Knowledge Modelling, Strategy Designing, and Agent Engineering for Reconnaissance Blind Chess

Robin Stöhr [1][a], Shuai Wang[1][b] and Zhisheng Huang[1][c]
[1]*Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands*
rb.stoehr@gmail.com, {shuai.wang | z.huang}@vu.nl

Keywords: Reconnaissance Blind Chess, knowledge modelling, Theory of Mind.

Abstract: Reconnaissance Blind Chess (RBC) is a unique chess variant where players have limited visibility of a 3x3 square in each round. This paper offers a comparative analysis of the performance of extant agents, along with an assessment of their ability to model their opponents' knowledge. On the basis of our analytical findings, we propose novel and efficient sensing and movement strategies. Subsequently, these strategies are tested through agent-based gameplay. Furthermore, our experimentation extends to the inference of new knowledge through a strategy based on the Theory of Mind. Collectively, these insights contribute to the selection of the most promising strategies for the design of our Scorca agent. By the time of the paper's submission, it occupies the second position on the global leaderboard for the RBC game. To conclude, we engage in a discussion of the inherent limitations of the extant agents and offer a glimpse into potential future strategies.

1 Introduction

Introduced by researchers at the Johns Hopkins University Applied Physics Laboratory (JHU/APL) as a variant of chess with imperfect information, Reconnaissance Blind Chess (RBC)[1] engenders a 'fog of war' mechanic: a player is unaware of the opponent's pieces' positions and must "sense" a 3x3 area of the 8x8 board each round to glean information (Newman et al., 2016). This added complexity of limited knowledge significantly changes the dynamics and strategies of the game. Tournaments of RBC have been held as part of the Conference on Neural Information Processing Systems (NeurIPS) conferences in 2019, 2021, and 2022 (Perrotta et al., 2019).

Figure 1 is a real example[2] where the agent playing black gains (true) knowledge in this turn by deciding to sense the 3x3 square highlighted in black. Subsequently, the player attempts to move their queen to capture the opponent's white queen, presuming that there is no obstruction between them. However, this action is found to be invalid, resulting in the system executing a legitimate move. Consequently, an unanticipated caption occurs. The player is only notified that the intended move failed, but with something captured unexpectedly (but not informed what piece) at some intermediate location. The player is always aware of the location of its pieces. Unlike other agent systems, in this game, agents do not communicate directly. The information passes through the platform as a notification to the agents.

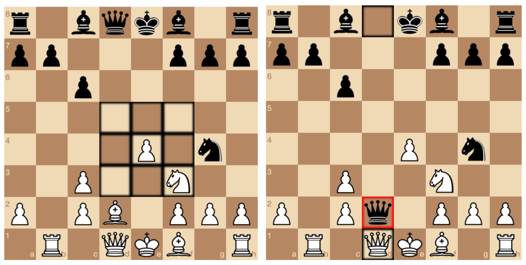
Figure 1: Example of an interrupted move. Black senses, and then tries to move the queen d8→d1. But as there is a bishop on d2, the move stops there and the bishop is captured.

This example illustrates that accurate modelling of the opponent's knowledge (about the position of the pieces) and an estimate of the player's strategy are crucial for the player's decision of their immediate next move as well as their strategy towards winning. The RBC game can be used as an alternative to the Poker game to explore and evaluate the mod-

[a] https://orcid.org/0000-0002-2600-6318
[b] https://orcid.org/0000-0002-1261-9930
[c] https://orcid.org/0000-0003-3794-9829
[1]https://rbc.jhuapl.edu/. Detailed rules of the game are at https://rbc.jhuapl.edu/gameRules.
[2]The entire game is online at https://rbc.jhuapl.edu/games/628294.

elling of knowledge inspired by the Theory of Mind (ToM). The game platform makes it easy to evaluate strategies and compare different parametric settings.

The unique mechanics of RBC contribute to a profound strategic depth that significantly surpasses traditional chess. The addition of the 'sensing' action, the incomplete knowledge of the opponent's pieces, and the uncertainty introduced by blind captures result in a complex set of possible states of the chess board associated with the uncertainty of knowledge that players must navigate. For human players, this complexity not only requires strategic thinking but also introduces an element of psychological warfare, as players must try to anticipate their opponents' actions and decisions based on limited information. This game is not easy for computers either. For a comparison, classical chess has approximately 10^{43} possible states, while RBC has 10^{139} possible states, which indicates that the game can be 10^{96} times more complex (Markowitz et al., 2019).

There has been a substantial amount of research focused on perfect information games, particularly chess, which is well-documented in the literature (Apt and Simon, 2021; Silver et al., 2017). However, the study of imperfect information games, such as Reconnaissance Blind Chess, remains relatively unexplored. The will to fill the gap in our understanding forms the basis of the following research questions: **RQ1:** How can better knowledge modeling benefit the estimation of an opponent's knowledge, which results in enhanced performance? **RQ2:** What are the most effective sensing strategies to diminish the game's inherent uncertainty? Finally, with the understanding of uncertainty in knowledge acquisition, our third research question is **RQ3:** What is the most efficient moving strategy?

Our contributions are the following: (1) We present a comprehensive analysis of the game, discussing potential ways to model player knowledge and manage the expansive state space, amidst prevalent uncertainties. (2) We investigate various sensing and moving strategies and evaluate their respective impacts on agent performance. (3) Based on our analytical results, we choose the best strategies for our agent, Scorca[3], and publish it as an open source project[4]. The agent ranks second on the global leaderboard[5] at the time of submission of the article (23/10/2023).

The paper is organized as follows. Section 2 provides recent research on the game. Section 3 explains knowledge modelling in the RBC game. Section 4 and 5 discuss different sensing and moving strategies, respectively. Since the rationale for the design of our *Scorca* agent is explained in Section 3, 4, and 5 along with their analysis, we include only a summary of the strategies chosen with references to the sections in which the corresponding decisions were made in Section 6. The evaluation is included in 7, followed by the discussion in Section 8 and conclusion and future work in Section 9.

2 Related Work

A rich landscape of RBC agents have emerged, each with its uniquestrategy and design during the NeurIPS tournaments held in 2019 (Gardner et al.,), 2021 (Perrotta et al., 2019), and 2022 (Gardner et al., 2022). The strategies, algorithms, and methodologies adopted by these agents have influenced our research. These agents provide references and benchmarks to our approach. The following paragraphs give a summary of these agents and their respective mechanisms based on the description in the Tournament reports.

StrangeFish[6] maintains an exhaustive set of possible board states, expanding and filtering based on game events and private observations. It chooses sensing actions to maximize the expected impact on its next move selection. Its moving strategy evaluates options across the set of boards using a weighted combination of best-case, worst-case, and average outcome scores calculated with Stockfish[7] (a widely used chess engine) and RBC-specific heuristics. This approach of tracking uncertainty while leveraging chess knowledge allowed StrangeFish to win the inaugural NeurIPS RBC tournament in 2019 (Gardner et al.,).

StrangeFish2[8], the successor to *StrangeFish*, retains its predecessor's framework for tracking board states but adds several enhancements. It chooses sensing actions by estimating outcome probabilities for hypothetical moves after each possible observation. This results in selecting the sense with the greatest expected value based on potential move outcomes. StrangeFish2 also improves efficiency through parallelization and improved search algorithms.

The *Fianchetto* agent[9] (Taufeeque et al., 2022) is

[3]The performance and game records can be found here: https://rbc.jhuapl.edu/users/48973

[4]The code is available on GitHub at https://github.com/Robinbux/Scorca with DOI 10.5281/zenodo.10412786. All the data and supplementary material are on Zenodo with DOI 10.5281/zenodo.10412840.

[5]https://rbc.jhuapl.edu/

[6]https://rbc.jhuapl.edu/users/713 with code at https://github.com/ginop/reconchess-strangefish.

[7]https://stockfishchess.org/

[8]https://rbc.jhuapl.edu/users/1987

[9]https://rbc.jhuapl.edu/users/12368

a derivative of *StrangeFish*'s source code with significant modifications to the original. It replaced Stockfish with the Leela Chess Zero engine (see Section 5.2 for more details) for board evaluation, significantly accelerating the evaluation process. Fianchetto scores all potential moves simultaneously, leveraging the capabilities of a neural network to perform a forward pass. This amplified efficiency provides the bandwidth to expand the search tree, incorporating a probabilistic model that simulates the opponent's likely moves. The agent continuously updates its belief system of plausible board states at each turn, employing the POMDP (Partially observable Markov decision process) belief update equation, creating a dynamic environment of constant recalibration and adaption (Taufeeque et al., 2022).

Penumbra[10], grounded in the principles of deep synoptic Monte Carlo planning, stands out due to its ability to maintain a record of all conceivable opponent board states (Clark, 2021). The agent maintains a belief state using an unweighted particle filter and plans its strategy using an upper confidence bound tree search. Its algorithm uses stochastic abstraction to create approximations of information states, encoding sets of boards with compact, fixed-size synopses. This synthesis of data feeds into a deep neural network that informs and guides the search algorithm, thereby contributing to *Penumbra*'s unique approach (Clark, 2021).

The *Kevin*[11] agent maintains a record of all possible board states and their associated probability distribution. When selecting sensing actions, it strives to minimize the expected difference between its next move and the optimal move. In addition, it uses a depth-limited counterfactual regret minimization (CFR), which approximates an opponent's Nash equilibrium strategy.

The *Oracle*[12] agent employs a meticulous approach to tracking board states. The agent's methodology involves selecting sensing actions to either identify potential checks or minimize the expected number of possible board states. Subsequently, it selects the move most commonly recommended by Stockfish across all plausible board states.

The *JKU-CODA*[13] agent adapt the early AlphaGo framework (Bertram et al., 2022). It first obtains a neural network by supervised training, which was then improved by playing against itself. To tackle imperfect information, the agent used the history of observations as input and avoid any attempt to guess

Agent	Tracks all board states	Tracks opponent infosets	Tracks full game state	Models opponent moves	Models opponent sense	Uses a chess engine	Senses to minimize states	Elo rank	Elo
StrangeFish2	•					•		1	1762
Fianchetto	•			•		•		2	1644
Kevin	•	•		•	•	•		3	1623
penumbra/Châteaux	•	•		•	•			4	1621
Oracle	•					•	•	6	1465
trout						•		11	1116
attacker								12	1099
random								15	893
Scorca	•	•		•	○	•	•	-	-

Table 1: Overview of selected agent capabilities and performance in the NeurIPS Reconnaissance Blind Chess competition in 2022 (Gardner et al., 2022). Scorca is the custom agent proposed in this paper and explained in later sections. In one version, but not the final one, Scorca models the opponent's senses, hence the outlined dot.

or directly reconstruct the unknown full game state. Furthermore, instead of searching for the best move, it only uses the trained policy network to play.

Some other agents are used as baselines or for comparison. The *random* agent takes a random action at each round. The *attacker* agent has nothing but a set of pre-designed strategies that aggressively attack the king of the opponent. Finally, the *trout* agent chooses the move recommended by Stockfish based on a single board state estimate. It senses the location where the capture just happened, or a capture could happen in the next turn. Otherwise, a random location is chosen for sensing, where none of its own pieces is present.

A comparative overview of the performance of these agents and their respective mechanisms, is available in Table 1 (extracted from (Gardner et al., 2022)). It's interesting to note that agents with more complexity and features often underperform compared to their simpler counterparts. Furthermore, of the agents that demonstrated higher performance, only a few were specifically trained for RBC without relying on preexisting chess engines.

It is important to note that in many cases only short text descriptions of the agents are publicly available. The absence of research papers or public repositories for most agents limits the potential for a more comprehensive evaluation. Some agents are not open source but developers can play against them on the RBC platform, e.g. *Châteaux*.

A vital aspect evaluated for all agents in the NeurIPS 2022 competition is the management of game states, depicted in Table 2. Two key columns of interest here are 'median states' and 'engine move agreement.' 'Median states' refers to the median number of potential board states that an agent deems plausible at any given moment. A noticeable trend

[10]https://rbc.jhuapl.edu/simple/users/936
[11]https://rbc.jhuapl.edu/simple/users/8822
[12]https://rbc.jhuapl.edu/simple/users/2
[13]https://rbc.jhuapl.edu/users/15730

Agent	Median # States	# States Rank	# States Δ Rank	Engine Move Agreement	Move Agree. Rank	Move Agree. Δ Rank	Elo rank	Elo
StrangeFish2	32	10	+9	62%	6	+5	1	1762
Fianchetto	20	4	+2	69%	3	+1	2	1644
Kevin	16	2	-1	74%	1	-2	3	1623
penumbra/Châteaux	21	5	+1	53%	9	+5	4	1621
Oracle	16	2	-4	67%	4	-2	6	1465
trout	214	13	+2	44%	10	-1	11	1116
attacker	11356	15	+3	5%	15	+3	12	1099
random	2276	14	+1	8%	14	-1	15	893

Table 2: Comparison of state management of the different agents in the NeurIPS 2022 competition (Gardner et al., 2022).

can be observed: agents with a lower median state number tend to demonstrate superior performance. This observation could indicate some potential correlation between the simplification of an agent's state space and its success in the game. The 'engine move agreement' is a measure of the degree to which the agent's chosen moves align with the top three suggested moves given the current actual board state. Consistently, a lower median state count correlates with a higher engine move agreement, indicating that a streamlined perception of the game state often aligns more closely with the suggested best moves.

However, there are intriguing exceptions to these observations. For instance, *Penumbra*, despite scoring only 43% on the engine move agreement column, outperformed *Kevin* and *Oracle*. This lower score could be attributed to *Penumbra*'s unique approach that does not rely on a traditional chess engine. Despite having higher engine move agreement scores and better uncertainty management, *Kevin* and *Oracle* were not able to outperform *Penumbra*. This suggests that specifically tailored training for RBC, rather than solely relying on pre-existing knowledge and algorithms from classical chess, can yield significant advantages. Thus, the incorporation of RBC-specific strategies and mechanisms might be a promising avenue to explore in future work.

3 Knowledge Modeling in the RBC Game

The main difference between RBC and classical chess lies in the uncertainty of information within the game. A key knowledge in RBC is the information that a player has about its opponent. At each round, there are three distinct moments when a player receives new information about the game state: the sensing result (the absolute information of the nine sensed squared), move result (information by the game, on the actual taken move by the piece and whether a capture happened or not), and opponent move result (information by the game, whether a piece was captured by the op). Following the receipt of these notifications, a player can update their knowledge according to the new information. Specifically, we can track the knowledge of a player exhaustively by listing all possible game states and expanding or reducing this list according to the new information. At each round, in the game, the agent maintains a set of possible states as its knowledge. For a transition to the next round, we expand each state and keep only those expanded states that align with the opponent's move results.

Possible States update flow

1. **Set of possible states:** Let S_t denote the set of all possible states at time t.
2. **Update after opponent's move:** After getting notified by the game about the opponent's move, the set of possible states expands to a new set S_{t+1}.
3. **Sensing:** We then scan a 3x3 area. Let R be the 3x3 sensed area. The updated set of states, after sensing, includes only those boards $B \in S_{t+1}$ where the squares in the sensed area of B match the sensed area R:

$$S'_{t+1} = \{B \in S_{t+1} \mid B[i,j] = R[i,j],\ \forall (i,j) \in R\} \quad (1)$$

4. **Update after agents' own move:** The agent then decides on a move and makes it. Afterwards, informed about the result of the move is obtained, which leads to a new set of possible states, S_{t+2}.

This process repeats until an agent wins or goes timeout.

3.1 Knowledge modeling for sensing

The most crucial information the agents receive comes after the sensing action. After sensing the 3x3 square on the board, and receiving complete insight for the nine individual squares with their pieces on them, the agents can remove all currently possible states that are not consistent with the sensing result. This is done by removing states that do not have the same pieces on the same squares as the sensing result.

3.2 Knowledge modeling for moving

After the agent has decided on a move, the platform informs the agent about the result of the move. This result includes whether the move was successful, whether a piece was captured, and if so, on which

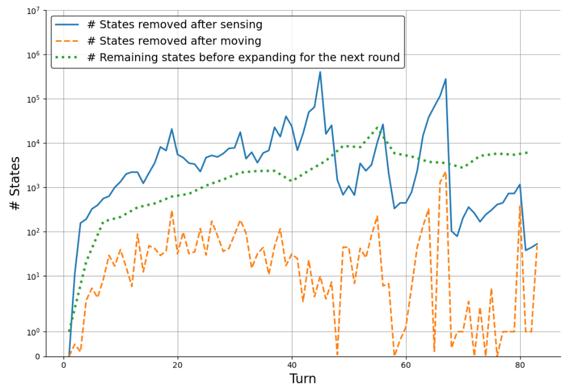

Figure 2: Number of removed states after sensing, moving compared against the number of remaining states. Omitted in this plot are the states added by expanding, after the opponents turn.

square. The type of the captured piece however is not revealed. If a capture happened at a location different from the intended destination, the agent's piece stays at the capture square (Figure 1). This new information allows further refinement of the agent's knowledge, as game states that conflict with the move results can be excluded. For example, if a piece was captured, all game states that do not contain a piece on the capture square are discarded. Similarly, if a sliding move was successful, all boards with squares with a piece between the original and destination squares are purged from consideration.

In certain instances, there can be overlapping information obtained from the sensing and moving phases, such as when the 3x3 region sensed overlaps with the trajectory of the agent's move. To study RQ1, we evaluated the knowledge obtained from different actions and observations by measuring the number of moved states. We compared the gain in information by examining the number of states removed due to sensing and moving. As Figure 2 indicates, sensing can remove a significantly greater amount of states than moving[14]. This is because sensing provides complete knowledge of nine squares (including the type and location of pieces), while moving can provide at most partial information about seven squares (either empty squares or an unknown piece type). Thus, information gain relies on a good sensing strategy.

[14]To standardize the metrics, we utilized the naive entropy sensing mechanism within the game evaluations. We replicated agent movements from a corpus of 500 games, sourced from the publicly accessible archives at https://rbc.jhuapl.edu/about. Subsequently, we replaced the original sensing method with our naive entropy sensing and made measurements throughout the games.

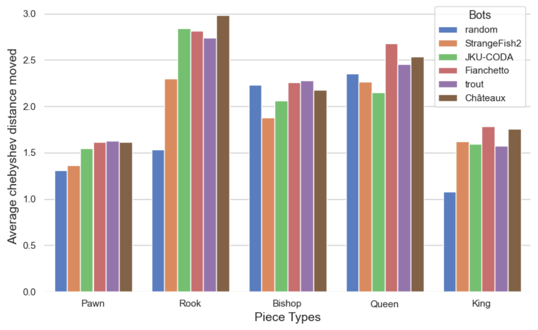

Figure 3: Average distance moved in one turn per piece type for different agents. Data taken from the NeurIPS 2022 RBC tournament.

3.3 Insights with Theory of Mind

The information provided after an opponent's move is limited to whether one of the agent's own pieces was captured and, if so, the location of the square on which this occurred. Armed with this information, we consider all new possible states, based on all the moves the opponent could have executed in all possible positions. In this paper, however, we do not assess the ability to mimic the opponent's moving strategy, owing to its complexity.

In Reconnaissance Blind Chess (RBC), understanding hidden information and effectively using this knowledge are crucial. A valuable but sometimes neglected aspect is analyzing the movement characteristics of pieces. By examining piece movements, players can guess possible strategies and measure the aggressiveness of their opponents. Figure 3 provides an insightful view of this concept by showing the average distance moved by different chess pieces over multiple turns. This data has been gathered from various agents that participated in the NeurIPS 2022 competition. Understanding the distances moved by pieces can be informative. For example, a tendency for long sliding movements with pieces like rooks or bishops might indicate an aggressive strategy, as it can be used to obtain more information but can also lead to a greater risk. On the other hand, shorter movements may suggest a more defensive approach.

By analyzing the data in Figure 3, one can see patterns among the agents. Some agents might often use pieces like queens and bishops for longer moves, indicating an aggressive strategy. On the contrary, some others show a preference for shorter moves with certain pieces, hinting at a more defensive approach such as StrangeFish2. Moreover, the insights gained from these movement patterns can be vital in predicting an opponent's moves and understanding the opponent's strategy, especially in RBC, where information is partially hidden. By incorporating an analytical approach to piece movements, players can make more informed decisions and adjust their strategies. However, a complete analysis of the opponent's strategy and knowledge using ToM requires replaying the game at every round before sensing (with knowledge about the opponent's knowledge updated), which can be computationally expensive and not feasible in real games with the time limit and RAM constraint.

4 Sensing Strategy

As mentioned above, sensing is the main source of information gain, hence having a good sensing strategy in RBC is crucial for an agent. Balancing the strategy between sensing the location of the opponent's pieces and the agent's planned moves can be difficult. Next, we analyze different sensing strategies and tactics and compare their effectiveness in reducing uncertainty.

4.1 Entropy sensing

A naive way of handling sensing to remove uncertainty is to sense the 3x3 area with the most unknown information. This can be materialized as the highest combined Shannon entropy:

$$H(s) = -\sum_{i=1}^{13} p_i(s) \log_2 p_i(s)$$

In this formula, $H(s)$ represents the entropy of a specific square s on the board. The term $p_i(s)$ is the probability of the i-th piece being present at square s, with i ranging from 1 to 13 to cover all possible piece types (six per color), including the absence of a piece. The probability here is naively defined as the percentage of boards in which the piece is present at each square divided by all possible board states. The overall entropy of the 3x3 area, $H(A)$, where A is a 3x3 region of the board, is then the sum of the entropies of its individual squares. The agent's goal is to select the 3x3 area with the highest overall entropy for the next sensing action, as this area represents the region of maximum uncertainty and thus has the potential for the greatest information gain. Agents using this entropy-based sensing strategy would always choose to sense areas with the most amount of missing information. An example of how the entropy map after white's first move can be seen in Figure 4.

Figure 5 illustrates the average number of remaining states using the naive entropy sensing strategy compared against the history of the senses made from 500 games from he NeurIPS 2022 tournament. For the comparison, we replayed each game twice, once

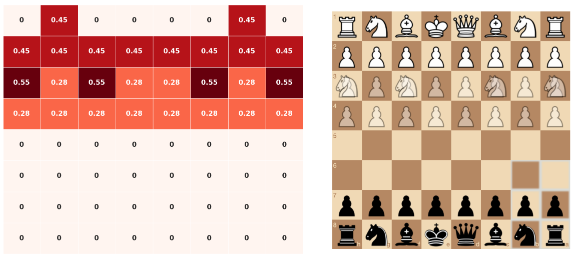
Figure 4: Example of how the entropy map looks from black's view. after white makes their first move. All fields on which the piece can't move on the first turn has an entropy of 0, as all 20 expanded possible states agree with the figures on these squares. The highest entropies can be observed on the fields that can be reached by the knights, as two pieces can be on those fields. Using the naive entropy sense, black would here scan b3 or g3, as the surrounding 3x3 squares have the highest combined entropy.

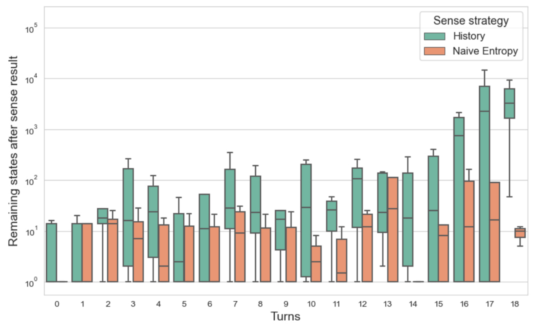

Figure 5: Comparison of the average number of remaining states after the sense action for corresponding turn.

using the historic sense that was actually used in the game, and once using the naive entropy sense. We then calculated the remaining states. It is obvious that the naive entropy sensing strategy outperforms agents' average sensing strategy by a large margin[15]. Another interesting observation is, that the naive entropy leads to a stable amount of states even as the game progresses, while the historic senses lead to a higher amount of states in the later game.

4.2 Heuristics

In a game with equally probable move actions, a naive entropy sensing strategy optimally reduces uncertainty. However, in real scenarios, move distribu-

[15]Note that these games also include agents with "weake" sensing strategies, such as sensing the last square where a capture happened, or estimating where the opponent's king is based on the last known position. Moreover, some agents' strategies do not aim to minimize the number of possible states, but rather try to minimize the uncertainty in a specific area, e.g. the agent's piece to move in.

tions aren't uniform. Thus, enhancing the naive entropy sensing strategy with heuristics offers improvements.

Piece value weighting We introduce weights for the six chess piece types, asserting that a piece's value correlates with the importance of knowing its location. For instance, discerning an opponent's king location is more critical than a pawn's. Piece values are commonly expressed in *Centipawn* values. Assignments are: Pawn: 100, Knight: 290, Bishop: 310, Rook: 500, Queen: 900, and King: 3000. Although a king's capture isn't feasible in classical chess (its value could be infinite), we assign it 3000 centipawns to avoid a singular focus on the king. These values, when multiplied by the probability of a piece's presence and the square's entropy, yield weighted entropy sensing.

King safety Factoring in king safety is another heuristic. This involves assigning a bonus to squares threatening our king and adjusting the square's entropy. The underlying logic is to mitigate surprise attacks. In Figure 6, a knight's move to d6 seems disadvantageous in classical chess, but in RBC, it could set a trap if the black player is unaware. Similarly, squares threatening the opponent's king get a higher value, allowing us to potentially set traps.

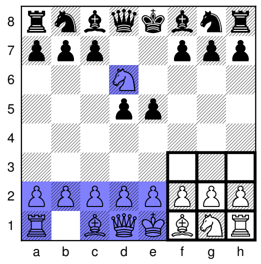

Figure 6: The board depicts both actual positions and the black player's awareness. Blue denotes black's knowledge gaps. After sensing around g2, black's awareness is limited, potentially overlooking threats.

Time penalty Finally, we implemented a time penalty for previously sensed squares. For each square, we track the last sensing turn, then calculate a penalty, $P = d^t \times pv$, where d (set to 0.9) is the decay factor, t is turns since the last scan, and pv (set to 1000) is the penalty value. This penalty is subtracted from the square's entropy, discouraging repetitive sensing of recently checked regions.

Combining these heuristics results in the *Adapted entropy sense*. Figure 7 compares it to the *Naive entropy sense*. By replaying the same game histories and applying different sensing algorithms, the adapted version showed significant superiority, underscoring the heuristic's effectiveness.

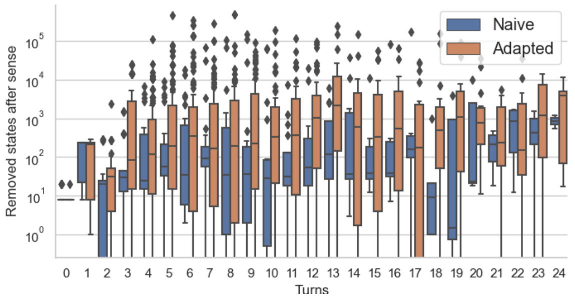

Figure 7: Logarithmic comparison between the naive and heuristically-adapted entropy senses.

4.3 Opponents predicted moves

Opponents move weights Another strategy for sensing revolves around factoring in our knowledge about the opponent. For that, using a method explained in Section 5, for all possible boards we assign scores to all possible moves. We save a combination of these scores for all moves in a dictionary. The same way as done with the entropy, we create an 8x8 matrix representing the board. Here, we add the score assigned for each move to the to square of the move, and the from square of the move. I.e. when moving the rook from 'A1' to 'A4', and the score has a value of 10, we add 10 to the squares 'A1' and 'A4' in the matrix.
Next, in the same way as done with the entropy, we search for the 3x3 square with the highest combined sum of the individual squares in it. The intuition for this strategy is, that we anticipate the opponent's moves and preemptively prepare our strategies. Assigning a score to each potential move facilitates the prediction of the opponent's next move. This is predicated on the assumption that an opponent is more likely to move pieces to squares with higher scores. The accumulation of scores in the squares 'A1' and 'A4' essentially registers the potential for this rook move to occur.

The 8x8 matrix acts as a sensory map of the game state, with higher scores indicating areas of the board that are likely to see more activity. This map provides a strategic blueprint for agents anticipating the opponent's moves and planning our own. It underpins the notion that knowledge of the opponent's likely moves can be as valuable as knowing the current state of the game.

Likely board states With a generally similar idea, we can also use the calculated best opponent moves for all states. As mentioned in Section 3, all possible board states are expanded into all next possible board states, based on all pseudo-legal moves for each state. Now with the move scores for all moves in each state, we are able to expand into all **possible** states, and also note all *n* **likely** states, based on the best *n* moves for each state. The likely states can then be used for further calculations instead of all states.. The intuition here is, that there is little incentive to try to calculate the entropy regarding states, that are very unlikely to the truth.

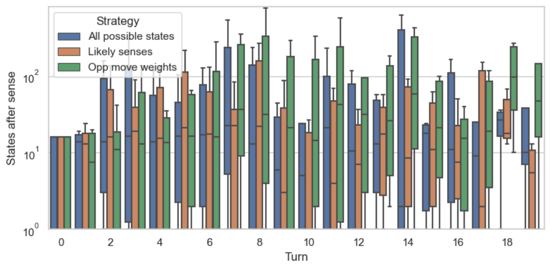

Figure 8: Comparison of the three different sensing strategies in a logarithmic scale. All data is based on the same history games and the value indicates the mean of amount of states **after** the sense action.

In Figure 8 we can see the comparison of the three different sensing strategies applied on the same history games. It becomes clear, that sensing purely based on likely states opposed to all possible states performs way better. An interesting obervation is, that the sensing based on the opponents move weights by far performs the worst, with on average roughly 10x more states compared to the normal entropy sense, and 100x more states compared to the likely states entropy sense.

5 Moving Strategies

The moving strategy complements the sensing strategy. Together, they are the core of the design of an agent. In this section, we analyze different move strategies and tactics and compare their effectiveness in winning games. Also, we will come back to **RQ1** and **RQ3**, and analyze if and how modelling of opponent's knowledge can be used to improve the agents.

5.1 Classical chess engine

To answer the first research question, we first need to establish a baseline for the move strategy. For that, we use the classical chess engine *Sunfish* (Ahle, 2022). Sunfish is a simple chess engine, written in Python. It uses the Minimax algorithm with alpha-beta pruning and a simple evaluation function to determine the best moves given a board state. The evaluation function is based on the material value of the pieces on the board, and the positional value of the pieces. The decision to pick Sunfish over more established and clasically stronger engines like Stockfish was prompted by Sunfish's advantageous lightweight architecture which, in contrast to more powerful engines such as Stockfish, allowed for greater flexibility in manipulating the source code to accommodate our specific requirements. Notably, we introduced modifications such as enabling the ability to castle, traversing through checks, and altering the termination condition of the underlying minimax algorithm from a checkmate state to the explicit capture of the king.

5.1.1 Baseline move strategy

The chess move strategy utilizes a combination of advanced algorithms and heuristics to select the optimal move in a given position. The strategy is primarily centered around evaluation functions, probabilistic scores, and exploitation of an opening book to obtain an advantageous position over the opponent.

Opening Book Utilization To capitalize on well-established initial moves, an opening book is employed in the early phase of the game. Opening books are databases of opening moves that have been played in countless games across history. By utilizing an opening book, the strategy can bypass the need for complex computations in the initial stages and save resources for later stages of the game. To handle the uncertainty, we check for all currently possible game states, for which opening move responses exist. The one picked most often, is the one played by the baseline agent. If no board state is in the opening phase anymore, we continue with the normal move strategy.

Evaluation of Board States Once the game has progressed beyond the opening phase, the engine proceeds to analyze the various potential board states. The number of board states can be extensive, so a sampling technique is employed to select a random subset of board states to analyze, if the amount of possible boards exceeds a threshold. This reduces the computational burden while still providing a representative set of positions for evaluation.

Each board state is assigned a score based on various chess-specific heuristics and the chess engine's evaluation function. Notably, the evaluation considers whether a move puts the king in check and whether it is a capturing move. Based on the scores, the top and bottom few boards are selected for further analysis. This is premised on the assumption that moves leading to the highest and lowest scores are the most impactful as in the best boards the agent might have a chance to win and in the lowest scored boards the agent might need to avoid losing.

Processing Selected Board States The highest and lower *n* ranked board states are considered for this step. For each of these boards, all possible moves are executed. If a move is not legal on a certain board, i.e. a sliding move through an opponents piece, it will count towards the next legal move. For all moves on a specific board, a second evaluation of the board it done after the move execution, and the difference between the post move and pre move board score is added to a dictionary, with the move as the key. This difference is multiplied with the difference of the win probability and 0.5, again to assign a higher significance to very good and very bad board states. Additionally, a longer search time is allocated to more extreme boards for the Sunfish engine to allow for deeper searches in extreme positions.

Move Selection Finally, the strategy selects the move with the highest evaluation as the optimal move. This move is chosen because it maximizes the position's strength based on the employed heuristics and evaluation functions. The selected move is then played on the board.

5.1.2 Baseline move strategy with estimated opponents knowledge

In addition to the baseline move strategy, we also implemented a version, in which we estimate the opponents knowledge, in order to investigate **RQ1** in more depth. For that, we try to not simply score all possible states with our chess engine and then procede with the moves evaluations, but rather first try to estimate what the opponent knows about the current true board state, for each of the possible boards. For this, we implemented a graph structure that represents our knowledge for the whole game, with the head node being the initial board state. From every node, the graph is then extended with all possible moves the opponent could do in position. During the game, certain leaf nodes are deleted based on the sensing actions, move actions and opponents' move actions.

The difference to the baseline agent comes in the following board evaluation step. For every possible leaf node, we replay the game entirely from the opponents point of view. With the graph strucutre, we know for every leaf node which moves the opponent must have made to end in a specific node. As we don't know the senses of the opponent, we assume a naive entropy sense. Subsequently, we average the scores of all possible board states from the opponent's perspective and integrate these into a composite ranking that encapsulates our evaluation and the opponent's potential assessment. In this version, the new evaluations for the board states replace the simpler ones from the baseline agent. The rest of the move strategy stays the same.

It is important to note that this evaluation of replaying for all leaf nodes is very computationally expensive as it requires the game to be played from the beginning with all the states tracked. This is especially the case given that this game can face a high number of possible board states at each step. To have this version of the agent play offline, we needed to stop the game timer for this part of the calculation. Hence, the results of this agent design compared to the others can be seen more as a proof-of-concept, rather than something that can be utilized in its current form in real games.

5.2 Utilizing the Leela Chess Zero Engine

To have a deeper look into **RQ3** (what is the most efficient moving strategy), we also implement a strategy, that utilizes the Leela Chess Zero engine. Leela Chess Zero (LCZero or Lc0), an open-source project, is a next-generation chess engine. Unlike traditional chess engines that rely on hand-designed evaluation functions and complex search algorithms to analyze the billions of possibilities in a position, LCZero utilizes a Machine Learning approach. Neural networks have been trained through Supervised Learning and Reinforcement Learning to precisely estimate the value of a single position and to successfully suggest powerful moves.

5.2.1 Lc0 functionality

The basis of LCZero is a recuperation of the techniques used in Google's AlphaZero project (Haque et al., 2022; Silver et al., 2017). The ground-breaking achievements of AlphaZero set an impressive precedent which LCZero attempts to emulate, albeit without the colossal computational resources available to Google. LCZero employs a neural network combined with a Monte-Carlo Tree Search (MCTS). MCTS is a statistical search algorithm where the focus of the search deepens on more promising areas, this technique starkly contrasts the Minimax search trees implemented by other chess engines such as Stockfish. The utilization of MCTS leads to the evaluation of fewer nodes in the game tree but these nodes are evaluated more accurately due to the neural network's superior ability to generalize from its extensive self-play training regimen. One of the big advantages of LCZero is that though it's neural network, a single board evaluation can provide agents, a score for the board as well as scores for all possible moves. Especially in a game like Reconnaissance Blind Chess, this can be a huge advantage if there are hundrets or thousands of possible board states to be evaluated.

Given the efficiency that LCZero provides in evaluating board states, we are not forced to pick just the best and worst *n* boards, but rather can evaluate all possible boards. Subsequently, for each individual board, we utilize the Leela chess zero network to give an evaluation of all possible moves on that board. Those moves are ranked best to worst and, same as with the baseline agent, stored in a dictionary. As the move scores that LCZero provides are not in centipawns, and in general rather arbitrary depending on the board, instead of working with specific scores for each move we work with a set minus that every move gets depending on their index in the ranking.

$$pv = 4 \cdot \log(idx + 1)$$

Here, *pv* stands for the penalty value received by the move, based on the index. As the order of the first *n* moves is much more significant than the order of the last n moves, a logarithmic function is used to calculate said penalty values. To account for the specifics of RBC, some new scoring heuristics needed to be implemented as well. For example, trying to capture the king while leaving the agent's king in check is legal in RBC, but not the case in classical chess. Similarly, trying to execute an illegal move for a pawn in classical chess can be a valid option in RBC. For example, in the case where a pawn is trying to capture a piece at a given location where nothing is placed. Such special cases are not in Lc0 by design and thus handled as exceptions when adapted for use in our agent.

6 Agent Engineering

Given that the rationale for the choice of strategies of our agent *Scorca* is scattered in Section 3, 4, and 5, here we provide a summary. For the sensing strategy, we chose the adapted entropy sensing with likely board state filtering, based on the evaluations in Section 3 and 4. This combination provided superior uncertainty reduction capabilities. For the moving strategy, we utilized the Leela Chess Zero engine, as discussed in Section 5. The neural network approach demonstrated significant advantages in efficiently evaluating numerous board positions and moves. This efficiency is critical for navigating the expansive state space of RBC. We chose this pairing for the complementary uncertainty management and tactical advantages that were showcased during the testing. Parameters and hyperparameters were fine-tuned after testing various combinations of strategy through self-play.

	Scorca		Trout	StrangeFish2
	(S)	(S+ToM)		
Scorca (S)	-	-	60%	-
Scorca(S+ToM)	70%	-	70%	-
Scorca(L)	100%	100%	90%	70%

Table 3: Comparison of performance of the different move strategies. Winrate to be read as row agent won n% of games against column agent. 10 games were conducted per matchup.

The offline evaluation results were obtained using a laptop equipped with an Apple M1 Max chip. This chip comprises a 10-Core CPU (base clock speed of 3.2 GHz), a 32-Core GPU, and 32GB of RAM in total. All *Scorca* (v36) [3] games were completed by interacting with the server at Johns Hopkins University between 27/09/2023 and 05/10/2023.

7 Evaluation

To compare the efficiency of the different move strategies, we had them play against each other and one of the provided baseline agents, *Trout*. As can be seem in Table 3, the move strategy utilizing Leela chess Zero heavily outperforms the other strategies. The resons for that could be a general superiority against the Sunfish engine, even with manual tweaks, but also the speed that LC0 provides. It can be seen, however, that adding the estimation of the opponent's knowledge to the Sunfish baseline agent can add a significant improvement boost to the agent[16]. We took the best agent and also had it compete offline against the current No.1 agent, *StrangeFish2*. In our offline runs, it won 7/10 times against *StrangeFish2*.[17] The latest version of Scorca (v36)[3] at the time of writing has 7 wins out of 11 total matches against *StrangeFish2*, resulting in a winning rate of 64%.

Upon submission (22/10/2023), Scorca ranked second among 101 agents/human players. Despite its

[16]Disregarding the lower efficiency in the calculation of the scores.

[17]All games are attached in the supplementary material.

individual match superiority against the current best agent, it appears that the Elo ranking emphasizes sustained performance over a series of matches, which might be a reason why our agent has not yet ascended to the top. Furthermore, some games were close to a tie with *Châteaux*, which also led to Elo decreases.

8 Discussion

For deep learning-based agents like *Penumbra* and *JKU-CODA*, despite being specialized for RBC, still lose to agents relying on classical chess engines like StrangeFish. One reason may be that current deep learning approaches do not effectively incorporate knowledge modeling into their decision-making, favoring board position evaluation over belief modeling. Combining learned features with explicit belief tracking remains a problem in this game.

Our proposed Scorca agent has a transparent pipeline for modeling knowledge, sensing, and moving. This modular design allows examination of its reasoning and facilitates debugging of component strategies. The pipeline demonstrates the importance of specialized RBC techniques over just relying on classical chess engines and algorithms. As an open source project, Scorca provides a strong baseline for future research to build upon with innovative approaches such as deep reinforcement learning.

Modelling an opponent's knowledge, actions, and potential strategies is a complex endeavor with agents risks and rewards. The difficulty of learning and reasoning about an opponent's strategy should not be underestimated. With a fixed strategy opponent, modeling is feasible through fine-tuning of parameters. But each RBC game has a limited number of steps, making it challenging to estimate strategy parameters, especially for neural network based agents. The search space explodes combinatorially with the number of possible boards and moves.

A key weakness of current agent designs is the inability to handle extremely risky opponent strategies. For instance, an opponent may make unlikely sacrifices that classical chess engines deem poor moves. Specialized training to recognize unconventional offbeat openings could make agents more robust. Overall, continued research is needed to develop agents that surpass current capabilities in knowledge representation, reasoning, and opponent modeling.

9 Conclusion and Future Work

In conclusion, this paper systematically evaluated different knowledge modelling, sensing, and moving strategies. We presented a novel combination of strategies for sensing and moving, and evaluated the corresponding agent. Recall that for RQ1, we studied the modeling of knowledge and its impact on the performance of agents. As shown in a comparison of the different agents, most notably in the comparison of the baseline agent with and without the estimated opponent's knowledge, we can observe that the estimation of the opponent's knowledge can lead to a noticable improvement in the game. This shows a clear advantage of the inclusion of Theories of Mind in the game of Reconnaissance Blind Chess.

As for RQ2, we evaluated different sensing strategies to diminish the game's inherent uncertainty. We were able to show that the naive entropy sense over all possible states provides a strong baseline in terms of removed uncertainty. We then showed that an improved version of it with manually picked heuristics can further significantly improve the performance. If we then combine it with an estimate of which states are the most likely to be in, based on the opponent's estimated moves, we can further reduce the uncertainty roughly by a factor of 10.

For RQ3, based on our understanding of the modelling of knowledge with uncertainty, we studied some efficient move strategies. Analyzing the three different investigated move strategies, we observed that the most promising one is the Neural Network approach utilizing Leela Chess Zero. The reason for that is the combined information about the current board evaluation as well as the evaluation for all different moves, with a single forward pass in the trained network. This speed advantage over classical chess engines have shown an advantage in Reconnaissance Blind Chess, where the uncertainty of the game leads to a significantly higher state-space.

Our proposed Scorca agent employs a transparent pipeline, which could be used for the examination of decisions, and debugging of future strategies. Moreover, as an open-source project, our agent can serve as a baseline for the development of agents in the future. By using a faster chess engine, our agent reacts faster than *StrageFish2*, making it more suitable for testing purposes in the development stage. Our approach could inspire the handling of events with imperfect information and uncertainty handling and inspire new weighting schemes for each piece in the RBC game.

One of the most promising approaches that do not rely on classical chess engines is the approach of the *Penumbra* and *JKU-CODA*. Although they were specifically designed and trained for RBC, they are still behind the leading agent, *StrangeFish2*.

Using a neural-symbolic representation could have great potential. Some recent research proposed a scalable method to approximate the belief structures using recursive deep generative models used for belief modeling and to obtain representations useful to acting in complex tasks (Moreno et al., 2021). The problem of scalability could still remain. Most recently, some research explored the use of large language models (LLMs) for task planning (Singh et al., 2023). Given that LLMs can act fast and further fine-tuned, by feeding recorded games in past tournaments, we could potentially obtain a model for fast estimation of the belief state with flexible strategies proposed for the best next actions to take. This could then be combined with our approach or other existing approaches mentioned in Section 2.

The modeling of an opponent's knowledge, actions, and potential strategies is a complex endeavor with agents' risks and rewards. As evidenced by the baseline agent augmented with opponent knowledge estimation, some performance gains are achievable through opponent modeling. However, the associated computational costs can be prohibitive.

As RBC agents grow stronger, opponent modeling could become more advantageous to exploit subtle weaknesses. But the models themselves would need to become more sophisticated, requiring innovations in belief-state representation. Future RBC research would benefit from a deeper investigation into different methods and architectures for opponent modeling, along with an analysis of when benefits materialize against baselines without modeling. Deep reinforcement learning methods that leverage neural networks to estimate value functions may be a promising approach to handle this complexity. Integrating neural and symbolic techniques may allow agents to reap the rewards of opponent insights without prohibitively high complexity.

The RBC game is a challenging game due to its extremely large state space. None of the existing agent has claimed a successful mechanism to guess the opponent's knowledge and strategy. Modifying the game by reducing the state space and extending the time could be a possible means of triggering future agents to investigate further into approaches mainly focusing on ToM.

Acknowledgments

The authors appreciate the help from Anna-Maja Kazarian, Boy Kouwenberg, Frank van Harmelen, Peter Bloem, and the host of the RBC game.

REFERENCES

Ahle, T. D. (2022). Sunfish chess engine. GitHub repository: https://github.com/thomasahle/sunfish. Date of access: 2023-10-08.

Apt, K. R. and Simon, S. (2021). Well-founded extensive games with perfect information. 335:7–21.

Bertram, T. et al. (2022). Supervised and reinforcement learning from observations in reconnaissance blind chess. In *IEEE Conference on Games, 2022*, pages 608–611. IEEE.

Clark, G. (2021). Deep Synoptic Monte-Carlo Planning in Reconnaissance Blind Chess. In *Advances in Neural Information Processing Systems*, volume 34, pages 4106–4119. Curran Associates, Inc.

Gardner, R. W. et al. The first international competition in machine reconnaissance blind chess. In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, pages 121–130. PMLR. ISSN: 2640-3498.

Gardner, R. W. et al. (2022). The machine reconnaissance blind chess tournament of NeurIPS 2022. In Ciccone, M., Stolovitzky, G., and Albrecht, J., editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 119–132. PMLR.

Haque, R. et al. (2022). On the road to perfection? evaluating leela chess zero against endgame tablebases. In *Advances in Computer Games*, volume 13262, pages 142–152. Springer International Publishing.

Markowitz, J. et al. (2019). On the complexity of reconnaissance blind chess. *ArXiv*, abs/1811.03119.

Moreno, P. et al. (2021). Neural recursive belief states in multi-agent reinforcement learning. *ArXiv*, abs/2102.02274.

Newman, A. J. et al. (2016). Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management. In *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, volume 9842, pages 62–81. SPIE.

Perrotta, G. et al. (2019). The second NeurIPS tournament of reconnaissance blind chess. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, pages 53–65. PMLR. ISSN: 2640-3498.

Silver, D. et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Singh, I. et al. (2023). Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Taufeeque, M. et al. (2022). Fianchetto: Speed, belief, guile, caution to win at reconnaissance blind chess. Bachelor's thesis. https://taufeeque9.github.io/assets/pdf/bachelors_thesis.pdf.