласно ограничениям класса

Оператор присваивания перегружен — при попытки присваивания выбрасывает ошибку

**Наследники класса характеристик:**

**Health — класс здоровья**

Добавляется приватное поле игрока — is\_dead, которое принимает только булевые типы данных.

Также добавляется метод на проверку наличия здоровья у игрока.

**Armor — класс брони**

**Count — класс количества**

**Money — класс денег**

**Power — класс силы**

**Score — класс очков**

**Item. - класс предмета**

В приватных полях лежит название предмета и его характеристики(они же и инициализируются в конструкторе).

Методы:

getName — вовзвращает имя

setName — изменяет имя

**Weapon — класс оружия**

В приватных полях класса лежит урон, наносимый оружием и название оружия.

Методы класса:

getDamage — вовзвращает урон

getName — вовзвращает имя

setDamage — изменяет урон

setName — изменяет имя

use — изменяет значение силы игрока на значение урона

**Coordinate** **— класс координаты**

В приватных полях имеет координаты по X и Y.

Дружественный класс для MoveManager и Square

Публичные методы:

getX — возвращает X

getY — возвращает Y

getPair — возвращает пару

**MoveManager — класс контроллера над игроком**

Класс, отвечающий за контроль движения игрока на поле.

Конструктор класса принимает игрока, над которым будут производится действия и поле, на котором будет находиться игрока.

В приватных полях находится ссылка на игрока, над которым воздействует менеджер и ссылка поле на котором, находится игрок. Также в приватном поле находится координата игрока.

Методы в зависимости от переданного значения изменяют координату игрока.

Реализован метод checkCoord, который возвращает булевое значение - запрета перемещения в клетку, где стоит стена — или выход за пределы поля.

Изменения к 2 ЛР:

**Field — класс поля**

Конструктор принимает высоту и ширину, а также координаты старта и конца. Каждый аргумент имеет дефолтное значение, поэтому можно создать поле, не указывая аргументов в конструкторе. Поле проверяется на допустимость размера. Выделяется память под поле.

Реализован конструктор копирования, перемещения и деструктор.

Публичные методы:

getSquare — перегруженный метод, который может принимать координаты, а также целочисленные значения x и y. Так реализуется приницы полиморфизма. Возвращает ссылка на клетку

GetHeight и getWidth — методы возвращающие числовое значение высоты и ширины.

GetStart и getEnd — методы, возвращающие ссылки на координаты старта и конца.

CheckSize — перегруженный метод, который может принимать координаты, а также целочисленные значения x и y. Так реализуется приницы полиморфизма. Возвращает булевое значение корректности размера поля

**Square**

Констуктор принимает проходимость клетки и ее координату.

В приватных полях класса находится координата и проходимость.

Реализованы следующие методы класса:

setPassable — задает проходимость указанной клетки

getPassable — возвращает проходимость указанной клетки

getCoordinate — возвращает ссылку на координату указанной клетки

оператор присваивания перегружен — присвоить клетку можно только в случае, если у нее задано дефолтное значение.

Изменения к 3 ЛР:

**Event**

Виртуальный класс, у которого есть виртуальные методы клонирования ивента и старта ивента. Далее во всех класса клон переопредлен так, что возвращает новый указатель на ивент.\

Реализованы следующие методы класса:

virtual void clone() - далее переопределен у каждого нужен для клонирования ивента.

virtual void start() - далее переопределен у каждого нужен для старта ивента

**Наследники класса Event:**

**TrapEvent —** ивент уменьшающий здоровье. Происходит проверка на то, умер ли персонаж.

Реализованы следующие методы класса:

TreatEvent(int damage) — конструктор принимающий сколько здоровья надо отнять

void clone() - возвращает новый указатель на новый ивент

void start() - убавляет столько здоровья, сколько передано в конструкторе

**TreatEvent -** ивент увеличивающий здоровье. Происходит проверка на то, ожил ли персонаж.

Реализованы следующие методы класса:

TreatEvent(int heal) - конструктор принимающий сколько здоровья надо прибавить

void clone() - возвращает новый указатель на новый ивент

void start(MoveManager& playerControl) - старт: прибавляет здоровья, согласно тому, сколько передано в конструкторе

**TeleportEvent** — ивент перемещающий игрока по указанным координатам с помощью метода телепорт класса MoveManager.

Реализованы следующие методы класса:

TeleportEvent(Coordinate coord) - конструктор принимающий координату, куда надо переместить игрока

void clone() - возвращает новый указатель на новый ивент

void start(MoveManager& playerControl) - старт: прибавляет здоровья, согласно тому, сколько передано в конструкторе

**NoneEvent** — дефолтный ивент, увеличивающий очки игркоа на 1.

Реализованы следующие методы класса:

void clone() - возвращает новый указатель на новый ивент

void start(MoveManager& playerControl) - старт: прибавляет одну единицу очков.

**FieldCreator**

Класс создающий поле. В приватных полях лежит два метода создания поля — для каждого уровня. Далее можно дописать несколько уровней, соответственно несколько полей.

Каждый метод создает свой уровень:

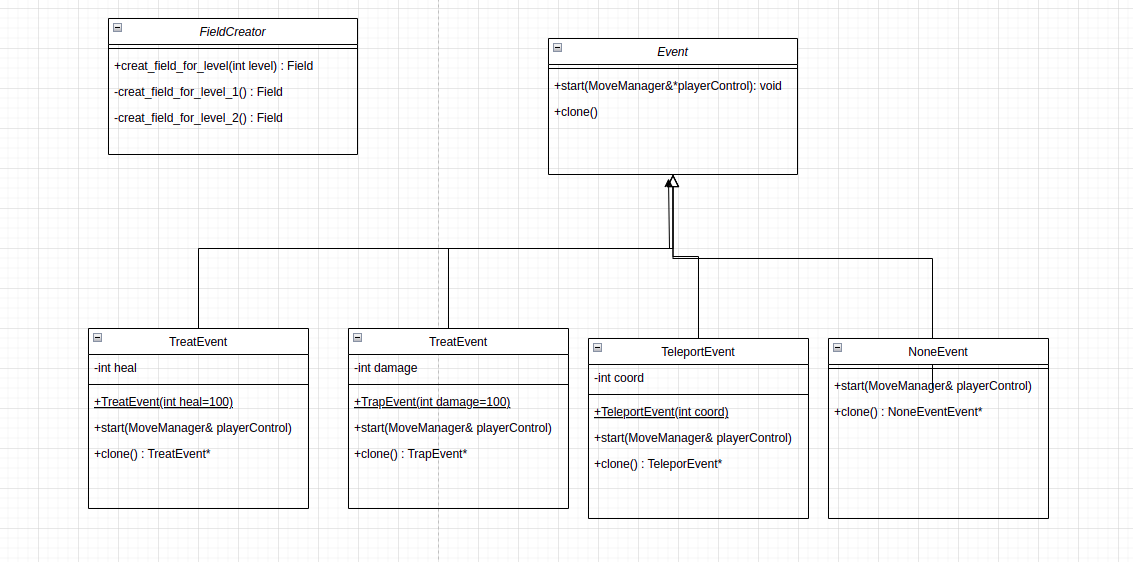
Приватные методы:

Field creat\_field\_for\_level\_1() - создает уровень 1 по некоторому правилу — возвращает новое поле

Field creat\_field\_for\_level\_2() - создает уровень 2 по некоторому правилу — возвращает новое поле

Публичные методы:

Field creat\_field\_for\_level(int level) — вызывает один из методов, согласно переданному уровню



## Выводы

Были изучены методы работы с классами. Были написаны классы игрока, его характеристик и классы взаимодействия игрока с его координатами. Был написан класс поля и клетки. К классу поле реализованы конструкторы копирования и перемешение, а также деструктор. К классу поле был реаилозован класс создания уровня.

# Приложение А Исходный код программы

# https://github.com/DoAlvaro/OOP\_2382