applied sciences

MDPI

Article

A Rule-Based AI Method for an Agent Playing Big Two

Sugiyanto, **Gerry Fernando** and **Wen-Kai Tai** *

Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan; d10615813@mail.ntust.edu.tw (S.); m10615814@mail.ntust.edu.tw (G.F.)
* Correspondence: wktai@mail.ntust.edu.tw

Abstract: Big Two is a popular multiplayer card game in Asia. This research proposes a new method, named rule-based AI, for an agent playing Big Two. The novel method derives rules based on the number of cards the AI agent has left and the control position. The rules for two to four cards left are used to select the card combination to discard based on the number of cards remaining in the agent's hand. The rules for more than four cards left conditionally prioritize discarding the card combination in the classified cards with lower priority. A winning strategy provides guidelines to guarantee that the AI agent will win when a win is achievable within three moves. We also design the rules for the AI agent without control for holding cards and splitting cards. The experimental results show that our proposed AI agent can play Big Two well and outperform randomized AI, conventional AI, and human players, presenting winning rates of 89.60%, 73.00%, and 55.05%, respectively, with the capability of maximizing the winning score and minimizing the number of cards left when the chance of winning is low.

Keywords: game bot; agent; rule-based AI; imperfect information; Big Two

check for updates

Citation: Sugiyanto; Fernando, G.; Tai, W.-K. A Rule-Based AI Method for an Agent Playing Big Two. *Appl. Sci.* **2021**, *11*, 4206. https://doi.org/10.3390/app11094206

Academic Editors: Tobias Meisen and Federico Divina

Received: 1 January 2021
Accepted: 28 April 2021
Published: 5 May 2021

1. Introduction

Developing an agent (artificial player) for playing a strategy game (e.g., chess, mahjong, and Big Two) is crucial [1]. The players' decision-making skills can significantly determine the outcome. The experience of playing strategy games with a developed agent can help human players to improve their decision-making skills. Human players can practice and play the game anytime they want. Playing a game will be more exciting and popular if there are smart opponents in the game [2]. The first famous agent to beat a reigning world champion in a game was the chess program called Deep Blue [3]. Since then, research on how to develop an agent that can play against human players has gained a lot of traction [4,5]. Big Two is a multiplayer card game that is incredibly popular in China and other Asian countries. Although this game has many rule variations, it has one primary aim: each player has 13 cards and tries to discard them all as soon as possible based on valid card combinations. This study proposes an approach for developing an artificial intelligence (AI) agent to play Big Two [6–8].

Previous research has explained the framework of our web-based Big Two game [6]. Big Two requires four players to play. The game needs AI agents that can join if there are not enough players. Preliminary research in developing an AI agent has commonly used rule-based AI [9]. AI agents for playing games such as first-person shooter [10], Super Mario Bros. [11], Ms. Pac-Man [12–14], samurai [15], Battleship [16], Nomic game [17], Domineering [18], poker [19], and Xiangqi [20] have successfully been developed. One advantage of rule-based AI is its flexibility when encoding and modifying the system in the future [21]. Although some studies have developed sophisticated AI agents for playing games such as Air War [22], Mythic RPG [23], Chinese Chess [24], Sudoku [25], the fighting game [26], Star Trek [27], and Hanabi card [28], they used rule-based AI as an essential part of the new method. Developing sophisticated AI agents means they used sophisticated methods (e.g., fuzzy rule, deep learning) that involve computationally complex tasks.

Other studies of multiplayer card games such as poker [29], Hearts [30,31], Magic [32], Spades [33], 7 Wonders [34], Hanabi [35], and Scopone [36] have used rule-based AI as a baseline player as well.

This research proposes a new method, named rule-based AI, for an agent playing Big Two. The novel method includes rules for situations in which two to four cards are left, there is a winning strategy, four more cards are left, and the agents are not in the control position. The rules for situations in which two, three, and four cards are left are used to select the card combination that is to be discarded based on the number of cards the AI agent has left. A winning strategy provides guidelines for the AI agent to guarantee that the AI agent will win when a win is achievable within three moves. The rules for occasions in which more than four cards are left prioritize conditionally discarding a card combination in the card class with the lower priority. If an agent is not in a control position, the agent discards a card combination of lower-class classification. We designed the rules for an AI agent who was not in a control position to include the hold-back function for checking whether the selected combination is essential for holding, and the split-card function for splitting a five-card and a pair when an opponent only has one card remaining to win. In addition, we introduced the bi-five-card function for finding two five-card combinations when the agent has more than ten cards left, and the best-five-card function for determining the best five-card combination when the agent has more than one possible five-card combination.

The experimental results show that our AI agent plays well and significantly outperforms every opponent in all the experiments that have been conducted. We performed three experiments to evaluate the performance of our rule-based AI agent. In each experiment, the proposed AI agent played against randomized AI, conventional AI, and human players. The proposed AI achieved winning rates of 89.60%, 73.00%, and 55.05%, respectively, better than all opponents with winning rates of 10.40%, 27.00%, and 44.95%. Overall, rule-based AI played 6000 games, winning 4353 games (72.55%) and losing 1647 games (27.45%). The proposed AI agent ended the game with higher positive scores (won) than those of opponents. Additionally, the proposed AI agent could maximize the winning score when winning, and minimize the number of cards left when the chance of winning was low.

The main contributions of this research are as follows. (1) We introduce a card classification system to classify all the cards in the agent's hand based on a list of cards that have been played. In Big Two, it is very imperative to ensure which combinations can take control by 100% (we called it class A), which combinations have the potential to take control (class B), which combinations do not possess the potential to be promoted into class A (class C), and which combinations cannot be played without control (class D). We introduced a card classification system to identify them and used it as an essential part of rule-based AI. (2) We found two essential features: the number of cards left in the agent's hand, and the control position (control or no control) of the agent. Additionally, our proposed AI with several rules achieved high winning rates; therefore, further research can use this rule-based AI as a baseline player or consider these rules when defining policies for new methods. Additionally, this AI could potentially be used to give hints to beginners or players hesitating about their turn. (3) This study demonstrates the development of AI agents using limited resources, utilizing only a 3.40 GHz Intel Core i7-6700 and 16 GB of random-access memory. Game developers with limited resources can use this method to develop a lightweight AI agent. With the limited resources, the AI agent took less than one second to decide a selected card combination for each of its turns.

The rest of the paper is laid out as follows. Section 2 describes the related work of developing an agent to play a game. Section 3 specifies the proposed method to develop an AI agent for playing Big Two. In Section 4, we show and discuss the experimental results of the proposed rule-based AI. Section 5 presents the conclusion and possible future work.

2. Literature Review

2.1. Basic Rules of Big Two

Big Two is a card game with exactly four players [6–8]. Each player is dealt 13 cards and plays them using valid combinations. Valid combinations in this game are one card, two cards (a pair), and five cards. The rank of the cards is 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A < 2. If two cards have the same number, then the winner is the card with the highest suit. The rank of suits is diamond (D) < club (C) < heart (H) < spade (S). The rank of five-card combinations is straight (five consecutive cards) < flush (five cards of the same suit) < full house (three cards of one rank + a pair) < four of a kind (four cards of one rank + a single card) < straight flush (five consecutive cards of the same suit).

The player who has the three of diamonds (3D) starts the game. The player can be in one of two positions during the game: in control or not in control. The player who wins the round is in control of the next round of play, and they can lay down any valid card combination. Other players can only discard a high-card combination following the type of card played or choose to pass. The winner's score is derived by totaling the number of cards opponents have left. For example, if Player 1 wins and the three opponents have two, five, and nine cards left, then the winner's score would be 16. Each opponent receives a negative score equal to their remaining number of cards. Using the same example, the opponents would score -2, -5, and -9.

2.2. Agent-Based Gaming

Agents in the game are non-player characters (NPCs) behaving by programmed algorithms [37]. In this article, we use the term "AI agent" to refer to a computer player (e.g., Deep Blue, AlphaGo) that uses particular algorithms to determine the best strategy to compete with players. AI agents have individually owned variables that describe their internal state and conduct certain computations or run instructions. In multiplayer games, the game developer usually provides AI agents as opponents to play with the human players if the game does not have enough human players.

The algorithm which developers commonly uses to make the AI agent is the Min-Max algorithm [38], which assumes that the opponent always performs optimal actions. In Tic-Tac-Toe [39], the Min-Max algorithm is used to predict steps and win matches. The algorithm evaluates the minimum opponent's chances and maximum AI agent's chances of the win. The technique that can optimize the Min-Max algorithm is Alpha-Beta Pruning [40]. Two extra parameters are passed in the Min-Max function: alpha and beta. At each node in the game tree, alpha stores the maximum score, and beta stores the minimum score. When applied to a Min-Max tree, this technique prunes away branches that are unlikely to be the final decision to save computation time. However, the Min-Max algorithm and Alpha-Beta Pruning are typically used in perfect information games where both players have the same information about the game, such as chess, Tic-Tac-Toe, and Go. It can be time-consuming to compute moves for games with high branches.

Rule-based AI [9] is the most widely used method for agent-based gaming such as first-person shooters [10], Super Mario Bros. [11], Ms. Pac-Man [12–14], samurai [15], Battleship [16], Nomic game [17], Domineering [18], poker [19], and Xiangqi [20]. In their simplest form, rule-based AI consists of a set of if–then-style rules used to make actionable decisions. Rules, primarily conditional statements, are stored in unordered clusters. Rules are matched with existing facts. The collection of knowledge and facts is a dynamic set, constantly being adjusted. It describes what is known about the current state of the game. In rule-based AI, various conditions are carefully driven to achieve better performance. Rule-based AI is relatively easy to program and manage because the knowledge encoded in the rules is modular. This advantage gives some flexibility both when coding the game and modifying the game at a later time.

2.3. Related Work

Previous research has shown that our web-based Big Two is of good quality and is able to run smoothly [6]. A critical issue concerning this game is the number of players. Big Two requires precisely four players in one room to play. If there are not enough players, the player must wait in the game lobby. The game needs a lightweight AI agent that can handle this situation. If there are not enough players at the specified time, the AI agent will play against human players. Several previous studies have succeeded in creating agents for playing first-person shooters [10], Super Mario Bros. [11], Ms. Pac-Man [12–14], samurai [15], Battleship [16], Nomic game [17], Domineering [18], and Xiangqi [20] using rule-based AI. They chose to use rule-based AI because it is easy to implement and requires fewer computational resources. Rule-based AI uses a set of if–then statements to select actions based on several conditions in the game [9]. In game AI, intelligence is built through adequate rules with effective execution. However, all of these studies have been based on games with perfect information. In the games with perfect information, all the information about the actions and state of the game is available. Therefore, the computer can analyze all possible actions and strategies of the opponent [41]. Big Two is a multiplayer card game with imperfect information. The computer has incomplete information about the actions and state of the game [42,43]. The opponent's hand is hidden; thus, the computer has difficulty analyzing the possible actions and strategies of the opponent.

Developing an AI agent for playing multiplayer games with imperfect information is challenging because such games involve multiplayer interactions, imperfect information, and computationally complex tasks. The use of rule-based AI has successfully developed an AI agent for playing a type of poker called No-Limit Texas Hold'em [19]. During each agent's turn, the agent will follow three steps sequentially: (1) reading all the information from the poker table; (2) searching the most suitable rules for the table circumstances; and (3) choosing the rule to follow. The AI agent successfully inferred strategies from the human player in one-on-one matches, but there were no consistent results in multiplayer games against humans.

We have successfully developed a rule-based AI agent for playing Taiwanese mahjong, and the AI is currently being used in the industry [5]. The proposed mahjong AI is easy to adjust for reducing players' waiting time and improving players' gaming experience. However, the main difference between mahjong and Big Two is the acceptable combinations. In mahjong, a combination can be either a chow (three consecutive tiles of the same suit), a pong (three identical tiles), or a kong (four identical tiles). In Big Two, it is more complex that a combination can be either a five-card (straight, flush, full house, four of a kind, and straight flush), a pair, or a single. Although mahjong uses four tiles for every three suits—characters, dots, and bamboos—that are numbered from one to nine, there is no ranking (e.g., a kong from the nine of dots and kong from the six of bamboos is the same value). In contrast, Big Two is very specific about ranking and card value. The rank of five-card combinations is straight < flush < full house < four of a kind < straight flush. The rank of the cards is 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A < 2. The rank of suits is diamonds < clubs < hearts < spades. The ranking system and the number of possible combinations make Big Two have more computationally complex tasks than mahjong.

The Big Two game is rarely studied, and thorough research is lacking in the literature. Although there are several servers where one can play Big Two online, those servers that feature AI agents playing against humans are scarce, and the AI algorithm itself is not yet open to the public. Some experts have tried to formalize how to play Big Two. They described a systematic approach [7] and some probability calculations [8] for playing Big Two. A systematic approach is used to decide which play will maximize the chances of winning. However, this approach is a narrative form and is unstructured. In this paper, we transform the strategies into a computational approach as a conventional AI and a baseline player in the second experiment.

Table 1 illustrates the complexity of Big Two depending on the number of cards left in hand. The formula for calculating tree size is B^D. B is the total number of possible moves from all actions (i.e., five-card, pair, single, and pass action). D is the average length of Big Two. We have observed 250 games and found out that the average length of Big Two is 37.3725 turn. Big Two has a complicated action space with up to 1379 possible actions and tree size of up to 2.15×10^{117}. Applying sophisticated methods is not suitable for game developers with limited resources due to computational resource usage. Rule-based AI appears to be the correct method for developing a lightweight AI agent that can play Big Two.

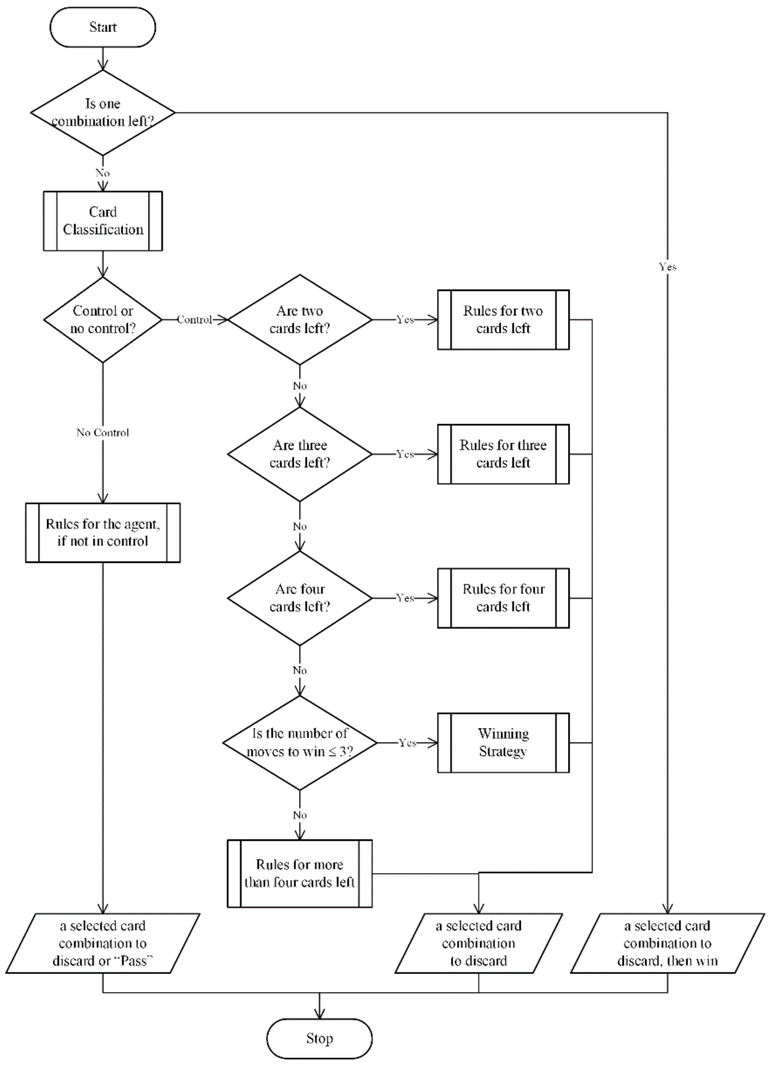
Table 1. The complexity of Big Two.

Number of Player Cards (n)	**Five-Card Actions (nC_5)**	**Pair Actions (nC_2)**	**Single Actions (nC_1)**	**Pass Action**	**Possible Moves (B)**	**Average Game Length (D)**	**Strategic Complexity ($\sqrt{B/D}$)**	**Tree Size (B^D)**
13	1287	78	13	1	1379	37.3725	0.9936	2.15×10^{117}
12	792	66	12	1	871	37.3725	0.7897	7.51×10^{109}
11	462	55	11	1	529	37.3725	0.6154	6.06×10^{101}
10	252	45	10	1	308	37.3725	0.4696	1.01×10^{93}
9	126	36	9	1	172	37.3725	0.3509	3.53×10^{83}
8	56	28	8	1	93	37.3725	0.2580	3.69×10^{73}
7	21	21	7	1	50	37.3725	0.1892	3.12×10^{63}
6	6	15	6	1	28	37.3725	0.1416	1.21×10^{54}
5	1	10	5	1	17	37.3725	0.1103	9.66×10^{45}
4	0	6	4	1	11	37.3725	0.0887	8.31×10^{38}
3	0	3	3	1	7	37.3725	0.0708	3.83×10^{31}
2	0	1	2	1	4	37.3725	0.0535	3.16×10^{22}
1	0	0	1	1	2	37.3725	0.0378	1.78×10^{11}
Average possible moves (B)					267			
Average game length (D)						37.3725		
Strategic complexity ($\sqrt{B/D}$)					0.4372			
Tree size (B^D)					4.8394×10^{90}			

3. Proposed Method

This section specifies the proposed method to develop an agent for playing Big Two. We used six primary rules with 29 sub-rules for the AI agent to play against human players. The six primary rules are rules for two, three, and four cards left, rules (a winning strategy) for the number of moves to win in three moves, rules for more than four cards left, and rules for the agent if they are not in the control position. We have played with the proposed AI in thousands of games to make sure that there are no conflicting rules. We managed all rules and defined the rules' priority using two features: the number of cards left in the agent's hand and the agent's control position (control or no control). Figure 1 shows a flowchart of the proposed rule-based AI. If one combination (e.g., five-card, pair, single) is left, the AI agent automatically discards it and then immediately wins. If more than one combination remains, proposed rules are applied to output an action which the AI agent will execute during the game. The first process is card classification, which classifies all cards in the agent's hand based on a list of cards that have been played (field history).

The agent can be in one of two positions during the game, in control or not in control. A player who has control has the opportunity to choose a type of card combination to play and discard a low-card combination. Other players can only discard a high-card combination following the type of card played or choose to pass. Based on our observations, we found out that the critical moment in the game is when the agent has control and fewer than five cards left in hand. The agent must be able to finish the game and anticipate the opponent's last card combination. If an opponent has one card left, then the agent discards a pair. If an opponent has two cards left, then the agent discards a single card. The rules for situations in which two, three, and four cards are left are used by the agent to select a card combination to be discarded. When the agent has control with four or more cards, sometimes a win is achievable within three moves. For example, the agent could have 11 cards consisting of a straight, a full house, and a single card. In this condition, the agent uses the winning strategy. Under normal conditions, the agent uses the rules for more than four cards left when the AI agent is in a control position. Additionally, we propose rules for the AI agent if they are not in the control position.



Start
Is one combination left?
Yes
No
Card Classification
Control or no control?
Control
No Control
Are two cards left?
Yes
Rules for two cards left
No
Are three cards left?
Yes
Rules for three cards left
No
Are four cards left?
Yes
Rules for four cards left
No
Is the number of moves to win ≤ 3?
Yes
Winning Strategy
No
Rules for more than four cards left
Rules for the agent, if not in control
a selected card combination to discard or "Pass"
a selected card combination to discard
a selected card combination to discard, then win
Stop

Figure 1. Flowchart of the proposed method.

3.1. Card Classification

The first step in classifying a card is to generate all possible card combinations (single, pair, and five-card) of the opponents using Algorithm 1. The opponent's cards include all cards that are not in the agent's hand and have not yet been played. Note that we use pair and five-card functions to generate all pair and five-card combinations.

We classify card combinations that are in the agent's hand into four classes. A card combination which is of a higher rank than all of the opponent's card combinations will be classified in class A. Discarding a card combination of class A, the agent takes control while playing the game. By contrast, a card combination which is of a lower rank than all the opponent's card combinations will be classified in class D. The combinations of class D are impossible to play without control. The card combinations of class B have a high chance of being discarded and could serve as controls. Based on our observations, we classify the card combinations in class B if the rank of the combination is at least in the top 30% (approximately Queen) of the highest-ranking among the opponent's card combinations. Then, class C contains card combinations that are between class B and class D.

Algorithm 1. Generate all opponent's card combinations

```
Input: a list of cards in the agent's hand $C_a$, a list of cards that have played (field history) $C_p$.
1: opponent card $C_o \leftarrow$ all cards $\notin (C_a \cup C_p)$
2: single combinations $CO_s \leftarrow \{x \mid$ for all x in $C_o\}$
3: pair combinations $CO_p \leftarrow$ getPair ($C_o$)
4: five-card combinations $CO_{fc} \leftarrow$ getFivecard ($C_o$)
5: $CO = CO_s \cup CO_p \cup CO_{fc}$
Output: all opponent's card combinations $CO$.

function getPair ($C_o$)
1: pair = {}                    //a set of pairs
2: for i = 0 to (length ($C_o$) − 1):
3:     for j = i + 1 to (length ($C_o$)):
4:         if number ($C_o$[i]) == number ($C_o$[j]):        //2 cards have the same number
5:             pair = pair ∪ {([$C_o$[i], $C_o$[j]])}
6: return pair

function getFivecard ($C_o$)
1: five-card = {}          //a set of five-cards
2: straight ← getStraight ($C_o$)                //a function to get five consecutive cards from $C_o$
3: flush ← getFlush ($C_o$)                      //a function to get five cards of the same suit from $C_o$
4: full-house ← getFullHouse ($C_o$)             //a function to get three cards of one rank + a pair from $C_o$
5: four-of-a-kind ← getFourOfAKind ($C_o$)       //a function to get four cards of one rank + a single card from $C_o$
6: straight-flush ← getStraightFlush ($C_o$)     //a function to get five consecutive cards of the same suit from $C_o$
7: return five-card {straight ∪ flush ∪ full-house ∪ four-of-a-kind ∪ straight-flush}
```

An example of the agent's hand is shown in Figure 2, in which the card suits are diamond (D), club (C), heart (H), and spade (S). The field history is a list of the cards that have been discarded. Note that class D is empty because of the lowest card (3D and 3C) in the opponent's hand. Class A has a pair of cards [2C, 2S] and a single card [2S]. Class B has a pair of cards [QD, QH] and three single cards [KS, AD, 2C]. Other card combinations are in class C. The card classification automatically updates at every turn.

- Hand = [3H, 5D, 6D, 6S, 7H, 8D, 10C, QD, QH, KS, AD, 2C, 2S];
- Field history = empty, because of the beginning of the game.

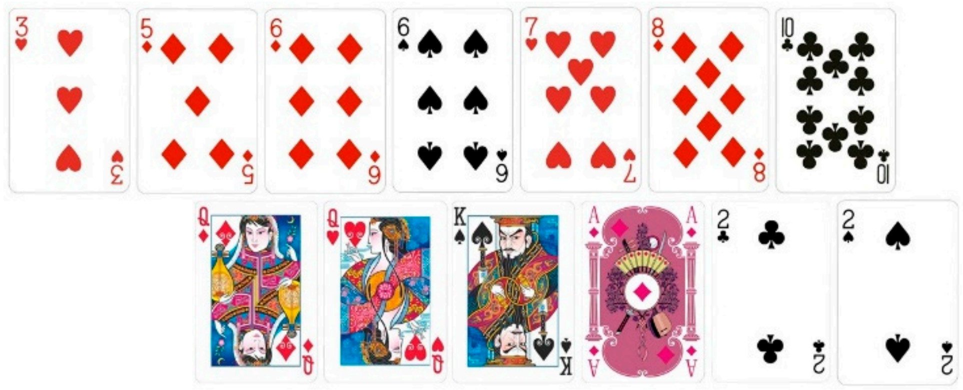


Figure 2. An example of the agent's hand.

The card classification results for this example are as follows:

- Class A = [2C, 2S], [2S];
- Class B = [QD, QH], [KS], [AD], [2C];
- Class C = [5D, 6D, 8D, QD, AD], [6D, 6S], [3H], [5D], [6D], [6S], [7H], [8D], [10C], [QD], [QH];
- Class D = empty, because of the lowest card in the opponent's hand.

Another example is with the same agent's hand but using a different field history.

- Hand = [3H, 5D, 6D, 6S, 7H, 8D, 10C, QD, QH, KS, AD, 2C, 2S];
- Field history = [3D, 3C, 4H, 6H, 7D, 7S, 9C, 9H, 9S, QS, KD, KC, KH, AC, AS, 2D].

The card classification results are as follows:

- Class A = [QD, QH], [2C, 2H], [2S];
- Class B = [QD], [QH], [KS], [AD], [2C];

- Class C = [5D, 6D, 8D, QD, AD], [6D, 6S], [5D], [6D], [6S], [7H], [8D], [10C];
- Class D = [3H].

3.2. Rules for Two Cards Left

When the agent has control and only two cards remain in the hand, the agent will use the rules listed in Algorithm 2 with three parameters as inputs. The first parameter is a list of cards that are currently in the agent's hand, C_a, the second is the classification results for the class that is only class A, and the third is the length (how many cards left) of each opponent's hand.

Algorithm 2. Rules for two cards left

```
Input: a list of cards in the agent's hand $C_a$, the card classification results (class-A), and the length of each opponent's hand (opponent 1, opponent 2, opponent 3).
1: pair ← getPair ($C_a$)
2: if pair exists:                          //two cards left is a pair
3:      return pair
4: else if class-A exists or length (opponent's card) == 1:
5:      return $C_a$[1]                     //the highest-ranking single card (index 1)
6: else:
7:      return $C_a$[0]                     //the lowest-ranking single card (index 0)
Output: a selected card combination that the agent will discard.
```

The getPair function generates any possible combination of pairs from cards in the agent's hand. Lines 2–7 select a card combination to discard based on the following conditions: (1) If a pair exists, the agent will discard a pair and win the game; (2) If any single card of class A is in the agent's hand, or if an opponent only has one card left, then the selected card is the highest-ranking single card (card index 1). Making this move ensures that the agent will win in the next turn because the agent will gain control and can then discard the last card; (3) If there is no single card in class A, the selected card is the lowest single card (card index 0).

3.3. Rules for Three Cards Left

The rules for three cards left (Algorithm 3) require all card classification results (class A, class B, class C, and class D) as input parameters. When the agent has a pair, the best combination to discard may not be a pair because the agent still has a single left in hand. Lines 4–12 are used to select the combination to discard based on the following conditions: (1) If a pair is in class A, then the selected combination is a pair. The agent still has control and will discard the last single card to win; (2) If a single card is in class A, then the agent discards the single card and will then discard a pair to win the game; (3) If an opponent has one card left, then the selected combination is a pair. This selection is a strategy to minimize losses; (4) However, if an opponent has two cards left, then the agent discards a single card; (5) If none of these conditions are fulfilled, then the agent discards a combination based on the lower class of card classification results.

If the agent does not have a pair, then the three cards left are singles. Lines 14–16 are used to select the single to discard based on the following conditions: (1) If the highest single card (card index 2) is in class A, the agent discards the middle single (card index 1) and will then discard the highest single on the next turn to gain control. The agent can then discard the lowest single to win the game; (2) If an opponent has one card left, then the agent discards the highest single to minimize losses; (3) If both of these conditions are not fulfilled, then the lowest single (card index 0) should be selected.

3.4. Rules for Four Cards Left

The rules for four cards left (Algorithm 4) are an expansion of the previous rules (Algorithm 3) and have the same input parameters. The agent may have two pairs, one pair and two single cards, or four single cards. If the agent has two pairs, Lines 2–7 are used to select a pair to discard based on the following conditions: (1) If the higher pair (pair index 1) is in class A, then the agent discards the higher pair and has control. The agent will then discard the last pair and win the game; (2) If an opponent has two cards left, the agent discards the higher pair to minimize losses; (3) If both of these conditions are not fulfilled, then the lower pair (pair index 0) is discarded.

Algorithm 3. Rules for three cards left

```
Input: a list of cards in the agent's hand $C_a$, the card classification results (class-A, class-B, class-C, class-D), and the length of each opponent's hand (opponent 1, opponent 2, opponent 3).
1: pair ← getPair($C_a$)
2:     if pair exists:                                   //the agent has a pair
3:     single ← $C_a \notin$ pair
4:     if pair in class-A: return pair[0]
5:     else if single in class A: return single[0]
6:     else if length(opponent's card) == 1: return pair[0]
7:     else if length (opponent's card) == 2: return single[0]
8:     else:
9:          if class-D exists: return class-D[0]
10:           if class-C exists: return class-C[0]
11:           if class-B exists: return class-B[0]
12:           if class-A exists: return class-A[0]
13: else:
14:      if $C_a$[2] in class-A: return $C_a$[1]
15:      else if an opponent has one card left: return $C_a$[2]
16:      else: return $C_a$[0]
Output: a selected card combination that the agent will discard.
```

Algorithm 4. Rules for four cards left

```
Input: a list of cards in the agent's hand $C_a$, the card classification results (class-A, class-B, class-C, class-D), and the length of each opponent's hand (opponent 1, opponent 2, opponent 3).
1: pair ← getPair ($C_a$)
2: if pair exists:
3:     if length (pair) == 2:                  //the agent has two pairs
4:          if pair [1] in class-A: return pair[1]
5:          else if length (opponent's card) == 2: return pair[1]
6:          else: return pair[0]
7:     else if length (pair) == 1:             //the agent has one pair
8:          single ← $C_a \notin$ pair
9:          if class-A exists:
10:              if single[1] in class-A: return single[0]
11:              else if length (opponent's card) == 1: return pair
12:              else if length (opponent's card) == 2: return single[0]
13:              else: return single[0]
14:           else:
15:              if class-D exists: return class-D[0]
16:              if class-C exists: return class-C[0]
17:              if class-B exists: return class-B[0]
18:              if class-A exists: return class-A[0]
19: else:
20:     if class-A exists: return $C_a$[1]
21:     else if length (opponent's card) == 1: return $C_a$[3]
22:     else: return $C_a$[0]
Output: a selected card combination that the agent will discard.
```

When the agent has one pair, checking the two single cards is essential. Lines 7–18 are used to select a combination to discard based on the following conditions: (1) If the higher single is in class A, then the agent discards the lower single (single index 0). The agent will have three cards left and can then play using Algorithm 3. The agent will have control after discarding the highest single card and can then discard the last pair to win; (2) If an opponent has one card left, then the agent discards a pair; (3) If an opponent has two cards left, then the agent discards a single; (4) If a pair is in class A, then the lower single should be selected. Discarding the highest pair is useless because the agent still has two singles; (5) If none of these conditions are fulfilled, then the agent discards a combination based on the lower class of card classification results.

If the agent does not have a pair, then the four remaining cards are singles. Lines 20–22 provide guidelines to select a single card based on the following conditions: (1) If any single card is in class A, then the agent discards the card index 1; (2) If an opponent has one card left, then the agent discards the highest single (card index 3); (3) If both of these conditions are not fulfilled, the lowest single (card index 0) is selected.

3.5. Rules for More Than Four Cards Left

When the agent has control and has more than four cards in hand, the agent uses the rules for more than four cards left. These rules determine all possible moves of the agent (single, pair, and five-card). These rules (Algorithm 5) prioritize discarding the card combination in the lower class. The agent must discard a card combination of class D and class C as soon as possible; otherwise, these card combinations are difficult to play without control.

Algorithm 5. Rules for more than four cards left

Input: all possible moves of the agent (single, pair, and five-card), the card classification results (class-A, class-B, class-C, class-D), and the length of each opponent's hand (opponent 1, opponent 2, opponent 3).

```
1: if length (opponent's card) == 1:
2:     if five-card exists: return five-card[0]
3:     else if pair exists: return pair[0]
4:     else:
5:         if class-A exists: return class-A[0]
6:         if class-B exists: return class-B[0]
7:         if class-C exists: return class-C[0]
8:         if class-D exists: return class-D[0]
9: else:
10:    if (not five-card) $\wedge$ (length (pair) > length (single)):
11:        return pair[0]
12:    else:
13:        if class-D exists: return class-D[0]
14:        if class-C exists: return class-C[0]
15:        if class-B exists: return class-B[0]
16:        if class-A exists: return class-A[0]
```

Output: a selected card combination that the agent will discard.

There are exceptions to this idea: (1) If an opponent has one card left, then the agent discards a five-card or a pair. The priority order for discarding is as follows: five-card, pair, and then single; (2) If the agent does not have a five-card but has many pairs instead, then the agent discards a pair. If these two exceptions are not fulfilled, then the agent discards a combination based on the lower class of card classification results.

3.6. Winning Strategy

When the agent has control with more than four cards, we generate all possible moves and count how many moves will be needed to win. For example, the agent has 11 cards consisting of a straight, a full house, and a single card. The winning move (WM) for this example hand would be three moves. The winning strategy (Algorithm 6) only applies when the agent has more than four cards and fewer than four moves to end the game.

Algorithm 6. Winning strategy

Input: winning move calculation WM, the card classification results (class-A), all possible moves of the agent (single, pair, and five-card), and the length of each opponent's hand (opponent 1, opponent 2, opponent 3).

```
1: if $WM \leq 2$:
2:     if class-A exists: return class-A[0]
3:     else:
4:         if five-card exists: return five-card[0]
5:         if pair exists:
6:             if length (opponent's card) == 2:
7:                 return pair[$-1$]          //the highest pair (first index from right)
8:             else: return pair[0]           //the lowest pair (first index from left)
9:         if single exists: return single[0]
10: if $WM == 3$:
11:    if length (class-A) > 1: return class-A[0]
12:    else:
13:        if five-card exists: return five-card[0]
14:        if pair exists:
15:            if length (opponent's card) == 2:
16:                return pair[$-1$]          //the highest pair (first index from right)
17:            else: return pair[0]           //the lowest pair (first index from left)
18:        if single exists: return single[0]
```

Output: a selected card combination that the agent will discard.

If the winning move is less than or equal to two, it is essential to play a card combination of class A first and win immediately. If class A does not exist, the order for discarding the cards is a five-card combination, a pair of cards, and then a single card. This strategy can not only minimize the risk, but also maximize the winning score. We apply the same strategy for winning moves that are equal to three. This strategy will guarantee that the agent wins if the number of card combinations in class A is two. The agent discards a card combination of class A twice and plays the last card to win.

3.7. Rules for the Agent, If Not in Control

If they are not in control, the agent can only discard a valid combination according to the type of card combination which is currently being played by the opponent. The agent cannot independently discard a card combination. For example, if the type of card combination played currently is a pair, the agent can only discard a higher pair. We will generate all valid combinations and always update them at every turn. The basic idea of these rules is to discard a card combination of lower-class classification. The critical action when the agent does not have control is to decide when to hold back a selected combination and when to split a card combination.

We designed the rules for the AI agent if not in control (Algorithm 7) by including two essential functions: hold-back function (Algorithm 8) and split-card function (Algorithm 9). The agent uses the hold-back function to check if the selected combination is essential for holding. If the agent decides to hold back, then it passes its turn. The split-card function is a function for splitting a five-card and a pair to minimize loss when an opponent only has one card remaining to win.

Algorithm 7. Rules for the agent, if not in control

```
Input: a list of cards in the agent's hand $C_a$, type of combination played now $CO_f$, the card classification results (class-A $CO_A$, class-B $CO_B$, class-C $CO_C$, class-D $CO_D$), all possible moves of the agent (single, pair, and five-card), all possible moves of the agent according to the type of combination played now $CO_{pm}$, the number of turns $t$, and the length of each opponent's hand $O_1$, $O_2$, $O_3$.
1: class-D valid combinations $CO_{DV} \leftarrow \{x \mid$ for all x in $CO_D$ if $x \in CO_{pm}\}$
2: class-C valid combinations $CO_{CV} \leftarrow \{x \mid$ for all x in $CO_C$ if $x \in CO_{pm}\}$
3: class-B valid combinations $CO_{BV} \leftarrow \{x \mid$ for all x in $CO_B$ if $x \in CO_{pm}\}$
4: class-A valid combinations $CO_{AV} \leftarrow \{x \mid$ for all x in $CO_A$ if $x \in CO_{pm}\}$
5: if $CO_{DV}$ is not empty:
6:     for all $CO_{sc}$ in $CO_{DV}$:
7:         if not hold-back ($C_a$, $CO_f$, $CO_A$, $CO_B$, $CO_C$, $CO_D$, $t$, $O_1$, $O_2$, $O_3$, $CO_{sc}$): return $CO_{sc}$
8: if $CO_{CV}$ is not empty:
9:     for all $CO_{sc}$ in $CO_{CV}$:
10:        if not hold-back ($C_a$, $CO_f$, $CO_A$, $CO_B$, $CO_C$, $CO_D$, $t$, $O_1$, $O_2$, $O_3$, $CO_{sc}$): return $CO_{sc}$
11: if $CO_{BV}$ is not empty:
12:     for all $CO_{sc}$ in $CO_{BV}$:
13:         if not hold-back ($C_a$, $CO_f$, $CO_A$, $CO_B$, $CO_C$, $CO_D$, $t$, $O_1$, $O_2$, $O_3$, $CO_{sc}$): return $CO_{sc}$
14: if $CO_{AV}$ is not empty:
15:     for all $CO_{sc}$ in $CO_{AV}$:
16:         if not hold-back ($C_a$, $CO_f$, $CO_A$, $CO_B$, $CO_C$, $CO_D$, $t$, $O_1$, $O_2$, $O_3$, $CO_{sc}$): return $CO_{sc}$
17: if length (opponent's card) == 1:
18:     return split-card ($C_a$, $CO_{pm}$, single, pair, five-card, $O_1$, $O_2$, $O_3$)
19: else: return ["Pass"]
Output: a selected card combination ($CO_{sc}$) that the agent will discard or Pass.
```

The hold-back function (Algorithm 8) consists of three possible combinations based on the type of combination just played: a single card (lines 1–6), a pair (lines 8–12), and a five-card combination (lines 13–19). When the selected combination is a single card, lines 1–6 determine whether to hold based on the following conditions: (1) if the agent's hand contains two cards at most, then the agent discards the selected card; (2) if an opponent has fewer than three cards left, then the agent discards the selected card; (3) if class-A combination has fewer card combinations than all other classes combined or if all opponents have at least six cards, the highest single card (card index -1) will be held to retain value in the agent's hand. In most cases, discarding the highest single during the early game, especially a non-class-A card, has no purpose, because opponents can easily discard a higher single.

When the selected combination is a pair, lines 7–11 determine if a pair should be held based on the following conditions: (1) if the agent's hand has three cards left at most, then the agent will discard the selected pair; (2) if the selected pair is a 2-pair, then the agent will hold as long as all opponents have more than two cards left. The agent can split a 2-pair to beat two singles and take control twice. Typically, a 2-pair is classified in class A, but some conditions can cause a 2-pair classified in class B. If 2-spade and one of 2-diamond/2-club/2-heart is in the opponent's hand, then a 2-pair will classify in class B. In this case, we can identify a 2-pair more accurately using a score. We use a card range of 1 to 52 (3-diamond to 2-spade), and only take a score from the highest card in a pair. For example, a 3-pair [3D, 3S] has a score of 4. Therefore, we identify that a 2-pair has a score of at least 50.

Algorithm 8. Hold-back Function

Input: a list of cards in the agent's hand C_a, type of combination played now CO_f, the card classification results (class-A CO_A, class-B CO_B, class-C CO_C, class-D CO_D), all possible moves of the agent (five-card), the number of turns t, the length of each opponent's hand O_1, O_2, O_3, and a selected card combination CO_{sc}.

```
1: if length ($CO_f$) == 1:                                  //if a single is now played
2:     if length ($C_a$) $\leq$ 2: return False
3:     if length (opponent's card) < 3: return False
4:     if length ($CO_A$) < (length ($CO_B$) + length ($CO_C$) + length ($CO_D$)) or min (length ($C_a$), $O_1$, $O_2$, $O_3$) > 6:
5:         if $CO_{sc}$ == $C_a[-1]$: return True
6:         else: return False
7: else if length ($CO_f$) == 2:                             //if a pair is now played
8:     if length ($C_a$) $\leq$ 3: return False
9:     if all opponents have more than two cards left:
10:            if pair's score ($CO_{sc}$) $\geq$ 50: return True     //if a 2-pair
11:        else: return False
12: else if length ($CO_f$) == 5:                            //if a five-card is now played
13:        if length ($C_a$) == 5: return False
14:        if min (length ($C_a$), $O_1$, $O_2$, $O_3$) > 6 and $t \leq 4$:
15:            if an opponent "Pass":
16:                if length (five-card) == 2 and ($CO_{sc}$ in $CO_A$ or $CO_{sc}$ in $CO_B$): return True
18:        else: return False
```

Output: True = hold the selected card combination (CO_{sc}); False = discard the selected card combination (CO_{sc})

The crucial moment is when the agent has two five-card combinations (e.g., a straight and a full house), which require the agent to be aware when there are opponent passes at the beginning of a game. For instance, if the first opponent discards a low full house and the second and third opponents pass, the agent must determine whether to discard a full house is the right choice or not. The second or third opponents may have higher rank straights or flushes, which will cause the agent to lose control after discarding a straight and has no full house anymore. Such a situation usually occurs during the beginning period of playing the game (before turn 4).

The function for splitting a five-card combination and a pair (Algorithm 9) is a closing strategy for minimizing losses. This function will be invoked when an opponent has one card remaining to win. If the type of combination played now is a pair, a pair can only be obtained by splitting a four-of-a-kind or a full house. In such a situation, the agent will discard a higher pair than a currently played pair. If a pair in hand does not meet that requirement, the agent will pass.

Algorithm 9. Split-card Function

Input: type of card combination played now CO_f, all possible moves of the agent according to the type of combination played now CO_{pm}, all possible moves of the agent (pair, and five-card).

```
1: if length ($CO_f$) == 2:                      //if a pair is now played
2:     if five-card exists:
3:         if five-card is four-of-a-kind or five-card is full-house:
4:             pair $\leftarrow$ getPair (five-card)
5:             if pair $\in CO_{pm}$: return pair[0]
6:     else: return ["Pass"]
7: if length ($CO_f$) == 1:                      //if a single is now played
8:     if pair exists:
9:         single $\leftarrow$ getSingle (pair)
10:        if single $\in CO_{pm}$: return single[0]
11:    else if five-card exists:
12:        single $\leftarrow$ getSingle (five-card)
13:        if single $\in CO_{pm}$: return single[0]
14: else: return ["Pass"]
```

Output: a selected card combination that the agent will discard or Pass.

If a single card is played, the agent can split a pair or any five-card combination to take control. The agent will prioritize splitting a pair with the intent to play a five-card combination after taking control. However, if no single cards can be played and a pair cannot be split, the agent will split a five-card combination to take control and play another pair or a single card before losing.

3.8. Additional Functions for Five-Card Combinations

We designed two additional functions for selecting five-card combinations. Bi-five-card function (Algorithm 10) checks whether the agent has two five-card combinations or not. This algorithm will run after a five-card combination is found. If a second five-card combination exists, it is listed by strength (e.g., a flush and a straight) with the first combination. The agent will prioritize these two five-card combinations for discarding. The chance of winning will increase because the agent can discard ten cards quickly and leave a few remaining cards in hand.

Algorithm 10. Bi-five-card Function

```
Input: a list of cards in the agent's hand $C_a$, type of combination played now $CO_f$.
1: five-card combinations $CO_{fc}$ ← getFivecard ($C_a$)
2: for all five-card in $CO_{fc}$:
3:      second five-card ← getFivecard ($C_a \notin$ five-card)
4:      if second five-card exists:
5: add [five-card, second five-card] to $CO_{fc}$
6:           return $CO_{fc}$
Output: double five-card combinations.
```

The best-five-card function (Algorithm 11) simulates and determines the best five-card combination for discard. The best combination should contain several cards that are difficult to discard. The cards remaining in hand also have a maximum value compared to other possible five-card combination discards.

Algorithm 11. Best-five-card Function

```
Input: a list of cards in the agent's hand $C_a$, type of combination played now $CO_f$.
1: selectedFive-card = {}, and first-hand = {}
2: five-card combinations $CO_{fc}$ ← getFivecard ($C_a$)
3: for all five-card in $CO_{fc}$:
4:      cards-left ← $C_a \notin$ five-card
5:      if length (selectedFive-card) == 0:                    //First five-card
6:           pair ← getPair (cards-left)
7:           first-hand = first-hand ∪ pair
8:           single ← cards-left ∉ pair
9:           first-hand = first-hand ∪ single
10:          first-hand.score ← maxValue (first-hand)
11:          first-five-card.score ← score (five-card)
12:          selectedFive-card ← five-card
13:     else if length (selectedFive-card) > 0:                //Second five-card
14:          second-hand = {}
15:          pair ← getPair (cards-left)
16:          second-hand = second-hand ∪ pair
17:          single ← cards-left ∉ pair
18:          second-hand = second-hand ∪ single
19:          second-hand.score ← maxValue (second-hand)
20:          second five-card.score ← score (five-card)
21:          if first-hand.score < second-hand.score:
22:               first-hand ← second-hand
23:               selectedFive-card ← five-card
24:          else if first-hand.score == second-hand.score:
25:               if length (first-hand) > length (second-hand):
26:                    first-hand ← second-hand
27:                    selectedFive-card ← five-card
28:               else if length (first-hand) == length (second-hand):
29:                    if first-five-card.score < second-five-card.score:
30:                         first-hand ← second-hand
31:                         selectedFive-card ← five-card
32: return selectedFive-card
Output: a selected five-card with the best card left.
```

The first-hand contains cards that will remain regardless of a five-card combination. The second-hand contains the cards remaining regardless of another five-card combination. This function calculates the remaining card scores from the first-hand (lines 4–11) and second-hand (lines 12–19). Lines 20–30 select the best five-card combination for discard based on the following conditions: (1) the highest value of cards left; (2) the fewest number of card combinations left; and (3) the highest rank of selected five-card combinations. This function will repeat the process for all five-card combinations until the best possible five-card combination discard is determined.

4. Experimental Results and Discussion

We have performed three experiments to evaluate the performance of our rule-based AI agent. Big Two is a four-player card game; therefore, two rule-based AI agents played against two opponents and play log data were recorded in each of these experiments. The data presented in this study are available online in our repository (see Supplementary Materials). For the first and second experiments, two types of AI opponents were built: randomized AI and conventional AI, which have been implemented in Python as the rule-based agents. Note that we generated all possible moves of an agent (single, pair, and five-card) in each turn. Randomized AI randomly chooses a card combination from the possible moves to play. The first experiment was the baseline for verifying whether rule-based AI showed any intelligence in playing Big Two.

Conventional AI played the game according to the systematic approach [7] and probability calculations [8] which some experts have tried when formalizing how to play Big Two. The main idea of this AI is to play the lowest possible card combination to maximize the chances of winning. We generated all possible card combinations (single, pair, and five-card) of the agent. The value of possible combinations can be determined based on the probability of opponents having a single, pair, and five-card of a certain rank. In each turn, the agent will select the lowest possible card combination. If the agent does not have a valid combination to play, then it will pass its turn. The second experiment is designed to evaluate whether rule-based AI outperforms the existing approaches.

For the third experiment, our rule-based AI connected to our web-based Big Two [6] and played against human players. Figure 3 shows a screenshot of our web-based Big Two game (our web-based Big Two game is freely available to try using Google Chrome at http://web.bigtwo.gamelab.com.tw/, accessed on 28 December 2020). The green 19 in Figure 3 is the timer countdown. If the active player does not discard a valid combination at the specified time (20 s), then it will pass its turn. We selected 50 graduate students and assured that there was no cheating between human players. Every two random participants played against two rule-based AI agents in different game rooms. Before starting the experiment, each player played more than 20 games to become familiar with the game. Then, each player played 80 games against rule-based AI agents via our web-based Big Two.

We stated the following hypotheses:

Hypothesis 1 (H1). *Rule-based AI significantly outperforms randomized AI.*

Hypothesis 2 (H2). *Rule-based AI significantly outperforms conventional AI.*

Hypothesis 3 (H3). *Rule-based AI significantly outperforms human players.*

We used a Wilcoxon signed-rank test [44,45] to measure the effect size (r) and check whether each hypothesis was accepted and statistically significant using p-values. The hypothesis was accepted and statistically significant if the p-value < 0.01; otherwise, the hypothesis was rejected. Furthermore, we used Pearson's r to interpret the effect size [46] as small ($0.10 < r < 0.30$), medium ($0.30 < r < 0.50$), and large ($r > 0.50$).

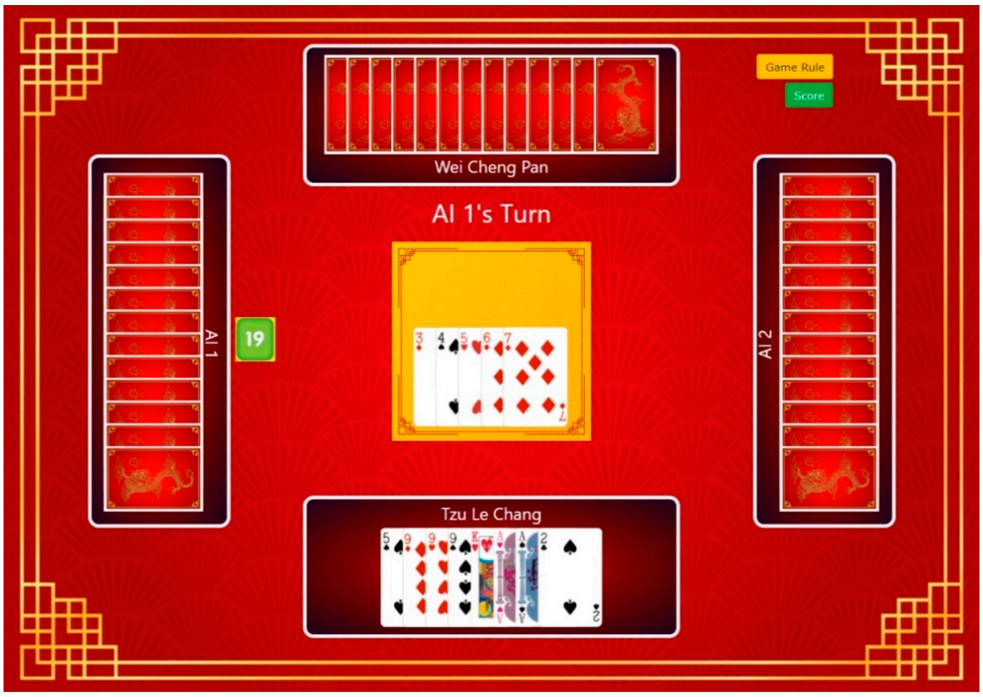


Figure 3. A screenshot of our web-based Big Two game.

Figure 4 shows a histogram of rule-based AI's win–loss records achieved after playing 2000 games in each experiment against various opponents. The number above the blue bar indicates the number of wins, and the number above the orange bar indicates the number of losses. In the first experiment against randomized AI, the rule-based AI won 1792 games and lost 208 games. In the second experiment against conventional AI, the rule-based AI won 1460 games and lost 540 games. In the third experiment against human players, the rule-based AI won 1101 games and lost 899 games. Overall, rule-based AI played 6000 games, winning 4353 games and losing 1647 games. Rule-based AI demonstrated better performance than all of its different opponents.

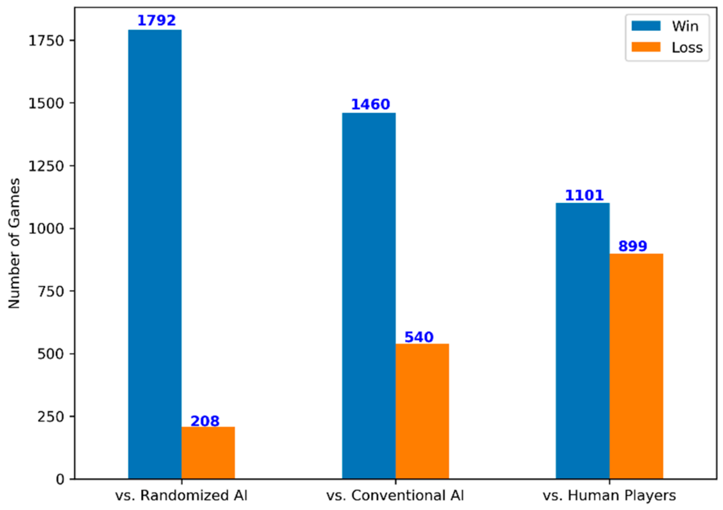
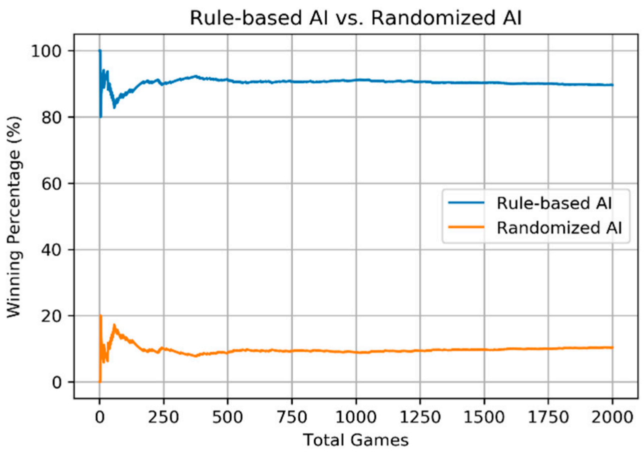
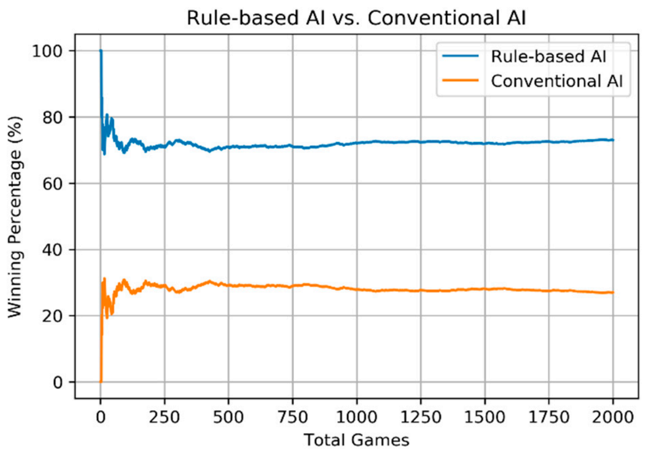


Figure 4. Histogram of the rule-based AI's wins and losses after playing 2000 games versus various opponents.

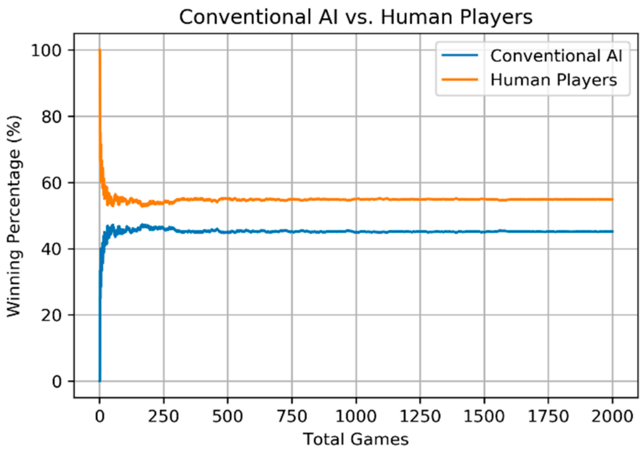
Figure 5 shows that rule-based AI's performance was stable during the experiments, especially after 250 games. The x-axis denotes the total number of games played, and the y-axis denotes the winning percentages of the AI and the opponent. The winning percentage is defined as wins divided by the total number of games played. In the first experiment, the winning percentage of rule-based AI was 89.60%, and that of the randomized AI was 10.40%. In the second experiment, the winning percentage of rule-based AI was 73.00%, compared to 27.00% for the conventional AI. Figure 5c shows the performance curves for conventional AI and human players. It was observed that the winning percentage of conventional AI was 45.15%, and that of the human players was 54.85%. In the third experiment, the winning percentage of rule-based AI was 55.05%, and that of the human players was 44.95%. Based on the number of games won and the winning percentages, we can conclude that the rule-based AI agents played well because they outperformed all opponents in each experiment.



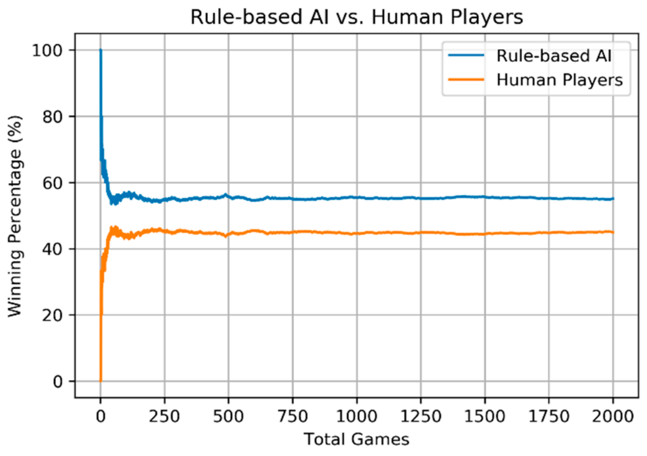
(**a**)



(**b**)



(**c**)



(**d**)

Figure 5. The performance curves for (**a**) rule-based AI and randomized AI, (**b**) rule-based AI and conventional AI, (**c**) conventional AI and human players, and (**d**) rule-based AI and human players.

4.1. Experiment 1: Rule-Based AI versus Randomized AI

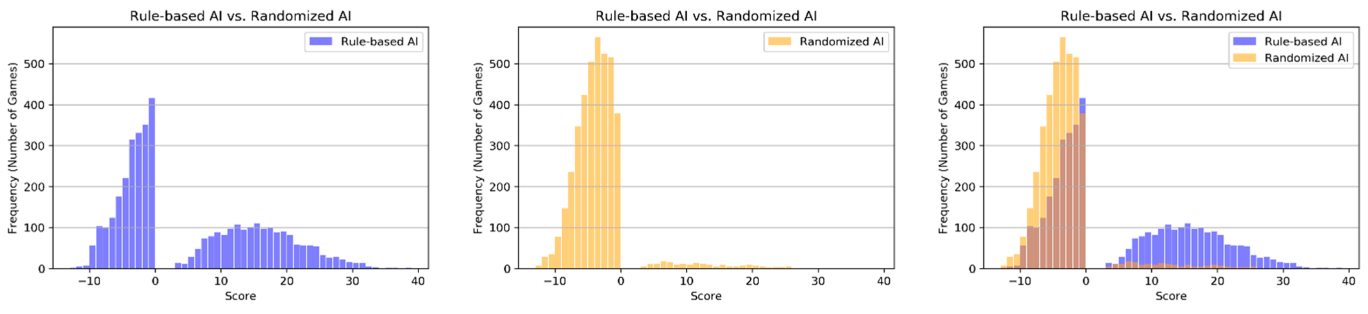
In the first experiment, two rule-based AI agents played 2000 games against two randomized AI agents. In every game, two rule-based AI agents played against two randomized AI agents; therefore, we gathered 4000 play logs from each set—4000 from the two rule-based AI agents and 4000 from the two randomized AI agents. Table 2 shows the statistics of play log data. The rule-based AI won 1792 games (89.60%) and lost 208 games (10.40%). We analyzed the frequency (f) of scores in the positive zone (when the AI won and achieved a positive score) and the frequency (f) of scores in the negative zone (when the AI lost and achieved a negative score). Rule-based AI finished with a positive score 1792 times and a negative score 2208 times. In contrast, randomized AI finished with a positive score 208 times and a negative score 3792 times.

Rule-based AI's mean score when winning was 15.64 (*SD* = 6.43) and when losing it was −4.01 (*SD* = 2.57). Randomized AI's mean score when winning was 12.32 (*SD* = 6.53), and when losing it was −4.64 (*SD* = 2.53). When testing the first hypothesis at a confidence level of 95%, we obtained a *p*-value of 1.0669×10^{-283}. The statistics confirm that the results are statistically significant (*p*-value < 0.01). Thus, rule-based AI significantly outperforms randomized AI. The effect size (*r*) was 0.5921, which is a large effect (*r* > 0.50), according to Pearson's classification of effect sizes.

Table 2. Statistics of play log data from 2000 games of rule-based AI versus randomized AI.

Statistic	**Rule-Based AI**	**Randomized AI**
Win rate	89.60% (1792 games)	10.40% (208 games)
Scores in the positive zone		
f	1792	208
Mean score	15.64	12.32
SD score	6.43	6.53
Scores in the negative zone		
f	2208	3792
Mean score	–4.01	–4.64
SD score	2.57	2.53
Overall statistics concerning all scores		
Total	4000 scores	4000 scores
p-value	1.0669×10^{-283}	
Effect size (*r*)	0.5921	

Figure 6 shows the frequency distribution of rule-based AI agents and randomized AI agents from 4000 scores. The *x*-axis denotes the score achieved in each game, and the *y*-axis denotes the frequency (number of games) of the players that achieved that score. In Big Two, each player receives 13 cards to play. Although each player can only lose 13 cards, they have a chance to win up to 39 cards from the three opponents. The winner's game score is determined by the total number of cards left by the three opponents. For example, if player 1 wins and the three opponents have 2, 6, and 9 cards left, then the scores will be 17; −2; −6; −9. Note that at the end of each game, the winner achieves a positive score and three losers achieve negative scores.



(**a**) (**b**) (**c**)

Figure 6. The frequency distribution of scores for (**a**) rule-based AI, (**b**) randomized AI, (**c**) rule-based AI and randomized AI. The brown bars represent the intersection of purple bars and orange bars.

In Figure 6a, rule-based AI agents finished with a positive score in 1792 games and a negative score in 2208 games. In Figure 6b, randomized AI agents finished with a positive score in 208 games and a negative score in 3792 games. We compared the frequency distribution of all game scores between rule-based AI and randomized AI, as shown in Figure 6c. In the positive zone, the rule-based AI scored positively more often than the randomized AI. The percentage of rule-based AI was 89.60%, and the percentage of randomized AI was 10.40%. The rule-based AI agents ended the game with a positive score (won) 79.20% more often than the randomized AI agents. The rule-based AI agents tended to end a game with a positive score of 8 to 21, higher than the randomized AI agents with a positive score of 5 to 18. The rule-based AI agents outperformed the randomized AI agents, ending the game early and leaving the randomized AI agents with many cards. In the negative zone, the randomized AI scored negatively more often than the rule-based AI. The percentage of rule-based AI was 36.80%, and the percentage of randomized AI was 63.20%. The randomized AI agents lost with many cards left 26.40% more often than the rule-based AI agents. The rule-based AI agents tended to end a game with a negative score of −5 to −1, better than the randomized AI agents, which had negative scores of −7 to −2. These results indicate that the strategies used to minimize losses in rule-based AI are effective.

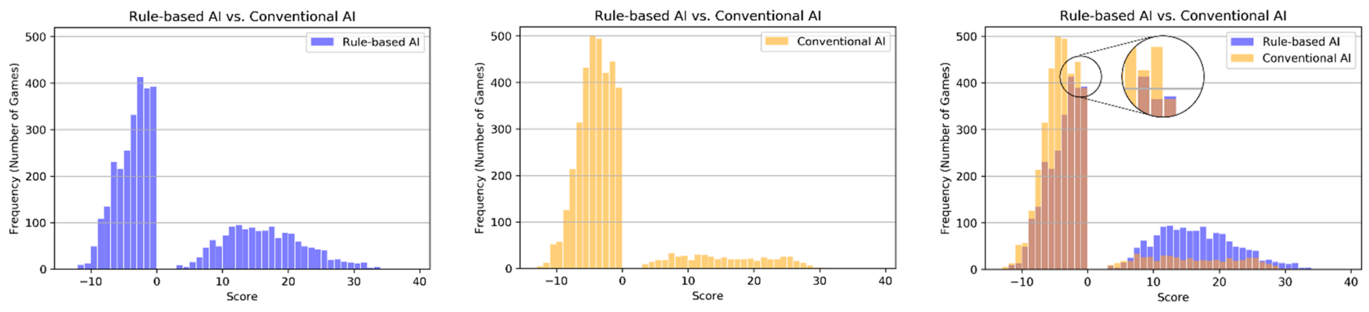
4.2. Experiment 2: Rule-Based AI versus Conventional AI

In the second experiment, two rule-based AI agents played 2000 games against two conventional AI agents. We have gathered 4000 play logs from each set—4000 from the two rule-based AI agents and 4000 from the two conventional AI agents. Table 3 shows the statistics of play log data. The rule-based AI won 1460 games (73.00%) and lost 540 games (27.00%). Rule-based AI finished with a positive score 1460 times and a negative score 2540 times. Rule-based AI's mean score when winning was 16.08 (*SD* = 6.16) and when losing it was −4.22 (*SD* = 2.55). In contrast, conventional AI finished with a positive score 540 times and a negative score 3460 times. Conventional AI's mean score when winning was 15.16 (*SD* = 6.98), and when losing it was −4.63 (*SD* = 2.51). When testing the second hypothesis at a confidence level of 95%, we obtained a p-value of 2.5233×10^{-100}. The statistics confirm that the results are statistically significant (p-value < 0.01). Thus, rule-based AI significantly outperforms conventional AI. The effect size (r) was 0.3545, which is a medium effect ($0.30 < r < 0.50$).

Table 3. Statistics of play log data from 2000 games of rule-based AI versus conventional AI.

Statistic	**Rule-Based AI**	**Conventional AI**
Win rate	73.00% (1460 games)	27.00% (540 games)
	Scores in the positive zone	
f	1460	540
Mean score	16.08	15.16
SD score	6.16	6.98
	Scores in the negative zone	
f	2540	3460
Mean score	−4.22	–4.63
SD score	2.55	2.51
	Overall statistics concerning all scores	
Total	4000 scores	4000 scores
p-value	2.5233×10^{-100}	
Effect size (r)	0.3545	

Figure 7 shows the frequency distribution of rule-based AI agents and conventional AI agents from 4000 scores. In Figure 7a, rule-based AI agents finished with a positive score in 1460 games and a negative score in 2540 games. In Figure 7b, conventional AI agents finished with a positive score in 540 games and a negative score in 3460 games. We compared the frequency distribution of all game scores between rule-based AI and conventional AI, as shown in Figure 7c. In the positive zone, the percentage of rule-based AI was 73.00%, and the percentage of conventional AI was 27.00%. The rule-based AI agents ended the game with a positive score (won) 46% more often than the conventional AI agents. The rule-based AI agents tended to end a game with a positive score of 10 to 23, higher than the conventional AI agents, which had positive scores of 8 to 21. The strategies to maximize the winning score worked because rule-based AI agents often achieved a high winning score. In the negative zone, the percentage of rule-based AI was 42.33%, and the percentage of conventional AI was 57.66%. The conventional AI agents lost with many cards left 15.33% more often than the rule-based AI agents. The rule-based AI agents tended to end a game with a negative score of −7 to −1, better than the conventional AI agents, which had negative scores of −8 to −1. The rule-based AI agents could minimize the number of cards left when the chance of winning was low. We can conclude that rule-based AI significantly outperforms conventional AI.



(**a**) (**b**) (**c**)

Figure 7. The frequency distribution of scores for (**a**) rule-based AI, (**b**) conventional AI, (**c**) rule-based AI and conventional AI. The brown bars represent the intersection of purple bars and orange bars.

4.3. Experiment 3: Rule-Based AI versus Human Players

In the third experiment, two rule-based AI agents played 2000 games against two human players. We have gathered 4000 play logs from each set—4000 from the two rule-based AI agents and 4000 from the human players. Table 4 shows the statistics of play log data.

Table 4. Statistics of play log data from 2000 games of rule-based AI versus human players.

Statistic	Rule-Based AI	Human Players
Win rate	55.05% (1101 games)	44.95% (899 games)
	Scores in the positive zone	
f	1101	899
Mean score	18.25	16.10
SD score	7.24	7.71
	Scores in the negative zone	
f	2899	3101
Mean score	−4.72	−5.31
SD score	3.16	3.33
	Overall statistics concerning all scores	
Total	4000 scores	4000 scores
p-value	2.5325×10^{-14}	
Effect size (*r*)	0.1442	

The rule-based AI won 1101 games (55.05%) and lost 899 games (44.95%). Rule-based AI finished with a positive score 1101 times and a negative score 2899 times. Human players finished with a positive score 899 times and a negative score 3101 times. Rule-based AI's mean score when winning was 18.25 (*SD* = 7.24) and when losing it was −4.72 (*SD* = 3.16). Human players' mean score when winning was 16.10 (*SD* = 7.71), and when losing it was −5.31 (*SD* = 3.33). When testing the third hypothesis at a confidence level of 95%, we obtained a p-value of 2.5325×10^{-14}. The statistics confirm that the results are statistically significant (p-value < 0.01). Thus, rule-based AI significantly outperforms human players. The effect size (r) was 0.1442, which is a small effect ($0.10 < r < 0.30$).

Figure 8 shows the frequency distribution of rule-based AI agents and human players from 4000 scores. In Figure 8a, rule-based AI finished with a positive score in 1101 games and with a negative score in 2899 games. In Figure 8b, human players finished with a positive score in 899 games and with a negative score in 3101 games. We compared the frequency distribution of all game scores between rule-based AI and human players, as shown in Figure 8c.

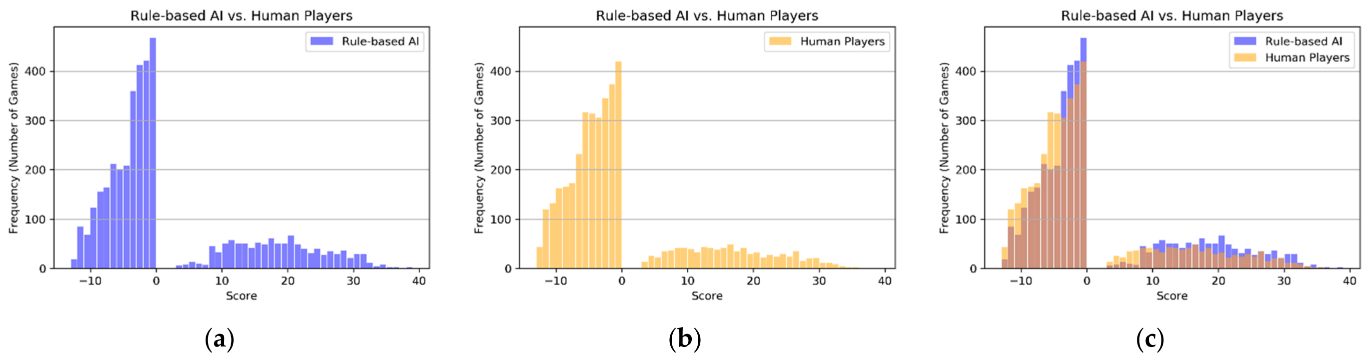


Figure 8. The frequency distribution of scores for (**a**) rule-based AI, (**b**) human players, (**c**) rule-based AI and human players. The brown bars represent the intersection of purple bars and orange bars.

In the positive zone, the percentage of rule-based AI was 55.05%, and the percentage of human players was 44.95%. The rule-based AI agents ended the game with a positive score (won) 10.10% more often than the human players. The rule-based AI agents tended to end a game with a positive score of 11 to 25, higher than the human players, which had positive scores of 8 to 22. The strategies to maximize the winning score worked because rule-based AI agents often achieved a high winning score. The rule-based AI agents outperformed the human players to end a game early and leave the human players with many cards.

In the negative zone, the percentage of rule-based AI was 48.32%, and the percentage of human players was 51.68%. The human players lost with many cards left 3.36% more often than the rule-based AI agents. The rule-based AI agents tended to end a game with a negative score of −7 to −1, better than the human players, which had negative scores of −9 to −1. These results indicate that the strategies used to minimize losses in rule-based AI are effective. The rule-based AI agents could minimize the number of cards left when the chance of winning was low. We can conclude that rule-based AI significantly outperforms human players.

Overall, the experimental results show that our AI agent played well in all the experiments that have been conducted. In each experiment, the proposed AI agent ended the game with more wins than all the opponents. The strategies to maximize the winning score work because rule-based AI agents often achieved a high winning score when they won, ending the game early and leaving every opponent with many cards. When the chance of winning is low, the strategies used to minimize losses in rule-based AI are effective because rule-based AI agents can minimize the number of cards left.

We have measured the average time taken by our rule-based AI to decide its future actions. The server is a CPU with 3.40 GHz Intel Core i7-6700 and 16 GB of random-access memory. Table 5 illustrates the results of this measurement. The evaluation was conducted in three different field conditions: one (single), two (pair), or five (five-card). As noted, the AI agent could only play the respective number of cards from its hand based on the field condition. If no valid combination was possible, then the AI agent skipped a turn. We have taken measurements in three different cases: 13 cards, 8 cards, and 5 cards that were left in the agent's hand for the average time needed by the AI agent to find and choose the correct card needed for the combination. As Table 5 shows, the AI agent took less than one second to decide a selected card combination for each of its turns.

Table 5. Average time taken depending on the number of cards left in the agent's hand.

Number of Cards Left	One-Card Field	Two-Card Field	Five-Card Field
13	0.766 s	0.558 s	0.943 s
8	0.077 s	0.061 s	0.063 s
5	0.052 s	0.054 s	0.058 s

Furthermore, we analyzed the instances where our AI lost to the human player so that we could minimize its weaknesses and improve the algorithm of the proposed AI. We found two factors causing the AI agent to lose. The first is a bad initial set of cards, which highly diminishes the agent's chances of winning the game. This situation is because, in the beginning, cards are distributed at random; therefore, every player has a chance of drawing weak cards. The second factor that can result in a loss is the large number of potential combinations of cards in the opponent's hand. The AI agents could not see the cards in the opposing player's hand; thus, they could only attempt to predict their cards. This behavior can result in a defeat because sometimes the predictions are made incorrectly, causing the AI to play the wrong card and eventually lose.

5. Conclusions

In this paper, we have proposed a rule-based AI method for an agent playing Big Two. The proposed method includes rules for two to four cards left, rules for more than four cards left, a winning strategy, and rules for the agent if they are not in the control position. The experimental results show that all hypotheses were accepted, because the proposed AI significantly outperformed randomized AI, conventional AI, and human players. The proposed AI achieved winning rates of 89.60%, 73.00%, and 55.05%, respectively, better than all opponents with winning rates of 10.40%, 27.00%, and 44.95%. The proposed AI exhibited better performance than conventional AI when playing against human players, with a winning percentage of 55.05% compared to 45.15%. Further research can use this rule-based AI as a baseline player. Furthermore, our AI agent could play Big Two well with the capability to maximize the winning score and minimize the number of cards left when the chance of winning was low. This AI could potentially be used to give hints to beginners or players hesitating about their turn. However, playing against an agent that is too strong is not fun. Players want to play with opponents whose skills are close to theirs. The proposed method can be used as a starting point for the next stage of research. We plan to develop a multiple-level AI agent that will entertain and make playing Big Two enjoyable.

Supplementary Materials: The following are openly available online in our repository at https://github.com/sugiyantoID/RuleBasedAgents, Data S1: Experiment 1–3; Computer Code S1: Rule-based Agents; Video S1: Demonstration Video.mp4.

Author Contributions: Conceptualization, S., G.F. and W.-K.T.; methodology, S., G.F. and W.-K.T.; software, S. and G.F.; validation, S., G.F. and W.-K.T.; resources, W.-K.T.; data curation, S.; visualization, S. and G.F.; writing—original draft preparation, S.; writing—review and editing, S. and W.-K.T.; supervision, W.-K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available online at https://github.com/sugiyantoID/RuleBasedAgents, accessed on 28 April 2021.

Acknowledgments: We would like to acknowledge the support from the GAME Lab at the National Taiwan University of Science and Technology (NTUST), at which this Big Two game was developed. We would also like to thank every person who has played the game.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sun, L.; Jiao, P.; Xu, K.; Yin, Q.; Zha, Y. Modified Adversarial Hierarchical Task Network Planning in Real-Time Strategy Games. *Appl. Sci.* **2017**, *7*, 872. [CrossRef]
2. Duarte, F.F.; Lau, N.; Pereira, A.; Reis, L.P. A Survey of Planning and Learning in Games. *Appl. Sci.* **2020**, *10*, 4529. [CrossRef]
3. van den Herik, H.J. Computer Chess: From Idea to DeepMind. *ICGA J.* **2018**, *40*, 160–176. [CrossRef]
4. Browne, C. What Can Game AI Teach Us? *ICGA J.* **2014**, *37*, 161–165. [CrossRef]
5. Chang, T.-L.; Sugiyanto; Pan, W.-C.; Tai, W.-K.; Chang, C.-C.; Way, D.-L. Opponent Behavior Prediction in a Multi-Player Game with Imperfect Information. In Proceedings of the 2020 IEEE Graphics and Multimedia (GAME), Kota Kinabalu, Malaysia, 17–19 November 2020 ; pp. 43–48.
6. Sugiyanto; Tai, W.K.; Fernando, G. The Development and Evaluation of Web-Based Multiplayer Games with Imperfect Information Using WebSocket. In Proceedings of the IEEE 2019 12th International Conference on Information & Communication Technology and System (ICTS), Surabaya, Indonesia, 18 July 2019; pp. 252–257.
7. Meng, T.K.; Siong, R.K.Y.; Yong, J.A.K.; Chiang, I.L.W. 3 << 2: Dai-Di Analysis. *Pagat* **2000**, *6*, 1–28.
8. Haw, P.; Yongzhi, S. Investigating a Winning Strategy for Big Two. In Proceedings of the Singapore Mathematics Project Festival 2009, Bukit Timah, Singapore, 21 March 2009; pp. 1–27.
9. Kirby, N. *Introduction to Game AI*; Course Technology, a Part of Cengage Learning: Boston, MA, USA, 2011; ISBN 978-1-59863-998-8.
10. Small, R.; Congdon, C.B. Agent Smith: Towards an Evolutionary Rule-Based Agent for Interactive Dynamic Games. In Proceedings of the 2009 IEEE Congress on Evolutionary Computation, CEC 2009, Trondheim, Norway, 18–21 May 2009; pp. 660–666.
11. Bojarski, S.; Congdon, C.B. REALM: A Rule-Based Evolutionary Computation Agent That Learns to Play Mario. In Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010, Copenhagen, Denmark, 18–21 August 2010; pp. 83–90.
12. Ali, R.B.; Ali, M.; Farooqi, A.H. Analysis of Rule Based Look-Ahead Strategy Using Pacman Testbed. In Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011, Shanghai, China, 10–12 June 2011; Volume 3, pp. 480–483.
13. Gan, X.; Bao, Y.; Han, Z. Real-Time Search Method in Nondeterministic Game—Ms. Pac-Man. *ICGA J.* **2011**, *34*, 209–222. [CrossRef]
14. Gagne, D.J.; Congdon, C.B. FRIGHT: A Flexible Rule-Based Intelligent Ghost Team for Ms. Pac-Man. In Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, 11–14 September 2012; pp. 273–280.
15. Rushing, J.; Tiller, J. Rule Learning Approaches for Symmetric Multiplayer Games. In Proceedings of the CGAMES'2011 USA—16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games, Louisville, KY, USA, 27–30 July 2011; pp. 121–125.
16. Srisuphab, A.; Silapachote, P. Rule-Based Systems Made Easy with Battleship Games: A Well-Received Classroom Experience. In Proceedings of the 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bali, Indonesia, 26–29 August 2013; pp. 560–564.
17. Holland, S.; Pitt, J.; Sanderson, D.; Busquets, D. Reasoning and Reflection in the Game of Nomic: Self-Organising Self-Aware Agents with Mutable Rule-Sets. In Proceedings of the IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASOW 2013, Philadelphia, PA, USA, 9–13 September 2014; pp. 101–106.
18. Uiterwijk, J.W.H.M. The Impact of Safe Moves on Part 1: Analysis of and Experiments with 1-Step Safe Moves. *ICGA J.* **2014**, *37*, 97–105. [CrossRef]
19. Teófilo, L.F.; Reis, L.P.; Cardoso, H.L.; Mendes, P. Rule Based Strategies for Large Extensive-Form Games: A Specification Language for No-Limit Texas Hold'em Agents. *Comput. Sci. Inf. Syst.* **2014**, *11*, 1249–1269. [CrossRef]
20. Pham, N.H. A Completed Implementation for Xiangqi Rules. *ICGA J.* **2018**, *40*, 305–317. [CrossRef]
21. Calimeri, F.; Germano, S.; Ianni, G.; Pacenza, F.; Perri, S.; Zangari, J. Integrating Rule-Based AI Tools into Mainstream Game Development. In Proceedings of the International Joint Conference on Rules and Reasoning RuleML+RR 2018, University of Luxembourg, Luxembourg, 18–21 September 2018; Springer: Cham, Switzerland, 2018; pp. 310–317.
22. Rao, D.V.; Kaur, J. A Fuzzy Rule-Based Approach to Design Game Rules in a Mission Planning and Evaluation System. In Proceedings of the IFIP Advances in Information and Communication Technology, Larnaca, Cyprus, 6–7 October 2010; Springer: Berlin/Heidelberg Germany; Volume 339, pp. 53–61.

23. Ballinger, C.A.; Turner, D.A.; Concepcion, A.I. Artificial Intelligence Design in a Multiplayer Online Role Playing Game. In Proceedings of the 2011 8th International Conference on Information Technology: New Generations, ITNG 2011, Las Vegas, NV, USA, 11–13 April 2011; pp. 816–821.
24. Chen, B.N.; Chang, H.J.; Hsu, S.C.; Chen, J.C.; Hsu, T.S. Advanced Meta-Knowledge for Chinese Chess Endgame Knowledge Bases. *ICGA J.* **2014**, *37*, 17–24. [CrossRef]
25. Posthoff, C.; Steinbach, B. Solving the Game of Sudoku. *ICGA J.* **2014**, *37*, 111–116. [CrossRef]
26. Sato, N.; Temsiririrkkul, S.; Sone, S.; Ikeda, K. Adaptive Fighting Game Computer Player by Switching Multiple Rule-Based Controllers. In Proceedings of the 3rd International Conference on Applied Computing and Information Technology and 2nd International Conference on Computational Science and Intelligence, ACIT-CSI 2015, Okayama, Japan, 12–16 July 2015; pp. 52–59.
27. Vorachart, V.; Takagi, H. Evolving Fuzzy Logic Rule-Based Game Player Model for Game Development. *Int. J. Innov. Comput. Inf. Control* **2017**, *13*, 1941–1951. [CrossRef]
28. van den Bergh, M.J.H.; Hommelberg, A.; Kosters, W.A.; Spieksma, F.M. Aspects of the Cooperative Card Game Hanabi. In Proceedings of the Benelux Conference on Artificial Intelligence, Amsterdam, The Netherlands, 10–11 November 2016; Springer: Cham, Switzerland; Volume 765, pp. 32–46.
29. Castro-Wunsch, K.; Maga, W.; Anton, C. BeeMo, a Monte Carlo Simulation Agent for Playing Parameterized Poker Squares. In Proceedings of the Sixth Conference on Artificial Intelligence (AAAI 2016), Phoenix, AZ, USA, 12–17 February 2016; pp. 4071–4074.
30. Nimoto, K.; Takahashi, K.; Inaba, M. Construction of a Player Agent for a Card Game Using an Ensemble Method. In Proceedings of the Procedia Computer Science: 20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems, York, UK, 5–7 September 2016; Volume 96, pp. 772–781.
31. Nimoto, K.; Takahashi, K.; Inaba, M. Improvement of Agent Learning for a Card Game Based on Multi-Channel ART Networks. *J. Comput.* **2016**, *11*, 341–352. [CrossRef]
32. Ward, C.D.; Cowling, P.I. Monte Carlo Search Applied to Card Selection in Magic: The Gathering. In Proceedings of the CIG2009—2009 IEEE Symposium on Computational Intelligence and Games, Milano, Italy, 7–10 September 2009; pp. 9–16.
33. Whitehouse, D.; Cowling, P.I.; Powley, E.J. Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. In Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013, Boston, MA, USA, 14–18 October 2013; pp. 100–106.
34. Robilliard, D.; Fonlupt, C.; Teytaud, F. Monte-Carlo Tree Search for the Game of "7 Wonders". *Commun. Comput. Inf. Sci.* **2014**, *504*, 64–77. [CrossRef]
35. Osawa, H. Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information. In Proceedings of the Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015; Volume WS-15-07, pp. 37–43.
36. Palma, S.D.; Lanzi, P.L. Traditional Wisdom and Monte Carlo Tree Search Face-to-Face in the Card Game Scopone. *IEEE Trans. Games* **2018**, *10*, 317–332. [CrossRef]
37. Jager, W.; van der Vegt, G. Management of Complex Systems: Toward Agent-Based Gaming for Policy. In *Policy Practice and Digital Science*; Janssen, M., Wimmer, M.A., Deljoo, A., Eds.; Public Administration and Information Technology; Springer International Publishing: Cham, Switzerland, 2015; Volume 10, pp. 291–303, ISBN 978-3-319-12783-5.
38. ChePa, N.; Alwi, A.; Din, A.M.; Safwan, M. The Application of Neural Networks and Min-Max Algorithm in Digital Congkak. In Proceedings of the 4th International Conference on Computing and Informatics (ICOCI), Sarawak, Malaysia, 28–30 August 2013; Volume 4, pp. 222–227.
39. Garg, R.; Nayak, D.P. Game of Tic-Tac-Toe: Simulation Using Min-Max Algorithm. *Int. J. Adv. Res. Comput. Sci. (IJARCS)* **2017**, *8*, 1074–1077. [CrossRef]
40. Nasa, R.; Didwania, R.; Maji, S.; Kumar, V. Alpha-Beta Pruning in Mini-Max Algorithm—An Optimized Approach for a Connect-4 Game. *Int. Res. J. Eng. Technol. (IRJET)* **2018**, *5*, 1637–1641.
41. Flesch, J.; Kuipers, J.; Yaakovi, A.M.; Schoenmakers, G.; Solan, E.; Vrieze, K. Perfect Information Games with Lower-Semicontinuous Payoffs. *Math. Oper. Res.* **2010**, *35*, 742–755. [CrossRef]
42. Yoshimura, K.; Hochin, T.; Nomiya, H. Estimation of Rates Arriving at the Winning Hands in Multi-Player Games with Imperfect Information. In Proceedings of the 4th International Conference on Applied Computing and Information Technology/3rd International Conference on Computational Science/Intelligence and Applied Informatics/1st International Conference on Big Data, Cloud Computing, Data Science & Engineering, Las Vegas, NV, USA, 12–14 December 2016; pp. 99–104.
43. Konishi, M.; Okubo, S.; Nishino, T.; Wakatsuki, M. A Decision Tree Analysis of a Multi-Player Card Game with Imperfect Information. *Int. J. Softw. Innov.* **2018**, *6*, 1–17. [CrossRef]
44. Majchrzak, K.; Quadflieg, J.; Rudolph, G. Advanced Dynamic Scripting for Fighting Game AI. In *Entertainment Computing—ICEC 2015*; Chorianopoulos, K., Divitini, M., Baalsrud Hauge, J., Jaccheri, L., Malaka, R., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2015; Volume 9353, pp. 86–99, ISBN 978-3-319-24588-1.
45. Kerby, D.S. The Simple Difference Formula: An Approach to Teaching Nonparametric Correlation. *Compr. Psychol.* **2014**, *3*, 1–9. [CrossRef]
46. Lee, D.K. Alternatives to P Value: Confidence Interval and Effect Size. *Korean J Anesth.* **2016**, *69*, 555. [CrossRef] [PubMed]