monster_unit.py & monsters_db.py

monster_unit.py

```
"""Monster combat unit - wraps MonsterTemplate with battle state."""
from __future__ import annotations
import random
from dataclasses import dataclass, field
from typing import List, Optional
from data_types import *
from monsters_db import MonsterTemplate
from skills_db import SKILL_DB


@dataclass
class MonsterUnit:
    template: MonsterTemplate
    instance_name: str  # e.g. "Goblin A"

    current_hp: int = 0
    current_mp: int = 0
    shield_points: int = 0
    is_broken: bool = False
    break_turns: int = 0
    status_effects: List[StatusEffect] = field(default_factory=list)
    temp_buffs: dict = field(default_factory=dict)

    def __post_init__(self):
        self.current_hp = self.template.base_stats.hp
        self.current_mp = self.template.base_stats.mp
        self.shield_points = self.template.shield_points

    @property
    def is_dead(self): return self.current_hp <= 0
```

```
    @property
    def patk(self):
        v = self.template.base_stats.patk
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.ATTACK_DOWN: v = int(v*0.7)
        return v

    @property
    def matk(self):
        v = self.template.base_stats.matk
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.MAGIC_DOWN: v = int(v*0.7)
        return v

    @property
    def pdef(self):
        v = self.template.base_stats.pdef
        if self.is_broken: return 0
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.DEFENSE_DOWN: v = int(v*0.7)
        return v

    @property
    def mdef(self):
        v = self.template.base_stats.mdef
        if self.is_broken: return 0
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.MAGIC_DOWN: v = int(v*0.7)
        return v

    @property
    def spd(self):
        v = self.template.base_stats.spd
        for se in self.status_effects:
            if se.effect_type == StatusEffectType.SPEED_DOWN: v = int(v*0.7)
            if se.effect_type == StatusEffectType.HASTE: v = int(v*1.5)
        return v

    @property
    def eva(self):
```

```
    if self.is_broken: return 0.0
    v = self.template.base_stats.eva
    if self.has_status(StatusEffectType.BLIND): v = max(0, v - 0.3)
    return v

@property
def acc(self):
    v = self.template.base_stats.acc
    if self.has_status(StatusEffectType.ACCURACY_DOWN): v *= 0.6
    return v

@property
def max_hp(self): return self.template.base_stats.hp

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):
    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 is_incapacitated(self) -> bool:
    incap = {StatusEffectType.SLEEP, StatusEffectType.STUN,
             StatusEffectType.FREEZE, StatusEffectType.PARALYZE,
             StatusEffectType.PETRIFY}
    return any(se.effect_type in incap for se in self.status_effects) or self.is_broken

def take_damage(self, amount: int) -> int:
    """Returns actual damage dealt."""
    actual = min(amount, self.current_hp)
    self.current_hp -= actual
    return actual

def hit_weakness(self, attack_weapon: Optional[WeaponType],
```

```
                attack_element: Optional[Element]) -> bool:
        weaknesses = self.template.weaknesses
        if attack_weapon and attack_weapon in weaknesses:
            return True
        if attack_element and attack_element != Element.NONE and attack_element in weaknesses:
            return True
        return False

    def reduce_shield(self, hits: int) -> bool:
        """Returns True if break triggered."""
        if self.is_broken: return False
        self.shield_points = max(0, self.shield_points - hits)
        if self.shield_points == 0:
            self.is_broken = True
            self.break_turns = 1
            return True
        return False

    def tick_break(self):
        if self.is_broken:
            self.break_turns -= 1
            if self.break_turns <= 0:
                self.is_broken = False
                self.shield_points = self.template.shield_points // 2 + 1

    def tick_status_effects(self) -> List[str]:
        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.instance_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.instance_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.instance_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.instance_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.instance_name} bleeds! (-{dmg} HP)"
        elif se.effect_type == StatusEffectType.DOOM:
            if se.duration <= 1:
                self.current_hp = 0
                return f"  ☠ DOOM strikes {self.instance_name}!"
            return f"  □ DOOM countdown: {se.duration} turns for {self.instance_name}."
        return None

    def choose_action(self, party: list) -> dict:
        """AI decides what to do."""
        ai = self.template.ai_type
        skill_ids = self.template.skill_ids
        # Filter to valid skills in DB
        valid_skills = [sid for sid in skill_ids if sid in SKILL_DB]

        hp_ratio = self.current_hp / max(1, self.max_hp)

        # Simple AI logic
        if ai == AIType.AGGRESSIVE:
            if valid_skills and random.random() < 0.4:
                return {"action": "skill", "skill_id": random.choice(valid_skills)}
            return {"action": "attack"}

        elif ai == AIType.DEFENSIVE:
            if hp_ratio < 0.3 and random.random() < 0.5:
                return {"action": "defend"}
            if valid_skills and random.random() < 0.3:
```

```
            return {"action": "skill", "skill_id": random.choice(valid_skills)}
        return {"action": "attack"}

    elif ai == AIType.SUPPORT:
        if valid_skills and random.random() < 0.6:
            skill_id = random.choice(valid_skills)
            return {"action": "skill", "skill_id": skill_id}
        return {"action": "attack"}

    elif ai == AIType.TACTICAL:
        if hp_ratio > 0.7:
            if valid_skills and random.random() < 0.5:
                return {"action": "skill", "skill_id": random.choice(valid_skills)}
            return {"action": "attack"}
        elif hp_ratio > 0.4:
            if valid_skills and random.random() < 0.35:
                return {"action": "skill", "skill_id": random.choice(valid_skills)}
            return {"action": "attack"}
        else:
            if valid_skills and random.random() < 0.7:
                return {"action": "skill", "skill_id": random.choice(valid_skills)}
            return {"action": "attack"}

    elif ai == AIType.BERSERKER:
        if valid_skills and random.random() < 0.5:
            return {"action": "skill", "skill_id": random.choice(valid_skills)}
        return {"action": "attack"}

    else:  # RANDOM / BALANCED
        r = random.random()
        if valid_skills and r < 0.35:
            return {"action": "skill", "skill_id": random.choice(valid_skills)}
        return {"action": "attack"}

def shield_bar(self) -> str:
    sp = self.shield_points
    total = self.template.shield_points
    filled = "◆" * sp + "◇" * (total - sp)
    return f"[{filled}]"
```

```
    def hp_bar(self, width=16) -> 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 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) + "]"
```

monsters_db.py

```
"""Monster database with 50 monsters."""
from __future__ import annotations
from data_types import *
from dataclasses import dataclass, field
from typing import List, Dict, Tuple
import random


@dataclass
class MonsterTemplate:
    id: int
    name: str
    level: int
    base_stats: Stats
    weaknesses: List  # List of WeaponType | Element
    shield_points: int
    skill_ids: List[int]
    ai_type: AIType
    exp_reward: int
    tier: str  # early / mid / high / boss


def build_monster_db() -> Dict[int, MonsterTemplate]:
    db = {}

    def m(id, name, lv, hp, mp, pa, ma, pd, md, spd, eva, acc, crit,
```

```
      weak, sp, skills, ai, exp, tier):
    db[id] = MonsterTemplate(
        id, name, lv,
        Stats(hp, mp, pa, ma, pd, md, spd, eva, acc, crit),
        weak, sp, skills, ai, exp, tier
    )

W = WeaponType; E = Element; AI = AIType

# ═══════════════════════════════════════════════════════
# EARLY MONSTERS (lv 1-10)
# ═══════════════════════════════════════════════════════
m(1,"Goblin",           2, 80,  10, 12, 4,  8,  5,  8,  0.08,0.85,0.05,
  [W.SWORD,E.FIRE,W.BOW],         3, [1,2],    AI.BALANCED, 15, "early")
m(2,"Goblin Archer",    3, 70,  10, 10, 4,  6,  4,  10, 0.12,0.88,0.08,
  [W.SWORD,E.FIRE],               2, [1,20],   AI.AGGRESSIVE, 20, "early")
m(3,"Goblin Shaman",    4, 65,  30, 8,  14, 5,  10, 7,  0.05,0.82,0.04,
  [W.SWORD,E.LIGHT],              2, [50,76],  AI.SUPPORT, 25, "early")
m(4,"Wolf",             2, 90,  0,  14, 0,  6,  4,  14, 0.15,0.9, 0.1,
  [E.FIRE,W.SPEAR],               2, [1,15],   AI.AGGRESSIVE, 18, "early")
m(5,"Dire Wolf",        5, 150, 0,  20, 0,  10, 6,  16, 0.18,0.88,0.15,
  [E.FIRE,W.SPEAR,W.AXE],         3, [1,2,15], AI.AGGRESSIVE, 40, "early")
m(6,"Slime",            1, 60,  0,  6,  0,  12, 8,  4,  0.0, 0.8, 0.02,
  [E.FIRE,W.SWORD],               2, [1],      AI.RANDOM, 10, "early")
m(7,"Fire Slime",       4, 80,  10, 8,  10, 6,  14, 5,  0.0, 0.8, 0.03,
  [E.ICE,W.SPEAR],                2, [50,51],  AI.BALANCED, 30, "early")
m(8,"Ice Slime",        4, 80,  10, 8,  10, 6,  8,  5,  0.0, 0.8, 0.03,
  [E.FIRE,W.AXE],                 2, [55,56],  AI.BALANCED, 30, "early")
m(9,"Bat",              2, 55,  0,  8,  0,  4,  4,  12, 0.20,0.85,0.08,
  [E.LIGHTNING,W.BOW],            2, [1,15],   AI.AGGRESSIVE, 12, "early")
m(10,"Giant Bat",       5, 120, 0,  18, 0,  8,  6,  15, 0.22,0.87,0.12,
  [E.LIGHTNING,W.BOW,W.AXE],      3, [1,2,3],  AI.AGGRESSIVE, 45, "early")

# ═══════════════════════════════════════════════════════
# MID MONSTERS (lv 8-20)
# ═══════════════════════════════════════════════════════
m(11,"Orc",             8, 280, 10, 28, 5,  18, 10, 8,  0.05,0.82,0.08,
  [W.SPEAR,E.FIRE],               3, [2,11,30], AI.AGGRESSIVE, 80, "mid")
m(12,"Orc Warrior",     10,350, 15, 35, 8,  22, 12, 9,  0.07,0.83,0.1,
  [W.SPEAR,E.FIRE,E.LIGHTNING],   3, [2,11,13], AI.AGGRESSIVE, 110, "mid")
```

```
m(13,"Orc Shaman",      10,250, 60, 20, 25, 14, 18, 7,  0.05,0.82,0.06,
  [W.SWORD,E.LIGHT],              2, [50,77,76],AI.SUPPORT, 100, "mid")
m(14,"Lizardman",       9, 310, 20, 30, 10, 16, 14, 12, 0.1, 0.85,0.1,
  [E.ICE,W.AXE],                  3, [1,3,29],  AI.BALANCED, 90, "mid")
m(15,"Harpy",           10,260, 20, 25, 15, 12, 16, 18, 0.18,0.87,0.12,
  [E.LIGHTNING,W.BOW,W.AXE],      2, [1,65,66], AI.AGGRESSIVE, 105, "mid")
m(16,"Skeleton",        8, 240, 0,  24, 0,  10, 18, 10, 0.08,0.85,0.12,
  [E.LIGHT,W.AXE,W.SWORD],        3, [1,2,3],   AI.AGGRESSIVE, 75, "mid")
m(17,"Zombie",          9, 320, 0,  22, 10, 14, 12, 5,  0.05,0.78,0.05,
  [E.FIRE,E.LIGHT,W.AXE],         2, [1,16],    AI.AGGRESSIVE, 85, "mid")
m(18,"Ghoul",           11,380, 20, 28, 14, 16, 16, 8,  0.1, 0.82,0.1,
  [E.FIRE,E.LIGHT],               3, [1,2,90],  AI.AGGRESSIVE, 120, "mid")
m(19,"Wraith",          12,300, 50, 18, 22, 8,  24, 10, 0.15,0.85,0.1,
  [E.LIGHT,W.SWORD],              3, [75,77,79],AI.TACTICAL, 130, "mid")
m(20,"Dark Mage",       14,320, 80, 15, 35, 10, 28, 9,  0.08,0.87,0.08,
  [E.LIGHT,W.SWORD],              2, [75,77,79,76],AI.TACTICAL, 160, "mid")
m(21,"Stone Golem",     12,500, 0,  35, 0,  30, 25, 4,  0.0, 0.75,0.05,
  [E.LIGHTNING,W.AXE],            4, [2,70,71], AI.DEFENSIVE, 150, "mid")
m(22,"Earth Golem",     15,600, 0,  38, 0,  32, 28, 3,  0.0, 0.75,0.05,
  [E.LIGHTNING,W.AXE,E.ICE],      4, [2,70,72], AI.DEFENSIVE, 200, "mid")
m(23,"Bandit",          7, 220, 10, 22, 8,  12, 10, 14, 0.15,0.88,0.12,
  [E.LIGHT,W.SPEAR],              2, [1,15,16], AI.BALANCED, 70, "mid")
m(24,"Bandit Leader",   11,350, 20, 30, 12, 18, 14, 16, 0.18,0.88,0.15,
  [E.LIGHT,W.SPEAR,E.FIRE],       3, [1,2,17],  AI.TACTICAL, 140, "mid")
m(25,"Dark Knight",     15,480, 30, 38, 15, 26, 20, 11, 0.1, 0.85,0.12,
  [E.LIGHT,W.SPEAR],              4, [8,9,75,77],AI.TACTICAL, 210, "mid")

# ═══════════════════════════════════════════════════
# HIGH MONSTERS (lv 18-35)
# ═══════════════════════════════════════════════════
m(26,"Demon",           18,700, 80, 48, 42, 28, 32, 14, 0.12,0.87,0.15,
  [E.LIGHT,W.SWORD],              4, [75,77,79,91],AI.AGGRESSIVE, 350, "high")
m(27,"High Demon",      22,900, 100,55, 50, 32, 38, 15, 0.15,0.87,0.18,
  [E.LIGHT,W.SPEAR],              4, [79,77,91,99],AI.AGGRESSIVE, 500, "high")
m(28,"Vampire",         20,650, 60, 42, 38, 22, 30, 16, 0.2, 0.88,0.2,
  [E.LIGHT,W.AXE],                3, [90,15,75,77],AI.TACTICAL, 420, "high")
m(29,"Lich",            25,800, 120,25, 65, 18, 55, 10, 0.1, 0.88,0.1,
  [E.LIGHT,W.SWORD,E.FIRE],       3, [79,99,77,76],AI.TACTICAL, 600, "high")
m(30,"Basilisk",        20,750, 0,  45, 0,  38, 34, 9,  0.05,0.82,0.08,
  [E.FIRE,W.AXE,E.LIGHTNING],     4, [1,2,70,72], AI.DEFENSIVE, 380, "high")
```

```
m(31,"Chimera",         22,850, 40, 50, 35, 30, 30, 12, 0.12,0.85,0.15,
  [E.ICE,W.SPEAR,W.BOW],          4, [1,2,50,65], AI.BALANCED, 450, "high")
m(32,"Hydra",           24,1000,30, 48, 20, 35, 28, 8,  0.08,0.82,0.1,
  [E.LIGHTNING,W.SWORD,E.FIRE],   5, [1,2,3,90],  AI.AGGRESSIVE, 520, "high")
m(33,"Medusa",          21,700, 60, 40, 45, 25, 35, 14, 0.15,0.88,0.15,
  [E.FIRE,W.BOW],                 3, [76,77,90,99],AI.TACTICAL, 430, "high")
m(34,"Manticore",       23,880, 20, 52, 15, 32, 26, 16, 0.18,0.87,0.18,
  [E.ICE,W.SPEAR,W.SWORD],        4, [1,2,20,22], AI.AGGRESSIVE, 480, "high")
m(35,"Golem King",      26,1200,0,  58, 0,  45, 40, 5,  0.0, 0.78,0.05,
  [E.LIGHTNING,W.AXE,E.ICE],      5, [2,70,72,92],AI.DEFENSIVE, 650, "high")

# ═══════════════════════════════════════════════════
# DRAGONS (lv 30-50)
# ═══════════════════════════════════════════════════
m(36,"Dragon",          30,2000,80, 70, 60, 50, 55, 16, 0.12,0.87,0.15,
  [E.ICE,W.SPEAR],                5, [1,2,50,51,52],AI.TACTICAL, 1000, "high")
m(37,"Fire Dragon",     32,2200,80, 72, 65, 52, 50, 18, 0.15,0.87,0.18,
  [E.ICE,W.SPEAR,W.AXE],          5, [50,51,52,53,54],AI.AGGRESSIVE, 1200, "high")
m(38,"Ice Dragon",      32,2200,80, 65, 70, 50, 58, 14, 0.1, 0.85,0.15,
  [E.FIRE,W.SPEAR,W.AXE],         5, [55,56,57,58,59],AI.AGGRESSIVE, 1200, "high")
m(39,"Thunder Dragon",  33,2300,80, 68, 72, 48, 60, 19, 0.15,0.88,0.18,
  [E.EARTH,W.SPEAR,W.BOW],        5, [60,61,62,63,64],AI.AGGRESSIVE, 1300, "high")
m(40,"Earth Dragon",    33,2400,60, 75, 55, 58, 50, 12, 0.08,0.83,0.1,
  [E.LIGHTNING,W.AXE,W.SPEAR],    5, [70,71,72,73,74],AI.DEFENSIVE, 1300, "high")
m(41,"Shadow Dragon",   35,2500,100,70, 75, 50, 60, 18, 0.2, 0.87,0.2,
  [E.LIGHT,W.SPEAR],              5, [75,77,79,91,99],AI.TACTICAL, 1500, "high")
m(42,"Holy Dragon",     38,2800,100,65, 80, 55, 65, 16, 0.12,0.88,0.15,
  [E.DARK,W.AXE],                 5, [80,82,84,47,44],AI.TACTICAL, 1800, "high")

# ═══════════════════════════════════════════════════
# ELITE / BOSS-TYPE (lv 40-60)
# ═══════════════════════════════════════════════════
m(43,"Arch Demon",      40,4000,150,85, 90, 60, 75, 18, 0.18,0.88,0.22,
  [E.LIGHT,W.SPEAR,W.SWORD],      5, [79,77,91,99,14],AI.TACTICAL, 2500, "boss")
m(44,"Fallen Angel",    42,3800,150,80, 95, 55, 80, 20, 0.2, 0.9, 0.2,
  [E.DARK,W.BOW,W.SWORD],         5, [80,84,47,46,82],AI.TACTICAL, 2600, "boss")
m(45,"Ancient Golem",   45,5000,0,  90, 0,  70, 65, 6,  0.0, 0.8, 0.05,
  [E.LIGHTNING,W.AXE],            5, [2,72,74,92,70],AI.DEFENSIVE, 3000, "boss")
m(46,"Chaos Dragon",    50,6000,200,100,100,75, 85, 20, 0.25,0.88,0.25,
  [E.LIGHT,E.ICE,W.SPEAR],        5, [54,59,64,74,79,14],AI.TACTICAL, 4000, "boss")
```

```
    m(47,"Death Knight",    45,4500,100,95, 70, 65, 72, 16, 0.15,0.88,0.18,
      [E.LIGHT,W.SPEAR,W.SWORD],      5, [8,9,75,79,90,99],AI.TACTICAL, 3200, "boss")
    m(48,"Storm Titan",     48,5500,150,80, 105,65, 80, 18, 0.18,0.88,0.2,
      [E.EARTH,W.AXE,W.BOW],          5, [60,62,63,64,65],AI.AGGRESSIVE, 3800, "boss")
    m(49,"Abyssal Horror",  52,7000,200,95, 110,70, 90, 14, 0.15,0.87,0.2,
      [E.LIGHT,W.SPEAR,W.SWORD],      5, [79,99,77,76,19,14],AI.TACTICAL, 5000, "boss")
    m(50,"World Eater",     60,15000,300,120,130,90, 110,20, 0.3, 0.88,0.3,
      [E.LIGHT,E.ICE,W.SPEAR,W.SWORD],5, [14,54,59,64,74,79,100,34],AI.TACTICAL, 10000,"boss")

    return db

MONSTER_DB: Dict[int, MonsterTemplate] = build_monster_db()

# Difficulty tiers for battle selection
EARLY_MONSTERS = [id for id,m in MONSTER_DB.items() if m.tier == "early"]
MID_MONSTERS   = [id for id,m in MONSTER_DB.items() if m.tier == "mid"]
HIGH_MONSTERS  = [id for id,m in MONSTER_DB.items() if m.tier == "high"]
BOSS_MONSTERS  = [id for id,m in MONSTER_DB.items() if m.tier == "boss"]


def get_encounter(battle_number: int) -> List[MonsterTemplate]:
    """Return a list of monsters for the given battle number."""
    import copy
    if battle_number <= 5:
        pool, count = EARLY_MONSTERS, random.randint(1, 2)
    elif battle_number <= 15:
        pool = EARLY_MONSTERS + MID_MONSTERS[:5]
        count = random.randint(1, 3)
    elif battle_number <= 30:
        pool, count = MID_MONSTERS, random.randint(1, 3)
    elif battle_number <= 50:
        pool = MID_MONSTERS[5:] + HIGH_MONSTERS[:10]
        count = random.randint(1, 3)
    elif battle_number <= 75:
        pool, count = HIGH_MONSTERS, random.randint(1, 3)
    else:
        pool = HIGH_MONSTERS + BOSS_MONSTERS
        count = random.randint(1, 2)

    chosen = random.choices(pool, k=count)
```

```
    # Scale stats slightly based on battle number
    result = []
    for mid in chosen:
        t = copy.deepcopy(MONSTER_DB[mid])
        # Scale-up beyond base level
        extra_levels = max(0, (battle_number // 5) - t.level // 5)
        scale = 1.0 + extra_levels * 0.05
        t.base_stats.hp   = int(t.base_stats.hp   * scale)
        t.base_stats.patk = int(t.base_stats.patk * scale)
        t.base_stats.matk = int(t.base_stats.matk * scale)
        t.base_stats.pdef = int(t.base_stats.pdef * scale)
        t.base_stats.mdef = int(t.base_stats.mdef * scale)
        result.append(t)
    return result
```

Revision #1
Created 2026-03-18 16:24:12 UTC by Samuel Lee
Updated 2026-03-18 16:25:22 UTC by Samuel Lee