```
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
                        print(f"{self.name} 保護了展示區的點數牌:
{protected_card} (界域: {protected_card.suit}{protected_card.rank})，使其本回合
免受影響。") # 加入界域資訊
```

```
                    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
# --- 界域卡定義 ---
class RealmCard:
        def __init__(self, realm_type):
                self.realm_type = realm_type # 界域類型 (日月界, 風雲靈界, 海底浮島界, 水陸界, 沼泥界)
                self.is_controlled = None # 控制者，可以是 Player 物件或 None (無人控制)

        def __str__(self):
```

```
        return f"界域: {self.realm_type}"

# --- 界域牌組定義 ---
class RealmDeck:
    def __init__(self):
        self.cards = self.create_deck()
        random.shuffle(self.cards) # 洗牌

    def create_deck(self):
        realm_types = ["日月界", "風雲靈界", "海底浮島界", "水陸界", "沼泥界"]
        deck = []
        for realm_type in realm_types:
            for _ in range(4): # 每種界域 4 張，共 20 張
                deck.append(RealmCard(realm_type))
        return deck

    def draw_card(self):
        if self.cards:
            return self.cards.pop(0)
        else:
            return None # 界域牌組空了

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

 # --- 遊戲控制 ---
 # --- 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
        self.extra_action_available = False
        self.last_played_function_card_by_player = {}

        # --- 界域機制 ---
        self.realm_deck = RealmDeck() # 創建界域牌組
        self.realm_area = self.realm_deck.draw_cards(5) # 抽取 5 張界域卡作為可爭奪區域
        print("\n--- 可爭奪的界域 ---")
        for realm_card in self.realm_area:
            print(f"  {realm_card}")

        # 發初始手牌
        for player in self.players:
            for _ in range(5):
                player.draw_card(self.deck)

    def get_current_player(self):
        """
        取得目前回合的玩家。
        """
        return self.players[self.current_player_index]

    def get_opponent_player(self, current_player):
        """
        取得對手玩家。
        """
        player_index = self.players.index(current_player)
        opponent_index = 1 - player_index  # 假設只有兩個玩家，索引 0 和 1
        return self.players[opponent_index]

    def next_player(self):
        """
        將當前玩家索引移到下一位玩家。
        """
        self.current_player_index = (self.current_player_index + 1) %
```

```
len(self.players)
        self.extra_action_available = False #  重置額外行動狀態，每回合開始時重置
        self.last_played_function_card = None #  清空上回合功能牌紀錄
        self.last_played_function_card_by_player = {} #  清空每位玩家上回合功能牌紀錄

    def start_game(self):
        print("---  遊戲開始  ---")
        player_names = ", ".join([player.name for player in self.players])
        print(f"玩家: {player_names}")

        round_num = 0
        while round_num < 10: #  示例回合數，可以調整
            round_num += 1
            print(f"\n---  第  {round_num}  回合  ---")
            for i in range(len(self.players)): #  每個玩家一個回合
                current_player = self.get_current_player()
                print(f"\n 輪到玩家: {current_player.name}")
                print(f"手牌: {', '.join(map(str, current_player.hand))}")

                action_count = 0
                while action_count < (2 if self.extra_action_available else 1): # 基礎行動次數為 1，狂暴衝鋒時為 2
                    action_count += 1
                    print(f"\n---  行動  {action_count} ---")
                    action = input(f"選擇行動  (1:  出牌, 2:  抽牌, 3:  控制界域, 0:  結束回合): ") #  新增  "3:  控制界域"  選項
                    if action == '1':
                        self.player_turn_play_card(current_player)
                    elif action == '2':
                        self.player_turn_draw_card(current_player)
                    elif action == '3':
                        self.player_turn_control_realm(current_player) #  處理控制界域行動
                    elif action == '0':
                        print("結束回合")
                        break
```

```
                    else:
                        print("無效的行動選擇。")

                self.next_player()
                self.last_played_function_card_by_player = {} # 每一回合結束時清空紀錄

            if not self.deck.cards: # 牌堆耗盡結束遊戲
                print("\n 牌堆耗盡，遊戲即將結束。")
                break

        self.end_game() # 遊戲結束計分

    def player_turn_play_card(self, player):
        """
        處理玩家回合的出牌行動。
        """
        if not player.hand:
            print(f"{player.name} 手牌已空，無法出牌。")
            return

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

        while True:
            card_index_input = input(f"{player.name}, 選擇要打出的手牌索引({card_choice_str}, -1: 取消): ")
            if card_index_input == '-1':
                print("取消出牌。")
                return
            try:
                card_index = int(card_index_input)
                if 0 <= card_index < len(player.hand):
                    card_to_play = player.hand[card_index]
                    player.play_card(card_index, self) # 玩家執行出牌動作，並傳遞 game 物件
                    if card_to_play.card_type == 'function':
                        self.last_played_function_card = card_to_play # 記錄
```

```
最後打出的功能牌
                                self.last_played_function_card_by_player[player] =
card_to_play #  記錄玩家該回合打出的第一張功能牌
                            break #  成功出牌後跳出迴圈
                        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 player_turn_draw_card(self, player):
        """
        處理玩家回合的抽牌行動。
        """
        player.draw_card(self.deck)
        print(f"{player.name}  抽了一張牌。")

    def player_turn_control_realm(self, player):
        """
        處理玩家控制界域的行動。
        簡化規則：支付  2  張點數牌來控制一個未被控制的界域。
        """
        available_realms = [realm for realm in self.realm_area if
realm.is_controlled is None] #  篩選未被控制的界域
        if not available_realms:
            print("目前沒有可控制的界域。")
            return

        if len([card for card in player.hand if card.card_type == 'point']) < 2: #  檢查
是否有點數牌支付
            print("手牌中點數牌不足，無法控制界域  (需要  2  張點數牌)。")
            return

        realm_choices = [f"{i}: {realm}" for i, realm in enumerate(available_realms)]
        realm_choice_str = ", ".join(realm_choices)

        while True:
            realm_index_input = input(f"{player.name},  選擇要控制的界域索引
({realm_choice_str}, -1:  取消): ")
```

```
            if realm_index_input == '-1':
                print("取消控制界域。")
                return
            try:
                realm_index = int(realm_index_input)
                if 0 <= realm_index < len(available_realms):
                    chosen_realm = available_realms[realm_index]

                    # 棄掉 2 張點數牌
                    discarded_cards = []
                    point_cards_in_hand = [card for card in player.hand if
card.card_type == 'point']
                    point_card_indices_to_discard = []

                    print("選擇要棄掉的 2 張點數牌以控制界域：")
                    point_card_choices = [f"{i}: {card}" for i, card in
enumerate(point_cards_in_hand)]
                    point_card_choice_str = ", ".join(point_card_choices)

                    while len(discarded_cards) < 2:
                        card_index_input = input(f"選擇要棄掉的點數牌索引
({point_card_choice_str}): ")
                        try:
                            card_index = int(card_index_input)
                            if 0 <= card_index < len(point_cards_in_hand) and
card_index not in point_card_indices_to_discard:
                                card_to_discard =
point_cards_in_hand[card_index]
                                discarded_cards.append(card_to_discard)

point_card_indices_to_discard.append(card_index)
                                print(f"已選擇棄掉: {card_to_discard}")
                                if len(discarded_cards) == 2:
                                    break # 湊齊 2 張就跳出迴圈
                            else:
                                print("無效的索引，或已選擇過的牌，請
重新選擇。")
                        except ValueError:
```

```
                    print("請輸入有效的數字索引。")

                if len(discarded_cards) == 2: # 成功選擇 2 張點數牌
                    for card_to_discard in discarded_cards:
                        player.hand.remove(card_to_discard) # 從手牌移除
                        self.discard_pile.append(card_to_discard) # 加入棄牌堆

                    chosen_realm.is_controlled = player # 標記界域被該玩家控制
                    print(f"{player.name} 成功控制了界域: {chosen_realm}，支付了 2 張點數牌。")
                    return True # 控制成功
                else:
                    print("控制界域失敗，點數牌選擇不完整。")
                    return False # 控制失敗
            else:
                print("無效的界域索引，請重新選擇。")
        except ValueError:
            print("請輸入有效的數字索引。")
    return False # 控制失敗


def calculate_score(self, player):
    score = 0
    # --- 點數牌計分 ---
    for card in player.score_area:
        if card.card_type == 'point':
            if card.rank.isdigit(): # 數字牌
                score += int(card.rank)
            elif card.rank in ['J', 'Q', 'K']: # J, Q, K 百搭牌算 10 分
                score += 10
            elif card.rank == 'A': # A 算 1 分
                score += 1
        elif card.card_type == 'wild':
            score += 10 # 百搭牌基礎 10 分
        elif card.card_type == 'function': # 功能牌也算分
```

```
                score += 0 # 功能牌基礎 0 分，部分功能牌有額外加分

        # --- 界域控制得分 ---
        realm_score = 0
        controlled_realms = [realm for realm in self.realm_area if
realm.is_controlled == player]
        realm_score = len(controlled_realms) * 5 # 每個控制的界域提供 5 分，
分數可調整
        score += realm_score
        print(f"{player.name} 控制界域得分: {realm_score} (控制了
{len(controlled_realms)} 個界域)")

        # --- 埋伏得分 (蛇) ---
        if hasattr(player, 'ambush_card') and player.ambush_card:
            if len(player.hand) <= 2: # 手牌少於等於 2 張
                ambush_score = 15 # 埋伏成功額外 15 分，分數可調整
                score += ambush_score
                print(f"{player.name} 埋伏得分成功! 額外獲得
{ambush_score} 分 (埋伏牌: {player.ambush_card}, 手牌數量: {len(player.hand)})")
            else:
                print(f"{player.name} 埋伏得分失敗，手牌數量超過 2 張 (埋
伏牌: {player.ambush_card}, 手牌數量: {len(player.hand)})")


        # --- 忠誠護衛加分 (狗) ---
        loyalty_bonus_cards = [card for card in self.discard_pile if card.function ==
'loyalty_bonus']
        if loyalty_bonus_cards:
            loyalty_bonus = len(loyalty_bonus_cards) * 3 # 每張忠誠護衛額外 3
分，分數可調整
            score += loyalty_bonus
            print(f"{player.name} 忠誠護衛加分! 額外獲得 {loyalty_bonus} 分
(忠誠護衛牌數量: {len(loyalty_bonus_cards)})")


        player.score = score
        print(f"{player.name} 的總分: {score}")
```

```
    def end_game(self):
        print("\n---  遊戲結束，計分  ---")
        for player in self.players:
            self.calculate_score(player)

        # 找出最高分玩家
        winner = max(self.players, key=lambda player: player.score)
        print(f"\n---  遊戲結果  ---")
        print(f"獲勝者: {winner.name},  總分: {winner.score}")

        # 顯示所有玩家得分
        for player in self.players:
            print(f"{player.name}  總分: {player.score}")



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

    def get_opponent_player(self, player):
        player_index = self.players.index(player)
        opponent_index = 1 - player_index
        return self.players[opponent_index]

def next_player(self):
        self.current_player_index = (self.current_player_index + 1) %
len(self.players)
        self.extra_action_available = False #  重置額外行動狀態
        self.last_played_function_card = None #  清空上回合功能牌紀錄
        self.last_played_function_card_by_player = {} #  清空每位玩家上回合功
能牌紀錄



def player_turn_control_realm(self, player):
        """
        處理玩家控制界域的行動。
        暴力簡化版本：直接控制一個未被控制的界域，無需支付點數牌。
        """
```

```
        available_realms = [realm for realm in self.realm_area if
realm.is_controlled is None] #  篩選未被控制的界域
        if not available_realms:
            print("目前沒有可控制的界域。")
            return

        realm_choices = [f"{i}: {realm}" for i, realm in enumerate(available_realms)]
        realm_choice_str = ", ".join(realm_choices)

        while True:
            realm_index_input = input(f"{player.name},  選擇要控制的界域索引
({realm_choice_str}, -1:  取消): ")
            if realm_index_input == '-1':
                print("取消控制界域。")
                return
            try:
                realm_index = int(realm_index_input)
                if 0 <= realm_index < len(available_realms):
                    chosen_realm = available_realms[realm_index]

                    # ---  暴力簡化：直接控制，無需支付點數牌  ---
                    chosen_realm.is_controlled = player #  標記界域被該玩家
控制
                    print(f"{player.name}  成功控制了界域: {chosen_realm} (簡
化版，無需支付點數牌)。")
                    return True #  控制成功
                else:
                    print("無效的界域索引，請重新選擇。")
            except ValueError:
                print("請輸入有效的數字索引。")
        return False #  控制失敗

# ---  遊戲啟動  ---
if __name__ == "__main__":
    player_names = ["玩家 1", "玩家 2"]
    game = Game(player_names)
    game.start_game()
```

遊戲總覽

這是一個雙人卡牌遊戲，結合了卡牌打出、功能發動、區域控制和計分等元素。玩家的目標是透過打出卡牌、發動功能、控制「界域」來獲得分數，最終得分最高的玩家獲勝。遊戲使用了一副特製的卡牌，包含點數牌、功能牌和百搭牌，以及另一組「界域卡」代表遊戲中的可爭奪區域。

遊戲設置

玩家準備: 遊戲開始時，需要兩位玩家，程式碼中以 "玩家 1" 和 "玩家 2" 作為預設名稱。
創建牌組: 遊戲會自動創建一副標準的卡牌組 (Deck)，包含四種花色（梅花、方塊、紅心、黑桃）和多種牌面（A, 2-10, J, Q, K）。牌組會洗牌。
創建界域牌組: 遊戲同時創建一副界域牌組 (RealmDeck)，包含五種界域類型（日月界、風雲靈界、海底浮島界、水陸界、沼泥界），每種界域各 4 張，共 20 張。界域牌組也會洗牌。
發放初始手牌: 每位玩家從卡牌組中抽取 5 張牌作為起始手牌。
展示可爭奪界域: 從界域牌組中抽取 5 張牌，面朝上放置作為「可爭奪的界域」。這些界域在遊戲中可以被玩家控制，提供分數。
卡牌種類

遊戲中的卡牌主要分為三種類型：

點數牌 (point):
牌面：A, 8, 9, 10。
功能：主要用於計分。牌面上的數字直接對應分數，J, Q, K 等級的百搭牌作為點數牌打出時也算分。
功能牌 (function):
牌面：2, 3, 4, 5, 6, 7 (特定花色)。
功能：具有特殊效果，例如偷牌、移除對方點數、預覽牌堆、交換手牌、額外抽牌等。功能牌的效果會在被打出時立即發動。
不同花色和點數的功能牌對應不同的動物名稱和功能：
梅花 2: 鼠 (steal_card) - 偷牌換牌
紅心 2: 牛 (defense_shield) - 防禦護盾
梅花 3: 虎 (remove_points) - 移除點數
紅心 3: 兔 (preview_deck) - 預見未來
梅花 4: 龍 (card_exchange) - 卡牌交換
紅心 4: 蛇 (ambush_score) - 埋伏得分
梅花 5: 馬 (extra_draw) - 疾速抽牌

紅心 5: 羊 (lucky_draw) - 福澤綿羊
梅花 6: 猴/雞 (copy_ability) - 能力複製
紅心 6: 狗 (loyalty_bonus) - 忠誠護衛
梅花 7: 豬 (furious_charge) - 狂暴衝鋒
百搭牌 (wild):
牌面：J, Q, K。
功能：可以作為點數牌計分，基礎分數為 10 分。程式碼中提到百搭牌的功能尚未完全實作，可能在更完善的規則中會有特殊用途。
遊戲流程

遊戲主要以回合制進行，預設進行 10 個回合，或直到牌組耗盡為止。每個回合中，每位玩家依序進行行動。

玩家回合

在自己的回合中，玩家可以從以下行動中選擇一個執行（基礎行動次數為 1，部分功能牌如「豬 - 狂暴衝鋒」可以提供額外行動次數）：

出牌 (play_card):

玩家可以從手牌中選擇一張牌打出。
點數牌: 打出的點數牌會進入玩家的計分區 (score_area)，直接用於計分。
功能牌: 打出的功能牌會進入計分區，同時立即發動牌面上的功能效果。遊戲會根據功能牌的名稱調用相應的 function_xxx 方法來執行效果。例如，打出梅花 2（鼠 - 偷牌換牌）會觸發 function_steal_card 方法。
百搭牌: 打出的百搭牌會進入計分區，作為點數牌計分。
抽牌 (draw_card):

玩家可以從牌組中抽取一張牌，加入自己的手牌。
控制界域 (control_realm):

玩家可以嘗試控制一個尚未被控制的「界域」。
原始規則 (程式碼中第一個 player_turn_control_realm 函數): 玩家需要支付 2 張手牌中的點數牌才能控制一個界域。玩家需要選擇要棄掉的兩張點數牌。
簡化規則 (程式碼中第二個 player_turn_control_realm 函數 - "暴力簡化版本"): 玩家可以直接控制一個未被控制的界域，無需支付任何牌。
成功控制界域後，該界域卡會被標記為由該玩家控制，並在遊戲結束時提供分數。

結束回合 (end turn):

玩家可以選擇結束當前回合，將遊戲控制權交給下一位玩家。
功能牌效果詳解

程式碼中實作了多種功能牌的效果，以下列出部分功能的說明 (詳細效果請參考程式碼中的 function_xxx 方法)：

鼠 - 偷牌換牌 (steal_card): 從一位對手玩家手中隨機偷取一張手牌，然後自己從牌組中抽一張牌。
牛 - 防禦護盾 (defense_shield): 保護自己展示區的一張點數牌，使其在本回合免受特定負面效果影響（具體效果實作較簡化，需要完善）。玩家需要選擇要保護的點數牌。
虎 - 移除點數 (remove_points): 使一位對手玩家減少一定分數（預設為 2 分）。
兔 - 預見未來 (preview_deck): 可以偷看牌組頂部的若干張牌（預設為 2 張），讓玩家預知接下來可能抽到的牌。
龍 - 卡牌交換 (card_exchange): 與一位對手玩家交換手牌。玩家需要各自選擇一張手牌進行交換。
蛇 - 埋伏得分 (ambush_score): 玩家可以選擇一張手牌作為埋伏牌。在遊戲結束計分時，如果玩家的手牌數量少於等於 2 張，則可以獲得額外分數。玩家需要選擇要埋伏的手牌。
馬 - 疾速抽牌 (extra_draw): 允許玩家額外抽取一張牌。
羊 - 福澤綿羊 (lucky_draw): 玩家可以從牌組頂部抽取三張牌，選擇其中一張加入手牌，其餘兩張棄置。
猴/雞 - 能力複製 (copy_ability): 複製上一個玩家回合內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。
狗 - 忠誠護衛 (loyalty_bonus): 在計分階段提供額外分數，具體加分規則與棄牌堆中「忠誠護衛」牌的數量有關。此功能牌本身沒有立即發動的效果。
豬 - 狂暴衝鋒 (furious_charge): 使玩家在本回合可以額外執行最多一個行動，即該回合可以執行最多兩個行動。
遊戲結束與計分

遊戲會在進行到預設回合數（例如 10 回合）或牌組耗盡時結束。遊戲結束後，會進行計分。計分方式包括：

點數牌計分: 玩家計分區中的點數牌會根據牌面上的數字或牌型（例如百搭牌）提供分數。

界域控制得分: 玩家控制的每個界域都會提供固定的分數（預設為每個界域 5 分）。
埋伏得分 (蛇): 如果玩家成功發動「蛇 - 埋伏得分」功能，並且在遊戲結束時手牌數量少於等於 2 張，則可以獲得額外分數。
忠誠護衛加分 (狗): 根據棄牌堆中「狗 - 忠誠護衛」功能牌的數量，玩家可以獲得額外分數。
遊戲結束後，程式碼會計算每位玩家的總分，並找出得分最高的玩家作為獲勝者。

遊玩方式總結

遊戲開始: 按照遊戲設置步驟準備遊戲。
回合進行: 玩家輪流進行回合，在自己的回合中選擇執行一個行動（或在「狂暴衝鋒」效果下最多兩個行動）。
行動選擇: 玩家可以選擇出牌、抽牌或控制界域。
出牌策略:
打出點數牌和百搭牌積累基本分數。
適時打出功能牌，利用其特殊效果干擾對手、輔助自己得分或控制局勢。
根據手牌和局勢判斷是否需要埋伏手牌 (蛇)。
策略性地使用「防禦護盾」(牛) 保護高分點數牌。
考慮使用「能力複製」(猴/雞) 來模仿對手的功能牌效果。
界域控制: 在合適時機控制界域，以增加額外得分。根據遊戲規則選擇是否需要支付點數牌來控制界域。
遊戲結束: 當遊戲回合數達到上限或牌組耗盡時，遊戲結束。
計分與勝負: 計算每位玩家的總分，得分最高的玩家獲勝。