解析プログラムについて

```
import pandas as pd
from itertools import combinations
import copy
import random # random  モジュールを追加
import math
import numpy as np
from collections import Counter, defaultdict
from scipy.stats import chisquare
from scipy import stats
from typing import Any, Dict, List
from tqdm import tqdm

# ============================================================================
# 1. 牌の定義と変換ユーティリティ
# ============================================================================
# 牌の文字列表現とソート用の内部表現の対応
# z: 1=東, 2=南, 3=西, 4=北, 5=白, 6=発, 7=中
# 内部表現はソート可能かつ一意になるように定義
# 通常の数牌: 11-19 (1m-9m), 21-29 (1p-9p), 31-39 (1s-9s)
# 字牌: 41-47 (1z-7z)
# 赤ドラ: 51 (0m), 52 (0p), 53 (0s) - 通常の 5 とは異なる値とする
TILES_MAP = {
    **{f'{i}m': 10+i for i in range(1, 10)},
    **{f'{i}p': 20+i for i in range(1, 10)},
    **{f'{i}s': 30+i for i in range(1, 10)},
    **{f'{i}z': 40+i for i in range(1, 8)}, # 1z-7z (東南西北白発中)
    '0m': 51, # 赤ドラ 5m
    '0p': 52, # 赤ドラ 5p
    '0s': 53  # 赤ドラ 5s
}
INV_TILES_MAP = {v: k for k, v in TILES_MAP.items()}

def to_str(tile_int):
    """内部表現(int)を文字列に変換"""
    return INV_TILES_MAP.get(tile_int, '?')

def to_int(tile_str):
    """文字列を内部表現(int)に変換"""
    return TILES_MAP.get(tile_str, 0)
```

```
def format_hand(hand_ints, sort=True):
    """手牌リストを整形して文字列で表示"""
    if sort:
        hand_ints.sort()
    return ' '.join([to_str(t) for t in hand_ints])

def parse_tile_list_string(tile_list_str: str) -> List[str]:
    if not isinstance(tile_list_str, str) or not tile_list_str: return []
    tiles, i = [], 0
    while i < len(tile_list_str):
        if i + 1 < len(tile_list_str):
            num_char, suit_char = tile_list_str[i], tile_list_str[i+1]
            if (num_char.isdigit() or num_char == '0') and suit_char in ['m', 'p', 's', 'z']:
                tiles.append(num_char + suit_char); i += 2
            else: i += 1
        else: i += 1
    return tiles

# ==========================================================================
# 牌山生成ロジック
# ==========================================================================
def generate_mahjong_wall():
    """
    オンライン麻雀ゲームで一般的に採用されている、シード値を使わない
    標準的な牌山生成ロジック（全 136 枚）を生成します。
    """
    # 萬子（マンズ）: 1m - 9m 各 4 枚（通常の 5m も含む）
    manzu = [10 + i for i in range(1, 10)] * 4
    # 筒子（ピンズ）: 1p - 9p 各 4 枚（通常の 5p も含む）
    pinzu = [20 + i for i in range(1, 10)] * 4
    # 索子（ソーズ）: 1s - 9s 各 4 枚（通常の 5s も含む）
    souzu = [30 + i for i in range(1, 10)] * 4
    # 字牌（ツーパイ）: 東西南北白発中 各 4 枚
    zihai = [40 + i for i in range(1, 8)] * 4 # 7 種類

    all_tiles = manzu + pinzu + souzu + zihai

    red_dora_ints = [51, 52, 53] # 0m, 0p, 0s の内部表現
    normal_five_ints = [15, 25, 35] # 5m, 5p, 5s の内部表現

    for red_dora_int, normal_five_int in zip(red_dora_ints, normal_five_ints):
        try:
```

```
            idx = all_tiles.index(normal_five_int)
            all_tiles[idx] = red_dora_int
        except ValueError:
            print(f"警告: 通常の牌 {to_str(normal_five_int)} が牌山に見つかりませんでした。")
            return None # 異常終了

    if len(all_tiles) != 136:
        print(f"エラー: 牌山生成数が不正です ({len(all_tiles)}枚)")
        return None

    random.shuffle(all_tiles)

    return all_tiles

# ==========================================================================
# 分析ロジック
# ==========================================================================
def verify_sequence_randomness(row_index,tile_sequence: list[int],analyze_str: str = "",player: str = "",
alpha: float = 0.0025):
    """
    牌の時系列リストを受け取り、そのシーケンスのランダム性を検証する。

    カイ二乗検定（分布の偏り）とランズ・テスト（並びの偏り）を用いて多角的に評価し、
    結果をまとめた辞書を返す。

    Args:
        tile_sequence (list[int]): 検証対象の牌のシーケンス。
        alpha (float, optional): 統計的検定の有意水準。デフォルトは 0.05。

    Returns:
        dict: 各検定の結果を格納した辞書。
    """
    return_value = 0
    n_samples = len(tile_sequence)

    # --- ランズ・テスト (並びの偏り) ---
    # 中央値より上か下かで系列を 2 値化 (1 or 0)
    median_val = np.median(tile_sequence)
    binary_sequence = [1 if x > median_val else 0 for x in tile_sequence]

    # ランの数を数える
    runs = 1
```

```
    for i in range(1, len(binary_sequence)):
        if binary_sequence[i] != binary_sequence[i-1]:
            runs += 1

    n1 = sum(binary_sequence)
    n2 = n_samples - n1

    # Z 検定で p 値を計算
    if n1 == 0 or n2 == 0:
        z_score, p_runs = (0, 1.0) #  全て同じ値の場合はランダムでないが検定不能
    else:
        expected_runs = 2 * n1 * n2 / (n1 + n2) + 1
        variance_runs = (2 * n1 * n2 * (2 * n1 * n2 - n1 - n2)) / ((n1 + n2)**2 * (n1 + n2 - 1))
        std_dev_runs = np.sqrt(variance_runs)

        z_score = (runs - expected_runs) / std_dev_runs
        #  両側検定の p 値
        p_runs = stats.norm.sf(abs(z_score)) * 2

    if p_runs < alpha:
        if analyze_str == "csv":
            print(f"局 index:{row_index} プレイヤー:{player} 検定結果:{p_runs} 対象リスト:{[to_str(t) for t
in(tile_sequence)]}")

        return_value = 1

    return return_value

# ============================================================================
#  配牌とツモリストの生成
# ============================================================================
def analyze_hand_drawn_tile(row_index,wall_ints,analyze_str):
    """
    シャッフルされた内部表現の牌山（int のリスト）から、
    各プレイヤーの初期手牌とツモ牌リストを生成し、
    分析関数が期待する「文字列リスト」形式で返す。
    """
    analyze_true_count = 0
    # --- 配牌とツモリストの生成 ---
    # players 辞書は内部で使用するが、最終的に分析用の全牌リストを返す
    players = {i: {'name': name} for i, name in enumerate(['東家', '南家', '西家', '北家'])}
```

```
current_wall = list(wall_ints) # 入力が整数のリスト wall_ints であることを想定

# 各プレイヤーの初期手牌とツモ牌を結合したリスト（分析用）
all_player_tiles_int = []

# 配牌
for i in range(4):
    players[i]['hand'] = []
    players[i]['hand_draws'] = []

# 4 枚ずつ 3 回
for _ in range(3):
    for i in range(4):
        if not current_wall:
            print("警告: 配牌中に牌山が不足しました。")
            break # 牌山がなくなったら終了
        drawn_tiles = current_wall[:4]
        players[i]['hand'].extend(drawn_tiles)
        players[i]['hand_draws'].extend(drawn_tiles)
        current_wall = current_wall[4:]
    if not current_wall and _ < 2: # 3 回目の 4 枚配り切る前に牌山が尽きた場合
        break

# 1 枚ずつ
if current_wall: # 牌山が残っている場合のみ
    for i in range(4):
        if not current_wall:
            break # 牌山がなくなったら終了
        drawn_tile = current_wall.pop(0)
        players[i]['hand'].append(drawn_tile)
        players[i]['hand_draws'].append(drawn_tile)

for _ in range(18): # 18 巡分
    for i in range(4):
        if not current_wall: break
        players[i]['hand_draws'].append(current_wall.pop(0))
ii = 0
for i in range(4):
    player = players[i]
    all_player_tiles_int = player['hand_draws']
    ii = ii + verify_sequence_randomness(row_index,all_player_tiles_int, analyze_str, player['name'])
```

```
    if ii > 0:
        analyze_true_count = 1

    return analyze_true_count

# ================================================================================
# 実行
# ================================================================================
def read_csv():
    # --- CSV ファイルから牌山を読み込み、なければ生成 ---
    tile_list_str = ""
    try:
        df = pd.read_csv('game_records.csv')
        # CSV の tile_list 列を Python のリストに変換
        tile_list_str = df['tile_List'].iloc[0]
        if tile_list_str == "":
           return ""
    except FileNotFoundError:
        tile_list_str = ""

    return tile_list_str

#    ================================================================================#
実行
# ================================================================================
if __name__ == '__main__':
     num_iterations = 1000000 # 例として 1000 回に設定
     analyze_str = ""

     total_ng_counts = defaultdict(int)
     total_iterations = 0 # 成功した分析回数を記録
     row_index = 0
     wall_list = read_csv()

     if wall_list == "":
         analyze_str = "random"
         for i in tqdm(range(num_iterations), desc="Analyzing Random Walls"):
             row_index = row_index + 1
             wall_ints = generate_mahjong_wall()

             if wall_ints is None:
                 print(f"Iteration {i + 1}: エラー: 牌山の生成に失敗しました。スキップします。")
```

```
                continue # 次の繰り返しに進む
            total_iterations = total_iterations + analyze_hand_drawn_tile(row_index,wall_ints,analyze_str)

    else:
        analyze_str = "csv"
        df = pd.read_csv('game_records.csv')
        num_iterations = 0
        for index, row in df.iterrows():
            row_index = row_index + 1
            tile_list_str = row['tile_List']
            wall_str = parse_tile_list_string(tile_list_str)
            #print(f"{wall_str}")
            wall = [to_int(t) for t in wall_str if t]
            total_iterations = total_iterations + analyze_hand_drawn_tile(row_index,wall,analyze_str)
            num_iterations = num_iterations + 1

# ループ完了後の処理（必要に応じて）
print(f"¥nAnalysis completed for {total_iterations} / {num_iterations} iterations.")
print(f"{analyze_str}")
```

このプログラムは、一体何をしているの？

一言でいうと、**「オンライン麻雀の配牌やツモは、本当にランダム（公平）なのか？それとも何か偏り（かたより）があるのか？」を調べるための探偵プログラム**です。

みんなが遊ぶオンライン麻雀ゲームでは、「牌がシャッフルされて、プレイヤーに配られる」ということが毎回行われます。このプログラムは、その「シャッフル」や「配られ方」が、サイコロを振るのと同じように、完全に偶然で決まっているのか、それとも「特定の牌が特定のプレイヤーに集まりやすい」といった、怪しいパターン（偏り）がないかを、数学的な方法でチェックします。

プログラムは、大きく分けて 2 つの動き方をします。

1. **実際のゲーム記録を分析するモード**: game_records.csv というファイルがあれば、その中に記録された過去のゲームの牌の並びを読み込んで、「このゲーム、怪しい偏りはなかったかな？」と分析します 。
2. **シミュレーションモード**: もしゲーム記録ファイルがなければ、プログラム自身がものすごい回数（このコードでは 10 万回！ ）、自分でランダムな牌山を作ってゲームのシミュレーションをし、「ランダムに作ったら、偏りってどのくらいの確率で起こるんだろう？」というのを検証します 。

それでは、プログラムの中身を順番に見ていきましょう！

プログラムの仕組みを分解してみよう！

プログラムは、いくつかの部品（機能）に分かれています。料理のレシピみたいに、一つ一つの手順を見ていくと分かりやすいですよ。

1. 牌の定義と変換ユーティリティ（麻雀牌の翻訳機）

まず、プログラムは麻雀牌を扱う必要があります。でも、コンピュータは「一萬（イーワン）」や「中（チュン）」といった漢字を直接計算するのが苦手です。そこで、それぞれの牌に**コンピュータが分かりやすい「番号（ID）」**をつけてあげることにしました。これが翻訳機（

TILES_MAP）の役割です 。

- 萬子(マンズ)の「1m」は

11、「2m」は 12 ...

- 筒子(ピンズ)の「1p」は

21、「2p」は 22 ...

- 索子(ソーズ)の「1s」は

31、「2s」は 32 ...

- 字牌(ジハイ)の「東(トン)」は

41、「南(ナン)」は 42 ...

- 特別な赤い牌（赤ドラ）も、「0m」を

51 のように特別な番号にしています 。

こうすることで、コンピュータは数字の大小で牌を簡単に並べ替えたり、区別したりできるようになります。

2. 牌山生成ロジック（カードをシャッフルする人）

これは、ゲームの開始前に、136 枚の全ての牌を使って「牌山（はいやま）」を作る部分です。トランプで遊ぶ前にカードをシャッフルするのと同じですね。

1. まず、萬子・筒子・索子・字牌をそれぞれ全部用意します 。
2. 普通の「5m」「5p」「5s」の中から 1 枚ずつを、特別な「赤ドラ」の牌と交換します 。
3. 最後に、

random.shuffle()という命令を使って、136 枚の牌の順番がぐちゃぐちゃになるように、よーくかき混ぜます 。

ここでしっかりランダムに混ぜることが、「公平なゲーム」の第一歩です。

3. 分析ロジック（怪しいパターンを見つける探偵）

ここがこのプログラムの心臓部、**探偵役**の verify_sequence_randomness 関数です。この探偵は、渡された牌の並びが「偶然できたものか、それとも不自然か」を判定します。

この探偵が使っているすごい技が**「ランズ・テスト」**という統計学のテクニックです。

＜ランズ・テストの簡単なイメージ＞ 例えば、コインを 10 回投げて「表裏裏表裏表表裏裏表」となったとします。 これを、グループ（塊）に分けてみましょう。 「表」「裏裏」「表」「裏」「表表」「裏裏」「表」→7 つのグループ（＝7 ラン）

もし、コイン投げの結果が「表表表表表裏裏裏裏裏」だったらどうでしょう？ 「表表表表表」「裏裏裏裏裏」→2 つのグループ（＝2 ラン） なんだか不自然な感じがしますよね？ランダムなら、もっと表と裏が細かく入れ替わりそうです。

このプログラムも同じようなことをやっています。

1. あるプレイヤーに配られた牌のリスト（手牌とツモ牌）を受け取ります 。
2. 牌の番号の真ん中の値（中央値）を見つけます 。
3. それぞれの牌が、真ん中の値より「大きいか」「小さいか」で、牌の並びを「大・小・小・大...」のような単純な列に変換します 。
4. この「大」と「小」の並びの中に、グループ（ラン）がいくつあるかを数えます 。
5. 数学的に「本当にランダムなら、グループの数はこのくらいになるはず」という期待値が計算できます 。
6. 実際に数えたグループの数が、期待される数と大きくズレていたら、「これは偶然とは考えにくい、何か偏りがあるかもしれない！」と判定します 。

4. 配牌とツモリストの生成（ゲームの進行役）

この部分は、シャッフルされた牌山から、実際に 4 人のプレイヤーに牌を配るシミュレーションをします 。

1. 東家、南家、西家、北家の 4 人を用意します 。

2. 牌山から、実際の麻雀と同じように、各プレイヤーに 13 枚ずつの最初の牌（配牌）を配ります 。
3. さらに、その後 18 回分のツモ牌（順番に引いてくる牌）も各プレイヤーのリストに加えておきます 。
4. こうしてできた「配牌＋ツモ牌」の全リストを、プレイヤーごとに、先ほどの**探偵（分析ロジック）**に渡して、「このプレイヤーの牌の配られ方は不自然じゃないか？」とチェックしてもらいます 。

5. 実行（全体の司令塔）

ここがプログラムのスタート地点（
if __name__ == '__main__':）です。全体の流れをコントロールします 。

1. まず、

game_records.csv というファイルがあるか探しに行きます 。

2. **もしファイルがあったら**、そのファイルに書かれているゲーム記録を 1 行ずつ読み込み、その牌の並びが偏っていないか分析します 。このとき、

プログレスバー（処理の進み具合を示すメーター）が出てきて、あとどれくらいで分析が終わるか教えてくれます 。

3. **もしファイルがなかったら**、シミュレーションモードになります 。自分でランダムな牌山作り（ステップ 2）と、配牌シミュレーション＆分析（ステップ 4）を、設定された回数（ここでは 10 万回）だけひたすら繰り返します。ここでもプログレスバーが出て、大変な作業が今どのくらい進んでいるかを見せてくれます 。
4. 全ての分析が終わったら、最終的に「分析したゲームの総数」と「その中で『偏りがあるかも』と判定されたゲームの数」を表示して、結果を報告してくれます 。

まとめ

このプログラムは、

- **翻訳機**で牌をコンピュータが分かる言葉に直し、
- **シャッフル係**が牌山をランダムにかき混ぜ、
- **ゲーム進行役**がプレイヤーに牌を配るマネをして、
- **探偵**がその配られ方に怪しい点はないか厳しくチェックする

という役割分担で動いています。そして、**司令塔**が全体の流れを管理して、たくさんのゲームを効率よく分析してくれる、とても賢いプログラムなのです。

分析結果
日時　　総局数　観測局数
2022/8/10 18:12　8　　2
局 index:4 プレイヤー:東家 検定結果:0.0005580174102337017 対象リスト:['1m', '2p', '7p', '1s', '4s', '7s', '9s', '1z', '2z', '3z', '4z', '4z', '6z', '7z', '4z', '2p', '8s', '7z', '2z', '5z', '6m', '7m', '0m', '7z', '6p', '7m', '4p', '9p', '1s', '8s', '2m']
局 index:5 プレイヤー:東家 検定結果:0.0005580174102337017 対象リスト:['7m', '8m', '1p', '3p', '8p', '1s', '2s', '3s', '7s', '8s', '9s', '9s', '1z', '5z', '5p', '4m', '8m', '0m', '5p', '8m', '4p', '1s', '7m', '2z', '3s', '1z', '6s', '6z', '1p', '9p', '6m']
局 index:5 プレイヤー:北家 検定結果:0.0005230230780571317 対象リスト:['1m', '1m', '3m', '4m', '6m', '9m', '9m', '3p', '4p', '9p', '6s', '9s', '7z', '6z', '2z', '5z', '4m', '6m', '7m', '0p', '6p', '4s', '4s', '6p', '6z', '1m', '3m', '4z', '2p', '1m', '3m']
Analysis completed for 2 / 8 iterations.

2022/8/11 5:07　8　　0

2022/8/11 15:00　12　　2
局 index:7 プレイヤー:北家 検定結果:0.0005230230780571317 対象リスト:['1m', '4m', '5m', '7m', '8m', '3p', '2s', '5s', '7s', '8s', '9s', '2z', '3z', '7z', '6z', '3z', '5z', '8p', '1p', '9p', '7s', '9m', '1s', '2m', '3p', '8p', '4s', '7z', '4z', '2m', '6p']
局 index:12 プレイヤー:北家 検定結果:0.0019161214301012704 対象リスト:['1m', '6m', '8m', '1p', '3p', '3p', '7p', '9p', '4s', '6s', '3z', '6z', '5z', '3z', '8s', '2z', '1s', '2s', '2s', '7s', '8s', '9s', '6z', '7z', '3m', '2m', '8p', '2p', '5z', '2m', '3s']
Analysis completed for 2 / 12 iterations.

2022/8/11 15:37　14　　6
局 index:1 プレイヤー:南家 検定結果:0.0019161214301012704 対象リスト:['1m', '3m', '7m', '2p', '3p', '5p', '7p', '1s', '8s', '8s', '2z', '4z', '6z', '2m', '1p', '8p', '0m', '6p', '9p', '6m', '3s', '2z', '7z', '7s', '4s', '6s', '4s', '3p', '1m', '1p', '9s']
局 index:5 プレイヤー:東家 検定結果:0.0005580174102337017 対象リスト:['2m', '5m', '9m', '1p', '3p', '4p', '6p', '6p', '8p', '9p', '9p', '3s', '4z', '7z', '6z', '1s', '5s', '8s', '8p', '7p', '2m', '5s', '7m', '1m', '8m', '3m', '2p', '6p', '2p', '9p', '4m']
局 index:7 プレイヤー:東家 検定結果:0.0005230230780571317 対象リスト:['1m', '4m', '0m', '7m', '8m', '2p', '0p', '2s', '3s', '4s', '7s', '8s', '2z', '6z', '1z', '2s', '7z', '1m', '3m', '5p', '7p', '3p', '6z', '5s', '3z', '1p', '2m', '9m', '2p', '9m', '9p']
局 index:10 プレイヤー:南家 検定結果:0.0006368010434241206 対象リスト:['6m', '8m', '1p', '2p', '3p', '7p', '7p', '1s', '1s', '2s', '2s', '6s', '7s', '3s', '2z', '8m', '1s', '6p', '6z', '6z', '3p', '4p', '6m', '9m', '4s', '6z', '4z', '5z', '7s', '2p', '9p']
局 index:11 プレイヤー:東家 検定結果:0.0019161214301012704 対象リスト:['1m', '3m', '8m', '1p', '3p', '4p', '5p', '7p', '2s', '2z', '4z', '6z', '7z', '7z', '2s', '7z', '9p', '5m', '5z', '2p', '7m', '9m', '1p', '6z', '4m', '8p', '8p', '7z', '3z', '7s', '7s']
局 index:14 プレイヤー:東家 検定結果:0.0019161214301012704 対象リスト:['1m', '2m', '2m', '3m', '3m', '6m', '8m', '9m', '2p', '0p', '4s', '5s', '6z', '1z', '9s', '4m', '5m', '1m', '5z', '7z', '1s', '1s', '3s', '6p', '7m', '2s', '1z', '9m', '1p', '9p', '9s']

Analysis completed for 10 / 14 iterations.
csv
2022/8/13 7:54　10　　3
局 index:2 プレイヤー:東家 検定結果:0.0019161214301012704 対象リスト:['9m', '3p', '3p', '3s', '4s', '5s', '6s', '1z', '2z', '2z', '2z', '4z', '2z', '7z', '2p', '3p', '4p', '8s', '5z', '9s', '1m', '4p', '4z', '7z', '9m', '4m', '3m', '1m', '2s', '1p', '7z']
局 index:4 プレイヤー:西家 検定結果:0.0005230230780571317 対象リスト:['2m', '4m', '7m', '8m', '2p', '9p', '2s', '6s', '8s', '9s', '4z', '4z', '5z', '4s', '5s', '9s', '8s', '6s', '5m', '6p', '1s', '8p', '9m', '2p', '3s', '2s', '1s', '0p', '3z', '3s', '1m']
局 index:4 プレイヤー:北家 検定結果:2.667150155169957e-05 対象リスト:['2m', '6m', '7m', '3p', '6p', '7p', '7p', '8p', '9p', '6s', '6s', '9s', '2z', '1m', '1p', '3m', '3p', '5m', '4m', '4m', '5s', '1z', '1s', '1z', '6z', '6z', '6z', '5z', '7s', '3s', '9m']
局 index:10 プレイヤー:東家 検定結果:0.0020539789769586383 対象リスト:['1m', '2m', '2m', '3m', '3m', '5m', '8m', '8p', '3s', '6s', '8s', '9s', '7z', '5z', '1z', '8s', '4s', '5s', '4p', '4s', '3p', '9m', '6p', '4p', '8p', '8p', '1p', '5z', '0p', '7p', '9p']
Analysis completed for 3 / 10 iterations.

2022/8/13 8:28　11　　3
局 index:2 プレイヤー:西家 検定結果:0.0019161214301012704 対象リスト:['4m', '6m', '6m', '2p', '4p', '4p', '1s', '4s', '0s', '7s', '2z', '3z', '4z', '3s', '1s', '2m', '8m', '7m', '7z', '5m', '8s', '6z', '6s', '6z', '3z', '5z', '5m', '8p', '7p', '4s', '5z']
局 index:3 プレイヤー:東家 検定結果:0.00015010770461887715 対象リスト:['3p', '8p', '2s', '4s', '4s', '7s', '1z', '1z', '2z', '2z', '3z', '5z', '6z', '7z', '3m', '6m', '9p', '7m', '7s', '5m', '8s', '6z', '2z', '7s', '3m', '8p', '4p', '7p', '1m', '0s', '8s']

局index:3 プレイヤー:西家 検定結果:0.0019161214301012704 対象リスト:['2m', '2m', '4m', '4m', '5m', '2p', '6p', '7p', '2s', '2s', '4s', '4z', '5z', '8p', '3p', '5p', '2p', '4p', '4m', '5s', '8s', '3z', '1p', '5m', '8m', '7z', '3s', '1s', '8m', '7p', '5z']
局 index:11 プレイヤー:東家 検定結果:0.0019161214301012704 対象リスト:['3m', '4m', '6m', '7m', '8m', '1p', '6p', '8p', '1s', '4s', '5s', '6s', '7z', '4z', '5m', '7p', '4s', '7m', '6s', '9p', '7z', '4m', '5p', '4m', '2m', '9m', '7m', '6p', '9s', '0p', '6z']
Analysis completed for 3 / 11 iterations.