character.py

```
"""Character class - handles stats, leveling, skills, status effects."""
from __future__ import annotations
import random
from dataclasses import dataclass, field
from typing import List, Dict, Optional, Tuple
from data_types import *
from skills_db import SKILL_DB, JOB_SKILL_POOL


# ── Base stats per job ──────────────────────────────────────────────
JOB_BASE_STATS: Dict[JobClass, Stats] = {
    JobClass.WARRIOR:    Stats(220,40, 28,8,  18,12,12,0.06,0.88,0.10),
    JobClass.KNIGHT:     Stats(280,50, 22,8,  28,18,10,0.04,0.87,0.06),
    JobClass.PALADIN:    Stats(260,80, 20,18, 24,22,11,0.05,0.87,0.07),
    JobClass.BERSERKER:  Stats(240,30, 35,6,  14,10,13,0.05,0.83,0.18),
    JobClass.ASSASSIN:   Stats(180,60, 26,14, 12,14,20,0.18,0.90,0.22),
    JobClass.RANGER:     Stats(190,55, 24,12, 14,14,18,0.14,0.92,0.16),
    JobClass.HUNTER:     Stats(195,55, 24,12, 15,14,17,0.13,0.91,0.15),
    JobClass.SPEARMAN:   Stats(210,45, 26,8,  16,14,15,0.08,0.88,0.10),
    JobClass.DRAGOON:    Stats(220,50, 28,10, 16,15,16,0.09,0.88,0.12),
    JobClass.MONK:       Stats(230,45, 30,8,  16,12,18,0.10,0.89,0.12),
    JobClass.CLERIC:     Stats(180,100,14,26, 16,24,12,0.05,0.87,0.05),
    JobClass.PRIEST:     Stats(170,120,12,28, 14,26,11,0.04,0.86,0.04),
    JobClass.FIRE_MAGE:  Stats(160,110,10,34, 10,18,13,0.07,0.87,0.08),
    JobClass.ICE_MAGE:   Stats(160,110,10,32, 12,20,12,0.07,0.87,0.08),
    JobClass.STORM_MAGE: Stats(160,110,10,33, 10,18,14,0.08,0.87,0.08),
    JobClass.WIND_MAGE:  Stats(155,105,10,30, 10,18,16,0.10,0.88,0.08),
    JobClass.EARTH_MAGE: Stats(170,105,12,30, 14,20,11,0.06,0.87,0.07),
    JobClass.DARK_MAGE:  Stats(165,110,10,35, 10,16,14,0.09,0.87,0.12),
    JobClass.LIGHT_MAGE: Stats(165,110,10,33, 12,20,13,0.07,0.87,0.08),
    JobClass.ARCANE_SAGE:Stats(175,130,12,36, 12,22,12,0.07,0.88,0.10),
}

JOB_GROWTH: Dict[JobClass, GrowthRates] = {
    JobClass.WARRIOR:    GrowthRates(0.85,0.40,0.65,0.30,0.55,0.40,0.40,0.20,0.35,0.25),
```

```
    JobClass.KNIGHT:     GrowthRates(0.90,0.45,0.50,0.25,0.70,0.55,0.30,0.15,0.30,0.15),
    JobClass.PALADIN:    GrowthRates(0.88,0.60,0.48,0.42,0.60,0.60,0.32,0.18,0.32,0.18),
    JobClass.BERSERKER:  GrowthRates(0.82,0.30,0.75,0.20,0.40,0.30,0.45,0.15,0.28,0.40),
    JobClass.ASSASSIN:   GrowthRates(0.65,0.55,0.58,0.35,0.35,0.42,0.65,0.50,0.55,0.60),
    JobClass.RANGER:     GrowthRates(0.70,0.55,0.55,0.38,0.38,0.42,0.55,0.45,0.60,0.45),
    JobClass.HUNTER:     GrowthRates(0.72,0.55,0.55,0.38,0.40,0.42,0.52,0.42,0.58,0.42),
    JobClass.SPEARMAN:   GrowthRates(0.78,0.45,0.60,0.28,0.48,0.40,0.50,0.25,0.40,0.28),
    JobClass.DRAGOON:    GrowthRates(0.80,0.48,0.62,0.32,0.48,0.42,0.52,0.28,0.42,0.32),
    JobClass.MONK:       GrowthRates(0.80,0.42,0.65,0.25,0.50,0.38,0.55,0.30,0.45,0.35),
    JobClass.CLERIC:     GrowthRates(0.72,0.75,0.28,0.58,0.45,0.65,0.35,0.20,0.32,0.15),
    JobClass.PRIEST:     GrowthRates(0.68,0.80,0.25,0.62,0.42,0.70,0.32,0.18,0.30,0.12),
    JobClass.FIRE_MAGE:  GrowthRates(0.60,0.78,0.22,0.72,0.28,0.48,0.40,0.22,0.35,0.22),
    JobClass.ICE_MAGE:   GrowthRates(0.60,0.78,0.22,0.70,0.30,0.50,0.38,0.22,0.35,0.22),
    JobClass.STORM_MAGE: GrowthRates(0.60,0.78,0.22,0.72,0.28,0.48,0.42,0.22,0.35,0.22),
    JobClass.WIND_MAGE:  GrowthRates(0.58,0.75,0.22,0.68,0.28,0.48,0.50,0.28,0.38,0.20),
    JobClass.EARTH_MAGE: GrowthRates(0.65,0.75,0.25,0.68,0.35,0.50,0.35,0.18,0.32,0.18),
    JobClass.DARK_MAGE:  GrowthRates(0.60,0.78,0.20,0.75,0.26,0.46,0.42,0.28,0.35,0.30),
    JobClass.LIGHT_MAGE: GrowthRates(0.62,0.78,0.22,0.72,0.30,0.52,0.40,0.22,0.34,0.20),
    JobClass.ARCANE_SAGE:GrowthRates(0.65,0.82,0.28,0.78,0.32,0.55,0.42,0.25,0.38,0.25),
}


@dataclass
class Character:
    name: str
    job: JobClass
    weapon: WeaponType
    element: Element
    rarity: int  # 0=player, 3/4/5=companion

    base_stats: Stats
    growth: GrowthRates

    # All skills the character CAN have (assigned at creation)
    skill_pool: List[int] = field(default_factory=list)  # 5 skill ids
    # Skills currently UNLOCKED
    unlocked_skills: List[int] = field(default_factory=list)

    level: int = 1
    exp: int = 0
```

```
    exp_to_next: int = 100

    # Runtime state
    current_hp: int = 0
    current_mp: int = 0
    status_effects: List[StatusEffect] = field(default_factory=list)
    temp_buffs: Dict[str, Tuple[float,int]] = field(default_factory=dict)  # stat -> (mult,
turns_remaining)

    is_dead: bool = False  # permanent death
    ko_turns: int = 0      # turns spent at 0 hp (KO counter)
    is_ko: bool = False    # currently knocked out in battle

    battle_count: int = 0  # total battles participated

    def __post_init__(self):
        if self.current_hp == 0:
            self.current_hp = self.base_stats.hp
        if self.current_mp == 0:
            self.current_mp = self.base_stats.mp

    # ── Stat helpers ──────────────────────────────────────────────
    def effective_stat(self, stat_name: str) -> float:
        base = getattr(self.base_stats, stat_name)
        mult = 1.0
        for effect_name, (m, turns) in self.temp_buffs.items():
            if effect_name == stat_name:
                mult *= m
        # Status effects
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.ATTACK_DOWN and stat_name == "patk":
                mult *= 0.7
            elif se.effect_type == StatusEffectType.DEFENSE_DOWN and stat_name == "pdef":
                mult *= 0.7
            elif se.effect_type == StatusEffectType.MAGIC_DOWN and stat_name == "matk":
                mult *= 0.7
            elif se.effect_type == StatusEffectType.SPEED_DOWN and stat_name == "spd":
                mult *= 0.7
            elif se.effect_type == StatusEffectType.ACCURACY_DOWN and stat_name == "acc":
                mult *= 0.6
```

```
        elif se.effect_type == StatusEffectType.BERSERK:
            if stat_name == "patk": mult *= 1.5
            if stat_name == "pdef": mult *= 0.7
        elif se.effect_type == StatusEffectType.GUARD_UP:
            if stat_name in ("pdef","mdef"): mult *= 1.4
        elif se.effect_type == StatusEffectType.FOCUS:
            if stat_name in ("acc","crit"): mult *= 1.5
        elif se.effect_type == StatusEffectType.MAGIC_BOOST:
            if stat_name == "matk": mult *= 1.4
        elif se.effect_type == StatusEffectType.HASTE:
            if stat_name == "spd": mult *= 1.5
    return base * mult

@property
def max_hp(self): return self.base_stats.hp
@property
def max_mp(self): return self.base_stats.mp
@property
def spd(self): return int(self.effective_stat("spd"))
@property
def patk(self): return int(self.effective_stat("patk"))
@property
def matk(self): return int(self.effective_stat("matk"))
@property
def pdef(self): return int(self.effective_stat("pdef"))
@property
def mdef(self): return int(self.effective_stat("mdef"))
@property
def eva(self): return self.effective_stat("eva")
@property
def acc(self): return self.effective_stat("acc")
@property
def crit(self): return self.effective_stat("crit")

def is_incapacitated(self) -> bool:
    """Cannot act at all."""
    incap = {StatusEffectType.SLEEP, StatusEffectType.STUN,
             StatusEffectType.FREEZE, StatusEffectType.PARALYZE,
             StatusEffectType.PETRIFY, StatusEffectType.TIME_STOP}
    return any(se.effect_type in incap for se in self.status_effects) or self.is_ko
```

```
    def can_use_magic(self) -> bool:
        blocked = {StatusEffectType.SILENCE, StatusEffectType.MANA_BURN}
        return not any(se.effect_type in blocked for se in self.status_effects)

    def can_use_skills(self) -> bool:
        return not any(se.effect_type == StatusEffectType.SKILL_SEAL for se in
self.status_effects)

    def can_use_items(self) -> bool:
        return not any(se.effect_type == StatusEffectType.ITEM_SEAL for se in
self.status_effects)

    def can_be_healed(self) -> bool:
        return not any(se.effect_type == StatusEffectType.HEAL_BLOCK for se in
self.status_effects)

    def has_status(self, stype: StatusEffectType) -> bool:
        return any(se.effect_type == stype for se in self.status_effects)

    def add_status(self, effect: StatusEffect):
        # Don't stack same type (refresh duration instead)
        for existing in self.status_effects:
            if existing.effect_type == effect.effect_type:
                existing.duration = max(existing.duration, effect.duration)
                return
        self.status_effects.append(effect)

    def remove_status(self, stype: StatusEffectType):
        self.status_effects = [s for s in self.status_effects if s.effect_type != stype]

    def add_buff(self, stat: str, mult: float, duration: int):
        if stat in self.temp_buffs:
            old_mult, old_dur = self.temp_buffs[stat]
            self.temp_buffs[stat] = (max(old_mult, mult), max(old_dur, duration))
        else:
            self.temp_buffs[stat] = (mult, duration)

    def tick_status_effects(self) -> List[str]:
        """Process status effects at turn end. Returns list of messages."""
```

```
        messages = []
        to_remove = []
        for se in self.status_effects:
            msg = self._apply_status_tick(se)
            if msg:
                messages.append(msg)
            if not se.tick():
                to_remove.append(se)
        self.status_effects = [s for s in self.status_effects if s not in to_remove]
        for expired in to_remove:
            messages.append(f"  {self.name}'s {expired.effect_type.value} wore off.")
        return messages

    def _apply_status_tick(self, se: StatusEffect) -> Optional[str]:
        if se.effect_type == StatusEffectType.POISON:
            dmg = max(1, int(self.max_hp * 0.05))
            self.take_damage(dmg)
            return f"  {self.name} is poisoned! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.VENOM:
            dmg = max(1, int(self.max_hp * 0.08))
            self.take_damage(dmg)
            return f"  {self.name} suffers venom! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.BURN:
            dmg = max(1, int(self.max_hp * 0.06))
            self.take_damage(dmg)
            return f"  {self.name} is burning! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.BLEED:
            dmg = max(1, int(self.max_hp * 0.04))
            self.take_damage(dmg)
            return f"  {self.name} bleeds! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.CURSE:
            dmg = max(1, int(self.max_hp * 0.03))
            self.take_damage(dmg)
            return f"  {self.name} is cursed! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.REGEN:
            heal = max(1, int(self.mdef * 1.5))
            self.heal(heal)
            return f"  {self.name} regenerates! (+{heal} HP)"
        elif se.effect_type == StatusEffectType.MANA_REGEN:
            mp = max(1, int(self.max_mp * 0.05))
```

```
            self.current_mp = min(self.max_mp, self.current_mp + mp)
            return f"  {self.name} regenerates MP! (+{mp} MP)"
        elif se.effect_type == StatusEffectType.DOOM:
            if se.duration <= 1:
                self.current_hp = 0
                self.is_ko = True
                return f"  ☠ DOOM strikes {self.name}! Instant KO!"
            else:
                return f"  □ DOOM countdown: {se.duration} turns left for {self.name}!"
        return None

    def tick_buffs(self) -> List[str]:
        messages = []
        expired = [k for k,(m,d) in self.temp_buffs.items() if d <= 1]
        self.temp_buffs = {k:(m,d-1) for k,(m,d) in self.temp_buffs.items() if d > 1}
        for k in expired:
            messages.append(f"  {self.name}'s {k} buff expired.")
        return messages

    def take_damage(self, amount: int):
        # Check shield
        if self.has_status(StatusEffectType.SHIELD):
            se = next(s for s in self.status_effects if s.effect_type ==
StatusEffectType.SHIELD)
            shield_val = int(se.power)
            if shield_val >= amount:
                se.power -= amount
                if se.power <= 0:
                    self.remove_status(StatusEffectType.SHIELD)
                return
            else:
                amount -= shield_val
                self.remove_status(StatusEffectType.SHIELD)
        self.current_hp = max(0, self.current_hp - amount)
        if self.current_hp == 0:
            self.is_ko = True

    def heal(self, amount: int):
        if not self.can_be_healed():
            return 0
```

```
        actual = min(amount, self.max_hp - self.current_hp)
        self.current_hp += actual
        if self.current_hp > 0:
            self.is_ko = False
            self.ko_turns = 0
        return actual

    def revive(self, hp_percent: int):
        self.is_ko = False
        self.ko_turns = 0
        self.current_hp = max(1, int(self.max_hp * hp_percent / 100))

    # ── Leveling ──────────────────────────────────────────────────────
    def gain_exp(self, amount: int) -> List[str]:
        messages = []
        self.exp += amount
        while self.exp >= self.exp_to_next:
            self.exp -= self.exp_to_next
            messages += self._level_up()
        return messages

    def _level_up(self) -> List[str]:
        self.level += 1
        self.exp_to_next = int(self.exp_to_next * 1.15)
        g = self.growth
        messages = [f"  ★ {self.name} reached Level {self.level}!"]

        def roll(rate, lo, hi):
            return random.randint(lo, hi) if random.random() < rate else 0

        hp_gain   = roll(g.hp,   5, 12)
        mp_gain   = roll(g.mp,   3, 8)
        patk_gain = roll(g.patk, 1, 4)
        matk_gain = roll(g.matk, 1, 4)
        pdef_gain = roll(g.pdef, 1, 3)
        mdef_gain = roll(g.mdef, 1, 3)
        spd_gain  = roll(g.spd,  1, 1)
        eva_gain  = roll(g.eva,  0, 0)  # tracked differently
        acc_gain  = 0
        crit_gain = 0
```

```
        self.base_stats.hp   += hp_gain
        self.base_stats.mp   += mp_gain
        self.base_stats.patk += patk_gain
        self.base_stats.matk += matk_gain
        self.base_stats.pdef += pdef_gain
        self.base_stats.mdef += mdef_gain
        self.base_stats.spd  += spd_gain
        if random.random() < g.eva:  self.base_stats.eva = min(0.95, self.base_stats.eva +
0.01)
        if random.random() < g.acc:  self.base_stats.acc = min(1.0,  self.base_stats.acc +
0.005)
        if random.random() < g.crit: self.base_stats.crit= min(0.95, self.base_stats.crit+
0.005)

        # Heal to full on level up
        self.current_hp = self.base_stats.hp
        self.current_mp = self.base_stats.mp

        gains = []
        if hp_gain:   gains.append(f"HP+{hp_gain}")
        if mp_gain:   gains.append(f"MP+{mp_gain}")
        if patk_gain: gains.append(f"PATK+{patk_gain}")
        if matk_gain: gains.append(f"MATK+{matk_gain}")
        if pdef_gain: gains.append(f"PDEF+{pdef_gain}")
        if mdef_gain: gains.append(f"MDEF+{mdef_gain}")
        if spd_gain:  gains.append(f"SPD+{spd_gain}")
        if gains:
            messages.append(f"    Stats: {', '.join(gains)}")

        # Unlock skills
        unlock_msgs = self._check_skill_unlocks()
        messages.extend(unlock_msgs)
        return messages

    def _check_skill_unlocks(self) -> List[str]:
        msgs = []
        # Intermediate: lv 5 and 10
        intermediate_ids = [sid for sid in self.skill_pool
                            if sid in SKILL_DB and SKILL_DB[sid].tier ==
```

```
SkillTier.INTERMEDIATE]
        if self.level == 5 and len(intermediate_ids) >= 1:
            sid = intermediate_ids[0]
            if sid not in self.unlocked_skills:
                self.unlocked_skills.append(sid)
                msgs.append(f"    ✦ Skill Unlocked: {SKILL_DB[sid].name}!")
        if self.level == 10 and len(intermediate_ids) >= 2:
            sid = intermediate_ids[1]
            if sid not in self.unlocked_skills:
                self.unlocked_skills.append(sid)
                msgs.append(f"    ✦ Skill Unlocked: {SKILL_DB[sid].name}!")
        # Ultimate: lv 20
        if self.level == 20:
            ultimate_ids = [sid for sid in self.skill_pool
                            if sid in SKILL_DB and SKILL_DB[sid].tier == SkillTier.ULTIMATE]
            if ultimate_ids:
                sid = ultimate_ids[0]
                if sid not in self.unlocked_skills:
                    self.unlocked_skills.append(sid)
                    msgs.append(f"    ★ ULTIMATE SKILL UNLOCKED: {SKILL_DB[sid].name}!!")
        return msgs

    def hp_bar(self, width=20) -> str:
        ratio = self.current_hp / max(1, self.max_hp)
        filled = int(ratio * width)
        bar = "█" * filled + "░" * (width - filled)
        return f"[{bar}] {self.current_hp}/{self.max_hp}"

    def mp_bar(self, width=10) -> str:
        ratio = self.current_mp / max(1, self.max_mp)
        filled = int(ratio * width)
        bar = "▓" * filled + "░" * (width - filled)
        return f"[{bar}] {self.current_mp}/{self.max_mp}"

    def status_str(self) -> str:
        if not self.status_effects:
            return ""
        tags = [se.effect_type.value[:3].upper() for se in self.status_effects]
        return " [" + ",".join(tags) + "]"
```

```
    def short_status(self) -> str:
        if self.is_dead:   return "DEAD"
        if self.is_ko:     return f"KO({self.ko_turns})"
        return "OK"


def create_player_character(name: str, job: JobClass,
                            weapon: WeaponType, element: Element) -> Character:
    base = JOB_BASE_STATS[job].copy()
    growth = JOB_GROWTH[job]
    pool = JOB_SKILL_POOL[job]

    # Pick 2 basic, 2 intermediate, 1 ultimate from pool
    basics = [sid for sid in pool if SKILL_DB[sid].tier == SkillTier.BASIC][:4]
    inters = [sid for sid in pool if SKILL_DB[sid].tier == SkillTier.INTERMEDIATE][:4]
    ultims = [sid for sid in pool if SKILL_DB[sid].tier == SkillTier.ULTIMATE][:2]

    chosen_basic = random.sample(basics, min(2, len(basics)))
    chosen_inter = random.sample(inters, min(2, len(inters)))
    chosen_ultim = random.sample(ultims, min(1, len(ultims)))

    skill_pool = chosen_basic + chosen_inter + chosen_ultim
    unlocked = chosen_basic[:]  # Only basics at level 1

    char = Character(
        name=name, job=job, weapon=weapon, element=element, rarity=0,
        base_stats=base, growth=growth,
        skill_pool=skill_pool, unlocked_skills=unlocked
    )
    return char
```

Revision #1
Created 2026-03-18 16:20:59 UTC by Samuel Lee
Updated 2026-03-18 16:21:25 UTC by Samuel Lee