Fun Practicals

These are not tested in 9618 exam, it just for fun, independent, further study.

- [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
- [HTML] DIY Roulette
  - Fair Roulette
  - Biased Roulette
- [Python] Bank Account Simulator
  - main.py
  - account.json
- [Python] Connect 4
  - main.py
- [HTML] Simulation RPG Combat Sample
  - Battle Sample
- [Python] Chess
  - Version 1
  - Version 2

[Python] Turn-Based Battle Game

[Python] Turn-Based Battle Game

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()
```

[Python] Turn-Based Battle Game

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]")
```

[Python] Turn-Based Battle Game

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,"Blind
ing 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],
}
```

[HTML] DIY Roulette

[HTML] DIY Roulette

Fair Roulette

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Fortune Wheel</title>
<link
href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700;900&family=Cormorant+
Garamond:wght@300;500&display=swap" rel="stylesheet" />
<style>
  :root {
    --gold: #c9a84c;
    --gold-light: #f0d080;
    --gold-dim: #7a6020;
    --bg: #0a0a0f;
    --surface: #13131a;
    --surface2: #1c1c28;
    --text: #e8dfc8;
    --text-dim: #8a7e60;
    --red: #8b1a1a;
    --green: #1a4a2e;
    --navy: #1a2040;
    --purple: #3a1a4a;
    --teal: #0f3a3a;
    --crimson: #6b1020;
    --glow: rgba(201, 168, 76, 0.4);
  }

  * { box-sizing: border-box; margin: 0; padding: 0; }

  body {
    background: var(--bg);
    color: var(--text);
    font-family: 'Cormorant Garamond', Georgia, serif;
```

```
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    padding: 24px 16px;
    overflow-x: hidden;
  }

  /* Subtle noise texture overlay */
  body::before {
    content: '';
    position: fixed;
    inset: 0;
    background-image: url("data:image/svg+xml,%3Csvg viewBox='0 0 200 200'
xmlns='http://www.w3.org/2000/svg'%3E%3Cfilter id='n'%3E%3CfeTurbulence type='fractalNoise'
baseFrequency='0.75' numOctaves='4' stitchTiles='stitch'/%3E%3C/filter%3E%3Crect
width='100%25' height='100%25' filter='url(%23n)' opacity='0.04'/%3E%3C/svg%3E");
    pointer-events: none;
    z-index: 0;
    opacity: 0.5;
  }

  .wrapper {
    position: relative;
    z-index: 1;
    width: 100%;
    max-width: 560px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 28px;
  }

  header {
    text-align: center;
  }

  header h1 {
    font-family: 'Playfair Display', serif;
```

```
    font-size: clamp(2rem, 6vw, 3rem);
    font-weight: 900;
    letter-spacing: 0.08em;
    background: linear-gradient(135deg, var(--gold-light) 0%, var(--gold) 50%, var(--gold-dim)
100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
    text-transform: uppercase;
  }

  header p {
    color: var(--text-dim);
    font-size: 0.9rem;
    letter-spacing: 0.2em;
    text-transform: uppercase;
    margin-top: 4px;
  }

  /* Controls */
  .controls {
    background: var(--surface);
    border: 1px solid rgba(201,168,76,0.15);
    border-radius: 4px;
    padding: 20px 24px;
    width: 100%;
    display: flex;
    align-items: flex-end;
    gap: 16px;
    flex-wrap: wrap;
  }

  .field {
    display: flex;
    flex-direction: column;
    gap: 6px;
    flex: 1;
    min-width: 80px;
  }
```

```
.field label {
  font-size: 0.7rem;
  letter-spacing: 0.2em;
  text-transform: uppercase;
  color: var(--gold);
  font-family: 'Cormorant Garamond', serif;
  font-weight: 500;
}

.field input[type="number"] {
  background: var(--surface2);
  border: 1px solid rgba(201,168,76,0.2);
  color: var(--text);
  font-family: 'Playfair Display', serif;
  font-size: 1.3rem;
  padding: 8px 12px;
  border-radius: 3px;
  width: 100%;
  outline: none;
  transition: border-color 0.2s;
  -moz-appearance: textfield;
}
.field input::-webkit-outer-spin-button,
.field input::-webkit-inner-spin-button { -webkit-appearance: none; }
.field input:focus { border-color: var(--gold); }

.range-sep {
  color: var(--text-dim);
  font-size: 1.4rem;
  padding-bottom: 10px;
  font-family: 'Playfair Display', serif;
}

.btn-build {
  background: transparent;
  border: 1px solid var(--gold);
  color: var(--gold);
  font-family: 'Cormorant Garamond', serif;
  font-size: 0.75rem;
  letter-spacing: 0.2em;
```

```
  text-transform: uppercase;
  padding: 10px 18px;
  border-radius: 3px;
  cursor: pointer;
  transition: background 0.2s, color 0.2s;
  white-space: nowrap;
  align-self: flex-end;
}
.btn-build:hover { background: var(--gold); color: var(--bg); }

/* Wheel area */
.wheel-area {
  position: relative;
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
  width: 100%;
}

.wheel-container {
  position: relative;
  width: clamp(280px, 80vw, 420px);
  height: clamp(280px, 80vw, 420px);
}

/* Pointer / arrow */
.pointer {
  position: absolute;
  top: -14px;
  left: 50%;
  transform: translateX(-50%);
  z-index: 10;
  filter: drop-shadow(0 0 6px var(--gold));
}
.pointer svg { display: block; }

/* Glow ring behind canvas */
.wheel-glow {
  position: absolute;
```

```
  inset: -10px;
  border-radius: 50%;
  background: radial-gradient(circle, rgba(201,168,76,0.08) 60%, transparent 75%);
  pointer-events: none;
  transition: opacity 0.4s;
  opacity: 0;
}
.wheel-glow.active { opacity: 1; }

canvas {
  border-radius: 50%;
  display: block;
  width: 100%;
  height: 100%;
}

/* Center hub */
.hub {
  position: absolute;
  top: 50%; left: 50%;
  transform: translate(-50%, -50%);
  width: 40px; height: 40px;
  border-radius: 50%;
  background: radial-gradient(circle at 35% 35%, #e8d090, #8a6018);
  border: 3px solid #1a1410;
  box-shadow: 0 0 12px rgba(0,0,0,0.8), 0 0 6px rgba(201,168,76,0.3);
  z-index: 5;
}

/* Spin button */
.btn-spin {
  position: relative;
  background: linear-gradient(135deg, #8a6018 0%, var(--gold) 50%, #8a6018 100%);
  border: none;
  color: #0a0a0f;
  font-family: 'Playfair Display', serif;
  font-size: 1rem;
  font-weight: 700;
  letter-spacing: 0.15em;
  text-transform: uppercase;
```

```
    padding: 14px 48px;
    border-radius: 3px;
    cursor: pointer;
    box-shadow: 0 4px 20px rgba(201,168,76,0.25);
    transition: transform 0.1s, box-shadow 0.2s, opacity 0.2s;
    overflow: hidden;
  }
  .btn-spin::before {
    content: '';
    position: absolute;
    inset: 0;
    background: linear-gradient(135deg, transparent 30%, rgba(255,255,255,0.2) 50%,
transparent 70%);
    transform: translateX(-100%);
    transition: transform 0.5s;
  }
  .btn-spin:hover::before { transform: translateX(100%); }
  .btn-spin:hover { box-shadow: 0 6px 30px rgba(201,168,76,0.45); transform: translateY(-1px);
}
  .btn-spin:active { transform: translateY(0); }
  .btn-spin:disabled { opacity: 0.5; cursor: not-allowed; transform: none; }

  /* Result display */
  .result-card {
    background: var(--surface);
    border: 1px solid rgba(201,168,76,0.2);
    border-radius: 4px;
    padding: 18px 32px;
    text-align: center;
    width: 100%;
    min-height: 78px;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    gap: 4px;
    transition: border-color 0.4s;
  }
  .result-card.highlight { border-color: var(--gold); }
```

```
.result-label {
  font-size: 0.65rem;
  letter-spacing: 0.25em;
  text-transform: uppercase;
  color: var(--text-dim);
}

.result-value {
  font-family: 'Playfair Display', serif;
  font-size: clamp(2.2rem, 8vw, 3.5rem);
  font-weight: 900;
  color: var(--gold-light);
  line-height: 1;
  text-shadow: 0 0 30px rgba(240,208,128,0.4);
  transition: opacity 0.3s;
}

.result-sub {
  font-size: 0.75rem;
  color: var(--text-dim);
  letter-spacing: 0.1em;
}

.placeholder-msg {
  color: var(--text-dim);
  font-size: 0.85rem;
  letter-spacing: 0.1em;
  font-style: italic;
}

/* Error */
.error-msg {
  color: #e05050;
  font-size: 0.8rem;
  letter-spacing: 0.1em;
  text-align: center;
  min-height: 18px;
}

/* Note about 3 */
```

```
  .note {
    font-size: 0.72rem;
    color: var(--text-dim);
    letter-spacing: 0.08em;
    text-align: center;
    line-height: 1.6;
  }
  .note span { color: var(--gold); }

  /* Spinning animation on wheel */
  @keyframes pulseGold {
    0%, 100% { box-shadow: 0 0 18px rgba(201,168,76,0.2); }
    50% { box-shadow: 0 0 40px rgba(201,168,76,0.6); }
  }
  .result-card.highlight { animation: pulseGold 1.5s ease-in-out 3; }

  /* Divider */
  .divider {
    width: 100%;
    height: 1px;
    background: linear-gradient(90deg, transparent, rgba(201,168,76,0.2), transparent);
  }
</style>
</head>
<body>
<div class="wrapper">
  <header>
    <h1>Fortune Wheel</h1>
    <p>Spin &amp;amp; Discover Your Number</p>
  </header>

  <div class="controls">
    <div class="field">
      <label>Lower Bound</label>
      <input type="number" id="lb" value="1" step="1" />
    </div>
    <div class="range-sep">–</div>
    <div class="field">
      <label>Upper Bound</label>
      <input type="number" id="ub" value="8" step="1" />
```

```
    </div>
    <button class="btn-build" onclick="buildWheel()">Build</button>
  </div>

  <div class="error-msg" id="errorMsg"></div>

  <div class="wheel-area">
    <div class="wheel-container" id="wheelContainer">
      <div class="wheel-glow" id="wheelGlow"></div>
      <!-- Pointer arrow at top -->
      <div class="pointer">
        <svg width="24" height="28" viewBox="0 0 24 28" fill="none">
          <path d="M12 28 L0 4 Q12 0 24 4 Z" fill="#c9a84c"/>
          <path d="M12 26 L2 6 Q12 2 22 6 Z" fill="#f0d080"/>
        </svg>
      </div>
      <canvas id="wheelCanvas"></canvas>
      <div class="hub"></div>
    </div>

    <button class="btn-spin" id="spinBtn" onclick="spin()" disabled>SPIN</button>
  </div>

  <div class="divider"></div>

  <div class="result-card" id="resultCard">
    <p class="placeholder-msg">Build your wheel &amp;amp; spin to reveal</p>
  </div>

  <p class="note">
    Numbers from <span id="noteRange">—</span> each have a perfectly <span>equal chance</span>
     ·  40% chance of a reverse <span>bounce</span> on each spin
  </p>
</div>

<script>
// ─── State ──────────────────────────────────────────────────────────
let wheelData = null;
let currentRotation = 0;
let isSpinning = false;
```

```
let animFrame = null;

const SEGMENT_COLORS = [
  ['#6b1010','#9b2020'],
  ['#1a3a1a','#2a5a2a'],
  ['#0f1f4a','#1a3070'],
  ['#2a0f4a','#441880'],
  ['#0f2f3a','#175060'],
  ['#3a1a0a','#6a3010'],
  ['#1a0a2a','#302050'],
  ['#1f1a0f','#40381a'],
];

// ── Build Wheel ──────────────────────────────────────────────────────────
function buildWheel() {
  const lbEl = document.getElementById('lb');
  const ubEl = document.getElementById('ub');
  const errEl = document.getElementById('errorMsg');

  const lb = parseInt(lbEl.value);
  const ub = parseInt(ubEl.value);

  errEl.textContent = '';

  if (isNaN(lb) || isNaN(ub)) { errEl.textContent = 'Please enter valid integers.'; return; }
  if (lb > ub) { errEl.textContent = 'Lower bound must be ≤ upper bound.'; return; }
  if (ub - lb > 49) { errEl.textContent = 'Range too large — please keep it within 50
numbers.'; return; }

  const numbers = [];
  for (let i = lb; i <= ub; i++) numbers.push(i);
  const N = numbers.length;

  // All equal probability
  const probs = numbers.map(() => 1 / N);

  // Build segments — equal angular slices
  const segAngle = (2 * Math.PI) / N;
  let angle = -Math.PI / 2;
  const segments = numbers.map((n, i) => {
```

```
    const start = angle;
    const end = angle + segAngle;
    const mid = angle + segAngle / 2;
    angle += segAngle;
    return { number: n, prob: 1 / N, startAngle: start, endAngle: end, midAngle: mid,
colorIdx: i % SEGMENT_COLORS.length };
  });

  wheelData = { numbers, probs, segments };
  currentRotation = 0;

  document.getElementById('noteRange').textContent = `${lb} to ${ub}`;
  document.getElementById('spinBtn').disabled = false;
  document.getElementById('resultCard').className = 'result-card';
  document.getElementById('resultCard').innerHTML = '<p class="placeholder-msg">Ready — hit
SPIN!</p>';

  drawWheel(0);
}

// ─── Draw ──────────────────────────────────────────────────────────────
function drawWheel(rotation) {
  if (!wheelData) return;

  const canvas = document.getElementById('wheelCanvas');
  const container = document.getElementById('wheelContainer');
  const size = container.clientWidth;
  canvas.width = size * devicePixelRatio;
  canvas.height = size * devicePixelRatio;
  canvas.style.width = size + 'px';
  canvas.style.height = size + 'px';

  const ctx = canvas.getContext('2d');
  ctx.scale(devicePixelRatio, devicePixelRatio);

  const cx = size / 2;
  const cy = size / 2;
  const R = size / 2 - 8;

  ctx.clearRect(0, 0, size, size);
```

```
// Outer gold ring
ctx.beginPath();
ctx.arc(cx, cy, R + 6, 0, Math.PI * 2);
const goldRing = ctx.createRadialGradient(cx, cy, R, cx, cy, R + 6);
goldRing.addColorStop(0, '#8a6018');
goldRing.addColorStop(0.5, '#c9a84c');
goldRing.addColorStop(1, '#f0d080');
ctx.fillStyle = goldRing;
ctx.fill();

wheelData.segments.forEach((seg) => {
  const start = seg.startAngle + rotation;
  const end = seg.endAngle + rotation;
  const [dark, light] = SEGMENT_COLORS[seg.colorIdx];

  ctx.beginPath();
  ctx.moveTo(cx, cy);
  ctx.arc(cx, cy, R, start, end);
  ctx.closePath();

  const grad = ctx.createRadialGradient(cx, cy, 0, cx, cy, R);
  grad.addColorStop(0, light);
  grad.addColorStop(1, dark);
  ctx.fillStyle = grad;
  ctx.fill();

  ctx.strokeStyle = 'rgba(0,0,0,0.5)';
  ctx.lineWidth = 1.5;
  ctx.stroke();

  // Label
  const labelR = R * 0.68;
  const midAngle = (start + end) / 2;
  const lx = cx + labelR * Math.cos(midAngle);
  const ly = cy + labelR * Math.sin(midAngle);

  ctx.save();
  ctx.translate(lx, ly);
  ctx.rotate(midAngle + Math.PI / 2);
```

```
    const sweep = seg.endAngle - seg.startAngle;
    const fontSize = Math.max(9, Math.min(sweep * R * 0.38, R * 0.22));
    ctx.font = `bold ${fontSize}px 'Playfair Display', Georgia, serif`;
    ctx.fillStyle = '#ffffff';
    ctx.textAlign = 'center';
    ctx.textBaseline = 'middle';
    ctx.shadowColor = 'rgba(0,0,0,0.8)';
    ctx.shadowBlur = 4;
    ctx.fillText(String(seg.number), 0, 0);
    ctx.restore();

    // Tick mark
    const tickX = cx + (R - 4) * Math.cos(start);
    const tickY = cy + (R - 4) * Math.sin(start);
    ctx.beginPath();
    ctx.arc(tickX, tickY, 2, 0, Math.PI * 2);
    ctx.fillStyle = 'rgba(201,168,76,0.5)';
    ctx.fill();
  });

  // Inner circle
  ctx.beginPath();
  ctx.arc(cx, cy, R * 0.24, 0, Math.PI * 2);
  ctx.fillStyle = '#0d0d14';
  ctx.fill();
  ctx.strokeStyle = '#c9a84c';
  ctx.lineWidth = 2;
  ctx.stroke();
}

// ─── Find which segment is under the pointer ──────────────────────────
function segmentAtPointer(rotation) {
  // Pointer is at angle -π/2 in world space.
  // A point on the wheel at angle θ (in wheel space) appears at θ + rotation.
  // We want θ + rotation ≡ -π/2  →  θ ≡ -π/2 - rotation
  const pointerAngle = ((-Math.PI / 2 - rotation) % (2 * Math.PI) + 2 * Math.PI) % (2 *
Math.PI);
  // Segment angles are stored from -π/2, convert to [0, 2π)
  return wheelData.segments.find(seg => {
```

```
    const s = ((seg.startAngle + Math.PI / 2 + 2 * Math.PI) % (2 * Math.PI));
    const e = ((seg.endAngle   + Math.PI / 2 + 2 * Math.PI) % (2 * Math.PI));
    if (s < e) return pointerAngle >= s && pointerAngle < e;
    return pointerAngle >= s || pointerAngle < e; // wraps around 0
  }) || wheelData.segments[0];
}

// ── Spin ──────────────────────────────────────────────────────────────
function spin() {
  if (isSpinning || !wheelData) return;
  isSpinning = true;

  const spinBtn   = document.getElementById('spinBtn');
  const resultCard = document.getElementById('resultCard');
  const glow      = document.getElementById('wheelGlow');

  spinBtn.disabled = true;
  resultCard.className = 'result-card';
  resultCard.innerHTML = '<p class="placeholder-msg">Spinning…</p>';

  // Pick winner uniformly
  const winIdx = Math.floor(Math.random() * wheelData.segments.length);
  const winner = wheelData.segments[winIdx];

  // Land at a random offset within the winning segment (not exactly center)
  // Keep away from edges by 15% of segment width
  const sweep = winner.endAngle - winner.startAngle;
  const margin = sweep * 0.15;
  const randomOffset = margin + Math.random() * (sweep - 2 * margin);
  const targetAngleOnWheel = winner.startAngle + randomOffset; // where we want pointer to
point

  // Compute how much we need to rotate so targetAngleOnWheel sits under the pointer (-π/2)
  const rawTarget = -Math.PI / 2 - targetAngleOnWheel - currentRotation;
  const normalised = ((rawTarget % (2 * Math.PI)) + 2 * Math.PI) % (2 * Math.PI);
  const EXTRA_SPINS = 6 + Math.floor(Math.random() * 4);
  const primarySpin = normalised + EXTRA_SPINS * 2 * Math.PI;

  // Decide if we do a reverse-bounce (40% chance)
  const doBounce = Math.random() < 0.40;
```

```
  // Bounce: overshoot by a small amount, then spring back
  const bounceOvershoot = doBounce ? (0.04 + Math.random() * 0.10) * 2 * Math.PI : 0; //
14°–36°

  const startRotation = currentRotation;
  glow.classList.add('active');

  // ── Phase 1: main deceleration spin ──────────────────────────────────
  const phase1End = startRotation + primarySpin + bounceOvershoot;
  const phase1Duration = 4200 + Math.random() * 1400;

  // ── Phase 2: bounce back (if doBounce) ───────────────────────────────
  const phase2Start = phase1End;
  const phase2End   = startRotation + primarySpin; // back to where it should land
  const phase2Duration = 500 + Math.random() * 300;

  const startTime = performance.now();
  let phase = 1;

  // Strong ease-out: fast start, very slow finish — feels like real friction
  function easeOutQuint(t) {
    return 1 - Math.pow(1 - t, 5);
  }
  // Ease-out then ease-in for bounce-back (like a spring)
  function easeInOutCubic(t) {
    return t < 0.5 ? 4 * t * t * t : 1 - Math.pow(-2 * t + 2, 3) / 2;
  }

  function animate(now) {
    if (phase === 1) {
      const elapsed = now - startTime;
      const t = Math.min(elapsed / phase1Duration, 1);
      const eased = easeOutQuint(t);
      currentRotation = startRotation + (phase1End - startRotation) * eased;
      drawWheel(currentRotation);

      if (t < 1) {
        animFrame = requestAnimationFrame(animate);
      } else {
        currentRotation = phase1End;
```

```
        if (doBounce) {
          phase = 2;
          // store phase 2 start time
          animate._phase2Start = now;
          animFrame = requestAnimationFrame(animate);
        } else {
          finish(winner);
        }
      }
    } else {
      // Phase 2: spring back
      const elapsed = now - animate._phase2Start;
      const t = Math.min(elapsed / phase2Duration, 1);
      const eased = easeInOutCubic(t);
      currentRotation = phase2Start + (phase2End - phase2Start) * eased;
      drawWheel(currentRotation);

      if (t < 1) {
        animFrame = requestAnimationFrame(animate);
      } else {
        currentRotation = phase2End;
        finish(winner);
      }
    }
  }

  animFrame = requestAnimationFrame(animate);
}

function finish(winner) {
  const spinBtn    = document.getElementById('spinBtn');
  const glow       = document.getElementById('wheelGlow');
  const isSpinningRef = isSpinning;

  isSpinning = false;
  spinBtn.disabled = false;
  drawWheel(currentRotation);

  // Re-detect actual segment under pointer (accounts for random offset)
  const actual = segmentAtPointer(currentRotation);
```

```
  showResult(actual || winner);
  setTimeout(() => glow.classList.remove('active'), 3000);
}

// ─── Show Result ──────────────────────────────────────────────────────────
function showResult(winner) {
  const resultCard = document.getElementById('resultCard');
  const pct = (winner.prob * 100).toFixed(1);

  resultCard.innerHTML = `
    <span class="result-label">The wheel has spoken</span>
    <span class="result-value">${winner.number}</span>
    <span class="result-sub">${pct}% probability</span>
  `;
  resultCard.className = 'result-card highlight';
}

// ─── Init ─────────────────────────────────────────────────────────────────
window.addEventListener('DOMContentLoaded', () => {
  const canvas = document.getElementById('wheelCanvas');
  const container = document.getElementById('wheelContainer');
  const size = container.clientWidth || 380;
  canvas.width = size * devicePixelRatio;
  canvas.height = size * devicePixelRatio;
  canvas.style.width = size + 'px';
  canvas.style.height = size + 'px';
  const ctx = canvas.getContext('2d');
  ctx.scale(devicePixelRatio, devicePixelRatio);
  const cx = size / 2, cy = size / 2, R = size / 2 - 8;
  ctx.beginPath();
  ctx.arc(cx, cy, R + 6, 0, Math.PI * 2);
  const gr = ctx.createRadialGradient(cx, cy, R, cx, cy, R + 6);
  gr.addColorStop(0, '#8a6018'); gr.addColorStop(0.5, '#c9a84c'); gr.addColorStop(1,
'#f0d080');
  ctx.fillStyle = gr; ctx.fill();
  ctx.beginPath();
  ctx.arc(cx, cy, R, 0, Math.PI * 2);
  ctx.fillStyle = '#13131a'; ctx.fill();
  ctx.font = `italic ${Math.floor(R * 0.14)}px 'Cormorant Garamond', Georgia, serif`;
  ctx.fillStyle = 'rgba(201,168,76,0.4)';
```

```
  ctx.textAlign = 'center';
  ctx.textBaseline = 'middle';
  ctx.fillText('Set bounds & build', cx, cy);
  buildWheel();
});

window.addEventListener('resize', () => {
  if (wheelData) drawWheel(currentRotation);
});
</script>
</body>
</html>
```

[HTML] DIY Roulette

Biased Roulette

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Fortune Wheel</title>
<link
href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700;900&family=Cormorant+
Garamond:wght@300;500&display=swap" rel="stylesheet" />
<style>
  :root {
    --gold: #c9a84c;
    --gold-light: #f0d080;
    --gold-dim: #7a6020;
    --bg: #0a0a0f;
    --surface: #13131a;
    --surface2: #1c1c28;
    --text: #e8dfc8;
    --text-dim: #8a7e60;
    --red: #8b1a1a;
    --green: #1a4a2e;
    --navy: #1a2040;
    --purple: #3a1a4a;
    --teal: #0f3a3a;
    --crimson: #6b1020;
    --glow: rgba(201, 168, 76, 0.4);
  }

  * { box-sizing: border-box; margin: 0; padding: 0; }

  body {
    background: var(--bg);
    color: var(--text);
    font-family: 'Cormorant Garamond', Georgia, serif;
```

```
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    padding: 24px 16px;
    overflow-x: hidden;
  }

  /* Subtle noise texture overlay */
  body::before {
    content: '';
    position: fixed;
    inset: 0;
    background-image: url("data:image/svg+xml,%3Csvg viewBox='0 0 200 200'
xmlns='http://www.w3.org/2000/svg'%3E%3Cfilter id='n'%3E%3CfeTurbulence type='fractalNoise'
baseFrequency='0.75' numOctaves='4' stitchTiles='stitch'/%3E%3C/filter%3E%3Crect
width='100%25' height='100%25' filter='url(%23n)' opacity='0.04'/%3E%3C/svg%3E");
    pointer-events: none;
    z-index: 0;
    opacity: 0.5;
  }

  .wrapper {
    position: relative;
    z-index: 1;
    width: 100%;
    max-width: 560px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 28px;
  }

  header {
    text-align: center;
  }

  header h1 {
    font-family: 'Playfair Display', serif;
```

```
    font-size: clamp(2rem, 6vw, 3rem);
    font-weight: 900;
    letter-spacing: 0.08em;
    background: linear-gradient(135deg, var(--gold-light) 0%, var(--gold) 50%, var(--gold-dim)
100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
    text-transform: uppercase;
  }

  header p {
    color: var(--text-dim);
    font-size: 0.9rem;
    letter-spacing: 0.2em;
    text-transform: uppercase;
    margin-top: 4px;
  }

  /* Controls */
  .controls {
    background: var(--surface);
    border: 1px solid rgba(201,168,76,0.15);
    border-radius: 4px;
    padding: 20px 24px;
    width: 100%;
    display: flex;
    align-items: flex-end;
    gap: 16px;
    flex-wrap: wrap;
  }

  .field {
    display: flex;
    flex-direction: column;
    gap: 6px;
    flex: 1;
    min-width: 80px;
  }
```

```
.field label {
  font-size: 0.7rem;
  letter-spacing: 0.2em;
  text-transform: uppercase;
  color: var(--gold);
  font-family: 'Cormorant Garamond', serif;
  font-weight: 500;
}

.field input[type="number"] {
  background: var(--surface2);
  border: 1px solid rgba(201,168,76,0.2);
  color: var(--text);
  font-family: 'Playfair Display', serif;
  font-size: 1.3rem;
  padding: 8px 12px;
  border-radius: 3px;
  width: 100%;
  outline: none;
  transition: border-color 0.2s;
  -moz-appearance: textfield;
}
.field input::-webkit-outer-spin-button,
.field input::-webkit-inner-spin-button { -webkit-appearance: none; }
.field input:focus { border-color: var(--gold); }

.range-sep {
  color: var(--text-dim);
  font-size: 1.4rem;
  padding-bottom: 10px;
  font-family: 'Playfair Display', serif;
}

.btn-build {
  background: transparent;
  border: 1px solid var(--gold);
  color: var(--gold);
  font-family: 'Cormorant Garamond', serif;
  font-size: 0.75rem;
  letter-spacing: 0.2em;
```

```
  text-transform: uppercase;
  padding: 10px 18px;
  border-radius: 3px;
  cursor: pointer;
  transition: background 0.2s, color 0.2s;
  white-space: nowrap;
  align-self: flex-end;
}
.btn-build:hover { background: var(--gold); color: var(--bg); }

/* Wheel area */
.wheel-area {
  position: relative;
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
  width: 100%;
}

.wheel-container {
  position: relative;
  width: clamp(280px, 80vw, 420px);
  height: clamp(280px, 80vw, 420px);
}

/* Pointer / arrow */
.pointer {
  position: absolute;
  top: -14px;
  left: 50%;
  transform: translateX(-50%);
  z-index: 10;
  filter: drop-shadow(0 0 6px var(--gold));
}
.pointer svg { display: block; }

/* Glow ring behind canvas */
.wheel-glow {
  position: absolute;
```

```
  inset: -10px;
  border-radius: 50%;
  background: radial-gradient(circle, rgba(201,168,76,0.08) 60%, transparent 75%);
  pointer-events: none;
  transition: opacity 0.4s;
  opacity: 0;
}
.wheel-glow.active { opacity: 1; }

canvas {
  border-radius: 50%;
  display: block;
  width: 100%;
  height: 100%;
}

/* Center hub */
.hub {
  position: absolute;
  top: 50%; left: 50%;
  transform: translate(-50%, -50%);
  width: 40px; height: 40px;
  border-radius: 50%;
  background: radial-gradient(circle at 35% 35%, #e8d090, #8a6018);
  border: 3px solid #1a1410;
  box-shadow: 0 0 12px rgba(0,0,0,0.8), 0 0 6px rgba(201,168,76,0.3);
  z-index: 5;
}

/* Spin button */
.btn-spin {
  position: relative;
  background: linear-gradient(135deg, #8a6018 0%, var(--gold) 50%, #8a6018 100%);
  border: none;
  color: #0a0a0f;
  font-family: 'Playfair Display', serif;
  font-size: 1rem;
  font-weight: 700;
  letter-spacing: 0.15em;
  text-transform: uppercase;
```

```
    padding: 14px 48px;
    border-radius: 3px;
    cursor: pointer;
    box-shadow: 0 4px 20px rgba(201,168,76,0.25);
    transition: transform 0.1s, box-shadow 0.2s, opacity 0.2s;
    overflow: hidden;
  }
  .btn-spin::before {
    content: '';
    position: absolute;
    inset: 0;
    background: linear-gradient(135deg, transparent 30%, rgba(255,255,255,0.2) 50%,
transparent 70%);
    transform: translateX(-100%);
    transition: transform 0.5s;
  }
  .btn-spin:hover::before { transform: translateX(100%); }
  .btn-spin:hover { box-shadow: 0 6px 30px rgba(201,168,76,0.45); transform: translateY(-1px);
}
  .btn-spin:active { transform: translateY(0); }
  .btn-spin:disabled { opacity: 0.5; cursor: not-allowed; transform: none; }

  /* Result display */
  .result-card {
    background: var(--surface);
    border: 1px solid rgba(201,168,76,0.2);
    border-radius: 4px;
    padding: 18px 32px;
    text-align: center;
    width: 100%;
    min-height: 78px;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    gap: 4px;
    transition: border-color 0.4s;
  }
  .result-card.highlight { border-color: var(--gold); }
```

```
.result-label {
  font-size: 0.65rem;
  letter-spacing: 0.25em;
  text-transform: uppercase;
  color: var(--text-dim);
}

.result-value {
  font-family: 'Playfair Display', serif;
  font-size: clamp(2.2rem, 8vw, 3.5rem);
  font-weight: 900;
  color: var(--gold-light);
  line-height: 1;
  text-shadow: 0 0 30px rgba(240,208,128,0.4);
  transition: opacity 0.3s;
}

.result-sub {
  font-size: 0.75rem;
  color: var(--text-dim);
  letter-spacing: 0.1em;
}

.placeholder-msg {
  color: var(--text-dim);
  font-size: 0.85rem;
  letter-spacing: 0.1em;
  font-style: italic;
}

/* Error */
.error-msg {
  color: #e05050;
  font-size: 0.8rem;
  letter-spacing: 0.1em;
  text-align: center;
  min-height: 18px;
}

/* Note about 3 */
```

```
  .note {
    font-size: 0.72rem;
    color: var(--text-dim);
    letter-spacing: 0.08em;
    text-align: center;
    line-height: 1.6;
  }
  .note span { color: var(--gold); }

  /* Spinning animation on wheel */
  @keyframes pulseGold {
    0%, 100% { box-shadow: 0 0 18px rgba(201,168,76,0.2); }
    50% { box-shadow: 0 0 40px rgba(201,168,76,0.6); }
  }
  .result-card.highlight { animation: pulseGold 1.5s ease-in-out 3; }

  /* Divider */
  .divider {
    width: 100%;
    height: 1px;
    background: linear-gradient(90deg, transparent, rgba(201,168,76,0.2), transparent);
  }
</style>
</head>
<body>
<div class="wrapper">
  <header>
    <h1>Fortune Wheel</h1>
    <p>Spin &amp;amp; Discover Your Number</p>
  </header>

  <div class="controls">
    <div class="field">
      <label>Lower Bound</label>
      <input type="number" id="lb" value="1" step="1" />
    </div>
    <div class="range-sep">–</div>
    <div class="field">
      <label>Upper Bound</label>
      <input type="number" id="ub" value="8" step="1" />
```

```
    </div>
    <button class="btn-build" onclick="buildWheel()">Build</button>
  </div>

  <div class="error-msg" id="errorMsg"></div>

  <div class="wheel-area">
    <div class="wheel-container" id="wheelContainer">
      <div class="wheel-glow" id="wheelGlow"></div>
      <!-- Pointer arrow at top -->
      <div class="pointer">
        <svg width="24" height="28" viewBox="0 0 24 28" fill="none">
          <path d="M12 28 L0 4 Q12 0 24 4 Z" fill="#c9a84c"/>
          <path d="M12 26 L2 6 Q12 2 22 6 Z" fill="#f0d080"/>
        </svg>
      </div>
      <canvas id="wheelCanvas"></canvas>
      <div class="hub"></div>
    </div>

    <button class="btn-spin" id="spinBtn" onclick="spin()" disabled>SPIN</button>
  </div>

  <div class="divider"></div>

  <div class="result-card" id="resultCard">
    <p class="placeholder-msg">Build your wheel &amp;amp; spin to reveal</p>
  </div>

  <p class="note">
    Numbers from <span id="noteRange">—</span> each have an equal chance
     ·  <span>3</span> always carries a <span>+20% boost</span> when in range
  </p>
</div>

<script>
// ─── State ──────────────────────────────────────────────────────────
let wheelData = null;       // { segments, numbers, probs }
let currentRotation = 0;    // current total rotation in radians
let isSpinning = false;
```

```
let animFrame = null;

const SEGMENT_COLORS = [
  ['#6b1010','#9b2020'],  // deep red
  ['#1a3a1a','#2a5a2a'],  // forest green
  ['#0f1f4a','#1a3070'],  // navy
  ['#2a0f4a','#441880'],  // purple
  ['#0f2f3a','#175060'],  // teal
  ['#3a1a0a','#6a3010'],  // brown
  ['#1a0a2a','#302050'],  // indigo
  ['#1f1a0f','#40381a'],  // olive-dark
];

// ── Build Wheel ──────────────────────────────────────────────────────────
function buildWheel() {
  const lbEl = document.getElementById('lb');
  const ubEl = document.getElementById('ub');
  const errEl = document.getElementById('errorMsg');

  const lb = parseInt(lbEl.value);
  const ub = parseInt(ubEl.value);

  errEl.textContent = '';

  if (isNaN(lb) || isNaN(ub)) { errEl.textContent = 'Please enter valid integers.'; return; }
  if (lb > ub) { errEl.textContent = 'Lower bound must be ≤ upper bound.'; return; }
  if (ub - lb > 49) { errEl.textContent = 'Range too large — please keep it within 50
numbers.'; return; }

  const numbers = [];
  for (let i = lb; i <= ub; i++) numbers.push(i);
  const N = numbers.length;
  const has3 = numbers.includes(3);

  // Build probabilities
  let probs;
  if (!has3 || N === 1) {
    probs = numbers.map(() => 1 / N);
  } else {
    const p3 = 0.20;
```

```
    const pOther = 0.80 / (N - 1);
    probs = numbers.map(n => n === 3 ? p3 : pOther);
  }

  // Build segments (angles start at -π/2 = top, go clockwise)
  let angle = -Math.PI / 2;
  const segments = numbers.map((n, i) => {
    const sweep = probs[i] * 2 * Math.PI;
    const mid = angle + sweep / 2;
    const seg = { number: n, prob: probs[i], startAngle: angle, endAngle: angle + sweep,
midAngle: mid, colorIdx: i % SEGMENT_COLORS.length };
    angle += sweep;
    return seg;
  });

  wheelData = { numbers, probs, segments };
  currentRotation = 0;

  document.getElementById('noteRange').textContent = `${lb} to ${ub}`;
  document.getElementById('spinBtn').disabled = false;
  document.getElementById('resultCard').className = 'result-card';
  document.getElementById('resultCard').innerHTML = '<p class="placeholder-msg">Ready — hit
SPIN!</p>';

  drawWheel(0);
}

// ─── Draw ──────────────────────────────────────────────────────────────
function drawWheel(rotation) {
  if (!wheelData) return;

  const canvas = document.getElementById('wheelCanvas');
  const container = document.getElementById('wheelContainer');
  const size = container.clientWidth;
  canvas.width = size * devicePixelRatio;
  canvas.height = size * devicePixelRatio;
  canvas.style.width = size + 'px';
  canvas.style.height = size + 'px';

  const ctx = canvas.getContext('2d');
```

```
ctx.scale(devicePixelRatio, devicePixelRatio);

const cx = size / 2;
const cy = size / 2;
const R = size / 2 - 8;    // outer radius
const innerR = R * 0.12;   // hub hole

ctx.clearRect(0, 0, size, size);

// Outer gold ring
ctx.beginPath();
ctx.arc(cx, cy, R + 6, 0, Math.PI * 2);
const goldRing = ctx.createRadialGradient(cx, cy, R, cx, cy, R + 6);
goldRing.addColorStop(0, '#8a6018');
goldRing.addColorStop(0.5, '#c9a84c');
goldRing.addColorStop(1, '#f0d080');
ctx.fillStyle = goldRing;
ctx.fill();

// Draw each segment
wheelData.segments.forEach((seg, i) => {
  const start = seg.startAngle + rotation;
  const end = seg.endAngle + rotation;
  const [dark, light] = SEGMENT_COLORS[seg.colorIdx];

  // Fill
  ctx.beginPath();
  ctx.moveTo(cx, cy);
  ctx.arc(cx, cy, R, start, end);
  ctx.closePath();

  const grad = ctx.createRadialGradient(cx, cy, 0, cx, cy, R);
  grad.addColorStop(0, light);
  grad.addColorStop(1, dark);
  ctx.fillStyle = grad;
  ctx.fill();

  // Subtle border between segments
  ctx.strokeStyle = 'rgba(0,0,0,0.5)';
  ctx.lineWidth = 1.5;
```

```
    ctx.stroke();

    // Label
    const labelR = R * 0.68;
    const midAngle = (start + end) / 2;
    const lx = cx + labelR * Math.cos(midAngle);
    const ly = cy + labelR * Math.sin(midAngle);

    ctx.save();
    ctx.translate(lx, ly);
    ctx.rotate(midAngle + Math.PI / 2);

    const sweep = seg.endAngle - seg.startAngle;
    const fontSize = Math.max(9, Math.min(sweep * R * 0.38, R * 0.22));
    ctx.font = `bold ${fontSize}px 'Playfair Display', Georgia, serif`;
    ctx.fillStyle = '#ffffff';
    ctx.textAlign = 'center';
    ctx.textBaseline = 'middle';
    ctx.shadowColor = 'rgba(0,0,0,0.8)';
    ctx.shadowBlur = 4;

    // Special gold color for 3
    if (seg.number === 3 && wheelData.numbers.includes(3)) {
      ctx.fillStyle = '#f0d080';
      ctx.shadowColor = 'rgba(0,0,0,0.9)';
    }

    ctx.fillText(String(seg.number), 0, 0);
    ctx.restore();

    // Gold dot at segment start radius (decorative tick)
    const tickX = cx + (R - 4) * Math.cos(start);
    const tickY = cy + (R - 4) * Math.sin(start);
    ctx.beginPath();
    ctx.arc(tickX, tickY, 2, 0, Math.PI * 2);
    ctx.fillStyle = 'rgba(201,168,76,0.5)';
    ctx.fill();
  });

  // Inner dark circle
```

```
  ctx.beginPath();
  ctx.arc(cx, cy, innerR * 2, 0, Math.PI * 2);
  ctx.fillStyle = '#0d0d14';
  ctx.fill();
  ctx.strokeStyle = '#c9a84c';
  ctx.lineWidth = 2;
  ctx.stroke();
}

// ─── Weighted random pick ──────────────────────────────────────────
function pickWinner() {
  const { numbers, probs } = wheelData;
  let r = Math.random();
  for (let i = 0; i < numbers.length; i++) {
    r -= probs[i];
    if (r <= 0) return i;
  }
  return numbers.length - 1;
}

// ─── Spin ──────────────────────────────────────────────────────────
function spin() {
  if (isSpinning || !wheelData) return;
  isSpinning = true;

  const spinBtn = document.getElementById('spinBtn');
  const resultCard = document.getElementById('resultCard');
  const glow = document.getElementById('wheelGlow');

  spinBtn.disabled = true;
  resultCard.className = 'result-card';
  resultCard.innerHTML = '<p class="placeholder-msg" style="animation: none;">Spinning…</p>';

  // Pick winner
  const winIdx = pickWinner();
  const winner = wheelData.segments[winIdx];

  // We want the pointer (at top = -π/2 absolute) to land on the winner's midAngle.
  // After adding `currentRotation`, the displayed midAngle is: winner.midAngle +
currentRotation
```

```
  // We want that to equal -π/2 + 2πk for some integer k.
  // So additional spin needed: targetRotation = -π/2 - winner.midAngle - currentRotation (mod 2π)
  // Then add extra full spins for drama.

  const EXTRA_SPINS = 6 + Math.floor(Math.random() * 4); // 6–9 full rotations
  const rawTarget = -Math.PI / 2 - winner.midAngle - currentRotation;
  const normalised = ((rawTarget % (2 * Math.PI)) + 2 * Math.PI) % (2 * Math.PI);
  const totalSpin = normalised + EXTRA_SPINS * 2 * Math.PI;

  const startRotation = currentRotation;
  const endRotation = currentRotation + totalSpin;
  const duration = 4000 + Math.random() * 1500; // 4–5.5 s
  const startTime = performance.now();

  glow.classList.add('active');

  function easeOut(t) {
    // Cubic ease-out with a slight bounce feel
    return 1 - Math.pow(1 - t, 3.5);
  }

  function animate(now) {
    const elapsed = now - startTime;
    const t = Math.min(elapsed / duration, 1);
    const eased = easeOut(t);
    currentRotation = startRotation + (endRotation - startRotation) * eased;
    drawWheel(currentRotation);

    if (t < 1) {
      animFrame = requestAnimationFrame(animate);
    } else {
      // Done
      currentRotation = endRotation;
      isSpinning = false;
      spinBtn.disabled = false;
      showResult(winner);
    }
  }
```

```
  animFrame = requestAnimationFrame(animate);
}

// ── Show Result ──────────────────────────────────────────────────────────
function showResult(winner) {
  const resultCard = document.getElementById('resultCard');
  const glow = document.getElementById('wheelGlow');
  const is3 = winner.number === 3;
  const pct = (winner.prob * 100).toFixed(1);

  resultCard.innerHTML = `
    <span class="result-label">The wheel has spoken</span>
    <span class="result-value">${winner.number}</span>
    <span class="result-sub">${is3 ? '★ Lucky 3 · ' : ''}${pct}% probability</span>
  `;
  resultCard.className = 'result-card highlight';

  setTimeout(() => glow.classList.remove('active'), 3000);
}

// ── Init ─────────────────────────────────────────────────────────────────
window.addEventListener('DOMContentLoaded', () => {
  // Draw a placeholder empty disc
  const canvas = document.getElementById('wheelCanvas');
  const container = document.getElementById('wheelContainer');
  const size = container.clientWidth || 380;
  canvas.width = size * devicePixelRatio;
  canvas.height = size * devicePixelRatio;
  canvas.style.width = size + 'px';
  canvas.style.height = size + 'px';
  const ctx = canvas.getContext('2d');
  ctx.scale(devicePixelRatio, devicePixelRatio);
  const cx = size / 2, cy = size / 2, R = size / 2 - 8;
  // Gold ring
  ctx.beginPath();
  ctx.arc(cx, cy, R + 6, 0, Math.PI * 2);
  const gr = ctx.createRadialGradient(cx, cy, R, cx, cy, R + 6);
  gr.addColorStop(0, '#8a6018'); gr.addColorStop(0.5, '#c9a84c'); gr.addColorStop(1,
'#f0d080');
  ctx.fillStyle = gr; ctx.fill();
```

```
  // Dark disc
  ctx.beginPath();
  ctx.arc(cx, cy, R, 0, Math.PI * 2);
  ctx.fillStyle = '#13131a'; ctx.fill();
  // Center text
  ctx.font = `italic ${Math.floor(R * 0.14)}px 'Cormorant Garamond', Georgia, serif`;
  ctx.fillStyle = 'rgba(201,168,76,0.4)';
  ctx.textAlign = 'center';
  ctx.textBaseline = 'middle';
  ctx.fillText('Set bounds & build', cx, cy);

  // Build default wheel
  buildWheel();
});

window.addEventListener('resize', () => {
  if (wheelData) drawWheel(currentRotation);
});
</script>
</body>
</html>
```

[Python] Bank Account Simulator

[Python] Bank Account Simulator

main.py

```
import json
import datetime

# load account if exists
def load_account():
    try:
        with open("account.json", "r") as f:
            account = json.load(f)
            return account
    except FileNotFoundError:
        return -1

# create new account
def create_account():
    account = {
        "balance": 0.0,
        "transactions": []
    }
    with open("account.json", "w") as f:
        json.dump(account, f)
    return account

# save account state
def save_account(account):
    with open("account.json", "w") as f:
        json.dump(account, f)

# deposit function
def deposit(account, amount):
    account["balance"] += amount
    account["transactions"].append({
        "transaction_id": transaction_id_generator(len(account["transactions"])+1),
        "type": "deposit",
        "amount": amount,
```

```
        "date": str(datetime.datetime.now()),
        "balance_after": account["balance"]
    })
    save_account(account)

# Generate a unique 10-digit transaction ID
def transaction_id_generator(number):
    return str(number).zfill(10)

# withdraw function
def withdraw(account, amount):
    account["balance"] -= amount
    account["transactions"].append({
        "transaction_id": transaction_id_generator(len(account["transactions"])+1),
        "type": "withdraw",
        "amount": amount,
        "date": str(datetime.datetime.now()),
        "balance_after": account["balance"]
    })
    save_account(account)

# check balance function
def check_balance(account):
    return account["balance"]

# view transactions function
def view_transactions(account):
    if len(account["transactions"]) == 0:
        return -1
    else:
        return account["transactions"]

# search transaction by ID function
def search_transaction_by_id(account, transaction_id):
    for i in account["transactions"]:
        if i["transaction_id"] == transaction_id:
            return i
    return -1

# search transaction by amount function
```

```
def search_transaction_by_amount(account, amount):
    results = []
    for i in account["transactions"]:
        if i["amount"] == amount:
            results.append(i)
    if len(results) == 0:
        return -1
    else:
        return results

# search transaction by date function
def search_transaction_by_date(account, date):
    results = []
    for i in account["transactions"]:
        if date in i["date"]:
            results.append(i)
    if len(results) == 0:
        return -1
    else:
        return results

# main program
def main():
    # load or create account
    account = load_account()
    if account == -1:
        account = create_account()

    program_state = True
    while program_state:
        # main menu input
        while True:
            try:
                print("\nInput option:\n1. Deposit\n2. Withdraw\n3. Check Balance\n4. View
Transactions\n5. Search Transaction\n6. Exit")
                choice = int(input("Enter choice (1-6): "))
                if choice >= 1 and choice <= 6:
                    break
                else:
                    print("Invalid choice. Please enter a number between 1 and 6.")
```

```
            except ValueError:
                print("Invalid input. Please enter a number between 1 and 6.")

        # carry out the chosen operations with:
        # 1. input with exception handling
        # 2. perform operation
        # 3. display result with formatting

        # options:
        # 1. deposit
        if choice == 1:
            while True:
                try:
                    amount = float(input("Enter amount to deposit: "))
                    if amount > 0:
                        deposit(account, amount)
                        print(f"Deposited: ${amount:.2f}")
                        break
                    else:
                        print("Amount must be positive.")
                except ValueError:
                    print("Invalid input. Please enter a valid amount.")

        # 2. withdraw
        elif choice == 2:
            if account["balance"] <= 0:
                print("Insufficient balance to withdraw.")
            else:
                while True:
                    try:
                        amount = float(input("Enter amount to withdraw: "))
                        if amount > 0 and amount <= account["balance"]:
                            withdraw(account, amount)
                            print(f"Withdrew: ${amount:.2f}")
                            break
                        else:
                            print("Invalid amount. Please enter a positive amount not
exceeding your balance.")
                    except ValueError:
                        print("Invalid input. Please enter a valid amount.")
```

```
        # 3. check balance
        elif choice == 3:
            balance = check_balance(account)
            print(f"Current Balance: ${balance:.2f}")

        # 4. view transactions
        elif choice == 4:
            transactions = view_transactions(account)
            if transactions == -1:
                print("Record is empty.")
            else:
                print("Transaction History:")
                for i in transactions:
                    print(f"ID: {i['transaction_id']} | Type: {i['type']} | Amount:
${i['amount']:.2f} | Date: {i['date']} | Balance After: ${i['balance_after']:.2f}")

        elif choice == 5:
            # search transaction submenu
            while True:
                try:
                    print("Search options:\n1. By Transaction ID\n2. By Amount\n3. By Date\n4.
Back to Main Menu")
                    search_choice = int(input("Enter choice (1-4): "))
                    if search_choice >= 1 and search_choice <= 4:
                        break
                    else:
                        print("Invalid choice. Please enter a number between 1 and 4.")
                except ValueError:
                    print("Invalid input. Please enter a number between 1 and 4.")

            # 1. search by transaction ID
            if search_choice == 1:
                # validate id input
                while True:
                    try:
                        search_id = input("Enter Transaction ID to search: ")
                        if len(search_id) == 10 and search_id.isdigit():
                            break
                        else:
```

```
                        print("Invalid Transaction ID. It must be a 10-digit number.")
                except ValueError:
                    print("Invalid Transaction ID. It must be a 10-digit number.")

            # perform search
            result = search_transaction_by_id(account, search_id)
            if result == -1:
                print("Transaction not found.")
            else:
                print("Transaction Found:")
                print(f"ID: {result['transaction_id']} | Type: {result['type']} | Amount:
${result['amount']:.2f} | Date: {result['date']} | Balance After:
${result['balance_after']:.2f}")

        # 2. search by amount
        elif search_choice == 2:
            # validate amount input
            while True:
                try:
                    search_amount = float(input("Enter Amount to search: "))
                    if search_amount > 0:
                        break
                    else:
                        print("Amount must be positive.")
                except ValueError:
                    print("Invalid input. Please enter a valid amount.")

            # perform search
            results = search_transaction_by_amount(account, search_amount)
            if results == -1:
                print("No transactions found with specified amount.")
            else:
                print("Transactions Found:")
                for i in results:
                    print(f"ID: {i['transaction_id']} | Type: {i['type']} | Amount:
${i['amount']:.2f} | Date: {i['date']} | Balance After: ${i['balance_after']:.2f}")

        # 3. search by date
        elif search_choice == 3:
            # validate date input
```

```
                while True:
                    try:
                        search_year = input("Enter Year (YYYY) to search: ")
                        search_month = input("Enter Month (MM) to search: ")
                        search_day = input("Enter Day (DD) to search: ")
                        if (len(search_year) == 4 and search_year.isdigit() and
                            len(search_month) == 2 and search_month.isdigit() and
                            len(search_day) == 2 and search_day.isdigit()):
                            search_date = f"{search_year}-{search_month}-{search_day}"
                            break
                        else:
                            print("Invalid date format. Please enter valid year, month, and
day.")
                    except ValueError:
                        print("Invalid date format. Please enter valid year, month, and day.")

                # perform search
                results = search_transaction_by_date(account, search_date)
                if results == -1:
                    print("No transactions found on the specified date.")
                else:
                    print("Transactions Found:")
                    for i in results:
                        print(f"ID: {i['transaction_id']} | Type: {i['type']} | Amount:
${i['amount']:.2f} | Date: {i['date']} | Balance After: ${i['balance_after']:.2f}")

            # 4. back to main menu
            elif search_choice == 4:
                main()

        # 6. exit
        elif choice == 6:
            program_state = False

# run main program
main()
exit()
```

account.json

{"balance": 1038.0, "transactions": [{"transaction_id": "0000000001", "type": "deposit", "amount": 50.0, "date": "2025-12-07 22:51:28.043060", "balance_after": 50.0}, {"transaction_id": "0000000002", "type": "deposit", "amount": 50.0, "date": "2025-12-07 22:51:32.691213", "balance_after": 100.0}, {"transaction_id": "0000000003", "type": "withdraw", "amount": 99.0, "date": "2025-12-07 22:51:34.545436", "balance_after": 1.0}, {"transaction_id": "0000000004", "type": "deposit", "amount": 1230.0, "date": "2025-12-07 22:51:39.531519", "balance_after": 1231.0}, {"transaction_id": "0000000005", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:18.858317", "balance_after": 1232.0}, {"transaction_id": "0000000006", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:20.003650", "balance_after": 1233.0}, {"transaction_id": "0000000007", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:20.664279", "balance_after": 1234.0}, {"transaction_id": "0000000008", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:21.148810", "balance_after": 1235.0}, {"transaction_id": "0000000009", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:21.730856", "balance_after": 1236.0}, {"transaction_id": "0000000010", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:54:54.403182", "balance_after": 1237.0}, {"transaction_id": "0000000011", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:55:00.269094", "balance_after": 1238.0}, {"transaction_id": "0000000012", "type": "deposit", "amount": 1.0, "date": "2025-12-07 22:55:02.152792", "balance_after": 1239.0}, {"transaction_id": "0000000013", "type": "withdraw", "amount": 2.0, "date": "2025-12-07 22:55:40.729659", "balance_after": 1237.0}, {"transaction_id": "0000000014", "type": "withdraw", "amount": 99.0, "date": "2025-12-07 22:55:42.905709", "balance_after": 1138.0}, {"transaction_id": "0000000015", "type": "withdraw", "amount": 200.0, "date": "2025-12-07 22:55:48.474076", "balance_after": 938.0}, {"transaction_id": "0000000016", "type": "deposit", "amount": 100.0, "date": "2025-12-08 00:05:34.048016", "balance_after": 1038.0}]}

[Python] Connect 4

[Python] Connect 4

main.py

```
board=[[' ']*7,
       [' ']*7,
       [' ']*7,
       [' ']*7,
       [' ']*7,
       [' ']*7]
fin=False
valid=True
r=0

def display():
    global board
    row=' '
    for i in range(6):
        row='□'+board[i][0]
        for j in range(6):
            row=row+'□'+board[i][j+1]
        row=row+'□'
        print(row)

def check_horizontal():
    global board
    global fin
    for i in range(6):
        for j in range(4):
            if board[i][j]!=' ':
                if board[i][j]==board[i][j+1] and board[i][j+1]==board[i][j+2] and
board[i][j+2]==board[i][j+3]:
                    fin=True

def check_vertical():
    global board
    global fin
```

```
    for j in range(7):
        for i in range(3):
            if board[i][j]!=' ':
                if board[i][j]==board[i+1][j] and board[i+1][j]==board[i+2][j] and
board[i+2][j]==board[i+3][j]:
                    fin=True

def check_diagnal_right():
    global board
    global fin

    for i in range(3):
        for j in range(4):
            if board[i][j]!=' ':
                if board[i][j]==board[i+1][j+1] and board[i+1][j+1]==board[i+2][j+2] and
board[i+2][j+2]==board[i+3][j+3]:
                    fin=True

def check_diagnal_left():
    global board
    global fin

    for i in range(3):
        for j in range(4):
            if board[5-i][j]!=' ':
                if board[5-i][j]==board[4-i][j+1] and board[4-i][j+1]==board[3-i][j+2] and
board[3-i][j+2]==board[2-i][j+3]:
                    fin=True

def check_win():
    global board
    global fin
    check_horizontal()
    check_vertical()
    check_diagnal_right()
    check_diagnal_left()



def enter(x,lim):
    while True:
```

```
        try:
            x=int(input('type column(1~7):'))
            if x>=0 and x<=lim:
                x-=1
                break
            else:
                print('out of range')
        except ValueError:
            print('wrong format')
    return x

def put(x,n):
    global board
    global valid
    found=False
    i=0
    while not(found) and i<=5:
        if board[5-i][x]==' ':
            found=True
        else:
            i+=1

    if found:
        if n==1:
            board[5-i][x]='R'
        else:
            board[5-i][x]='Y'
        valid=True
    else:
        print('column already full')

while not(fin):

    display()
    check_win()
    if fin:
        print('player 2 win')
        break
    valid=False
    turn=1
```

```
print('player 1 turn (Red)')
while not(valid):
    put(enter(r,7),1)



display()
check_win()
if fin:
    print('player 1 win')
    break
valid=False
turn=2
print('player 2 turn (Yellow)')
while not(valid):
    put(enter(r,7),2)
```

[HTML] Simulation RPG Combat Sample

[HTML] Simulation RPG Combat Sample

Battle Sample

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<style>
*{box-sizing:border-box;margin:0;padding:0}
body{font-family:sans-serif;background:#1a1a2e;color:#eee;user-select:none;min-height:100vh}
#app{display:flex;flex-direction:column;align-items:center;padding:12px;gap:10px}
h1{font-size:18px;font-weight:500;color:#c9a96e;letter-spacing:2px}
#top{display:flex;gap:16px;align-items:flex-start;width:100%;max-width:720px}
#map-wrap{position:relative}
canvas{display:block;border:2px solid #444;border-radius:4px}
#sidebar{flex:1;min-width:180px;display:flex;flex-direction:column;gap:8px}
.panel{background:#16213e;border:1px solid #2a3a5e;border-radius:6px;padding:10px;font-
size:13px}
.panel h3{font-size:12px;color:#c9a96e;margin-bottom:6px;text-transform:uppercase;letter-
spacing:1px}
#unit-name{font-size:15px;font-weight:500;margin-bottom:4px}
.bar-row{display:flex;align-items:center;gap:6px;margin:3px 0}
.bar-label{width:28px;font-size:11px;color:#aaa}
.bar-bg{flex:1;height:8px;background:#2a2a3e;border-radius:4px;overflow:hidden}
.bar-fill{height:100%;border-radius:4px;transition:width .3s}
.hp-bar{background:#4caf50}
.mp-bar{background:#5599ff}
#log{height:120px;overflow-y:auto;font-size:12px;line-height:1.6;color:#bbb}
#log div{padding:1px 0;border-bottom:1px solid #1e2a40}
#bottom{display:flex;gap:8px;flex-wrap:wrap;justify-content:center}
button{padding:6px 14px;background:#1e3a5f;border:1px solid #3a5a8f;color:#c9d8f0;border-
radius:4px;cursor:pointer;font-size:13px;transition:background .15s}
button:hover{background:#2a4a7f}
button:disabled{opacity:.4;cursor:default}
button.danger{background:#5f1e1e;border-color:#8f3a3a;color:#f0c9c9}
button.danger:hover{background:#7f2a2a}
#phase{font-size:13px;color:#c9a96e;text-align:center}
```

```
#weapon-info{display:grid;grid-template-columns:1fr 1fr;gap:4px}
.wi{font-size:11px;padding:3px 5px;background:#0d1626;border-radius:3px;border-left:3px solid
#3a5a8f}
.wi b{color:#c9a96e}
</style>
</head>
<body>
<div id="app">
<h1>⚔ SRPG Simulator</h1>
<div id="top">
  <div id="map-wrap"><canvas id="c" width="400" height="400"></canvas></div>
  <div id="sidebar">
    <div class="panel" id="info-panel">
      <h3>Selected unit</h3>
      <div id="unit-name" style="color:#aaa">Select Unit</div>
      <div id="unit-stats"></div>
    </div>
    <div class="panel">
      <h3>Weapon Types</h3>
      <div id="weapon-info">
        <div class="wi"><b>□□Sword</b><br>Atk 15 Acc 90%<br>Range 1 Balanced</div>
        <div class="wi"><b>□□Bow</b><br>Atk 10 Acc 80%<br>Range 2 Distance</div>
        <div class="wi"><b>□□Spear</b><br>Atk 13 Acc 85%<br>Range 1 Counter+</div>
        <div class="wi"><b>□□Axe</b><br>Atk 20 Acc 65%<br>Range 1 Critical+</div>
      </div>
    </div>
    <div class="panel">
      <h3>Combat log</h3>
      <div id="log"></div>
    </div>
  </div>
</div>
<div id="phase">Turn 1 — Player turn</div>
<div id="bottom">
  <button id="btn-end" onclick="endPlayerTurn()">Turn End</button>
  <button id="btn-wait" onclick="waitUnit()" disabled>Wait</button>
  <button id="btn-restart" class="danger" onclick="initGame()">Restart</button>
</div>
</div>
<script>
```

```
const COLS=10,ROWS=10,TS=40;
const cv=document.getElementById('c');
cv.width=COLS*TS;cv.height=ROWS*TS;
const ctx=cv.getContext('2d');

const WEAPONS={
  sword:{name:'Sword',icon:'□',atk:15,hit:90,range:1,color:'#4fc3f7'},
  bow:  {name:'Bow',icon:'□',atk:10,hit:80,range:2,color:'#a5d6a7'},
  spear:{name:'Spear',icon:'□',atk:13,hit:85,range:1,color:'#ce93d8',ctrAtk:5},
  axe:  {name:'Axe',icon:'□',atk:20,hit:65,range:1,color:'#ffab91'},
};

const TERRAIN={
  plain:{name:'Yard',color:'#2d4a2d',def:0,move:1},
  forest:{name:'Bush',color:'#1a3a1a',def:2,move:2},
  hill:{name:'Hall',color:'#4a3a2a',def:1,move:2},
  water:{name:'Water',color:'#1a2a4a',def:0,move:999},
};

let MAP=[],units=[],sel=null,phase='player',turn=1,moveHighlight=[],attackHighlight=[];

function mkMap(){
  MAP=[];
  const layout=[
    'pppppppppf',
    'ppfppphhpp',
    'pffpppphpp',
    'ppppwwpppp',
    'ppphwwphpp',
    'pppphhhppp',
    'ppfpppppfp',
    'pppppppppf',
    'pfpphhpppf',
    'pppppppppp',
  ];
  const keys={p:'plain',f:'forest',h:'hill',w:'water'};
  for(let r=0;r<ROWS;r++){MAP[r]=[];for(let
c=0;c<COLS;c++)MAP[r][c]=keys[layout[r][c]]||'plain';}
}
```

```
function mkUnits(){
  units=[
    {id:0,name:'Sword
man',team:'player',weapon:'sword',hp:30,maxHp:30,atk:0,def:3,spd:5,x:0,y:8,moved:false,acted:f
alse,color:'#4fc3f7'},
    {id:1,name:'Spear
man',team:'player',weapon:'spear',hp:28,maxHp:28,atk:0,def:2,spd:4,x:1,y:9,moved:false,acted:f
alse,color:'#ce93d8'},

{id:2,name:'Archer',team:'player',weapon:'bow',hp:22,maxHp:22,atk:0,def:1,spd:6,x:0,y:7,moved:
false,acted:false,color:'#a5d6a7'},
    {id:3,name:'Axe
man',team:'player',weapon:'axe',hp:35,maxHp:35,atk:0,def:4,spd:3,x:2,y:9,moved:false,acted:fal
se,color:'#ffab91'},
    {id:4,name:'Sword
goblin',team:'enemy',weapon:'sword',hp:20,maxHp:20,atk:0,def:1,spd:4,x:9,y:1,moved:false,acted
:false,color:'#ef5350'},
    {id:5,name:'Axe
goblin',team:'enemy',weapon:'axe',hp:22,maxHp:22,atk:0,def:1,spd:3,x:8,y:0,moved:false,acted:f
alse,color:'#ef5350'},
    {id:6,name:'Spear
goblin',team:'enemy',weapon:'spear',hp:18,maxHp:18,atk:0,def:2,spd:5,x:9,y:2,moved:false,acted
:false,color:'#ff7043'},
    {id:7,name:'Bow
goblin',team:'enemy',weapon:'bow',hp:16,maxHp:16,atk:0,def:0,spd:6,x:7,y:0,moved:false,acted:f
alse,color:'#ef5350'},
  ];
}

function initGame(){
  mkMap();mkUnits();sel=null;phase='player';turn=1;moveHighlight=[];attackHighlight=[];
  document.getElementById('log').innerHTML='';
  setPhaseText();
  render();
  document.getElementById('btn-end').disabled=false;
}

function getUnit(x,y){return units.find(u=>u.x===x&&u.y===y&&u.hp>0);}

function inBounds(x,y){return x>=0&&y>=0&&x<COLS&&y<ROWS;}
```

```
function terrainMoveCost(x,y){return TERRAIN[MAP[y][x]].move;}

function getMoveCells(unit){
  const moveRange=3;
  const visited={};
  const q=[{x:unit.x,y:unit.y,cost:0}];
  visited[`${unit.x},${unit.y}`]=0;
  const cells=[];
  while(q.length){
    const cur=q.shift();
    cells.push({x:cur.x,y:cur.y});
    const dirs=[[1,0],[-1,0],[0,1],[0,-1]];
    for(const [dx,dy] of dirs){
      const nx=cur.x+dx,ny=cur.y+dy;
      const key=`${nx},${ny}`;
      if(!inBounds(nx,ny))continue;
      const cost=cur.cost+terrainMoveCost(nx,ny);
      if(cost>moveRange)continue;
      const occ=getUnit(nx,ny);
      if(occ&&occ.team!==unit.team)continue;

if(visited[key]===undefined||visited[key]>cost){visited[key]=cost;q.push({x:nx,y:ny,cost});}
    }
  }
  return cells.filter(c=>!(c.x===unit.x&&c.y===unit.y));
}

function getAttackCells(unit,fromX,fromY){
  const wep=WEAPONS[unit.weapon];
  const cells=[];
  for(let r=0;r<ROWS;r++)for(let c=0;c<COLS;c++){
    const dist=Math.abs(c-fromX)+Math.abs(r-fromY);
    if(dist>=1&&dist<=wep.range)cells.push({x:c,y:r});
  }
  return cells;
}

function dist(a,b){return Math.abs(a.x-b.x)+Math.abs(a.y-b.y);}
```

```
function calcDmg(attacker,defender){
  const wep=WEAPONS[attacker.weapon];
  const hit=Math.random()*100<wep.hit;
  if(!hit)return{dmg:0,hit:false};
  const dmg=Math.max(1,wep.atk+attacker.atk-
defender.def+TERRAIN[MAP[defender.y][defender.x]].def*-1);
  return{dmg,hit:true};
}

function doAttack(attacker,defender){
  const awep=WEAPONS[attacker.weapon];
  const dwep=WEAPONS[defender.weapon];
  const {dmg,hit}=calcDmg(attacker,defender);
  if(hit){defender.hp=Math.max(0,defender.hp-dmg);addLog(`${attacker.name}→${defender.name}
${dmg} Damage!`);}
  else addLog(`${attacker.name}→${defender.name} Miss!`);

  if(defender.hp>0){
    const defRange=dwep.range;
    const d=dist(attacker,defender);
    if(d<=defRange){
      const bonus=(defender.weapon==='spear'?dwep.ctrAtk||0:0);
      const r2=calcDmg({...defender,atk:defender.atk+bonus},attacker);
      if(r2.hit){attacker.hp=Math.max(0,attacker.hp-r2.dmg);addLog(`${defender.name}'s
Counter! ${r2.dmg} Damage!`);}
      else addLog(`${defender.name} Counter - Miss!`);
    }
  }
  if(defender.hp<=0)addLog(`☠ ${defender.name} was defeat!`);
  if(attacker.hp<=0)addLog(`☠ ${attacker.name} was defeat!`);
  checkWin();
}

function addLog(msg){
  const el=document.getElementById('log');
  const d=document.createElement('div');d.textContent=msg;
  el.appendChild(d);el.scrollTop=el.scrollHeight;
}

function checkWin(){
```

```
  const pAlive=units.filter(u=>u.team==='player'&&u.hp>0);
  const eAlive=units.filter(u=>u.team==='enemy'&&u.hp>0);
  if(pAlive.length===0){setTimeout(()=>{addLog('□□Enemies are victorious!');},100);}
  else if(eAlive.length===0){setTimeout(()=>{addLog('□□Player is victorious!');},100);}
}

function setPhaseText(){
  document.getElementById('phase').textContent=`Turn ${turn} — ${phase==='player'?'Player
turn':'Enemy turn'}`;
}

// Rendering
const COLORS={
  moveable:'rgba(80,180,255,0.25)',
  attackable:'rgba(255,80,80,0.28)',
  selected:'rgba(255,220,80,0.35)',
};

function render(){
  ctx.clearRect(0,0,cv.width,cv.height);
  // terrain
  for(let r=0;r<ROWS;r++)for(let c=0;c<COLS;c++){
    ctx.fillStyle=TERRAIN[MAP[r][c]].color;
    ctx.fillRect(c*TS,r*TS,TS,TS);
    ctx.strokeStyle='rgba(0,0,0,0.3)';ctx.lineWidth=0.5;
    ctx.strokeRect(c*TS,r*TS,TS,TS);
  }
  // highlights
  for(const h of moveHighlight){
    ctx.fillStyle=COLORS.moveable;ctx.fillRect(h.x*TS,h.y*TS,TS,TS);
  }
  for(const h of attackHighlight){
    ctx.fillStyle=COLORS.attackable;ctx.fillRect(h.x*TS,h.y*TS,TS,TS);
  }
  if(sel){
    ctx.fillStyle=COLORS.selected;ctx.fillRect(sel.x*TS,sel.y*TS,TS,TS);
  }
  // units
  for(const u of units){
    if(u.hp<=0)continue;
```

```
    const px=u.x*TS,py=u.y*TS;
    ctx.fillStyle=u.team==='player'?(u.moved?'#555':u.color):(u.color);
    ctx.beginPath();ctx.roundRect(px+5,py+5,TS-10,TS-10,4);ctx.fill();
    if(u===sel){ctx.strokeStyle='#ffe066';ctx.lineWidth=2;ctx.stroke();}
    const wep=WEAPONS[u.weapon];
    ctx.font='16px serif';ctx.textAlign='center';ctx.textBaseline='middle';
    ctx.fillText(wep.icon,px+TS/2,py+TS/2);
    // hp bar
    const barW=TS-8,barH=4,bx=px+4,by=py+TS-8;
    ctx.fillStyle='#222';ctx.fillRect(bx,by,barW,barH);
    ctx.fillStyle=u.team==='player'?'#4caf50':'#f44336';
    ctx.fillRect(bx,by,barW*(u.hp/u.maxHp),barH);
    // team indicator
    ctx.fillStyle=u.team==='player'?'#4fc3f7':'#ef5350';
    ctx.fillRect(px+4,py+4,8,3);
  }
}

// State machine
let state='idle'; // idle | selected | moved
let selMoves=[];
let selAttacks=[];
let movedPos=null;

cv.addEventListener('click',e=>{
  const rect=cv.getBoundingClientRect();
  const scaleX=cv.width/rect.width,scaleY=cv.height/rect.height;
  const mx=Math.floor((e.clientX-rect.left)*scaleX/TS);
  const my=Math.floor((e.clientY-rect.top)*scaleY/TS);
  if(!inBounds(mx,my))return;
  handleClick(mx,my);
});

function handleClick(cx,cy){
  if(phase!=='player')return;
  const clicked=getUnit(cx,cy);

  if(state==='idle'){
    if(clicked&&clicked.team==='player'&&!clicked.acted){
      sel=clicked;
```

```
      selMoves=getMoveCells(clicked);
      selAttacks=getAttackCells(clicked,clicked.x,clicked.y);
      moveHighlight=selMoves;
      attackHighlight=selAttacks.filter(a=>{const e=getUnit(a.x,a.y);return
e&&e.team==='enemy';});
      state='selected';
      showUnitInfo(clicked);
      document.getElementById('btn-wait').disabled=false;
    }else{
      clearSel();
    }
  } else if(state==='selected'){
    // Click same unit = deselect
    if(clicked===sel){clearSel();return;}
    // Click enemy in range = attack
    if(clicked&&clicked.team==='enemy'){
      const inRange=selAttacks.some(a=>a.x===cx&&a.y===cy);
      if(inRange){
        doAttack(sel,clicked);
        sel.acted=true;sel.moved=true;
        clearSel();render();return;
      }
    }
    // Click move cell
    if(selMoves.some(m=>m.x===cx&&m.y===cy)&&!getUnit(cx,cy)){
      movedPos={x:sel.x,y:sel.y};
      sel.x=cx;sel.y=cy;
      sel.moved=true;
      // Show attack options from new pos
      selAttacks=getAttackCells(sel,cx,cy);
      const enemies=selAttacks.filter(a=>{const e=getUnit(a.x,a.y);return
e&&e.team==='enemy';});
      moveHighlight=[];
      attackHighlight=enemies;
      state='moved';
      render();return;
    }
    // Click different ally
    if(clicked&&clicked.team==='player'&&!clicked.acted){
      sel=clicked;
```

```
      selMoves=getMoveCells(clicked);
      selAttacks=getAttackCells(clicked,clicked.x,clicked.y);
      moveHighlight=selMoves;
      attackHighlight=selAttacks.filter(a=>{const e=getUnit(a.x,a.y);return
e&&e.team==='enemy';});
      showUnitInfo(clicked);
      render();return;
    }
    clearSel();
  } else if(state==='moved'){
    // Click enemy to attack
    if(clicked&&clicked.team==='enemy'){
      const inRange=selAttacks.some(a=>a.x===cx&&a.y===cy);
      if(inRange){
        doAttack(sel,clicked);
        sel.acted=true;
        clearSel();render();return;
      }
    }
    // Click empty = undo move
    if(!clicked){
      if(movedPos){sel.x=movedPos.x;sel.y=movedPos.y;sel.moved=false;}
      selMoves=getMoveCells(sel);
      selAttacks=getAttackCells(sel,sel.x,sel.y);
      moveHighlight=selMoves;
      attackHighlight=selAttacks.filter(a=>{const e=getUnit(a.x,a.y);return
e&&e.team==='enemy';});
      state='selected';movedPos=null;render();return;
    }
  }
  render();
}

function clearSel(){
  sel=null;state='idle';moveHighlight=[];attackHighlight=[];movedPos=null;
  document.getElementById('btn-wait').disabled=true;
  document.getElementById('unit-name').textContent='Select Unit';
  document.getElementById('unit-name').style.color='#aaa';
  document.getElementById('unit-stats').innerHTML='';
}
```

```
function waitUnit(){
  if(!sel)return;
  sel.moved=true;sel.acted=true;
  clearSel();render();
}

function showUnitInfo(u){
  const wep=WEAPONS[u.weapon];
  document.getElementById('unit-name').textContent=`${wep.icon} ${u.name}`;
  document.getElementById('unit-name').style.color=u.color;
  document.getElementById('unit-stats').innerHTML=`
    <div class="bar-row"><span class="bar-label">HP</span><div class="bar-bg"><div class="bar-
fill hp-bar" style="width:${(u.hp/u.maxHp)*100}%"></div></div><span style="font-
size:11px;margin-left:4px">${u.hp}/${u.maxHp}</span></div>
    <div style="font-size:11px;color:#aaa;margin-top:4px">
      Weapon: ${wep.name} | Atk: ${wep.atk} | Acc: ${wep.hit}%<br>
      Range: ${wep.range} | Def: ${u.def} | Spd: ${u.spd}
    </div>
    <div style="font-size:11px;color:${u.moved?'#f66':'#4fc'};margin-
top:3px">${u.acted?'Acted':u.moved?'Moved':'Movable'}</div>
  `;
}

function endPlayerTurn(){
  clearSel();
  phase='enemy';
  setPhaseText();
  document.getElementById('btn-end').disabled=true;
  render();
  setTimeout(enemyTurn,600);
}

function enemyTurn(){
  const enemies=units.filter(u=>u.team==='enemy'&&u.hp>0);
  const players=units.filter(u=>u.team==='player'&&u.hp>0);
  let i=0;
  function doNext(){
    if(i>=enemies.length){
      // Reset
```

```
      units.forEach(u=>{u.moved=false;u.acted=false;});
      phase='player';turn++;
      setPhaseText();
      document.getElementById('btn-end').disabled=false;
      render();return;
    }
    const enemy=enemies[i];i++;
    if(enemy.hp<=0){doNext();return;}
    // Find nearest player
    const pAlive=units.filter(u=>u.team==='player'&&u.hp>0);
    if(!pAlive.length){render();return;}
    const target=pAlive.reduce((a,b)=>dist(enemy,a)<dist(enemy,b)?a:b);
    const wep=WEAPONS[enemy.weapon];
    const d=dist(enemy,target);
    if(d<=wep.range){
      // Attack
      doAttack(enemy,target);
    } else {
      // Move toward target
      const moves=getMoveCells(enemy);
      // pick cell closest to target
      let best=null,bestD=999;
      for(const m of moves){
        const md=Math.abs(m.x-target.x)+Math.abs(m.y-target.y);
        if(md<bestD&&!getUnit(m.x,m.y)){bestD=md;best=m;}
      }
      if(best){enemy.x=best.x;enemy.y=best.y;}
      // Try attack from new pos
      const atks=getAttackCells(enemy,enemy.x,enemy.y);
      const inRange=atks.find(a=>a.x===target.x&&a.y===target.y);
      if(inRange)doAttack(enemy,target);
    }
    enemy.moved=true;enemy.acted=true;
    render();
    setTimeout(doNext,500);
  }
  doNext();
}

initGame();
```

```
</script>
</body>
</html>
```

[Python] Chess

[Python] Chess

Version 1

Requires:
- Pygame
- Stockfish 18

Which can be downloaded by running:

```
pip install pygame
```

```
brew install stockfish
```

Code:

```
"""
Chess Game — Lichess-style UI  (v3 — all fixes applied)

Fixes:
 1. Move list: white & black on SAME ROW  (1. e4  e5)
 2. Piece rendering: platform-aware font fallback, letter-in-box if no Unicode glyphs
 3. Increment added to the player who JUST MOVED (not the opponent)
 4. Arrow keys navigate review in both directions
 5. History screen: confirm dialog before opening a saved-game review
"""

import pygame, sys, copy, random, time, json, os, platform, subprocess, threading, shutil
pygame.init()

# ── Layout ──────────────────────────────────────────────────────────────
BOARD_SIZE  = 640
PANEL_WIDTH = 300
WINDOW_W    = BOARD_SIZE + PANEL_WIDTH
WINDOW_H    = BOARD_SIZE
SQ          = BOARD_SIZE // 8

# ── Colours ─────────────────────────────────────────────────────────────
```

```
C_BG        = (22,  21,  18)
C_DARK_SQ   = (181, 136,  99)
C_LIGHT_SQ  = (240, 217, 181)
C_HIGHLIGHT = (205, 210, 106)
C_SELECTED  = (246, 246, 105)
C_PANEL     = (28,  27,  24)
C_PANEL2    = (38,  37,  33)
C_PANEL3    = (52,  50,  44)
C_TEXT      = (222, 220, 215)
C_TEXT2     = (140, 135, 125)
C_TEXT3     = (88,  85,  78)
C_ACCENT    = (128, 196, 127)
C_GOLD      = (255, 188,  66)
C_RED       = (210,  90,  90)
C_BLUE      = (100, 155, 220)
C_BTN       = (55,  53,  47)
C_BTN_H     = (75,  73,  65)
C_BTN_A     = (100, 155, 220)
C_CHECK     = (200,  55,  55)
C_BORDER    = (65,  62,  55)
C_SEP       = (50,  48,  43)

# ── Fonts ──────────────────────────────────────────────────────────────
FNT_XL      = pygame.font.SysFont("segoeui",  32, bold=True)
FNT_LG      = pygame.font.SysFont("segoeui",  22, bold=True)
FNT_MD      = pygame.font.SysFont("segoeui",  17)
FNT_MDB     = pygame.font.SysFont("segoeui",  17, bold=True)
FNT_SM      = pygame.font.SysFont("segoeui",  14)
FNT_SMB     = pygame.font.SysFont("segoeui",  14, bold=True)
FNT_XS      = pygame.font.SysFont("segoeui",  12)
FNT_MONO    = pygame.font.SysFont("consolas", 14)
FNT_MONO_SM = pygame.font.SysFont("consolas", 13)
FNT_CLK     = pygame.font.SysFont("consolas", 28, bold=True)

# ── Piece rendering  (FIX #2) ──────────────────────────────────────────
UNICODE_SYMS = {'K':'♔','Q':'♕','R':'♖','B':'♗','N':'♘','P':'♙',
                'k':'♚','q':'♛','r':'♜','b':'♝','n':'♞','p':'♟'}

def _make_piece_font(size):
    plat = platform.system()
```

```
    if plat == "Windows":
        cands = ["segoeuisymbol","seguisym","segoeui","arial unicode ms"]
    elif plat == "Darwin":
        cands = ["apple symbols","arial unicode ms","lucida grande"]
    else:
        cands = ["dejavusans","symbola","freesans","unifont"]
    cands += [None]
    for name in cands:
        try:
            f = pygame.font.SysFont(name, size) if name else pygame.font.Font(None, size)
            surf = f.render("♔", True, (255,255,255))
            if surf.get_width() > 4:
                return f, True
        except Exception:
            pass
    return pygame.font.SysFont("consolas", size, bold=True), False

_PF_CACHE = {}
def _pfont(size):
    if size not in _PF_CACHE:
        _PF_CACHE[size] = _make_piece_font(size)
    return _PF_CACHE[size]

def draw_piece_at(surf, piece, px, py, size=SQ):
    is_white = piece.isupper()
    fs = int(size * 0.74)
    font, use_uni = _pfont(fs)
    if not use_uni:
        pad = max(4, size//8)
        bg  = (245,230,200) if is_white else (55,50,45)
        fg  = (40,35,30)    if is_white else (240,230,215)
        pygame.draw.rect(surf, bg,
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         border_radius=max(2, size//10))
        pygame.draw.rect(surf, (0,0,0),
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         1, border_radius=max(2, size//10))
        t = font.render(piece.upper(), True, fg)
        surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))
        return
```

```
    sym = UNICODE_SYMS.get(piece, '?')
    oc  = (230,228,224) if not is_white else (28,26,22)
    for ox,oy in ((-1,0),(1,0),(0,-1),(0,1)):
        o = font.render(sym, True, oc)
        surf.blit(o, (px+size//2-o.get_width()//2+ox, py+size//2-o.get_height()//2+oy))
    clr = (255,255,255) if is_white else (22,20,18)
    t   = font.render(sym, True, clr)
    surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))

# ── Helpers ──────────────────────────────────────────────────────────────
WHITE = 'w'; BLACK = 'b'
SAVE_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "chess_history.json")

TIME_CONTROLS = [
    {"label":"5 min",  "name":"Blitz",     "base":300,  "inc":0 },
    {"label":"10 min", "name":"Blitz",     "base":600,  "inc":0 },
    {"label":"15+10",  "name":"Rapid",     "base":900,  "inc":10},
    {"label":"30 min", "name":"Rapid",     "base":1800, "inc":0 },
    {"label":"60 min", "name":"Classical", "base":3600, "inc":0 },
    {"label":"90+30",  "name":"Classical", "base":5400, "inc":30},
]

PIECE_VALUES = {'P':100,'N':320,'B':330,'R':500,'Q':900,'K':20000}
PST = {
'P':[ 0, 0, 0, 0, 0, 0, 0, 0,50,50,50,50,50,50,50,50,
     10,10,20,30,30,20,10,10, 5, 5,10,25,25,10, 5, 5,
      0, 0, 0,20,20, 0, 0, 0, 5,-5,-10,0,0,-10,-5, 5,
      5,10,10,-20,-20,10,10,5, 0, 0, 0, 0, 0, 0, 0, 0],
'N':[-50,-40,-30,-30,-30,-30,-40,-50,-40,-20,0,0,0,0,-20,-40,
     -30,0,10,15,15,10,0,-30,-30,5,15,20,20,15,5,-30,
     -30,0,15,20,20,15,0,-30,-30,5,10,15,15,10,5,-30,
     -40,-20,0,5,5,0,-20,-40,-50,-40,-30,-30,-30,-30,-40,-50],
'B':[-20,-10,-10,-10,-10,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,10,10,5,0,-10,-10,5,5,10,10,5,5,-10,
     -10,0,10,10,10,10,0,-10,-10,10,10,10,10,10,10,-10,
     -10,5,0,0,0,0,5,-10,-20,-10,-10,-10,-10,-10,-10,-20],
'R':[ 0,0,0,0,0,0,0,0, 5,10,10,10,10,10,10,5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5, 0,0,0,5,5,0,0,0],
```

```
'Q':[-20,-10,-10,-5,-5,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,5,5,5,0,-10,-5,0,5,5,5,5,0,-5,
      0,0,5,5,5,5,0,-5,-10,5,5,5,5,5,0,-10,
     -10,0,5,0,0,0,0,-10,-20,-10,-10,-5,-5,-10,-10,-20],
'K':[-30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -20,-30,-30,-40,-40,-30,-30,-20,-10,-20,-20,-20,-20,-20,-20,-10,
      20,20,0,0,0,0,20,20,20,30,10,0,0,10,30,20],
}

def rr(surf,color,rect,r=8,brd=0,bc=None):
    pygame.draw.rect(surf,color,rect,border_radius=r)
    if brd and bc: pygame.draw.rect(surf,bc,rect,brd,border_radius=r)

def tc(surf,txt,fnt,clr,cx,cy):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(cx-t.get_width()//2,cy-t.get_height()//2))

def tl(surf,txt,fnt,clr,x,y):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(x,y)); return t.get_width()

def fmt(secs):
    secs=max(0,int(secs)); m,s=divmod(secs,60); return f"{m}:{s:02d}"

# ═══════════════════════════════════════════════════════════════════════
# Chess Engine
# ═══════════════════════════════════════════════════════════════════════
class ChessBoard:
    INIT = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
    def __init__(self):
        self.board=[[None]*8 for _ in range(8)]; self.turn=WHITE
        self.castling={'K':True,'Q':True,'k':True,'q':True}
        self.ep_square=None; self.halfmove=0; self.fullmove=1
        self.history=[]; self.result=None; self.result_reason=''
        self._fen(self.INIT)

    def _fen(self,fen):
        p=fen.split()
        for r,row in enumerate(p[0].split('/')):
            c=0
            for ch in row:
```

```
                if ch.isdigit(): c+=int(ch)
                else: self.board[r][c]=ch; c+=1
        self.turn=WHITE if p[1]=='w' else BLACK
        cs=p[2]; self.castling={'K':'K' in cs,'Q':'Q' in cs,'k':'k' in cs,'q':'q' in cs}
        if p[3]!='-': self.ep_square=(8-int(p[3][1]),ord(p[3][0])-ord('a'))
        self.halfmove=int(p[4]); self.fullmove=int(p[5])

    def col(self,p): return WHITE if p and p.isupper() else (BLACK if p else None)
    def pt(self,p):  return p.upper() if p else None

    def pseudo(self,row,col):
        p=self.board[row][col]
        if not p: return []
        c=self.col(p); t=self.pt(p); mv=[]
        def ok(r,c2): return 0<=r<8 and 0<=c2<8
        def slide(dirs):
            for dr,dc in dirs:
                r,c2=row+dr,col+dc
                while ok(r,c2):
                    tgt=self.board[r][c2]
                    if tgt is None: mv.append((r,c2))
                    elif self.col(tgt)!=c: mv.append((r,c2)); break
                    else: break
                    r+=dr; c2+=dc
        def jump(ds):
            for dr,dc in ds:
                r,c2=row+dr,col+dc
                if ok(r,c2) and self.col(self.board[r][c2])!=c: mv.append((r,c2))
        if   t=='R': slide([(1,0),(-1,0),(0,1),(0,-1)])
        elif t=='B': slide([(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='Q': slide([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='N': jump([(2,1),(2,-1),(-2,1),(-2,-1),(1,2),(1,-2),(-1,2),(-1,-2)])
        elif t=='K':
            jump([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
            if c==WHITE and row==7 and col==4:
                if self.castling['K'] and not self.board[7][5] and not self.board[7][6]:
mv.append((7,6,'cK'))
                if self.castling['Q'] and not self.board[7][3] and not self.board[7][2] and
not self.board[7][1]: mv.append((7,2,'cQ'))
            elif c==BLACK and row==0 and col==4:
```

```
                if self.castling['k'] and not self.board[0][5] and not self.board[0][6]:
mv.append((0,6,'ck'))
                if self.castling['q'] and not self.board[0][3] and not self.board[0][2] and
not self.board[0][1]: mv.append((0,2,'cq'))
        elif t=='P':
            d=-1 if c==WHITE else 1; sr=6 if c==WHITE else 1; r2=row+d
            if ok(r2,col) and not self.board[r2][col]:
                mv.append((r2,col))
                if row==sr and not self.board[row+2*d][col]: mv.append((row+2*d,col))
            for dc in(-1,1):
                r2,c2=row+d,col+dc
                if ok(r2,c2):
                    tgt=self.board[r2][c2]
                    if tgt and self.col(tgt)!=c: mv.append((r2,c2))
                    elif self.ep_square==(r2,c2): mv.append((r2,c2,'ep'))
        return mv

    def _chk(self,color,board):
        king='K' if color==WHITE else 'k'
        kp=next(((r,c) for r in range(8) for c in range(8) if board[r][c]==king),None)
        if not kp: return True
        orig=self.board; self.board=board
        att=any((m[0],m[1])==kp for r in range(8) for c in range(8)
                if self.col(board[r][c]) is not None and self.col(board[r][c])!=color
                for m in self.pseudo(r,c))
        self.board=orig; return att

    def _apply(self,board,castling,ep,fr,fc,tr,tc,sp=None,promo='Q'):
        b=copy.deepcopy(board); p=b[fr][fc]; c=self.col(p); t2=p.upper() if p else None
        nc=dict(castling); ne=None
        if sp in('cK','cQ','ck','cq'):
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None
            rm={'cK':(7,7,7,5),'cQ':(7,0,7,3),'ck':(0,7,0,5),'cq':(0,0,0,3)}
            rr2,rc,rt,rtc2=rm[sp]; b[rt][rtc2]=b[rr2][rc]; b[rr2][rc]=None
        elif sp=='ep':
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None; b[fr][tc]=None
        else:
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None
        if t2=='P' and (tr==0 or tr==7): b[tr][tc]=promo if c==WHITE else promo.lower()
        if p=='K': nc['K']=nc['Q']=False
```

```
        if p=='k': nc['k']=nc['q']=False
        for sq,key in[((7,0),'Q'),((7,7),'K'),((0,0),'q'),((0,7),'k')]:
            if (fr,fc)==sq or (tr,tc)==sq: nc[key]=False
        if t2=='P' and abs(tr-fr)==2: ne=((fr+tr)//2,fc)
        return b,nc,ne

    def legal(self,row,col):
        p=self.board[row][col]
        if not p: return []
        c=self.col(p); res=[]
        for m in self.pseudo(row,col):
            tr,tc2=m[0],m[1]; sp=m[2] if len(m)>2 else None
            if sp in('cK','cQ','ck','cq'):
                if self._chk(c,self.board): continue
                mc=5 if tc2==6 else 3
                b2,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,row,mc)
                if self._chk(c,b2): continue
            nb,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,tr,tc2,sp)
            if not self._chk(c,nb): res.append(m)
        return res

    def all_legal(self,color=None):
        if color is None: color=self.turn
        return [(r,c)+m for r in range(8) for c in range(8)
                if self.col(self.board[r][c])==color for m in self.legal(r,c)]

    def is_check(self):      return self._chk(self.turn,self.board)
    def is_checkmate(self):  return not self.all_legal() and self.is_check()
    def is_stalemate(self):  return not self.all_legal() and not self.is_check()
    def is_insuf(self):
        ps=[(p.upper(),r,c) for r in range(8) for c in range(8)
            if (p:=self.board[r][c]) and p.upper()!='K']
        return len(ps)==0 or (len(ps)==1 and ps[0][0] in('N','B'))

    def _san(self,fr,fc,tr,tc,sp,promo,board):
        p=board[fr][fc];
        if not p: return ''
        pt=p.upper(); CL='abcdefgh'; RL='87654321'; dest=CL[tc]+RL[tr]
        if sp in('cK','ck'): return 'O-O'
        if sp in('cQ','cq'): return 'O-O-O'
```

```
        cap='x' if board[tr][tc] or sp=='ep' else ''
        if pt=='P':
            s=(CL[fc]+cap+dest) if cap else dest
            if tr==0 or tr==7: s+='='+promo
            return s
        return pt+cap+dest

    def fen(self):
        rows=[]
        for r in range(8):
            e=0; s2=''
            for c in range(8):
                p=self.board[r][c]
                if not p: e+=1
                else:
                    if e: s2+=str(e); e=0
                    s2+=p
            if e: s2+=str(e)
            rows.append(s2)
        f='/'.join(rows)+' '+('w' if self.turn==WHITE else 'b')+' '
        cs=''.join(k for k in('K','Q','k','q') if self.castling[k])
        f+=(cs or '-')+' '
        if self.ep_square: f+='abcdefgh'[self.ep_square[1]]+str(8-self.ep_square[0])
        else: f+='-'
        f+=f' {self.halfmove} {self.fullmove}'
        return f

    def make_move(self,fr,fc,tr,tc,sp=None,promo='Q'):
        p=self.board[fr][fc]; cap=self.board[tr][tc]
        if sp=='ep': cap=self.board[fr][tc]
        old_b=copy.deepcopy(self.board); san=self._san(fr,fc,tr,tc,sp,promo,old_b)
        nb,nc,ne=self._apply(self.board,self.castling,self.ep_square,fr,fc,tr,tc,sp,promo)

info={'from':(fr,fc),'to':(tr,tc),'piece':p,'captured':cap,'special':sp,'promo':promo,'san':sa
n}
        self.history.append({'move':info,'board':old_b,'castling':dict(self.castling),
                             'ep':self.ep_square,'hm':self.halfmove,'turn':self.turn})
        self.board=nb; self.castling=nc; self.ep_square=ne
        pt=p.upper() if p else None
        self.halfmove=0 if (pt=='P' or cap) else self.halfmove+1
```

```
        if self.turn==BLACK: self.fullmove+=1
        self.turn=BLACK if self.turn==WHITE else WHITE
        if   self.is_checkmate():  self.result=WHITE if self.turn==BLACK else BLACK;
self.result_reason='checkmate'
        elif self.is_stalemate():  self.result='draw'; self.result_reason='stalemate'
        elif self.is_insuf():      self.result='draw'; self.result_reason='insufficient
material'
        elif self.halfmove>=100:   self.result='draw'; self.result_reason='50-move rule'
        return info

# ═══════════════════════════════════════════════════════════════════════════
# Fallback Python AI  (used when Stockfish is unavailable)
# ═══════════════════════════════════════════════════════════════════════════
class AI:
    D={1:1,2:1,3:2,4:2,5:3,6:3,7:4,8:4,9:5,10:6,11:6}
    N={1:350,2:250,3:180,4:120,5:70,6:40,7:20,8:8,9:2,10:0,11:0}
    def __init__(self,lv=5): self.lv=lv
    def eval(self,cb):
        s=0
        for r in range(8):
            for c in range(8):
                p=cb.board[r][c]
                if not p: continue
                pt=p.upper(); v=PIECE_VALUES.get(pt,0); idx=r*8+c if p.isupper() else (7-
r)*8+c
                s+=(v+PST[pt][idx]) if p.isupper() else -(v+PST[pt][idx])
        # mobility removed for speed
        return s
    def _ord(self,cb,moves):
        def sc(m):
            fr,fc,tr,tc=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
            cap=cb.board[tr][tc] if sp!='ep' else cb.board[m[0]][tc]
            return PIECE_VALUES.get((cap or'').upper(),0)-
PIECE_VALUES.get(cb.board[fr][fc].upper(),0)//10
        return sorted(moves,key=sc,reverse=True)
    def _cl(self,cb):
        n=ChessBoard.__new__(ChessBoard)
        n.board=copy.deepcopy(cb.board); n.turn=cb.turn; n.castling=dict(cb.castling)
        n.ep_square=cb.ep_square; n.halfmove=cb.halfmove; n.fullmove=cb.fullmove
        n.history=[]; n.result=cb.result; n.result_reason=''; return n
```

```
    def _ab(self,cb,d,a,b,mx):
        if d==0 or cb.result: return self.eval(cb)
        ms=cb.all_legal()
        if not ms: return (-99999 if mx else 99999) if cb.is_check() else 0
        ms=self._ord(cb,ms)
        if mx:
            v=-999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=max(v,self._ab(c2,d-1,a,b,False)); a=max(a,v)
                if b<=a: break
            return v
        else:
            v=999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=min(v,self._ab(c2,d-1,a,b,True)); b=min(b,v)
                if b<=a: break
            return v
    def best(self,cb):
        depth=self.D.get(self.lv,3); noise=self.N.get(self.lv,0)
        ms=cb.all_legal()
        if not ms: return None
        ms=self._ord(cb,ms); mx=(cb.turn==WHITE); bv=-999999 if mx else 999999; bms=[]
        for m in ms:
            c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
            v=self._ab(c2,depth-1,-999999,999999,not mx)+random.randint(-noise,noise)
            if(mx and v>bv)or(not mx and v<bv): bv=v; bms=[m]
            elif v==bv: bms.append(m)
        return random.choice(bms) if bms else random.choice(ms)

# ═══════════════════════════════════════════════════════════════
# Stockfish AI  (async, non-blocking)
# ═══════════════════════════════════════════════════════════════
class StockfishAI:
    # Level → UCI_Elo
    LEVEL_ELO  = {1:500, 2:800, 3:1200, 4:1600, 5:1800,
                  6:2000, 7:2200, 8:2300, 9:2400, 10:2500, 11:2700}
    # Level → movetime (ms)
    LEVEL_TIME = {1:80, 2:100, 3:150, 4:200, 5:300,
```

```
              6:500, 7:800, 8:1200, 9:1800, 10:2500, 11:4000}

    def __init__(self, level=5):
        self.level   = level
        self._proc   = None
        self._thread = None
        self._result = None   # (fr,fc,tr,tc,sp,promo) when ready
        self._lock   = threading.Lock()
        self.ready   = False
        self._init()

    # ── process management ──────────────────────────────────────────
    def _find(self):
        # 1) same folder as this script
        base = os.path.dirname(os.path.abspath(__file__))
        for name in ("stockfish","stockfish.exe",
                     "stockfish-windows-x86-64-avx2.exe",
                     "stockfish-windows-x86-64-modern.exe"):
            p = os.path.join(base, name)
            if os.path.isfile(p): return p
        # 2) system PATH  (covers brew install on macOS)
        return shutil.which("stockfish")

    def _init(self):
        path = self._find()
        if not path:
            print("[Stockfish] Not found — using fallback Python AI.")
            return
        try:
            self._proc = subprocess.Popen(
                [path],
                stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                stderr=subprocess.DEVNULL, text=True, bufsize=1)
            self._send("uci");        self._wait("uciok")
            self._apply_level(self.level)
            self._send("isready");    self._wait("readyok")
            self.ready = True
            print(f"[Stockfish] Ready  level={self.level}  "
                  f"elo={self.LEVEL_ELO[self.level]}")
        except Exception as e:
```

```
        print(f"[Stockfish] Failed to start: {e}")
        self.ready = False

    def _send(self, cmd):
        if self._proc:
            self._proc.stdin.write(cmd + "\n")
            self._proc.stdin.flush()

    def _wait(self, token, timeout=10.0):
        deadline = time.time() + timeout
        while time.time() < deadline:
            line = self._proc.stdout.readline()
            if token in line: return line
        return ""

    def _apply_level(self, level):
        elo = self.LEVEL_ELO.get(level, 1750)
        self._send("setoption name UCI_LimitStrength value true")
        self._send(f"setoption name UCI_Elo value {elo}")

    def set_level(self, level):
        self.level = level
        if self.ready:
            self._apply_level(level)
            self._send("isready"); self._wait("readyok")

    def close(self):
        if self._proc:
            try: self._send("quit"); self._proc.terminate()
            except: pass
            self._proc = None
        self.ready = False

    # ── async move request ──────────────────────────────────────────
    def start_thinking(self, fen):
        if not self.ready: return
        with self._lock: self._result = None
        self._thread = threading.Thread(target=self._think, args=(fen,), daemon=True)
        self._thread.start()
```

```
    def _think(self, fen):
        mt = self.LEVEL_TIME.get(self.level, 500)
        self._send(f"position fen {fen}")
        self._send(f"go movetime {mt}")
        line = self._wait("bestmove", timeout=mt/1000 + 8)
        parts = line.strip().split()
        if len(parts) >= 2 and parts[0] == "bestmove" and parts[1] != "(none)":
            with self._lock:
                self._result = self._parse(parts[1])

    def is_thinking(self):
        return self._thread is not None and self._thread.is_alive()

    def get_result(self):
        """Returns move tuple or None if still thinking."""
        if self.is_thinking(): return None
        with self._lock:
            r = self._result; self._result = None
        return r

    # ── UCI string → internal move tuple ──────────────────────────────
    def _parse(self, uci):
        """
        'e2e4'  → (6,4,4,4, None, 'Q')
        'e7e8q' → (1,4,0,4, None, 'Q')   promotion
        'e1g1'  → (7,4,7,6, 'cK', 'Q')   castling
        """
        fc  = ord(uci[0]) - ord('a')
        fr  = 8 - int(uci[1])
        tc2 = ord(uci[2]) - ord('a')
        tr  = 8 - int(uci[3])
        promo = uci[4].upper() if len(uci) == 5 else 'Q'
        # Castling: king moves exactly 2 squares horizontally
        sp = None
        if   uci == 'e1g1': sp = 'cK'
        elif uci == 'e1c1': sp = 'cQ'
        elif uci == 'e8g8': sp = 'ck'
        elif uci == 'e8c8': sp = 'cq'
        return (fr, fc, tr, tc2, sp, promo)
```

```
# ═══════════════════════════════════════════════════════════════════════
# Persistence
# ═══════════════════════════════════════════════════════════════════════
def load_hist():
    try:
        with open(SAVE_FILE,'r') as f: return json.load(f)
    except: return []

def save_hist(recs):
    try:
        with open(SAVE_FILE,'w') as f: json.dump(recs[-10:],f,indent=2)
    except: pass

def record_game(cb,mode,ai_lv,tc_label,wt,bt):
    recs=load_hist()
    rs="White wins" if cb.result==WHITE else ("Black wins" if cb.result==BLACK else "Draw")
    recs.append({'date':time.strftime('%Y-%m-%d %H:%M'),'mode':mode,'ai_level':ai_lv,
                 'tc':tc_label,'result':rs,'reason':cb.result_reason,
                 'moves':[h['move']['san'] for h in cb.history],
                 'white_time_left':round(wt,1),'black_time_left':round(bt,1),
                 'total_moves':len(cb.history)})
    save_hist(recs)

# ═══════════════════════════════════════════════════════════════════════
# Screen IDs
# ═══════════════════════════════════════════════════════════════════════
class S:
    MAIN=0; SETUP_BOT=1; SETUP_LOC=2; PLAYING=3; PROMOTION=4
    REVIEW=5; HISTORY=6; HIST_CONFIRM=7

# ═══════════════════════════════════════════════════════════════════════
# Game
# ═══════════════════════════════════════════════════════════════════════
class Game:
    def __init__(self):
        self.surf=pygame.display.set_mode((WINDOW_W,WINDOW_H))
        pygame.display.set_caption("Chess")
        self.clk=pygame.time.Clock()
        self.state=S.MAIN
        # setup
```

```
    self.opt_ai=5; self.opt_tc=0; self.opt_col=WHITE
    # game
    self.cb=None; self.ai=StockfishAI(level=5); self.mode=None
    self.player_col=WHITE; self.flipped=False
    # clocks
    self.wt=0.0; self.bt=0.0; self.inc=0
    self.clk_last=0.0; self.clk_run=False
    # board UI
    self.sel=None; self.ltgts=[]; self.lm=None
    self.drag_p=None; self.drag_pos=None; self.drag_fr=None
    # promo
    self.promo=None
    # ai
    self.ai_busy=False
    # review
    self.rev_hist=[]; self.rev_idx=0; self.rev_boards=[]
    self.rev_src=S.PLAYING  # where Back goes
    # history
    self.hist_recs=[]; self.hist_scroll=0; self.hist_confirm=None
    # notif
    self.notif=''; self.notif_exp=0

# ── coords ──────────────────────────────────────────────────────────
def s2p(self,r,c):
    return ((7-c)*SQ,(7-r)*SQ) if self.flipped else (c*SQ,r*SQ)
def p2s(self,x,y):
    if not(0<=x<BOARD_SIZE and 0<=y<BOARD_SIZE): return None,None
    return (7-y//SQ,7-x//SQ) if self.flipped else (y//SQ,x//SQ)

# ── clock ───────────────────────────────────────────────────────────
def tick(self):
    if not self.clk_run or not self.cb or self.cb.result: return
    now=time.time(); dt=now-self.clk_last; self.clk_last=now
    if self.cb.turn==WHITE: self.wt=max(0,self.wt-dt)
    else:                   self.bt=max(0,self.bt-dt)
    if self.wt<=0 or self.bt<=0:
        self.cb.result=BLACK if self.wt<=0 else WHITE
        self.cb.result_reason='timeout'
        self.clk_run=False; self._end()
```

```
    # FIX #3 — increment goes to the MOVER
    def _add_inc(self,mover):
        if self.inc<=0: return
        if mover==WHITE: self.wt+=self.inc
        else:            self.bt+=self.inc

    # ── lifecycle ──────────────────────────────────────────────────────
    def start(self,mode):
        self.mode=mode; tc=TIME_CONTROLS[self.opt_tc]
        self.wt=float(tc['base']); self.bt=float(tc['base']); self.inc=tc['inc']
        self.cb=ChessBoard()
        # Close previous Stockfish process before creating a new one
        if isinstance(self.ai, StockfishAI): self.ai.close()
        self.ai=StockfishAI(level=self.opt_ai)
        # If Stockfish unavailable, fall back to Python AI
        if not self.ai.ready: self.ai=AI(self.opt_ai)
        self.sel=None; self.ltgts=[]; self.lm=None
        self.drag_p=None; self.promo=None; self.ai_busy=False
        self.clk_run=False; self.clk_last=time.time()
        if mode=='bot':
            self.player_col=self.opt_col; self.flipped=(self.opt_col==BLACK)
        else:
            self.player_col=WHITE; self.flipped=False
        self.state=S.PLAYING

    def _end(self):
        record_game(self.cb,self.mode,self.opt_ai,
                    TIME_CONTROLS[self.opt_tc]['label'],self.wt,self.bt)

    def open_review(self,history,src=S.PLAYING):
        self.rev_hist=history; self.rev_src=src
        self._build_rev()
        self.rev_idx=len(self.rev_hist); self.state=S.REVIEW

    def open_review_game(self):
        if self.cb and self.cb.history: self.open_review(self.cb.history[:],S.PLAYING)

    def _build_rev(self):
        self.rev_boards=[]
        tmp=ChessBoard(); self.rev_boards.append(copy.deepcopy(tmp.board))
```

```
        for h in self.rev_hist:
            m=h['move']

tmp.make_move(m['from'][0],m['from'][1],m['to'][0],m['to'][1],m['special'],m['promo'])
            self.rev_boards.append(copy.deepcopy(tmp.board))

    def open_hist(self):
        self.hist_recs=load_hist(); self.hist_scroll=0; self.hist_confirm=None
        self.state=S.HISTORY

    # ── move execution ──────────────────────────────────────────────────
    def try_move(self,fr,fc,tr,tc2):
        legal=self.cb.legal(fr,fc)
        m=next((x for x in legal if x[0]==tr and x[1]==tc2),None)
        if m is None:
            p=self.cb.board[tr][tc2]
            if p and self.cb.col(p)==self.cb.turn:
                self.sel=(tr,tc2); self.ltgts=self.cb.legal(tr,tc2)
            else:
                self.sel=None; self.ltgts=[]
            return
        sp=m[2] if len(m)>2 else None
        if self.cb.pt(self.cb.board[fr][fc])=='P' and (tr==0 or tr==7):
            self.promo=(fr,fc,tr,tc2,sp); self.state=S.PROMOTION
            self.sel=None; self.ltgts=[]; return
        self._do(fr,fc,tr,tc2,sp,'Q')

    def _do(self,fr,fc,tr,tc2,sp,promo):
        mover=self.cb.turn
        info=self.cb.make_move(fr,fc,tr,tc2,sp,promo)
        self._add_inc(mover)                    # FIX #3
        self.lm=((fr,fc),(tr,tc2))
        self.sel=None; self.ltgts=[]
        self.clk_last=time.time()
        if not self.clk_run and len(self.cb.history)>=1: self.clk_run=True
        if self.cb.result: self.clk_run=False; self._end()
        self.notif=info['san']; self.notif_exp=time.time()+2.2

    def ai_tick(self):
        if self.state!=S.PLAYING or self.mode!='bot' or self.cb.result: return
```

```
        if self.cb.turn==self.player_col: return

        if isinstance(self.ai, StockfishAI):
            if not self.ai_busy:
                self.ai.start_thinking(self.cb.fen())
                self.ai_busy=True
            else:
                result=self.ai.get_result()
                if result is not None:
                    fr,fc,tr,tc2,sp,promo=result
                    # Re-detect castling from board if not already flagged
                    # (Stockfish sends e1g1 etc. which _parse() already handles)
                    # Validate the move is legal before applying
                    legal=self.cb.legal(fr,fc)
                    matched=next((m for m in legal if m[0]==tr and m[1]==tc2),None)
                    if matched is not None:
                        sp2=matched[2] if len(matched)>2 else sp
                        self._do(fr,fc,tr,tc2,sp2,promo)
                    self.ai_busy=False
        else:
            # Fallback Python AI (blocking but keeps working)
            if self.ai_busy: return
            self.ai_busy=True
            m=self.ai.best(self.cb); self.ai_busy=False
            if m: self._do(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None,'Q')

    # ─── find move by san for record replay ──────────────────────────────
    def _find_san(self,cb,san):
        for m in cb.all_legal():
            fr,fc,tr,tc2=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
            for pr in('Q','R','B','N'):
                if '=' in san and san[-1]!=pr: continue
                if cb._san(fr,fc,tr,tc2,sp,pr,cb.board)==san: return m
        return None

    def _open_rec_review(self,idx):
        recs=self.hist_recs
        if not recs or idx>=len(recs): return
        rec=recs[-(idx+1)]
        tmp=ChessBoard()
```

```
        for san in rec.get('moves',[]):
            m=self._find_san(tmp,san)
            if m is None: break
            tmp.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
        self.open_review(tmp.history[:],S.HISTORY)

    # ── event loop ──────────────────────────────────────────────────────
    def run(self):
        while True:
            self.clk.tick(60); self.tick()
            for ev in pygame.event.get():
                if ev.type==pygame.QUIT:
                    if isinstance(self.ai, StockfishAI): self.ai.close()
                    pygame.quit(); sys.exit()
                if ev.type==pygame.KEYDOWN:         self._key(ev.key)
                if ev.type==pygame.MOUSEBUTTONDOWN: self._click(ev.pos,ev.button)
                if ev.type==pygame.MOUSEBUTTONUP:   self._rel(ev.pos)
                if ev.type==pygame.MOUSEMOTION:
                    if self.drag_p: self.drag_pos=ev.pos
            if self.state==S.PLAYING: self.ai_tick()
            self._draw()

    # ── key ─────────────────────────────────────────────────────────────
    def _key(self,key):
        if self.state==S.REVIEW:                                  # FIX #4
            if key in(pygame.K_LEFT,pygame.K_a):  self.rev_idx=max(0,self.rev_idx-1)
            if key in(pygame.K_RIGHT,pygame.K_d):
self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
            if key==pygame.K_HOME: self.rev_idx=0
            if key==pygame.K_END:  self.rev_idx=len(self.rev_hist)
            if key==pygame.K_ESCAPE: self.state=self.rev_src
        elif self.state==S.PLAYING:
            if key==pygame.K_ESCAPE: self.state=S.MAIN
            if key==pygame.K_f:      self.flipped=not self.flipped
            if key==pygame.K_r:      self.open_review_game()
        elif self.state==S.HIST_CONFIRM:
            if key==pygame.K_ESCAPE: self.state=S.HISTORY
        elif key==pygame.K_ESCAPE: self.state=S.MAIN

    # ── click dispatcher ────────────────────────────────────────────────
```

```
def _click(self,pos,btn):
    {S.MAIN:self._c_main,S.SETUP_BOT:self._c_sbot,S.SETUP_LOC:self._c_sloc,
     S.PLAYING:self._c_play,S.PROMOTION:self._c_promo,S.REVIEW:self._c_rev,
     S.HISTORY:self._c_hist,S.HIST_CONFIRM:self._c_hconf
    }.get(self.state,lambda p:None)(pos)

def _rel(self,pos):
    if self.state not in(S.PLAYING,S.PROMOTION) or not self.drag_p: return
    x,y=pos; r,c=self.p2s(x,y)
    if r is not None and self.drag_fr:
        fr,fc=self.drag_fr
        if(fr,fc)!=(r,c): self.try_move(fr,fc,r,c)
    self.drag_p=None; self.drag_pos=None; self.drag_fr=None

def _c_main(self,pos):
    x,y=pos; cx=WINDOW_W//2
    if cx-160<=x<=cx-10  and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_BOT
    elif cx+10<=x<=cx+160 and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_LOC
    elif cx-100<=x<=cx+100 and WINDOW_H//2+60<=y<=WINDOW_H//2+110: self.open_hist()

def _c_sbot(self,pos):
    x,y=pos; cx=WINDOW_W//2
    # 11 difficulty buttons — same layout as _d_sbot
    bw,bh=78,48; row1=6; row2=5
    for i in range(11):
        if i<row1:
            total_w=row1*bw+(row1-1)*6
            bx=cx-total_w//2+i*(bw+6); by=222
        else:
            j=i-row1
            total_w=row2*bw+(row2-1)*6
            bx=cx-total_w//2+j*(bw+6); by=278
        if bx<=x<=bx+bw and by<=y<=by+bh: self.opt_ai=i+1
    # Play as buttons
    if cx-160<=x<=cx-20 and 362<=y<=402:  self.opt_col=WHITE
    elif cx+20<=x<=cx+160 and 362<=y<=402: self.opt_col=BLACK
    # Time controls
    for i in range(len(TIME_CONTROLS)):
        bx=cx-210+(i%3)*142; by=440+(i//3)*52
        if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
```

```
        # Start
        if cx-100<=x<=cx+100 and 590<=y<=630: self.start('bot')
        # Back
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_sloc(self,pos):
        x,y=pos; cx=WINDOW_W//2
        for i in range(len(TIME_CONTROLS)):
            bx=cx-210+(i%3)*142; by=262+(i//3)*52
            if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
        if cx-100<=x<=cx+100 and 448<=y<=488: self.start('local')
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_play(self,pos):
        x,y=pos
        if x>=BOARD_SIZE: self._c_panel(pos); return
        if self.cb.result: return
        r,c=self.p2s(x,y)
        if r is None: return
        if self.mode=='bot' and self.cb.turn!=self.player_col: return
        p=self.cb.board[r][c]
        if p and self.cb.col(p)==self.cb.turn:
            self.sel=(r,c); self.ltgts=self.cb.legal(r,c)
            self.drag_p=p; self.drag_pos=pos; self.drag_fr=(r,c)
        elif self.sel:
            self.try_move(self.sel[0],self.sel[1],r,c)

    def _c_panel(self,pos):
        x,y=pos; px=x-BOARD_SIZE
        if 558<=y<=588:
            if  10<=px<=95:  self.open_review_game()
            elif 105<=px<=190: self.start(self.mode)
            elif 200<=px<=285: self.state=S.MAIN

    def _c_promo(self,pos):
        x,y=pos; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2
        for i,p in enumerate(['Q','R','B','N']):
            bx=cx-105+i*54; by=cy-22
            if bx<=x<=bx+52 and by<=y<=by+50:
                fr,fc,tr,tc2,sp=self.promo
```

```
            self._do(fr,fc,tr,tc2,sp,p)
            self.promo=None; self.state=S.PLAYING; return

    def _c_rev(self,pos):
        x,y=pos; cx=BOARD_SIZE//2; by0=BOARD_SIZE-46
        for bx,by,bw,bh,act in[(cx-132,by0,40,36,'first'),(cx-84,by0,40,36,'prev'),
                               (cx+44, by0,40,36,'next'),(cx+92,by0,40,36,'last')]:
            if bx<=x<=bx+bw and by<=y<=by+bh:
                if act=='first': self.rev_idx=0
                elif act=='prev': self.rev_idx=max(0,self.rev_idx-1)
                elif act=='next': self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
                elif act=='last': self.rev_idx=len(self.rev_hist)
                return
        if x>=BOARD_SIZE:
            px=x-BOARD_SIZE
            if 558<=y<=588 and 10<=px<=PANEL_WIDTH-10:
                self.state=self.rev_src

    def _c_hist(self,pos):
        x,y=pos; cx=WINDOW_W//2
        if WINDOW_H-55<=y<=WINDOW_H-20 and cx-80<=x<=cx+80:
            self.state=S.MAIN; return
        item_h=70; list_y0=88
        if 100<=y<=WINDOW_H-80 and 30<=x<=WINDOW_W-30:
            idx=(y-list_y0)//item_h+self.hist_scroll
            if 0<=idx<len(self.hist_recs):
                self.hist_confirm=idx; self.state=S.HIST_CONFIRM  # FIX #5

    def _c_hconf(self,pos):
        x,y=pos; cx,cy=WINDOW_W//2,WINDOW_H//2
        if cx-120<=x<=cx-10 and cy+20<=y<=cy+65:  self._open_rec_review(self.hist_confirm)
        elif cx+10<=x<=cx+120 and cy+20<=y<=cy+65: self.state=S.HISTORY

    # ═══════════════════════════════════════════════════════════════════
    # Drawing
    # ═══════════════════════════════════════════════════════════════════
    def _draw(self):
        s=self.surf; s.fill(C_BG)
        if   self.state==S.MAIN:      self._d_main()
        elif self.state==S.SETUP_BOT: self._d_sbot()
```

```
        elif self.state==S.SETUP_LOC: self._d_sloc()
        elif self.state in(S.PLAYING,S.PROMOTION):
            self._d_play()
            if self.state==S.PROMOTION: self._d_promo()
        elif self.state==S.REVIEW:    self._d_rev()
        elif self.state==S.HISTORY:   self._d_hist()
        elif self.state==S.HIST_CONFIRM: self._d_hist(); self._d_hconf()
        pygame.display.flip()

    # ── Main menu ──────────────────────────────────────────────────────────
    def _d_main(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        for r in range(8):
            for c in range(8):
                clr=(50,46,40) if(r+c)%2==0 else(38,35,30)

pygame.draw.rect(s,clr,(c*(WINDOW_W//8),r*(WINDOW_H//8),WINDOW_W//8,WINDOW_H//8))
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((14,13,11,218));
s.blit(ov,(0,0))
        tc(s,"CHESS",FNT_XL,(235,210,150),cx,cy-155)
        tc(s,"Full Rules  ·  10 AI Levels  ·  Time Controls",FNT_SM,C_TEXT2,cx,cy-112)
        pygame.draw.line(s,C_BORDER,(cx-230,cy-92),(cx+230,cy-92),1)
        for bx,by,bw,bh,lbl,clr in[(cx-160,cy-30,150,60,"vs Bot",C_BLUE),(cx+10,cy-
30,150,60,"Local 2P",C_ACCENT)]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            c2=tuple(min(255,v+22) for v in clr) if hov else clr
            rr(s,c2,(bx,by,bw,bh),10,1,C_BORDER)
            tc(s,lbl,FNT_MDB,(10,10,10),bx+bw//2,by+bh//2)
        hbx,hby,hbw,hbh=cx-100,cy+60,200,50
        hov=(hbx<=mx<=hbx+hbw and hby<=my<=hby+hbh)
        rr(s,C_BTN_H if hov else C_BTN,(hbx,hby,hbw,hbh),8,1,C_BORDER)
        tc(s,"Game Records",FNT_MD,C_TEXT,hbx+hbw//2,hby+hbh//2)
        tc(s,"F: flip  |  R: review  |  ESC: menu",FNT_XS,C_TEXT3,cx,WINDOW_H-18)

    # ── Setup screens ──────────────────────────────────────────────────────
    def _d_sbot(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Play vs Bot",FNT_LG,C_TEXT,cx,42)
        rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
        tc(s,"AI Difficulty",FNT_MD,C_TEXT2,cx,196)
```

```
    names=["Beginner","Novice","Amateur","Casual","Intermediate",
           "Advanced","Expert","Master","IM","GM","Super GM"]
    elos =[500,800,1200,1600,1800,2000,2200,2300,2400,2500,2700]
    # 11 buttons: first row 6, second row 5 — centred
    row1=6; row2=5
    bw,bh=78,48
    for i in range(11):
        if i<row1:
            total_w=row1*bw+(row1-1)*6
            bx=cx-total_w//2+i*(bw+6); by=222
        else:
            j=i-row1
            total_w=row2*bw+(row2-1)*6
            bx=cx-total_w//2+j*(bw+6); by=278
        sel=(self.opt_ai==i+1)
        c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw and by<=my<=by+bh) else C_BTN)
        rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
        tc(s,str(i+1),FNT_SMB,(255,255,255) if sel else C_TEXT2,bx+bw//2,by+12)
        tc(s,names[i],FNT_XS,(255,255,255) if sel else C_TEXT3,bx+bw//2,by+27)
        tc(s,str(elos[i]),FNT_XS,C_GOLD if sel else C_TEXT3,bx+bw//2,by+39)
    tc(s,"Play as",FNT_MD,C_TEXT2,cx,348)
    for lbl,col2,bx in[("White",WHITE,cx-160),("Black",BLACK,cx+20)]:
        bw2,bh2=140,40; by=362; sel=(self.opt_col==col2)
        c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw2 and by<=my<=by+bh2) else C_BTN)
        rr(s,c,(bx,by,bw2,bh2),7,1,C_BORDER)
        tc(s,lbl,FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw2//2,by+bh2//2)
    tc(s,"Time Control",FNT_MD,C_TEXT2,cx,420)
    self._d_tc(s,cx,mx,my,440)
    sbx,sby,sbw,sbh=cx-100,590,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
    rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
    tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

def _d_sloc(self):
    s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
    tc(s,"Local 2-Player",FNT_LG,C_TEXT,cx,42)
    rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
    tc(s,"Time Control",FNT_MD,C_TEXT2,cx,238)
    self._d_tc(s,cx,mx,my,258)
    sbx,sby,sbw,sbh=cx-100,448,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
    rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
```

```
        tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

    def _d_tc(self,s,cx,mx,my,y0):
        for i,t2 in enumerate(TIME_CONTROLS):
            bx=cx-210+(i%3)*142; by=y0+(i//3)*52; bw,bh=132,42; sel=(self.opt_tc==i)
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            c=C_BTN_A if sel else(C_BTN_H if hov else C_BTN)
            rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
            tc(s,t2['label'],FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw//2,by+14)
            tc(s,t2['name'], FNT_XS, (210,210,210) if sel else C_TEXT3,bx+bw//2,by+30)

    # ── Playing ──────────────────────────────────────────────────────────
    def _d_play(self):
        self._d_board(self.surf)
        self._d_pieces(self.surf,self.cb.board if self.cb else None)
        self._d_panel(self.surf)
        self._d_drag(self.surf)
        if self.cb and self.cb.result: self._d_result(self.surf)

    def _d_board(self,s,lm=None,sel=None,tgts=None):
        fl=self.flipped
        lm  =lm   if lm   is not None else self.lm
        sel =sel  if sel  is not None else self.sel
        tgts=tgts if tgts is not None else self.ltgts
        chk_sq=None
        if self.cb and self.cb.is_check() and not self.cb.result:
            king='K' if self.cb.turn==WHITE else 'k'
            for r in range(8):
                for c in range(8):
                    if self.cb.board[r][c]==king: chk_sq=(r,c)
        for r in range(8):
            for c in range(8):
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                base=C_LIGHT_SQ if(r+c)%2==0 else C_DARK_SQ
                pygame.draw.rect(s,base,(px,py,SQ,SQ))
                if lm and((r,c)==lm[0] or(r,c)==lm[1]):
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_HIGHLIGHT,170));
s.blit(hl,(px,py))
                if sel and(r,c)==sel:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA);
```

```
hl.fill((*C_SELECTED[:3],210)); s.blit(hl,(px,py))
                if chk_sq and(r,c)==chk_sq:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_CHECK,185));
s.blit(hl,(px,py))
        cb_b=self.cb.board if self.cb else [[None]*8 for _ in range(8)]
        for m in tgts:
            tr2,tc2=m[0],m[1]; px=(7-tc2)*SQ if fl else tc2*SQ; py=(7-tr2)*SQ if fl else
tr2*SQ
            hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA)
            if cb_b[tr2][tc2]: pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//2-2,5)
            else:               pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//6)
            s.blit(hl,(px,py))
        for i in range(8):
            ci=7-i if fl else i; ri=i if fl else 7-i
            lc=C_DARK_SQ if(7+i)%2==0 else C_LIGHT_SQ
            lr=C_DARK_SQ if i%2==1    else C_LIGHT_SQ
            lt=FNT_XS.render('abcdefgh'[ci],True,lc); s.blit(lt,(i*SQ+SQ-lt.get_width()-
3,BOARD_SIZE-lt.get_height()-2))
            ln=FNT_XS.render(str(ri+1),True,lr);      s.blit(ln,(3,i*SQ+2))

    def _d_pieces(self,s,board,fl=None):
        if board is None: return
        if fl is None: fl=self.flipped
        df=self.drag_fr
        for r in range(8):
            for c in range(8):
                if df and(r,c)==df: continue
                p=board[r][c]
                if not p: continue
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                draw_piece_at(s,p,px,py)

    def _d_drag(self,s):
        if self.drag_p and self.drag_pos:
            x,y=self.drag_pos; draw_piece_at(s,self.drag_p,x-SQ//2,y-SQ//2)

    def _d_result(self,s):
        ov=pygame.Surface((BOARD_SIZE,78),pygame.SRCALPHA); ov.fill((18,17,15,215))
        s.blit(ov,(0,BOARD_SIZE//2-39))
        r=self.cb.result
```

```
        msg,clr=("White wins",C_ACCENT) if r==WHITE else("Black wins",C_RED) if r==BLACK
else("Draw",C_TEXT2)
        tc(s,msg,FNT_LG,clr,BOARD_SIZE//2,BOARD_SIZE//2-11)
        tc(s,self.cb.result_reason.capitalize(),FNT_SM,C_TEXT2,BOARD_SIZE//2,BOARD_SIZE//2+16)

    # ── helpers for captured pieces ──────────────────────────────────────
    def _captured(self):
        """Return (white_captured, black_captured, advantage_color, adv_pts).
        white_captured = pieces white has taken (i.e. black pieces removed from board).
        black_captured = pieces black has taken (i.e. white pieces removed from board)."""
        PV = {'P':1,'N':3,'B':3,'R':5,'Q':9}
        start = {'P':8,'N':2,'B':2,'R':2,'Q':1}
        on_board = {}
        for r in range(8):
            for c in range(8):
                p = self.cb.board[r][c]
                if p: on_board[p] = on_board.get(p,0)+1
        # pieces white captured = missing black pieces
        w_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt.lower(),0)
            w_cap.extend([pt]*missing)
        # pieces black captured = missing white pieces
        b_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt,0)
            b_cap.extend([pt]*missing)
        ws = sum(PV.get(p,0) for p in w_cap)
        bs = sum(PV.get(p,0) for p in b_cap)
        adv = ws-bs
        return w_cap, b_cap, adv

    def _draw_caps(self, s, caps, score_adv, x, y, row_w):
        """Draw captured piece symbols in a compact row."""
        PV={'P':1,'N':3,'B':3,'R':5,'Q':9}
        # sort by value
        caps_sorted = sorted(caps, key=lambda p: PV.get(p,0))
        font, use_uni = _pfont(14)
        cx2 = x
        for p in caps_sorted:
```

```
            sym = UNICODE_SYMS.get(p.lower(),'?') if use_uni else p
            t = font.render(sym, True, C_TEXT2)
            s.blit(t,(cx2, y)); cx2 += t.get_width()+1
            if cx2 > x+row_w-20: break   # don't overflow
        if score_adv > 0:
            adv_t = FNT_XS.render(f"+{score_adv}", True, C_ACCENT)
            s.blit(adv_t,(cx2+4, y+1))

    # ── Panel ──────────────────────────────────────────────────────────────
    def _d_panel(self,s):
        px0=BOARD_SIZE; PW=PANEL_WIDTH; W=PW-28; x=px0+14; mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,PW,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        y=10

        # ── Header: mode + ELO ──────────────────────────────────────────────
        t2=TIME_CONTROLS[self.opt_tc]
        if self.mode=='bot':
            elo = StockfishAI.LEVEL_ELO.get(self.opt_ai, '?')
            lbl = f"vs Bot · Lv{self.opt_ai} · {elo} Elo · {t2['label']}"
        else:
            lbl = f"Local 2P · {t2['label']}"
        tc(s,lbl,FNT_XS,C_TEXT2,px0+PW//2,y+7); y+=18

        # ── Captured pieces + material advantage ────────────────────────────
        w_cap, b_cap, adv = self._captured()
        # adv > 0 → white ahead,  adv < 0 → black ahead

        # Determine which player is "opponent" (top) and "player" (bottom)
        # If playing as black: top=black(player), bottom=white(opponent)
        # Default / white / local: top=black(opponent), bottom=white(player)
        player_is_black = (self.mode=='bot' and self.player_col==BLACK)

        if player_is_black:
            # top = black (player), bottom = white (opponent)
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            top_label, bot_label = "Black (You)", "White (Bot)"
            top_cap,   bot_cap   = b_cap, w_cap   # black captured whites; white captured
blacks
```

```
            top_adv  = max(0,-adv)   # black advantage
            bot_adv  = max(0, adv)   # white advantage
        else:
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            if self.mode=='bot':
                top_label = "Black (Bot)"; bot_label = "White (You)"
            else:
                top_label = "Black"; bot_label = "White"
            top_cap,   bot_cap   = b_cap, w_cap
            top_adv  = max(0,-adv)
            bot_adv  = max(0, adv)

        # ── TOP player box ──────────────────────────────────────────
        top_act = (self.cb.turn==top_color and self.clk_run and not self.cb.result)
        rr(s,(52,50,44) if top_act else C_PANEL2,(x,y,W,44),6,1,C_BORDER)
        tc(s,top_label,FNT_XS,C_TEXT2,x+W//2,y+10)
        tc(s,fmt(top_time),FNT_CLK,C_GOLD if top_act else C_TEXT2,x+W//2,y+30)
        y+=50

        # Pieces captured BY top player = opponent's pieces they took
        # top captures opponent pieces: if top=BLACK → took white pieces (uppercase)
        #                               if top=WHITE → took black pieces (lowercase)
        top_caps_disp = w_cap if top_color==BLACK else b_cap   # white pieces BLACK took /
black pieces WHITE took
        top_adv_pts   = max(0, -adv) if top_color==BLACK else max(0, adv)
        if top_caps_disp:
            self._draw_caps(s, top_caps_disp, top_adv_pts, x, y, W)
        y+=16

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8

        # ── Move list ───────────────────────────────────────────────
        hist=self.cb.history; ROW=17; max_vis=13
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''; i+=1
            pairs.append((len(pairs)+1,ws,bs))
```

```
start_p=max(0,len(pairs)-max_vis); ml_y=y
for rel,(mn,ws,bs) in enumerate(pairs[start_p:]):
    ry=ml_y+rel*ROW
    tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,x,ry)
    tl(s,ws,      FNT_MONO_SM,(225,220,205),x+30,ry)
    if bs: tl(s,bs,FNT_MONO_SM,(205,205,220),x+118,ry)
y=ml_y+max_vis*ROW+4

pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8
if self.cb.is_check() and not self.cb.result:
    tc(s,"Check!",FNT_SMB,C_RED,px0+PW//2,y+8); y+=22
if self.ai_busy:
    sf=isinstance(self.ai,StockfishAI) and self.ai.ready
    lbl2="Stockfish thinking..." if sf else "AI thinking..."
    tc(s,lbl2,FNT_SM,C_GOLD,px0+PW//2,y+8); y+=20
if self.notif and time.time()<self.notif_exp:
    tc(s,self.notif,FNT_SM,C_ACCENT,px0+PW//2,y+8)

# ── BOTTOM captured pieces ──────────────────────────────────
bot_caps_disp = b_cap if bot_color==BLACK else w_cap   # symmetric
bot_adv_pts   = max(0, -adv) if bot_color==BLACK else max(0, adv)
bot_cap_y = 488
if bot_caps_disp:
    self._draw_caps(s, bot_caps_disp, bot_adv_pts, x, bot_cap_y, W)

# ── BOTTOM player box ──────────────────────────────────────
bot_act = (self.cb.turn==bot_color and self.clk_run and not self.cb.result)
rr(s,(52,50,44) if bot_act else C_PANEL2,(x,506,W,44),6,1,C_BORDER)
tc(s,bot_label,FNT_XS,C_TEXT2,x+W//2,506+10)
tc(s,fmt(bot_time),FNT_CLK,C_GOLD if bot_act else C_TEXT,x+W//2,506+30)

# ── Buttons ────────────────────────────────────────────────
for lbl2,bx,by,bw2,bh,c in[("Review",10,558,84,30,C_GOLD),
                           ("Rematch",104,558,80,30,C_BTN),
                           ("Menu",194,558,84,30,C_BTN)]:
    ax=px0+bx; hov=(ax<=mx<=ax+bw2 and by<=my<=by+bh)
    rr(s,C_BTN_H if hov else c,(ax,by,bw2,bh),5,1,C_BORDER)
    tc(s,lbl2,FNT_SM,C_TEXT,ax+bw2//2,by+bh//2)
tc(s,"F:flip  R:review  ESC:menu",FNT_XS,C_TEXT3,px0+PW//2,WINDOW_H-10)
```

```
    def _d_promo(self):
        s=self.surf; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((BOARD_SIZE,BOARD_SIZE),pygame.SRCALPHA); ov.fill((0,0,0,168));
s.blit(ov,(0,0))
        rr(s,(46,44,38),(cx-125,cy-60,250,118),12,1,C_BORDER)
        tc(s,"Promote to:",FNT_MD,C_TEXT2,cx,cy-42)
        for i,p in enumerate(['Q','R','B','N']):
            piece=p if self.cb.turn==WHITE else p.lower()
            bx=cx-105+i*54; by=cy-22; hov=(bx<=mx<=bx+52 and by<=my<=by+50)
            rr(s,C_BTN_H if hov else C_BTN,(bx,by,52,50),7,1,C_BORDER)
            draw_piece_at(s,piece,bx,by,50)

    # ── Review ──────────────────────────────────────────────────────────
    def _d_rev(self):
        s=self.surf; idx=self.rev_idx
        board=self.rev_boards[idx] if idx<len(self.rev_boards) else(self.rev_boards[-1] if
self.rev_boards else None)
        lm=None
        if idx>0: m=self.rev_hist[idx-1]['move']; lm=(m['from'],m['to'])
        # Draw board without live highlights
        old_lm=self.lm; old_sel=self.sel; old_tgts=self.ltgts
        old_cb_board=self.cb.board if self.cb else None
        self.lm=lm; self.sel=None; self.ltgts=[]
        if self.cb: self.cb.board=[[None]*8 for _ in range(8)]
        self._d_board(s)
        if self.cb and old_cb_board: self.cb.board=old_cb_board
        self.lm=old_lm; self.sel=old_sel; self.ltgts=old_tgts
        if board:
            old_df=self.drag_fr; self.drag_fr=None
            self._d_pieces(s,board); self.drag_fr=old_df
        self._d_rev_nav(s); self._d_rev_panel(s)

    def _d_rev_nav(self,s):
        cx=BOARD_SIZE//2; by0=BOARD_SIZE-46; mx,my=pygame.mouse.get_pos()
        for bx,by,bw,bh,lbl in[(cx-132,by0,40,36,'|<'),(cx-84,by0,40,36,'<'),
                               (cx+44, by0,40,36,'>'),(cx+92,by0,40,36,'|>')]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            rr(s,C_BTN_H if hov else C_PANEL3,(bx,by,bw,bh),6,1,C_BORDER)
            tc(s,lbl,FNT_MDB,C_TEXT,bx+bw//2,by+bh//2)
        tc(s,f"{self.rev_idx} / {len(self.rev_hist)}",FNT_SM,C_TEXT2,cx,by0+18)
```

```
        tc(s,"Arrow keys or buttons to navigate",FNT_XS,C_TEXT3,cx,BOARD_SIZE-7)

    def _d_rev_panel(self,s):
        px0=BOARD_SIZE; PW=PANEL_WIDTH; x=px0+14; mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,PW,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        y=14; tc(s,"Game Review",FNT_LG,C_GOLD,px0+PW//2,y+10); y+=36
        pygame.draw.line(s,C_SEP,(x,y),(x+PW-28,y),1); y+=10

        # Paired move list in review panel (FIX #1)
        hist=self.rev_hist; cur=self.rev_idx; ROW=17; max_vis=18
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''; i+=1
            pairs.append((len(pairs)+1,ws,bs))
        cur_row=(cur-1)//2 if cur>0 else 0
        start=max(0,cur_row-max_vis//2); end=min(len(pairs),start+max_vis); start=max(0,end-
max_vis)
        for rel,(mn,ws,bs) in enumerate(pairs[start:end]):
            pi=start+rel; ry=y+rel*ROW
            w_act=(cur==pi*2+1); b_act=(cur==pi*2+2)
            if w_act: pygame.draw.rect(s,(68,65,48),(px0+8,ry-1,108,ROW),border_radius=3)
            if b_act: pygame.draw.rect(s,(68,65,48),(px0+8+118,ry-1,108,ROW),border_radius=3)
            tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,x,ry)
            tl(s,ws, FNT_MONO_SM,C_GOLD if w_act else(225,220,205),x+30,ry)
            if bs: tl(s,bs,FNT_MONO_SM,C_GOLD if b_act else(205,205,220),x+118,ry)

        bx,by2,bw,bh=px0+10,558,PW-20,30; hov=(bx<=mx<=bx+bw and by2<=my<=by2+bh)
        rr(s,C_BTN_H if hov else C_BTN,(bx,by2,bw,bh),5,1,C_BORDER)
        back="< Back to Game" if self.rev_src==S.PLAYING else "< Back to Records"
        tc(s,back,FNT_SM,C_TEXT,px0+PW//2,by2+bh//2)

    # ── History screen ──────────────────────────────────────────────
    def _d_hist(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Game Records",FNT_LG,C_TEXT,cx,45)
        pygame.draw.line(s,C_BORDER,(40,70),(WINDOW_W-40,70),1)
        recs=self.hist_recs
```

```
        if not recs: tc(s,"No games recorded yet.",FNT_MD,C_TEXT2,cx,WINDOW_H//2)
        else:
            item_h=70; list_y0=88; vis=min(7,(WINDOW_H-170)//item_h)
            for rel in range(vis):
                idx=rel+self.hist_scroll
                if idx>=len(recs): break
                rec=recs[-(idx+1)]; ry=list_y0+rel*item_h
                hov=(30<=mx<=WINDOW_W-30 and ry<=my<=ry+item_h-3)
                bg=C_PANEL3 if hov else(C_PANEL2 if rel%2==0 else C_PANEL)
                rr(s,bg,(30,ry,WINDOW_W-60,item_h-4),6,1,C_BORDER)
                res=rec.get('result','?')
                rclr=C_ACCENT if'White' in res else(C_RED if'Black' in res else C_TEXT2)
                tl(s,res,FNT_MDB,rclr,50,ry+6)
                reason=rec.get('reason','').capitalize()
                if reason: tl(s,f"by {reason}",FNT_XS,C_TEXT2,50,ry+24)
                mode='vs Bot' if rec.get('mode')=='bot' else'Local 2P'
                lvl=f" · Lv{rec.get('ai_level','?')}" if rec.get('mode')=='bot' else''
                tl(s,f"{mode}{lvl}  ·  {rec.get('tc','?')}",FNT_SM,C_TEXT,230,ry+8)
                tl(s,f"{rec.get('total_moves','?')} moves",FNT_XS,C_TEXT2,230,ry+26)
                tl(s,rec.get('date',''),FNT_XS,C_TEXT3,WINDOW_W-215,ry+8)
                moves=rec.get('moves',[]); prev=' '.join(f"{(i//2)+1}.{m}" if i%2==0 else m
for i,m in enumerate(moves[:10]))
                if len(moves)>10: prev+='...'
                tl(s,prev,FNT_XS,C_TEXT3,50,ry+44)
                if hov: tc(s,"Click to review",FNT_XS,C_GOLD,WINDOW_W-90,ry+item_h//2-4)
            total=len(recs)
            if total>vis:
                area=WINDOW_H-170; sbh=max(20,int(vis/total*area))
                sby=list_y0+int(self.hist_scroll/max(1,total-vis)*(area-sbh))
                pygame.draw.rect(s,C_PANEL3,(WINDOW_W-14,list_y0,6,area))
                pygame.draw.rect(s,C_TEXT2,(WINDOW_W-14,sby,6,sbh),border_radius=3)
        bbx,bby,bbw,bbh=cx-80,WINDOW_H-48,160,34; hov=(bbx<=mx<=bbx+bbw and bby<=my<=bby+bbh)
        rr(s,C_BTN_H if hov else C_BTN,(bbx,bby,bbw,bbh),6,1,C_BORDER)
        tc(s,"< Main Menu",FNT_SM,C_TEXT,cx,bby+bbh//2)

    # FIX #5 — confirm overlay
    def _d_hconf(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((0,0,0,155));
s.blit(ov,(0,0))
```

```
        rr(s,(46,44,38),(cx-185,cy-80,370,170),12,1,C_BORDER)
        tc(s,"Review this game?",FNT_LG,C_TEXT,cx,cy-52)
        if self.hist_confirm is not None and self.hist_recs:
            rec=self.hist_recs[-(self.hist_confirm+1)]
            info=f"{rec.get('result','?')}  ·  {rec.get('total_moves','?')} moves  ·
{rec.get('date','')}"
            tc(s,info,FNT_SM,C_TEXT2,cx,cy-22)
        for lbl2,bx,c in[("Review >",cx-120,C_BLUE),("Cancel",cx+10,C_BTN)]:
            bw2,bh2=110,46; by=cy+18; hov=(bx<=mx<=bx+bw2 and by<=my<=by+bh2)
            rr(s,tuple(min(255,v+20) for v in c) if hov else c,(bx,by,bw2,bh2),8,1,C_BORDER)
            tc(s,lbl2,FNT_MDB,C_TEXT,bx+bw2//2,by+bh2//2)

# ═══════════════════════════════════════════════════════════════════════════
if __name__ == '__main__':
    Game().run()
```

[Python] Chess

Version 2

Requires:

```
pip install pygame
brew install stockfish
```

Code:

```
"""
Chess Game — Lichess-style UI  (v3 — all fixes applied)

Fixes:
 1. Move list: white & black on SAME ROW  (1. e4  e5)
 2. Piece rendering: platform-aware font fallback, letter-in-box if no Unicode glyphs
 3. Increment added to the player who JUST MOVED (not the opponent)
 4. Arrow keys navigate review in both directions
 5. History screen: confirm dialog before opening a saved-game review
"""

import pygame, sys, copy, random, time, json, os, platform, subprocess, threading, shutil
pygame.init()

# ── Layout ──────────────────────────────────────────────────────────────
BOARD_SIZE  = 640
PANEL_WIDTH = 300
WINDOW_W    = BOARD_SIZE + PANEL_WIDTH
WINDOW_H    = BOARD_SIZE
SQ          = BOARD_SIZE // 8

# ── Colours ─────────────────────────────────────────────────────────────
C_BG        = (22,  21,  18)
C_DARK_SQ   = (181, 136,  99)
C_LIGHT_SQ  = (240, 217, 181)
C_HIGHLIGHT = (205, 210, 106)
C_SELECTED  = (246, 246, 105)
C_PANEL     = (28,  27,  24)
```

```
C_PANEL2    = (38,  37,  33)
C_PANEL3    = (52,  50,  44)
C_TEXT      = (222, 220, 215)
C_TEXT2     = (140, 135, 125)
C_TEXT3     = (88,  85,  78)
C_ACCENT    = (128, 196, 127)
C_GOLD      = (255, 188,  66)
C_RED       = (210,  90,  90)
C_BLUE      = (100, 155, 220)
C_BTN       = (55,  53,  47)
C_BTN_H     = (75,  73,  65)
C_BTN_A     = (100, 155, 220)
C_CHECK     = (200,  55,  55)
C_BORDER    = (65,  62,  55)
C_SEP       = (50,  48,  43)

# ─── Move classification colours ──────────────────────────────────────
CLF_BRILLIANT = (0,   200, 215)   # cyan
CLF_GREAT     = (90,  130, 170)   # blue-grey
CLF_BEST      = (100, 185, 100)   # green
CLF_GOOD      = (160, 210, 100)   # lime
CLF_MISTAKE   = (220, 140,  50)   # orange
CLF_BLUNDER   = (210,  70,  70)   # red
CLF_NONE      = (80,   78,  72)   # neutral

# name → (colour, symbol, description)
CLF_INFO = {
    'brilliant': (CLF_BRILLIANT, '!!', 'Brilliant — sacrifice with hidden value'),
    'great':     (CLF_GREAT,     '!',  'Great Move — only good reply'),
    'best':      (CLF_BEST,      '*',  'Best Move — top engine choice'),
    'good':      (CLF_GOOD,      'v',  'Good Move — solid play'),
    'mistake':   (CLF_MISTAKE,   '?',  'Mistake — clear positional loss'),
    'blunder':   (CLF_BLUNDER,   '??', 'Blunder — game-changing error'),
    'none':      (CLF_NONE,      '',   ''),
}

# ─── Fonts ────────────────────────────────────────────────────────────
FNT_XL      = pygame.font.SysFont("segoeui",  32, bold=True)
FNT_LG      = pygame.font.SysFont("segoeui",  22, bold=True)
FNT_MD      = pygame.font.SysFont("segoeui",  17)
```

```
FNT_MDB     = pygame.font.SysFont("segoeui",  17, bold=True)
FNT_SM      = pygame.font.SysFont("segoeui",  14)
FNT_SMB     = pygame.font.SysFont("segoeui",  14, bold=True)
FNT_XS      = pygame.font.SysFont("segoeui",  12)
FNT_MONO    = pygame.font.SysFont("consolas", 14)
FNT_MONO_SM = pygame.font.SysFont("consolas", 13)
FNT_CLK     = pygame.font.SysFont("consolas", 28, bold=True)

# ── Piece rendering  (FIX #2) ──────────────────────────────────────────
UNICODE_SYMS = {'K':'♔','Q':'♕','R':'♖','B':'♗','N':'♘','P':'♙',
                'k':'♚','q':'♛','r':'♜','b':'♝','n':'♞','p':'♟'}

def _make_piece_font(size):
    plat = platform.system()
    if plat == "Windows":
        cands = ["segoeuisymbol","seguisym","segoeui","arial unicode ms"]
    elif plat == "Darwin":
        cands = ["apple symbols","arial unicode ms","lucida grande"]
    else:
        cands = ["dejavusans","symbola","freesans","unifont"]
    cands += [None]
    for name in cands:
        try:
            f = pygame.font.SysFont(name, size) if name else pygame.font.Font(None, size)
            surf = f.render("♔", True, (255,255,255))
            if surf.get_width() > 4:
                return f, True
        except Exception:
            pass
    return pygame.font.SysFont("consolas", size, bold=True), False

_PF_CACHE = {}
def _pfont(size):
    if size not in _PF_CACHE:
        _PF_CACHE[size] = _make_piece_font(size)
    return _PF_CACHE[size]

def draw_piece_at(surf, piece, px, py, size=SQ):
    is_white = piece.isupper()
    fs = int(size * 0.74)
```

```
    font, use_uni = _pfont(fs)
    if not use_uni:
        pad = max(4, size//8)
        bg  = (245,230,200) if is_white else (55,50,45)
        fg  = (40,35,30)    if is_white else (240,230,215)
        pygame.draw.rect(surf, bg,
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         border_radius=max(2, size//10))
        pygame.draw.rect(surf, (0,0,0),
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         1, border_radius=max(2, size//10))
        t = font.render(piece.upper(), True, fg)
        surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))
        return
    sym = UNICODE_SYMS.get(piece, '?')
    oc  = (230,228,224) if not is_white else (28,26,22)
    for ox,oy in ((-1,0),(1,0),(0,-1),(0,1)):
        o = font.render(sym, True, oc)
        surf.blit(o, (px+size//2-o.get_width()//2+ox, py+size//2-o.get_height()//2+oy))
    clr = (255,255,255) if is_white else (22,20,18)
    t   = font.render(sym, True, clr)
    surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))

# ── Helpers ──────────────────────────────────────────────────────────────
WHITE = 'w'; BLACK = 'b'
SAVE_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "chess_history.json")

TIME_CONTROLS = [
    {"label":"5 min",  "name":"Blitz",     "base":300,  "inc":0 },
    {"label":"10 min", "name":"Blitz",     "base":600,  "inc":0 },
    {"label":"15+10",  "name":"Rapid",     "base":900,  "inc":10},
    {"label":"30 min", "name":"Rapid",     "base":1800, "inc":0 },
    {"label":"60 min", "name":"Classical", "base":3600, "inc":0 },
    {"label":"90+30",  "name":"Classical", "base":5400, "inc":30},
]

PIECE_VALUES = {'P':100,'N':320,'B':330,'R':500,'Q':900,'K':20000}
PST = {
'P':[ 0, 0, 0, 0, 0, 0, 0, 0,50,50,50,50,50,50,50,50,
     10,10,20,30,30,20,10,10, 5, 5,10,25,25,10, 5, 5,
```

```
      0, 0, 0,20,20, 0, 0, 0, 5,-5,-10,0,0,-10,-5, 5,
      5,10,10,-20,-20,10,10,5, 0, 0, 0, 0, 0, 0, 0, 0],
'N':[-50,-40,-30,-30,-30,-30,-40,-50,-40,-20,0,0,0,0,-20,-40,
     -30,0,10,15,15,10,0,-30,-30,5,15,20,20,15,5,-30,
     -30,0,15,20,20,15,0,-30,-30,5,10,15,15,10,5,-30,
     -40,-20,0,5,5,0,-20,-40,-50,-40,-30,-30,-30,-30,-40,-50],
'B':[-20,-10,-10,-10,-10,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,10,10,5,0,-10,-10,5,5,10,10,5,5,-10,
     -10,0,10,10,10,10,0,-10,-10,10,10,10,10,10,10,-10,
     -10,5,0,0,0,0,5,-10,-20,-10,-10,-10,-10,-10,-10,-20],
'R':[ 0,0,0,0,0,0,0,0, 5,10,10,10,10,10,10,5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5, 0,0,0,5,5,0,0,0],
'Q':[-20,-10,-10,-5,-5,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,5,5,5,0,-10,-5,0,5,5,5,5,0,-5,
      0,0,5,5,5,5,0,-5,-10,5,5,5,5,5,0,-10,
     -10,0,5,0,0,0,0,-10,-20,-10,-10,-5,-5,-10,-10,-20],
'K':[-30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -20,-30,-30,-40,-40,-30,-30,-20,-10,-20,-20,-20,-20,-20,-20,-10,
      20,20,0,0,0,0,20,20,20,30,10,0,0,10,30,20],
}

def rr(surf,color,rect,r=8,brd=0,bc=None):
    pygame.draw.rect(surf,color,rect,border_radius=r)
    if brd and bc: pygame.draw.rect(surf,bc,rect,brd,border_radius=r)

def tc(surf,txt,fnt,clr,cx,cy):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(cx-t.get_width()//2,cy-t.get_height()//2))

def tl(surf,txt,fnt,clr,x,y):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(x,y)); return t.get_width()

def fmt(secs):
    secs=max(0,int(secs)); m,s=divmod(secs,60); return f"{m}:{s:02d}"

# ═══════════════════════════════════════════════════════════════════════
# Chess Engine
# ═══════════════════════════════════════════════════════════════════════
```

```
class ChessBoard:
    INIT = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
    def __init__(self):
        self.board=[[None]*8 for _ in range(8)]; self.turn=WHITE
        self.castling={'K':True,'Q':True,'k':True,'q':True}
        self.ep_square=None; self.halfmove=0; self.fullmove=1
        self.history=[]; self.result=None; self.result_reason=''
        self._fen(self.INIT)

    def _fen(self,fen):
        p=fen.split()
        for r,row in enumerate(p[0].split('/')):
            c=0
            for ch in row:
                if ch.isdigit(): c+=int(ch)
                else: self.board[r][c]=ch; c+=1
        self.turn=WHITE if p[1]=='w' else BLACK
        cs=p[2]; self.castling={'K':'K' in cs,'Q':'Q' in cs,'k':'k' in cs,'q':'q' in cs}
        if p[3]!='-': self.ep_square=(8-int(p[3][1]),ord(p[3][0])-ord('a'))
        self.halfmove=int(p[4]); self.fullmove=int(p[5])

    def col(self,p): return WHITE if p and p.isupper() else (BLACK if p else None)
    def pt(self,p):  return p.upper() if p else None

    def pseudo(self,row,col):
        p=self.board[row][col]
        if not p: return []
        c=self.col(p); t=self.pt(p); mv=[]
        def ok(r,c2): return 0<=r<8 and 0<=c2<8
        def slide(dirs):
            for dr,dc in dirs:
                r,c2=row+dr,col+dc
                while ok(r,c2):
                    tgt=self.board[r][c2]
                    if tgt is None: mv.append((r,c2))
                    elif self.col(tgt)!=c: mv.append((r,c2)); break
                    else: break
                    r+=dr; c2+=dc
        def jump(ds):
            for dr,dc in ds:
```

```
                r,c2=row+dr,col+dc
                if ok(r,c2) and self.col(self.board[r][c2])!=c: mv.append((r,c2))
        if   t=='R': slide([(1,0),(-1,0),(0,1),(0,-1)])
        elif t=='B': slide([(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='Q': slide([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='N': jump([(2,1),(2,-1),(-2,1),(-2,-1),(1,2),(1,-2),(-1,2),(-1,-2)])
        elif t=='K':
            jump([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
            if c==WHITE and row==7 and col==4:
                if self.castling['K'] and not self.board[7][5] and not self.board[7][6]:
mv.append((7,6,'cK'))
                if self.castling['Q'] and not self.board[7][3] and not self.board[7][2] and
not self.board[7][1]: mv.append((7,2,'cQ'))
            elif c==BLACK and row==0 and col==4:
                if self.castling['k'] and not self.board[0][5] and not self.board[0][6]:
mv.append((0,6,'ck'))
                if self.castling['q'] and not self.board[0][3] and not self.board[0][2] and
not self.board[0][1]: mv.append((0,2,'cq'))
        elif t=='P':
            d=-1 if c==WHITE else 1; sr=6 if c==WHITE else 1; r2=row+d
            if ok(r2,col) and not self.board[r2][col]:
                mv.append((r2,col))
                if row==sr and not self.board[row+2*d][col]: mv.append((row+2*d,col))
            for dc in(-1,1):
                r2,c2=row+d,col+dc
                if ok(r2,c2):
                    tgt=self.board[r2][c2]
                    if tgt and self.col(tgt)!=c: mv.append((r2,c2))
                    elif self.ep_square==(r2,c2): mv.append((r2,c2,'ep'))
        return mv

    def _chk(self,color,board):
        king='K' if color==WHITE else 'k'
        kp=next(((r,c) for r in range(8) for c in range(8) if board[r][c]==king),None)
        if not kp: return True
        orig=self.board; self.board=board
        att=any((m[0],m[1])==kp for r in range(8) for c in range(8)
                if self.col(board[r][c]) is not None and self.col(board[r][c])!=color
                for m in self.pseudo(r,c))
        self.board=orig; return att
```

```
def _apply(self,board,castling,ep,fr,fc,tr,tc,sp=None,promo='Q'):
    b=copy.deepcopy(board); p=b[fr][fc]; c=self.col(p); t2=p.upper() if p else None
    nc=dict(castling); ne=None
    if sp in('cK','cQ','ck','cq'):
        b[tr][tc]=b[fr][fc]; b[fr][fc]=None
        rm={'cK':(7,7,7,5),'cQ':(7,0,7,3),'ck':(0,7,0,5),'cq':(0,0,0,3)}
        rr2,rc,rt,rtc2=rm[sp]; b[rt][rtc2]=b[rr2][rc]; b[rr2][rc]=None
    elif sp=='ep':
        b[tr][tc]=b[fr][fc]; b[fr][fc]=None; b[fr][tc]=None
    else:
        b[tr][tc]=b[fr][fc]; b[fr][fc]=None
    if t2=='P' and (tr==0 or tr==7): b[tr][tc]=promo if c==WHITE else promo.lower()
    if p=='K': nc['K']=nc['Q']=False
    if p=='k': nc['k']=nc['q']=False
    for sq,key in[((7,0),'Q'),((7,7),'K'),((0,0),'q'),((0,7),'k')]:
        if (fr,fc)==sq or (tr,tc)==sq: nc[key]=False
    if t2=='P' and abs(tr-fr)==2: ne=((fr+tr)//2,fc)
    return b,nc,ne

def legal(self,row,col):
    p=self.board[row][col]
    if not p: return []
    c=self.col(p); res=[]
    for m in self.pseudo(row,col):
        tr,tc2=m[0],m[1]; sp=m[2] if len(m)>2 else None
        if sp in('cK','cQ','ck','cq'):
            if self._chk(c,self.board): continue
            mc=5 if tc2==6 else 3
            b2,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,row,mc)
            if self._chk(c,b2): continue
        nb,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,tr,tc2,sp)
        if not self._chk(c,nb): res.append(m)
    return res

def all_legal(self,color=None):
    if color is None: color=self.turn
    return [(r,c)+m for r in range(8) for c in range(8)
            if self.col(self.board[r][c])==color for m in self.legal(r,c)]
```

```
def is_check(self):      return self._chk(self.turn,self.board)
def is_checkmate(self):  return not self.all_legal() and self.is_check()
def is_stalemate(self):  return not self.all_legal() and not self.is_check()
def is_insuf(self):
    ps=[(p.upper(),r,c) for r in range(8) for c in range(8)
        if (p:=self.board[r][c]) and p.upper()!='K']
    return len(ps)==0 or (len(ps)==1 and ps[0][0] in('N','B'))

def _san(self,fr,fc,tr,tc,sp,promo,board):
    p=board[fr][fc];
    if not p: return ''
    pt=p.upper(); CL='abcdefgh'; RL='87654321'; dest=CL[tc]+RL[tr]
    if sp in('cK','ck'): return 'O-O'
    if sp in('cQ','cq'): return 'O-O-O'
    cap='x' if board[tr][tc] or sp=='ep' else ''
    if pt=='P':
        s=(CL[fc]+cap+dest) if cap else dest
        if tr==0 or tr==7: s+='='+promo
        return s
    return pt+cap+dest

def fen(self):
    rows=[]
    for r in range(8):
        e=0; s2=''
        for c in range(8):
            p=self.board[r][c]
            if not p: e+=1
            else:
                if e: s2+=str(e); e=0
                s2+=p
        if e: s2+=str(e)
        rows.append(s2)
    f='/'.join(rows)+' '+('w' if self.turn==WHITE else 'b')+' '
    cs=''.join(k for k in('K','Q','k','q') if self.castling[k])
    f+=(cs or '-')+' '
    if self.ep_square: f+='abcdefgh'[self.ep_square[1]]+str(8-self.ep_square[0])
    else: f+='-'
    f+=f' {self.halfmove} {self.fullmove}'
    return f
```

```
    def make_move(self,fr,fc,tr,tc,sp=None,promo='Q'):
        p=self.board[fr][fc]; cap=self.board[tr][tc]
        if sp=='ep': cap=self.board[fr][tc]
        old_b=copy.deepcopy(self.board); san=self._san(fr,fc,tr,tc,sp,promo,old_b)
        nb,nc,ne=self._apply(self.board,self.castling,self.ep_square,fr,fc,tr,tc,sp,promo)

info={'from':(fr,fc),'to':(tr,tc),'piece':p,'captured':cap,'special':sp,'promo':promo,'san':sa
n}
        self.history.append({'move':info,'board':old_b,'castling':dict(self.castling),
                             'ep':self.ep_square,'hm':self.halfmove,'turn':self.turn})
        self.board=nb; self.castling=nc; self.ep_square=ne
        pt=p.upper() if p else None
        self.halfmove=0 if (pt=='P' or cap) else self.halfmove+1
        if self.turn==BLACK: self.fullmove+=1
        self.turn=BLACK if self.turn==WHITE else WHITE
        if   self.is_checkmate():  self.result=WHITE if self.turn==BLACK else BLACK;
self.result_reason='checkmate'
        elif self.is_stalemate():  self.result='draw'; self.result_reason='stalemate'
        elif self.is_insuf():      self.result='draw'; self.result_reason='insufficient
material'
        elif self.halfmove>=100:   self.result='draw'; self.result_reason='50-move rule'
        return info

# ═══════════════════════════════════════════════════════════════════════════
# Fallback Python AI  (used when Stockfish is unavailable)
# ═══════════════════════════════════════════════════════════════════════════
class AI:
    D={1:1,2:1,3:2,4:2,5:3,6:3,7:4,8:4,9:5,10:6,11:6}
    N={1:350,2:250,3:180,4:120,5:70,6:40,7:20,8:8,9:2,10:0,11:0}
    def __init__(self,lv=5): self.lv=lv
    def eval(self,cb):
        s=0
        for r in range(8):
            for c in range(8):
                p=cb.board[r][c]
                if not p: continue
                pt=p.upper(); v=PIECE_VALUES.get(pt,0); idx=r*8+c if p.isupper() else (7-
r)*8+c
                s+=(v+PST[pt][idx]) if p.isupper() else -(v+PST[pt][idx])
```

```
        # mobility removed for speed
        return s
    def _ord(self,cb,moves):
        def sc(m):
            fr,fc,tr,tc=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
            cap=cb.board[tr][tc] if sp!='ep' else cb.board[m[0]][tc]
            return PIECE_VALUES.get((cap or'').upper(),0)-
PIECE_VALUES.get(cb.board[fr][fc].upper(),0)//10
        return sorted(moves,key=sc,reverse=True)
    def _cl(self,cb):
        n=ChessBoard.__new__(ChessBoard)
        n.board=copy.deepcopy(cb.board); n.turn=cb.turn; n.castling=dict(cb.castling)
        n.ep_square=cb.ep_square; n.halfmove=cb.halfmove; n.fullmove=cb.fullmove
        n.history=[]; n.result=cb.result; n.result_reason=''; return n
    def _ab(self,cb,d,a,b,mx):
        if d==0 or cb.result: return self.eval(cb)
        ms=cb.all_legal()
        if not ms: return (-99999 if mx else 99999) if cb.is_check() else 0
        ms=self._ord(cb,ms)
        if mx:
            v=-999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=max(v,self._ab(c2,d-1,a,b,False)); a=max(a,v)
                if b<=a: break
            return v
        else:
            v=999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=min(v,self._ab(c2,d-1,a,b,True)); b=min(b,v)
                if b<=a: break
            return v
    def best(self,cb):
        depth=self.D.get(self.lv,3); noise=self.N.get(self.lv,0)
        ms=cb.all_legal()
        if not ms: return None
        ms=self._ord(cb,ms); mx=(cb.turn==WHITE); bv=-999999 if mx else 999999; bms=[]
        for m in ms:
            c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
```

```
            v=self._ab(c2,depth-1,-999999,999999,not mx)+random.randint(-noise,noise)
            if(mx and v>bv)or(not mx and v<bv): bv=v; bms=[m]
            elif v==bv: bms.append(m)
        return random.choice(bms) if bms else random.choice(ms)

# ═══════════════════════════════════════════════════════════════════════
# Stockfish AI  (async, non-blocking)
# ═══════════════════════════════════════════════════════════════════════
class StockfishAI:
    # Level → UCI_Elo
    LEVEL_ELO  = {1:500, 2:800, 3:1200, 4:1600, 5:1800,
                  6:2000, 7:2200, 8:2300, 9:2400, 10:2500, 11:2700}
    # Level → movetime (ms)
    LEVEL_TIME = {1:80, 2:100, 3:150, 4:200, 5:300,
                  6:500, 7:800, 8:1200, 9:1800, 10:2500, 11:4000}

    def __init__(self, level=5):
        self.level   = level
        self._proc   = None
        self._thread = None
        self._result = None   # (fr,fc,tr,tc,sp,promo) when ready
        self._lock   = threading.Lock()
        self.ready   = False
        self._init()

    # ── process management ──────────────────────────────────────────
    def _find(self):
        # 1) same folder as this script
        base = os.path.dirname(os.path.abspath(__file__))
        for name in ("stockfish","stockfish.exe",
                     "stockfish-windows-x86-64-avx2.exe",
                     "stockfish-windows-x86-64-modern.exe"):
            p = os.path.join(base, name)
            if os.path.isfile(p): return p
        # 2) system PATH  (covers brew install on macOS)
        return shutil.which("stockfish")

    def _init(self):
        path = self._find()
        if not path:
```

```
            print("[Stockfish] Not found — using fallback Python AI.")
            return
        try:
            self._proc = subprocess.Popen(
                [path],
                stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                stderr=subprocess.DEVNULL, text=True, bufsize=1)
            self._send("uci");        self._wait("uciok")
            self._apply_level(self.level)
            self._send("isready");    self._wait("readyok")
            self.ready = True
            print(f"[Stockfish] Ready  level={self.level}  "
                  f"elo={self.LEVEL_ELO[self.level]}")
        except Exception as e:
            print(f"[Stockfish] Failed to start: {e}")
            self.ready = False

    def _send(self, cmd):
        if self._proc:
            self._proc.stdin.write(cmd + "\n")
            self._proc.stdin.flush()

    def _wait(self, token, timeout=10.0):
        deadline = time.time() + timeout
        while time.time() < deadline:
            line = self._proc.stdout.readline()
            if token in line: return line
        return ""

    def _apply_level(self, level):
        elo = self.LEVEL_ELO.get(level, 1750)
        self._send("setoption name UCI_LimitStrength value true")
        self._send(f"setoption name UCI_Elo value {elo}")

    def set_level(self, level):
        self.level = level
        if self.ready:
            self._apply_level(level)
            self._send("isready"); self._wait("readyok")
```

```
    def close(self):
        if self._proc:
            try: self._send("quit"); self._proc.terminate()
            except: pass
            self._proc = None
        self.ready = False

    # ── async move request ──────────────────────────────────────────
    def start_thinking(self, fen):
        if not self.ready: return
        with self._lock: self._result = None
        self._thread = threading.Thread(target=self._think, args=(fen,), daemon=True)
        self._thread.start()

    def _think(self, fen):
        mt = self.LEVEL_TIME.get(self.level, 500)
        self._send(f"position fen {fen}")
        self._send(f"go movetime {mt}")
        line = self._wait("bestmove", timeout=mt/1000 + 8)
        parts = line.strip().split()
        if len(parts) >= 2 and parts[0] == "bestmove" and parts[1] != "(none)":
            with self._lock:
                self._result = self._parse(parts[1])

    def is_thinking(self):
        return self._thread is not None and self._thread.is_alive()

    def get_result(self):
        """Returns move tuple or None if still thinking."""
        if self.is_thinking(): return None
        with self._lock:
            r = self._result; self._result = None
        return r

    # ── UCI string → internal move tuple ────────────────────────────
    def _parse(self, uci):
        """
        'e2e4'  → (6,4,4,4, None, 'Q')
        'e7e8q' → (1,4,0,4, None, 'Q')   promotion
        'e1g1'  → (7,4,7,6, 'cK', 'Q')   castling
```

```
        """
        fc  = ord(uci[0]) - ord('a')
        fr  = 8 - int(uci[1])
        tc2 = ord(uci[2]) - ord('a')
        tr  = 8 - int(uci[3])
        promo = uci[4].upper() if len(uci) == 5 else 'Q'
        # Castling: king moves exactly 2 squares horizontally
        sp = None
        if   uci == 'e1g1': sp = 'cK'
        elif uci == 'e1c1': sp = 'cQ'
        elif uci == 'e8g8': sp = 'ck'
        elif uci == 'e8c8': sp = 'cq'
        return (fr, fc, tr, tc2, sp, promo)

# ═══════════════════════════════════════════════════════════════════════
# Review Analyser  — uses Stockfish to score every position in a game
# ═══════════════════════════════════════════════════════════════════════
class ReviewAnalyser:
    """
    Runs Stockfish analysis on every position in a game (in a background thread)
    and classifies each half-move as brilliant / great / best / good / mistake / blunder.

    Results are stored as:
      self.evals  : list[float]  cp score (centipawns, white-positive) AFTER move i
                    index 0 = initial position, index N = after move N
      self.classif: list[str]   classification for move i (1-based, index 0 unused)
      self.done   : bool
    """
    DEPTH      = 16          # analysis depth
    ANALYSIS_MOVETIME = 300  # ms per position

    def __init__(self, sf_path):
        self.sf_path  = sf_path
        self.evals    = []
        self.classif  = []
        self.done     = False
        self._thread  = None

    def analyse(self, rev_hist, rev_boards):
        """Start background analysis. rev_hist / rev_boards from _build_rev."""
```

```
        self.evals   = [None] * (len(rev_hist) + 1)
        self.classif = ['none'] * (len(rev_hist) + 1)
        self.done    = False
        self._thread = threading.Thread(
            target=self._run, args=(rev_hist, rev_boards), daemon=True)
        self._thread.start()

    def _score_fen(self, proc, fen):
        """Ask Stockfish for a centipawn score for this FEN. Returns float (white POV)."""
        proc.stdin.write(f"position fen {fen}\n")
        proc.stdin.write(f"go movetime {self.ANALYSIS_MOVETIME}\n")
        proc.stdin.flush()
        score = None
        mate  = None
        deadline = time.time() + self.ANALYSIS_MOVETIME/1000 + 5
        while time.time() < deadline:
            line = proc.stdout.readline().strip()
            if line.startswith("info") and "score" in line:
                parts = line.split()
                try:
                    si = parts.index("score")
                    kind = parts[si+1]
                    val  = int(parts[si+2])
                    if kind == "cp":   score = val
                    elif kind == "mate": mate = val
                except (ValueError, IndexError):
                    pass
            if line.startswith("bestmove"):
                break
        if mate is not None:
            return 30000 if mate > 0 else -30000
        return float(score) if score is not None else 0.0

    def _run(self, rev_hist, rev_boards):
        path = self.sf_path
        if not path or not os.path.isfile(path):
            path = shutil.which("stockfish")
        if not path:
            self.done = True; return
        try:
```

```
            proc = subprocess.Popen(
                [path],
                stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                stderr=subprocess.DEVNULL, text=True, bufsize=1)
            proc.stdin.write("uci\n"); proc.stdin.flush()
            # wait for uciok
            deadline = time.time()+10
            while time.time()<deadline:
                if "uciok" in proc.stdout.readline(): break
            proc.stdin.write("setoption name UCI_LimitStrength value false\n")
            proc.stdin.write("isready\n"); proc.stdin.flush()
            deadline = time.time()+10
            while time.time()<deadline:
                if "readyok" in proc.stdout.readline(): break

            # Build a temporary ChessBoard to replay
            tmp = ChessBoard()
            self.evals[0] = self._score_fen(proc, tmp.fen())

            for i, h in enumerate(rev_hist):
                m = h['move']
                tmp.make_move(m['from'][0],m['from'][1],
                              m['to'][0],  m['to'][1],
                              m['special'], m['promo'])
                score = self._score_fen(proc, tmp.fen())
                self.evals[i+1] = score

            proc.stdin.write("quit\n"); proc.stdin.flush()
            proc.terminate()
        except Exception as e:
            print(f"[ReviewAnalyser] error: {e}")
        finally:
            self._classify(rev_hist)
            self.done = True

    def _classify(self, rev_hist):
        """Classify each move given the eval before and after."""
        for i, h in enumerate(rev_hist):
            before = self.evals[i]
            after  = self.evals[i+1]
```

```
            if before is None or after is None:
                self.classif[i+1] = 'none'; continue
            color = h['move'].get('piece','?')
            # delta from the mover's perspective (positive = good for mover)
            if color and color.isupper():   # white moved
                delta = after - before   # higher after = better for white
            else:                           # black moved
                delta = before - after   # lower after = better for black (cp is white-POV)

            # Was the best move actually played?
            # We approximate: if delta >= -10 it's essentially best
            if delta >= -10:
                # Check for brilliant: mover sacrificed material yet eval improved
                cap = h['move'].get('captured')
                cap_val = {'P':100,'N':320,'B':330,'R':500,'Q':900}.get(
                    (cap or '').upper(), 0)
                if cap_val == 0 and delta >= 50:
                    self.classif[i+1] = 'brilliant'
                elif cap_val > 0 and delta >= 0:
                    # sacrificed but still good → brilliant candidate
                    self.classif[i+1] = 'brilliant'
                else:
                    self.classif[i+1] = 'best'
            elif delta >= -30:
                self.classif[i+1] = 'great'
            elif delta >= -80:
                self.classif[i+1] = 'good'
            elif delta >= -200:
                self.classif[i+1] = 'mistake'
            else:
                self.classif[i+1] = 'blunder'

    def get_eval(self, idx):
        """Return eval at position idx (0=start), or None if not ready yet."""
        if idx < len(self.evals): return self.evals[idx]
        return None

    def get_classif(self, move_idx):
        """Return classification string for half-move move_idx (1-based)."""
        if move_idx < len(self.classif): return self.classif[move_idx]
```

```
        return 'none'
def load_hist():
    try:
        with open(SAVE_FILE,'r') as f: return json.load(f)
    except: return []

def save_hist(recs):
    try:
        with open(SAVE_FILE,'w') as f: json.dump(recs[-10:],f,indent=2)
    except: pass

def record_game(cb,mode,ai_lv,tc_label,wt,bt):
    recs=load_hist()
    rs="White wins" if cb.result==WHITE else ("Black wins" if cb.result==BLACK else "Draw")
    recs.append({'date':time.strftime('%Y-%m-%d %H:%M'),'mode':mode,'ai_level':ai_lv,
                 'tc':tc_label,'result':rs,'reason':cb.result_reason,
                 'moves':[h['move']['san'] for h in cb.history],
                 'white_time_left':round(wt,1),'black_time_left':round(bt,1),
                 'total_moves':len(cb.history)})
    save_hist(recs)

# ═══════════════════════════════════════════════════════════════════════════
# Screen IDs
# ═══════════════════════════════════════════════════════════════════════════
class S:
    MAIN=0; SETUP_BOT=1; SETUP_LOC=2; PLAYING=3; PROMOTION=4
    REVIEW=5; HISTORY=6; HIST_CONFIRM=7

# ═══════════════════════════════════════════════════════════════════════════
# Game
# ═══════════════════════════════════════════════════════════════════════════
class Game:
    def __init__(self):
        self.surf=pygame.display.set_mode((WINDOW_W,WINDOW_H))
        pygame.display.set_caption("Chess")
        self.clk=pygame.time.Clock()
        self.state=S.MAIN
        # setup
        self.opt_ai=5; self.opt_tc=0; self.opt_col=WHITE
        # game
```

```
    self.cb=None; self.ai=StockfishAI(level=5); self.mode=None
    self.player_col=WHITE; self.flipped=False
    # clocks
    self.wt=0.0; self.bt=0.0; self.inc=0
    self.clk_last=0.0; self.clk_run=False
    # board UI
    self.sel=None; self.ltgts=[]; self.lm=None
    self.drag_p=None; self.drag_pos=None; self.drag_fr=None
    # promo
    self.promo=None
    # ai
    self.ai_busy=False
    # review
    self.rev_hist=[]; self.rev_idx=0; self.rev_boards=[]
    self.rev_src=S.PLAYING  # where Back goes
    self.rev_analyser=None  # ReviewAnalyser instance
    # history
    self.hist_recs=[]; self.hist_scroll=0; self.hist_confirm=None
    # notif
    self.notif=''; self.notif_exp=0

# ── coords ──────────────────────────────────────────────────────────
def s2p(self,r,c):
    return ((7-c)*SQ,(7-r)*SQ) if self.flipped else (c*SQ,r*SQ)
def p2s(self,x,y):
    if not(0<=x<BOARD_SIZE and 0<=y<BOARD_SIZE): return None,None
    return (7-y//SQ,7-x//SQ) if self.flipped else (y//SQ,x//SQ)

# ── clock ───────────────────────────────────────────────────────────
def tick(self):
    if not self.clk_run or not self.cb or self.cb.result: return
    now=time.time(); dt=now-self.clk_last; self.clk_last=now
    if self.cb.turn==WHITE: self.wt=max(0,self.wt-dt)
    else:                   self.bt=max(0,self.bt-dt)
    if self.wt<=0 or self.bt<=0:
        self.cb.result=BLACK if self.wt<=0 else WHITE
        self.cb.result_reason='timeout'
        self.clk_run=False; self._end()

# FIX #3 — increment goes to the MOVER
```

```
    def _add_inc(self,mover):
        if self.inc<=0: return
        if mover==WHITE: self.wt+=self.inc
        else:            self.bt+=self.inc

    # ── lifecycle ──────────────────────────────────────────────────────────
    def start(self,mode):
        self.mode=mode; tc=TIME_CONTROLS[self.opt_tc]
        self.wt=float(tc['base']); self.bt=float(tc['base']); self.inc=tc['inc']
        self.cb=ChessBoard()
        # Close previous Stockfish process before creating a new one
        if isinstance(self.ai, StockfishAI): self.ai.close()
        self.ai=StockfishAI(level=self.opt_ai)
        # If Stockfish unavailable, fall back to Python AI
        if not self.ai.ready: self.ai=AI(self.opt_ai)
        self.sel=None; self.ltgts=[]; self.lm=None
        self.drag_p=None; self.promo=None; self.ai_busy=False
        self.clk_run=False; self.clk_last=time.time()
        if mode=='bot':
            self.player_col=self.opt_col; self.flipped=(self.opt_col==BLACK)
        else:
            self.player_col=WHITE; self.flipped=False
        self.state=S.PLAYING

    def _end(self):
        record_game(self.cb,self.mode,self.opt_ai,
                    TIME_CONTROLS[self.opt_tc]['label'],self.wt,self.bt)

    def open_review(self,history,src=S.PLAYING):
        self.rev_hist=history; self.rev_src=src
        self._build_rev()
        self.rev_idx=len(self.rev_hist); self.state=S.REVIEW
        # Start Stockfish analysis in background
        sf_path = shutil.which("stockfish")
        self.rev_analyser = ReviewAnalyser(sf_path)
        self.rev_analyser.analyse(self.rev_hist, self.rev_boards)

    def open_review_game(self):
        if self.cb and self.cb.history: self.open_review(self.cb.history[:],S.PLAYING)
```

```
    def _build_rev(self):
        self.rev_boards=[]
        tmp=ChessBoard(); self.rev_boards.append(copy.deepcopy(tmp.board))
        for h in self.rev_hist:
            m=h['move']

tmp.make_move(m['from'][0],m['from'][1],m['to'][0],m['to'][1],m['special'],m['promo'])
            self.rev_boards.append(copy.deepcopy(tmp.board))

    def open_hist(self):
        self.hist_recs=load_hist(); self.hist_scroll=0; self.hist_confirm=None
        self.state=S.HISTORY

    # ── move execution ──────────────────────────────────────────────
    def try_move(self,fr,fc,tr,tc2):
        legal=self.cb.legal(fr,fc)
        m=next((x for x in legal if x[0]==tr and x[1]==tc2),None)
        if m is None:
            p=self.cb.board[tr][tc2]
            if p and self.cb.col(p)==self.cb.turn:
                self.sel=(tr,tc2); self.ltgts=self.cb.legal(tr,tc2)
            else:
                self.sel=None; self.ltgts=[]
            return
        sp=m[2] if len(m)>2 else None
        if self.cb.pt(self.cb.board[fr][fc])=='P' and (tr==0 or tr==7):
            self.promo=(fr,fc,tr,tc2,sp); self.state=S.PROMOTION
            self.sel=None; self.ltgts=[]; return
        self._do(fr,fc,tr,tc2,sp,'Q')

    def _do(self,fr,fc,tr,tc2,sp,promo):
        mover=self.cb.turn
        info=self.cb.make_move(fr,fc,tr,tc2,sp,promo)
        self._add_inc(mover)                    # FIX #3
        self.lm=((fr,fc),(tr,tc2))
        self.sel=None; self.ltgts=[]
        self.clk_last=time.time()
        if not self.clk_run and len(self.cb.history)>=1: self.clk_run=True
        if self.cb.result: self.clk_run=False; self._end()
        self.notif=info['san']; self.notif_exp=time.time()+2.2
```

```
def ai_tick(self):
    if self.state!=S.PLAYING or self.mode!='bot' or self.cb.result: return
    if self.cb.turn==self.player_col: return

    if isinstance(self.ai, StockfishAI):
        # ── Stockfish path (already async) ──────────────────────────────
        if not self.ai_busy:
            self.ai.start_thinking(self.cb.fen())
            self.ai_busy=True
        else:
            result=self.ai.get_result()
            if result is not None:
                fr,fc,tr,tc2,sp,promo=result
                legal=self.cb.legal(fr,fc)
                matched=next((m for m in legal if m[0]==tr and m[1]==tc2),None)
                if matched is not None:
                    sp2=matched[2] if len(matched)>2 else sp
                    self._do(fr,fc,tr,tc2,sp2,promo)
                self.ai_busy=False
    else:
        # ── Fallback Python AI — run in background thread ────────────────
        if self.ai_busy: return
        self.ai_busy=True
        # snapshot board state for the thread
        cb_snap=self.ai._cl(self.cb)  # lightweight clone, no history
        def _bg():
            m=self.ai.best(cb_snap)
            self._py_ai_result=m
        self._py_ai_result=None
        t=threading.Thread(target=_bg,daemon=True); t.start()
        self._py_ai_thread=t

def _py_ai_poll(self):
    """Called every frame to check if the Python AI thread finished."""
    if not self.ai_busy: return
    if isinstance(self.ai, StockfishAI): return
    t=getattr(self,'_py_ai_thread',None)
    if t and not t.is_alive():
        m=getattr(self,'_py_ai_result',None)
```

```
        self.ai_busy=False
        if m and self.state==S.PLAYING and not self.cb.result:
            self._do(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None,'Q')

# ── find move by san for record replay ──────────────────────────────
def _find_san(self,cb,san):
    for m in cb.all_legal():
        fr,fc,tr,tc2=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
        for pr in('Q','R','B','N'):
            if '=' in san and san[-1]!=pr: continue
            if cb._san(fr,fc,tr,tc2,sp,pr,cb.board)==san: return m
    return None

def _open_rec_review(self,idx):
    recs=self.hist_recs
    if not recs or idx>=len(recs): return
    rec=recs[-(idx+1)]
    tmp=ChessBoard()
    for san in rec.get('moves',[]):
        m=self._find_san(tmp,san)
        if m is None: break
        tmp.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
    self.open_review(tmp.history[:],S.HISTORY)

# ── event loop ──────────────────────────────────────────────────────
def run(self):
    while True:
        self.clk.tick(60); self.tick()
        for ev in pygame.event.get():
            if ev.type==pygame.QUIT:
                if isinstance(self.ai, StockfishAI): self.ai.close()
                pygame.quit(); sys.exit()
            if ev.type==pygame.KEYDOWN:         self._key(ev.key)
            if ev.type==pygame.MOUSEBUTTONDOWN: self._click(ev.pos,ev.button)
            if ev.type==pygame.MOUSEBUTTONUP:   self._rel(ev.pos)
            if ev.type==pygame.MOUSEMOTION:
                if self.drag_p: self.drag_pos=ev.pos
        if self.state==S.PLAYING: self.ai_tick(); self._py_ai_poll()
        self._draw()
```

```
    # ── key ──────────────────────────────────────────────────────────────
    def _key(self,key):
        if self.state==S.REVIEW:                                  # FIX #4
            if key in(pygame.K_LEFT,pygame.K_a):  self.rev_idx=max(0,self.rev_idx-1)
            if key in(pygame.K_RIGHT,pygame.K_d):
self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
            if key==pygame.K_HOME: self.rev_idx=0
            if key==pygame.K_END:  self.rev_idx=len(self.rev_hist)
            if key==pygame.K_ESCAPE: self.state=self.rev_src
        elif self.state==S.PLAYING:
            if key==pygame.K_ESCAPE: self.state=S.MAIN
            if key==pygame.K_f:      self.flipped=not self.flipped
            if key==pygame.K_r:      self.open_review_game()
        elif self.state==S.HIST_CONFIRM:
            if key==pygame.K_ESCAPE: self.state=S.HISTORY
        elif key==pygame.K_ESCAPE: self.state=S.MAIN

    # ── click dispatcher ─────────────────────────────────────────────────
    def _click(self,pos,btn):
        {S.MAIN:self._c_main,S.SETUP_BOT:self._c_sbot,S.SETUP_LOC:self._c_sloc,
         S.PLAYING:self._c_play,S.PROMOTION:self._c_promo,S.REVIEW:self._c_rev,
         S.HISTORY:self._c_hist,S.HIST_CONFIRM:self._c_hconf
        }.get(self.state,lambda p:None)(pos)

    def _rel(self,pos):
        if self.state not in(S.PLAYING,S.PROMOTION) or not self.drag_p: return
        x,y=pos; r,c=self.p2s(x,y)
        if r is not None and self.drag_fr:
            fr,fc=self.drag_fr
            if(fr,fc)!=(r,c): self.try_move(fr,fc,r,c)
        self.drag_p=None; self.drag_pos=None; self.drag_fr=None

    def _c_main(self,pos):
        x,y=pos; cx=WINDOW_W//2
        if cx-160<=x<=cx-10  and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_BOT
        elif cx+10<=x<=cx+160 and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_LOC
        elif cx-100<=x<=cx+100 and WINDOW_H//2+60<=y<=WINDOW_H//2+110: self.open_hist()

    def _c_sbot(self,pos):
        x,y=pos; cx=WINDOW_W//2
```

```
        # 11 difficulty buttons — same layout as _d_sbot
        bw,bh=78,48; row1=6; row2=5
        for i in range(11):
            if i<row1:
                total_w=row1*bw+(row1-1)*6
                bx=cx-total_w//2+i*(bw+6); by=222
            else:
                j=i-row1
                total_w=row2*bw+(row2-1)*6
                bx=cx-total_w//2+j*(bw+6); by=278
            if bx<=x<=bx+bw and by<=y<=by+bh: self.opt_ai=i+1
        # Play as buttons
        if cx-160<=x<=cx-20 and 362<=y<=402:  self.opt_col=WHITE
        elif cx+20<=x<=cx+160 and 362<=y<=402: self.opt_col=BLACK
        # Time controls
        for i in range(len(TIME_CONTROLS)):
            bx=cx-210+(i%3)*142; by=440+(i//3)*52
            if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
        # Start
        if cx-100<=x<=cx+100 and 590<=y<=630: self.start('bot')
        # Back
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_sloc(self,pos):
        x,y=pos; cx=WINDOW_W//2
        for i in range(len(TIME_CONTROLS)):
            bx=cx-210+(i%3)*142; by=262+(i//3)*52
            if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
        if cx-100<=x<=cx+100 and 448<=y<=488: self.start('local')
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_play(self,pos):
        x,y=pos
        if x>=BOARD_SIZE: self._c_panel(pos); return
        if self.cb.result: return
        r,c=self.p2s(x,y)
        if r is None: return
        if self.mode=='bot' and self.cb.turn!=self.player_col: return
        p=self.cb.board[r][c]
        if p and self.cb.col(p)==self.cb.turn:
```

```
            self.sel=(r,c); self.ltgts=self.cb.legal(r,c)
            self.drag_p=p; self.drag_pos=pos; self.drag_fr=(r,c)
        elif self.sel:
            self.try_move(self.sel[0],self.sel[1],r,c)

    def _c_panel(self,pos):
        x,y=pos; px=x-BOARD_SIZE
        if 558<=y<=588:
            if  10<=px<=95:  self.open_review_game()
            elif 105<=px<=190: self.start(self.mode)
            elif 200<=px<=285: self.state=S.MAIN

    def _c_promo(self,pos):
        x,y=pos; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2
        for i,p in enumerate(['Q','R','B','N']):
            bx=cx-105+i*54; by=cy-22
            if bx<=x<=bx+52 and by<=y<=by+50:
                fr,fc,tr,tc2,sp=self.promo
                self._do(fr,fc,tr,tc2,sp,p)
                self.promo=None; self.state=S.PLAYING; return

    def _c_rev(self,pos):
        x,y=pos; ox=self.BOARD_OX; cx=ox+BOARD_SIZE//2; by0=BOARD_SIZE-46
        for bx,by,bw,bh,act in[(cx-132,by0,40,36,'first'),(cx-84,by0,40,36,'prev'),
                               (cx+44, by0,40,36,'next'),(cx+92,by0,40,36,'last')]:
            if bx<=x<=bx+bw and by<=y<=by+bh:
                if act=='first': self.rev_idx=0
                elif act=='prev': self.rev_idx=max(0,self.rev_idx-1)
                elif act=='next': self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
                elif act=='last': self.rev_idx=len(self.rev_hist)
                return
        panel_x=ox+BOARD_SIZE
        if x>=panel_x:
            pw=WINDOW_W-panel_x
            if 558<=y<=588 and panel_x+6<=x<=WINDOW_W-6:
                self.state=self.rev_src

    def _c_hist(self,pos):
        x,y=pos; cx=WINDOW_W//2
        if WINDOW_H-55<=y<=WINDOW_H-20 and cx-80<=x<=cx+80:
```

```
            self.state=S.MAIN; return
        item_h=70; list_y0=88
        if 100<=y<=WINDOW_H-80 and 30<=x<=WINDOW_W-30:
            idx=(y-list_y0)//item_h+self.hist_scroll
            if 0<=idx<len(self.hist_recs):
                self.hist_confirm=idx; self.state=S.HIST_CONFIRM  # FIX #5

    def _c_hconf(self,pos):
        x,y=pos; cx,cy=WINDOW_W//2,WINDOW_H//2
        if cx-120<=x<=cx-10 and cy+20<=y<=cy+65:  self._open_rec_review(self.hist_confirm)
        elif cx+10<=x<=cx+120 and cy+20<=y<=cy+65: self.state=S.HISTORY

    # ═══════════════════════════════════════════════════════════════════════
    # Drawing
    # ═══════════════════════════════════════════════════════════════════════
    def _draw(self):
        s=self.surf; s.fill(C_BG)
        if   self.state==S.MAIN:      self._d_main()
        elif self.state==S.SETUP_BOT: self._d_sbot()
        elif self.state==S.SETUP_LOC: self._d_sloc()
        elif self.state in(S.PLAYING,S.PROMOTION):
            self._d_play()
            if self.state==S.PROMOTION: self._d_promo()
        elif self.state==S.REVIEW:    self._d_rev()
        elif self.state==S.HISTORY:   self._d_hist()
        elif self.state==S.HIST_CONFIRM: self._d_hist(); self._d_hconf()
        pygame.display.flip()

    # ── Main menu ──────────────────────────────────────────────────────────
    def _d_main(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        for r in range(8):
            for c in range(8):
                clr=(50,46,40) if(r+c)%2==0 else(38,35,30)

pygame.draw.rect(s,clr,(c*(WINDOW_W//8),r*(WINDOW_H//8),WINDOW_W//8,WINDOW_H//8))
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((14,13,11,218));
s.blit(ov,(0,0))
        tc(s,"CHESS",FNT_XL,(235,210,150),cx,cy-155)
        tc(s,"Full Rules  ·  10 AI Levels  ·  Time Controls",FNT_SM,C_TEXT2,cx,cy-112)
```

```
        pygame.draw.line(s,C_BORDER,(cx-230,cy-92),(cx+230,cy-92),1)
        for bx,by,bw,bh,lbl,clr in[(cx-160,cy-30,150,60,"vs Bot",C_BLUE),(cx+10,cy-
30,150,60,"Local 2P",C_ACCENT)]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            c2=tuple(min(255,v+22) for v in clr) if hov else clr
            rr(s,c2,(bx,by,bw,bh),10,1,C_BORDER)
            tc(s,lbl,FNT_MDB,(10,10,10),bx+bw//2,by+bh//2)
        hbx,hby,hbw,hbh=cx-100,cy+60,200,50
        hov=(hbx<=mx<=hbx+hbw and hby<=my<=hby+hbh)
        rr(s,C_BTN_H if hov else C_BTN,(hbx,hby,hbw,hbh),8,1,C_BORDER)
        tc(s,"Game Records",FNT_MD,C_TEXT,hbx+hbw//2,hby+hbh//2)
        tc(s,"F: flip  |  R: review  |  ESC: menu",FNT_XS,C_TEXT3,cx,WINDOW_H-18)

    # ── Setup screens ──────────────────────────────────────────────────
    def _d_sbot(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Play vs Bot",FNT_LG,C_TEXT,cx,42)
        rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
        tc(s,"AI Difficulty",FNT_MD,C_TEXT2,cx,196)
        names=["Beginner","Novice","Amateur","Casual","Intermediate",
               "Advanced","Expert","Master","IM","GM","Super GM"]
        elos =[500,800,1200,1600,1800,2000,2200,2300,2400,2500,2700]
        # 11 buttons: first row 6, second row 5 — centred
        row1=6; row2=5
        bw,bh=78,48
        for i in range(11):
            if i<row1:
                total_w=row1*bw+(row1-1)*6
                bx=cx-total_w//2+i*(bw+6); by=222
            else:
                j=i-row1
                total_w=row2*bw+(row2-1)*6
                bx=cx-total_w//2+j*(bw+6); by=278
            sel=(self.opt_ai==i+1)
            c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw and by<=my<=by+bh) else C_BTN)
            rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
            tc(s,str(i+1),FNT_SMB,(255,255,255) if sel else C_TEXT2,bx+bw//2,by+12)
            tc(s,names[i],FNT_XS,(255,255,255) if sel else C_TEXT3,bx+bw//2,by+27)
            tc(s,str(elos[i]),FNT_XS,C_GOLD if sel else C_TEXT3,bx+bw//2,by+39)
        tc(s,"Play as",FNT_MD,C_TEXT2,cx,348)
```

```
        for lbl,col2,bx in[("White",WHITE,cx-160),("Black",BLACK,cx+20)]:
            bw2,bh2=140,40; by=362; sel=(self.opt_col==col2)
            c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw2 and by<=my<=by+bh2) else C_BTN)
            rr(s,c,(bx,by,bw2,bh2),7,1,C_BORDER)
            tc(s,lbl,FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw2//2,by+bh2//2)
        tc(s,"Time Control",FNT_MD,C_TEXT2,cx,420)
        self._d_tc(s,cx,mx,my,440)
        sbx,sby,sbw,sbh=cx-100,590,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
        rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
        tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

    def _d_sloc(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Local 2-Player",FNT_LG,C_TEXT,cx,42)
        rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
        tc(s,"Time Control",FNT_MD,C_TEXT2,cx,238)
        self._d_tc(s,cx,mx,my,258)
        sbx,sby,sbw,sbh=cx-100,448,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
        rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
        tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

    def _d_tc(self,s,cx,mx,my,y0):
        for i,t2 in enumerate(TIME_CONTROLS):
            bx=cx-210+(i%3)*142; by=y0+(i//3)*52; bw,bh=132,42; sel=(self.opt_tc==i)
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            c=C_BTN_A if sel else(C_BTN_H if hov else C_BTN)
            rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
            tc(s,t2['label'],FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw//2,by+14)
            tc(s,t2['name'], FNT_XS, (210,210,210) if sel else C_TEXT3,bx+bw//2,by+30)

    # ── Playing ──────────────────────────────────────────────────────────
    def _d_play(self):
        self._d_board(self.surf)
        self._d_pieces(self.surf,self.cb.board if self.cb else None)
        self._d_panel(self.surf)
        self._d_drag(self.surf)
        if self.cb and self.cb.result: self._d_result(self.surf)

    def _d_board(self,s,lm=None,sel=None,tgts=None):
        fl=self.flipped
```

```
        lm  =lm   if lm   is not None else self.lm
        sel =sel  if sel  is not None else self.sel
        tgts=tgts if tgts is not None else self.ltgts
        chk_sq=None
        if self.cb and self.cb.is_check() and not self.cb.result:
            king='K' if self.cb.turn==WHITE else 'k'
            for r in range(8):
                for c in range(8):
                    if self.cb.board[r][c]==king: chk_sq=(r,c)
        for r in range(8):
            for c in range(8):
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                base=C_LIGHT_SQ if(r+c)%2==0 else C_DARK_SQ
                pygame.draw.rect(s,base,(px,py,SQ,SQ))
                if lm and((r,c)==lm[0] or(r,c)==lm[1]):
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_HIGHLIGHT,170));
s.blit(hl,(px,py))
                if sel and(r,c)==sel:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA);
hl.fill((*C_SELECTED[:3],210)); s.blit(hl,(px,py))
                if chk_sq and(r,c)==chk_sq:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_CHECK,185));
s.blit(hl,(px,py))
        cb_b=self.cb.board if self.cb else [[None]*8 for _ in range(8)]
        for m in tgts:
            tr2,tc2=m[0],m[1]; px=(7-tc2)*SQ if fl else tc2*SQ; py=(7-tr2)*SQ if fl else
tr2*SQ
            hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA)
            if cb_b[tr2][tc2]: pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//2-2,5)
            else:              pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//6)
            s.blit(hl,(px,py))
        for i in range(8):
            ci=7-i if fl else i; ri=i if fl else 7-i
            lc=C_DARK_SQ if(7+i)%2==0 else C_LIGHT_SQ
            lr=C_DARK_SQ if i%2==1    else C_LIGHT_SQ
            lt=FNT_XS.render('abcdefgh'[ci],True,lc); s.blit(lt,(i*SQ+SQ-lt.get_width()-
3,BOARD_SIZE-lt.get_height()-2))
            ln=FNT_XS.render(str(ri+1),True,lr);      s.blit(ln,(3,i*SQ+2))

    def _d_pieces(self,s,board,fl=None):
```

```
        if board is None: return
        if fl is None: fl=self.flipped
        df=self.drag_fr
        for r in range(8):
            for c in range(8):
                if df and(r,c)==df: continue
                p=board[r][c]
                if not p: continue
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                draw_piece_at(s,p,px,py)

    def _d_drag(self,s):
        if self.drag_p and self.drag_pos:
            x,y=self.drag_pos; draw_piece_at(s,self.drag_p,x-SQ//2,y-SQ//2)

    def _d_result(self,s):
        ov=pygame.Surface((BOARD_SIZE,78),pygame.SRCALPHA); ov.fill((18,17,15,215))
        s.blit(ov,(0,BOARD_SIZE//2-39))
        r=self.cb.result
        msg,clr=("White wins",C_ACCENT) if r==WHITE else("Black wins",C_RED) if r==BLACK
else("Draw",C_TEXT2)
        tc(s,msg,FNT_LG,clr,BOARD_SIZE//2,BOARD_SIZE//2-11)
        tc(s,self.cb.result_reason.capitalize(),FNT_SM,C_TEXT2,BOARD_SIZE//2,BOARD_SIZE//2+16)

    # ── helpers for captured pieces ──────────────────────────────────────
    def _captured(self):
        """Return (white_captured, black_captured, advantage_color, adv_pts).
        white_captured = pieces white has taken (i.e. black pieces removed from board).
        black_captured = pieces black has taken (i.e. white pieces removed from board)."""
        PV = {'P':1,'N':3,'B':3,'R':5,'Q':9}
        start = {'P':8,'N':2,'B':2,'R':2,'Q':1}
        on_board = {}
        for r in range(8):
            for c in range(8):
                p = self.cb.board[r][c]
                if p: on_board[p] = on_board.get(p,0)+1
        # pieces white captured = missing black pieces
        w_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt.lower(),0)
```

```
            w_cap.extend([pt]*missing)
        # pieces black captured = missing white pieces
        b_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt,0)
            b_cap.extend([pt]*missing)
        ws = sum(PV.get(p,0) for p in w_cap)
        bs = sum(PV.get(p,0) for p in b_cap)
        adv = ws-bs
        return w_cap, b_cap, adv

    def _draw_caps(self, s, caps, score_adv, x, y, row_w):
        """Draw captured piece symbols in a compact row."""
        PV={'P':1,'N':3,'B':3,'R':5,'Q':9}
        # sort by value
        caps_sorted = sorted(caps, key=lambda p: PV.get(p,0))
        font, use_uni = _pfont(14)
        cx2 = x
        for p in caps_sorted:
            sym = UNICODE_SYMS.get(p.lower(),'?') if use_uni else p
            t = font.render(sym, True, C_TEXT2)
            s.blit(t,(cx2, y)); cx2 += t.get_width()+1
            if cx2 > x+row_w-20: break   # don't overflow
        if score_adv > 0:
            adv_t = FNT_XS.render(f"+{score_adv}", True, C_ACCENT)
            s.blit(adv_t,(cx2+4, y+1))

    # ── Panel ──────────────────────────────────────────────────────────
    def _d_panel(self,s):
        px0=BOARD_SIZE; PW=PANEL_WIDTH; W=PW-28; x=px0+14; mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,PW,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        y=10

        # ── Header: mode + ELO ──────────────────────────────────────────
        t2=TIME_CONTROLS[self.opt_tc]
        if self.mode=='bot':
            elo = StockfishAI.LEVEL_ELO.get(self.opt_ai, '?')
            lbl = f"vs Bot · Lv{self.opt_ai} · {elo} Elo · {t2['label']}"
        else:
```

```
            lbl = f"Local 2P · {t2['label']}"
        tc(s,lbl,FNT_XS,C_TEXT2,px0+PW//2,y+7); y+=18

        # ── Captured pieces + material advantage ──────────────────────────
        w_cap, b_cap, adv = self._captured()
        # adv > 0 → white ahead,  adv < 0 → black ahead

        # Determine which player is "opponent" (top) and "player" (bottom)
        # If playing as black: top=black(player), bottom=white(opponent)
        # Default / white / local: top=black(opponent), bottom=white(player)
        player_is_black = (self.mode=='bot' and self.player_col==BLACK)

        if player_is_black:
            # top = black (player), bottom = white (opponent)
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            top_label, bot_label = "Black (You)", "White (Bot)"
            top_cap,   bot_cap   = b_cap, w_cap   # black captured whites; white captured
blacks
            top_adv  = max(0,-adv)   # black advantage
            bot_adv  = max(0, adv)   # white advantage
        else:
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            if self.mode=='bot':
                top_label = "Black (Bot)"; bot_label = "White (You)"
            else:
                top_label = "Black"; bot_label = "White"
            top_cap,   bot_cap   = b_cap, w_cap
            top_adv  = max(0,-adv)
            bot_adv  = max(0, adv)

        # ── TOP player box ────────────────────────────────────────────────
        top_act = (self.cb.turn==top_color and self.clk_run and not self.cb.result)
        rr(s,(52,50,44) if top_act else C_PANEL2,(x,y,W,44),6,1,C_BORDER)
        tc(s,top_label,FNT_XS,C_TEXT2,x+W//2,y+10)
        tc(s,fmt(top_time),FNT_CLK,C_GOLD if top_act else C_TEXT2,x+W//2,y+30)
        y+=50

        # Pieces captured BY top player = opponent's pieces they took
```

```
        # top captures opponent pieces: if top=BLACK → took white pieces (uppercase)
        #                               if top=WHITE → took black pieces (lowercase)
        top_caps_disp = w_cap if top_color==BLACK else b_cap   # white pieces BLACK took /
black pieces WHITE took
        top_adv_pts   = max(0, -adv) if top_color==BLACK else max(0, adv)
        if top_caps_disp:
            self._draw_caps(s, top_caps_disp, top_adv_pts, x, y, W)
        y+=16

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8

        # ── Move list ──────────────────────────────────────────────────────
        hist=self.cb.history; ROW=17; max_vis=13
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''; i+=1
            pairs.append((len(pairs)+1,ws,bs))
        start_p=max(0,len(pairs)-max_vis); ml_y=y
        for rel,(mn,ws,bs) in enumerate(pairs[start_p:]):
            ry=ml_y+rel*ROW
            tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,x,ry)
            tl(s,ws,      FNT_MONO_SM,(225,220,205),x+30,ry)
            if bs: tl(s,bs,FNT_MONO_SM,(205,205,220),x+118,ry)
        y=ml_y+max_vis*ROW+4

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8
        if self.cb.is_check() and not self.cb.result:
            tc(s,"Check!",FNT_SMB,C_RED,px0+PW//2,y+8); y+=22
        if self.ai_busy:
            sf=isinstance(self.ai,StockfishAI) and self.ai.ready
            lbl2="Stockfish thinking..." if sf else "AI thinking..."
            tc(s,lbl2,FNT_SM,C_GOLD,px0+PW//2,y+8); y+=20
        if self.notif and time.time()<self.notif_exp:
            tc(s,self.notif,FNT_SM,C_ACCENT,px0+PW//2,y+8)

        # ── BOTTOM captured pieces ─────────────────────────────────────────
        bot_caps_disp = b_cap if bot_color==BLACK else w_cap   # symmetric
        bot_adv_pts   = max(0, -adv) if bot_color==BLACK else max(0, adv)
```

```
        bot_cap_y = 488
        if bot_caps_disp:
            self._draw_caps(s, bot_caps_disp, bot_adv_pts, x, bot_cap_y, W)

        # ── BOTTOM player box ──────────────────────────────────────────
        bot_act = (self.cb.turn==bot_color and self.clk_run and not self.cb.result)
        rr(s,(52,50,44) if bot_act else C_PANEL2,(x,506,W,44),6,1,C_BORDER)
        tc(s,bot_label,FNT_XS,C_TEXT2,x+W//2,506+10)
        tc(s,fmt(bot_time),FNT_CLK,C_GOLD if bot_act else C_TEXT,x+W//2,506+30)

        # ── Buttons ────────────────────────────────────────────────────
        for lbl2,bx,by,bw2,bh,c in[("Review",10,558,84,30,C_GOLD),
                                   ("Rematch",104,558,80,30,C_BTN),
                                   ("Menu",194,558,84,30,C_BTN)]:
            ax=px0+bx; hov=(ax<=mx<=ax+bw2 and by<=my<=by+bh)
            rr(s,C_BTN_H if hov else c,(ax,by,bw2,bh),5,1,C_BORDER)
            tc(s,lbl2,FNT_SM,C_TEXT,ax+bw2//2,by+bh//2)
        tc(s,"F:flip  R:review  ESC:menu",FNT_XS,C_TEXT3,px0+PW//2,WINDOW_H-10)

    def _d_promo(self):
        s=self.surf; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((BOARD_SIZE,BOARD_SIZE),pygame.SRCALPHA); ov.fill((0,0,0,168));
s.blit(ov,(0,0))
        rr(s,(46,44,38),(cx-125,cy-60,250,118),12,1,C_BORDER)
        tc(s,"Promote to:",FNT_MD,C_TEXT2,cx,cy-42)
        for i,p in enumerate(['Q','R','B','N']):
            piece=p if self.cb.turn==WHITE else p.lower()
            bx=cx-105+i*54; by=cy-22; hov=(bx<=mx<=bx+52 and by<=my<=by+50)
            rr(s,C_BTN_H if hov else C_BTN,(bx,by,52,50),7,1,C_BORDER)
            draw_piece_at(s,piece,bx,by,50)

    # ── Review ─────────────────────────────────────────────────────────
    # Layout in review mode:
    #   x=0..18   : eval bar (white=bottom, black=top)
    #   x=18..658 : chess board (640px)  → board offset = 18
    #   x=658..958: panel (300px)
    EVAL_BAR_W  = 18
    BOARD_OX    = 18   # board x-offset in review mode

    def _d_rev(self):
```

```
        s=self.surf; idx=self.rev_idx
        board=self.rev_boards[idx] if idx<len(self.rev_boards) else (self.rev_boards[-1] if
self.rev_boards else None)
        lm=None
        if idx>0: m=self.rev_hist[idx-1]['move']; lm=(m['from'],m['to'])

        # ── Eval bar (left strip) ──────────────────────────────────────
        self._d_eval_bar(s)

        # ── Board (shifted right by EVAL_BAR_W) ────────────────────────
        # Draw squares
        fl=self.flipped
        chk_sq=None  # no check highlight in review
        for r in range(8):
            for c in range(8):
                px=self.BOARD_OX+((7-c)*SQ if fl else c*SQ)
                py=(7-r)*SQ if fl else r*SQ
                base=C_LIGHT_SQ if(r+c)%2==0 else C_DARK_SQ
                pygame.draw.rect(s,base,(px,py,SQ,SQ))
                if lm and ((r,c)==lm[0] or (r,c)==lm[1]):
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_HIGHLIGHT,170));
s.blit(hl,(px,py))
        # Classification highlight on destination square
        if idx>0:
            clf=self.rev_analyser.get_classif(idx) if self.rev_analyser else 'none'
            clf_clr,_,_=CLF_INFO.get(clf,CLF_INFO['none'])
            tr2,tc3=lm[1]
            px2=self.BOARD_OX+((7-tc3)*SQ if fl else tc3*SQ)
            py2=(7-tr2)*SQ if fl else tr2*SQ
            hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*clf_clr,90));
s.blit(hl,(px2,py2))
            # Badge circle on corner
            bx3=px2+(SQ-18 if not fl else 0); by3=py2
            pygame.draw.circle(s,clf_clr,(bx3+9,by3+9),9)
            sym=CLF_INFO.get(clf,('','',''))[1]
            if sym:
                f2,_=_pfont(10); st=f2.render(sym,True,(255,255,255))
                s.blit(st,(bx3+9-st.get_width()//2,by3+9-st.get_height()//2))
        # Coordinates
        for i in range(8):
```

```
            ci=7-i if fl else i; ri=i if fl else 7-i
            lc=C_DARK_SQ if(7+i)%2==0 else C_LIGHT_SQ
            lr=C_DARK_SQ if i%2==1    else C_LIGHT_SQ
            lt=FNT_XS.render('abcdefgh'[ci],True,lc)
            s.blit(lt,(self.BOARD_OX+i*SQ+SQ-lt.get_width()-3,BOARD_SIZE-lt.get_height()-2))
            ln=FNT_XS.render(str(ri+1),True,lr)
            s.blit(ln,(self.BOARD_OX+3,i*SQ+2))
        # Pieces
        if board:
            for r in range(8):
                for c in range(8):
                    p=board[r][c]
                    if not p: continue
                    px=self.BOARD_OX+((7-c)*SQ if fl else c*SQ)
                    py=(7-r)*SQ if fl else r*SQ
                    draw_piece_at(s,p,px,py)

        self._d_rev_nav(s)
        self._d_rev_panel(s)

    def _d_eval_bar(self,s):
        """Draw a vertical eval bar on the left edge."""
        ra=self.rev_analyser; idx=self.rev_idx
        ev=None
        if ra: ev=ra.get_eval(idx)
        BW=self.EVAL_BAR_W; H=BOARD_SIZE
        # Background
        pygame.draw.rect(s,(30,28,24),(0,0,BW,H))
        # Convert eval to 0..1 (white fraction of bar, measured from BOTTOM)
        if ev is None:
            wfrac=0.5
        else:
            # clamp to ±1000cp, sigmoid-ish
            clamped=max(-1000,min(1000,ev))
            wfrac=0.5+clamped/2000.0
        wfrac=max(0.05,min(0.95,wfrac))
        black_h=int(H*(1-wfrac)); white_h=H-black_h
        # Black portion (top)
        pygame.draw.rect(s,(30,30,30),(0,0,BW,black_h))
        # White portion (bottom)
```

```
        pygame.draw.rect(s,(220,215,200),(0,black_h,BW,white_h))
        # Midline
        pygame.draw.line(s,(80,78,72),(0,H//2),(BW,H//2),1)
        # Score text
        if ev is not None:
            if abs(ev)>=29000: txt="M" if ev>0 else "-M"
            else: txt=f"{ev/100:+.1f}" if abs(ev)<1000 else f"{ev/100:+.0f}"
            fs=10; f2,_=_pfont(fs)
            st=FNT_XS.render(txt,True,(200,200,200) if abs(ev)<50 else (C_ACCENT if ev>0 else
C_RED))
            # rotate 90° for vertical text
            st_r=pygame.transform.rotate(st,90)
            s.blit(st_r,(BW//2-st_r.get_width()//2, H//2-st_r.get_height()//2))
        # "Analysing..." if not done
        if ra and not ra.done:
            dot_y=H-24
            dt=FNT_XS.render("…",True,C_GOLD)
            s.blit(dt,(BW//2-dt.get_width()//2,dot_y))

    def _d_rev_nav(self,s):
        ox=self.BOARD_OX; cx=ox+BOARD_SIZE//2; by0=BOARD_SIZE-46; mx,my=pygame.mouse.get_pos()
        for bx,by,bw,bh,lbl in[(cx-132,by0,40,36,'|<'),(cx-84,by0,40,36,'<'),
                               (cx+44, by0,40,36,'>'),(cx+92,by0,40,36,'|>')]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            rr(s,C_BTN_H if hov else C_PANEL3,(bx,by,bw,bh),6,1,C_BORDER)
            tc(s,lbl,FNT_MDB,C_TEXT,bx+bw//2,by+bh//2)
        tc(s,f"{self.rev_idx} / {len(self.rev_hist)}",FNT_SM,C_TEXT2,cx,by0+18)
        tc(s,"Arrow keys or buttons to navigate",FNT_XS,C_TEXT3,cx,BOARD_SIZE-7)

    def _d_rev_panel(self,s):
        # Panel starts at BOARD_OX + BOARD_SIZE
        px0=self.BOARD_OX+BOARD_SIZE; PW=PANEL_WIDTH-(self.BOARD_OX); x=px0+14
        mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,WINDOW_W-px0,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        W=WINDOW_W-px0-28; y=14
        tc(s,"Game Review",FNT_LG,C_GOLD,px0+(WINDOW_W-px0)//2,y+10); y+=34

        # ── Analysis status / current move feedback ──────────────────────
        ra=self.rev_analyser; idx=self.rev_idx
```

```
        if idx>0 and ra:
            clf=ra.get_classif(idx)
            clf_clr,sym,desc=CLF_INFO.get(clf,CLF_INFO['none'])
            if clf!='none':
                rr(s,(*clf_clr,40),(x,y,W,38),6)  # won't work with alpha directly

pygame.draw.rect(s,(clf_clr[0]//4,clf_clr[1]//4,clf_clr[2]//4),(x,y,W,38),border_radius=6)
                pygame.draw.rect(s,clf_clr,(x,y,W,38),1,border_radius=6)
                tc(s,f"{sym}  {clf.upper()}",FNT_SMB,clf_clr,x+W//2,y+12)
                tc(s,desc.split('—')[1].strip() if '—' in desc else
desc,FNT_XS,C_TEXT2,x+W//2,y+27)
                y+=46
        elif ra and not ra.done:
            tc(s,"Analysing game...",FNT_SM,C_GOLD,px0+(WINDOW_W-px0)//2,y+10); y+=24
        else:
            y+=4

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8

        # ── Paired move list with classification badges ──────────────────────
        hist=self.rev_hist; cur=self.rev_idx; ROW=18; max_vis=16
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; wclf=ra.get_classif(i+1) if ra else 'none'; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''
            bclf=ra.get_classif(i+1) if (ra and i<len(hist)) else 'none'; i+=1
            pairs.append((len(pairs)+1, ws,wclf, bs,bclf))
        cur_row=(cur-1)//2 if cur>0 else 0
        start=max(0,cur_row-max_vis//2); end=min(len(pairs),start+max_vis); start=max(0,end-
max_vis)

        COL_NUM=x; COL_W=x+28; COL_B=x+W//2+4
        for rel,(mn,ws,wclf,bs,bclf) in enumerate(pairs[start:end]):
            pi=start+rel; ry=y+rel*ROW
            w_act=(cur==pi*2+1); b_act=(cur==pi*2+2)
            if w_act: pygame.draw.rect(s,(68,65,48),(px0+4,ry-1,W//2,ROW),border_radius=3)
            if b_act: pygame.draw.rect(s,(68,65,48),(px0+4+W//2,ry-
1,W//2,ROW),border_radius=3)
            tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,COL_NUM,ry)
```

```
            # White move + badge
            wclr_b,wsym,_=CLF_INFO.get(wclf,CLF_INFO['none'])
            wclr=C_GOLD if w_act else (wclr_b if wclf not in('none','best') else
(225,220,205))
            tl(s,ws,FNT_MONO_SM,wclr,COL_W,ry)
            if wsym and wclf not in('none',):
                bt=FNT_XS.render(wsym,True,wclr_b)
                s.blit(bt,(COL_W+40,ry+1))
            # Black move + badge
            if bs:
                bclr_b,bsym,_=CLF_INFO.get(bclf,CLF_INFO['none'])
                bclr=C_GOLD if b_act else (bclr_b if bclf not in('none','best') else
(205,205,220))
                tl(s,bs,FNT_MONO_SM,bclr,COL_B,ry)
                if bsym and bclf not in('none',):
                    bt2=FNT_XS.render(bsym,True,bclr_b)
                    s.blit(bt2,(COL_B+40,ry+1))

        bx,by2,bw,bh=px0+6,558,WINDOW_W-px0-12,30
        hov=(bx<=mx<=bx+bw and by2<=my<=by2+bh)
        rr(s,C_BTN_H if hov else C_BTN,(bx,by2,bw,bh),5,1,C_BORDER)
        back="< Back to Game" if self.rev_src==S.PLAYING else "< Back to Records"
        tc(s,back,FNT_SM,C_TEXT,px0+(WINDOW_W-px0)//2,by2+bh//2)

    # ── History screen ──────────────────────────────────────────────
    def _d_hist(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Game Records",FNT_LG,C_TEXT,cx,45)
        pygame.draw.line(s,C_BORDER,(40,70),(WINDOW_W-40,70),1)
        recs=self.hist_recs
        if not recs: tc(s,"No games recorded yet.",FNT_MD,C_TEXT2,cx,WINDOW_H//2)
        else:
            item_h=70; list_y0=88; vis=min(7,(WINDOW_H-170)//item_h)
            for rel in range(vis):
                idx=rel+self.hist_scroll
                if idx>=len(recs): break
                rec=recs[-(idx+1)]; ry=list_y0+rel*item_h
                hov=(30<=mx<=WINDOW_W-30 and ry<=my<=ry+item_h-3)
                bg=C_PANEL3 if hov else(C_PANEL2 if rel%2==0 else C_PANEL)
                rr(s,bg,(30,ry,WINDOW_W-60,item_h-4),6,1,C_BORDER)
```

```
                res=rec.get('result','?')
                rclr=C_ACCENT if'White' in res else(C_RED if'Black' in res else C_TEXT2)
                tl(s,res,FNT_MDB,rclr,50,ry+6)
                reason=rec.get('reason','').capitalize()
                if reason: tl(s,f"by {reason}",FNT_XS,C_TEXT2,50,ry+24)
                mode='vs Bot' if rec.get('mode')=='bot' else'Local 2P'
                lvl=f" · Lv{rec.get('ai_level','?')}" if rec.get('mode')=='bot' else''
                tl(s,f"{mode}{lvl}  ·  {rec.get('tc','?')}",FNT_SM,C_TEXT,230,ry+8)
                tl(s,f"{rec.get('total_moves','?')} moves",FNT_XS,C_TEXT2,230,ry+26)
                tl(s,rec.get('date',''),FNT_XS,C_TEXT3,WINDOW_W-215,ry+8)
                moves=rec.get('moves',[]); prev=' '.join(f"{(i//2)+1}.{m}" if i%2==0 else m
for i,m in enumerate(moves[:10]))
                if len(moves)>10: prev+='...'
                tl(s,prev,FNT_XS,C_TEXT3,50,ry+44)
                if hov: tc(s,"Click to review",FNT_XS,C_GOLD,WINDOW_W-90,ry+item_h//2-4)
            total=len(recs)
            if total>vis:
                area=WINDOW_H-170; sbh=max(20,int(vis/total*area))
                sby=list_y0+int(self.hist_scroll/max(1,total-vis)*(area-sbh))
                pygame.draw.rect(s,C_PANEL3,(WINDOW_W-14,list_y0,6,area))
                pygame.draw.rect(s,C_TEXT2,(WINDOW_W-14,sby,6,sbh),border_radius=3)
        bbx,bby,bbw,bbh=cx-80,WINDOW_H-48,160,34; hov=(bbx<=mx<=bbx+bbw and bby<=my<=bby+bbh)
        rr(s,C_BTN_H if hov else C_BTN,(bbx,bby,bbw,bbh),6,1,C_BORDER)
        tc(s,"< Main Menu",FNT_SM,C_TEXT,cx,bby+bbh//2)

    # FIX #5 — confirm overlay
    def _d_hconf(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((0,0,0,155));
s.blit(ov,(0,0))
        rr(s,(46,44,38),(cx-185,cy-80,370,170),12,1,C_BORDER)
        tc(s,"Review this game?",FNT_LG,C_TEXT,cx,cy-52)
        if self.hist_confirm is not None and self.hist_recs:
            rec=self.hist_recs[-(self.hist_confirm+1)]
            info=f"{rec.get('result','?')}  ·  {rec.get('total_moves','?')} moves  ·
{rec.get('date','')}"
            tc(s,info,FNT_SM,C_TEXT2,cx,cy-22)
        for lbl2,bx,c in[("Review >",cx-120,C_BLUE),("Cancel",cx+10,C_BTN)]:
            bw2,bh2=110,46; by=cy+18; hov=(bx<=mx<=bx+bw2 and by<=my<=by+bh2)
            rr(s,tuple(min(255,v+20) for v in c) if hov else c,(bx,by,bw2,bh2),8,1,C_BORDER)
```

```
            tc(s,lbl2,FNT_MDB,C_TEXT,bx+bw2//2,by+bh2//2)

# ═══════════════════════════════════════════════════════════════════════════
if __name__ == '__main__':
    Game().run()
```