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]")
```

Revision #1
Created 2026-03-18 16:23:11 UTC by Samuel Lee
Updated 2026-03-18 16:23:31 UTC by Samuel Lee