

```

import random

# --- 卡牌定義 ---
class Card:
    def __init__(self, suit, rank, card_type, function=None):
        self.suit = suit # 花色 (梅花, 方块, 红心, 黑桃)
        self.rank = rank # 牌面 (A, 2, 3, ..., 10, J, Q, K)
        self.card_type = card_type # 牌類型 (function, point, wild)
        self.function = function # 功能牌的特殊功能 (function 名稱字串)

    def __str__(self):
        return f"{self.suit}{self.rank}"

# --- 玩家定義 ---
class Player:
    def __init__(self, name):
        self.name = name
        self.hand = [] # 手牌列表
        self.score_area = [] # 計分區 (打出的點數牌)
        self.score = 0 # 總分

    def draw_card(self, deck):
        card = deck.draw_card()
        if card:
            self.hand.append(card)

    def play_card(self, card_index, game):
        card = self.hand.pop(card_index)
        if card.card_type == 'point':
            self.score_area.append(card)
            print(f"{self.name} 打出了點數牌: {card}")
        elif card.card_type == 'function':
            print(f"{self.name} 打出了功能牌: {card}，發動功能: {card.function}")
            self.activate_function(card, game) # 執行功能牌效果
        elif card.card_type == 'wild':
            print(f"{self.name} 打出了百搭牌: {card} (百搭牌功能尚未完全實作)") # 百搭牌功能需要更詳細的規則來實作

```

```

        # 百搭牌的處理邏輯需要根據具體規則進一步完善
    else:
        print(f"未知的卡牌類型: {card}")

    def activate_function(self, card, game):
        function_name = card.function
        if function_name == 'steal_card': # 鼠 - 偷牌換牌
            self.function_steal_card(game)
        elif function_name == 'defense_shield': # 牛 - 防禦護盾 (實作略，通常是標記自己或卡牌獲得保護狀態)
            self.function_defense_shield(game)
        elif function_name == 'remove_points': # 虎 - 移除點數
            self.function_remove_points(game)
        elif function_name == 'preview_deck': # 兔 - 預見未來
            self.function_preview_deck(game)
        elif function_name == 'card_exchange': # 龍 - 卡牌交換
            self.function_card_exchange(game)
        elif function_name == 'ambush_score': # 蛇 - 埋伏得分 (實作略，需要延遲計分機制)
            self.function_ambush_score(game)
        elif function_name == 'extra_draw': # 馬 - 疾速抽牌
            self.function_extra_draw(game)
        elif function_name == 'lucky_draw': # 羊 - 福澤綿羊
            self.function_lucky_draw(game)
        elif function_name == 'copy_ability': # 猴/雞 - 能力複製 (實作複雜，需要記錄上個玩家的行動)
            self.function_copy_ability(game)
        elif function_name == 'loyalty_bonus': # 狗 - 忠誠護衛 (通常是計分時的加成)
            self.function_loyalty_bonus(game)
        elif function_name == 'furious_charge': # 豬 - 狂暴衝鋒 (實作略，允許額外行動)
            self.function_furious_charge(game)
        else:
            print(f"未知的卡牌功能: {function_name}")

    # --- 以下是功能牌的具體效果實作 (需要完善) ---
    def function_steal_card(self, game): # 鼠

```

```

target_player = game.get_opponent_player(self)
if target_player and target_player.hand:
    discarded_card_index = random.randint(0, len(target_player.hand) - 1)
    discarded_card = target_player.hand.pop(discarded_card_index)
    self.draw_card(game.deck) # 作為簡化，偷牌後自己抽一張
    print(f"{self.name} 對 {target_player.name} 使用了偷牌，
{target_player.name} 隨機失去一張手牌。")
else:
    print(f"{self.name} 使用偷牌失敗，沒有目標或目標沒有手牌。")

def function_defense_shield(self, game): # 牛 - 防禦護盾
    """
    效果：保護自己的一張已出牌或手牌，使其在本回合或下回合免受指
    定負面效果影響。
    這裡簡化為保護一張展示區的點數牌一回合不受事件卡或虎牌的效果
    影響。
    """
    if not self.score_area:
        print(f"{self.name} 發動防禦護盾失敗，你沒有展示區點數牌可以
        保護。")
        return

    card_choices = [f"{i}: {card}" for i, card in enumerate(self.score_area)]
    card_choice_str = ", ".join(card_choices)
    while True:
        card_index_input = input(f"{self.name}, 選擇要保護的展示區點數牌
        索引 ({card_choice_str}, -1: 取消): ")
        if card_index_input == '-1':
            print("取消防禦護盾。")
            return
        try:
            card_index = int(card_index_input)
            if 0 <= card_index < len(self.score_area):
                protected_card = self.score_area[card_index]
                protected_card.is_protected = True # 為卡牌添加保護標記
                print(f"{self.name} 保護了展示區的點數牌:
                {protected_card}，使其本回合免受影響。")
                break

```

```

        else:
            print("無效的索引，請重新選擇。")
    except ValueError:
        print("請輸入有效的數字索引。")

def function_remove_points(self, game): # 虎
    target_player = game.get_opponent_player(self)
    if target_player:
        points_to_remove = 2 # 可以調整移除的分數
        target_player.score -= points_to_remove
        if target_player.score < 0:
            target_player.score = 0
        print(f"{self.name} 對 {target_player.name} 使用了移除點數，減少 {points_to_remove} 分。")
    else:
        print(f"{self.name} 使用移除點數失敗，沒有目標。")

def function_preview_deck(self, game): # 兔
    preview_count = 2
    previewed_cards = game.deck.cards[:preview_count] # 偷看牌堆頂的牌
    preview_str = ", ".join(str(card) for card in previewed_cards)
    print(f"{self.name} 使用了預見未來，偷看了牌堆頂的 {preview_count} 張牌: {preview_str}")
    # 注意：這裡只是展示，實際預覽後牌堆順序不變，也沒做任何操作

def function_card_exchange(self, game): # 龍
    target_player = game.get_opponent_player(self)
    if target_player and self.hand and target_player.hand:
        my_card_index = int(input(f"{self.name}, 選擇你要交換的手牌 (0-{len(self.hand)-1}): "))
        target_card_index = int(input(f"{self.name}, 選擇要交換 {target_player.name} 的手牌 (0-{len(target_player.hand)-1}): "))
        if 0 <= my_card_index < len(self.hand) and 0 <= target_card_index < len(target_player.hand):
            my_card = self.hand[my_card_index]
            target_card = target_player.hand[target_card_index]
            self.hand[my_card_index] = target_card
            target_player.hand[target_card_index] = my_card

```

```

        print(f"{self.name} 和 {target_player.name} 交換了手牌。")
    else:
        print(f"無效的手牌索引，交換失敗。")
    else:
        print(f"{self.name} 使用卡牌交換失敗，沒有目標或自己/目標沒有手牌。")

```

```

def function_ambush_score(self, game): # 蛇 - 埋伏得分
    """
    效果：隱藏一張手牌作為埋伏牌，在遊戲結束時，若手牌數量少於等於 2 張，則獲得額外分數。
    """

```

```

    if not self.hand:
        print(f"{self.name} 發動埋伏得分失敗，你沒有手牌可以埋伏。")
        return

```

```

    card_choices = [f"{i}: {card}" for i, card in enumerate(self.hand)]
    card_choice_str = ", ".join(card_choices)

```

```

    while True:
        card_index_input = input(f"{self.name}, 選擇要埋伏的手牌索引 ({card_choice_str}, -1: 取消): ")
        if card_index_input == '-1':
            print("取消埋伏得分。")
            return
        try:
            card_index = int(card_index_input)
            if 0 <= card_index < len(self.hand):
                ambush_card = self.hand.pop(card_index)
                self.ambush_card = ambush_card # 玩家記錄埋伏的牌
                print(f"{self.name} 埋伏了一張手牌: {ambush_card}。將在遊戲結束時根據條件計分。")
                break
            else:
                print("無效的索引，請重新選擇。")
        except ValueError:
            print("請輸入有效的數字索引。")

```

```

def function_extra_draw(self, game): # 馬
    self.draw_card(game.deck)
    print(f"{self.name} 使用了疾速抽牌，額外抽了一張牌。")

def function_lucky_draw(self, game): # 羊
    cards = game.deck.draw_cards(3) # 抽取三張牌
    if cards:
        print(f"{self.name} 使用了福澤綿羊，翻開了 {', '.join(map(str,
cards)))}")
        chosen_index = int(input(f"{self.name}, 選擇一張加入手牌 (0-2): "))
        if 0 <= chosen_index <= 2:
            self.hand.append(cards[chosen_index])
            discarded_cards = [card for i, card in enumerate(cards) if i !=
chosen_index]
            game.discard_pile.extend(discarded_cards) # 其餘棄置
            print(f"{self.name} 選擇了 {cards[chosen_index]} 加入手牌，
其餘牌已棄置。")
        else:
            print(f"無效的選擇，所有翻開的牌都將被棄置。")
            game.discard_pile.extend(cards)
    else:
        print(f"{self.name} 使用福澤綿羊失敗，牌堆已空。")

def function_copy_ability(self, game): # 猴/雞 - 能力複製
    """
    效果：複製上一個玩家 *回合內* 打出的 *第一張* 功能牌效果一
    次。
    如果上一個玩家該回合沒有打出功能牌，則無法複製。
    """
    last_player = game.get_last_player() # 需要在 Game 類別中記錄上一個
    玩家
    if not last_player or not
game.last_played_function_card_by_player.get(last_player):
        print(f"{self.name} 發動能力複製失敗，上一個玩家本回合沒有打
    出功能牌。")
    return

    copied_card = game.last_played_function_card_by_player[last_player] #

```

取得上個玩家回合內的第一張功能牌

```
print(f"{self.name} 複製了上一個玩家 {last_player.name} 打出的功能牌: {copied_card.function}")
```

```
self.activate_function(copied_card, game) # 再次調用相同的功能
```

```
def function_loyalty_bonus(self, game): # 狗 - 忠誠護衛 (計分時觸發，這裡僅做標記，實際計分在 Game 的 score_game 中)
```

```
print(f"{self.name} 打出了忠誠護衛 (狗)，組合得分時可能有額外加分。")
```

```
pass # 實際加分在計分階段處理
```

```
def function_furious_charge(self, game): # 豬 - 狂暴衝鋒
```

```
"""
```

效果：本回合可以額外執行最多 1 個行動 (總共最多 2 個行動)。

在 Game 類別的回合控制中，會檢查玩家是否已發動狂暴衝鋒，並允許額外行動。

```
"""
```

```
game.extra_action_available = True # 標記本回合可以額外行動
```

```
print(f"{self.name} 發動了狂暴衝鋒 (豬)，本回合可以額外執行最多一個行動。")
```

--- 卡牌定義 ---

```
class Card:
```

```
def __init__(self, suit, rank, card_type, function_name=None):
```

```
    self.suit = suit
```

```
    self.rank = rank
```

```
    self.card_type = card_type
```

```
    self.function = function_name # 功能牌的名稱 (e.g., 'steal_card')
```

```
def __str__(self):
```

```
    return f"{self.suit}{self.rank}"
```

--- 玩家定義 ---

```
class Player:
```

```
def __init__(self, name):
```

```
    self.name = name
```

```
    self.hand = []
```

```

self.score_area = [] # 玩家的計分區，用於放置已得分的牌
self.score = 0
self.ambush_card = None # 玩家的埋伏牌

def draw_card(self, deck):
    card = deck.draw_card()
    if card:
        self.hand.append(card)

def play_card(self, card_index, game):
    if 0 <= card_index < len(self.hand):
        played_card = self.hand.pop(card_index)

        print(f"{self.name} 打出了 {played_card}")

        if played_card.card_type == 'point':
            self.score_area.append(played_card) # 點數牌直接加入計分區
        elif played_card.card_type == 'wild':
            self.handle_wild_card(played_card, game) # 處理百搭牌
        elif played_card.card_type == 'function':
            self.handle_function_card(played_card, game) # 處理功能牌
        else:
            print("不 Recognized 的牌類型")

def handle_wild_card(self, card, game):
    print(f"{card} 是百搭牌，無特殊效果。")
    self.score_area.append(card) # 百搭牌加入計分區

def handle_function_card(self, card, game):
    print(f"{card} 是功能牌，觸發功能: {card.function}")
    if card.function == 'steal_card':
        self.function_steal_card(game)
    elif card.function == 'defense_shield':
        self.function_defense_shield(game)
    elif card.function == 'remove_points':
        self.function_remove_points(game)
    elif card.function == 'preview_deck':
        self.function_preview_deck(game)

```



```

elif card.function == 'card_exchange':
    self.function_card_exchange(game)
elif card.function == 'ambush_score':
    self.function_ambush_score(card, game) # 傳遞牌和遊戲物件
elif card.function == 'extra_draw':
    self.function_extra_draw(game)
elif card.function == 'lucky_draw':
    self.function_lucky_draw(game)
elif card.function == 'copy_ability':
    self.function_copy_ability(game)
elif card.function == 'loyalty_bonus':
    self.function_loyalty_bonus(game)
elif card.function == 'furious_charge':
    self.function_furious_charge(game)
self.score_area.append(card) # 功能牌也加入計分區 (根據遊戲規則調整)

# --- 功能牌效果 ---
def function_steal_card(self, game):
    opponent = game.get_opponent_player(self)
    if opponent and opponent.hand:
        stolen_card = opponent.hand.pop(0) # 偷取對方手牌的第一張
        self.hand.append(stolen_card)
        print(f"{self.name} 使用了 偷牌 功能，從 {opponent.name} 手中偷
取了一張牌。")
    else:
        print("沒有可以偷牌的對象或對方沒有手牌。")

def function_defense_shield(self, game):
    print(f"{self.name} 使用了 防禦盾 功能，免疫一次負面效果。(目前尚
未實作效果)")
    # ... 實作防禦盾效果，例如設定一個標記免疫下一次被偷牌 ...

def function_remove_points(self, game):
    opponent = game.get_opponent_player(self)
    if opponent and opponent.score_area:
        removed_card = opponent.score_area.pop(0) # 移除對方計分區的第一
張牌
        game.discard_pile.append(removed_card) # 將移除的牌加入棄牌堆

```

```
        print(f"{self.name} 使用了 移除點數 功能，移除了 {opponent.name} 計分區的一張牌。")
```

```
    else:
```

```
        print("沒有可以移除點數的對象或對方計分區沒有牌。")
```

```
def function_preview_deck(self, game):
```

```
    print(f"{self.name} 使用了 預覽牌堆 功能，看到了牌堆頂的三張牌: ")
```

```
    preview_cards = game.deck.cards[:3] # 預覽牌堆頂的三張牌
```

```
    for card in preview_cards:
```

```
        print(card)
```

```
def function_card_exchange(self, game):
```

```
    opponent = game.get_opponent_player(self)
```

```
    if opponent and opponent.hand and self.hand:
```

```
        give_card = self.hand.pop(0) # 玩家給出一張手牌
```

```
        receive_card = opponent.hand.pop(0) # 對手給出一張手牌
```

```
        self.hand.append(receive_card)
```

```
        opponent.hand.append(give_card)
```

```
        print(f"{self.name} 使用了 交換手牌 功能，與 {opponent.name} 交換了一張手牌。")
```

```
    else:
```

```
        print("沒有可以交換手牌的對象或雙方手牌不足。")
```

```
def function_ambush_score(self, card, game):
```

```
    if not self.ambush_card: # 檢查是否已經有埋伏牌
```

```
        self.ambush_card = card # 設定埋伏牌
```

```
        print(f"{self.name} 使用了 埋伏計分 功能，將 {card} 設定為埋伏牌。")
```

```
    else:
```

```
        print(f"{self.name} 已經有埋伏牌 {self.ambush_card}，無法重複設定。")
```

```
def function_extra_draw(self, game):
```

```
    game.extra_action_available = True # 標記當前回合可以額外行動一次
```

```
    print(f"{self.name} 使用了 額外抽牌 功能，本回合可以額外行動一次。")
```

```
def function_lucky_draw(self, game):
```

```
drawn_cards = game.deck.draw_cards(3) # 抽三張牌
self.hand.extend(drawn_cards) # 加入玩家手牌
print(f"{self.name} 使用了 幸運抽牌 功能，額外抽了三張牌。")
```

```
def function_copy_ability(self, game):
    if game.last_played_function_card and
game.last_played_function_card.function:
        ability_to_copy = game.last_played_function_card.function
        # 避免無限複製 'copy_ability' 自身
        if ability_to_copy != 'copy_ability':
            print(f"{self.name} 使用了 複製能力 功能，複製了上一個功能牌
{game.last_played_function_card} 的效果: {ability_to_copy}")
            # 這裡需要動態調用複製的功能
            if ability_to_copy == 'steal_card':
                self.function_steal_card(game)
            elif ability_to_copy == 'remove_points':
                self.function_remove_points(game)
            # ... 繼續擴展其他可以被複製的功能 ...
        else:
            print("上一個功能牌是 複製能力，無法複製自身。")
    else:
        print("沒有可複製的功能，上一個玩家沒有打出功能牌。")
```

```
def function_loyalty_bonus(self, game):
    print(f"{self.name} 打出了 忠誠護衛， 狗牌已在棄牌堆，計分時將獲得
額外加分。")
    # 效果在計分階段觸發
```

```
def function_furious_charge(self, game):
    game.extra_action_available = True # 標記當前回合可以額外行動一次
    print(f"{self.name} 使用了 狂暴衝鋒 功能，本回合可以額外行動一次。
")
```

--- 牌組定義 ---

```
class Deck:
```

```
    def __init__(self):
        self.cards = self.create_deck()
        random.shuffle(self.cards) # 洗牌
```

```

def create_deck(self):
    suits = ["梅花", "方块", "红心", "黑桃"]
    ranks = ["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
    function_ranks = ["2", "3", "4", "5", "6", "7"] # 設定哪些點數的牌為功能牌
    function_suits_1 = ["梅花", "方块"] # 梅花和方块的部分點數為功能牌
    function_suits_2 = ["红心", "黑桃"] # 红心和黑桃的部分點數為功能牌

    deck = []
    for suit in suits:
        for rank in ranks:
            card_type = 'point' # 預設為點數牌
            function_name = None

            if rank == "A" or rank in ["8", "9", "10"]:
                card_type = 'point'
            elif rank in ["J", "Q", "K"]:
                card_type = 'wild' # J, Q, K 為百搭牌
            elif rank in function_ranks: # 2-7 的牌根據花色設定功能
                card_type = 'function'
                if suit == function_suits_1[0] and rank == "2": function_name =
'steal_card' # 梅花 2: 鼠
                elif suit == function_suits_2[0] and rank == "2": function_name =
'defense_shield' # 红心 2: 牛
                elif suit == function_suits_1[0] and rank == "3": function_name =
'remove_points' # 梅花 3: 虎
                elif suit == function_suits_2[0] and rank == "3": function_name =
'preview_deck' # 红心 3: 兔
                elif suit == function_suits_1[0] and rank == "4": function_name =
'card_exchange' # 梅花 4: 龍
                elif suit == function_suits_2[0] and rank == "4": function_name =
'ambush_score' # 红心 4: 蛇
                elif suit == function_suits_1[0] and rank == "5": function_name =
'extra_draw' # 梅花 5: 馬
                elif suit == function_suits_2[0] and rank == "5": function_name =
'lucky_draw' # 红心 5: 羊
                elif suit == function_suits_1[0] and rank == "6": function_name =
'copy_ability' # 梅花 6: 猴/雞

```

```

        elif suit == function_suits_2[0] and rank == "6": function_name =
'loyalty_bonus' # 红心 6: 狗
        elif suit == function_suits_1[0] and rank == "7": function_name =
'furious_charge' # 梅花 7: 豬
        # ... 可以繼續擴展功能牌 ...

```

```

        deck.append(Card(suit, rank, card_type, function_name))
    return deck

```

```

def draw_card(self):
    if self.cards:
        return self.cards.pop(0)
    else:
        return None # 牌堆空了

```

```

def draw_cards(self, count):
    drawn_cards = []
    for _ in range(count):
        card = self.draw_card()
        if card:
            drawn_cards.append(card)
        else:
            break # 牌堆空了就停止
    return drawn_cards

```

--- 遊戲控制 ---

```
class Game:
```

```

    def __init__(self, player_names):
        self.players = [Player(name) for name in player_names]
        self.deck = Deck()
        self.discard_pile = []
        self.current_player_index = 0
        self.last_played_function_card = None # 記錄上一個玩家打出的功能牌
        self.extra_action_available = False # 標記當前回合是否可以額外行動
        self.last_played_function_card_by_player = {} # 記錄每個玩家回合內打出

```

的第一張功能牌

```

        # 發初始手牌

```

```

    for player in self.players:
        for _ in range(5): # 每人發 5 張手牌
            player.draw_card(self.deck)

    def get_current_player(self):
        return self.players[self.current_player_index]

    def get_opponent_player(self, current_player):
        opponent_players = [player for player in self.players if player !=
current_player]
        if opponent_players:
            return random.choice(opponent_players) # 隨機選擇一個對手，可以
根據需要修改選擇邏輯
        return None

    def next_player(self):
        self.last_played_function_card_by_player = {} # 每回合開始時清空紀錄
        self.current_player_index = (self.current_player_index + 1) % len(self.players)
        self.extra_action_available = False # 重置額外行動標記

    def get_last_player(self): # 取得上一個玩家
        last_player_index = (self.current_player_index - 1 + len(self.players)) %
len(self.players)
        return self.players[last_player_index]

    def player_turn_play_card(self, player):
        # ... (與之前的代碼相同的出牌流程) ...
        if not player.hand:
            print("手牌為空，無法出牌。")
            return

        while True:
            card_index_input = input(f"選擇要打出的手牌索引 (0-
{len(player.hand)-1}, -1: 取消出牌): ")
            if card_index_input == '-1':
                print("取消出牌。")
                return False # 返回 False 表示沒有成功出牌

```

```

try:
    card_index = int(card_index_input)
    if 0 <= card_index < len(player.hand):
        played_card = player.hand[card_index]

        if played_card.card_type == 'function':
            if player not in self.last_played_function_card_by_player: #
記錄該玩家回合內打出的第一張功能牌
                self.last_played_function_card_by_player[player] =
played_card

                self.last_played_function_card = played_card # 仍然保留最
後一張功能牌的記錄，可能在其他地方用到

            player.play_card(card_index, self) # 玩家出牌並執行效果
            self.discard_pile.append(played_card) # 將打出的牌加入棄牌堆
            return True # 返回 True 表示成功出牌
        else:
            print("無效的手牌索引，請重新選擇。")
    except ValueError:
        print("請輸入有效的數字索引。")

def player_turn_draw_card(self, player):
    if self.deck.cards:
        player.draw_card(self.deck)
        print(f"{player.name} 抽了一張牌。")
        return True
    else:
        print("牌堆已空，無法抽牌。")
        return False

def end_game(self):
    print("\n--- 遊戲結束 ---")
    for player in self.players:
        self.calculate_score(player) # 計算每個玩家的分數

    winner = max(self.players, key=lambda p: p.score) # 找出分數最高的玩家
    print(f"\n--- 遊戲結果 ---")
    for player in self.players:

```

```

        print(f"{player.name}: 分數 {player.score}")
    print(f"\n 獲勝者: {winner.name}，分數 {winner.score}!")

def calculate_score(self, player):
    score = 0
    # --- 點數牌計分 ---
    for card in player.score_area:
        if card.rank.isdigit(): # 數字牌
            score += int(card.rank)
        elif card.rank == 'A':
            score += 1
        elif card.rank in ['8', '9', '10']: # A, 8, 9, 10 算點數
            rank_value = int(card.rank) if card.rank.isdigit() else 10 # 處理 '10'
            score += rank_value
        elif card.rank == '7' and card.suit == '紅心': # 紅心 7 高分牌
            score += 10 # 紅心 7 算 10 分，可以調整分數

    # --- 組合得分 (範例，可以根據規則擴展) ---
    # ... 例如：同花色組合、特定生肖組合等 ...

    # --- 特殊能力加分 (範例，根據狗牌等效果) ---
    has_dog_card = any(card.function == 'loyalty_bonus' for card in
self.discard_pile if card.card_type == 'function') # 檢查棄牌堆是否有狗牌
    if has_dog_card:
        score += 5 # 例如，有狗牌額外加 5 分

    # --- 埋伏得分 (蛇) ---
    if hasattr(player, 'ambush_card') and player.ambush_card:
        if len(player.hand) <= 2: # 檢查手牌數量條件
            score += 8 # 例如，手牌少於等於 2 張，額外加 8 分
            print(f"{player.name} 的埋伏牌 {player.ambush_card} 觸發，手牌
數量少於等於 2 張，額外獲得 8 分。")
        else:
            print(f"{player.name} 的埋伏牌 {player.ambush_card} 未觸發，手
牌數量超過 2 張。")

    player.score = score

```



```
print(f"{player.name} 的總分: {score}")
# ... 顯示其他計分項 ...

# --- 遊戲啟動 ---
if __name__ == "__main__":
    import random
    player_names = ["玩家 1", "玩家 2"] # 可以讓玩家輸入名字
    game = Game(player_names)
    game.start_game()
```

遊戲目標：

遊戲的目標是透過打出點數牌到你的計分區，並利用功能牌的效果來獲取更高的分數。遊戲結束時，總分最高的玩家獲勝。

遊戲組件：

卡牌 (Card): 遊戲中使用一副標準的撲克牌組，加上一些特殊的功能設計。卡牌主要分為三種類型：

點數牌 (Point Card): 這些牌的主要目的是為了得分。通常是數字牌 (A, 2, 3, ..., 10) 以及部分的花牌 (J, Q, K 在這個遊戲中是百搭牌)。

功能牌 (Function Card): 這些牌具有特殊效果，可以影響遊戲的進程或對手。功能牌通常是點數較小的牌 (2, 3, 4, 5, 6, 7)，並與十二生肖中的動物相關聯，具有不同的功能名稱，例如「偷牌」、「防禦護盾」等。

百搭牌 (Wild Card): 在這個遊戲中，花牌 J, Q, K 被設定為百搭牌。百搭牌本身沒有特殊功能，主要作為點數牌計分。

每張卡牌都有以下屬性：

花色 (Suit): 梅花, 方块, 红心, 黑桃。

牌面 (Rank): A, 2, 3, ..., 10, J, Q, K。

卡牌類型 (Card Type): function, point, wild。

功能 (Function): 僅功能牌有此屬性，標示功能牌的名稱，例如 'steal_card' (偷牌)。

牌組 (Deck): 遊戲開始時，使用一副洗好的完整卡牌組作為牌組，玩家從牌組中抽牌。

棄牌堆 (Discard Pile): 玩家打出的牌以及遊戲中被移除的牌會放入棄牌堆。

玩家 (Player): 參與遊戲的個體。每個玩家有：

名稱 (Name): 玩家的識別名稱。

手牌 (Hand): 玩家當前持有的卡牌。

計分區 (Score Area): 玩家打出的點數牌和功能牌會放置在這裡，用於計分。

分數 (Score): 玩家在遊戲中累積的總分。

埋伏牌 (Ambush Card): 玩家使用「蛇」功能牌時，可以埋伏一張手牌，用於遊戲結束時的額外計分。

遊戲流程：

遊戲準備 (Game Setup):

建立牌組: 程式碼會自動建立一副標準的卡牌組，包含點數牌、功能牌和百搭牌，並洗牌。

建立玩家: 遊戲開始時，需要決定參與遊戲的玩家人數，並為每位玩家設定名稱。程式碼範例中預設為 "玩家 1" 和 "玩家 2"。

發牌: 每位玩家從牌組中抽取 5 張牌作為起始手牌。

玩家回合 (Player Turn): 遊戲以回合制進行，依照玩家順序輪流進行回合。在自己的回合中，玩家可以選擇執行以下行動（根據程式碼，玩家每回合的操作流程可能需要更明確的規則說明，以下是一種可能的流程）：

抽牌階段: 回合開始時，玩家需要從牌組中抽取一張牌，加入到手牌中。如果牌組已空，則無法抽牌。

出牌階段: 玩家可以選擇打出一張手牌。打出的牌根據卡牌類型有不同的處理方式：

點數牌: 直接放入玩家的計分區。

功能牌: 打出時，會印出功能牌的名稱和功能，並立即執行該功能牌的效果。功能牌執行效果後，也會放入玩家的計分區。程式碼中定義了以下功能牌及其效果：

梅花 2 (鼠 - 偷牌換牌): 從隨機一位對手手中隨機偷取一張手牌，然後自己從牌組中抽一張牌。

紅心 2 (牛 - 防禦護盾): 選擇自己計分區中的一張點數牌，使其在本回合免受

負面效果影響（程式碼中目前防禦效果尚未完全實作）。

梅花 3 (虎 - 移除點數): 隨機選擇一位對手，減少對手 2 分。

紅心 3 (兔 - 預見未來): 偷看牌組頂的 2 張牌，並展示給自己看，但不會改變牌組順序。

梅花 4 (龍 - 卡牌交換): 與一位對手交換手牌。玩家需要選擇自己和對手的手牌進行交換。

紅心 4 (蛇 - 埋伏得分): 玩家選擇一張手牌作為埋伏牌，秘密放置。在遊戲結束時，如果玩家手牌數量少於等於 2 張，則可獲得額外分數。

梅花 5 (馬 - 疾速抽牌): 額外從牌組中抽取一張牌。

紅心 5 (羊 - 福澤綿羊): 從牌組頂翻開三張牌，玩家選擇其中一張加入手牌，其餘兩張牌則棄置到棄牌堆。如果牌組牌數不足三張，則盡可能抽牌。

梅花 6 (猴/雞 - 能力複製): 複製上一個玩家在上一個回合打出的第一張功能牌的效果。如果上一個玩家該回合沒有打出功能牌，則無法複製。如果上一個功能牌是「能力複製」本身，則無法再次複製。

紅心 6 (狗 - 忠誠護衛): 打出此牌時沒有立即效果，但在遊戲結束計分時，若棄牌堆中有「狗」牌，玩家可獲得額外加分。

梅花 7 (豬 - 狂暴衝鋒): 在本回合可以額外執行最多 1 個行動（例如，額外打出一張牌或額外抽一張牌，具體額外行動種類可能需要更詳細的規則說明）。

百搭牌: 打出時，沒有特殊效果，直接放入玩家的計分區。百搭牌主要用於計分。

結束回合: 玩家完成出牌階段後，回合結束，輪到下一位玩家。遊戲會記錄上一個玩家打出的功能牌，以便「猴/雞 - 能力複製」功能牌使用。

遊戲結束與計分 (Game End & Scoring):

遊戲結束的條件可能需要根據具體規則來設定，例如：

當牌組中的牌被抽完時。

進行固定回合數後。

有玩家達到預設的分數目標。

程式碼範例中，遊戲結束的觸發條件並不明確，可能需要手動結束遊戲流程。

遊戲結束時，會進行計分。計分方式如下：

點數牌計分: 計算玩家計分區中所有點數牌的分數總和。

數字牌 (2-10): 分數等於牌面數字。

A: 1 分。

8, 9, 10: 分數等於牌面數字。

紅心 7: 10 分 (高分牌，分數可調整)。

J, Q, K (百搭牌): 在這個計分規則中，百搭牌 J, Q, K 似乎沒有特別的點數，可能預設為 0 分，或者也可能算作點數牌，具體規則需要確認。(程式碼中計分部分，針對 `score_area` 的牌只判斷 `rank` 是否為數字或 A, 8, 9, 10, 7(紅心)，沒有針對 J, Q, K 的計分，所以百搭牌可能在此遊戲規則中不計分，或者有另一套計分規則。)

組合得分 (Combo Score): 程式碼中提到「組合得分 (範例，可以根據規則擴展) ... 例如：同花色組合、特定生肖組合等 ...」，這部分目前沒有具體實作，但遊戲可以進一步擴展，加入例如收集特定花色或特定生肖功能牌的組合，以獲得額外分數。

特殊能力加分 (Special Ability Bonus):

忠誠護衛 (狗) 加分: 如果棄牌堆中存在功能牌「狗」(紅心 6 - 忠誠護衛)，則玩家可以獲得額外 5 分 (分數可調整)。

埋伏得分 (Ambush Score - 蛇): 如果玩家在遊戲中使用過「蛇」(紅心 4 - 埋伏得分) 功能牌，並埋伏了一張牌，且在遊戲結束時，玩家的手牌數量少於等於 2 張，則可獲得額外 8 分 (分數可調整)。

勝負判定 (Winning Condition): 遊戲結束時，比較所有玩家的總分，總分最高的玩家獲勝。如果有平分，可以平手或依照額外規則判定勝負 (規則未定義)。

如何開始遊戲 (Running the Game):

複製程式碼: 將您提供的 Python 程式碼複製到您的 Python 環境中 (例如，Python IDLE, VS Code, Jupyter Notebook 等)。

執程式碼: 執程式碼。在程式碼的 `if __name__ == "__main__":` 區塊中，遊戲被啟動。

輸入玩家名稱: 程式碼預設玩家名稱為 "玩家 1" 和 "玩家 2"。您可以修改 `player_names = ["玩家 1", "玩家 2"]` 這一行程式碼，將玩家名稱更改為您想要的名稱，例如 `player_names = ["小明", "小華"]`。

依照指示操作: 程式碼執行後，會在終端機 (或命令列) 印出遊戲訊息和提示。您需要依照提示，輸入數字索引來選擇要打出的手牌、選擇要保護的牌、選擇要交換的手牌等等。

觀察遊戲流程和結果: 遊戲會自動進行玩家回合輪替、抽牌、出牌、執行功能牌效果、計分等流程，並在終端機印出相關訊息。遊戲結束時，會顯示所有玩

家的分數和獲勝者。

遊戲操作提示 (Game Operation Tips):

手牌索引: 當程式提示您選擇手牌時，會顯示您的手牌列表，每張手牌前面都有一個數字索引 (從 0 開始)。您需要輸入對應的數字索引來選擇要打出的手牌。

取消操作: 在某些選擇手牌或目標的提示中，您可能會看到 "-1: 取消" 的選項。輸入 "-1" 可以取消當前操作。

功能牌效果理解: 仔細閱讀程式碼中功能牌的說明 (註解) 以及遊戲過程中印出的訊息，理解每張功能牌的效果，才能更好地運用功能牌來取得優勢。

策略思考: 這個遊戲包含一定的策略性。您需要考慮如何有效地運用手牌中的點數牌和功能牌來得分和干擾對手。例如，何時打出點數牌得分，何時使用功能牌偷牌、移除對手分數、或是為自己創造更有利的局面。

遊戲規則注意事項 (Game Rule Notes):

百搭牌計分: 程式碼中對於百搭牌 (J, Q, K) 的計分規則可能不明確，需要確認是否計分，以及如何計分 (例如，固定分數或是依照牌面數字計分)。目前的計分程式碼似乎沒有計算百搭牌的分數。

功能牌組合與擴展: 遊戲目前的功能牌設計是基於十二生肖，功能種類和效果可以進一步擴展和調整，例如加入更多不同效果的功能牌，或者設計功能牌之間的組合效果。

遊戲結束條件: 遊戲結束的條件可以更明確地定義，例如設定牌組抽完時遊戲結束，或是設定回合數限制。

多人遊戲: 目前的程式碼範例是雙人遊戲。如果需要支援更多玩家，需要修改遊戲流程和規則，例如調整玩家輪替方式、功能牌效果的目標選擇等。

使用者介面: 目前的遊戲操作是透過終端機文字輸入和輸出來進行的，使用者介面較為簡陋。如果需要提升遊戲體驗，可以開發圖形使用者介面 (GUI)。