



An Efficient Method for Assessing the Strength of Mahjong Programs

Shih-Chieh Tang¹^a, Jr-Chang Chen^{2,*}^b and I-Chen Wu^{1,3}^c

¹Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

²Department of Computer Science and Information Engineering, National Taipei University, New Taipei City, Taiwan

³Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan

Keywords: Mahjong, Stochastic Board Game, Skill Assessment.

Abstract: Mahjong, a tile-based game, is a complex four-player stochastic game of imperfect information involving both strategy and luck. Due to its inherent randomness, accurately assessing the strength of players requires a large number of games, which is time-consuming. This randomness primarily originates from two factors: (1) the initial arrangement of the wall and (2) tile stealing by players. Both affect the tiles players draw and thus influence game outcomes. To address the effect of these factors, especially the randomness introduced by stealing, we propose a novel method, called the *stable draw wall* (abbr. *SDW*). The *SDW* partitions the original wall into individual sub-walls for each player, ensuring that the tile drawing order of each player remains consistent and does not change by stealing from any player. The experimental results showed that when playing a small number of games, the win rate of a player by using the *SDW* is more accurate than by using the original wall. Consequently, our proposed method significantly mitigates the randomness effect caused by changing the order of draws, allowing a more reliable evaluation of the strength of players, which should focus on strategic decision making.

1 INTRODUCTION


Mahjong is a traditional tile-based game that originates in China and is popular in eastern Asia. It is a four-player stochastic imperfect information game. The game involves strategy and a degree of luck, as players aim to complete a winning hand by drawing, stealing, and discarding tiles. There are many games that include randomness during the gameplay, such as Texas Hold'em, Blackjack, and Chinese dark chess. Mahjong's gameplay is complex due to large number of tiles and rounds, and hidden information. The number of information sets and the average size of the information sets are 10^{121} and 10^{48} , respectively. This indicates that Mahjong has more hidden information than bridge and Texas Hold'em, making it challenging to develop a strong Mahjong AI (Li et al., 2022).


The inherent randomness in Mahjong competitions requires a greater number of games for researchers and contest organizers to accurately assess the strength of players. The outcome of Mahjong

competitions is often influenced by randomness, providing weaker players with opportunities to win. Although the element of randomness in competition can provide excitement and tension, it concurrently decreases the precision in evaluating the strength of players. Thus, more games are necessary to generate a stable assessment.

Two primary factors are identified as contributing to this randomness: (1) the initial arrangement of the wall and (2) the decisions made by the players to steal tiles. The first factor, the initial arrangement of the wall, plays a crucial role because the players draw the tiles in a predetermined order. If no player steals a tile during the game, the order of tiles drawn from the wall is the same for all players, assuming that the same wall is reused. In competitions for computer program players, such as the Computer Olympiad, multiple games are often played using the same wall, with players switching seats between the games. (Lin et al., 2011; Chen and Chen, 2022). After playing these games, the same initial hands will be dealt to all players, preventing any particularly good or bad hand from being experienced by only a subset of players. This method mitigates the effect of randomness from the initial arrangement of the wall, allowing for

^a <https://orcid.org/0009-0002-6678-711X>

^b <https://orcid.org/0000-0002-7973-2049>

^c <https://orcid.org/0000-0003-2535-0587>

*Corresponding author

a more stable assessment of the strength of the players. However, the second factor, stealing by players, also changes the order of draws. For example, if a player steals a tile, he/she will forgo his/her next draw. Consequently, the subsequent tile may instead be drawn by another player, leading to a different order of draws. Moreover, players' decisions to steal tiles affect not only his/her immediate draw but all draws of every player in the future. Thus, the player's outcomes are often changed by stealing, even though he/she plays with the same wall and the same strategy.

In this paper, we introduce a method for the construction of a specialized wall, called *stable draw wall* (abbr. *SDW*), which is designed to significantly alleviate the impact of the change in the draw order by stealing during the game. The main idea is to partition the original wall into *subwalls* for all players so that each player only draws tiles from their own subwalls. This method prevents drawing another tile caused by stealing. Note that the SDW must be used with the aforementioned method, which uses the same wall in multiple games and switches players' seats across these games. Thus, by using our proposed wall structure, the negative effect on a player can be reduced when an opponent makes a different choice. This, in turn, makes the players' actions more decisive in determining the outcome of the games. Finally, our goal is to distinguish the relative strength of two computer players more efficiently and to use a smaller number of games to accurately evaluate their relative win rates using the SDW.

The rest of this article is organized as follows. In section 2, we review some Mahjong competition platforms and Mahjong agents. In Section 3, we present our methods for constructing the SDW and using it in Mahjong game. In Section 4, we present the experimental results. In Section 5, we make the concluding remarks.

2 BACKGROUND

In this section, we briefly review the general rules of Mahjong in Subsection 2.1 and related works on Mahjong competition platforms and player programs in Subsection 2.2.

2.1 Rules of Mahjong

We introduce the rules and terms of Taiwanese Mahjong. There are 144 tiles in Mahjong game, categorized as four *types*, 34 *patterns*, and flowers.¹

¹Flower tiles are excluded in this paper.

These types are categorized into three *suits* and an *honor*. The suits consist of 27 patterns which are numbers 1 to 9 *Character* (or *Man*, represented by C_1 to C_9), 1 to 9 *Dot* (or *Pin*, denoted by D_1 to D_9), and 1 to 9 *Bamboo* (or *Sou*, denoted by B_1 to B_9). The honor consists of four *Winds* (East, South, West, and North) and three *Dragons* (White, Green, and Red). Each pattern has four identical tiles.

To set up the initial game state, all tiles are shuffled, placed face down, and arranged into the *wall*. Starting with the dealer, each player draws four tiles at a time from the front of the wall, repeating this process four times. These 16 tiles form the player's initial *hand*. The goal of each player is to complete a *winning hand*, typically consisting of five sets and one pair. The players take turns *drawing* a tile from the front of the wall or *stealing* a discarded tile from an opponent to complete their winning hand. Stealing includes *chow*, *pong* and *gong*. Chow signifies that a player takes a tile discarded by the left player in the previous turn and forming a sequence (three consecutive number tiles of the same suit) with it. Pong signifies that a player takes a tile discarded by any other player in the previous turn and forming a triplet (three identical tiles). Gong signifies that a player takes a tile discarded by any other player in the previous turn, forming a quadruplet (four identical tiles), and must then pick another tile. After drawing or stealing a tile, if a player accomplishes a winning hand, he/she wins the round; otherwise, they must discard a tile. The game ends when a player completes a winning hand or when only 16 tiles remain in the wall, which is called the *dead wall*.

We introduce additional ways to draw tiles from the wall. In addition to the standard draw, players can also draw tiles after applying some specific actions such as the gong. Unlike the standard drawing, where players take a tile from the front of the wall, drawing after these actions requires taking a tile from the back of the wall, specifically from the dead wall. In these cases, the tile drawn from the dead wall is referred to as a *supplementary tile*.²

2.2 Related Works

In this section, we introduce research related to Mahjong, focusing primarily on studies involving competition platforms and computer player programs. In Subsubsection 2.2.1, we present several platforms that provide interfaces for interaction with computer player programs. In Subsubsection 2.2.2, we discuss research on various computer player programs, high-

²For more information, please refer to <http://mahjong-europe.org/>.

lighting those that have been ranked or actively participated in competitions hosted on these platforms.

2.2.1 Competition Platforms

In the context of AI-driven Mahjong competitions, two notable studies have provided important contributions. (Lin et al., 2011) proposed a tournament framework for computer Mahjong competitions. This framework focused on organizing and facilitating fair and competitive environments for AI agents playing Mahjong. The authors addressed key aspects such as game scheduling, ranking systems, and the handling of randomness in the game, ensuring that AI players were evaluated under standardized conditions. Specifically in handling randomness, the framework used a wall arrangement in several games and rotated the seats of players. This framework was influential in the promotion of the development and evaluation of AI Mahjong programs by providing a structured competitive platform. It had been used in the Mahjong contests of Computer Olympiad until 2021. Similarly, (Chen and Chen, 2022) designed a Mahjong framework that was extended from the existing framework of Chinese dark chess. The framework also used the same method as in (Lin et al., 2011) to handle the problem of randomness.

BOTZONE is an online multi-agent competitive platform designed for AI education (Zhou et al., 2018). It supports various competitive games, including Mahjong, allowing students and researchers to develop, test, and improve AI agents. The platform provides multiplayer real-time environments and extensive logging of game data, which are valuable for analyzing AI agent performance. BOTZONE’s flexibility and accessibility has made it a widely used platform in both educational and research settings, promoting the development of AI strategies in competitive gaming environments.

Mjx is an open source Mahjong framework for Riichi Mahjong (Koyamada et al., 2022). This framework aimed to improve execution speed and provide human-friendly framework.

2.2.2 Mahjong Player Programs

We introduce some Mahjong player programs in variant rules. In Taiwanese rules, (Chen et al., 2022) designed a computer Mahjong program SIMCAT, using Monte Carlo simulation techniques to improve decision making. The program generated hands after applying each legal action and simulated the win rate of these hands using an optimistic strategy. The program selected the action whose hand, after applying it, obtained the best win rate. Furthermore,

SIMCAT designed heuristic methods to handle some special cases for better performance. (Lin and Lin, 2021) designed a computer Mahjong program SEOFON, which evaluated a hand by deconstructing its composition and excluded unnecessary deconstructions based on the deficiency number. Throughout the game, SEOFON collected information from discarded tiles, which was then used to infer the tiles the opponents likely wanted. In the end game, this information was crucial in defense strategies and in predicting the number of tiles remaining of each type in the wall.

In Japanese rules (Riichi Mahjong), (Mizukami and Tsuruoka, 2015) built the program BAKUUCHI, which adopted Monte-Carlo simulation and trained policy models and opponent models using supervised learning. SUPHX was developed by (Li et al., 2020) and used supervised learning and reinforcement learning to train models. It also used global reward prediction, oracle guiding, and parametric Monte-Carlo policy adaptation to improve performance.

3 OUR METHODS

This section describes the method of constructing the SDW and the usage of the SDW during the game. The SDW ensures that when the SDW is used several times in multiple games, the player in the specific seat will draw the same tile in the same round. For example, a player draws C_1 in the i -th round of the first game. When playing the second game using the SDW, the player sitting in the same seat will draw C_1 in the i -th round as well.

We introduce the method for constructing the SDW in Subsection 3.1 and the wall usage in Subsection 3.2.

3.1 Design of the SDW

We introduce the method for constructing the SDW from a given original wall. The SDW consists of four sets, each containing a *front subwall* and a *rear subwall*, and each player owns one set. The front subwall contains the tiles which players draw during normal play, and the rear subwall contains the supplementary tiles which players draw from the end of the original wall after stealing by gong. Let $W^o = [w_0^o, w_1^o, \dots, w_{135}^o]$ denote the arrangement of 136 tiles w_i^o in the original wall, where $i = 0, \dots, 135$. Let $FSW_p = [fsw_{p,0}, fsw_{p,1}, \dots, fsw_{p,n_p-1}]$ and $RSW_p = [rsw_{p,0}, rsw_{p,1}, rsw_{p,2}, fsw_{p,3}]$ denote the front subwall and the rear subwall of the player p , respectively, where $p \in \{0, 1, 2, 3\}$ and n_p is the number of tiles in

the front subwall of p . When $p = 0$, p is the dealer.

The steps to construct the front subwall are as follows. First, we take 16 tiles as each player's initial hand from the original wall and place them in the player's own front subwalls. According to the Mahjong rules mentioned in Subsection 2.1, a game starts from the dealer, and then each of the four players takes turns picking four tiles from the front of the original wall and repeats this process four times. Eq. 1 shows that $fsw_{p,4n+k}$, the $(4n+k)$ -th tile in the front subwall of p , is retrieved from the $(16n+4p+k)$ -th tile in the original wall.

$$fsw_{p,4n+k} = w_{16n+4p+k}^o \quad (1)$$

where $0 \leq n, k \leq 3$. Next, each player takes turns drawing tiles from the original wall until only 16 tiles remain in the dead wall. Thus, excluding 4×16 tiles in hands and 16 tiles in the dead wall, there are 56 tiles that can be drawn by four players during gameplay. These tiles are placed sequentially in the front subwall of each player, so 14 tiles are added in each front subwall. Eq. 2 shows that $fsw_{p,16+k}$, the $(16+k)$ -th tile in the front subwall of p , is retrieved from the $(64+4k+p)$ -th tile in the original wall.

$$fsw_{p,16+k} = w_{64+4k+p}^o \quad (2)$$

where $0 \leq k \leq 13$. Hence, the front subwall of each player consists of 30 tiles.

The steps to construct the rear subwall are as follows. Beginning with the dealer once again, each player takes turns picking a tile from the end of the dead wall and placing it in his/her own rear subwall. Eq. 3 shows that $rsw_{p,k}$, the k -th tile in the rear subwall of p , is retrieved from the $(135 - (4k+p))$ -th tile in the original wall. Hence, the rear subwall of each player consists of 4 tiles.

$$rsw_{p,k} = w_{135-(4k+p)}^o \quad (3)$$

Note that the idea of the rear subwall is the same as that of the front subwall, but is from the end of the original wall. More specifically, a player always draws the same tile by a gong no matter whether another gong by other players occurs before. Algorithm 1 shows the pseudocode for constructing the SDW from an original wall.

3.2 Using the SDW in Gameplay

We apply the constructed SDW during a Mahjong game. The players take their initial hand and draw tiles from their own front subwall. Two key issues must be addressed when using the SDW in a game. First, a player should draw the $(16+i)$ -th tile from the front subwall at the i -th round. However, if a player

Function CONSTRUCTING_SDW:

Input: W^o : List of 136 tiles arranged in the original wall.

Output: FSW , RSW : List of front subwalls FSW_p and rear subwalls RSW_p , respectively, where $p \in \{0, \dots, 3\}$. The subwalls are also lists.

```

/* Construct hand tile part of
front subwalls. */
for n = 0 to 3 do
  for p = 0 to 3 do
    for k = 0 to 3 do
      fswp,4n+k ← Wo[idx];
      FSWp.push_back(fswp,4n+k);
      idx ← idx + 1;
    end
  end
end
/* Construct remaining part of
front subwalls. */
for k = 0 to 13 do
  for p = 0 to 3 do
    fswp,16+k ← Wo[idx];
    FSWp.push_back(fswp,16+k);
    idx ← idx + 1;
  end
end
/* Construct rear subwalls. */
idx ← 0;
for k = 0 to 3 do
  for p = 0 to 3 do
    rswp,k ← Wo[135 - idx];
    RSWp.push_back(rswp,k);
    idx ← idx + 1;
  end
end

```

Algorithm 1: Pseudocode of Constructing the SDW.

steals a tile at the i -th round instead, he/she will draw the $(16+i)$ -th tile at the $(i+1)$ -th round, which is supposed to draw the $(17+i)$ -th tile. This is inconsistent with our purpose: to prevent drawing another tile caused by stealing. Second, a player may draw more than 14 tiles, exhausting all tiles in his/her front subwall. This happens because, although the drawn tiles are fixed using the SDW, the playing order may be changed due to stealing. As a result, some players may draw more tiles than others. For example, after a player steals the discarded tile from the player on his/her right side by pong, the turn goes back, and that player draws one more tile.

To address the first issue, when a player steals a tile at the i -th round, it implicitly indicates that he/she relinquishes the opportunity to draw a tile. We move

the first tile from his/her front subwall to a pile, called a *relinquished-tile pile*. Hence, we ensure that the tile drawn at the $(i + 1)$ -th round is exactly the $(i + 1)$ -th tile in his/her front subwall. Combined with drawing tiles only from the front subwall of each player, the subsequent drawn tiles will not be changed by his/her stealing.

To address the second issue, we design a method called *reshuffle*. Let n_r be the total number of tiles in the relinquished-tile pile and the eight subwalls. Assume that the turn goes to the player p_{turn} , who exhausts all tiles in his/her front subwall or rear subwall. We collect the n_r tiles, and all subwalls become empty. These tiles are shuffled into an arrangement $W' = [w'_0, w'_1, \dots, w'_{n_r-1}]$, and then are redistributed to four players, similar to the method described in Subsection 3.1. More specifically, $n_r - 16$ tiles are used to construct front subwalls. Let $q = (p - p_{turn}) \bmod 4$, where q represents the position of the player p relative to p_{turn} . For example, if $p = 3$ and $p_{turn} = 2$, $q = 1$, representing p is the next player of p_{turn} . Starting from p_{turn} , each player takes turns draw a tile from W' and place it in his/her front subwall. Thus, the arrangement of tiles in each player's front subwall is shown in Eq. 4.

$$fsw_{p,k} = w'_{4k+q} \quad (4)$$

where $k \geq 0$ and $4k + q \leq n_r$. Then, we use the last 16 tiles to construct rear subwalls. The arrangement of tiles in each player's rear subwall is shown in Eq. 5.

$$rsw_{p,k} = w'_{n_r-(4k+q)} \quad (5)$$

where $0 \leq k \leq 3$. The detailed implementation for reshuffling the SDW is presented in Algorithm 2.

After the reshuffle, the game resumes with p_{turn} by drawing a tile from his/her reshuffled front subwall. Algorithm 3 presents the pseudocode for the entire procedure of drawing a tile from the SDW.

4 EXPERIMENTS

In the experiments, we analyzed the efficiency for the assessment of the relative strength of game-playing programs using the SDW. We used the game-playing program, SIMCAT (Chen et al., 2022), and created a weaker variant, called SIMCAT- ϵ , whose strength is adjusted by the parameter ϵ . More specifically, SIMCAT- ϵ selected the action given by SIMCAT with a probability of $1 - \epsilon$ or a random action with a probability of ϵ . Note that to prevent a significant drop in the strength of programs, random actions of SIMCAT- ϵ were restricted to those that maintain the deficiency number. For example, for the hand $\{C_2C_2C_2C_6C_9\}$,

Function RESHUFFLE_SDW:

Input: FSW, RSW : List of front subwalls FSW_p and rear subwalls RSW_p , respectively, where $p \in \{0, \dots, 3\}$. The subwalls are also lists.
 RTP : List indicating the pile of the relinquished tiles.
 p_{turn} : an integer indicating the current player.

Output: None.

```

/* Collecting the tiles remained in
subwalls. */
W' ← [];
for FSWp in FSW do
    while FSWp is not empty do
        t ← FSWp.front();
        W'.append(t);
        FSWp.pop_front();
    end
end
for RSWp in RSW do
    while RSWp is not empty do
        t ← RSWp.front();
        W'.append(t);
        RSWp.pop_front();
    end
end
for rt in RTP do
    W'.append(rt);
end
/* Reshuffle them. */
nr ← W'.size();
idx ← 0;
for k = 0 to 3 do
    for p = 0 to 3 do
        rswp,k ← W'[nr - idx];
        RSWp.push_back(rswp,k);
        idx ← idx + 1;
    end
end
p ← pturn;
while W' is not empty do
    fswp,k ← W'.front();
    FSWp.push_back(fswp,k);
    W'.pop_front();
    p ← (p + 1) mod 4;
end

```

Algorithm 2: Pseudocode of Reshuffle.

$C_2C_2C_2$ is a triplet, and discarding a tile from the triplet makes the deficiency number increase, so the random actions considered only include C_6 and C_9 . Obviously, SIMCAT- ϵ is stronger with a smaller ϵ . In the experiments, the values of ϵ were set to 1.0, 0.5,

Function DRAW_A_TILE_FROM_SDW:
Input: p : an integer that indicate the player.
 $draw_from_rear$: True if drawing from rear or not.
 $is_relinquished_draw$: True if player stole a tile and relinquished to draw.
Output: t : NULL or the tile drawn from the SDW.

```

if  $is\_relinquished\_draw$  is True then
   $t \leftarrow$  NULL;
   $rt \leftarrow$   $FSW_p$ .front();
   $RTP$ .push_back( $rt$ );
   $FSW_p$ .pop_front();
else
  if  $draw\_from\_rear$  is True then
    if  $RSW_p$  is empty then
      reshuffle( $FSW$ ,  $RSW$ ,  $RTP$ ,  $p$ );
    end
     $t \leftarrow$   $RSW_p$ .front();
     $RSW_p$ .pop_front();
  else
    if  $FSW_p$  is empty then
      reshuffle( $FSW$ ,  $RSW$ ,  $RTP$ ,  $p$ );
    end
     $t \leftarrow$   $FSW_p$ .front();
     $FSW_p$ .pop_front();
  end
end

```

Algorithm 3: Pseudocode of Drawing Tiles from the SDW.

and 0.2. There are two teams, one using SIMCAT and the other using SIMCAT- ϵ . Two players in each team used the same program and sat on the opposite sides of the square table. To mitigate the effects of the initial wall arrangement, each wall was played twice, with players rotating to the seat on their right side after the first game as mentioned in Section 1.

The experiments were conducted on a computer with an AMD Ryzen 5 2600 6-core processor and 32GB of memory. In Subsection 4.1, we analyzed the average number of actions and stealing by players. In Subsection 4.2, we compared the consistency of the draws between the original wall and the SDW. In Subsection 4.3, we analyzed data on reshuffles that occurred when using the SDW. Finally, in Subsection 4.4, we compared the win rate and the error between the original wall and the SDW in a small number of games.

4.1 The Number of Actions in a Game

We analyzed the number of actions as the parameter ϵ varies from a large to a small value. Let n_{steal} denote the average number of stealing actions by a player per

game. Let n_{total} denote the total number of actions, including stealing and drawing, by a player per game. Let $r_{steal} = n_{steal}/n_{total}$ denote the the frequency of stealing by a player per game.

The experimental results are shown in Table 1. First, n_{steal} increases as ϵ decreases. The reason is that the program with a smaller ϵ has a higher possibility of choosing to steal. Second, n_{total} decreases as ϵ decreases. The reason is that stronger programs win a game more quickly, resulting in fewer actions. The trends of n_{total} and n_{steal} are opposite, with one increasing and the other decreasing as ϵ varies. Third, r_{steal} ranges from 13.26% to 16.13%. Fourth, whether using the original wall or the SDW have very little influence on the number of actions, both n_{steal} and n_{total} . This suggests that using the SDW instead of the original wall almost does not affect the duration of a game and the frequency of stealing.

Table 1: Average count of draw and stealing.

Opponent		$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
Original	n_{steal}	1.31	1.44	1.50
	n_{total}	9.88	9.61	9.33
	r_{steal}	13.26%	14.98%	16.08%
SDW	n_{steal}	1.31	1.44	1.50
	n_{total}	9.86	9.61	9.30
	r_{steal}	13.29%	14.98%	16.13%

We divide the course of a game into five intervals based on the number of actions. The ratios of actions within each interval are shown in Table 2. Most games finish when a player makes 5 ~ 14 actions, ranging from 89.26% to 92.90%.

Table 2: Ratios of actions.

n_{total}	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
0 ~ 4	3.04%	3.07%	3.22%
5 ~ 9	43.78%	47.19%	51.45%
10 ~ 14	45.48%	44.11%	41.14%
15 ~ 19	7.70%	5.63%	4.20%
≥ 20	0.00%	0.00%	0.00%

A player can steal at most five times in a game. Table 3 shows the percentage of games based on n_{steal} . Given an ϵ , each column shows the ratio that a player makes n_{steal} stealing actions. When $n_{steal} = 0$, it indicates that the player did not steal in the game. By observing the first row, the ratio of no stealing actions decreases from 24.42% to 15.37% as ϵ decreases. It indicates that more stealing actions are made for a stronger program. Moreover, a player steals less than two times in most games.

Table 3: Ratios of stealing actions.

n_{steal}	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
0	24.42%	17.51%	15.37%
1	36.00%	36.96%	36.44%
2	26.34%	31.27%	32.88%
3	11.14%	12.32%	13.25%
4	2.06%	1.89%	2.02%
5	0.05%	0.04%	0.04%

4.2 Consistency in Draws

We investigate whether a player can draw the same tile when other players may take different stealing actions. Assume that the wall $W = [w_0, w_1, \dots, w_{135}]$, where w_0, \dots, w_{63} are used in the initial hands of all players. If no stealing is allowed, the player p will pick $w_{64+4k+p}$ in the k -th round. If stealing is allowed and the tile player p draws in the k -th round is the same as $w_{64+4k+p}$, we call the draw *consistent* with W . The *consistent rate* of a game log to W is the ratio of consistent draws among all draws, that is, the number of consistent draws divided by the number of all draws.

We compared the consistent rates of the logs played with the original wall and with the SDW. The experimental results are shown in Table 4. The data reveal that when using the original wall, the consistent rates for all ϵ ranged from 20.98% to 23.43%, indicating that on average, 76.57% ~ 79.02% of the drawn tiles were affected by changes in the order of draws caused by stealing. This result demonstrates that such a high percentage of tiles is changed, so that simply rotating the player seat, as discussed in Section 1, is insufficient to mitigate randomness in the game. In contrast, when using the SDW, the consistent rate increased to 94.72% ~ 95.00%, indicating that only 5.00% ~ 5.28% of the tiles were different. This result shows that the use of the SDW effectively reduces the likelihood of changes in the tiles drawn by the players. Consequently, in competition, the difference in the tiles drawn by players who sat in the same seat is significantly reduced.

Table 4: Ratio of consistent draws.

Consistent rate	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
Original	20.98%	22.49%	23.43%
SDW	94.72%	94.71%	95.00%

4.3 Effect of Reshuffle

We analyze the influence of reshuffle described in Subsection 3.2 on the consistent rate. In a game, let n_{rsf} be the times of reshuffles, and let n_{rad} be the num-

ber of the available draws in the SDW when reshuffling.

Table 5 shows the percentage of 20,000 games based on n_{rsf} . By observing $n_{rsf} = 0$, most games do not need to reshuffle, ranged from 87.10% to 92.47%. When ϵ decreases, the percentage of games with $n_{rsf} = 0$ increases, indicating that the times of reshuffles decrease. It may be caused by more early termination of games mentioned in Subsection 4.1, so there are still tiles in the front wall when a game ends. Moreover, by observing $n_{rsf} = 1$, for games need to reshuffle, most of them are reshuffled only once.

Table 5: The average times of reshuffle in a game.

n_{rsf}	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
0	87.10%	90.37%	92.47%
1	11.12%	8.48%	6.91%
2	1.61%	1.02%	0.54%
3	0.18%	0.13%	0.09%
4	0.08%	0.01%	0.00%
≥ 5	0.00%	0.00%	0.00%

Let $g_{n_{rad}}$ be the number of games shuffled with n_{rad} tiles. The average n_{rad} is calculated by dividing the weighted sum of $g_{n_{rad}}$, where each n_{rad} is multiplied by $g_{n_{rad}}$, by the total number of games as follows.

$$\text{Average } n_{rad} = \frac{\sum_{n_{rad}=1}^{56} (n_{rad} \times g_{n_{rad}})}{\sum_{n_{rad}=1}^{56} g_{n_{rad}}}$$

Although the maximum number of draws is 56, the average n_{rad} are 6.54, 7.47, and 8.11 when $\epsilon = 1.0$, 0.5, and 0.2, respectively. When ϵ decreases, the average n_{rad} increases, indicating that reshuffles occur earlier. The reason may be that programs with lower ϵ steal more tiles as mentioned in Subsection 4.1, so more drawing turns of players were skipped as mentioned in Subsection 3.2. It makes more possibly happen that some players have more turns, so they exhaust all tiles in his/her front subwall and need to reshuffle in earlier stage.

Table 6 shows the results of n_{rad} in the first reshuffle only. By adding the values of n_{rad} between 1 and 15, the percentages of games whose reshuffle happen when there are less than or equal to 15 draws range from 98.22% to 99.64%.

A concluding remark is as follows. When using the SDW, at least 87.10% of all games are unaffected by reshuffle. Moreover, among the affected games (at most 12.90%), most of them draw the same tiles during the first 41 ($= 56 - 15$) draws. Hence, only a small number of draws in all games is changed by reshuffle.

Table 6: The remaining available draws in the first reshuffle.

n_{rad}	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
1 ~ 5	25.72%	17.08%	13.98%
6 ~ 10	61.55%	61.42%	60.35%
11 ~ 15	12.37%	20.46%	23.89%
16 ~ 20	0.35%	1.04%	1.77%
≥ 21	0.00%	0.00%	0.00%

Table 7: Comparison between the two walls.

(a) 500 games in a match

Opponent	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
wr_{gr}	71.58%	61.70%	54.90%
Avg. <i>err</i> (Original)	1.88%	1.54%	1.73%
Avg. <i>err</i> (SDW)	1.27%	1.51%	1.29%

(b) 1,000 games in a match

Opponent	$\epsilon = 1.0$	$\epsilon = 0.5$	$\epsilon = 0.2$
wr_{gr}	71.58%	61.70%	54.90%
Avg. <i>err</i> (Original)	1.33%	1.27%	1.45%
Avg. <i>err</i> (SDW)	0.73%	0.93%	0.90%

4.4 Competitions Using Different Walls

We compare the accuracy of win rates using the original wall and the SDW. We play a total of 20,000 games using the original wall and compute wr_{gr} , the win rate of SIMCAT, as the ground truth. Next, let a match consist of a small number of games such as 500 or 1,000. For each match, we compute the win rate wr of SIMCAT and the error $err = wr - wr_{gr}$ that represents the deviation between the match and the ground truth. To obtain more accurate experiment results, we play several matches and compute the average errors of them.

In Table 7a, 40 matches of 500 games are played. The average errors are 1.54% ~ 1.88% for the original wall and 1.27% ~ 1.51% for the SDW. In Table 7b, 20 matches of 1,000 games are played. The average errors are 1.27% ~ 1.45% for the original wall and 0.73% ~ 0.93% for the SDW. Both results show that the error values for matches using the SDW are consistently lower than those using the original wall for all ϵ . When playing a small number of games, using the SDW can obtain more reliable win rate than using the original wall. Moreover, the average errors of 1,000 games are reduced more than those of 500 games, as more games provide better accuracy.

5 CONCLUSIONS

In this paper, we proposed a newly designed wall for Mahjong, called the stable draw wall (SDW). The

SDW prevents 94.72% to 95.00% of the drawn tiles from being changed due to an opponent's stealing. By using the SDW, the impact of randomness from stealing is alleviated, making the players' actions more decisive in determining the outcome of the games. The experimental results show that the win rate using the SDW is more accurate compared to the original wall when only a small number of games are played. Hence, if we want to distinguish the relative strength of players by playing fewer games due to time constraints in real competitions, using the proposed SDW instead of the original wall is more likely to achieve it.

There are still many interesting topics for future research. The remaining 5% to 5.28% of the draws that can be changed due to stealing require further investigation. It is worthwhile to develop a clever design to manage this. Our idea to design the SDW can be extended to other stochastic games, including other variants of Mahjong, tile-based games, and card games. A fast evaluation system for assessing the strength of human and program players can also be designed based on our proposed method.

ACKNOWLEDGEMENTS

This research was partially supported by National Science and Technology Council (NSTC) of Taiwan under grant numbers 113-2221-E-305-004-MY3 and 113-2221-E-A49-127-.

REFERENCES

- Chen, J.-C., Tang, S.-C., and Wu, I.-C. (2022). Monte-carlo simulation for mahjong. *Journal of Information Science & Engineering*, 38(4):775–790.
- Chen, K.-C. and Chen, J.-C. (2022). Design and implementation of computer mahjong platform. Master's thesis, National Taipei University (in Chinese).
- Koyamada, S., Habara, K., Goto, N., Okano, S., Nishimori, S., and Ishii, S. (2022). Mjx: A framework for mahjong ai research. In *2022 IEEE Conference on Games (CoG)*, pages 504–507. IEEE.
- Li, J., Koyamada, S., Ye, Q., Liu, G., Wang, C., Yang, R., Zhao, L., Qin, T., Liu, T.-Y., and Hon, H.-W. (2020). Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*.
- Li, J., Wu, S., Fu, H., Fu, Q., Zhao, E., and Xing, J. (2022). Speedup training artificial intelligence for mahjong via reward variance reduction. In *2022 IEEE Conference on Games (CoG)*, pages 345–352. IEEE.
- Lin, C.-H., Shan, Y.-C., and Wu, I.-C. (2011). Tournament framework for computer mahjong competitions. In *2011 International Conference on Technologies and*

Applications of Artificial Intelligence, pages 286–291. IEEE.

- Lin, Z.-H. and Lin, S.-S. (2021). Using the enhancement strategy from discarded-tiles information to improve mahjong program. Master's thesis, National Taiwan Normal University (in Chinese).
- Mizukami, N. and Tsuruoka, Y. (2015). Building a computer mahjong player based on monte carlo simulation and opponent models. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 275–283. IEEE.
- Zhou, H., Zhang, H., Zhou, Y., Wang, X., and Li, W. (2018). Botzone: an online multi-agent competitive platform for ai education. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 33–38.

