Designing Knowledge-Based Rule-Based Agents in Schnapsen: Leveraging Human Winning Strategies

VU Amsterdam

No Institute Given

Abstract. This research focuses on developing a rule-based computer bot for the Schnapsen game, a trick-taking card game known for its strategic depth. In contrast to well-studied board games like chess and Go, Schnapsen poses unique challenges due to incomplete information and the influence of chance events. Leveraging insights from human strategies, we construct a rule-based system for the bot, with a particular focus on moderately aggressive gameplay. The experiment setup involves measuring the bot's performance against random, rdeep, and ml bots, with a sample of 1000 plays for each opponent. The results showcase the bot's competitive winning rates, providing insights into the efficacy of rule-based strategies in the context of Schnapsen.

Keywords: Rule-based Agent · Knowledge-based Agent · Human Strategies · Computer poker · Intelligenet Agent keyword.

1 Introduction

Developing computer players for adversarial strategy games has been a focal point of modern AI research for many years. Beginning with A. L. Samuel's world's first self-learning computer program in 1959, employing minimax search and alpha-beta pruning to play checkers, research for construct intelligent players for adversial games have produced notable success stories [1]. From Deep Blue defeating reigning world champion Garry Kasparov in chess in 1997[2] to the highly publicised AlphaGo's triumph over the human European Go champion by 5 games to 0 through embedded neural networks [3], computers have showcased their power to challenge human supremacy in various games, including chess, Tic-Tac-Toe, Hex, and Go.
However, despite these famous 2-player, zero-sum board games, card games such as poker and bridge can be more challenging for AI researchers. A critical difference between poker and those well-studied board games like Chess and Go is that the actions and states of poker are not complete for players (i.e. the inability to see opponents' hands), unlike the complete information provided by the fully observable board of chess-like games. Moreover, the existence of chance events in poker games — random private and public cards that are dealt each hand, will introduce significant uncertainty. While agents for perfect information

games mostly rely on tree search, the imperfect information nature of poker will make similar search techniques computationally more expensive.
From the vast majority of previous research on poker-playing agents, agent-building approaches can be categorized into 1) knowledge-based systems, 2) Monte-Carlo simulations, 3) game theoretic principles, and 4) adaptive imperfect information game trees [4]. However, developing high-performance AI agents often means the use of sophisticated methods (e.g., deep learning, neural networks) that involve computationally complex tasks far away from the knowledge we've learned.
In our research, we aim to focus on the side we naturally master: human logic and strategies. We include strategies and specific cases from domain experts to construct a rule-based computer bot. Strategies and tips for playing the Schnapsen game from experts will be condensed into a single winning strategy to provide guidelines for the agent. Literature on poker strategies already shows that aggressive strategies usually outperform passive ones [5], so the overall strategy is formulated in a moderately aggressive way.
The subsequent sections of the paper are structured as follows: Section 2 describes the rules of the Schnapsen game and explores related works on rule-based agents. Sections 3 and 4 outline our research question and experiment setup. Sections 5 present and discuss our experimental results. Finally, in Sections 6 and7, we offer conclusions and suggest possible future work.

2 Background and related work

2.1 The Schnapsen game

introduction to the game Schnapsen is a well-known trick-taking card game, which is popular in Bavaria, Austria, Hungary, and areas of the former Austro-Hungarian Empire. As the national card game of Austria and Hungary, it blends elements from both Point Trick and Trick-and-Draw card game types and known for its strategic depth. The game is mostly about earning points by winning tricks with high-value cards and making "marriages" – pairs of kings and queens. Players use a trick-and-draw approach, which gives them flexibility in how they play their cards. They don't need to follow suit until the talon (the pile of remaining cards) runs out or closed, and this period is called phase 1. Once the talon is closed, the game will enter phase 2 and becomes stricter about following suit and leading tricks. The goal of the game is to reach 66 points, and players have to keep track of their scores in their heads as using a scoresheet isn't allowed. If you miscalculate your score, you might face penalties or lose by mistake, which adds an element of challenge. Schnapsen is also a game of incomplete information, which means players can't see the entire state of the game. It creates a lot of uncertainties. With billions of possible situations, players must constantly adjust their strategies based on the limited information they have. This makes Schnapsen a game that's both mentally challenging and continuously engaging.

Rules of the game Schnapsen is a trick taking game that needs two players. A game consists of many deals and a single deal is won by the first player who manages to collect 66 points. In each deal, players collect cards by winning ‘tricks’. And the points can be obtained through each collected card and declaring a marriage which is a king and a queen in the same suit. Up to 3 points are able to be obtained in each deal depending on the trick points at the end of the deal. And the player who wins 7 points from deals wins the whole game. Schnapsen is played with 20 cards(5 cards in each suit). The rank and value are listed as follows: ACE–11, TEN–10, KING–4, QUEEN–3, JACK–2.

2.2 Knowledge-based Intelligent Agent

Rule-based The rule-based system utilises a vast array of predefined, human-crafted rules to facilitate decision-making and problem-solving in specific domains. Mimicking human decision processes, it applies these rules to relevant conditions and data to make decisions.[4] Typically, it uses an 'if-then' structure as its logical rules: 'if' means the condition or criteria, while 'then' indicates the consequent action. While highly effective in domains where expert knowledge can be distinctly formulated as rules, its efficacy is limited the rule set’s comprehensiveness and accuracy, and it lacks the capacity to adapt or 'learn' from new data.

Formula based The formula-based system, unlike the rule-based systems that operate on predefined rules, takes a border range of input data into account and uses mathematical formulas to calculate the probabilities of different results. The formula-based system is more generalised so it can be used in a wider range of situations by interpreting current state. But the formula-based system also has some limitations. The decision making progress involves intensive computations especially in complex situations. This can lead to high computational resource demands and may not be practical in real-time gaming environments where quick decisions are needed.

Cased-based reasoning agent The Case-based reasoning agent is another approach which uses past experience to analyse, understand and solve new problems. It makes decisions and predictions depending on the data of past cases. The Case-based reasoning agent has the ability to improve overtime by learning new cases. Also, it can be applied to various domains and problems since it does not rely on domain-specific rules. However, it requires relevant cases which could be difficult to find when the database is large. And how to reuse and adapt solutions from the past cases is also challenging.

Choice of the project For this project, we choose the rule-based agent. First, a rule-based system is relatively straightforward to implement. And due to the fact that this project is required to be finished in a short time, this approach is more suitable. Second, such an approach needs low computational resources. In that case, it is possible to make real-time responses in the games which is crucial. Additionally, the rule-based system is easy to make adjustments and maintain because the structure of the program is relatively simple.

3 Research question

Our research question is: How can the incorporation of human winning strategies help create a knowledge-based, rule-based agent in the Schnapsen game?

4 Experiment setup

4.1 Introduction to strategies

Poker-winning strategies typically fall into two categories: specific instances that guide optimal reactions in particular situations and overarching principles that dictate behavior across a multitude of games.. Considering the time constraint of the project, we will exclude case-specific strategies and only focused on combining strategies and tips into a unified winning approach.

Typically, pokers strategies can be classified based on several features, such as tight and loose. One important feature in describing a player's strategy is agressive or defensive. In the first phase of schnapsen, different levels of strategies can be conclude as follows:

1. Aggressive: Play trump, claim marriage when you have to win maximum points
2. Neutral: Keep trump, use the non-trump. Claim marriage if you have.
3. Defensive: Keep trump and marriage, use non-trump JQK.

Many research have shown that aggressive strategies usually dominate their passive counterparts. [The Dynamics of Human Behaviour in Poker]. Consequently, our focus will center on strategies that can be seen as rational and aggressive.

4.2 Rule-based systems

We control the rules with four primary sets of conditions, each comprising several sub-rules. The sub-rules will be selectively applied to the bot depending on whether the bot is in phase 1 or 2 of the game, and whether the bot is lead or not. When the conditions are met, corresponding sub-rules will be applied. The details of sub-rules are outlined below:

5 Experiment results

The bot is set to play against random, rdeep, and ml bots, keeping track of wins and losses. We chose a sample of 1000, meaning our bot will play against these

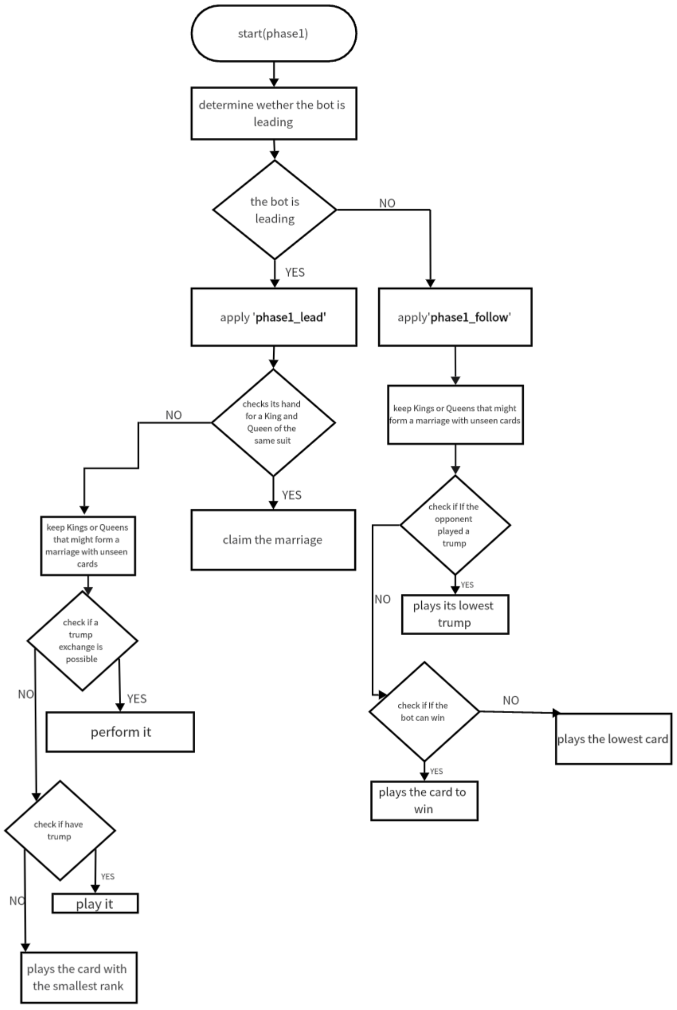
start(phase1)
determine wether the bot is leading
the bot is leading
NO
YES
apply 'phase1_lead'
apply'phase1_follow'
checks its hand for a King and Queen of the same suit
NO
YES
claim the marriage
keep Kings or Queens that might form a marriage with unseen cards
keep Kings or Queens that might form a marriage with unseen cards
check if If the opponent played a trump
YES
plays its lowest trump
NO
check if a trump exchange is possible
NO
YES
perform it
check if If the bot can win
NO
plays the lowest card
YES
plays the card to win
check if have trump
YES
play it
NO
plays the card with the smallest rank

Fig. 1. Flowchart of Phase1

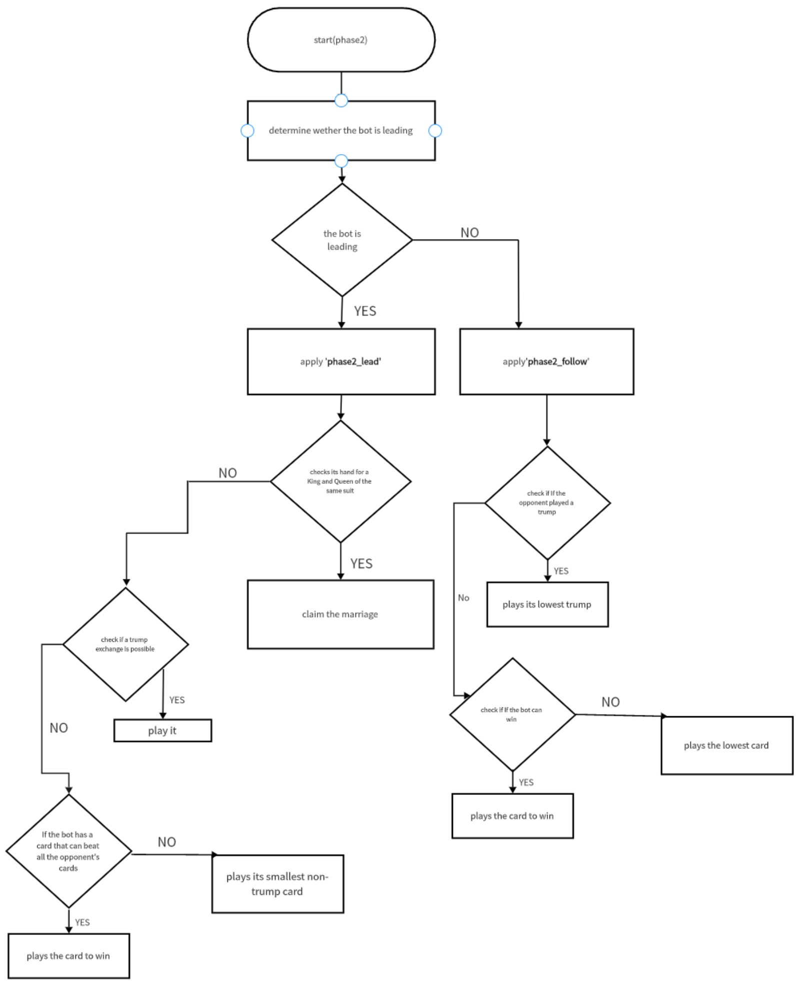
start(phase2)
determine wether the bot is leading
the bot is leading
NO
YES
apply 'phase2_lead'
apply'phase2_follow'
checks its hand for a King and Queen of the same suit
NO
YES
claim the marriage
check if If the opponent played a trump
YES
plays its lowest trump
No
check if a trump exchange is possible
YES
play it
NO
check if If the bot can win
NO
plays the lowest card
YES
plays the card to win
If the bot has a card that can beat all the opponent's cards
NO
plays its smallest non-trump card
YES
plays the card to win

Fig. 2. Flowchart of Phase2

Table 1. Bot Decision Rules

Condition	**Sub-rules**
Phase 1 and Lead	1. If there is a marriage: claim marriage if trump exchange is available, do trump exchange.
	2. If King or Queen cards in hand can claim marriage with unseen cards, keep the King or Queen.
	3. If there is trump jack on hand, play trump jack.
	4. If there is no trump jack, play the card with the smallest rank.
Phase 1 and Follow	1. If King or Queen cards in hand can claim marriage with unseen cards, keep the King or Queen.
	2. If opponent plays trump: play card with smallest rank.
	3. If opponent plays nontrump: if available, play the largest winning card, else play card with smallest rank.
Phase 2 and Lead	1. If there is a marriage: claim marriage.
	2. If have trump card in hand: play largest trump card.
	3. If no trump card in hand: 1) if a card can beat all opponent cards, play the card; 2) else play smallest nontrump card.
Phase 2 and Follow	1. If opponent plays trump: play card with smallest rank.
	2. If opponent plays nontrump: if available, play the largest winning card, else play card with smallest rank.

bots 1000 times, and then we calculate the win rate. The RuleBot demonstrates a winning performance against different opponents better than predicted. Its winning rate over the randbot stands 68.2, reflecting 682 victories compared to 318 losses. When runing against the rdeep bot, RuleBot has a winning rate of 52.3(523 wins against 477 losses). When facing mlbot, RuleBot achieving a winning rate of 37.7 (377 wins and 623 losses).

The code for our bot is constructed upon the Schnapsen framework, which incorporates game rules, maintains game states, and furnishes intelligent agents with a set of allowable actions.

6 Conclusion

In conclusion, our research presents a rule-based card-playing agent for the Schnapsen game, drawing inspiration from human strategies. The bot demonstrates competitive performance against different opponents, highlighting the potential of rule-based approaches in complex and uncertain gaming environments. Future work should explore strategies against human players, consider psychological gameplay elements, and delve into game theory. Additionally, investigating alternative methods, such as machine learning for adaptive agents, could enhance the bot's capabilities in tackling the complexity of the Schnapsen strategy.

7 Future work

In future work, it is essential to carefully consider strategies that play against human players. While strategies for 2-layer, non-sum games like chess, with perfect information and optimal solutions, may perform well, strategies may become intricate in the face of significant uncertainty arising from chance events. Decision-making in such scenarios requires consideration of numerous factors.

Additionally, some human strategies involve psychological gameplay against opponents. Random bots, which do not take into account the opponent's card distribution, may not effectively measure performance. Therefore, challenging expert human players could be a more optimal choice.

Many strategies delve into complex game theory, which might pose challenges for comprehension. Exploring and understanding these intricate game-theoretic elements would be a domain for future investigation.

Given the complexity of the Schnapsen strategy, it may not be ideal for rule-based approaches. Investigating alternative methods such as learning agents using machine learning or other adaptive techniques, could be a fruitful direction for further research.

References

1. Gio Wiederhold and John McCarthy. Arthur samuel: Pioneer in machine learning. *IBM Journal of Research and Development*, 36(3):329–331, 1992.
2. Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
3. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
4. Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5):958–987, 2011. Special Review Issue.
5. Marc Ponsen, Karl Tuyls, Steven Dejong, Jan Ramon, Tom Croonenborghs, and Kurt Driessens. The dynamics of human behaviour in poker. In *Proceedings of the 20th Belgian-Dutch Conference on Artificial Intelligence*, pages 225–232. Universiteit Twente, 2008.

References

References

1. Gio Wiederhold and John McCarthy. Arthur samuel: Pioneer in machine learning. *IBM Journal of Research and Development*, 36(3):329–331, 1992.
2. Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
3. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
4. Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5):958–987, 2011. Special Review Issue.
5. Marc Ponsen, Karl Tuyls, Steven Dejong, Jan Ramon, Tom Croonenborghs, and Kurt Driessens. The dynamics of human behaviour in poker. In *Proceedings of the 20th Belgian-Dutch Conference on Artificial Intelligence*, pages 225–232. Universiteit Twente, 2008.

A Appendix

A.1 Code of RuleBot

Listing 1.1. rulebot.py

```
from schnapsen.game import Bot, Move, PlayerPerspective, GamePhase, RegularMove, Ma
from schnapsen.game import SchnapsenTrickScorer
from schnapsen.deck import Card, Suit, Rank



class RuleBot(Bot):
    def get_move(self, perspective: PlayerPerspective, leader_move: Move | None) -> 
        if perspective.get_phase() == GamePhase.ONE:
            if perspective.am_i_leader():
                return self.phase1_lead(perspective, )
            else:
                return self.phase1_follow(perspective, leader_move)
        else:
            if perspective.am_i_leader():
                return self.phase2_lead(perspective)
            else:
                return self.phase2_follow(perspective, leader_move)
        return perspective.valid_moves[0]

    def phase1_lead(self, perspective: PlayerPerspective) -> Move:
        '''
        If there is a marraige: claim marriage. If trump exchange is available, do t
        If it's possible for the King or Queen cards in hand to claim marriage with
                If there is trump jack on hand, play trump jack.
        If there is no trump jack, play the card with the smallest rank.
            '''
        hand = perspective.get_hand()
        valid_moves = perspective.valid_moves()
        known_cards = perspective.get_opponent_won_cards()._cards + perspective.get_
        marriage = []
        for card1 in hand:
            for card2 in hand:
                if card1.suit == card2.suit and card1.rank == Rank.KING and card2.ra
                    return Marriage(card2, card1)
        for move in valid_moves:
            if move.is_trump_exchange():
                return move.as_trump_exchange()
        filtered_moves = []
        for mycard in hand:
            for card in known_cards:
                if (mycard.Rank == Rank.QUEEN or mycard.Rank == Rank.KING)\
                    and (mycard.Suit == card.Suit and mycard.Rank == card.Rank):
```

```
                continue
            else:
                filtered_moves.append(RegularMove(mycard))
                break
    for move in filtered_moves:
        if move.suit == perspective.get_trump_suit() and move.card.rank == Rank.J
            return move
    cards = [move.card for move in filtered_moves]
    cards_ranked = sorted(cards, key=lambda c: SchnapsenTrickScorer().rank_to_po
    return Move(cards_ranked[0])
    return perspective.valid_moves()


def phase1_follow(self, perspective: PlayerPerspective, leader_move: Move | None
    """
    1.If it's possible for any King or Queen cards in hand to claim marriage wit
    2.If opponent play trump: play card with smallest rank.
    3.If opponent play nontrump: if available, play the largest card that is abl
    """
    hands = perspective.get_hand()
    valid_moves = perspective.valid_moves()
    known_cards = perspective.get_opponent_won_cards()._cards + perspective.get_
    filtered_moves = []
    for mycard in hands: # rule 1
        for card in known_cards:
            if (mycard.rank == Rank.QUEEN or mycard.rank == Rank.KING)\
                and (mycard.suit == card.suit and mycard.rank == card.rank):
                continue
            else:
                filtered_moves.append(RegularMove(mycard))
                break
    cards = [move.card for move in filtered_moves]
    cards_ranked = sorted(cards, key=lambda c: SchnapsenTrickScorer().rank_to_po
    if leader_move.is_marriage():
        if leader_move.underlying_regular_move().card.suit == perspective.get_tru
            return RegularMove(cards_ranked[0])
    elif leader_move.card.suit == perspective.get_trump_suit():
        return RegularMove(cards_ranked[0])
    else: # rule 3
        for card in reversed(cards_ranked):
            if card.rank.value > leader_move.card.rank.value:
                return RegularMove(card)
        return RegularMove(cards_ranked[0])
    perspective.valid_moves()[0]

def phase2_lead(self, perspective: PlayerPerspective) -> Move:
    '''
    1. If there is a marriage: claim marriage.
    2. If have trump card in hand: play largest trump card
    3. If no trump card in hand: 1)if a card can beat the all cards the opponent
```

```
        2) else play the smallest nontrump card.
        '''
        valid_moves = perspective.valid_moves()
        opponent_hand = perspective.get_opponent_hand_in_phase_two().cards
        # sort card in terms of points in decending order
        cards = [move.card for move in valid_moves]
        cards_ranked = sorted(cards, key=lambda c: SchnapsenTrickScorer().rank_to_po
        # works for rule 1
        for card1 in hand:
            for card2 in hand:
                if card1.suit == card2.suit and card1.rank == Rank.KING and card2.ra
                    return Marriage(card1, card2)
        for card in cards_ranked: # works for rule 2
            if card.hand.is_trump():
                return RegularMove(card)
        for card in cards_ranked: # works for rule 3, part 1
            beat = False
            beat = any(card.suit == oppcard.suit and card.rank.value > oppcard.rank.
            if beat == True:
                return RegularMove(card)
        return RegularMove(cards_ranked[-1]) # works for rule 3, part 2
        return perspective.valid_moves()[0]

    def phase2_follow(self, perspective: PlayerPerspective, leader_move: Move | None
        '''
        If opponent play trump: play card with smallest rank.
        If opponent play nontrump: if available, play the largest card that is able
        '''
        valid_moves = perspective.valid_moves()
        cards = [move.card for move in valid_moves]
        cards_ranked = sorted(cards, key=lambda c: SchnapsenTrickScorer().rank_to_po
        if leader_move.is_marriage():
            if leader_move.underlying_regular_move().card.suit == perspective.get_tr
                return RegularMove(cards_ranked[0])
        elif leader_move.card.suit == perspective.get_trump_suit():
            return RegularMove(cards_ranked[0])
        else: # rule 3
            for card in reversed(cards_ranked):
                if card.suit == leader_move.card.suit and card.rank > leader_move.ca
                    return RegularMove(card)
            return RegularMove(cards_ranked[0])
        return perspective.valid_moves()[0]
```