

```

from game_cli import main_game_loop

if __name__ == "__main__":
    main_game_loop()
    这是主程序
import pickle
from game_board import GameBoard
from units import create_units_for_player, Unit
from game_mechanics import handle_player_turn, check_win_condition

def main_game_loop():
    """
    主游戏循环 (命令行界面版本).
    """
    board = GameBoard() # 创建棋盘

    # 选择种族 (简化为直接指定，可以扩展为让玩家选择)
    player1_race = "鼠族"
    player2_race = "牛族"

    units_player1 = create_units_for_player(player_num=1, race_name=player1_race)
    units_player2 = create_units_for_player(player_num=2, race_name=player2_race)
    units = units_player1 + units_player2 # 将所有单位放在一个列表中方便管理

    # 初始单位部署
    initial_positions_player1 = [(0, 0), (0, board.size - 1)] # 玩家 1 单位初始位置 (棋盘上方边缘)
    initial_positions_player2 = [(board.size - 1, 0), (board.size - 1, board.size - 1)] # 玩家 2 单位初始位置 (棋盘下方边缘)

    for i in range(len(units_player1)):
        units_player1[i].location = initial_positions_player1[i]
    for i in range(len(units_player2)):
        units_player2[i].location = initial_positions_player2[i]

    player_resources = [0, 0] # 初始化玩家资源
    current_player = 1 # 玩家 1 先行

```

```

# 加载保存的游戏（可选）
load_game = input("是否加载上次的游戏进度？(y/n): ")
if load_game.lower() == 'y':
    try:
        with open("game_save.pkl", "rb") as f:
            saved_data = pickle.load(f)
            board.board = saved_data['board']
            units = saved_data['units']
            player_resources = saved_data['player_resources']
            current_player = saved_data['current_player']
            print("游戏进度已加载!")
    except FileNotFoundError:
        print("没有找到保存的游戏进度，从新游戏开始。")

while True:
    handle_player_turn(current_player, units, board, player_resources)

    winner = check_win_condition(player_resources, units, board)
    if winner:
        print(f"\n 恭喜! 玩家 {winner} 获胜!")
        break

# 保存游戏
save_game = input("是否保存游戏进度？(y/n): ")
if save_game.lower() == 'y':
    game_state = {
        'board': board.board,
        'units': units,
        'player_resources': player_resources,
        'current_player': current_player
    }
    with open("game_save.pkl", "wb") as f:
        pickle.dump(game_state, f)
    print("游戏进度已保存!")

# 切换到下一个玩家
current_player = 3 - current_player # 玩家 1 -> 2, 玩家 2 -> 1

```

```
print("\n--- 游戏结束 ---")
```

```
if __name__ == "__main__":
```

```
    main_game_loop()
```

```
    这是 game cli.py
```

```
    class Unit:
```

```
        def __init__(self, race, unit_type, symbol, attack, defense, movement, economy):
```

```
            self.race = race
```

```
            self.unit_type = unit_type
```

```
            self.symbol = symbol
```

```
            self.attack = attack
```

```
            self.defense = defense
```

```
            self.movement = movement
```

```
            self.economy = economy
```

```
            self.location = None
```

```
            self.player = None
```

```
            self.is_defeated = False
```

```
        def move(self, board, direction=None, new_position=None):
```

```
            """
```

```
            移动单位到新的位置，支持坐标和方向移动。
```

```
            """
```

```
            if new_position is not None: # 坐标移动
```

```
            if not board.is_valid_position(new_position):
```

```
                print("无效的移动位置，超出棋盘边界！")
```

```
            return False
```

```
            current_row, current_col = self.location
```

```
            new_row, new_col = new_position
```

```
            distance = abs(new_row - current_row) + abs(new_col - current_col)
```

```
            if distance > self.movement:
```

```
                print(f"移动距离 {distance} 超过单位移动力 {self.movement} ! ")
```

```
            return False
```

```
            self.location = new_position
```

```
            return True
```

```
        if direction and self.location: # 方向移动
```

```

row, col = self.location
if direction == "上" and row > 0:
    new_position = (row - 1, col)
elif direction == "下" and row < board.size - 1:
    new_position = (row + 1, col)
elif direction == "左" and col > 0:
    new_position = (row, col - 1)
elif direction == "右" and col < board.size - 1:
    new_position = (row, col + 1)
else:
    print("无法朝该方向移动，已到棋盘边缘！")
    return False
self.location = new_position
return True
return False

```

```

def attack_unit(self, target_unit):

```

```

    """

```

攻击目标单位。简化战斗逻辑：攻击力 > 防御力 则击败对方。

```

    """

```

```

if self.attack > target_unit.defense:

```

```

    print(f"{self.race} {self.unit_type} 成功击败 {target_unit.race}
    {target_unit.unit_type}!")

```

```

    target_unit.is_defeated = True

```

```

    target_unit.location = None # 被击败单位移除

```

```

    return True

```

```

else:

```

```

    print(f"{self.race} {self.unit_type} 攻击 {target_unit.race} {target_unit.unit_type} 失
    败!")

```

```

    return False

```

```

def create_units_for_player(player_num, race_name):

```

```

    """

```

为玩家创建一组单位。

```

    """

```

```

    units = []

```

```

    if race_name == "鼠族":

```

```

        units.append(Unit(race="鼠族", unit_type="经济鼠", symbol="鼠", attack=2,

```

```
defense=2, movement=2, economy=3))
elif race_name == "牛族":
    units.append(Unit(race="牛族", unit_type="战斗牛", symbol="牛", attack=4,
defense=3, movement=1, economy=1))
elif race_name == "虎族":
    units.append(Unit(race="虎族", unit_type="掠夺虎", symbol="虎", attack=3,
defense=1, movement=3, economy=0))
elif race_name == "兔族":
    units.append(Unit(race="兔族", unit_type="独立兔", symbol="兔", attack=2,
defense=2, movement=2, economy=2))
elif race_name == "龙族":
    units.append(Unit(race="龙族", unit_type="辅助龙", symbol="龙", attack=1,
defense=3, movement=2, economy=2))
elif race_name == "蛇族":
    units.append(Unit(race="蛇族", unit_type="游击蛇", symbol="蛇", attack=3,
defense=1, movement=2, economy=1))
elif race_name == "马族":
    units.append(Unit(race="马族", unit_type="贸易马", symbol="马", attack=1,
defense=1, movement=3, economy=3))
elif race_name == "羊族":
    units.append(Unit(race="羊族", unit_type="交易羊", symbol="羊", attack=2,
defense=2, movement=2, economy=2))
elif race_name == "猴族":
    units.append(Unit(race="猴族", unit_type="情报猴", symbol="猴", attack=1,
defense=1, movement=2, economy=2))
elif race_name == "鸡族":
    units.append(Unit(race="鸡族", unit_type="支援鸡", symbol="鸡", attack=1,
defense=2, movement=2, economy=2))
elif race_name == "狗族":
    units.append(Unit(race="狗族", unit_type="武器狗", symbol="狗", attack=5,
defense=2, movement=1, economy=1))
elif race_name == "猪族":
    units.append(Unit(race="猪族", unit_type="科技猪", symbol="猪", attack=2,
defense=3, movement=1, economy=2))
else:
    raise ValueError(f"未知的种族名称: {race_name}")

for unit in units:
```

```

unit.player = player_num # 设置单位所属玩家

return units
这是 uniti.py
def collect_resources(player, units, board):
    """
    玩家收集资源，简化为每个经济单位在其位置产出固定资源。
    """
    total_resources = 0
    for unit in units:
        if unit.player == player and unit.unit_type.startswith("经济") and unit.location:
            terrain_type = board.board[unit.location[0]][unit.location[1]]
            if terrain_type == "平原":
                total_resources += unit.economy
            else:
                total_resources += unit.economy // 2 # 非平原地形资源减半
    return total_resources

def handle_player_turn(player_num, units, board, player_resources):
    """
    处理玩家回合：允许移动和攻击，并显示单位详情。
    """
    player_units = [unit for unit in units if unit.player == player_num and not
                    unit.is_defeated]

    if not player_units:
        print(f"玩家 {player_num} 没有可操作的单位了!")
        return

    print(f"\n--- 玩家 {player_num} 回合 ---")
    board.display_board(units) # 显示当前棋盘

    player_resources[player_num - 1] += collect_resources(player_num, player_units,
    board)
    print(f"玩家 {player_num} 获得资源，当前资源: {player_resources[player_num -
    1]}")

# 显示玩家单位列表

```

```

print("\n 你的单位:")
for i, unit in enumerate(player_units):
    print(f"{i}. {unit.race} {unit.unit_type} (符号: {unit.symbol}) - 位置: {unit.location},
    攻击: {unit.attack}, 防御: {unit.defense}, 移动: {unit.movement}, 经济:
    {unit.economy}")

while True:
    print("\n 请选择操作 (输入数字):")
    print("1. 移动单位")
    print("2. 攻击单位")
    print("3. 结束回合")

    choice = input("请选择: ")

    if choice == '1':
        unit_index_str = input(f"选择要移动的单位 (输入 0-{len(player_units)-1}): ")
        try:
            unit_index = int(unit_index_str)
            if 0 <= unit_index < len(player_units):
                selected_unit = player_units[unit_index]
                print(f"你选择了 {selected_unit.race} {selected_unit.unit_type} (符号:
                {selected_unit.symbol}), 当前位置: {selected_unit.location}")
                move_type = input("选择移动方式 (1: 输入坐标, 2: 方向移动): ")
                if move_type == "1":
                    new_row_str = input(f"输入新的行坐标 (0-{board.size-1}): ")
                    new_col_str = input(f"输入新的列坐标 (0-{board.size-1}): ")
                    new_row, new_col = int(new_row_str), int(new_col_str)
                    if selected_unit.move(board, new_position=(new_row, new_col)):
                        print("单位移动成功!")
                    else:
                        print("单位移动失败。")
                elif move_type == "2":
                    direction = input("输入移动方向 (上/下/左/右): ")
                    if selected_unit.move(board, direction=direction):
                        print("单位移动成功!")
                    else:
                        print("单位移动失败。")
                else:
                    print("单位移动失败。")
            else:
                print("单位移动失败。")
        except ValueError:
            print("单位移动失败。")
    else:
        print("单位移动失败。")
    else:
        print("单位移动失败。")

```

```

print("无效的移动方式!")
else:
print("无效的单位索引!")
except ValueError:
print("输入无效，请输入数字或正确方向!")

elif choice == '2':
attacker_unit_index_str = input(f"选择攻击单位 (输入 0-{len(player_units)-1}): ")
try:
attacker_unit_index = int(attacker_unit_index_str)
if 0 <= attacker_unit_index < len(player_units):
attacker_unit = player_units[attacker_unit_index]
print(f"你选择了 {attacker_unit.race} {attacker_unit.unit_type} (符号: {attacker_unit.symbol}), 当前位置: {attacker_unit.location}")

# 显示可攻击的目标
enemy_units = [u for u in units if u.player != player_num and not u.is_defeated and u.location]
if not enemy_units:
print("没有可攻击的敌方单位!")
continue
print("\n 可攻击的目标:")
for i, unit in enumerate(enemy_units):
print(f"{i}. {unit.race} {unit.unit_type} (符号: {unit.symbol}) - 位置: {unit.location}, 防御: {unit.defense}")

target_unit_index_str = input(f"选择目标单位 (输入 0-{len(enemy_units)-1}): ")
target_unit_index = int(target_unit_index_str)
if 0 <= target_unit_index < len(enemy_units):
target_unit = enemy_units[target_unit_index]
attacker_unit.attack_unit(target_unit)
else:
print("无效的目标单位索引!")
else:
print("无效的攻击单位索引!")
except ValueError:
print("输入无效，请输入数字!")

```



```

elif choice == '3':
    print(f"玩家 {player_num} 结束回合。")
    print(f"回合摘要 - 玩家 {player_num}:")
    print(f"资源: {player_resources[player_num - 1]}")
    print(f"存活单位数量: {len(player_units)}")
    board.display_board(units) # 显示最新棋盘状态
    break
else:
    print("无效的选择，请重新输入 1, 2, 或 3。")

def check_win_condition(player_resources, units, board):
    """
    检查获胜条件：先达到一定资源数量的玩家获胜。
    """
    resource_win_limit = 200 # 资源胜利限制

    for i in range(len(player_resources)):
        if player_resources[i] >= resource_win_limit:
            return i + 1 # 返回获胜玩家编号 (1 or 2)

    return None # 没有玩家获胜

这是 game 机器.py
class GameBoard:
    def __init__(self, size=5): # 默认棋盘大小为 5x5，可以调整
        self.size = size
        self.board = self.create_board()

    def create_board(self):
        """
        创建棋盘，初始地形可以随机生成或者预设。
        这里为了简化，先都设置为 '平原'，可以后续扩展地形类型。
        """
        return [['平原' for _ in range(self.size)] for _ in range(self.size)]

    def display_board(self, units):
        """
        显示棋盘状态，包括地形和单位位置，带行列编号。
        """

```

```

print("----- 宇宙棋盘 -----")
# 打印列号
print(" ", end="")
for c in range(self.size):
    print(f" {c} ", end="")
print()

for r in range(self.size):
    row_str = f"{{r}} "
    for c in range(self.size):
        unit_symbol = " " # 默认空格
        for unit in units:
            if unit.location == (r, c):
                unit_symbol = unit.symbol # 使用单位的符号来显示
        break
    row_str += f"{{self.board[r][c][0]}}{{unit_symbol}}" # 显示地形首字母和单位符号
    print(row_str)
    print("-----")

def is_valid_position(self, position):
    """
    检查位置是否在棋盘内。
    """
    row, col = position
    return 0 <= row < self.size and 0 <= col < self.size

```