Bridging Imperfect and Perfect Information in Schnapsen Through Strategic Gameplay Enhancements and the AlphaBeta Algorithm

Khaled Altahan & Laith M. M. H. Al-kabodi

Vrije Universiteit, Faculty of Sciences, Amsterdam, Netherlands

Abstract. AI has become a critical component in the development of strategic game-playing agents, particularly in games with imperfect information. This research focuses on optimizing the ProbabilityUtilityRandomBot (PU) to outperform the RdeepBot in the game of Schnapsen, a two-player card game that requires complex decision-making and strategic thinking. By bridging the gap between imperfect and perfect information in Schnapsen through strategic gameplay enhancements, this study considerably improves the agent's decision-making abilities by incorporating modular additions such as Alpha-Beta pruning, risk management, opponent card information monitoring, and follower-specific strategies. Following extensive research with various sub-bots, the final version, PU_AFRO, shows a significant increase in win rates against RdeepBot, providing useful insights into AI strategies for imperfect information games. This work contributes to bridging the challenges of imperfect information in game theory and lays the groundwork for future research into dynamic tactics and adaptive learning approaches.

Keywords: Schnapsen, Alpha-Beta pruning, Imperfect information, Game theory, AI

1 Introduction

In the unstoppable pursuit of creating a more efficient and interconnected world, AI and game theory have emerged as critical research fields. Relying on decades of accumulated knowledge and rapid advancements in computational power, these domains continue to revolutionize decision-making and problem-solving in complex scenarios. Our research positions itself at the crossroads of AI and game theory, focusing on the Austrian card game Schnapsen. Specifically, we aim to enhance the capabilities of the ProbabilityUtilityRandomBot (PU), an agent used in our university's research in the game of Schnapsen, to outperform one of the strongest existing agents of the game, Rdeep, which was built upon the foundations of Monte Carlo methodologies described by Wisser (2015) [1].

Our efforts address the complex challenges posed by the imperfect information of the two-hands game Schnapsen [2], where strategic decision-making requires navigating uncertainties and incomplete knowledge. To bridge this gap between imperfect and perfect information, we implemented several advanced strategies, including the Alpha-Beta pruning algorithm, risk factor, opponent card awareness, and follower-specific tactics. These enhancements were modularly integrated into our bot's design, represented as PU_A (Alpha-Beta), PU_AF (follower strategies), PU_AFO (opponent-known cards), and PU_AFR (risk management), resulting in the comprehensively optimized PU_AFRO agent. This approach allows a systematic study of each strategy's contribution to the bot's performance.

Our research advances the state of AI agents in this domain by integrating these ideas with recent advancements in optimizing strategies for imperfect information games, as discussed by Hua et al. (2024) [3].

2 Background Information

2.1 Introduction to the Domain

Artificial intelligence has significantly advanced the domain of strategic game playing, particularly in games with incomplete observable states. These games challenge AI systems to excel in pattern recognition, probability calculations, and complex decision-making under uncertainty. Schnapsen, a traditional Austrian card game, exemplifies this environment, offering a dynamic and adaptable testing ground for AI research.

2.2 Overview of Schnapsen

Schnapsen is a two-player Austrian card game, played with a 20-card deck consisting of Jacks, Queens, Kings, Tens, and Aces, valued at 2, 3, 4, 10, and 11 points, respectively [4]. The game is divided into two phases, each governed by distinct rules. Players aim to score 66 points through winning tricks or melding marriages—specific combinations of Kings and Queens that yield bonuses of 20 or 40 points for royal marriages. The round concludes once a player reaches 66 points, and the overall winner is determined by accumulating seven game points across multiple rounds. Schnapsen has a state-space of approximately 10^{20}, making it a highly complex environment for strategic decision-making, as mentioned in [5]. Detailed rules and variations of the game are documented in sources such as Pagat [2].

2.3 Key Concepts and Techniques

Alpha-Beta pruning is a foundational technique in game AI, known for its efficiency in perfect information scenarios by pruning the search tree to focus on optimal moves [6]. This research adapts the Alpha-Beta algorithm to Schnapsen, accommodating its imperfect information structure. Complementary strategies, such as risk management and opponent knowledge tracking, are integrated to enhance the bot's ability to dynamically adjust its gameplay in response to evolving game states.

2.4 Overview of the Bots

This research leverages and evaluates several AI agents, each designed to address different strategic aspects of Schnapsen. Below are descriptions of the key bots utilized in this study:

- **ProbabilityUtilityRandomBot (PU)**: The baseline bot focuses on probabilistic calculations to minimize risks during gameplay. It uses probability theory [7] to estimate the likelihood of opponents holding higher-ranking cards and adapts its decisions accordingly.
- **AlphaBetaBot**: This bot employs the Alpha-Beta pruning algorithm to efficiently search game trees for optimal moves, enabling high performance in strategic decision-making, particularly in perfect information scenarios, like the second phase of Schnapsen card game.
- **RandBot**: A simple yet effective bot that selects moves randomly. While it lacks strategic depth, it serves as a control agent to benchmark the performance of more advanced bots.
- **RdeepBot**: A sophisticated agent based on Monte Carlo sampling and depth-limited simulations. Rdeep is renowned for its ability to balance exploration and exploitation, making it a challenging opponent in games like Schnapsen.
- **BullyBot**: The BullyBot ensures that all its moves are valid. When it holds cards of the trump suit, it selects one at random to play. If it is the follower and has cards matching the opponent's suit, it randomly chooses a card from those options. In cases where neither condition applies, the bot plays a card with the highest score, selected randomly.

These bots provided a robust framework for testing and validating the modular enhancements integrated into PU_AFRO, highlighting the strengths and limitations of each approach in addressing Schnapsen's unique challenges.

3 Research Question

The issue of improving a playing bot can be approached in various ways, either by incorporating human-like strategies or leveraging the power of AI algorithms. In this research, we aim to answer the question: "How does integrating four strategies—(1) strategic play as a follower during Phase 1, (2) refining the probability-based utility heuristic with a dynamic risk factor, (3) leveraging opponent-known cards to enhance probability calculations as the leader, and (4) implementing the AlphaBetaBot in Phase 2—affect the overall win rate of the ProbabilityUtilityRandBot in the Schnapsen card game?". We formulated four hypotheses for this research. The first hypothesis H_1 posits the final bot PU_AFRO will outperform both baseline bots PU_A and *RdeepBot*. The second hypothesis H_2 expects that the introduction of the follower strategy will significantly improve the performance of the bot by optimizing its response to the leader's moves and increasing its win rate. The third hypothesis H_3 is that the implementation of the risk factor strategy will significantly enhance the utility heuristic by dynamically adjusting the bot's risk tolerance based on its position in the game, leading to improved performance. Finally, the fourth hypothesis H_4 predicts that the inclusion of the opponent's known cards and marriage strategy will significantly enhance the utility heuristic by improving the accuracy of probability calculations, leading to a noticeable boost in the bot's win rate. Performance, in all hypotheses, is defined as the overall win rate of the bot.

To systematically test our hypotheses, we created eight different bots, each incorporating distinct combinations of strategies. Replacing the *RandBot* with the *AlphaBetaBot* in the second phase of the game results in the *PU_A* bot, where (A) represents AlphaBeta, and this bot will serve as a baseline for testing the second hypothesis. Similarly, *PU_AF* is a bot that incorporates the follower strategy (F) in the first phase as a follower while using *AlphaBetaBot* in the second phase. In *PU_AO*, the (O) represents the use of the opponent's known cards and marriage strategy, while the (R) in *PU_AR* refers to the risk factor strategy. Using these abbreviations, we systematically created the following bots, each representing different combinations of strategies: *PU_A*, *PU_AO*, *PU_AR*, *PU_ARO*, *PU_AF*, *PU_AFO*, *PU_AFR*, and the final bot *PU_AFRO*, which integrates all the strategies. This structured approach allows us to evaluate the individual and combined impact of these strategies on the bot's performance in Schnapsen.

4 Experimental Setup

4.1 Strategies and Changes

The improvement procedure involves four major changes. The first is the replacement of RandBot with AlphaBetaBot, introducing a more efficient algorithm for decision-making in the second phase of the game. The second improvement is the follower strategy, which addresses two primary conditions. The first condition occurs when the opponent leads with a trump card: the bot plays the first non-trump Jack, or, if unavailable, a non-trump Queen or King, preserving the trump Queen and King for a potential future royal marriage and saving trump cards for the second phase of the game. If the bot lacks a non-trump Jack, Queen, or King, it plays a higher-ranked card in the trump suit to win the trick. If none of these conditions are met, it plays the lowest-ranked non-trump card. The second condition arises when the leader card is not a trump card: in this case, the bot plays a higher-ranked card from the same suit as the leader card, if possible. If not, it evaluates whether the leader played a non-trump Ace or Ten and, if holding a trump card, plays the lowest-ranked trump card. If none of these criteria are met, the bot minimizes its losses by playing the lowest-ranked non-trump card.

The third improvement introduces a risk factor strategy grounded in game theory and risk dominance. This approach adjusts the bot's strategy based on its relative position in the game, reflecting the principle that players who are trailing adopt riskier strategies to improve their chances of success, while those in the lead tend to play conservatively to protect their advantage [8]. In our implementation, the utility heuristic calculation is increased when the bot is significantly ahead and decreased when it is far behind. The risk factor operates at different levels, each corresponding to a specific game condition, ranging from significantly ahead to significantly behind, with each level dictating whether the bot plays aggressively or conservatively.

The final improvement leverages knowledge of the opponent's cards to refine probability calculations in the first phase of the game. By incorporating information about known dangerous cards held by the opponent, the bot enhances its estimation of the likelihood that the opponent has a card capable of winning a given move, resulting in more accurate probability assessments. In addition, the opponent-known card strategy prioritizes moves involving marriages and trump exchanges, executing these actions first.

4.2 Experimental Approach

The four hypotheses introduced in this research require distinct experimental methodologies. To test whether the final improved bot *PU_AFRO* will outperform *RdeepBot*, we will conduct an indirect comparison experiment, where *PU_AFRO* will play a tournament against *BullyBot* and in another tournament will be *RdeepBot* vs. *BullyBot*. In addition to the indirect comparison, a direct comparison will be conducted between *RdeepBot* and *PU_AFRO* in order to make the difference in both bot's performance clear. Introducing RdeepBot to this stage allows us to measure how the final bot performs against the strongest bot—defined by win rate—among those included in this study.

Subsequently, another set of experiments was designed to address whether each individual development stage (F, R, and O) contributes a significant improvement to the bot's performance. In these experiments, the PU_A bot was used as the baseline: PU_AF (baseline + follower strategy) against PU_A, PU_AR (baseline + risk factor) against PU_A, and so on. This approach isolated the contribution of each strategy (F, R, O) and their combinations.

4.3 Test Framework

For each hypothesis of our four hypotheses, the null hypothesis $H_0 : \pi_A = \pi_B$ was tested against the alternative hypothesis $H_a : \pi_A > \pi_B$, where π_A represents the probability of the improved bot winning the tournament, and π_B represents the probability of the baseline bot winning. Each tournament consisted of 1,000 games between the improved bot and the baseline bot, ensuring that randomness had minimal influence on the results. Tournament implementation details can be found in the Appendix.1. To ensure reproducibility, a fixed seed value of 2025 was used for randomness, and for RdeepBot, we applied sample sizes of 10 and a depth of 5. The p-value for each tournament was calculated using the binomial test [9]. A significance level of 0.05 was used to evaluate whether the observed differences in win rates were statistically significant.

5 Results

5.1 Indirect Comparison Between *PU_ AFRO* and *RdeepBot*

The indirect comparison between *PU_ AFRO* and *RdeepBot* was conducted using *BullyBot* as an intermediary opponent (Figure 2). According to the results, *RdeepBot* defeated *BullyBot* 898 times out of 1000 games, while *PU_ AFRO* defeated *BullyBot* 869 times. At the 0.05 significance level, a z-test produced a p-value of 0.043, showing a statistically significant difference between *PU_ AFRO* and *RdeepBot*'s performance. This result supports the first null hypothesis, which posits that *PU_ AFRO* does not perform noticeably better than *RdeepBot*.

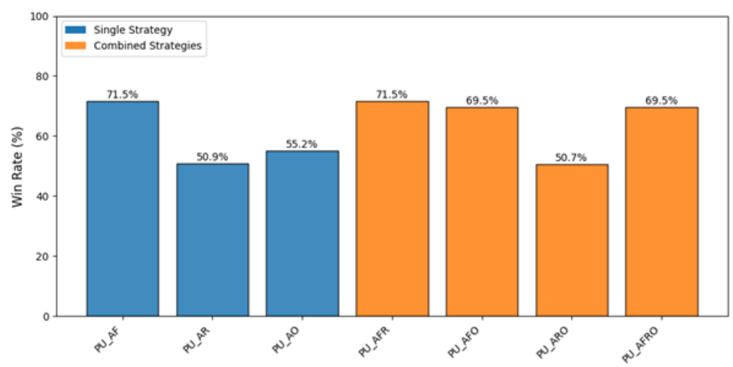

Fig. 1. This graph illustrates how each strategy performed against the baseline bot PU_A when they were tested in isolation (the blue bars), and when combined with other strategies as in the orange bars.

5.2 Direct Comparison Between *RdeepBot* and *PU_ AFRO*

Another experiment was conducted using a direct comparison method between *RdeepBot* and *PU_ AFRO*. The results were consistent with the indirect comparison, showing statistical significance (p-value: 0.004) in the performance of *RdeepBot* defeating *PU_ AFRO*, with win rates of 54.50% and 45.50%, respectively.

5.3 Analysis of Individual Strategies

To evaluate the impact of individual strategies, we conducted experiments with various combinations of enhancements against the baseline bot *PU_ A*. Table 1 summarizes the win rates and p-values for each strategy combination.

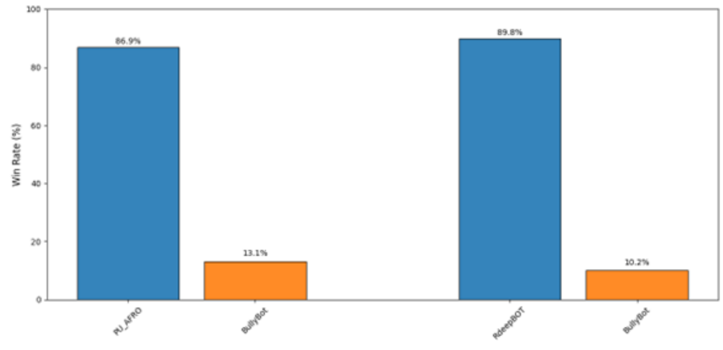

Fig. 2. Indirect comparison between *RdeepBot* and *PU_AFRO*.

Table 1. Win rates and p-values for all combinations of strategies against the baseline bot *PU_A*.

Strategy	Win Rate	P-value	Statistical Significance
PU_AF	71.50%	3.000132782487448e-43	Yes
PU_AO	55.20%	0.001115049179268907	Yes
PU_AR	50.90%	0.5908841078023002	No
PU_AFR	71.50%	3.000132782487448e-43	Yes
PU_AFO	69.50%	1.1667712491899422e-35	Yes
PU_ARO	50.70%	0.6810229832764916	No
PU_AFRO	69.50%	1.1667712491899422e-35	Yes

Follower Strategy The follower strategy demonstrated a dramatic improvement in win rates, achieving 71.50% when applied in isolation. Even when combined with the risk factor strategy, the win rate remained at 71.50%. However, when combined with the opponent's known cards strategy, the win rate decreased slightly to 69.50%. These results highlight the importance of incorporating human-like strategies to address the challenges posed by imperfect information situations, supporting the hypothesis that the follower strategy significantly improves the bot's performance.

Opponent's Known Cards Strategy The opponent's known cards strategy achieved a win rate of 55.20% when applied in isolation, with statistical significance. This supports the hypothesis that incorporating the opponent's known cards strategy enhances the utility heuristic by improving the accuracy of probability calculations. However, when combined with the risk factor strategy, the improvement was not statistically significant.

Risk Factor Strategy The risk factor strategy alone achieved a win rate of 50.90%, indicating no significant enhancement to heuristic efficiency. Similarly, when combined with the opponent's known cards strategy, it failed to yield any statistically significant improvement. These findings support the null hypothesis that the risk factor strategy does not significantly enhance the utility heuristic by dynamically adjusting the bot's risk tolerance based on its position in the game.

5.4 Alpha-Beta Algorithm in Perfect Information Environments

To highlight the importance of the Alpha-Beta algorithm in perfect information environments, an additional experiment was conducted where PU played against PU_A. The results demonstrated the robustness of the Alpha-Beta algorithm, achieving a win rate of 62.30%. This further underscores its effectiveness in optimizing decision-making under conditions of perfect information.

6 Findings

The previous analysis highlights several important observations. One key finding is the significance of incorporating the AlphaBeta algorithm in adversarial searches within perfect information environments. This is because the Alpha-Beta algorithm consistently identifies the optimal solution more efficiently than other search algorithms due to its pruning technique, which significantly reduces computational overhead.

Another critical observation is the value of including human-like strategies, such as the follower strategy, when dealing with imperfect information environments. The notable improvement observed with the follower strategy can be attributed to its human-inspired nature. In real-life scenarios, humans process vast and continuous streams of information yet manage to function effectively by employing such strategies, which rely on strategic planning rather than extensive computational power or memory resources.

Regarding the modest improvement observed with the opponent's known cards enhancement, it did not yield the same level of progress as the follower strategy. This outcome may be due to the fact that it merely refines the heuristic and probability calculations, unlike the follower strategy, which introduced an entirely new strategic approach.

As for the risk factor strategy, it failed to contribute to the enhancement of the utility heuristic. This shortcoming requires further analysis and observation of the game states to gain a deeper understanding of what went wrong. From an initial analysis of the results and associated graphs, it appears that the risk factor might be only useful if it is actually applied to the bot's decision-making logic. In our strategy, the risk factor is used to adjust the utility_heuristics value in the leader role. However, if the utility heuristic itself is not sensitive to the risk factor, the bot's behavior won't change. If base_utility is already very small or dominated by other factors (e.g., probability), multiplying it by the risk factor might not have a significant impact. This indicates that the risk factor strategy requires more refined implementation or alternative approaches to integrate better with the utility heuristic.

Table 2. Hypothesis Testing Results

Hypothesis	Description	Null Hypothesis Rejection
H_1	PU_AFRO > RdeepBot	No
H_2	Follower strategy will improve the bot win rate	Yes
H_3	Risk factor will improve the bot win rate	Yes
H_4	Opponent's known card strategy will improve the bot win rate	No

7 Conclusion

In this research, we aimed to optimize the ProbabilityUtilityRandomBot for the Austrian card game Schnapsen by leveraging various improvements targeting the decision-making process in the first phase of the game. Additionally, we implemented an aware playing process in the second phase of the game using the Alpha-Beta pruning bot. The introduction of Alpha-Beta pruning enabled efficient search in complex game trees, while the addition of opponent knowledge tracking allowed for more informed and adaptive gameplay. Furthermore, the follower strategies provided a careful and dynamic approach to responding to the opponent's moves. On the contrary, risk factor strategy did not add any value to the bot.

The resulting ultimate bot, PU_AFRO, achieved superior performance, consistently outperforming the baseline ProbabilityUtilityRandomBot and demonstrating competitive performance against RdeepBot. These results emphasize the importance of incorporating human-like normative strategies into AI agents, particularly in games characterized by incomplete information.

Despite these advancements, several limitations persist, such as the unexpectedly poor performance of the risk factor strategy. Furthermore, while the modular strategy was effective, incorporating future AI approaches such as reinforcement learning or deep neural networks could yield even greater benefits. Overall, this study provides valuable insights into the application of AI in strategic game play and establishes a foundation for further exploration and development

8 Future Work

The performance of the PU_AFRO bot can be further enhanced by exploring several potential directions. Some promising avenues for future development include:

- Refinement of Risk Factor Strategies: Future iterations could focus on fine-tuning the risk management algorithms to dynamically adjust to varying game states and enhance decision-making under uncertainty.
- Development of Context-Aware Strategies: Incorporating more adaptive strategies that enable the bot to analyze and respond effectively to different opponent play styles would contribute to its versatility and robustness.

– Integration of Machine Learning Techniques: Leveraging advanced techniques such as reinforcement learning could significantly improve the bot's decision-making processes during both the leader and follower roles in the first phase of the game.
– Enhanced Opponent Modeling: Introducing predictive models to anticipate opponent actions and adjust the bot's gameplay accordingly could bridge the gap between imperfect and perfect information strategies.

These enhancements represent the next steps in advancing PU_AFRO's capabilities, contributing to the broader field of AI and strategic game development.

References

1. Wisser, F. (2015). *An expert-level card playing agent based on a variant of perfect information Monte Carlo sampling.* Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), 125–131. Retrieved from `https://www.ijcai.org/Proceedings/15/Papers/025.pdf`.
2. Pagat. *Schnapsen rules.* Retrieved from `https://www.pagat.com/marriage/schnaps.html`.
3. Hua, W., Liu, O., Li, L., Amayuelas, A., Chen, J., Jiang, L., Jin, M., Fan, L., Sun, F., Wang, W., Wang, X., & Zhang, Y. (2024). *Game-theoretic LLM: Agent workflow for negotiation games.* Retrieved from `https://arxiv.org/abs/2411.05990`.
4. Wikipedia. *Schnapsen.* Retrieved from `https://en.wikipedia.org/wiki/Schnapsen`, Accessed: 2025-01-26.
5. Canvas. *Course Materials for Artificial Intelligence.* Retrieved from `https://canvas.vu.nl/courses/77653/files/8349213?wrap=1`, Accessed: 2025-01-26.
6. Newell, A., & Simon, H. A. *Computer Science as Empirical Inquiry: Symbols and Search.* Artificial Intelligence, **2**(3-4), 1975, 181–234. Retrieved from `https://www.sciencedirect.com/science/article/abs/pii/0004370275900193`, Accessed: 2025-01-26.
7. Wikipedia. *Probability theory.* Retrieved from `https://en.wikipedia.org/wiki/Probability_theory`, Accessed: 2025-01-26.
8. Wikipedia. *Risk Dominance.* Retrieved from `https://en.wikipedia.org/wiki/Risk_dominance`.
9. Statsmodels Development Team. (Year). *statsmodels.stats.proportion.binom_test.* Retrieved from `https://www.statsmodels.org/dev/generated/statsmodels.stats.proportion.binom_test.html`.
10. Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River, NJ: Prentice Hall. Retrieved from `http://aima.cs.berkeley.edu/`.
11. (Year). *Creating Possible Worlds Using Sims Tables for the Imperfect Information Card Game Schnapsen.* Retrieved from `https://doi.org/10.1109/ICTAI.2010.76`.

Appendix: Additional Details and Implementation Code

.1 Tournament Visualization

Below is a figure visualizing the results of the tournaments conducted:

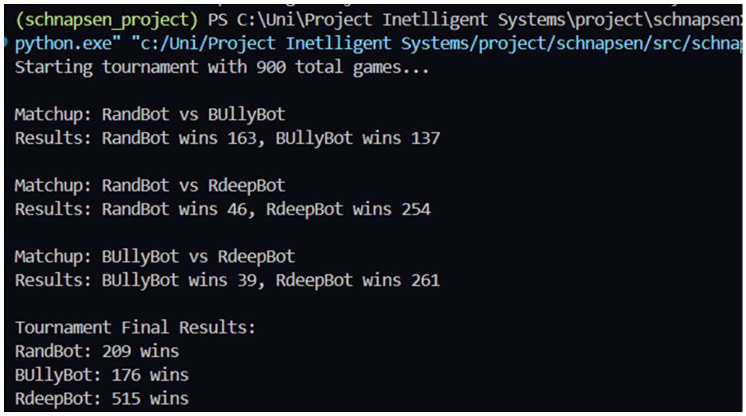

Fig. 3. Tournament results visualized for comparison.

.2 PU_AFRO Bot Implementation Code

The following Python code outlines the implementation of the `PU_AFRO` bot:

Listing 1.1. PU_AFRO Bot Implementation Code

```python

import random
from typing import Optional
from schnapsen.bots import AlphaBetaBot
from schnapsen.game import (
    Bot,
    Move,
    PlayerPerspective,
    GamePhase,
)
from schnapsen.deck import Card, Rank

class PU_FRO(Bot):
    """
    This bot is designed for Phase 1 of the game, where three strategies work together.
    In cases where the bot is the leader, two strategies are activated.
    The first is the risk factor strategy (referred to as R), which adjusts the value of the utility heuristic based on the game state.
    The second strategy is the opponent's known card strategy (O), which prioritizes playing a marriage move or exchange move first.
    In addition, this strategy utilizes the opponent's known cards to maximize the accuracy of the probability calculation that there is no higher
```

```python
    card of the same suit in the opponent's hand.
    If the bot is the follower, the follower strategy (F) is activated. This strategy is coded to minimize the loss of high-ranked cards
    and attempts to win the trick when a high-ranked card is led by the opponent.
    """
    def __init__(
        self, rand: random.Random, name: Optional[str] = "PU_FRO"
    ) -> None:
        super().__init__(name)
        self.rng = rand

    def calculate_risk_factor(self, perspective: PlayerPerspective) -> float:
        """Calculate risk factor based on game state.

        Args:
            perspective: The player's perspective of the game.

        Returns:
            float: The calculated risk factor.
        """
        my_score = perspective.get_my_score().direct_points
        opponent_score = perspective.get_opponent_score().direct_points
        total_my_score = my_score + perspective.get_my_score().pending_points
        total_opponent_score = opponent_score + perspective.get_opponent_score().pending_points
        score_difference = total_my_score - total_opponent_score

        # Base risk factor starts at 1.0
        risk_factor = 1.0

        # Adjust risk based on score difference and nearing endgame
        if total_opponent_score >= 50:  # Opponent close to winning
            if score_difference <= -20:
                risk_factor = 1.8  # High risk-taking if far behind
            else:
                risk_factor = 1.3
        elif total_my_score >= 50:  # We're close to winning
            if score_difference >= 20:
                risk_factor = 0.5  # Low risk when far ahead
```

```
            else:
                risk_factor = 0.8
        else:  # General cases based on score difference
            if score_difference >= 20:  # Significantly ahead
                risk_factor = 0.7
            elif score_difference >= 10:  # Moderately ahead
                risk_factor = 0.85
            elif score_difference <= -20:  # Significantly behind
                risk_factor = 1.5
            elif score_difference <= -10:  # Moderately behind
                risk_factor = 1.25

        return risk_factor

    def calculate_probability(self, unseen_cards: list[Card], dangerous_cards: list[Card]) -> float:
        """Calculate the probability of the opponent not having dangerous cards.

        Args:
            unseen_cards: List of cards not yet seen in the game.
            dangerous_cards: List of cards that could be dangerous if the opponent has them.

        Returns:
            float: The calculated probability.
        """
        u = len(unseen_cards)  # Total unseen cards
        d = len(dangerous_cards)  # Total dangerous cards

        if u < 5:
            return 0.0  # Avoid division by zero in edge cases

        probability = (   #Calculate the probability that the opponent has a card that would win our move
            (u - d) / u *
            (u - d - 1) / (u - 1) *
            (u - d - 2) / (u - 2) *
            (u - d - 3) / (u - 3) *
            (u - d - 4) / (u - 4)
        )
        return probability

    def get_move(self, perspective: PlayerPerspective, leader_move: Optional[Move]) -> Move:
        #get opponent knowen cards
```

```
        opponent_known_cards = perspective.get_known_cards_of_opponent_hand()
        scorer = perspective.get_engine().trick_scorer
        valid_moves = perspective.valid_moves()

        # Leader Role
        if leader_move is None:  # Bot is the leader
            for move in valid_moves:
                if move.is_marriage() or move.is_trump_exchange:
                    return move  # Prioritize marriages or trump exchanges
            #Get the unseen cards so far and exclude the cards that we already konw that the opponent have in their hand
            unseen_cards = [
                card for card in perspective.get_engine().deck_generator.get_initial_deck()
                if card not in perspective.seen_cards(leader_move=None)
                and card not in perspective.get_known_cards_of_opponent_hand()
            ]

            risk_factor = self.calculate_risk_factor(perspective) #calculate the risk factor

            max_utility_heuristics = float('-inf') #set the max value for the heuristic for the coming comparison
            chosen_move = None

            for move in valid_moves:
                move_card = move.cards[1] if move.is_marriage() else move.card

                # Identify dangerous cards for this move
                dangerous_cards = [
                    card for card in unseen_cards
                    if card.suit == move_card.suit and scorer.rank_to_points(card.rank) > scorer.rank_to_points(move_card.rank)
                ]

                #Add known opponent cards that are dangerous
                dangerous_cards = [
                    card for card in opponent_known_cards
                    if card.suit == move_card.suit and scorer.rank_to_points(card.rank)> scorer.rank_to_points(move_card.rank)
```

```
                ]


                # Calculate probability and utility
                probability = self.calculate_probability(unseen_cards, dangerous_cards)
                points = scorer.rank_to_points(move_card.rank)
                base_utility = probability / points #Divide by the value of the move(11, 10, 4, 3, 2) so that high cards value would have less probability
                utility_heuristics = base_utility * risk_factor#Multiply by the risk factor
                # find the move that has the highest probability to win the trick
                if utility_heuristics > max_utility_heuristics:
                    max_utility_heuristics = utility_heuristics
                    chosen_move = move

            return chosen_move or valid_moves[0]  # Fallback to first move

        # Follower Role
        else:
            #handel the case where the leader playes a marriage move
            if leader_move.is_marriage():
                leader_card = leader_move.cards[1]
            else: #not a marriage move
                leader_card = leader_move.card
            leader_card_suit = leader_card.suit

            # Find cards with the same suit, trump cards, and non-trump cards
            same_suit_cards = [move for move in valid_moves if move.card.suit == leader_card_suit]
            trump_cards = [move for move in valid_moves if move.card.suit == perspective.get_trump_suit()]
            non_trump_cards = [move for move in valid_moves if move.card.suit != perspective.get_trump_suit()]

            # When the trick is led by a trump card
            if leader_card.suit == perspective.get_trump_suit():
                for move in valid_moves:
```

```python
                    #if the move is non-trump jack
                    if move.card.rank == Rank.JACK and move.card.suit != perspective.get_trump_suit():
                        return move  # Play a non-trump jack

                for move in valid_moves:
                    #else the move is non-trump Queen or King
                    if move.card.rank in [Rank.QUEEN, Rank.KING] and move.card.suit != perspective.get_trump_suit():
                        return move  # Play non-trump Q/K with no marriage potential
                for move in trump_cards:
                    #else win the trick if the move rank is higher than the leader move
                    if scorer.rank_to_points(move.card.rank) > scorer.rank_to_points(leader_card.rank):
                        return move  # Play to win
                return min(non_trump_cards, key=lambda move: scorer.rank_to_points(move.card.rank))  # Lose lowest value

            # When the trick is led by a non-trump card
            else:
                for move in same_suit_cards:
                    if scorer.rank_to_points(move.card.rank) > scorer.rank_to_points(leader_card.rank):
                        return move  # Win with lowest non-trump card that isn't marriage potential
                for move in trump_cards:
                    #If the opponent led with a non-trump Ace or ten
                    if leader_card.rank in [Rank.ACE, Rank.TEN] and leader_card_suit != perspective.get_trump_suit() and trump_cards:
                        return min(trump_cards, key=lambda move: scorer.rank_to_points(move.card.rank)) # Play lowest trump

                return min(non_trump_cards, key=lambda move: scorer.rank_to_points(move.card.rank))  # Lose lowest value


class PU_AFRO(Bot):
```

```python
    """
    This is a two-stage bot. In the first stage, it applies PU_FRO bot. In the second stage, it uses Alphbeta Bot.

    Args:
        perspective: An object representing the perspective of the player.
        leader_move: The move of the leader. If this is None, we are the leader.

    Returns:
        A Move object
    """
    def __init__(self, rand: random.Random, name: Optional[str] = "PU_AFRO") -> None:
        super().__init__(name)
        self.rng = rand
        self.bot_phase1: Bot = PU_FRO(rand=self.rng, name="ProbabilityUtilityBot")
        self.bot_phase2: Bot = AlphaBetaBot(name="AlphaBetaBot")

    def get_move(self, perspective: PlayerPerspective, leader_move: Optional[Move]) -> Move:
        if perspective.get_phase() == GamePhase.ONE:
            return self.bot_phase1.get_move(perspective, leader_move)
        elif perspective.get_phase() == GamePhase.TWO:
            return self.bot_phase2.get_move(perspective, leader_move)
        else:
            raise AssertionError("Invalid game phase.")
```