[Python] Turn-Based Battle Game

- main.py
- battle.py
- character.py
- companions.py
- data_types.py
- game_state.py
- items_db.py
- monster_unit.py & monsters_db.py
- skills_db.py

main.py

```
"""Main game entry point: title screen, character creation, game loop."""
import sys
import os

sys.path.insert(0, os.path.dirname(__file__))

from data_types import *
from character import create_player_character, JOB_BASE_STATS
from game_state import GameState
from skills_db import SKILL_DB, JOB_SKILL_POOL


def clear():
    os.system('clear' if os.name == 'posix' else 'cls')


def print_title():
    clear()
    print("""
╔══════════════════════════════════════════════════════════╗
║                                                          ║
║          ✦  CHRONICLES OF THE ETERNAL TURN  ✦            ║
║                  A JRPG CLI ADVENTURE                    ║
║                                                          ║
╚══════════════════════════════════════════════════════════╝
""")


def create_character() -> tuple:
    clear()
    print("╔══════════════════════════════════════════════════════════╗")
    print("║                  CHARACTER CREATION                      ║")
    print("╚══════════════════════════════════════════════════════════╝")

    while True:
```

```
    name = input("\n  Enter your hero's name: ").strip()
    if name:
        break
    print("  Please enter a name.")

weapons = list(WeaponType)
weapon_descs = {
    WeaponType.SWORD:   "Balanced. Warriors, Knights, Paladins",
    WeaponType.SPEAR:   "Long reach. Spearman, Dragoon",
    WeaponType.AXE:     "High power. Berserker, Monk",
    WeaponType.DAGGER:  "Fast & precise. Assassin",
    WeaponType.BOW:     "Ranged. Ranger, Hunter",
    WeaponType.STAFF:   "Magical focus. All Mages and Healers",
}
print("\n  ── Choose Your Weapon ──")
for i, w in enumerate(weapons):
    print(f"    [{i+1}] {w.value:10s} - {weapon_descs[w]}")
while True:
    try:
        wi = int(input("  > ")) - 1
        if 0 <= wi < len(weapons): break
    except ValueError: pass
    print("  Invalid.")
weapon = weapons[wi]

elements = [e for e in Element if e != Element.NONE]
elem_descs = {
    Element.FIRE:      "Offensive burn effects",
    Element.ICE:       "Freeze and slow foes",
    Element.LIGHTNING: "Chain damage, high speed",
    Element.WIND:      "Evasive, speed-based",
    Element.EARTH:     "Defense-break, sturdy",
    Element.LIGHT:     "Holy power, vs Undead/Dark",
    Element.DARK:      "Debuff and drain",
}
print("\n  ── Choose Your Element ──")
for i, e in enumerate(elements):
    print(f"    [{i+1}] {e.value:10s} - {elem_descs[e]}")
while True:
    try:
```

```
        ei = int(input("  > ")) - 1
        if 0 <= ei < len(elements): break
    except ValueError: pass
    print("  Invalid.")
element = elements[ei]

all_jobs = list(JobClass)
job_weapon_affinity = {
    WeaponType.SWORD:  [JobClass.WARRIOR, JobClass.KNIGHT, JobClass.PALADIN,
                        JobClass.ASSASSIN, JobClass.BERSERKER, JobClass.DRAGOON,
                        JobClass.DARK_MAGE, JobClass.LIGHT_MAGE],
    WeaponType.SPEAR:  [JobClass.SPEARMAN, JobClass.DRAGOON, JobClass.WARRIOR,
                        JobClass.KNIGHT, JobClass.RANGER, JobClass.HUNTER],
    WeaponType.AXE:    [JobClass.BERSERKER, JobClass.WARRIOR, JobClass.MONK,
                        JobClass.EARTH_MAGE, JobClass.DRAGOON, JobClass.KNIGHT],
    WeaponType.DAGGER: [JobClass.ASSASSIN, JobClass.RANGER, JobClass.HUNTER,
                        JobClass.WIND_MAGE, JobClass.DARK_MAGE, JobClass.MONK],
    WeaponType.BOW:    [JobClass.RANGER, JobClass.HUNTER, JobClass.ASSASSIN,
                        JobClass.STORM_MAGE, JobClass.WIND_MAGE, JobClass.LIGHT_MAGE],
    WeaponType.STAFF:  [JobClass.CLERIC, JobClass.PRIEST, JobClass.FIRE_MAGE,
                        JobClass.ICE_MAGE, JobClass.STORM_MAGE, JobClass.WIND_MAGE,
                        JobClass.EARTH_MAGE, JobClass.DARK_MAGE, JobClass.LIGHT_MAGE,
                        JobClass.ARCANE_SAGE, JobClass.PALADIN],
}

suggested = job_weapon_affinity.get(weapon, all_jobs)
job_info = {
    JobClass.WARRIOR:    "Melee fighter, high PATK, balanced",
    JobClass.KNIGHT:     "Tank, high HP/DEF, Shield Bash",
    JobClass.PALADIN:    "Holy warrior, heal & attack",
    JobClass.BERSERKER:  "Rage fighter, highest PATK, low DEF",
    JobClass.ASSASSIN:   "Stealth, high CRIT/EVA, poison",
    JobClass.RANGER:     "Bow specialist, multi-hit, area",
    JobClass.HUNTER:     "Traps, dragon slayer, beast lore",
    JobClass.SPEARMAN:   "Spear master, sweep attacks",
    JobClass.DRAGOON:    "Dragon knight, jump attacks",
    JobClass.MONK:       "Unarmed master, combo strikes",
    JobClass.CLERIC:     "Healer & smiter, Light magic",
    JobClass.PRIEST:     "Pure healer, mass heals, revival",
    JobClass.FIRE_MAGE:  "Fire magic, burn effects",
```

```
    JobClass.ICE_MAGE:   "Ice magic, freeze/slow",
    JobClass.STORM_MAGE: "Lightning magic, chain effects",
    JobClass.WIND_MAGE:  "Wind magic, speed buffs",
    JobClass.EARTH_MAGE: "Earth magic, high power AOE",
    JobClass.DARK_MAGE:  "Dark magic, debuffs, drain",
    JobClass.LIGHT_MAGE: "Light magic, barriers, blind",
    JobClass.ARCANE_SAGE:"All-element magic, time manipulation",
}

print(f"\n  ── Choose Your Job (Weapon: {weapon.value}) ──")
print(f"  Recommended jobs shown. Enter A for all jobs.")
print()
for i, job in enumerate(suggested):
    print(f"    [{i+1:2d}] {job.value:15s} - {job_info.get(job,'')}")
print(f"    [ A] Show ALL {len(all_jobs)} jobs")

job = None
while True:
    raw = input("  > ").strip().upper()
    if raw == "A":
        print("\n  ALL JOBS:")
        for i, j in enumerate(all_jobs):
            print(f"    [{i+1:2d}] {j.value:15s} - {job_info.get(j,'')}")
        while True:
            try:
                ji = int(input("  > ")) - 1
                if 0 <= ji < len(all_jobs):
                    job = all_jobs[ji]
                    break
            except ValueError: pass
            print("  Invalid.")
        break
    else:
        try:
            idx = int(raw) - 1
            if 0 <= idx < len(suggested):
                job = suggested[idx]
                break
        except ValueError: pass
        print("  Invalid choice.")
```

```
    # Skill preview
    print(f"\n  ── Skill Preview for {job.value} ──")
    pool = JOB_SKILL_POOL[job]
    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]
    print("  Basic (start with 2):")
    for sid in basics:
        sk = SKILL_DB[sid]
        print(f"    • {sk.name:20s} MP:{sk.mp_cost:3d} PWR:{sk.power} - {sk.description}")
    print("  Intermediate (unlock lv5/10):")
    for sid in inters:
        sk = SKILL_DB[sid]
        print(f"    • {sk.name:20s} MP:{sk.mp_cost:3d} PWR:{sk.power} - {sk.description}")
    print("  Ultimate (unlock lv20):")
    for sid in ultims:
        sk = SKILL_DB[sid]
        print(f"    • {sk.name:20s} MP:{sk.mp_cost:3d} PWR:{sk.power} - {sk.description}")

    # Summary
    stats = JOB_BASE_STATS[job]
    print(f"\n  ── Summary ──")
    print(f"  Name: {name}  |  Job: {job.value}  |  Weapon: {weapon.value}  |  Element:
{element.value}")
    print(f"  HP:{stats.hp} MP:{stats.mp} PATK:{stats.patk} MATK:{stats.matk} "
          f"PDEF:{stats.pdef} MDEF:{stats.mdef} SPD:{stats.spd}")

    confirm = input("\n  Confirm? (Y/N) > ").strip().upper()
    if confirm != "Y":
        return create_character()

    return name, job, weapon, element


def main():
    print_title()
    print("""
  Welcome to Chronicles of the Eternal Turn!
```

```
  OBJECTIVE: Survive as many battles as you can.
             Collect companions, level up, grow powerful.

  COMBAT TIPS:
  • Attack enemies' Weaknesses to reduce their Shield Points
  • When Shield = 0, enemy is BROKEN (50% more damage, can't act)
  • Use Defend to reduce incoming damage and gain turn priority
  • If a party member is KO'd, revive within 3 turns or they DIE
  • If YOUR character dies → GAME OVER

  COMPANION SYSTEM:
  • Win battles to earn companions (3★/4★/5★)
  • Party maximum: 4 members
  • Companions can permanently die in battle
""")
    input("  [Press Enter to begin]")

    name, job, weapon, element = create_character()
    clear()

    gs = GameState()
    player = create_player_character(name, job, weapon, element)
    gs.player = player
    gs.party = [player]
    gs.all_party_members = [player]

    print(f"\n  ✦ {player.name} the {player.job.value} sets forth!")
    print(f"  Starting skills: " +
          ", ".join(SKILL_DB[s].name for s in player.unlocked_skills if s in SKILL_DB))
    input("\n  [Press Enter to begin your adventure]")

    # Main game loop
    while not gs.game_over:
        print(f"\n{'='*65}")
        print(f"  ✦ BATTLE {gs.battle_number + 1}")

        if gs.battle_number > 0:
            cont = gs.between_battles_menu()
            if not cont:
                print("\n  Thanks for playing! Farewell, hero.")
```

```
                print(f"  Final record: {gs.battle_number} battles | {gs.player.name}
Lv{gs.player.level}")
                break

        alive = [ch for ch in gs.party if not ch.is_dead]
        if not alive:
            gs.game_over = True
            break

        survived = gs.run_battle()

        if not survived or gs.game_over:
            gs.show_game_over_screen()
            break

    if not gs.game_over and gs.battle_number > 0:
        print(f"\n  Journey ended. {gs.battle_number} battles completed.")
        print(f"  {gs.player.name} reached Level {gs.player.level}. Well done!")


if __name__ == "__main__":
    main()
```

battle.py

```
"""Battle engine: handles all combat calculations and the battle loop."""
from __future__ import annotations
import random
import math
from typing import List, Optional, Tuple, Dict
from data_types import *
from character import Character
from monster_unit import MonsterUnit
from monsters_db import MonsterTemplate
from skills_db import SKILL_DB
from items_db import ITEM_DB


def _clear():
    import os; os.system('clear' if os.name == 'posix' else 'cls')


def calculate_physical_damage(attacker_patk: int, target_pdef: int,
                              power: float = 1.0, crit: bool = False,
                              element_bonus: float = 1.0,
                              weakness_bonus: float = 1.0,
                              break_bonus: float = 1.0) -> int:
    base = max(1, attacker_patk - target_pdef // 2)
    dmg = int(base * power * element_bonus * weakness_bonus * break_bonus)
    if crit:
        dmg = int(dmg * 1.5)
    dmg = max(1, int(dmg * random.uniform(0.9, 1.1)))
    return dmg


def calculate_magical_damage(attacker_matk: int, target_mdef: int,
                             power: float = 1.0, crit: bool = False,
                             element_bonus: float = 1.0,
                             weakness_bonus: float = 1.0,
                             break_bonus: float = 1.0) -> int:
```

```
    base = max(1, attacker_matk - target_mdef // 3)
    dmg = int(base * power * element_bonus * weakness_bonus * break_bonus)
    if crit:
        dmg = int(dmg * 1.5)
    dmg = max(1, int(dmg * random.uniform(0.9, 1.1)))
    return dmg


def calculate_heal(healer_mdef: int, power: float = 1.0) -> int:
    return max(1, int(healer_mdef * 2.5 * power * random.uniform(0.9, 1.1)))


def check_hit(attacker_acc: float, target_eva: float) -> bool:
    hit_chance = min(0.99, max(0.01, attacker_acc - target_eva))
    return random.random() < hit_chance


def check_crit(crit_rate: float) -> bool:
    return random.random() < crit_rate


class Battle:
    def __init__(self, party: List[Character], monsters: List[MonsterUnit]):
        self.party = party
        self.monsters = monsters
        self.turn = 0
        self.battle_log: List[str] = []
        self._defending: set = set()   # character names defending this round

    # ── Display helpers ──────────────────────────────────────────
    def display_battle_state(self):
        print("\n" + "="*65)
        print(" BATTLE STATUS")
        print("="*65)
        print("  ENEMIES:")
        for i, mu in enumerate(self.monsters):
            if mu.is_dead:
                print(f"  [{i+1}] {mu.instance_name} - ☠ DEFEATED")
            else:
                broken = " ⚡BREAK!" if mu.is_broken else ""
```

```
            print(f"  [{i+1}] {mu.instance_name} Lv{mu.template.level}{broken}")
            print(f"       HP: {mu.hp_bar()}")
            print(f"       Shield: {mu.shield_bar()}{mu.status_str()}")

    print()
    print("  ALLIES:")
    for i, ch in enumerate(self.party):
        if ch.is_dead:
            print(f"  [{i+1}] {ch.name} [DEAD]")
        elif ch.is_ko:
            print(f"  [{i+1}] {ch.name} [KO - {3 - ch.ko_turns} turns left]")
        else:
            print(f"  [{i+1}] {ch.name} Lv{ch.level} {ch.job.value}{ch.status_str()}")
            print(f"       HP: {ch.hp_bar()} | MP: {ch.mp_bar()}")
    print("="*65)

def display_turn_order(self, order: List):
    units = []
    for u in order:
        if isinstance(u, Character):
            units.append(f"{u.name}({u.spd})")
        elif isinstance(u, MonsterUnit):
            units.append(f"{u.instance_name}({u.spd})")
    print(f"\n  Turn Order: {' → '.join(units)}")

def _log(self, msg: str):
    self.battle_log.append(msg)
    print(msg)

# ── Turn order ──────────────────────────────────────────────────
def get_turn_order(self) -> List:
    units = []
    for ch in self.party:
        if not ch.is_ko and not ch.is_dead:
            units.append(ch)
    for mu in self.monsters:
        if not mu.is_dead:
            units.append(mu)
    # Sort by speed descending; defenders get priority
    units.sort(key=lambda u: (
```

```
        1 if (isinstance(u, Character) and u.name in self._defending) else 0,
        u.spd
    ), reverse=True)
    return units

# ── Main battle loop ──────────────────────────────────────────────
def run(self) -> bool:
    """Returns True if party wins."""
    print("\n" + "★"*65)
    print("  ★  BATTLE START!  ★")
    print("★"*65)
    input("  [Press Enter]")

    while True:
        self.turn += 1
        self._defending.clear()
        print(f"\n{'─'*65}")
        print(f"  ── TURN {self.turn} ──")
        self.display_battle_state()

        order = self.get_turn_order()
        self.display_turn_order(order)

        for unit in order:
            if self._check_battle_end():
                break
            if isinstance(unit, Character):
                if unit.is_ko or unit.is_dead:
                    continue
                self._player_turn(unit)
            elif isinstance(unit, MonsterUnit):
                if unit.is_dead:
                    continue
                self._monster_turn(unit)

        # End of round: status ticks, KO timers, break recovery
        self._end_of_round()

        if self._check_battle_end():
            break
```

```
        return self._is_victory()

    def _check_battle_end(self) -> bool:
        all_monsters_dead = all(mu.is_dead for mu in self.monsters)
        all_party_out = all(ch.is_ko or ch.is_dead for ch in self.party)
        return all_monsters_dead or all_party_out

    def _is_victory(self) -> bool:
        return all(mu.is_dead for mu in self.monsters)

    # ── Player turn ──────────────────────────────────────────────
    def _player_turn(self, ch: Character):
        print(f"\n  ── {ch.name}'s Turn ({ch.job.value}) ──")

        if ch.is_incapacitated():
            incap_se = [se for se in ch.status_effects
                        if se.effect_type in {StatusEffectType.SLEEP, StatusEffectType.STUN,
                                              StatusEffectType.FREEZE,
StatusEffectType.PARALYZE,
                                              StatusEffectType.PETRIFY}]
            if incap_se:
                self._log(f"  {ch.name} is {incap_se[0].effect_type.value}! Can't act.")
                # Wake up chance for sleep
                if incap_se[0].effect_type == StatusEffectType.SLEEP:
                    if random.random() < 0.25:
                        ch.remove_status(StatusEffectType.SLEEP)
                        self._log(f"  {ch.name} woke up!")
            return

        while True:
            print(f"\n  Actions: [1] Attack  [2] Skill  [3] Item  [4] Defend")
            choice = input("  > ").strip()

            if choice == "1":
                target = self._pick_enemy_target()
                if target:
                    self._do_basic_attack(ch, target)
                    break
            elif choice == "2":
```

```
            if not ch.can_use_skills():
                print("  Skills are sealed!")
                continue
            skill_id = self._pick_skill(ch)
            if skill_id is None:
                continue
            skill = SKILL_DB[skill_id]
            if not ch.can_use_magic() and skill.skill_type == SkillType.MAGICAL:
                print("  Cannot use magic (Silenced)!")
                continue
            if ch.current_mp < skill.mp_cost:
                print(f"  Not enough MP! (Need {skill.mp_cost}, have {ch.current_mp})")
                continue
            self._do_skill(ch, skill)
            break
        elif choice == "3":
            if not ch.can_use_items():
                print("  Items are sealed!")
                continue
            used = self._use_item_menu(ch)
            if used:
                break
        elif choice == "4":
            self._do_defend(ch)
            break
        else:
            print("  Invalid choice.")

def _pick_enemy_target(self) -> Optional[MonsterUnit]:
    alive = [mu for mu in self.monsters if not mu.is_dead]
    if not alive:
        return None
    if len(alive) == 1:
        return alive[0]
    print("  Choose target:")
    for i, mu in enumerate(alive):
        broken = " ⚡BREAK" if mu.is_broken else ""
        print(f"    [{i+1}] {mu.instance_name} HP:{mu.current_hp}/{mu.max_hp}{broken}")
    while True:
        try:
```

```
            idx = int(input("  > ")) - 1
            if 0 <= idx < len(alive):
                return alive[idx]
        except ValueError:
            pass
        print("  Invalid choice.")

    def _pick_ally_target(self, include_ko=False) -> Optional[Character]:
        if include_ko:
            valid = [ch for ch in self.party if not ch.is_dead]
        else:
            valid = [ch for ch in self.party if not ch.is_ko and not ch.is_dead]
        if not valid:
            return None
        if len(valid) == 1:
            return valid[0]
        print("  Choose ally:")
        for i, ch in enumerate(valid):
            status = "KO" if ch.is_ko else f"HP:{ch.current_hp}/{ch.max_hp}"
            print(f"    [{i+1}] {ch.name} ({status})")
        while True:
            try:
                idx = int(input("  > ")) - 1
                if 0 <= idx < len(valid):
                    return valid[idx]
            except ValueError:
                pass
            print("  Invalid choice.")

    def _pick_skill(self, ch: Character) -> Optional[int]:
        if not ch.unlocked_skills:
            print("  No skills available!")
            return None
        print("  Choose skill (0=back):")
        for i, sid in enumerate(ch.unlocked_skills):
            if sid in SKILL_DB:
                sk = SKILL_DB[sid]
                print(f"    [{i+1}] {sk.name} "
                      f"[{sk.tier.value}|{sk.skill_type.value}|{sk.element.value}] "
                      f"MP:{sk.mp_cost} PWR:{sk.power} ACC:{int(sk.accuracy*100)}%")
```

```
        while True:
            raw = input("  > ").strip()
            if raw == "0":
                return None
            try:
                idx = int(raw) - 1
                if 0 <= idx < len(ch.unlocked_skills):
                    return ch.unlocked_skills[idx]
            except ValueError:
                pass
            print("  Invalid choice.")

    def _use_item_menu(self, ch: Character) -> bool:
        """Returns True if an item was successfully used."""
        from game_state import GameState
        # Items accessed via global inventory - we'll use a simplified approach
        # The GameState will be passed in during actual game run
        print("  (Item system: handled by game state)")
        return False  # placeholder - overridden in actual game

    def _do_basic_attack(self, ch: Character, target: MonsterUnit):
        # Hit check
        hit = check_hit(ch.acc, target.eva)
        if not hit:
            self._log(f"  {ch.name} attacks {target.instance_name}... MISS!")
            return

        is_crit = check_crit(ch.crit)

        # Weakness check
        is_weak = target.hit_weakness(ch.weapon, ch.element)
        weakness_bonus = 1.3 if is_weak else 1.0
        break_bonus = 1.5 if target.is_broken else 1.0

        dmg = calculate_physical_damage(
            ch.patk, target.pdef, power=1.0,
            crit=is_crit, weakness_bonus=weakness_bonus, break_bonus=break_bonus
        )

        # Defending reduction
```

```
        if ch.name in self._defending:
            dmg = int(dmg * 0.7)

        actual = target.take_damage(dmg)

        crit_str = " CRITICAL!" if is_crit else ""
        weak_str = " □WEAKNESS□" if is_weak else ""
        self._log(f"  {ch.name} attacks {target.instance_name}!{crit_str}{weak_str}")
        self._log(f"    Dealt {actual} damage!")

        # Shield break
        if is_weak:
            broke = target.reduce_shield(1)
            if broke:
                self._log(f"  ⚡ {target.instance_name} is BROKEN! All defenses down for 1
turn!")
            else:
                self._log(f"  Shield: {target.shield_bar()} ({target.shield_points}
remaining)")

        # Counter
        if target.has_status(StatusEffectType.COUNTER) and not target.is_dead:
            cdmg = calculate_physical_damage(target.patk, ch.pdef, power=0.5)
            ch.take_damage(cdmg)
            self._log(f"  {target.instance_name} COUNTER! {ch.name} takes {cdmg} damage!")

    def _do_skill(self, ch: Character, skill: Skill):
        ch.current_mp -= skill.mp_cost

        # Determine targets
        targets_enemies = []
        targets_allies = []
        if skill.target == SkillTarget.SINGLE_ENEMY:
            t = self._pick_enemy_target()
            if t: targets_enemies = [t]
        elif skill.target == SkillTarget.ALL_ENEMIES:
            targets_enemies = [m for m in self.monsters if not m.is_dead]
        elif skill.target == SkillTarget.RANDOM_ENEMY:
            alive = [m for m in self.monsters if not m.is_dead]
            if alive:
```

```
                targets_enemies = random.choices(alive, k=skill.hits)
        elif skill.target == SkillTarget.SINGLE_ALLY:
            t = self._pick_ally_target(include_ko=skill.skill_type == SkillType.HEAL)
            if t: targets_allies = [t]
        elif skill.target == SkillTarget.ALL_ALLIES:
            targets_allies = [c for c in self.party if not c.is_dead]
        elif skill.target == SkillTarget.SELF:
            targets_allies = [ch]

        elem_name = f"[{skill.element.value}]" if skill.element != Element.NONE else ""
        self._log(f"\n  ✦ {ch.name} uses {skill.name}!{elem_name}")

        # Heal skills
        if skill.skill_type == SkillType.HEAL:
            for target in targets_allies:
                if skill.mp_cost == 20 and skill.id == 46:  # Resurrection special case
                    if target.is_ko:
                        target.revive(100)
                        self._log(f"    {target.name} is REVIVED with full HP!")
                    else:
                        self._log(f"    {target.name} is already standing.")
                else:
                    heal_amt = calculate_heal(ch.mdef, skill.power)
                    actual = target.heal(heal_amt)
                    self._log(f"    {target.name} recovers {actual} HP!")
            return

        # Buff skills
        if skill.skill_type == SkillType.BUFF:
            if skill.effect:
                for target in targets_allies:
                    se = StatusEffect(skill.effect.status, skill.effect.duration,
skill.effect.stat_modifier)
                    target.add_status(se)
                    self._log(f"    {target.name} gains {skill.effect.status.value}!")
            return

        # Debuff skills
        if skill.skill_type == SkillType.DEBUFF:
            for target in targets_enemies:
```

```
                if skill.effect and skill.effect.status:
                    chance = skill.effect.chance if skill.effect.chance > 0 else 0.8
                    if random.random() < chance:
                        se = StatusEffect(skill.effect.status, skill.effect.duration)
                        target.add_status(se)
                        self._log(f"    {target.instance_name} is afflicted with
{skill.effect.status.value}!")
                    else:
                        self._log(f"    {target.instance_name} resists!")
            return

        # Damage skills (physical / magical / special)
        num_hits = skill.hits if skill.target != SkillTarget.RANDOM_ENEMY else 1
        raw_targets = targets_enemies

        if skill.target == SkillTarget.RANDOM_ENEMY:
            alive = [m for m in self.monsters if not m.is_dead]
            raw_targets = []
            for _ in range(skill.hits):
                if alive: raw_targets.append(random.choice(alive))

        for target in raw_targets:
            for hit_n in range(num_hits if skill.target != SkillTarget.RANDOM_ENEMY else 1):
                # Hit check
                if not check_hit(ch.acc * skill.accuracy, target.eva):
                    self._log(f"    Hit {hit_n+1}: MISS on {target.instance_name}!")
                    continue

                is_crit = check_crit(ch.crit)
                is_weak = target.hit_weakness(
                    ch.weapon if skill.skill_type == SkillType.PHYSICAL else None,
                    skill.element if skill.element != Element.NONE else ch.element
                )
                weakness_bonus = 1.3 if is_weak else 1.0
                break_bonus = 1.5 if target.is_broken else 1.0
                # Weakness mark doubles weakness
                if target.has_status(StatusEffectType.WEAKNESS_MARK) and is_weak:
                    weakness_bonus *= 1.3

                if skill.skill_type == SkillType.PHYSICAL:
```

```
                    dmg = calculate_physical_damage(
                        ch.patk, target.pdef, skill.power,
                        is_crit, weakness_bonus=weakness_bonus, break_bonus=break_bonus)
                else:
                    # Magic boost
                    matk = ch.matk
                    if ch.has_status(StatusEffectType.MAGIC_BOOST):
                        matk = int(matk * 1.4)
                    # Reflect check
                    if target.has_status(StatusEffectType.REFLECT):
                        self._log(f"    {target.instance_name} REFLECTS the spell!")
                        friendly = [c for c in self.party if not c.is_ko and not c.is_dead]
                        if friendly:
                            rf_target = random.choice(friendly)
                            rdmg = calculate_magical_damage(matk, rf_target.mdef, skill.power)
                            rf_target.take_damage(rdmg)
                            self._log(f"    Reflected! {rf_target.name} takes {rdmg} damage!")
                        continue
                    dmg = calculate_magical_damage(
                        matk, target.mdef, skill.power,
                        is_crit, weakness_bonus=weakness_bonus, break_bonus=break_bonus)

                actual = target.take_damage(dmg)
                crit_str = " CRITICAL!" if is_crit else ""
                weak_str = " □WEAKNESS□" if is_weak else ""
                self._log(f"    {target.instance_name} takes {actual}
damage!{crit_str}{weak_str}")

                # Shield damage for weakness hits
                if is_weak:
                    broke = target.reduce_shield(1)
                    if broke:
                        self._log(f"  ⚡ {target.instance_name} is BROKEN!")
                    else:
                        self._log(f"  Shield: {target.shield_bar()}")

                # Apply effect
                if skill.effect and skill.effect.status and not target.is_dead:
                    chance = skill.effect.chance if skill.effect.chance > 0 else 0.5
                    if random.random() < chance:
```

```
                        se = StatusEffect(skill.effect.status, skill.effect.duration,
                                          skill.effect.stat_modifier)
                        target.add_status(se)
                        self._log(f"    {target.instance_name} afflicted with
{skill.effect.status.value}!")

    def _do_defend(self, ch: Character):
        self._defending.add(ch.name)
        ch.add_status(StatusEffect(StatusEffectType.DEFENDING, 1))
        self._log(f"  {ch.name} takes a defensive stance! (DMG -30% next turn, speed
priority)")

    # ── Monster turn ──────────────────────────────────────────────
    def _monster_turn(self, mu: MonsterUnit):
        if mu.is_incapacitated():
            self._log(f"\n  {mu.instance_name} is incapacitated and cannot act!")
            return

        alive_party = [ch for ch in self.party if not ch.is_ko and not ch.is_dead]
        if not alive_party:
            return

        action = mu.choose_action(alive_party)
        target = random.choice(alive_party)

        if action["action"] == "attack":
            self._monster_basic_attack(mu, target)
        elif action["action"] == "skill":
            skill_id = action["skill_id"]
            if skill_id in SKILL_DB:
                skill = SKILL_DB[skill_id]
                self._monster_skill(mu, skill, alive_party)
            else:
                self._monster_basic_attack(mu, target)
        elif action["action"] == "defend":
            self._log(f"  {mu.instance_name} braces for impact!")

    def _monster_basic_attack(self, mu: MonsterUnit, target: Character):
        hit = check_hit(mu.acc, target.eva)
        if not hit:
```

```
            self._log(f"\n  {mu.instance_name} attacks {target.name}... MISS!")
            return

        is_crit = check_crit(mu.template.base_stats.crit)
        # Defend reduction
        defending = target.has_status(StatusEffectType.DEFENDING)
        dmg = calculate_physical_damage(
            mu.patk, target.pdef, crit=is_crit)
        if defending:
            dmg = int(dmg * 0.7)

        # Confusion: might attack ally
        if mu.has_status(StatusEffectType.CONFUSION):
            alive_enemies = [m for m in self.monsters if not m.is_dead and m != mu]
            if alive_enemies and random.random() < 0.5:
                friendly_target = random.choice(alive_enemies)
                actual = friendly_target.take_damage(dmg)
                self._log(f"\n  {mu.instance_name} is confused and attacks
{friendly_target.instance_name}! ({actual} dmg)")
                return

        crit_str = " CRITICAL!" if is_crit else ""
        self._log(f"\n  {mu.instance_name} attacks {target.name}!{crit_str}")
        target.take_damage(dmg)
        self._log(f"    {target.name} takes {dmg} damage!")

        # Counter
        if target.has_status(StatusEffectType.COUNTER) and not target.is_ko:
            cdmg = calculate_physical_damage(target.patk, mu.pdef, power=0.5)
            mu.take_damage(cdmg)
            self._log(f"  {target.name} COUNTER! {mu.instance_name} takes {cdmg}!")

    def _monster_skill(self, mu: MonsterUnit, skill: Skill, alive_party: List[Character]):
        self._log(f"\n  {mu.instance_name} uses {skill.name}!")

        # Silence check for magic
        if skill.skill_type == SkillType.MAGICAL and mu.has_status(StatusEffectType.SILENCE):
            self._log(f"  {mu.instance_name} is silenced!")
            return
```

```
        target = random.choice(alive_party)

        if skill.skill_type in (SkillType.PHYSICAL, SkillType.MAGICAL, SkillType.SPECIAL):
            for _ in range(skill.hits):
                if target.is_ko: break
                hit = check_hit(mu.acc * skill.accuracy, target.eva)
                if not hit:
                    self._log(f"    MISS on {target.name}!")
                    continue

                is_crit = check_crit(mu.template.base_stats.crit)
                defending = target.has_status(StatusEffectType.DEFENDING)

                if skill.skill_type == SkillType.PHYSICAL:
                    dmg = calculate_physical_damage(mu.patk, target.pdef, skill.power,
is_crit)
                else:
                    # Reflect check
                    if target.has_status(StatusEffectType.REFLECT):
                        self._log(f"    {target.name} REFLECTS the spell!")
                        rdmg = calculate_magical_damage(mu.matk, mu.mdef, skill.power)
                        mu.take_damage(rdmg)
                        self._log(f"    Reflected! {mu.instance_name} takes {rdmg} damage!")
                        continue
                    dmg = calculate_magical_damage(mu.matk, target.mdef, skill.power, is_crit)

                if defending:
                    dmg = int(dmg * 0.7)

                crit_str = " CRITICAL!" if is_crit else ""
                target.take_damage(dmg)
                self._log(f"    {target.name} takes {dmg} damage!{crit_str}")

                if skill.effect and skill.effect.status and not target.is_ko:
                    chance = skill.effect.chance if skill.effect.chance > 0 else 0.4
                    if random.random() < chance:
                        se = StatusEffect(skill.effect.status, skill.effect.duration,
                                          skill.effect.stat_modifier)
                        target.add_status(se)
                        self._log(f"    {target.name} afflicted with
```

```
{skill.effect.status.value}!")

        elif skill.skill_type == SkillType.HEAL:
            heal_amt = calculate_heal(mu.template.base_stats.mdef, skill.power)
            # Heal self or an alive ally monster
            alive_m = [m for m in [mu] if not m.is_dead]
            for m in alive_m:
                old = m.current_hp
                m.current_hp = min(m.max_hp, m.current_hp + heal_amt)
                self._log(f"    {m.instance_name} recovers {m.current_hp - old} HP!")

    # ── End of round processing ──────────────────────────────────
    def _end_of_round(self):
        print(f"\n  ── End of Turn {self.turn} ──")

        # Status effect ticks for party
        for ch in self.party:
            if not ch.is_ko and not ch.is_dead:
                msgs = ch.tick_status_effects()
                msgs += ch.tick_buffs()
                for m in msgs: self._log(m)

        # KO timer management
        for ch in self.party:
            if ch.is_ko and not ch.is_dead:
                ch.ko_turns += 1
                if ch.ko_turns >= 3:
                    ch.is_dead = True
                    self._log(f"  □□{ch.name} has DIED! (Too long KO'd)")

        # Status ticks for monsters
        for mu in self.monsters:
            if not mu.is_dead:
                msgs = mu.tick_status_effects()
                for m in msgs: self._log(m)
                mu.tick_break()

        # Remove Defending status
        for ch in self.party:
            ch.remove_status(StatusEffectType.DEFENDING)
```

```
    if not self._check_battle_end():
        input("\n  [Press Enter to continue]")

def calculate_exp_reward(self) -> int:
    total = sum(mu.template.exp_reward for mu in self.monsters)
    return total

def use_item_in_battle(self, ch: Character, item_id: int,
                       inventory: Dict[int,int]) -> bool:
    """Use an item from inventory in battle. Returns True if used."""
    if item_id not in inventory or inventory[item_id] <= 0:
        print("  You don't have that item!")
        return False

    item = ITEM_DB[item_id]
    inventory[item_id] -= 1
    if inventory[item_id] <= 0:
        del inventory[item_id]

    self._log(f"  {ch.name} uses {item.name}!")

    if item.item_type == ItemType.RECOVERY:
        # Determine target
        if item.target in (SkillTarget.SINGLE_ALLY, SkillTarget.SELF):
            valid = [c for c in self.party if not c.is_dead]
            if item.target == SkillTarget.SELF:
                valid = [ch]
            if not valid: return True
            target = valid[0] if len(valid)==1 else self._pick_ally_target()
            if not target: return True
            targets = [target]
        else:
            targets = [c for c in self.party if not c.is_dead]

        for t in targets:
            if item.effect_value == -100:
                t.current_hp = t.max_hp
                t.current_mp = t.max_mp
                self._log(f"    {t.name} fully restored!")
```

```
            elif item.effect_value < 0:
                pct = abs(item.effect_value)
                amt = int(t.max_hp * pct / 100)
                actual = t.heal(amt)
                self._log(f"    {t.name} recovers {actual} HP!")
            elif item.id in (6,7,8,9,12,20):  # MP items
                mp_gain = min(item.effect_value, t.max_mp - t.current_mp)
                t.current_mp += mp_gain
                self._log(f"    {t.name} recovers {mp_gain} MP!")
            else:
                actual = t.heal(item.effect_value)
                self._log(f"    {t.name} recovers {actual} HP!")

        elif item.item_type == ItemType.REVIVAL:
            valid = [c for c in self.party if c.is_ko and not c.is_dead]
            if not valid:
                self._log("    No KO'd allies to revive!")
                return True
            target = valid[0] if len(valid)==1 else self._pick_ally_target(include_ko=True)
            if not target or not target.is_ko: return True
            if item.target == SkillTarget.ALL_ALLIES:
                for t in valid:
                    t.revive(item.effect_value)
                    self._log(f"    {t.name} revived with {item.effect_value}% HP!")
            else:
                target.revive(item.effect_value)
                self._log(f"    {target.name} revived with {item.effect_value}% HP!")

        elif item.item_type == ItemType.STATUS_CURE:
            valid = [c for c in self.party if not c.is_dead]
            if item.target == SkillTarget.ALL_ALLIES:
                targets = valid
            else:
                t = self._pick_ally_target()
                targets = [t] if t else []
            for t in targets:
                if item.status_cure:
                    for stype in item.status_cure:
                        t.remove_status(stype)
                    self._log(f"    {t.name} cleansed of ailments!")
```

```
elif item.item_type == ItemType.BUFF:
    valid = [c for c in self.party if not c.is_dead and not c.is_ko]
    if item.target == SkillTarget.ALL_ALLIES:
        targets = valid
    elif item.target == SkillTarget.SELF:
        targets = [ch]
    else:
        t = self._pick_ally_target()
        targets = [t] if t else []
    for t in targets:
        if item.stat_buff:
            for stat, mult in item.stat_buff.items():
                t.add_buff(stat, mult, item.buff_duration)
            self._log(f"    {t.name} gains buffs!")

elif item.item_type == ItemType.OFFENSIVE:
    alive_enemies = [m for m in self.monsters if not m.is_dead]
    if item.target in (SkillTarget.SINGLE_ENEMY,):
        t = self._pick_enemy_target()
        if not t: return True
        targets = [t]
    else:
        targets = alive_enemies
    for t in targets:
        is_weak = item.element and t.hit_weakness(None, item.element)
        w_bonus = 1.3 if is_weak else 1.0
        b_bonus = 1.5 if t.is_broken else 1.0
        dmg = int(item.effect_value * w_bonus * b_bonus)
        actual = t.take_damage(dmg)
        w_str = " □WEAKNESS□" if is_weak else ""
        self._log(f"    {t.instance_name} takes {actual} damage!{w_str}")
        if is_weak:
            broke = t.reduce_shield(1)
            if broke:
                self._log(f"  ⚡ {t.instance_name} is BROKEN!")

elif item.item_type in (ItemType.ADVANCED, ItemType.REVIVAL):
    # Handle advanced items
    if item.effect_value == -100 or item.effect_value == 9999:
```

```
            valid = [c for c in self.party if not c.is_dead]
            for t in valid:
                if item.target == SkillTarget.ALL_ALLIES:
                    pass
                elif item.target == SkillTarget.SINGLE_ALLY:
                    t = self._pick_ally_target()
                    valid = [t] if t else []
                    break
            for t in valid:
                if item.effect_value == 9999:
                    actual = t.heal(9999)
                    self._log(f"    {t.name} recovers {actual} HP!")
                else:
                    t.current_hp = t.max_hp
                    t.current_mp = t.max_mp
                    self._log(f"    {t.name} fully restored!")
                if item.stat_buff:
                    for stat, mult in item.stat_buff.items():
                        t.add_buff(stat, mult, item.buff_duration)

        return True
```

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
```

companions.py

```
"""Companion character database: 20x3★, 10x4★, 5x5★."""
from __future__ import annotations
import random
from data_types import *
from character import Character, JOB_BASE_STATS, JOB_GROWTH
from skills_db import SKILL_DB, JOB_SKILL_POOL


def _make_companion(name, job, weapon, element, rarity,
                    stat_mult=1.0, growth_mult=1.0) -> Character:
    base = JOB_BASE_STATS[job].copy()
    g = JOB_GROWTH[job]

    # Apply rarity scaling
    base.hp   = int(base.hp   * stat_mult)
    base.mp   = int(base.mp   * stat_mult)
    base.patk = int(base.patk * stat_mult)
    base.matk = int(base.matk * stat_mult)
    base.pdef = int(base.pdef * stat_mult)
    base.mdef = int(base.mdef * stat_mult)
    base.spd  = int(base.spd  * stat_mult)

    # Scale growth rates
    scaled_g = GrowthRates(
        hp   = min(0.99, g.hp   * growth_mult),
        mp   = min(0.99, g.mp   * growth_mult),
        patk = min(0.99, g.patk * growth_mult),
        matk = min(0.99, g.matk * growth_mult),
        pdef = min(0.99, g.pdef * growth_mult),
        mdef = min(0.99, g.mdef * growth_mult),
        spd  = min(0.99, g.spd  * growth_mult),
        eva  = min(0.99, g.eva  * growth_mult),
        acc  = min(0.99, g.acc  * growth_mult),
        crit = min(0.99, g.crit * growth_mult),
    )
```

```
    pool = JOB_SKILL_POOL[job]
    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[:]

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


# ═══════════════════════════════════════════════════════════════════════
# COMPANION TEMPLATES (factories — called fresh each game)
# ═══════════════════════════════════════════════════════════════════════

# stat_mult: 3★=0.90~0.95, 4★=1.10~1.20, 5★=1.35~1.50
# growth_mult: 3★=0.85~0.90, 4★=1.05~1.15, 5★=1.25~1.40

THREE_STAR_TEMPLATES = [
    # name, job, weapon, element, stat_mult, growth_mult
    ("Rook",    JobClass.WARRIOR,    WeaponType.SWORD,  Element.FIRE,      0.90, 0.88),
    ("Gara",    JobClass.KNIGHT,     WeaponType.SWORD,  Element.EARTH,     0.92, 0.87),
    ("Tifa",    JobClass.MONK,       WeaponType.AXE,    Element.WIND,      0.91, 0.88),
    ("Sera",    JobClass.CLERIC,     WeaponType.STAFF,  Element.LIGHT,     0.93, 0.88),
    ("Rex",     JobClass.RANGER,     WeaponType.BOW,    Element.ICE,       0.90, 0.87),
    ("Bram",    JobClass.BERSERKER,  WeaponType.AXE,    Element.FIRE,      0.89, 0.86),
    ("Nira",    JobClass.WIND_MAGE,  WeaponType.STAFF,  Element.WIND,      0.92, 0.87),
    ("Dag",     JobClass.SPEARMAN,   WeaponType.SPEAR,  Element.LIGHTNING, 0.91, 0.88),
    ("Ella",    JobClass.FIRE_MAGE,  WeaponType.STAFF,  Element.FIRE,      0.90, 0.87),
    ("Wynn",    JobClass.HUNTER,     WeaponType.BOW,    Element.EARTH,     0.93, 0.88),
    ("Bart",    JobClass.EARTH_MAGE, WeaponType.STAFF,  Element.EARTH,     0.91, 0.86),
```

```
    ("Yuna",    JobClass.PRIEST,     WeaponType.STAFF,  Element.LIGHT,     0.92, 0.87),
    ("Kage",    JobClass.ASSASSIN,   WeaponType.DAGGER, Element.DARK,      0.90, 0.86),
    ("Coral",   JobClass.ICE_MAGE,   WeaponType.STAFF,  Element.ICE,       0.91, 0.87),
    ("Miles",   JobClass.WARRIOR,    WeaponType.AXE,    Element.WIND,      0.90, 0.86),
    ("Dora",    JobClass.CLERIC,     WeaponType.STAFF,  Element.FIRE,      0.92, 0.87),
    ("Oryn",    JobClass.STORM_MAGE, WeaponType.STAFF,  Element.LIGHTNING, 0.90, 0.86),
    ("Fenn",    JobClass.PALADIN,    WeaponType.SWORD,  Element.LIGHT,     0.93, 0.88),
    ("Zell",    JobClass.DRAGOON,    WeaponType.SPEAR,  Element.WIND,      0.91, 0.87),
    ("Mira",    JobClass.DARK_MAGE,  WeaponType.STAFF,  Element.DARK,      0.90, 0.86),
]

FOUR_STAR_TEMPLATES = [
    ("Cael",    JobClass.KNIGHT,     WeaponType.SWORD,  Element.LIGHT,     1.12, 1.10),
    ("Lyra",    JobClass.ARCANE_SAGE,WeaponType.STAFF,  Element.LIGHTNING, 1.15, 1.12),
    ("Vance",   JobClass.BERSERKER,  WeaponType.AXE,    Element.DARK,      1.18, 1.14),
    ("Shira",   JobClass.ASSASSIN,   WeaponType.DAGGER, Element.ICE,       1.12, 1.10),
    ("Bael",    JobClass.DRAGOON,    WeaponType.SPEAR,  Element.FIRE,      1.14, 1.11),
    ("Noel",    JobClass.PRIEST,     WeaponType.STAFF,  Element.LIGHT,     1.15, 1.12),
    ("Kira",    JobClass.STORM_MAGE, WeaponType.STAFF,  Element.LIGHTNING, 1.16, 1.13),
    ("Roan",    JobClass.PALADIN,    WeaponType.SWORD,  Element.LIGHT,     1.14, 1.10),
    ("Zara",    JobClass.HUNTER,     WeaponType.BOW,    Element.WIND,      1.12, 1.10),
    ("Drax",    JobClass.MONK,       WeaponType.AXE,    Element.EARTH,     1.18, 1.14),
]

FIVE_STAR_TEMPLATES = [
    ("Seraph",  JobClass.LIGHT_MAGE, WeaponType.STAFF,  Element.LIGHT,     1.45, 1.38),
    ("Abyss",   JobClass.DARK_MAGE,  WeaponType.STAFF,  Element.DARK,      1.42, 1.35),
    ("Titan",   JobClass.BERSERKER,  WeaponType.AXE,    Element.EARTH,     1.50, 1.40),
    ("Oracle",  JobClass.ARCANE_SAGE,WeaponType.STAFF,  Element.LIGHTNING, 1.45, 1.40),
    ("Valkyrie",JobClass.PALADIN,    WeaponType.SWORD,  Element.LIGHT,     1.48, 1.38),
]

COMPANION_GACHA_POOL = {
    3: THREE_STAR_TEMPLATES,
    4: FOUR_STAR_TEMPLATES,
    5: FIVE_STAR_TEMPLATES,
}

GACHA_RATES = {3: 0.65, 4: 0.30, 5: 0.05}
```

```
def summon_companion(target_level: int = 2) -> Character:
    """Roll a random companion via gacha, leveled to target_level."""
    roll = random.random()
    cumulative = 0.0
    rarity = 3
    for r, rate in GACHA_RATES.items():
        cumulative += rate
        if roll < cumulative:
            rarity = r
            break

    templates = COMPANION_GACHA_POOL[rarity]
    tmpl = random.choice(templates)
    name, job, weapon, element, stat_mult, growth_mult = tmpl

    char = _make_companion(name, job, weapon, element, rarity,
                           stat_mult, growth_mult)

    # Level the character up silently
    for _ in range(1, target_level):
        char._level_up()
    char.level = target_level
    char.current_hp = char.max_hp
    char.current_mp = char.max_mp

    return char
```

data_types.py

```
"""Core data types, enums, and constants for the JRPG game."""
from __future__ import annotations
from dataclasses import dataclass, field
from enum import Enum, auto
from typing import Optional, List, Dict, Any


class WeaponType(Enum):
    SWORD = "Sword"
    SPEAR = "Spear"
    AXE = "Axe"
    DAGGER = "Dagger"
    BOW = "Bow"
    STAFF = "Staff"


class Element(Enum):
    FIRE = "Fire"
    ICE = "Ice"
    LIGHTNING = "Lightning"
    WIND = "Wind"
    EARTH = "Earth"
    LIGHT = "Light"
    DARK = "Dark"
    NONE = "None"


class JobClass(Enum):
    WARRIOR = "Warrior"
    KNIGHT = "Knight"
    PALADIN = "Paladin"
    BERSERKER = "Berserker"
    ASSASSIN = "Assassin"
    RANGER = "Ranger"
    HUNTER = "Hunter"
```

```
    SPEARMAN = "Spearman"
    DRAGOON = "Dragoon"
    MONK = "Monk"
    CLERIC = "Cleric"
    PRIEST = "Priest"
    FIRE_MAGE = "Fire Mage"
    ICE_MAGE = "Ice Mage"
    STORM_MAGE = "Storm Mage"
    WIND_MAGE = "Wind Mage"
    EARTH_MAGE = "Earth Mage"
    DARK_MAGE = "Dark Mage"
    LIGHT_MAGE = "Light Mage"
    ARCANE_SAGE = "Arcane Sage"


class SkillType(Enum):
    PHYSICAL = "Physical"
    MAGICAL = "Magical"
    HEAL = "Heal"
    BUFF = "Buff"
    DEBUFF = "Debuff"
    SPECIAL = "Special"


class SkillTier(Enum):
    BASIC = "Basic"
    INTERMEDIATE = "Intermediate"
    ULTIMATE = "Ultimate"


class SkillTarget(Enum):
    SINGLE_ENEMY = "SingleEnemy"
    ALL_ENEMIES = "AllEnemies"
    SINGLE_ALLY = "SingleAlly"
    ALL_ALLIES = "AllAllies"
    SELF = "Self"
    RANDOM_ENEMY = "RandomEnemy"


class StatusEffectType(Enum):
```

```
    # Damage over time
    POISON = "Poison"
    BURN = "Burn"
    BLEED = "Bleed"
    VENOM = "Venom"
    CURSE = "Curse"
    # Action restriction
    SLEEP = "Sleep"
    STUN = "Stun"
    FREEZE = "Freeze"
    PARALYZE = "Paralyze"
    PETRIFY = "Petrify"
    # Debuffs
    ATTACK_DOWN = "Attack Down"
    DEFENSE_DOWN = "Defense Down"
    MAGIC_DOWN = "Magic Down"
    SPEED_DOWN = "Speed Down"
    ACCURACY_DOWN = "Accuracy Down"
    # Buff blockers
    SILENCE = "Silence"
    SKILL_SEAL = "Skill Seal"
    ITEM_SEAL = "Item Seal"
    HEAL_BLOCK = "Heal Block"
    MANA_BURN = "Mana Burn"
    # Persistent effects
    REGEN = "Regen"
    MANA_REGEN = "Mana Regen"
    SHIELD = "Shield"
    REFLECT = "Reflect"
    COUNTER = "Counter"
    # Special
    CONFUSION = "Confusion"
    CHARM = "Charm"
    FEAR = "Fear"
    BLIND = "Blind"
    WEAKNESS_MARK = "Weakness Mark"
    # Enhancements
    BERSERK = "Berserk"
    HASTE = "Haste"
    FOCUS = "Focus"
```

```
    GUARD_UP = "Guard Up"
    MAGIC_BOOST = "Magic Boost"
    # Advanced
    DOOM = "Doom"
    TIME_STOP = "Time Stop"
    CURSE_MARK = "Curse Mark"
    BLOOD_LINK = "Blood Link"
    SOUL_DRAIN = "Soul Drain"
    # Defend
    DEFENDING = "Defending"


class AIType(Enum):
    AGGRESSIVE = "Aggressive"
    DEFENSIVE = "Defensive"
    BALANCED = "Balanced"
    SUPPORT = "Support"
    RANDOM = "Random"
    BERSERKER = "Berserker"
    TACTICAL = "Tactical"


class ItemType(Enum):
    RECOVERY = "Recovery"
    STATUS_CURE = "Status Cure"
    BUFF = "Buff"
    OFFENSIVE = "Offensive"
    ADVANCED = "Advanced"
    REVIVAL = "Revival"


@dataclass
class SkillEffect:
    status: Optional[StatusEffectType] = None
    duration: int = 0
    chance: float = 0.0
    stat_modifier: float = 1.0
    heal_percent: float = 0.0
    shield_amount: int = 0
```

```
@dataclass
class Skill:
    id: int
    name: str
    skill_type: SkillType
    power: float
    mp_cost: int
    accuracy: float
    element: Element
    hits: int
    target: SkillTarget
    tier: SkillTier
    description: str
    effect: Optional[SkillEffect] = None
    jobs: List[JobClass] = field(default_factory=list)


@dataclass
class StatusEffect:
    effect_type: StatusEffectType
    duration: int
    power: float = 1.0
    source_name: str = ""

    def tick(self) -> bool:
        """Returns True if effect is still active after tick."""
        if self.duration > 0:
            self.duration -= 1
        return self.duration > 0 or self.duration == -1  # -1 = permanent until removed


@dataclass
class Item:
    id: int
    name: str
    item_type: ItemType
    description: str
    effect_value: int
    target: SkillTarget
```

```
    element: Optional[Element] = None
    status_cure: Optional[List[StatusEffectType]] = None
    stat_buff: Optional[Dict[str, float]] = None
    buff_duration: int = 0


@dataclass
class Stats:
    hp: int
    mp: int
    patk: int
    matk: int
    pdef: int
    mdef: int
    spd: int
    eva: float
    acc: float
    crit: float

    def copy(self) -> 'Stats':
        return Stats(
            hp=self.hp, mp=self.mp, patk=self.patk, matk=self.matk,
            pdef=self.pdef, mdef=self.mdef, spd=self.spd,
            eva=self.eva, acc=self.acc, crit=self.crit
        )


@dataclass
class GrowthRates:
    hp: float
    mp: float
    patk: float
    matk: float
    pdef: float
    mdef: float
    spd: float
    eva: float
    acc: float
    crit: float
```

```
RARITY_COLORS = {
    3: "★★★",
    4: "★★★★",
    5: "★★★★★"
}
```

game_state.py

```
"""Game state: manages party, inventory, battle progression."""
from __future__ import annotations
import random
from dataclasses import dataclass, field
from typing import List, Dict, Optional, Tuple
from data_types import *
from character import Character, create_player_character
from monster_unit import MonsterUnit
from monsters_db import get_encounter, MONSTER_DB
from items_db import ITEM_DB
from skills_db import SKILL_DB
from battle import Battle
from companions_db import summon_companion


# Starting inventory
DEFAULT_INVENTORY = {
    1: 5,   # Potion x5
    6: 3,   # Ether x3
    31: 2,  # Phoenix Feather x2
    41: 2,  # Antidote x2
    86: 2,  # Fire Bomb x2
}

# Battle milestones for companion rewards
COMPANION_MILESTONES = {
    1:  2,    # After battle 1: lv2 companion
    5:  5,    # After battle 5: lv5 companion
    10: 7,    # After battle 10: lv7-8 companion
    20: 15,   # After battle 20: lv15 companion
    50: 40,   # After battle 50: lv40 companion
    75: 47,   # After battle 75: lv45-50 companion
    100:60,   # After battle 100: lv60 companion
}
```

```
class GameState:
    def __init__(self):
        self.player: Optional[Character] = None
        self.party: List[Character] = []
        self.inventory: Dict[int, int] = dict(DEFAULT_INVENTORY)
        self.battle_number: int = 0
        self.game_over: bool = False
        self.deceased_companions: List[Character] = []

        # Records for game over screen
        self.all_party_members: List[Character] = []

    # ── Inventory ──────────────────────────────────────────────────
    def add_item(self, item_id: int, count: int = 1):
        self.inventory[item_id] = self.inventory.get(item_id, 0) + count

    def display_inventory(self):
        if not self.inventory:
            print("  (Empty)")
            return
        print("  INVENTORY:")
        for iid, count in sorted(self.inventory.items()):
            if iid in ITEM_DB and count > 0:
                item = ITEM_DB[iid]
                print(f"    {item.name} x{count} - {item.description}")

    def use_item_outside_battle(self, item_id: int, target: Character) -> bool:
        if item_id not in self.inventory or self.inventory[item_id] <= 0:
            print("  You don't have that item!")
            return False
        item = ITEM_DB[item_id]
        self.inventory[item_id] -= 1
        if self.inventory[item_id] <= 0:
            del self.inventory[item_id]

        if item.item_type == ItemType.RECOVERY:
            if item.effect_value == -100:
                target.current_hp = target.max_hp
                target.current_mp = target.max_mp
```

```
                print(f"  {target.name} fully restored!")
            elif item.effect_value < 0:
                pct = abs(item.effect_value)
                amt = int(target.max_hp * pct / 100)
                actual = target.heal(amt)
                print(f"  {target.name} recovers {actual} HP!")
            elif item.id in (6,7,8,9,12,20):
                mp = min(item.effect_value, target.max_mp - target.current_mp)
                target.current_mp += mp
                print(f"  {target.name} recovers {mp} MP!")
            else:
                actual = target.heal(item.effect_value)
                print(f"  {target.name} recovers {actual} HP!")
        elif item.item_type == ItemType.REVIVAL:
            if target.is_ko and not target.is_dead:
                target.revive(item.effect_value)
                print(f"  {target.name} revived with {item.effect_value}% HP!")
            else:
                print(f"  {target.name} isn't KO'd.")
                self.inventory[item_id] = self.inventory.get(item_id, 0) + 1  # refund
                return False
        elif item.item_type == ItemType.STATUS_CURE:
            if item.status_cure:
                for stype in item.status_cure:
                    target.remove_status(stype)
                print(f"  {target.name} cleansed!")
        return True

    # ── Party management ──────────────────────────────────────────
    def display_party(self):
        print("\n  PARTY:")
        for i, ch in enumerate(self.party):
            rarity_str = f" {'★'*ch.rarity}" if ch.rarity > 0 else " [PLAYER]"
            status = ch.short_status()
            print(f"  [{i+1}] {ch.name} Lv{ch.level} {ch.job.value}{rarity_str} - {status}")
            print(f"       HP:{ch.current_hp}/{ch.max_hp} MP:{ch.current_mp}/{ch.max_mp}")
            print(f"       PATK:{ch.patk} MATK:{ch.matk} PDEF:{ch.pdef} MDEF:{ch.mdef}
SPD:{ch.spd}")

    def display_character_detail(self, ch: Character):
```

```
        print(f"\n  ── {ch.name} ──────────────────────────────")
        rarity_str = f"{'★'*ch.rarity}" if ch.rarity > 0 else "PLAYER"
        print(f"  Job: {ch.job.value}  Rarity: {rarity_str}")
        print(f"  Weapon: {ch.weapon.value}  Element: {ch.element.value}")
        print(f"  Level: {ch.level}  EXP: {ch.exp}/{ch.exp_to_next}")
        print(f"  HP: {ch.current_hp}/{ch.max_hp}  MP: {ch.current_mp}/{ch.max_mp}")
        print(f"  PATK:{ch.patk}  MATK:{ch.matk}  PDEF:{ch.pdef}  MDEF:{ch.mdef}")
        print(f"  SPD:{ch.spd}  EVA:{ch.base_stats.eva:.0%}  ACC:{ch.base_stats.acc:.0%}
CRIT:{ch.base_stats.crit:.0%}")
        print(f"  Skills:")
        for sid in ch.skill_pool:
            if sid in SKILL_DB:
                sk = SKILL_DB[sid]
                unlocked = "✓" if sid in ch.unlocked_skills else "✗"
                unlock_lv = ""
                if sk.tier == SkillTier.INTERMEDIATE:
                    unlock_lv = " (Lv5/10)"
                elif sk.tier == SkillTier.ULTIMATE:
                    unlock_lv = " (Lv20)"
                print(f"    [{unlocked}] {sk.name} [{sk.tier.value}]
MP:{sk.mp_cost}{unlock_lv}")

    def offer_companion(self, companion: Character):
        """Offer a new companion to the player."""
        print("\n" + "★"*65)
        rarity_str = "★" * companion.rarity
        print(f"  A new companion has appeared! [{rarity_str}]")
        print(f"  {companion.name} Lv{companion.level} | {companion.job.value}")
        print(f"  HP:{companion.max_hp} PATK:{companion.base_stats.patk}
MATK:{companion.base_stats.matk}")
        print(f"  SPD:{companion.base_stats.spd} Weapon:{companion.weapon.value}
Element:{companion.element.value}")
        alive_party = [ch for ch in self.party if not ch.is_dead]

        if len(alive_party) < 4:
            print(f"\n  Adding {companion.name} to the party!")
            self.party.append(companion)
            self.all_party_members.append(companion)
            input("  [Press Enter]")
        else:
```

```
        print(f"\n  Your party is full (4/4). Dismiss a member to make room?")
        self.display_party()
        print(f"  [1-{len(self.party)}] Dismiss member  [0] Decline companion")
        while True:
            try:
                choice = int(input("  > "))
                if choice == 0:
                    print(f"  Declined {companion.name}.")
                    input("  [Press Enter]")
                    return
                idx = choice - 1
                if 0 <= idx < len(self.party):
                    dismiss_target = self.party[idx]
                    if dismiss_target == self.player:
                        print("  Cannot dismiss player character!")
                        continue
                    self.party.remove(dismiss_target)
                    print(f"  {dismiss_target.name} has left the party.")
                    self.party.append(companion)
                    self.all_party_members.append(companion)
                    print(f"  {companion.name} joined the party!")
                    input("  [Press Enter]")
                    return
            except ValueError:
                pass
            print("  Invalid choice.")


# ── Battle ──────────────────────────────────────────────────────────
def run_battle(self) -> bool:
    """Run a battle. Returns True if player survived."""
    self.battle_number += 1

    # Get encounter
    templates = get_encounter(self.battle_number)
    # Name duplicates
    name_count: Dict[str, int] = {}
    monster_units = []
    for t in templates:
        name_count[t.name] = name_count.get(t.name, 0) + 1
```

```
        used: Dict[str, int] = {}
        for t in templates:
            used[t.name] = used.get(t.name, 0) + 1
            label = t.name if name_count[t.name] == 1 else f"{t.name} {chr(64 +
used[t.name])}"
            mu = MonsterUnit(template=t, instance_name=label)
            monster_units.append(mu)

        alive_party = [ch for ch in self.party if not ch.is_dead]
        battle = Battle(alive_party, monster_units)

        # Inject item use capability into battle
        battle._item_use_callback = self._battle_item_callback
        battle._original_use_item_menu = battle._use_item_menu
        battle._use_item_menu = lambda ch: self._battle_item_ui(ch, battle)

        victory = battle.run()

        if victory:
            exp = battle.calculate_exp_reward()
            print(f"\n  ✓ VICTORY! Earned {exp} EXP.")
            # Award loot
            self._award_loot()

            alive_after = [ch for ch in alive_party if not ch.is_dead]
            for ch in alive_after:
                msgs = ch.gain_exp(exp)
                for m in msgs:
                    print(m)
            input("  [Press Enter]")

            # Handle companion death
            for ch in alive_party:
                if ch.is_dead and ch != self.player:
                    print(f"\n  □□{ch.name} has permanently died and left the party.")
                    self.party.remove(ch)
                    self.deceased_companions.append(ch)

            # Check milestone companions
```

```
            self._check_companion_milestone()

        else:
            # Check if player character survived
            player_dead = self.player.is_dead or self.player.is_ko
            if player_dead:
                self.game_over = True
                print(f"\n  ☠ {self.player.name} has fallen... GAME OVER")
                return False
            else:
                print(f"\n  DEFEAT! Some party members have fallen.")
                for ch in alive_party:
                    if ch.is_dead and ch != self.player:
                        print(f"  □□{ch.name} has permanently died.")
                        self.party.remove(ch)
                        self.deceased_companions.append(ch)
                input("  [Press Enter]")

        # Post-battle: reset KO state for surviving members
        for ch in self.party:
            if not ch.is_dead:
                if ch.is_ko and ch.current_hp > 0:
                    ch.is_ko = False
                    ch.ko_turns = 0
                # Partial HP restore (30% if KO'd but saved)
                if ch.current_hp <= 0:
                    ch.current_hp = max(1, ch.max_hp // 5)
                    ch.is_ko = False
                    ch.ko_turns = 0

        return True

    def _battle_item_callback(self, ch, item_id: int, battle: Battle) -> bool:
        return battle.use_item_in_battle(ch, item_id, self.inventory)

    def _battle_item_ui(self, ch: Character, battle: Battle) -> bool:
        if not self.inventory:
            print("  Inventory is empty!")
            return False
        print("  INVENTORY (0=back):")
```

```
        items = [(iid, cnt) for iid, cnt in sorted(self.inventory.items()) if cnt > 0]
        for i, (iid, cnt) in enumerate(items):
            if iid in ITEM_DB:
                item = ITEM_DB[iid]
                print(f"    [{i+1}] {item.name} x{cnt} - {item.description}")
        while True:
            raw = input("  > ").strip()
            if raw == "0": return False
            try:
                idx = int(raw) - 1
                if 0 <= idx < len(items):
                    item_id = items[idx][0]
                    return battle.use_item_in_battle(ch, item_id, self.inventory)
            except ValueError:
                pass
            print("  Invalid choice.")

    def _award_loot(self):
        """Random item drops after victory."""
        # Simple loot: random item from early pool
        loot_pool = [1, 2, 6, 7, 31, 41, 42, 66, 67, 86, 87, 88]
        advanced_pool = [3, 5, 8, 32, 56, 68]
        rare_pool = [4, 10, 33, 57, 84, 85]

        # Base drop
        if random.random() < 0.7:
            item_id = random.choice(loot_pool)
            self.add_item(item_id)
            print(f"  Item found: {ITEM_DB[item_id].name}!")

        # Extra drop for later battles
        if self.battle_number > 20 and random.random() < 0.4:
            item_id = random.choice(advanced_pool)
            self.add_item(item_id)
            print(f"  Item found: {ITEM_DB[item_id].name}!")

        if self.battle_number > 50 and random.random() < 0.2:
            item_id = random.choice(rare_pool)
            self.add_item(item_id)
            print(f"  Rare item found: {ITEM_DB[item_id].name}!")
```

```
def _check_companion_milestone(self):
    if self.battle_number in COMPANION_MILESTONES:
        target_lv = COMPANION_MILESTONES[self.battle_number]
        if isinstance(target_lv, tuple):
            target_lv = random.randint(target_lv[0], target_lv[1])
        companion = summon_companion(target_lv)
        self.offer_companion(companion)

# ── Between battles menu ──────────────────────────────────────
def between_battles_menu(self):
    while True:
        print("\n" + "─"*65)
        print(f"  ── Camp ── (Battle {self.battle_number} completed)")
        print("  [1] View Party  [2] Character Details  [3] Use Item")
        print("  [4] View Inventory  [5] Next Battle  [6] Quit")
        choice = input("  > ").strip()

        if choice == "1":
            self.display_party()
        elif choice == "2":
            self.display_party()
            try:
                idx = int(input("  Choose character (0=back): ")) - 1
                if 0 <= idx < len(self.party):
                    self.display_character_detail(self.party[idx])
            except ValueError:
                pass
        elif choice == "3":
            self._outside_battle_item_menu()
        elif choice == "4":
            self.display_inventory()
        elif choice == "5":
            return True
        elif choice == "6":
            return False
        else:
            print("  Invalid choice.")

def _outside_battle_item_menu(self):
```

```
        self.display_inventory()
        items = [(iid, cnt) for iid, cnt in sorted(self.inventory.items()) if cnt > 0]
        if not items:
            return
        print("  Use item on (0=back):")
        raw = input("  Item # > ").strip()
        if raw == "0": return
        try:
            idx = int(raw) - 1
            if 0 <= idx < len(items):
                item_id = items[idx][0]
                self.display_party()
                tidx = int(input("  On character # > ")) - 1
                if 0 <= tidx < len(self.party):
                    self.use_item_outside_battle(item_id, self.party[tidx])
        except ValueError:
            pass

    # ── Game over screen ──────────────────────────────────────────
    def show_game_over_screen(self):
        print("\n" + "☠"*65)
        print("  GAME OVER")
        print("☠"*65)
        print(f"\n  {self.player.name} has fallen after {self.battle_number} battles.")
        print(f"\n  ── BATTLE RECORD ──")
        print(f"  Total Battles: {self.battle_number}")
        print(f"  Battles Won:   {self.battle_number - 1}")  # lost the last one

        print(f"\n  ── PARTY HISTORY ──")
        all_chars = list({id(c): c for c in self.all_party_members +
self.deceased_companions}.values())
        if self.player not in all_chars:
            all_chars.insert(0, self.player)

        for ch in all_chars:
            status = "□□DECEASED" if ch.is_dead else "ALIVE"
            rarity = f"{'★'*ch.rarity}" if ch.rarity else "PLAYER"
            print(f"\n  {ch.name} Lv{ch.level} {ch.job.value} [{rarity}] - {status}")
            print(f"    HP:{ch.max_hp} MP:{ch.max_mp} PATK:{ch.base_stats.patk} "
                  f"MATK:{ch.base_stats.matk} PDEF:{ch.base_stats.pdef} "
```

```
              f"MDEF:{ch.base_stats.mdef} SPD:{ch.base_stats.spd}")
        print(f"    Weapon:{ch.weapon.value} Element:{ch.element.value}")
        print(f"    Skills: ", end="")
        skill_names = []
        for sid in ch.skill_pool:
            if sid in SKILL_DB:
                unlocked = "✓" if sid in ch.unlocked_skills else "✗"
                skill_names.append(f"{SKILL_DB[sid].name}[{unlocked}]")
        print(", ".join(skill_names))
    print("\n" + "☠"*65)
    input("  [Press Enter to exit]")
```

items_db.py

```
"""Items database with 120 items."""
from data_types import *

def build_item_database() -> Dict[int, Item]:
    items = {}

    def add(id, name, itype, desc, val, target=SkillTarget.SINGLE_ALLY,
            elem=None, cures=None, buff=None, bdur=0):
        items[id] = Item(id, name, itype, desc, val, target, elem, cures, buff, bdur)

    SE = SkillTarget.SINGLE_ENEMY; SA = SkillTarget.SINGLE_ALLY
    AA = SkillTarget.ALL_ALLIES; SF = SkillTarget.SELF
    AE = SkillTarget.ALL_ENEMIES

    # ── RECOVERY (1-30) ──────────────────────────────────────────
    add(1,  "Potion",        ItemType.RECOVERY, "Restore 150 HP.",          150, SA)
    add(2,  "Hi-Potion",     ItemType.RECOVERY, "Restore 500 HP.",          500, SA)
    add(3,  "Mega Potion",   ItemType.RECOVERY, "Restore 1500 HP.",        1500, SA)
    add(4,  "X-Potion",      ItemType.RECOVERY, "Restore 3000 HP.",        3000, SA)
    add(5,  "Elixir",        ItemType.RECOVERY, "Restore 50% HP and MP.",   -50, SA)   # -50 =
50%
    add(6,  "Ether",         ItemType.RECOVERY, "Restore 50 MP.",            50, SA)
    add(7,  "Hi-Ether",      ItemType.RECOVERY, "Restore 150 MP.",          150, SA)
    add(8,  "Mega Ether",    ItemType.RECOVERY, "Restore 300 MP.",          300, SA)
    add(9,  "Max Ether",     ItemType.RECOVERY, "Fully restore MP.",        999, SA)
    add(10, "Mega Elixir",   ItemType.RECOVERY, "Restore full HP and MP.", -100, SA)   # -100
= full
    add(11, "Ultra Potion",  ItemType.RECOVERY, "Restore 5000 HP.",        5000, SA)
    add(12, "God Ether",     ItemType.RECOVERY, "Fully restore all allies' MP.", 999, AA)
    add(13, "Party Potion",  ItemType.RECOVERY, "Restore 300 HP to all allies.", 300, AA)
    add(14, "Party Hi-Potion",ItemType.RECOVERY,"Restore 800 HP to all allies.", 800, AA)
    add(15, "Life Water",    ItemType.RECOVERY, "Restore 200 HP.",          200, SA)
    add(16, "Mana Water",    ItemType.RECOVERY, "Restore 80 MP.",            80, SA)
    add(17, "Tonic",         ItemType.RECOVERY, "Restore 75 HP.",            75, SA)
    add(18, "Herb",          ItemType.RECOVERY, "Restore 50 HP.",            50, SA)
```

```
    add(19, "Potion II",     ItemType.RECOVERY, "Restore 250 HP.",          250, SA)
    add(20, "Ether II",      ItemType.RECOVERY, "Restore 100 MP.",          100, SA)
    add(21, "Spring Water",  ItemType.RECOVERY, "Restore 120 HP and 30 MP.", -21, SA) #
special combined
    add(22, "Phoenix Tears", ItemType.RECOVERY, "Restore 200 HP.",          200, SA)
    add(23, "Soul Potion",   ItemType.RECOVERY, "Restore 25% HP.",          -25, SA)
    add(24, "Omega Potion",  ItemType.RECOVERY, "Restore 8000 HP.",        8000, SA)
    add(25, "Blessed Water", ItemType.RECOVERY, "Restore 400 HP (Light element).", 400, SA,
Element.LIGHT)
    add(26, "Devil's Blood", ItemType.RECOVERY, "Restore 400 HP (Dark).",   400, SA,
Element.DARK)
    add(27, "Fire Extract",  ItemType.RECOVERY, "Restore 200 HP, grant fire boost.", 200, SA,
Element.FIRE)
    add(28, "Ice Extract",   ItemType.RECOVERY, "Restore 200 HP, grant ice boost.", 200, SA,
Element.ICE)
    add(29, "Thunder Extract",ItemType.RECOVERY,"Restore 200 HP, grant lightning.", 200, SA,
Element.LIGHTNING)
    add(30, "Wind Extract",  ItemType.RECOVERY, "Restore 200 HP, grant wind boost.", 200, SA,
Element.WIND)

    # ── REVIVAL (31-40) ──────────────────────────────────────────
    add(31, "Phoenix Feather",ItemType.REVIVAL, "Revive ally with 25% HP.",  25, SA)
    add(32, "Revival Stone", ItemType.REVIVAL,  "Revive ally with 50% HP.",  50, SA)
    add(33, "Life Gem",      ItemType.REVIVAL,  "Revive ally with 75% HP.",  75, SA)
    add(34, "Soul Crystal",  ItemType.REVIVAL,  "Revive ally with full HP.", 100, SA)
    add(35, "Phoenix Down",  ItemType.REVIVAL,  "Revive ally with 10% HP.",  10, SA)
    add(36, "Angel Wing",    ItemType.REVIVAL,  "Revive all fallen allies (25% HP).", 25, AA)
    add(37, "Goddess Tear",  ItemType.REVIVAL,  "Revive all allies (50% HP).", 50, AA)
    add(38, "Miracle Dust",  ItemType.REVIVAL,  "Revive ally with 30% HP.",  30, SA)
    add(39, "Star Fragment", ItemType.REVIVAL,  "Revive ally with 60% HP.",  60, SA)
    add(40, "Last Hope",     ItemType.REVIVAL,  "Revive all allies (10% HP).", 10, AA)

    # ── STATUS CURE (41-65) ──────────────────────────────────────
    add(41, "Antidote",      ItemType.STATUS_CURE, "Cure Poison.",       0, SA,
        cures=[StatusEffectType.POISON, StatusEffectType.VENOM])
    add(42, "Burn Cure",     ItemType.STATUS_CURE, "Cure Burn.",         0, SA,
        cures=[StatusEffectType.BURN])
    add(43, "Paralyze Cure", ItemType.STATUS_CURE, "Cure Paralyze.",     0, SA,
        cures=[StatusEffectType.PARALYZE])
    add(44, "Sleep Cure",    ItemType.STATUS_CURE, "Cure Sleep.",        0, SA,
```

```
    cures=[StatusEffectType.SLEEP])
add(45, "Blind Cure",    ItemType.STATUS_CURE, "Cure Blind.",        0, SA,
    cures=[StatusEffectType.BLIND])
add(46, "Freeze Cure",   ItemType.STATUS_CURE, "Cure Freeze.",       0, SA,
    cures=[StatusEffectType.FREEZE])
add(47, "Stun Cure",     ItemType.STATUS_CURE, "Cure Stun.",         0, SA,
    cures=[StatusEffectType.STUN])
add(48, "Bleed Cure",    ItemType.STATUS_CURE, "Cure Bleed.",        0, SA,
    cures=[StatusEffectType.BLEED])
add(49, "Fear Cure",     ItemType.STATUS_CURE, "Cure Fear.",         0, SA,
    cures=[StatusEffectType.FEAR])
add(50, "Silence Cure",  ItemType.STATUS_CURE, "Cure Silence.",      0, SA,
    cures=[StatusEffectType.SILENCE])
add(51, "Confusion Cure",ItemType.STATUS_CURE, "Cure Confusion.",    0, SA,
    cures=[StatusEffectType.CONFUSION])
add(52, "Charm Cure",    ItemType.STATUS_CURE, "Cure Charm.",        0, SA,
    cures=[StatusEffectType.CHARM])
add(53, "Petrify Cure",  ItemType.STATUS_CURE, "Cure Petrify.",      0, SA,
    cures=[StatusEffectType.PETRIFY])
add(54, "Curse Cure",    ItemType.STATUS_CURE, "Cure Curse.",        0, SA,
    cures=[StatusEffectType.CURSE, StatusEffectType.CURSE_MARK])
add(55, "Doom Stopper",  ItemType.STATUS_CURE, "Remove Doom.",       0, SA,
    cures=[StatusEffectType.DOOM])
add(56, "Panacea",       ItemType.STATUS_CURE, "Cure all status ailments.", 0, SA,
    cures=list(StatusEffectType))
add(57, "Party Panacea", ItemType.STATUS_CURE, "Cure all ailments for all.", 0, AA,
    cures=list(StatusEffectType))
add(58, "Holy Water",    ItemType.STATUS_CURE, "Cure Dark ailments.", 0, SA,
    cures=[StatusEffectType.CURSE, StatusEffectType.CURSE_MARK, StatusEffectType.DOOM,
           StatusEffectType.SOUL_DRAIN, StatusEffectType.BLOOD_LINK])
add(59, "Remedy",        ItemType.STATUS_CURE, "Cure Poison/Burn/Bleed/Venom.", 0, SA,
    cures=[StatusEffectType.POISON, StatusEffectType.BURN,
           StatusEffectType.BLEED, StatusEffectType.VENOM])
add(60, "Seal Breaker",  ItemType.STATUS_CURE, "Cure Seal effects.",  0, SA,
    cures=[StatusEffectType.SKILL_SEAL, StatusEffectType.ITEM_SEAL,
           StatusEffectType.SILENCE, StatusEffectType.HEAL_BLOCK])
add(61, "Speed Up",      ItemType.STATUS_CURE, "Cure Speed Down.",   0, SA,
    cures=[StatusEffectType.SPEED_DOWN])
add(62, "Clarity Potion",ItemType.STATUS_CURE, "Cure Sleep/Confusion/Charm.", 0, SA,
    cures=[StatusEffectType.SLEEP, StatusEffectType.CONFUSION, StatusEffectType.CHARM])
```

```
add(63, "Iron Tonic",    ItemType.STATUS_CURE, "Cure Defense Down.", 0, SA,
    cures=[StatusEffectType.DEFENSE_DOWN, StatusEffectType.ATTACK_DOWN])
add(64, "Mana Restore",  ItemType.STATUS_CURE, "Cure Mana Burn.",    0, SA,
    cures=[StatusEffectType.MANA_BURN])
add(65, "Full Remedy",   ItemType.STATUS_CURE, "Cure all stat debuffs.", 0, SA,
    cures=[StatusEffectType.ATTACK_DOWN, StatusEffectType.DEFENSE_DOWN,
           StatusEffectType.MAGIC_DOWN, StatusEffectType.SPEED_DOWN,
           StatusEffectType.ACCURACY_DOWN, StatusEffectType.WEAKNESS_MARK])

# ── BUFF ITEMS (66-85) ──────────────────────────────────────────
add(66, "Power Tonic",    ItemType.BUFF, "Raise PATK by 50% for 3 turns.",    0, SA,
    buff={"patk": 1.5}, bdur=3)
add(67, "Defense Tonic",  ItemType.BUFF, "Raise PDEF by 50% for 3 turns.",    0, SA,
    buff={"pdef": 1.5}, bdur=3)
add(68, "Magic Tonic",    ItemType.BUFF, "Raise MATK by 50% for 3 turns.",    0, SA,
    buff={"matk": 1.5}, bdur=3)
add(69, "Speed Tonic",    ItemType.BUFF, "Raise SPD by 50% for 3 turns.",     0, SA,
    buff={"spd": 1.5}, bdur=3)
add(70, "Guard Tonic",    ItemType.BUFF, "Raise MDEF by 50% for 3 turns.",    0, SA,
    buff={"mdef": 1.5}, bdur=3)
add(71, "Evasion Tonic",  ItemType.BUFF, "Raise EVA by 30% for 3 turns.",     0, SA,
    buff={"eva": 0.3}, bdur=3)
add(72, "Crit Tonic",     ItemType.BUFF, "Raise CRIT by 30% for 3 turns.",    0, SA,
    buff={"crit": 0.3}, bdur=3)
add(73, "Power Stone",    ItemType.BUFF, "Raise PATK by 80% for 2 turns.",    0, SA,
    buff={"patk": 1.8}, bdur=2)
add(74, "Magic Stone",    ItemType.BUFF, "Raise MATK by 80% for 2 turns.",    0, SA,
    buff={"matk": 1.8}, bdur=2)
add(75, "Shield Stone",   ItemType.BUFF, "Raise PDEF/MDEF by 80% for 2 turns.", 0, SA,
    buff={"pdef": 1.8, "mdef": 1.8}, bdur=2)
add(76, "Haste Potion",   ItemType.BUFF, "Gain Haste for 3 turns.",           0, SA,
    buff={"spd": 2.0}, bdur=3)
add(77, "Berserk Potion", ItemType.BUFF, "Raise PATK by 100%, lower PDEF.",   0, SA,
    buff={"patk": 2.0, "pdef": 0.5}, bdur=3)
add(78, "Focus Stone",    ItemType.BUFF, "Raise ACC and CRIT for 3 turns.",    0, SA,
    buff={"acc": 1.5, "crit": 1.5}, bdur=3)
add(79, "Party Power",    ItemType.BUFF, "Raise all allies' PATK by 30%.",    0, AA,
    buff={"patk": 1.3}, bdur=3)
add(80, "Party Shield",   ItemType.BUFF, "Raise all allies' DEF by 30%.",     0, AA,
    buff={"pdef": 1.3, "mdef": 1.3}, bdur=3)
```

```
    add(81, "War Banner",     ItemType.BUFF, "Raise all allies' stats by 20%.",   0, AA,
        buff={"patk":1.2,"matk":1.2,"pdef":1.2,"mdef":1.2,"spd":1.2}, bdur=3)
    add(82, "Regen Potion",   ItemType.BUFF, "Grant HP Regen for 5 turns.",       50, SA)
    add(83, "Mana Potion",    ItemType.BUFF, "Grant Mana Regen for 5 turns.",     20, SA)
    add(84, "Luck Up",        ItemType.BUFF, "Raise EVA, ACC, CRIT for 3 turns.", 0, SA,
        buff={"eva":0.2,"acc":1.3,"crit":1.3}, bdur=3)
    add(85, "Legend Stone",   ItemType.BUFF, "Massively boost all stats for 2 turns.", 0, SA,
        buff={"patk":2.0,"matk":2.0,"pdef":1.5,"mdef":1.5,"spd":1.5}, bdur=2)

    # ── OFFENSIVE ITEMS (86-105) ──────────────────────────────────
    add(86, "Fire Bomb",      ItemType.OFFENSIVE, "Fire damage to one enemy.",    300, SE,
Element.FIRE)
    add(87, "Ice Bomb",       ItemType.OFFENSIVE, "Ice damage to one enemy.",     300, SE,
Element.ICE)
    add(88, "Thunder Bomb",   ItemType.OFFENSIVE, "Lightning damage to one enemy.", 300, SE,
Element.LIGHTNING)
    add(89, "Wind Bomb",      ItemType.OFFENSIVE, "Wind damage to one enemy.",    300, SE,
Element.WIND)
    add(90, "Earth Bomb",     ItemType.OFFENSIVE, "Earth damage to one enemy.",   300, SE,
Element.EARTH)
    add(91, "Dark Bomb",      ItemType.OFFENSIVE, "Dark damage to one enemy.",    300, SE,
Element.DARK)
    add(92, "Holy Bomb",      ItemType.OFFENSIVE, "Light damage to one enemy.",   300, SE,
Element.LIGHT)
    add(93, "Mega Fire Bomb", ItemType.OFFENSIVE, "Heavy fire damage to all.",    500, AE,
Element.FIRE)
    add(94, "Mega Ice Bomb",  ItemType.OFFENSIVE, "Heavy ice damage to all.",     500, AE,
Element.ICE)
    add(95, "Mega Thunder Bomb",ItemType.OFFENSIVE,"Heavy lightning to all.",     500, AE,
Element.LIGHTNING)
    add(96, "Mega Wind Bomb", ItemType.OFFENSIVE, "Heavy wind damage to all.",    500, AE,
Element.WIND)
    add(97, "Mega Earth Bomb",ItemType.OFFENSIVE, "Heavy earth damage to all.",   500, AE,
Element.EARTH)
    add(98, "Chaos Bomb",     ItemType.OFFENSIVE, "Massive damage to all enemies.", 800, AE)
    add(99, "Poison Vial",    ItemType.OFFENSIVE, "Inflict Poison on an enemy.",  0, SE,
        cures=[])  # reuse cures field as apply-poison signal
    add(100,"Sleep Powder",   ItemType.OFFENSIVE, "Put an enemy to Sleep.",       0, SE)
    add(101,"Stone Grenade",  ItemType.OFFENSIVE, "Inflict Petrify (30%).",       0, SE)
    add(102,"Silence Orb",    ItemType.OFFENSIVE, "Inflict Silence on enemy.",    0, SE)
```

```
    add(103,"Paralysis Dart", ItemType.OFFENSIVE, "Inflict Paralyze on enemy.",   0, SE)
    add(104,"Doom Clock",     ItemType.OFFENSIVE, "Inflict Doom (5 turns).",      0, SE)
    add(105,"Ultimate Bomb",  ItemType.OFFENSIVE, "Colossal damage to one enemy.", 2000, SE)

    # ── ADVANCED / SPECIAL (106-120) ──────────────────────────────
    add(106,"Ultimate Potion",ItemType.ADVANCED, "Restore 9999 HP.",           9999, SA)
    add(107,"Ultima Elixir",  ItemType.ADVANCED, "Fully restore HP/MP all allies.", -100, AA)
    add(108,"Omega Elixir",   ItemType.ADVANCED, "Restore all HP/MP + cure all.", -100, SA)
    add(109,"God's Breath",   ItemType.ADVANCED, "Raise all stats 100% for 3 turns.", 0, SA,
        buff={"patk":2.0,"matk":2.0,"pdef":2.0,"mdef":2.0,"spd":2.0}, bdur=3)
    add(110,"World Crystal",  ItemType.ADVANCED, "Full restore + buff all allies.", -100, AA)
    add(111,"Aether",         ItemType.ADVANCED, "Restore 9999 MP to one ally.", 9999, SA)
    add(112,"Time Crystal",   ItemType.ADVANCED, "Grant Time Stop effect.",      0, SF)
    add(113,"Philosopher's Stone",ItemType.ADVANCED,"Double all stats for 5 turns.", 0, SF,
        buff={"patk":2.0,"matk":2.0,"pdef":2.0,"mdef":2.0,"spd":2.0,"eva":0.5}, bdur=5)
    add(114,"War God's Pill", ItemType.ADVANCED, "PATK x3, DEF/2 for 2 turns.", 0, SF,
        buff={"patk":3.0,"pdef":0.5,"mdef":0.5}, bdur=2)
    add(115,"Sage's Tincture",ItemType.ADVANCED, "MATK x3 for 2 turns.",        0, SF,
        buff={"matk":3.0}, bdur=2)
    add(116,"Hero's Elixir",  ItemType.ADVANCED, "Restore HP/MP + raise all stats.", 3000, SA,
        buff={"patk":1.5,"matk":1.5,"pdef":1.5,"mdef":1.5,"spd":1.5}, bdur=3)
    add(117,"Crystal Vial",   ItemType.ADVANCED, "Cure all + 5000 HP.",         5000, SA)
    add(118,"Dragon's Blood", ItemType.ADVANCED, "Restore 50% HP + grant Regen.", -50, SA,
        buff={"pdef":1.5,"mdef":1.5}, bdur=3)
    add(119,"Chaos Shard",    ItemType.ADVANCED, "9999 damage to all enemies.", 9999, AE)
    add(120,"Omnipotent Stone",ItemType.ADVANCED,"Restore full HP/MP + max all buffs.", -100,
SA,
        buff={"patk":3.0,"matk":3.0,"pdef":3.0,"mdef":3.0,"spd":3.0}, bdur=5)

    return items

ITEM_DB: Dict[int, Item] = build_item_database()
```

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
```

skills_db.py

```
"""Skills database with 100 skills across all job classes."""
from data_types import *


def build_skill_database() -> Dict[int, Skill]:
 skills = {}


 def s(id, name, stype, power, mp, acc, elem, hits, target, tier, desc, jobs, effect=None):
 skills[id] = Skill(id, name, stype, power, mp, acc, elem, hits, target, tier, desc, effect,
jobs)


 W = JobClass.WARRIOR; KN = JobClass.KNIGHT; PA = JobClass.PALADIN
 BE = JobClass.BERSERKER; AS = JobClass.ASSASSIN; RA = JobClass.RANGER
 HU = JobClass.HUNTER; SP = JobClass.SPEARMAN; DR = JobClass.DRAGOON
 MO = JobClass.MONK; CL = JobClass.CLERIC; PR = JobClass.PRIEST
 FM = JobClass.FIRE_MAGE; IM = JobClass.ICE_MAGE; ST = JobClass.STORM_MAGE
 WM = JobClass.WIND_MAGE; EM = JobClass.EARTH_MAGE; DM = JobClass.DARK_MAGE
 LM = JobClass.LIGHT_MAGE; SA = JobClass.ARCANE_SAGE


 SE = SkillTarget.SINGLE_ENEMY; AE = SkillTarget.ALL_ENEMIES
 SA_ = SkillTarget.SINGLE_ALLY; AA = SkillTarget.ALL_ALLIES
 SF = SkillTarget.SELF; RE = SkillTarget.RANDOM_ENEMY


 # ── WARRIOR (IDs 1-14) ──────────────────────────────
 s(1,"Slash",SkillType.PHYSICAL,1.2,0,0.9,Element.NONE,1,SE,SkillTier.BASIC,"A swift sword
slash.",[W,KN,PA])
 s(2,"Power Strike",SkillType.PHYSICAL,1.5,4,0.85,Element.NONE,1,SE,SkillTier.BASIC,"A
powerful focused strike.",[W,BE])
 s(3,"Double
Slash",SkillType.PHYSICAL,0.9,6,0.88,Element.NONE,2,SE,SkillTier.INTERMEDIATE,"Two rapid
slashes.",[W,AS])
```

```
 s(4,"Whirlwind
Slash",SkillType.PHYSICAL,0.85,8,0.85,Element.WIND,1,AE,SkillTier.INTERMEDIATE,"Spin attack
hitting all foes.",[W])
 s(5,"Blade Storm",SkillType.PHYSICAL,2.0,20,0.8,Element.NONE,3,SE,SkillTier.ULTIMATE,"Unleash
a storm of blades.",[W])


 s(6,"Shield Bash",SkillType.PHYSICAL,1.0,3,0.9,Element.NONE,1,SE,SkillTier.BASIC,"Bash with
shield, chance to stun.",
 [KN],SkillEffect(StatusEffectType.STUN,1,0.3))
 s(7,"Guard Stance",SkillType.BUFF,0,5,1.0,Element.NONE,1,SF,SkillTier.BASIC,"Raise defense
for 2 turns.",
 [KN,PA],SkillEffect(StatusEffectType.GUARD_UP,2))
 s(8,"Holy
Sword",SkillType.PHYSICAL,1.6,10,0.85,Element.LIGHT,1,SE,SkillTier.INTERMEDIATE,"Light-imbued
sword strike.",[PA,KN])
 s(9,"Provoke",SkillType.DEBUFF,0,4,0.95,Element.NONE,1,AE,SkillTier.INTERMEDIATE,"Draw all
enemy attacks.",[KN])
 s(10,"Divine
Blade",SkillType.PHYSICAL,2.5,25,0.8,Element.LIGHT,1,SE,SkillTier.ULTIMATE,"Sacred blade of
divine power.",[PA])


 s(11,"Reckless Strike",SkillType.PHYSICAL,2.0,5,0.8,Element.NONE,1,SE,SkillTier.BASIC,"High
power, ignores own defense.",[BE])
 s(12,"Frenzy",SkillType.PHYSICAL,0.8,8,0.75,Element.NONE,3,RE,SkillTier.INTERMEDIATE,"Attack
randomly 3 times.",
 [BE],SkillEffect(StatusEffectType.BERSERK,2))

s(13,"Bloodthirst",SkillType.PHYSICAL,1.8,10,0.85,Element.NONE,1,SE,SkillTier.INTERMEDIATE,"Ab
sorb HP on hit.",[BE])

s(14,"Apocalypse",SkillType.PHYSICAL,3.5,30,0.75,Element.DARK,1,AE,SkillTier.ULTIMATE,"Catastr
ophic strike of destruction.",[BE])


 # ── ASSASSIN (15-25) ──────────────────────────────
 s(15,"Backstab",SkillType.PHYSICAL,1.8,5,0.85,Element.NONE,1,SE,SkillTier.BASIC,"High crit
strike from shadows.",[AS])
 s(16,"Poison Blade",SkillType.PHYSICAL,1.0,4,0.9,Element.NONE,1,SE,SkillTier.BASIC,"Inflict
```

```
poison.",
 [AS,HU],SkillEffect(StatusEffectType.POISON,3,0.7))
 s(17,"Shadow
Step",SkillType.PHYSICAL,1.4,7,0.88,Element.DARK,1,SE,SkillTier.INTERMEDIATE,"Teleport strike,
ignore EVA.",[AS])
 s(18,"Venom
Strike",SkillType.PHYSICAL,1.1,8,0.88,Element.NONE,1,SE,SkillTier.INTERMEDIATE,"Inflict
venom.",
 [AS],SkillEffect(StatusEffectType.VENOM,4,0.65))
 s(19,"Death Mark",SkillType.SPECIAL,0,12,0.85,Element.DARK,1,SE,SkillTier.ULTIMATE,"Mark for
death - doom in 5 turns.",
 [AS],SkillEffect(StatusEffectType.DOOM,5,1.0))


 # ── RANGER / HUNTER (26-38) ──────────────────────────
 s(20,"Arrow Shot",SkillType.PHYSICAL,1.1,0,0.92,Element.NONE,1,SE,SkillTier.BASIC,"Basic
arrow attack.",[RA,HU])
 s(21,"Triple Arrow",SkillType.PHYSICAL,0.7,7,0.85,Element.NONE,3,SE,SkillTier.BASIC,"Fire
three arrows.",[RA])
 s(22,"Rain of
Arrows",SkillType.PHYSICAL,0.65,10,0.82,Element.NONE,1,AE,SkillTier.INTERMEDIATE,"Barrage of
arrows on all foes.",[RA])
 s(23,"Sniper
Shot",SkillType.PHYSICAL,2.2,12,0.9,Element.NONE,1,SE,SkillTier.INTERMEDIATE,"Precise high
damage shot, high crit.",[RA,HU])
 s(24,"Piercing
Arrow",SkillType.PHYSICAL,1.5,8,0.9,Element.NONE,1,AE,SkillTier.INTERMEDIATE,"Arrow pierces
all enemies.",[RA])
 s(25,"Meteor
Arrow",SkillType.PHYSICAL,3.0,28,0.82,Element.FIRE,1,SE,SkillTier.ULTIMATE,"Flaming arrow from
heavens.",[RA])
 s(26,"Trap Set",SkillType.DEBUFF,0,5,0.9,Element.NONE,1,SE,SkillTier.BASIC,"Set a trap -
slows enemy.",
 [HU],SkillEffect(StatusEffectType.SPEED_DOWN,3,0.8))
 s(27,"Beast Lore",SkillType.DEBUFF,0,6,0.95,Element.NONE,1,SE,SkillTier.INTERMEDIATE,"Reveal
and mark weakness.",
 [HU],SkillEffect(StatusEffectType.WEAKNESS_MARK,3))
 s(28,"Dragon
Slayer",SkillType.PHYSICAL,3.5,30,0.8,Element.LIGHT,1,SE,SkillTier.ULTIMATE,"Ultimate anti-
dragon technique.",[HU,DR])
```

```
 # ── SPEARMAN / DRAGOON (29-40) ──────────────────────
 s(29,"Lance Thrust",SkillType.PHYSICAL,1.3,3,0.9,Element.NONE,1,SE,SkillTier.BASIC,"Thrust
with spear.",[SP,DR])
 s(30,"Sweep",SkillType.PHYSICAL,0.9,5,0.87,Element.NONE,1,AE,SkillTier.BASIC,"Sweep all
enemies with spear.",[SP])
 s(31,"Spear
Dance",SkillType.PHYSICAL,0.85,9,0.85,Element.NONE,3,RE,SkillTier.INTERMEDIATE,"Dance of spear
strikes.",[SP])
 s(32,"Dragon
Dive",SkillType.PHYSICAL,2.0,12,0.85,Element.WIND,1,SE,SkillTier.INTERMEDIATE,"Leap and dive
with spear.",[DR])
 s(33,"Jump",SkillType.PHYSICAL,2.3,10,0.88,Element.NONE,1,SE,SkillTier.INTERMEDIATE,"Leap
attack from above.",[DR,SP])
 s(34,"Chaos
Nova",SkillType.PHYSICAL,3.2,30,0.78,Element.NONE,1,AE,SkillTier.ULTIMATE,"Explosive nova of
force.",[DR])


 # ── MONK (35-44) ────────────────────────────────────
 s(35,"Punch",SkillType.PHYSICAL,1.1,0,0.93,Element.NONE,1,SE,SkillTier.BASIC,"Basic unarmed
strike.",[MO])
 s(36,"Combo Strike",SkillType.PHYSICAL,0.75,5,0.9,Element.NONE,3,SE,SkillTier.BASIC,"Rapid
combo punches.",[MO])

s(37,"Shockwave",SkillType.PHYSICAL,1.3,8,0.87,Element.EARTH,1,AE,SkillTier.INTERMEDIATE,"Grou
nd shockwave hits all foes.",[MO])
 s(38,"Inner Focus",SkillType.BUFF,0,6,1.0,Element.NONE,1,SF,SkillTier.INTERMEDIATE,"Focus
power for next attack.",
 [MO],SkillEffect(StatusEffectType.FOCUS,2))
 s(39,"Fist of
Heaven",SkillType.PHYSICAL,3.0,25,0.82,Element.LIGHT,1,SE,SkillTier.ULTIMATE,"Divine fist of
heaven.",[MO])


 # ── CLERIC / PRIEST (40-54) ─────────────────────────
 s(40,"Heal",SkillType.HEAL,1.0,6,1.0,Element.LIGHT,1,SA_,SkillTier.BASIC,"Restore ally
HP.",[CL,PR,PA])
 s(41,"Smite",SkillType.MAGICAL,1.2,7,0.88,Element.LIGHT,1,SE,SkillTier.BASIC,"Light-based
```

```
strike.",[CL])
 s(42,"Cure Status",SkillType.HEAL,0,5,1.0,Element.NONE,1,SA_,SkillTier.BASIC,"Remove a status ailment.",
 [CL,PR])
 s(43,"Mass Heal",SkillType.HEAL,0.85,14,1.0,Element.LIGHT,1,AA,SkillTier.INTERMEDIATE,"Heal all allies.",[PR,CL])
 s(44,"Holy Light",SkillType.MAGICAL,1.8,15,0.85,Element.LIGHT,1,SE,SkillTier.INTERMEDIATE,"Brilliant holy beam.",[CL,PR])
 s(45,"Regen Aura",SkillType.BUFF,0,10,1.0,Element.LIGHT,1,AA,SkillTier.INTERMEDIATE,"Grant regen to all allies.",
 [PR],SkillEffect(StatusEffectType.REGEN,3))
 s(46,"Resurrection",SkillType.HEAL,0,20,0.95,Element.LIGHT,1,SA_,SkillTier.ULTIMATE,"Revive fallen ally with full HP.",[PR])
 s(47,"Divine Judgment",SkillType.MAGICAL,3.0,28,0.82,Element.LIGHT,1,AE,SkillTier.ULTIMATE,"Judgment of the divine.",[CL])
 s(48,"Blessing",SkillType.BUFF,0,8,1.0,Element.LIGHT,1,SA_,SkillTier.INTERMEDIATE,"Raise ally's all stats.",
 [PA,PR])
 s(49,"Silence Ward",SkillType.DEBUFF,0,7,0.85,Element.LIGHT,1,SE,SkillTier.BASIC,"Silence an enemy.",
 [CL],SkillEffect(StatusEffectType.SILENCE,2,0.75))


 # ── FIRE MAGE (50-59) ──────────────────────────────
 s(50,"Fire Bolt",SkillType.MAGICAL,1.3,7,0.9,Element.FIRE,1,SE,SkillTier.BASIC,"Basic fire projectile.",[FM,SA])
 s(51,"Fire Ball",SkillType.MAGICAL,1.1,10,0.87,Element.FIRE,1,AE,SkillTier.BASIC,"Explosive fireball hits all.",[FM])
 s(52,"Inferno",SkillType.MAGICAL,2.0,16,0.85,Element.FIRE,1,SE,SkillTier.INTERMEDIATE,"Column of infernal flames.",
 [FM],SkillEffect(StatusEffectType.BURN,2,0.5))
 s(53,"Flame Burst",SkillType.MAGICAL,1.5,12,0.87,Element.FIRE,1,AE,SkillTier.INTERMEDIATE,"Burst of flames over all foes.",[FM])
 s(54,"Hellfire",SkillType.MAGICAL,3.5,35,0.78,Element.FIRE,1,AE,SkillTier.ULTIMATE,"Hellfire scorches everything.",[FM])
```

```
 # ── ICE MAGE (55-64) ──────────────────────────────
 s(55,"Ice Shard",SkillType.MAGICAL,1.2,6,0.9,Element.ICE,1,SE,SkillTier.BASIC,"Sharp ice
projectile.",[IM,SA])
 s(56,"Blizzard",SkillType.MAGICAL,1.0,10,0.87,Element.ICE,1,AE,SkillTier.BASIC,"Ice storm
hits all enemies.",
 [IM],SkillEffect(StatusEffectType.SPEED_DOWN,2,0.4))
 s(57,"Frost
Lance",SkillType.MAGICAL,1.7,12,0.88,Element.ICE,1,SE,SkillTier.INTERMEDIATE,"Piercing lance
of frost.",
 [IM],SkillEffect(StatusEffectType.FREEZE,1,0.3))
 s(58,"Ice
Field",SkillType.MAGICAL,1.4,15,0.85,Element.ICE,1,AE,SkillTier.INTERMEDIATE,"Freeze the
entire battlefield.",[IM])
 s(59,"Absolute
Zero",SkillType.MAGICAL,4.0,40,0.72,Element.ICE,1,SE,SkillTier.ULTIMATE,"Reduce temperature to
absolute zero.",[IM])


 # ── STORM MAGE (60-69) ────────────────────────────
 s(60,"Thunder",SkillType.MAGICAL,1.2,7,0.88,Element.LIGHTNING,1,SE,SkillTier.BASIC,"Basic
lightning strike.",[ST,SA])
 s(61,"Lightning
Bolt",SkillType.MAGICAL,1.4,9,0.87,Element.LIGHTNING,1,SE,SkillTier.BASIC,"Focused lightning
bolt.",
 [ST])
 s(62,"Chain
Lightning",SkillType.MAGICAL,1.0,14,0.85,Element.LIGHTNING,1,AE,SkillTier.INTERMEDIATE,"Lightn
ing chains between all foes.",[ST])

s(63,"Thunderstorm",SkillType.MAGICAL,1.6,18,0.83,Element.LIGHTNING,1,AE,SkillTier.INTERMEDIAT
E,"Raging storm of lightning.",[ST])
 s(64,"Judgment
Thunder",SkillType.MAGICAL,3.8,38,0.75,Element.LIGHTNING,1,SE,SkillTier.ULTIMATE,"Ultimate
thunderbolt of judgment.",[ST])


 # ── WIND MAGE (65-72) ─────────────────────────────
 s(65,"Wind Slash",SkillType.MAGICAL,1.1,5,0.92,Element.WIND,1,SE,SkillTier.BASIC,"Blade of
wind.",[WM])
 s(66,"Gale",SkillType.MAGICAL,0.9,8,0.88,Element.WIND,1,AE,SkillTier.BASIC,"Gale force wind
```

```
hits all.",
 [WM],SkillEffect(StatusEffectType.SPEED_DOWN,2,0.35))

s(67,"Tornado",SkillType.MAGICAL,1.7,14,0.85,Element.WIND,1,SE,SkillTier.INTERMEDIATE,"Miniatu
re tornado engulfs enemy.",[WM])

s(68,"Hurricane",SkillType.MAGICAL,1.4,18,0.83,Element.WIND,1,AE,SkillTier.INTERMEDIATE,"Hurri
cane force winds.",[WM])

s(69,"Tempest",SkillType.MAGICAL,3.2,32,0.78,Element.WIND,1,AE,SkillTier.ULTIMATE,"Catastrophi
c tempest of wind.",[WM])


 # ── EARTH MAGE (70-77) ──────────────────────────────
 s(70,"Stone",SkillType.MAGICAL,1.2,5,0.9,Element.EARTH,1,SE,SkillTier.BASIC,"Hurl a
stone.",[EM])
 s(71,"Earth Spike",SkillType.MAGICAL,1.4,8,0.88,Element.EARTH,1,SE,SkillTier.BASIC,"Spike
from the ground.",[EM])

s(72,"Earthquake",SkillType.MAGICAL,1.5,15,0.85,Element.EARTH,1,AE,SkillTier.INTERMEDIATE,"Mas
sive earthquake.",[EM])
 s(73,"Rock
Slide",SkillType.MAGICAL,1.3,12,0.87,Element.EARTH,1,AE,SkillTier.INTERMEDIATE,"Avalanche of
rocks.",
 [EM],SkillEffect(StatusEffectType.SPEED_DOWN,2,0.4))
 s(74,"Meteor",SkillType.MAGICAL,4.2,42,0.70,Element.EARTH,1,AE,SkillTier.ULTIMATE,"Call down
a meteor.",[EM,SA])


 # ── DARK MAGE (75-82) ───────────────────────────────
 s(75,"Dark Bolt",SkillType.MAGICAL,1.2,6,0.9,Element.DARK,1,SE,SkillTier.BASIC,"Dark energy
bolt.",[DM])
 s(76,"Shadow Bind",SkillType.DEBUFF,0,8,0.82,Element.DARK,1,SE,SkillTier.BASIC,"Bind enemy in
shadows.",
 [DM],SkillEffect(StatusEffectType.PARALYZE,2,0.65))
 s(77,"Dark
Pulse",SkillType.MAGICAL,1.5,12,0.87,Element.DARK,1,AE,SkillTier.INTERMEDIATE,"Pulse of dark
energy.",
 [DM],SkillEffect(StatusEffectType.CURSE,2,0.4))
 s(78,"Void
```

```
Drain",SkillType.MAGICAL,1.3,10,0.87,Element.DARK,1,SE,SkillTier.INTERMEDIATE,"Drain MP from target.",
 [DM],SkillEffect(StatusEffectType.MANA_BURN,1))
 s(79,"Abyss",SkillType.MAGICAL,3.8,38,0.72,Element.DARK,1,SE,SkillTier.ULTIMATE,"Plunge enemy into the abyss.",[DM])


 # ── LIGHT MAGE (80-87) ──────────────────────────────
 s(80,"Photon",SkillType.MAGICAL,1.2,6,0.92,Element.LIGHT,1,SE,SkillTier.BASIC,"Photon burst.",[LM])
 s(81,"Shine",SkillType.MAGICAL,1.0,8,0.9,Element.LIGHT,1,AE,SkillTier.BASIC,"Flash of light blinds enemies.",
 [LM],SkillEffect(StatusEffectType.BLIND,2,0.5))

s(82,"Radiance",SkillType.MAGICAL,1.7,14,0.87,Element.LIGHT,1,SE,SkillTier.INTERMEDIATE,"Blinding radiance.",[LM])
 s(83,"Holy
Barrier",SkillType.BUFF,0,12,1.0,Element.LIGHT,1,SA_,SkillTier.INTERMEDIATE,"Shield ally with holy power.",
 [LM,PA],SkillEffect(StatusEffectType.SHIELD,3))
 s(84,"Judgement
Ray",SkillType.MAGICAL,3.5,35,0.78,Element.LIGHT,1,SE,SkillTier.ULTIMATE,"Divine ray of judgment.",[LM])


 # ── ARCANE SAGE (85-100) ────────────────────────────
 s(85,"Arcane Bolt",SkillType.MAGICAL,1.3,7,0.9,Element.NONE,1,SE,SkillTier.BASIC,"Pure arcane projectile.",[SA])
 s(86,"Mana Shield",SkillType.BUFF,0,8,1.0,Element.NONE,1,SF,SkillTier.BASIC,"Convert MP to a shield.",
 [SA],SkillEffect(StatusEffectType.SHIELD,2))
 s(87,"Arcane
Storm",SkillType.MAGICAL,1.3,16,0.87,Element.NONE,1,AE,SkillTier.INTERMEDIATE,"Storm of arcane energy.",[SA])
 s(88,"Time
Dilation",SkillType.SPECIAL,0,18,0.85,Element.NONE,1,SA_,SkillTier.INTERMEDIATE,"Accelerate ally's time.",
 [SA],SkillEffect(StatusEffectType.HASTE,3))
 s(89,"Meteor",SkillType.MAGICAL,4.0,45,0.70,Element.NONE,1,AE,SkillTier.ULTIMATE,"Ultimate arcane meteor barrage.",[SA])
```

```
 # ── EXTRA SKILLS (90-100) ──────────────────────────
 s(90,"Blood
Drain",SkillType.MAGICAL,1.4,10,0.87,Element.DARK,1,SE,SkillTier.INTERMEDIATE,"Drain blood
from enemy.",
 [DM,AS],SkillEffect(StatusEffectType.BLEED,3,0.6))
 s(91,"Phantom
Edge",SkillType.PHYSICAL,1.6,9,0.88,Element.DARK,1,SE,SkillTier.INTERMEDIATE,"Phantom blade
strike.",[AS,DM])
 s(92,"Nature's
Wrath",SkillType.MAGICAL,1.8,16,0.85,Element.EARTH,1,AE,SkillTier.INTERMEDIATE,"Earth's
fury.",[EM,MO])
 s(93,"Solar
Flare",SkillType.MAGICAL,2.0,18,0.83,Element.FIRE,1,AE,SkillTier.INTERMEDIATE,"Blinding solar
burst.",
 [FM,LM],SkillEffect(StatusEffectType.BLIND,2,0.6))
 s(94,"Frost
Nova",SkillType.MAGICAL,1.6,14,0.87,Element.ICE,1,AE,SkillTier.INTERMEDIATE,"Explosive ice
nova.",
 [IM],SkillEffect(StatusEffectType.FREEZE,1,0.4))
 s(95,"War Cry",SkillType.BUFF,0,8,1.0,Element.NONE,1,AA,SkillTier.INTERMEDIATE,"Boost all
allies' attack.",
 [W,BE],SkillEffect(StatusEffectType.BERSERK,2))
 s(96,"Stealth",SkillType.BUFF,0,6,1.0,Element.NONE,1,SF,SkillTier.BASIC,"Enter stealth
mode.",
 [AS,RA])
 s(97,"Eagle Eye",SkillType.BUFF,0,5,1.0,Element.NONE,1,SF,SkillTier.BASIC,"Boost accuracy and
crit.",
 [RA,HU],SkillEffect(StatusEffectType.FOCUS,2))
 s(98,"Shield Wall",SkillType.BUFF,0,10,1.0,Element.NONE,1,AA,SkillTier.INTERMEDIATE,"Raise
defense of all allies.",
 [KN,PA],SkillEffect(StatusEffectType.GUARD_UP,3))
 s(99,"Soul
Shatter",SkillType.MAGICAL,2.5,22,0.82,Element.DARK,1,SE,SkillTier.INTERMEDIATE,"Shatter the
enemy's soul.",
 [DM,SA],SkillEffect(StatusEffectType.SOUL_DRAIN,3))
 s(100,"Omega Strike",SkillType.PHYSICAL,5.0,50,0.75,Element.NONE,1,SE,SkillTier.ULTIMATE,"The
ultimate physical strike.",[W,BE,MO])
```

```
 return skills


SKILL_DB: Dict[int, Skill] = build_skill_database()


# Job -> list of skill IDs
JOB_SKILL_POOL: Dict[JobClass, List[int]] = {
 JobClass.WARRIOR: [1,2,3,4,5,11,95,98,100,6,7,34,33,29],
 JobClass.KNIGHT: [6,7,8,9,10,1,40,48,98,2,33,30,44,83],
 JobClass.PALADIN: [8,10,40,48,83,7,6,44,46,47,80,81,98,45,42],
 JobClass.BERSERKER: [11,12,13,14,2,5,95,1,3,100,34,4,92,37,79],
 JobClass.ASSASSIN: [15,16,17,18,19,3,96,90,91,75,76,78,99,77,25],
 JobClass.RANGER: [20,21,22,23,24,25,96,97,26,27,1,65,70,60,50],
 JobClass.HUNTER: [20,21,23,26,27,28,97,16,31,22,24,33,34,35,90],
 JobClass.SPEARMAN: [29,30,31,32,33,34,1,2,4,35,36,92,72,37,95],
 JobClass.DRAGOON: [29,32,33,34,28,31,8,65,4,3,5,92,25,23,100],
 JobClass.MONK: [35,36,37,38,39,2,1,92,72,70,95,100,33,34,30],
 JobClass.CLERIC: [40,41,42,43,44,45,46,47,48,49,80,81,8,82,84],
 JobClass.PRIEST: [40,43,45,46,48,42,49,44,83,84,41,47,80,82,81],
 JobClass.FIRE_MAGE: [50,51,52,53,54,93,85,86,87,75,77,60,65,70,76],
 JobClass.ICE_MAGE: [55,56,57,58,59,94,85,86,87,65,70,75,77,60,52],
 JobClass.STORM_MAGE: [60,61,62,63,64,85,86,87,50,55,65,93,88,75,77],
 JobClass.WIND_MAGE: [65,66,67,68,69,85,86,87,60,55,50,88,92,93,63],
 JobClass.EARTH_MAGE: [70,71,72,73,74,92,85,86,87,37,36,55,50,93,65],
 JobClass.DARK_MAGE: [75,76,77,78,79,90,91,99,85,86,87,19,88,18,15],
 JobClass.LIGHT_MAGE: [80,81,82,83,84,40,44,47,48,85,86,87,93,45,42],
 JobClass.ARCANE_SAGE:[85,86,87,88,89,74,64,59,54,79,84,99,93,92,100],
}
```