

《陰府棋局：十二生肖的命運之戰 (簡化文字版)》 遊戲規則

一、 遊戲目標

雖然程式碼中沒有明確定義遊戲的勝利條件，但根據棋局的類型推測，遊戲目標可能是：

消滅對方的所有棋子，或是達成特定的戰略目標 (例如：佔領對方的陰府)。由於是簡化文字版，具體的勝利條件可能需要進一步的規則說明，或者在實際遊玩中根據情境判斷。

二、 棋盤與棋子

棋盤： 使用 8x8 的西洋棋棋盤。棋盤的列以數字 1-8 表示，行以字母 A-H 表示。

棋子：

玩家 1：

前排 (第一排)：鼠、牛、虎、兔、龍、蛇、馬、羊 (A1-H1)

後排 (第二排)：猴、雞、狗、豬 (A2-D2)

陰府 (YF)：D1

玩家 2：

前排 (第八排)：鼠、牛、虎、兔、龍、蛇、馬、羊 (H8-A8，反向放置)

後排 (第七排)：猴、雞、狗、豬 (H7-E7，反向放置)

陰府 (YF)：D8

每種生肖棋子都有其特定的移動和攻擊方式 (詳見下方 棋子移動與攻擊方式)。

在新窗口中打开

royalchessmall.com

Initial Yin Fu Chess Board Setup

三、 遊戲流程

遊戲開始： 玩家 1 先行。

回合制： 玩家輪流進行回合，每個回合玩家可以執行以下指令之一：

list [棋子代號]：列出指定棋子的可移動位置，並會為每個位置編號。例如：**list** 鼠

move [數字]：根據 **list** 指令顯示的編號，選擇並移動棋子到對應編號的位置。例如：先 **list** 鼠，顯示編號後，輸入 **move 1** 移動到編號 1 的位置。

move [棋子代號] [起始位置] to [目標位置]：直接指定棋子和起終位置來移動棋子。例如：move 鼠 A1 to B2

attack [攻擊棋子代號] [目標位置]：命令特定棋子攻擊目標位置的敵方棋子。例如：attack 牛 C3 D4

help：顯示遊戲指令說明。

end_turn：結束當前回合，輪到對方玩家的回合。

遊戲結束：(勝利條件未明確定義，可能需要根據實際遊戲情況判斷)

四、 棋子代號與位置

棋子代號：

鼠、牛、虎、兔、龍、蛇、馬、羊、猴、雞、狗、豬、YF (陰府)

玩家 1 的棋子代號前綴為 1_，玩家 2 為 2_。例如，玩家 1 的鼠棋子代號為 1_鼠。

位置： 使用西洋棋盤座標，例如 A1, H8, C5。

五、 棋子移動與攻擊方式

以下是根據程式碼 `is_valid_move` 和 `is_valid_attack` 函數，整理出的各棋子移動與攻擊方式：

鼠 (Mouse):

移動： 直線或斜線移動 1-2 格。

攻擊： 斜線前方一格的敵方棋子。

牛 (Ox):

移動： 直線任意格數，但路徑上是否有阻礙 (未在簡化版中實作)。

攻擊： 沒有獨立的攻擊指令，攻擊可能與移動結合，或有特殊規則 (未在簡化版中實作)。

虎 (Tiger):

移動： 馬步 (如同象棋中的馬，走「日」字形)。

攻擊： 馬步範圍內有敵方棋子時，移動到該位置即可攻擊，攻擊與移動結合。

兔 (Rabbit):

移動： 直線或斜線移動 1-3 格。 可以跳過己方棋子，但不能跳過敵方棋子 (未在簡化版中實作)。

攻擊： 沒有獨立的攻擊指令，攻擊方式可能與移動或特殊規則有關 (未在簡化版中實作)。

龍 (Dragon):

移動：全方位 (直線、橫線、斜線) 移動 1-3 格。

攻擊：目標格位周圍一格內的敵方棋子。

蛇 (Snake):

移動：直線或橫線移動 1-2 格。可能具有穿梭能力 (未在簡化版中實作)。

攻擊：目標格位周圍一格內的敵方棋子。米字型攻擊範圍 (為簡化，僅實作周圍一格)。

馬 (Horse):

移動：直線或橫線移動 1-3 格。

攻擊：沒有獨立的攻擊指令，衝鋒攻擊可能與移動結合 (未在簡化版中實作)。

羊 (Goat):

移動：全方位 (直線、橫線、斜線) 移動 1 格。

攻擊：沒有主動攻擊能力，反擊可能在被攻擊時觸發 (未在簡化版中實作)。

猴 (Monkey):

移動：全方位 (直線、橫線、斜線) 移動 1 格。

攻擊：目標格位周圍一格內的敵方棋子。

雞 (Chicken):

移動：直線或橫線移動 1-2 格。

攻擊：前方一格的敵方棋子。

狗 (Dog):

移動：全方位 (直線、橫線、斜線) 移動 1 格。

攻擊：目標格位周圍一格內的敵方棋子。

豬 (Pig):

移動：直線或橫線移動 1 格。

攻擊：目標格位周圍一格內的敵方棋子。範圍衝撞攻擊 (為簡化，僅實作周圍一格)。

陰府 (YF):

移動：全方位 (直線、橫線、斜線) 移動 1 格。

攻擊：沒有主動攻擊能力。

六、 其他注意事項

移動限制：

棋子不能移動到棋盤外。

不能移動到或停留在己方棋子所在的位置。

攻擊限制：

不能攻擊己方棋子。

並非所有棋子都有主動攻擊能力 (例如：牛、兔、馬、羊、陰府)。

七、 簡化說明

此版本為《陰府棋局：十二生肖的命運之戰》的簡化文字版，部分進階規則 (如牛的阻礙判斷、兔子的跳躍、蛇的穿梭、馬的衝鋒、羊的反擊、豬的範圍衝撞等) 可能未完全實作或被簡化。 遊戲的重點在於理解基本規則和指令操作。

```
def get_player_prefix(player_name):
```

```
    """ 根據玩家名稱返回棋子代號前綴 # 陰府棋局：十二生肖的命運之戰
    (簡化文字版) """
```

```
    return "1_" if player_name == "玩家 1" else "2_"
```

```
def 初始化棋盤():
```

```
    """ 初始化棋盤，設定棋子初始位置 (修正版) """
```

```
    board = {}
```

```
    player1_pieces = ["鼠", "牛", "虎", "兔", "龍", "蛇", "馬", "羊", "猴", "雞", "狗",
    "豬"]
```

```
    player2_pieces = ["鼠", "牛", "虎", "兔", "龍", "蛇", "馬", "羊", "猴", "雞", "狗",
    "豬"]
```

```
    cols = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

```
    # 玩家 1 棋子
```

```
    for i in range(8): # 第一排：鼠 - 羊 (A1-H1)
```

```
        board[f"{cols[i]}1"] = f"1_{player1_pieces[i]}"
```

```
    for i in range(4): # 第二排：猴 - 豬 (A2-D2)
```

```
        board[f"{cols[i]}2"] = f"1_{player1_pieces[i+8]}" # player1_pieces 索引從
```

```
8 開始
```

```
    board["D1"] = "1_YF" # 玩家 1 陰府，放置在 D1 (注意：需在生肖棋子之後
    放置，避免覆蓋)
```

```

# 玩家 2 棋子
for i in range(8): # 第八排：鼠 - 羊 (H8-A8) - 反向放置
    board[f"{cols[7-i]}8"] = f"2_{player2_pieces[i]}" # cols 索引反向
for i in range(4): # 第七排：猴 - 豬 (H7-E7) - 反向放置
    board[f"{cols[7-i]}7"] = f"2_{player2_pieces[i+8]}" # cols 索引反向，
player2_pieces 索引從 8 開始
board["D8"] = "2_YF" # 玩家 2 陰府，放置在 D8 (注意：需在生肖棋子之後
放置，避免覆蓋)

print("棋盤初始化完成，棋子已自動擺放。") # 更新提示訊息，表示棋盤已
自動設定
return board

def 顯示棋盤狀態(board):
    """ 顯示棋盤的文字狀態 (簡化版，只顯示棋子位置) """
    print("\n--- 棋盤狀態 ---")
    for row in ['8', '7', '6', '5', '4', '3', '2', '1']:
        line = row + " |"
        for col in ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']:
            pos = col + row
            piece = board.get(pos, " ") # 若該位置沒有棋子，顯示空白
            line += piece + "|"
        print(line)
    print(" -----")
    print("      A  B  C  D  E  F  G  H\n")

def 取得玩家輸入(player_name):
    """ 取得玩家的文字指令輸入 """
    指令 = input(f"{player_name} 請輸入你的指令 (例如: move 鼠 A1 to B2, list
鼠, attack 牛 C3 D4, help): ").strip().lower()
    return 指令

def 顯示指令說明():
    """ 顯示可用的文字指令說明 (更新版) """
    print("\n--- 指令說明 ---")
    print("list [棋子代號]                                - 列出指定棋子的可移動位
置並編號 (例如: list 鼠)")
    print("move [數字]                                    - 選擇編號移動位置 (需

```

先使用 list 指令，例如: move 1)")

print("move [棋子代號] [起始位置] to [目標位置] - 移動棋子到指定位置
(例如: move 鼠 A1 to B2)")

print("attack [攻擊棋子代號] [目標位置] - 攻擊目標位置的棋子 (例
如: attack 牛 C3 D4)")

print("end_turn - 結束當前回合")

print("help - 顯示指令說明")

print("--- 請注意 ---")

print(" * 棋子代號：鼠, 牛, 虎, 兔, 龍, 蛇, 馬, 羊, 猴, 雞, 狗, 豬, YF
(陰府)")

print(" * 位置： 西洋棋盤座標，例如 A1, H8, C5")

print(" * 使用 'list' 指令後，可以使用 'move [數字]' 快速選擇移動目
標。 \n")

def pos_to_coords(pos):

""" 將棋盤位置 (例如 A1) 轉換為座標 (例如 (0, 0)) """

col = ord(pos[0]) - ord('A')

row = int(pos[1]) - 1

return col, row

def coords_to_pos(coords):

""" 將座標 (例如 (0, 0)) 轉換為棋盤位置 (例如 A1) """

col_char = chr(coords[0] + ord('A'))

row_char = str(coords[1] + 1)

return col_char + row_char

def is_valid_move(piece_name, start_pos, end_pos, board, current_player):

""" 檢查移動是否符合規則 (根據不同棋子類型詳細實作) """

start_col = start_pos[0].upper()

start_row = int(start_pos[1])

end_col = end_pos[0].upper()

end_row = int(end_pos[1])

if not ('A' <= start_col <= 'H' and 1 <= start_row <= 8 and 'A' <= end_col <= 'H'
and 1 <= end_row <= 8):

return False # 位置超出棋盤範圍

```

    if end_pos in board and
board[end_pos].startswith(get_player_prefix(current_player)):
        return False # 目標位置有己方棋子

    piece_type = piece_name.split('_')[0] # 取得棋子類型 (例如 "鼠", "牛", "虎
"...))

    if piece_type == "鼠":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 2 and row_diff == 0) or (col_diff == 0 and row_diff <= 2) or
(col_diff == row_diff and col_diff <= 2 and col_diff != 0): # 直線/斜線 1-2 格
            return True
    elif piece_type == "牛":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff == 0 and row_diff > 0) or (row_diff == 0 and col_diff > 0): # 直線
任意格
            # --- 這裡可以加入路徑是否有阻礙的判斷 (更進階的功能) ---
            return True
    elif piece_type == "虎":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff == 2 and row_diff == 1) or (col_diff == 1 and row_diff == 2): # 馬
步
            return True
    elif piece_type == "兔":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 3 and row_diff == 0) or (col_diff == 0 and row_diff <= 3) or
(col_diff == row_diff and col_diff <= 3 and col_diff != 0): # 直線/斜線 1-3 格
            # --- 這裡可以加入跳過己方棋子，但不跳過敵方棋子的判斷 (更
進階的功能) ---
            return True
    elif piece_type == "龍":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 3 and row_diff <= 3) and (col_diff != 0 or row_diff != 0): # 全

```

方位 1-3 格

```
        return True
    elif piece_type == "蛇":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 2 and row_diff == 0) or (col_diff == 0 and row_diff <= 2): # 直
線/橫線 1-2 格
```

--- 蛇的穿梭能力可以先簡化，不在此函數中判斷，在更進階的版本中再加入 ---

```
        return True
    elif piece_type == "馬":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 3 and row_diff == 0) or (col_diff == 0 and row_diff <= 3): # 直
線/橫線 1-3 格
```

```
        return True
    elif piece_type == "羊":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 1
格
```

```
        return True
    elif piece_type == "猴":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 1
格
```

```
        return True
    elif piece_type == "雞":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 2 and row_diff == 0) or (col_diff == 0 and row_diff <= 2): # 直
線/橫線 1-2 格
```

```
        return True
    elif piece_type == "狗":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 1
```


格

```
        return True
    elif piece_type == "豬":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff == 0) or (col_diff == 0 and row_diff <= 1): # 直
線/橫線 1 格
            return True
    elif piece_type == "YF": # 陰府
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 1
格
            return True

    return False # 其他情況，移動均為無效
```

```
def is_valid_attack(attacker_piece_name, target_pos, board, current_player):
```

```
    """ 檢查攻擊是否符合規則 (根據不同棋子類型詳細實作) """
```

```
    if target_pos not in board:
```

```
        return False # 目標位置沒有棋子
```

```
    target_piece = board[target_pos]
```

```
    if target_piece.startswith(get_player_prefix(current_player)):
```

```
        return False # 不能攻擊己方棋子
```

```
    attacker_pos_list = [pos for pos, piece in board.items() if piece ==
attacker_piece_name] # 找出攻擊棋子的位置
```

```
    if not attacker_pos_list:
```

```
        return False # 攻擊棋子不存在
```

```
    attacker_pos = attacker_pos_list[0] # 假設只找到一個符合的棋子
```

```
    start_col = attacker_pos[0].upper()
```

```
    start_row = int(attacker_pos[1])
```

```
    end_col = target_pos[0].upper()
```

```
    end_row = int(target_pos[1])
```

```
    attacker_piece_type = attacker_piece_name.split('_')[0] # 取得攻擊棋子類型
```

```

if attacker_piece_type == "鼠":
    col_diff = ord(end_col) - ord(start_col)
    row_diff = end_row - start_row
    if col_diff in [1, -1] and row_diff == 1: # 斜線前方一格
        return True
elif attacker_piece_type == "牛":
    # 牛的攻擊在移動時判斷，is_valid_attack 函數中牛不直接攻擊
    return False # 牛沒有獨立的攻擊指令
elif attacker_piece_type == "虎":
    col_diff = abs(ord(end_col) - ord(start_col))
    row_diff = abs(end_row - start_row)
    if (col_diff == 2 and row_diff == 1) or (col_diff == 1 and row_diff == 2): # 馬
        return True # 老虎的攻擊與移動結合，移動到攻擊範圍即可攻擊
elif attacker_piece_type == "兔":
    # 兔子的攻擊在移動時判斷，is_valid_attack 函數中兔子不直接攻擊
    return False # 兔子沒有獨立的攻擊指令
elif attacker_piece_type == "龍":
    # 龍的範圍攻擊需要更複雜的邏輯判斷 (超出簡化範例範圍)
    # 簡化為：龍的攻擊範圍為目標格位周圍一格
    col_diff = abs(ord(end_col) - ord(start_col))
    row_diff = abs(end_row - start_row)
    if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 周圍一格
        return True
elif attacker_piece_type == "蛇":
    # 蛇的米字型攻擊範圍 (超出簡化範例範圍)
    # 簡化為：蛇的攻擊範圍為目標格位周圍一格
    col_diff = abs(ord(end_col) - ord(start_col))
    row_diff = abs(end_row - start_row)
    if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 周圍一格
        return True
elif attacker_piece_type == "馬":
    # 馬的衝鋒攻擊在移動時判斷，is_valid_attack 函數中馬不直接攻擊
    return False # 馬沒有獨立的攻擊指令
elif attacker_piece_type == "羊":

```

```

        # 羊的反擊在被攻擊時觸發，is_valid_attack 函數中羊不主動攻擊
        return False # 羊不主動攻擊
    elif attacker_piece_type == "猴":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 周
            圍一格
            return True
    elif attacker_piece_type == "雞":
        col_diff = ord(end_col) - ord(start_col)
        row_diff = end_row - start_row
        if col_diff == 0 and row_diff == 1: # 前方一格
            return True
    elif attacker_piece_type == "狗":
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 周
            圍一格
            return True
    elif attacker_piece_type == "豬":
        # 豬的範圍衝撞攻擊 (超出簡化範例範圍)
        # 簡化為：豬的攻擊範圍為目標格位周圍一格
        col_diff = abs(ord(end_col) - ord(start_col))
        row_diff = abs(end_row - start_row)
        if (col_diff <= 1 and row_diff <= 1) and (col_diff != 0 or row_diff != 0): # 周
            圍一格
            return True
    elif attacker_piece_type == "YF": # 陰府
        # 陰府不主動攻擊
        return False # 陰府不主動攻擊

    return False # 其他情況，攻擊均為無效

```

```

def 列出可移動目錄(piece_name, start_pos, board, current_player):
    """ 列出指定棋子的所有合法移動目標位置 (更新版 - 加入編號) """
    possible_moves = []

```

```

for col in ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']:
    for row in range(1, 9):
        end_pos = f"{col}{row}"
        if is_valid_move(piece_name, start_pos, end_pos, board,
current_player):
            possible_moves.append(end_pos)

if possible_moves:
    print(f"{piece_name} (位於 {start_pos}) 的可移動位置：")
    for idx, move in enumerate(possible_moves): # 加入編號
        print(f" {idx+1}. {move}")
else:
    print(f"{piece_name} (位於 {start_pos}) 目前無合法移動位置。")

return piece_name, start_pos, possible_moves # 返回 棋子代號, 起始位置,
可移動位置列表

def 執行玩家指令(指令, board, current_player, last_listed=None): # 函數加入
last_listed 參數
    """ 執行玩家輸入的指令，並更新棋盤狀態 (更新版 - 支援數字選擇移動)
    """
    指令_parts = 指令.split()
    if not 指令_parts:
        print("無效指令，請重新輸入。")
        return board, last_listed # 返回更新後的 last_listed

    action = 指令_parts[0]

    if action == "help":
        顯示指令說明()
    elif action == "list":
        if len(指令_parts) != 2:
            print("指令格式錯誤，應為：list [棋子代號]")
            return board, last_listed
        piece_name = 指令_parts[1]
        piece_positions = [pos for pos, piece in board.items() if piece ==
f"{get_player_prefix(current_player)}{piece_name}"]

```

```

    if not piece_positions:
        print(f"未找到 {current_player} 的 {piece_name}，請確認棋子是否
        否存在。")
        return board, last_listed
    start_pos = piece_positions[0]
    last_listed_info = 列出可移動目錄(piece_name, start_pos, board,
    current_player) # 取得包含 棋子代號, 起始位置, 可移動位置列表 的資訊
    return board, last_listed_info # 返回更新後的 last_listed_info
    elif action == "move":
        if len(指令_parts) == 2 and 指令_parts[1].isdigit(): # 判斷是否為 "move
        [數字]" 格式
            if not last_listed:
                print("請先使用 'list [棋子代號]' 查看可移動位置。")
                return board, last_listed
            move_idx = int(指令_parts[1]) - 1
            piece_name, start_pos, moves = last_listed # 從 last_listed 中取出
            資訊
            if 0 <= move_idx < len(moves):
                end_pos = moves[move_idx]
                if is_valid_move(piece_name, start_pos, end_pos, board,
                current_player): # 再次驗證移動的合法性
                    piece = board.pop(start_pos)
                    board[end_pos] = piece
                    print(f"{current_player} 移動 {piece_name} 從
                    {start_pos} 到 {end_pos} (編號 {指令_parts[1]})")
                else:
                    print("選擇的移動位置不合法，請重新選擇。") # 針對編
                    號選擇移動，加入合法性檢查提示
            else:
                print(f"無效的移動選項，請輸入 1 到 {len(moves)} 之間的
                數字。")
        elif len(指令_parts) == 4 and 指令_parts[2] == "to": # 保留原有的 "move
        [棋子代號] [起始位置] to [目標位置]" 格式
            piece_name = 指令_parts[1]
            start_pos = 指令_parts[2] # 原代碼這裡有錯誤，應使用 指令
            _parts[1] 作為 piece_name, 指令_parts[3] 作為 end_pos
            end_pos = 指令_parts[3]
            if is_valid_move(piece_name, start_pos, end_pos, board,

```

```

current_player):
    if start_pos in board and
board[start_pos].startswith(get_player_prefix(current_player)):
        piece = board.pop(start_pos)
        board[end_pos] = piece
        print(f"{current_player} 移動 {piece_name} 從 {start_pos}
到 {end_pos}")
    else:
        print("起始位置沒有你的棋子或棋子不存在。")
    else:
        print("移動不合法，請重新輸入。")
    else:
        print("移動指令格式錯誤，應為：move [數字] 或 move [棋子代
號] [起始位置] to [目標位置]")
    elif action == "attack":
        # ... (attack 指令處理部分保持不變) ...
        pass
    elif action == "end_turn":
        print(f"{current_player} 結束回合。")
        return "end_turn", last_listed # 返回 "end_turn" 標誌, 並傳遞
last_listed
    else:
        print("未知的指令，請輸入 'help' 查看可用指令。")

return board, last_listed # 返回更新後的棋盤狀態和 last_listed

```

```

def 遊戲主循環():
    """ 遊戲主循環，自動設置棋盤 (更新版 - 傳遞 last_listed) """
    board = {}
    player1_pieces = ["鼠", "牛", "虎", "兔", "龍", "蛇", "馬", "羊", "猴", "雞", "狗",
"豬"]
    player2_pieces = ["鼠", "牛", "虎", "兔", "龍", "蛇", "馬", "羊", "猴", "雞", "狗",
"豬"]
    cols = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

    for i, piece in enumerate(player1_pieces[:8]):
        board[f"{cols[i]}1"] = f"1_{piece}"

```

```

for i, piece in enumerate(player1_pieces[8:]):
    board[f"{cols[i]}2"] = f"1_{piece}"
board["D1"] = "1_YF"

for i, piece in enumerate(player2_pieces[:8]):
    board[f"{cols[7-i]}8"] = f"2_{piece}"
for i, piece in enumerate(player2_pieces[8:]):
    board[f"{cols[7-i]}7"] = f"2_{piece}"
board["D8"] = "2_YF"

current_player_num = 1
player_names = {1: "玩家 1", 2: "玩家 2"}
game_over = False
last_listed = None # 初始化 last_listed 為 None

while not game_over:
    current_player = player_names[current_player_num]
    顯示棋盤狀態(board)
    指令 = 取得玩家輸入(current_player)
    執行結果, last_listed = 執行玩家指令(指令, board, current_player,
last_listed) # 傳遞和接收 last_listed

    if 執行結果 == "end_turn":
        current_player_num = 3 - current_player_num
    elif 執行結果 == "game_over":
        game_over = True

print("遊戲結束！")

# ... (其餘程式碼部分保持不變) ...

if __name__ == "__main__":
    print("歡迎來到 陰府棋局：十二生肖的命運之戰 (簡化文字版)")
    print("遊戲開始！ 玩家 1 先行。\\n")
    顯示指令說明()
    遊戲主循環()

```