《以撒的结合》游戏面向对象架构设计

一、类的成员变量与成员函数

1. Entity（基类）

成员变量：name: str — 实体名称
position: Tuple[float, float] — 当前坐标

成员函数：describe() -> str — 返回实体基本信息
update() -> None — 每帧更新状态
render() -> None — 渲染至屏幕

2. Character（可操作角色，继承自 Entity）

成员变量：health: int — 生命值（红心数）
damage: float — 伤害系数
speed: float — 移动速度
luck: int — 幸运值
inventory: List[Item] — 道具背包

成员函数：move(direction: str) -> None — 按方向移动
attack(target: Entity) -> None — 攻击目标
use_item(item: Item) -> None — 使用道具

3. Isaac（主角，继承自 Character）

新增成员变量：special_ability: str—天赋或特殊能力标识

新增成员函数：activate_special()->None —触发特殊能力

4. Enemy（敌人，继承自 Entity）

成员变量：health: int — 生命值
damage: float — 伤害值
behavior_type: str — 行为模式（如 “Melee”、“Ranged”）

成员函数：patrol() -> None — 巡逻逻辑
attack(target: Character) -> None—攻击玩家
drop_loot() -> List[Item]—掉落道具

5. Item（道具基类，继承自 Entity）

成员变量：description: str — 道具描述

成员函数：apply_effect(target: Character)->None —对角色生效

6. ActiveItem（主动道具，继承自 Item）

新增成员变量：cooldown: float — 冷却时间

新增成员函数：use() -> None — 触发效果并开始冷却

7. PassiveItem（被动道具，继承自 Item）

新增成员函数：on_pickup(target: Character) -> None — 拾取后生效

8. Room（房间，继承自 Entity）

成员变量：room_type: str—房间类型（Boss、Shop、Treasure 等）
enemies: List[Enemy]—敌人列表
items: List[Item —道具列表
is_cleared: bool—是否已清除

成员函数：enter() -> None—进入房间，初始化内容
clear() -> None—清除房间，打开奖励出口

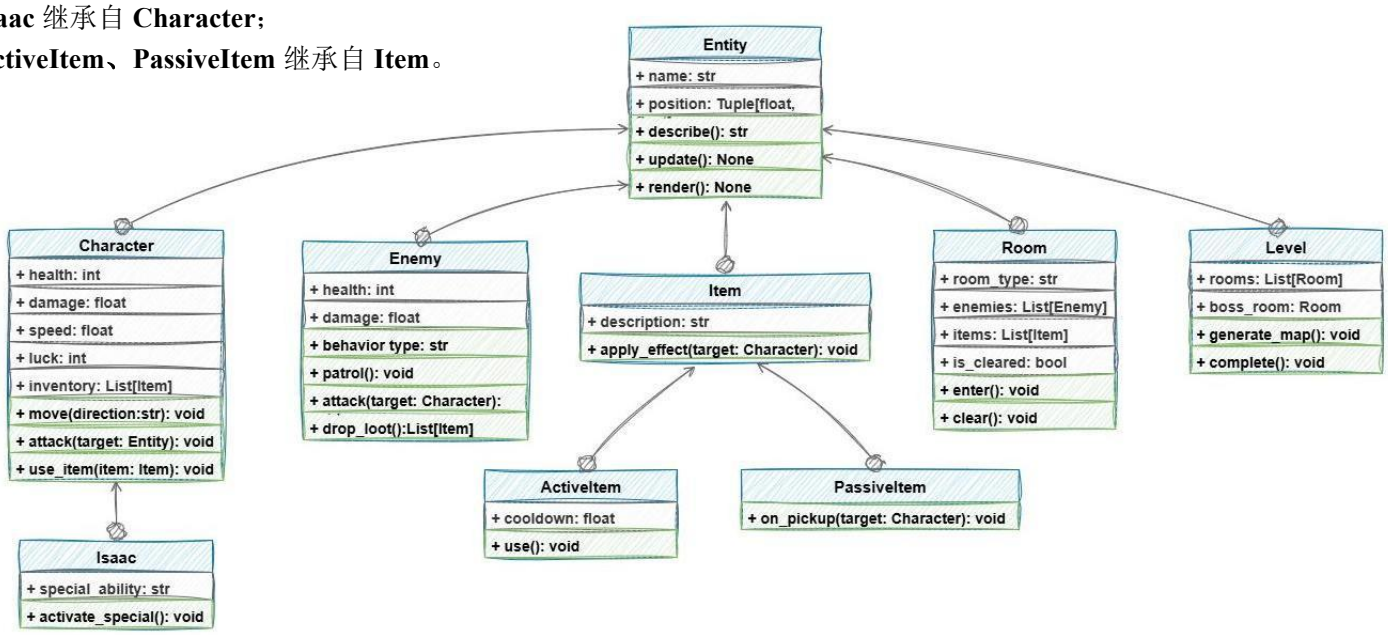
9. Level（关卡/Floor，继承自 Entity）

成员变量：rooms: List[Room] — 本层所有房间
boss_room: Room — Boss 房间引用

成员函数：generate_map()->None—按规则生成房间布局
complete() ->None — 完成本层，触发下一层

二、继承关系

Entity 为所有游戏实体的基类，定义了公用的 name、position 等属性及 describe()、update()、render() 三个基本方法。
Character、**Enemy**、**Item**、**Room**、**Level** 均直接继承自 **Entity**，获得基本属性与方法；
Isaac 继承自 **Character**；
ActiveItem、**PassiveItem** 继承自 **Item**。



三、程序设计范式：工厂模式

为解耦对象创建与使用，项目引入工厂模式，定义以下工厂类：

CharacterFactory：

create_character(type_name: str, **kwargs) -> Character

根据 type_name（如 "Isaac"、"Magdalene"）返回对应角色实例。

EnemyFactory：

create_enemy(type_name: str, **kwargs) -> Enemy

根据类型字符串（如 "Gapper"、"Monstro"）生成相应敌人/Boss。

ItemFactory：

create_item(type_name: str, **kwargs) -> Item

动态创建 ActiveItem、PassiveItem、Consumable 或 Trinket。

RoomFactory：

create_room(room_type: str, **kwargs) -> Room

根据房间类型初始化并返回 Room 对象。

LevelFactory：

create_level(floor_number: int, rooms: List[Room], **kwargs) -> Level

生成关卡 Level，对应 floor_number，并绑定房间列表

工厂方法将创建逻辑集中管理，客户端只需调用统一接口，无需关心具体子类或初始化细节；当新增角色、敌人或道具时，只需在对应工厂中注册新类型即可，极大增强了系统的可扩展性与可维护性。

例如：在关卡初始化时，我们首先调用 CharacterFactory 创建主角 Isaac，并为他设置名称、初始位置、生命值、伤害、速度、幸运值和特殊能力“Brimstone”；接着同样通过 EnemyFactory 生成一个普通敌人 Gaper，并指定其名称、位置、生命值、伤害和行为模式；然后利用 ItemFactory 创建一个主动道具 D6（描述为“重骰，可以重置道具池”，带有 10 秒冷却），并将其添加到 Isaac 的道具背包；随后，再借助 RoomFactory 构造一个 Boss 房间，将之前创建的 Gaper 作为房间内的敌人；最后，通过 LevelFactory 创建整层关卡 Basement I，将该 Boss 房间纳入关卡房间列表并标记为当前 Boss 房间，从而完成了从角色、敌人、道具到房间再到关卡的整体对象生成流程。

示例：关卡初始化流程

```
# 1. 生成主角 Isaac
isaac = CharacterFactory.create_character(
    "Isaac",
    name="Isaac",
    position=(0, 0),
    health=3,
    damage=1.0,
    speed=5.0,
    luck=0,
    special_ability="Brimstone"
)

# 2. 生成一个普通敌人 Gaper
gaper = EnemyFactory.create_enemy(
    "Gapper",
    name="Gaper",
    position=(10, 5),
    health=5,
    damage=0.5,
    behavior_type="Melee"
)

# 3. 创建主动道具 D6 并加入背包
d6 = ItemFactory.create_item(
    "D6",
    description="重骰，可以重置道具池",
    cooldown=10.0
)
isaac.inventory.append(d6)
# 4. 构造 Boss 房间，并添加敌人
boss_room = RoomFactory.create_room(
    "Boss",
    room_type="Boss",
    enemies=[gaper],
    items=[],
    is_cleared=False
)
# 5. 生成整层关卡 Basement I
basement_I = LevelFactory.create_level(
    floor_number=1,
    rooms=[boss_room],
    boss_room=boss_room
```