Graph Optimizer for Japanese Mahjong Tile-Discard-Finder

Jarvi Lyrer and Adil Sadikovic
Optimization Methods for Engineers

Introduction: Problem description

- Japanese Mahjong (Richii Mahjong) is a very complex game with a lot of rules and mechanics
- For the sake of simplicity, *we will reduce the game to the following rulesets*: At the start of the game, we put 14 Tiles aside and the 4 players draw 13 Tiles each. On a players turn, that player draws a Tile and then discards one if he does not have a winning hand yet. A winning hand consists of 4 groups of three Tiles (sequences of the same colour or identical Tiles) and a pair of identical Tiles.
- Our **goal**: Given a random Hand consisting of 14 Tiles which all have a colour and a value, find the best Tile to discard and reach a winning hand.
- Tiles can either be grouped into Sequences or Groups of equals and when a Tile belongs to multiple groups, we need to decide which grouping is more beneficial for the Hand.
- Based on the unknown Tiles in the Stack we would like to take the probability of drawing a certain Tile into consideration
- *Our Optimization only covers a part of the real Japanese Mahjong*, We do not consider things like ron (a Player completing their Hand with another's discard) or the other players winning.

Introduction: In mathematical Terms

- The Stack consists of 136 Tiles
- A Hand consists of 14 Tiles
- A Tile is a Tuple $(type, value)$
- There are 34 distinct Tiles, all of which are 4 times in the Stack
- Tiles of type $\{Pin, Character, Bamboo\}$ have values $\{1, \ldots, 9\}$
- Tiles of type $\{North, West, South, East, Red, Green, White\}$ only have value $\{1\}$
- **Shanten** is a positive Integer that denotes the $minimal\ edit\ distance$ for a ***ready Hand***
- A ***ready Hand*** consists of 13 Tiles such that the next draw could complete it to a ***winning Hand***
- A ***winning Hand*** consists of 14 Tiles such that there are 4 Triples and one Pair

The naïve Algorithm

What our naïve Algorithm does:

- Check if there are still Tiles in the Stack; if not end the game in a loss
- Draw a Tile from the Stack
- Calculate Shanten for the current Hand H
- If Shanten = -1 (a winning Hand); end the game as a win
- For all Tiles in our Hand $\mathrm{H} = \{t_1 \ldots t_{14}\}$ check if they contribute to lowering the current Shanten
- Mark those wo do not lower Shanten in the current hand as $possibleDiscard$
- From $possibleDiscard \subseteq H$ choose one Tile t_x at random and discard it
- We end our turn

This is a **greedy** algorithm that uses the knowledge of whether a tile is currently part of Shanten or not to find a discard that for sure does not worsen ones Hand H. It runs in $O(14) \leq O(1)$

The DrawAnalyzer Algorithm

What our improved Algorithm does:

- Generate a graph of distinct combinations of unordered draws up to $depth$ draws
- Use the deepest layer of the graph to find the tile that can be discarded in the most scenarios

Our **goal** is to…

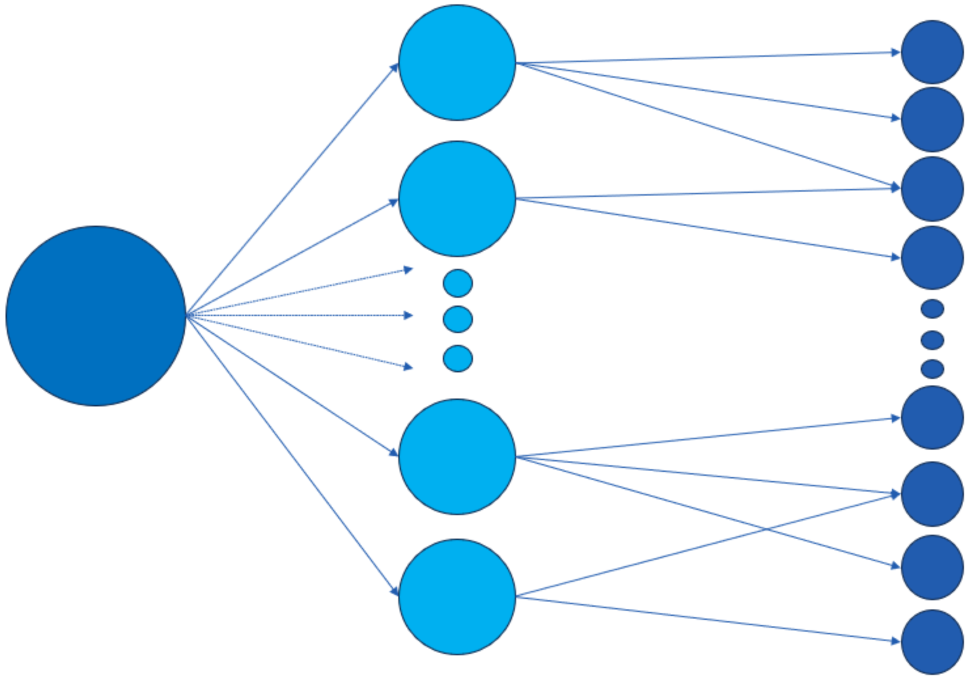
1. Improve the Win/Game ratio
2. Improve the reliability of our discard
3. Limit the average discard time to a maximum of 20s

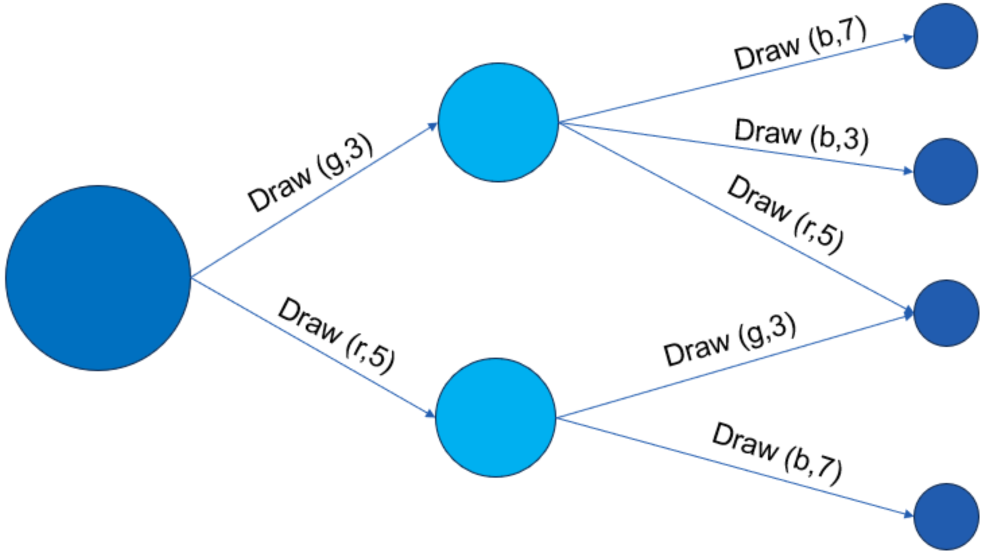
The DrawAnalyzer Graph

How we build the graph:

- Given a depth n:
- Generate the first $\binom{34}{1}$ vertices which represent a draw of 1 unique tile and calculate the number of possibilities of drawing that Tile.
- Generate the next $\binom{34+i-1}{i}\ for\ 2 \leq i \leq n$ vertices of distinct combinations of i tiles, calculate the number of possibilities of drawing that set of tiles and add an edge from a previous vertex to the new one if the new vertex is a draw order reachable from the previous one
- A total of $\sum_{i=1}^{n}\binom{34+i-1}{i}$ vertices will thus be generated
- Traversing the graph simulates following a draw order

DrawAnalyzer visualization





Example Graph with depth 2

The current Gamestate is connected to all 34 possible Tile-Draws, which in turn are connected to further draws.

With some more Detail

Gamestates that have had the same draws in a different order are considered as equal but more likely than other states.

DrawAnalyzer best discard

How we calculate the best discard:

- Given our previously generated graph, we iterate over the deepest layer until $depth > \#draws\ left$, then we iterate over the first layer to safe computation time
- For each vertex, we calculate the number of possibilities for this set of tiles to be drawn
- We then **simulate drawing the tiles** and calculate the Shanten
  - For n tiles drawn, we generate up to $2(14+n)^3 + 2(14+n)^2$ groups. However, the average will be significantly lower at an estimated $(14+n)^2$.
  - We then **generate all possible combinations** of up to 5 groups and calculate $\mathrm{G} = \{(t, a) \in T \times \mathbb{N} \mid t \notin \bigcap_{s \in S} s \cap H, a = comb\}$ where T is the set of all unique Tiles, S is the set of sets of unique tiles that form the largest possible groups in the given hand, H is the set of unique tiles in the hand and comb the possible number of combinations for drawing the set of tiles given by the vertex.
  - Finally, for each element in G, we add the value a to the tile t of the original hand and discard the tile with the highest number, thus the one that is discardable in most cases

How we compare the Algorithms

How much data

- We run 1000 Games on our naïve Algorithm and on DrawAnalyzer with depth 1
- We run 100 Games on DrawAnalyzer with depth 2 because it takes significantly longer

Significant measurements

- We measure the time it takes to draw per round
- We measure the average Shanten per round
- We measure in what round a Game is won (if at all)

Machine Specifics

- 2th Gen Intel(R) Core(TM) i5-12600K (16 CPUs), ~3.7GHz 32GB Ram NVIDIA GeForceRTX 2060

Results: Naïve Algorithm

1000 Games with the Naïve Algorithm

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Total Time	54.40266																	
Winning Hands	109																	
Avg Discard Time	0.00309																	
Avg Shanten per Round	3	3	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0
Avg Time per Round	0	0.00100	0.00118	0.00147	0.00165	0.00215	0.00246	0.00289	0.00296	0.00326	0.00379	0.00437	0.00411	0.00503	0.00504	0.00522	0.00603	0.00293
Ready Hand is reached	0	1	1	0	0	4	1	2	1	8	9	8	13	8	12	19	13	9

Results: DrawAnalyzer with depth 1

1000 Games of DrawAnalyzer with depth 1

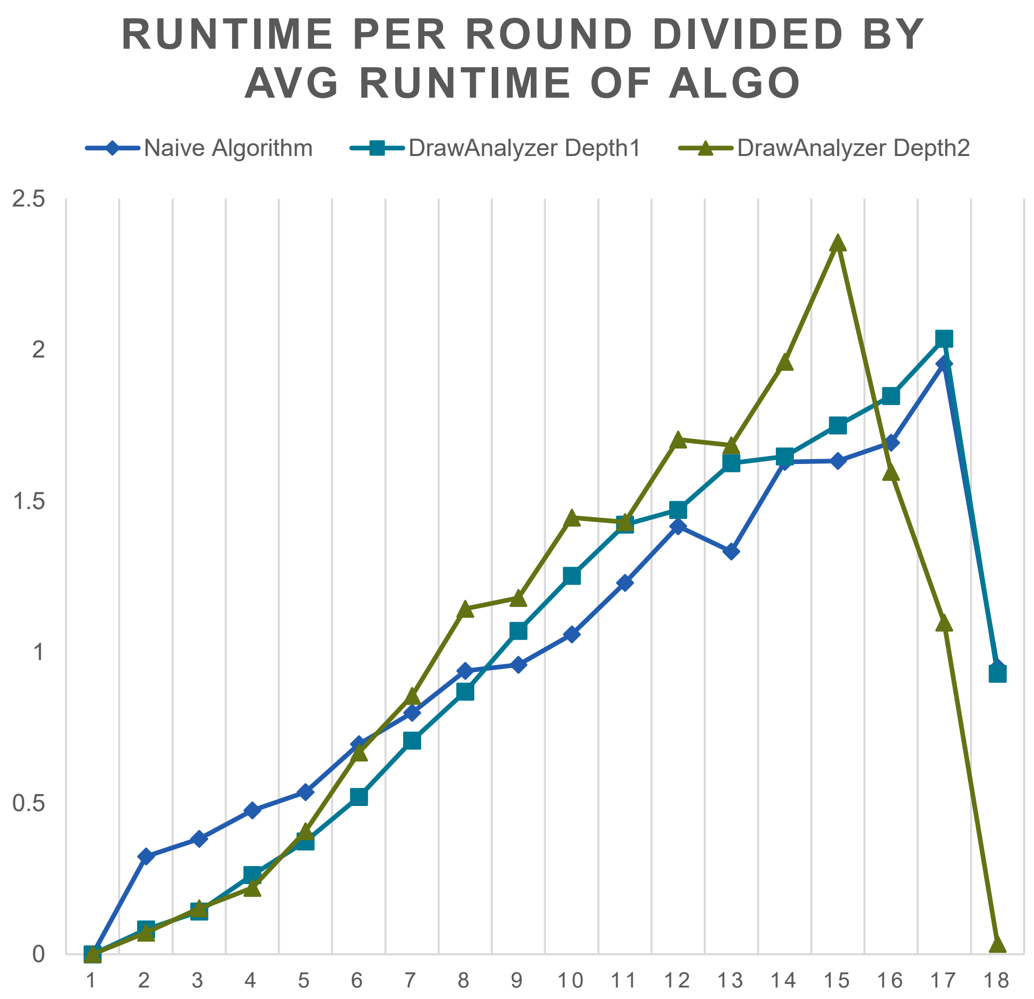
Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Total Time	5661.174																	
Winning Hands	196																	
Avg Discard Time	0.307256																	
Avg Shanten per Round	3	2	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Avg Time per Round	1.16E-05	0.02518	0.04354	0.08048	0.11459	0.15996	0.21706	0.26701	0.32887	0.38456	0.43677	0.45172	0.49916	0.50591	0.53749	0.56738	0.62565	0.28518
Ready Hand is reached	0	0	0	1	1	3	7	2	7	11	11	16	17	28	20	29	30	13

Results: DrawAnalyzer with depth 2

100 Games of DrawAnalyzer with depth 2

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Total Time	19402.21																	
Winning Hands	29																	
Avg Discard Time	11.69604																	
Avg Shanten per Round	3	2	2	2	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Avg Time per Round	0.00316	0.83298	1.77464	2.56912	4.76824	7.80642	9.99863	13.3688	13.7893	16.8931	16.7270	19.9140	19.6917	22.9336	27.5460	18.6651	12.8424	0.40397
Ready Hand is reached	0	0	0	0	0	1	0	0	1	5	0	1	2	2	6	6	2	3

Comparing Results I



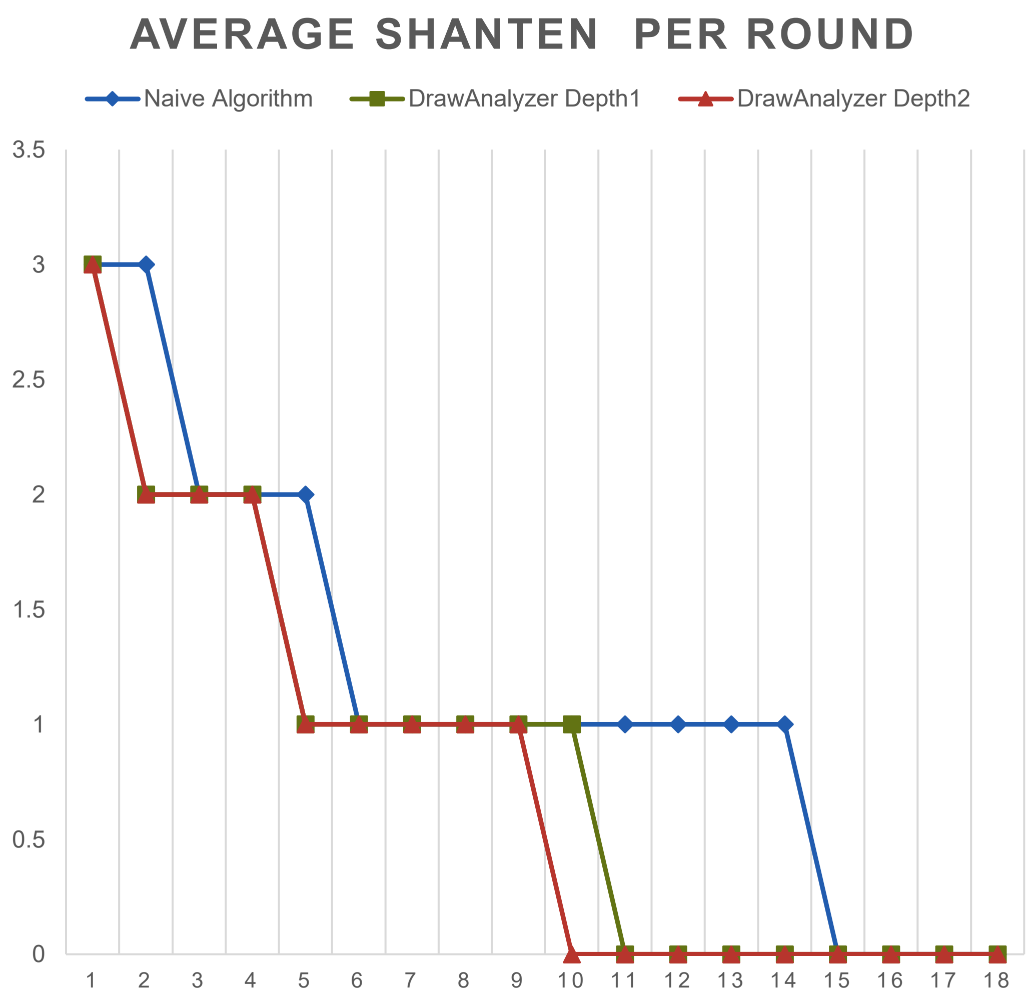
Comparison of times

- The naïve Algorithm takes an average of $3 * 10^{-3}s$ per draw
- DrawAnalyzer with depth 1 takes an average of $0.3s$ per draw
- DrawAnalyzer with depth 2 takes an average of $11.7s$ per draw

- DrawAnalyzer with depth 1 takes on average 90times longer than the naïve algorithm
- DrawAnalyzer with depth 2 takes ~3500times longer than the naïve algorithm on average

- Runtimes tend to be longer the more the game progresses, unless in the last few rounds

Comparing Results II



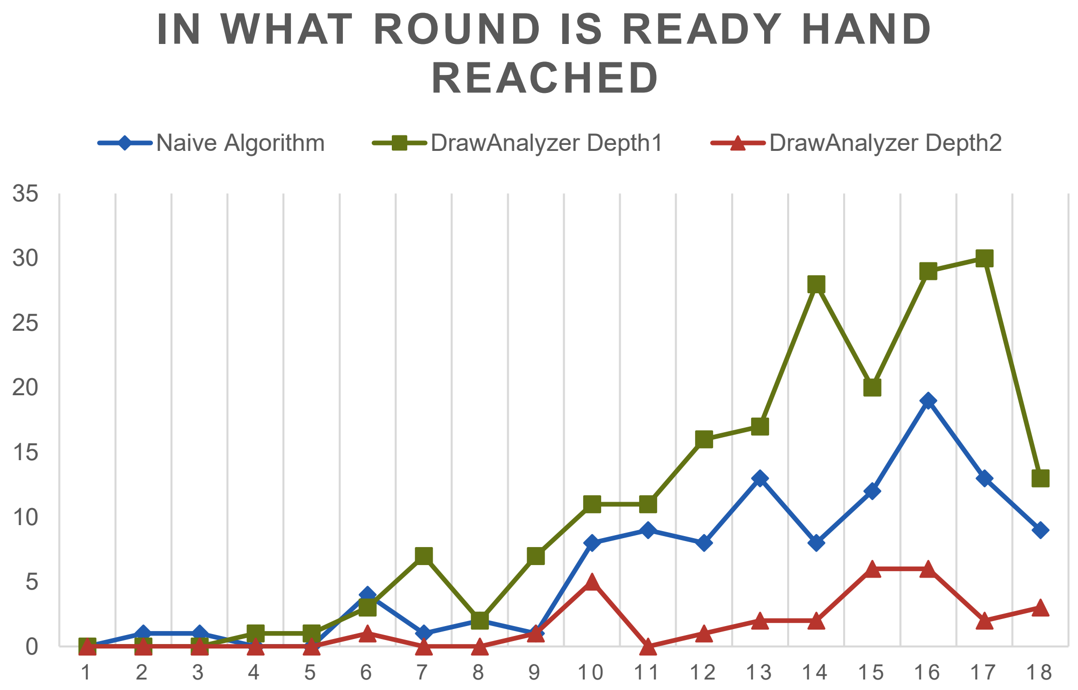
Winning Hands reached

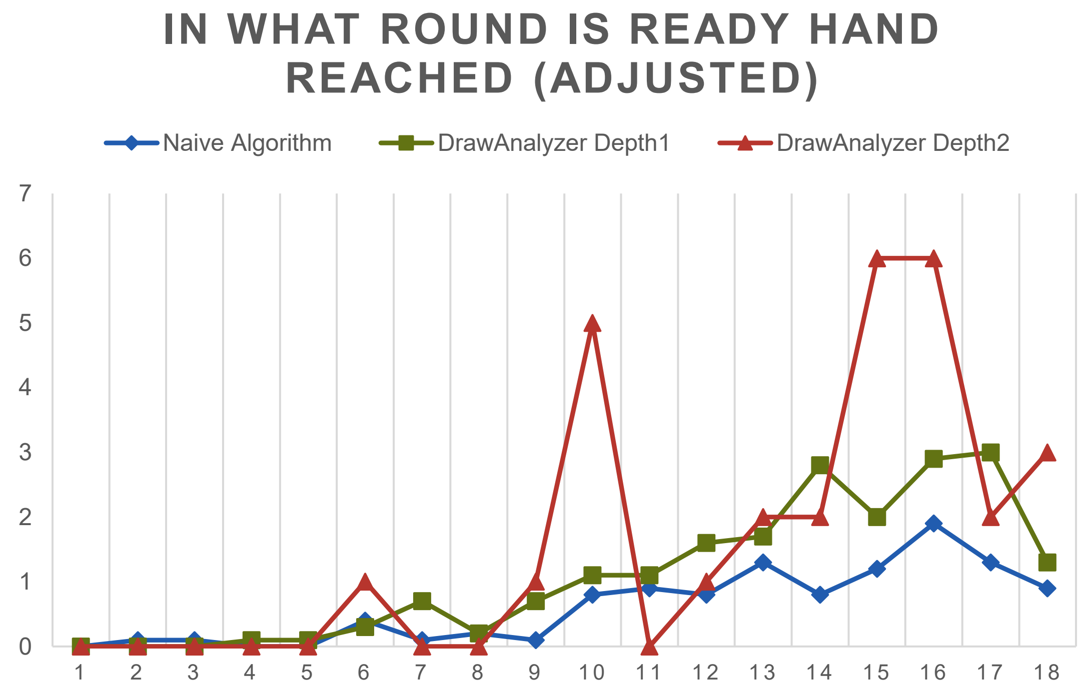
- With the naïve Algorithm ~11% of rounds end in a ready Hand
- With the DrawAnalyzer depth 1 ~20% of rounds end in a ready Hand
- With the DrawAnalyzer depth 2 ~30% of rounds end in a ready Hand

Improving Shanten

- The average ***Shanten*** per round improves significantly earlier when using DrawAnalyzer of either depth, which implies faster ***winning Hand*** due to better decisions.

Comparing Results III





Ready Hands are reached more consistent

In this unadjusted Diagram we see that DrawAnalyzer for depth1 clearly outperforms our naïve algorithm

Here we have adjusted the Diagram for DrawAnalyzer with depth 2. We also see a trend here that it would perform significantly better than the other two algorithms.

Discussion and Conclusion

- **We improve the success rate** from ~11% to >20% even with depth 1 of DrawAnalyzer practically doubling it at the cost of factor 90 in terms of calculation time
- With higher depth of DrawAnalyzer the performance seems to be improved even more; towards 30% success rate but with a cost of roughly factor 3500, which seems to be unproportionate.
- We see that the calculation time for DrawAnalyzer improves in the last few rounds because the depth is capped by the remaining tiles on the stack.
- **Shanten gets improved earlier** in the game with both iterations of DrawAnalyzer, which shows that the algorithm constantly performs goal-oriented and makes good decisions throughout the game.
- **DrawAnalyzer with depth 1 seems to be the current best solution** in terms of usability, because the DrawAnalyzer with depth 2 takes to long to calculate.

Links

GitHub Repository

- https://github.com/adidell01/MahjongOptimization

Image1 / Game Explanation

- https://www.tatlerasia.com/lifestyle/entertainment/hk-mahjong-beginners-guide-how-to-play