Teaching bot to play Thousand Schnapsen

Andżelika Domańska[1], Maria Ganzha[1][0000−0001−7714−4844], and Marcin Paprzycki[2][0000−0002−8069−2152]

[1] Warsaw University of Technology, Warsaw, Poland
M.Ganzha@mini.pw.edu.pl
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
marcin.paprzycki@ibspan.waw.pl

Abstract. During the recent years, AI found its ways to games. Here, imperfect information games, such as Thousand Schnapsen, bring about major challenges. In this work, rules and characteristics of this game have been described. Next, an overview of existing literature, focusing on similar problems, has been presented, followed by summary of selected methods for finding an optimal strategy along with applied modifications. Finally, results of experiments are discussed.

Keywords: Card games · Thousand Schnapsen · Machine learning.

1 Introduction

Thousand Schnapsen (also known as Russian Schnapsen) is one of the most popular card games in Poland. It is also played in countries such as Russia, Belarus and Ukraine. The goal of the game is to score a total of at least 1000 points. The first player to do this wins. The game can be played by 2, 3 or 4 people. In this work, a 4-player version is considered. Local variants of the game introduce additional rules, e.g. in different regions of Poland, different scoring for "marriages" is applied. Hence, let us summarize rules used in this contribution.

1.1 Game rules and characteristics

Consecutive games are played until one of the players scores 1000 points. Player who collected at least 800 points can get more if and only if (s)he wins an auction. In the 4-player version, the dealer is not an "active player". Still, (s)he can score points (see, **Declaration**). Each game consists of the following 5 stages.

Dealing A standard deck of 24 cards (9 – Ace) is used. Dealer deals 7 cards to each player, the remaining 3 constitute the stock (hidden cards).

Bidding Dealer does not participate in this stage. Bidding starts from the person sitting to his left, who must bid at least 100 points. The next player (to the left) may raise the declared number, by a multiple of 10, or pass. Bidding continues until all but one player have passed. Since the maximum number of points that can be scored in a single game is 360, this is the maximum value that can be declared. A player can score 360 points when (s)he has three strongest marriages (valued at 240 points) and an ace of their suit. With an extremely "bad hands" of all opponents, and their suboptimal game, it is possible to collect all cards (scoring 120 points), for a total of 360 points.

Declaration Next, the stock is revealed and the dealer receives points "found there" (Ace gives 10 points; marriages give values presented in Table 1).

Table 1. Values of marriages

Suit	♠	♣	♢	♡
Value	40	60	80	100

Stock cards are taken by the highest bidder, who gives one card to each opponent (from that game). Before the game starts, (s)he must declare how many points (s)he plans to score (number not lower than that from the bidding).

Gameplay The game consists of placing one card on the pile by successive players. It consists of 8 rounds (one for each card in hand). The winner of the bidding starts the game. Winner of each round is the player who played the highest card. The winner of the round starts the next one. Here, two rules apply.

1. **The same suit** – if possible, player must place a card with the same suit as the first card on the pile.
2. **Card with higher value** – if possible, player must place a card with higher value than the strongest card on the pile.

When the pile is empty, player may check-in (place queen or king of the same suit, provided that (s)he has both). Here, point bonuses are awarded as in Table 1.

Each card has a rank and color. Here, T denotes the "10" card in the standard deck. Note that, T is "stronger" than J, Q and K. Overall, set of cards is defined as $\mathcal{D} = \mathcal{R} \times \mathcal{C}$, where $\mathcal{R} = \{9, J, Q, K, T, A\}$ is set of ranks and $\mathcal{C} = \{C, D, H, S\}$ is set of colors. Each card has "value" depending on rank, defined by function $f : \mathcal{R} \longrightarrow \{0, 2, 3, 4, 10, 11\}$ represented in Table 2.

Table 2. Rank values

r	9	J	Q	K	T	A
$f(r)$	0	2	3	4	10	11

However, the "strongest" card in the pile depends on the current state of the game. Here, relevant are: (1) color of the first card on the pile $F \in \mathcal{C}$, and (2) current marriage color (if any marriage was checked-in) $M \in \mathcal{C} \cup \{\emptyset\}$. Thus, the *contextual value* of the game is defined by $h : \mathcal{D} \longrightarrow \mathbb{R}$. This function must meet the following conditions: (a) card of the same color with a higher value has higher contextual value; (b) card of color, other than the color of the first card, and the current marriage, has a lower contextual value than any card of the color of the first card, or of the current marriage; and (c) any card of color of the first card has a lower value than any card of color of the current marriage (if they differ).

Counting points At the end of the game, players count rank values of collected cards, and bonuses for checked-in marriages. If the winner of the auction has scored at least the number of declared points, this number is added to her total score, otherwise it is subtracted. The remaining players receive the scored points, unless they have exceeded the threshold of 800 points.

Thousand Schnapsen is a **trick-taking** card game. Players try to collect as many opponents' cards as possible. As the opponents' cards are unknown, it is an **imperfect information** game. Moreover, it is **not** a **zero-sum** game. The value of all cards, in a single game, is 120, while the number of checked-in marriages, depends both on the card-split and the implemented strategies. This complicates development of game playing strategies (and training of bots).

Another difficulty is the large number of states. Before the game starts each player has 8 cards (9,464,816,790 initial hands). The bidding winner knows her cards, and one card from each opponent. Thus the number of initial opponents' hands equals 3,432. The remaining players know their cards (they do not know what happened with two cards from the stock). Thus, the number of possible initial hands equals 12,870. Overall, the lower bound on the size of the game tree is 328,801, while the upper bound is 323,593,859,345,481. Obviously, these numbers are recalled only to illustrate the complexity of the game.

2 Related work

Thousand Schnapsen is not a widely known game. Hence, instead, we discuss card games with incomplete information. Here, authors, typically, focus on variants of Poker or Bridge. However, attempts to apply AI in Austrian Schnapsen [10,11], or the Swiss game Jass [4], can be found. In the latter article, overview of methods that can be used for this type of games can be found. Overall, in the literature, algorithms based on expert knowledge (see, [9], [7], and [5,1]), reinforcement learning (e.g. *Temporal Difference Learning*, *Policy Gradient*, *First Order Methods*, *Neural Fictious Self Play* [3], and *Counterfactual Regret Minimization* discussed in Section 2.1), Monte Carlo methods [11], and evolutionary algorithms, are considered. Interested readers should consult references for pertinent details. Upon careful examination of the literature, we have decided to experiment with CFR and its modification that uses neural networks for function approximation – Deep CFR [2] – which showed most promise in similar games.

2.1 Counterfactual Regret Minimization (CFR)

The CFR algorithm was proposed in 2007, for imperfect information games with large state spaces [12]. It uses iterative generation of new strategies (strategy profiles) that should converge to the Nash equilibrium. It's aim is to minimize regret value, introducing a new concept of counterfactual regret. Let u_i be the payoff function for player i, Σ_i set of his strategies, σ^t the strategy profile at time t and σ^t_{-i} strategies of opponents at time t. Then regret can be defined as:

$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} \left(u_i(\sigma_i^*, \sigma^t_{-i}) - u_i(\sigma^t) \right) \tag{1}$$

The concept introduced by the authors of the CFR algorithm is to define regret $R^T_{i,imm}(I)$ separately for each information set $I \in \mathcal{I}$. This approach is appropriate if and only if minimizing $R^T_{i,imm}(I)$ for each $I \in \mathcal{I}$ means minimizing R_i^T. On the basis of the calculated regret, it is possible to determine a new strategy using, for example, *regret matching.* The basic version of CFR requires the value of $R_i^T(I, a)$ to be stored for each possible state $I \in \mathcal{I}$, and action $a \in A(I)$. If $|\mathcal{I}|$ is very large, this is not feasible, due to limited RAM size. Two solutions are then proposed. (1) To use expert knowledge to create a state abstraction using advanced domain knowledge. (2) to use neural networks to approximate the function (see [8]). This modification is called Deep CFR and in application to the Limit Texas Hold'em it outperforms the NFSP [2].

Theoretical convergence of the CFR to the Nash equilibrium is ensured only for 2-player games [2]. Nevertheless, attempts to use it for 3-player games have shown its potential [6]. Even if, despite the lack of theoretical foundations, as a result of the CFR algorithm, a good approximation of the Nash equilibrium is calculated, it does not mean that an optimal strategy is found. Moreover, even finding the exact Nash equilibrium also does not, in theory, guarantee the calculation of the optimal strategy. This shows that games for more than 2 players are problematic for all algorithms based on Nash equilibrium calculation.

3 Applying selected approaches to the game playing

Let us elaborate the scope of our work. First, note that, during the game, decision-making is based on points scored by each player, auction winner and declared points. Using this information, in the representation of the state of the game, is non-trivial, and therefore it has been omitted. It allowed to consider each game completely independently of the previous ones. Finally, the 4-player version does not differs from the 3-player one, in the *game playing phase.* Therefore, in what follows, the 3-player version is considered.

In the game, two problems can be distinguished. (1) Optimal bidding, taking into account the points, cards in hand, and the state of the auction. (2) Finding the optimal strategy for playing cards, based on the history of the game, cards in hand, cards in the pile, and the color of the last checked-in marriage. Since these are two independent problems, the latter is the focus of this work.

3.1 Reasoning

As the game progresses, players reveal information about their cards, through successive moves. Therefore, it is important to efficiently reason, and remember which cards opponents definitely do/do not have. Based on the rules of the game, basic rules for inferring the status of opponents' cards can be defined.

If the opponent checked-in a "marriage" using Q or K, then (s)he is sure to have a second card of the pair. If, as first card on the pile, opponent chose a Q or K and didn't check-in, then (s)he does not to have the second card of the pair. The application of above stated rules, concerning cards that should be added to the pile, allows to determine which cards opponents do not have.

3.2 Payoff function

To implement the game, it is necessary to define the payoff function. Let H be the set of all states, $Z \subset H$ the set of final states, and $\mathcal{P} = \{1, 2, 3\}$ the set of players. Also, let the function $p_i : Z \longrightarrow \mathbb{Z} \cap [0, 360]$ return the number of points earned by i-th player in state $z \in Z$. Simple reward function defined as $u_i(z) = p_i(z)$ doesn't meet fixed-sum condition 2.

$$\forall_{z \in Z} \sum_{p \in \mathcal{P}} u_p(z) = 400 \tag{2}$$

Let $m : \mathcal{C} \longrightarrow \{40, 60, 80, 100\}$ be a function, which returns bonus for a marriage in a given color. Moreover let $C_{\text{unused}} : Z \longrightarrow 2^{\mathcal{C}}$ be a function that returns the set of colors of not checked-in marriages, for the final state. Modified function u_p can be defined as:

$$u_i(z) = p_i(z) + \frac{1}{|\mathcal{P}|} \sum_{c \in C_{\text{unused}}(z)} m(c) \tag{3}$$

Proof that function 3 meets the condition 2 is trivial.

3.3 Implementing CFR

The CFR requires a full traversal of the game tree, which can take a very long time (over 40 minutes on a moderate computer). Therefore, the External-sampling MCCFR variant, which does not require a full traversal, was chosen. Here, full tree traversal is replaced with the Monte Carlo sampling. Here, in the nodes where the movement belongs to the highlighted player, a full traversal is performed, in the remaining nodes a random action is selected. The probability of selection is defined by the current strategy.

Since the number of game states is very large, it was necessary to use abstraction. Moreover, proper encoding of the card set was needed. Since the used deck consists of 24 cards, it can be represented as a 24-bit number. If the i-th bit is set, it means that a card of index i is in the given set. For example set $\{(9, S), (Q, D), (A, H)\}$ is encoded by 8 404 993, with binary representation:

```
 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 | | | | | | | | | | | | | | | | | | | | | | | |
AH                 QD                           9S
```

The proposed representation saves memory, but requires additional computations to encode and decode the cards.

In addition to encoding the cards, reasoning, described in Section 3.1, was used. Ultimately, the state was defined using:

– encoded set of cards that can be used by player,
– encoded set of rest player's cards,
– encoded set of cards possibly owned by opponents,
– encoded sets of cards that are for sure owned by each opponent.

To reduce the state set it is possible to represent similar situations the same. Hence, it was proposed to "unify" the sets of cards by "shifting" the colors. This operation, shown in Example 1, consists of "cutting out" the encoding fragments corresponding to the colors, from which there is no card in any of the given sets, and completing the encoding, on the right, with zeros to the length of 24.

Example 1. There are two sets of card sets:

(a) {(9, S), (Q, D), (A, H)}
{(T, S), (K, D)}

(b) {(9, S), (Q, C), (A, H)}
{(T, S), (K, C)}

These sets are originally encoded as follows:

```
(a) 100000000100000000000001
    000000001000000000010000
```

```
(b) 100000000000000100000001
    000000000000001000010000
```

The code fragments, corresponding to the colors that are empty, in each encoding from a given set, have been marked in blue. After performing the unification, the obtained encodings are as follows:

```
(a) 100000000100000001000000
    000000001000010000000000
```

```
(b) 100000000100000001000000
    000000001000010000000000
```

3.4 Deep CFR

Deep CFR [2] is a variant of CFR, based on neural network function approximation. The method scheme is very similar to that for tabular CFR. Calculation of the current strategy, in a given state, is done using the *advantage network*. During subsequent traversals of the game tree, new training data is obtained for the network and stored in buffer(s) with a limited capacity. These buffers use the *reservoir sampling* method. At the end of each iteration, the *policy network* is also trained, which "stores" the calculated strategy. Both networks may, or may not, have the same architecture.

Encoding a game state, for use in Deep CFR, should meet different requirements than encoding for standard CFR. Use of neural networks eliminates the need to code sets of cards with a single integer. Moreover, unification described in Section 3.3 is not advisable, as this extra work may slow the learning process.

To reduce the number of states, without losing information that can be useful in the decision-making process, is to use inference about opponent's cards (see, Section 3.1). Remembering game history is redundant. Information about the course of the game can be represented by: (a) set of cards that can be owned by both opponents, (b) set of cards that can be owned by the first opponent, and (c) set of cards that can be owned by the second opponent. Moreover, encoding of checked-in marriages can be achieved coding with the set of colors, and the designation of the last checked-in.

Note that, while the first round is started by the winner of the bidding, the next is started by the winner of the previous round. However, based on the information which player starts which turn, it is possible to clearly determine which player will be next (clockwise). This allows to replace the original opponent IDs with IDs in the context of the current sequence. After all simplifications, the full game state consists of: (1) set of player's cards, (2) list of cards in the pile, (3) set of cards that can be owned by both opponents, (4) set of cards that are for sure owned by the first opponent, (5) set of cards that are for sure owned by the second opponent, (6) set of checked-in marriages, (7) trump.

Card-sets were encoded as binary vectors of length 24. Since there can be 0, 1 or 2 cards in the pile, it is encoded by 2 binary vectors (of length 24), corresponding to the first and the second card. When there is no second or, even, first card in the pile, the corresponding vectors are all zeros. Otherwise, in the position corresponding to the index of the card it encodes, there is a one.

Colors of checked-in marriages are encoded by a binary vector of length 4. The trump is also coded by a binary vector of length 4. When no check-in has occurred yet, the vector consists of all zeros. All codes are combined into one binary vector of length 152, summarized in Table 3.

Table 3. Game state encoding

Position	**Data**
0-23	set of player's cards
24-71	list of cards in the pile
72-95	set of cards that can be owned by both opponents
96-119	set of cards that are for sure owned by the first opponent
120-143	set of cards that are for sure owned by the second opponent
144-148	set of checked-in marriages
149-151	trump

In Section 3.2, an approach to modifying the reward function has been proposed so that the game can be classified as a fixed sum game. This solution was also used for the Deep CFR algorithm. Additionally, the standardization of the values expected on the *advantage network* output has been introduced. The payoff values are in the range $[0, 400]$. Their weighted average also is in this range. Due to the regret definition, it is in the range $[-400, 400]$. Standardizing by dividing by 400 reduces this range to $[-1, 1]$.

4 Experimental results

Let us now discuss experimental results obtained for the CFR and the Deep CFR. Obviously, since there are no known results of applying AI to Thousand Schnapsen, we had to implement a "random strategy" player, performance of which will be used as a "baseline". This player selected a random action, from the set of available actions (where probability of choosing each action is equal).

Overall, 10,000 games were played, all using random strategy, to establish performance, depending on players' order in the first round. It showed that the first player wins, on average, 38.48% of the games, the second 29.85%, and the third 31.67%. Hence, the player that starts the game has an advantage. Interestingly, the second best result does not belong to the second player, but to the third one. The reason for this may be that the third player, often, has smaller number of possible actions (due to the re-raise requirement). Therefore the probability of picking the "wrong move" is smaller than for the second player.

Interestingly, the experiment showed that 1,000 games gives a good enough estimate of effectiveness, because afterwards the number of wins for each player is close to that obtained after 10,000 games. This observation was used to determine the number of simulations to be performed to estimate the effectiveness training.

4.1 CFR results

After establishing the baseline, let us now look into results obtained by the training approaches. Despite use of minimalist representation of the game state, the number of possible states remained too large to save them in memory of a computer with 20 GB RAM, and the swap memory limit set at 15 GB. After about 70 million states were saved, training stopped due to lack of memory.

Therefore, an even more limited state representation was tested. It contained: (1) set of cards that can be used by player, (2) set of cards that are for sure owned by the first opponent, (3) set of cards that are for sure owned by the second opponent, and (4) information if the pile is empty. On the one hand, the number of states was still very large, on the other, available information was very limited. This led to situations where two states, in which completely different decisions should be made, were treated the same. The performance of the obtained strategy was exactly the same as random one.

A potential solution to these problems may be to develop a better abstraction of the state, both reducing the number of possible states and the loss of significant information. It could be also possible to use a much more powerful computer. However, the latter was not an objective of the research. It just illustrates one of the important technical issues related to use of the CFR-based approach.

4.2 Deep CFR results

The problems that made use of the tabular CFR unsuccessful, i.e. too many states and too much information limitation, did not occur in the Deep CFR method. Both *advantage network* and *policy network*, are MLP networks consisting of 4 hidden layers with 192, 96, 48 and 24 neurons. The activation function in each of the hidden layers was $TanH$. The remaining parameters were set to: (a) number of traversals per iteration – 10, (b) number of epochs – 30,000, (c) batch size – 100, (d) learning rate – $1e-5$, (e) *advantage network* buffer's size – $1e6$, (f) *policy network* buffer's size – $2e6$. After each learning iteration, the algorithm's effectiveness was evaluated. It consisted in conducting 1,000 games between the Deep CFR agent and the agents implementing the random strategy. The results are presented in Figure 1.

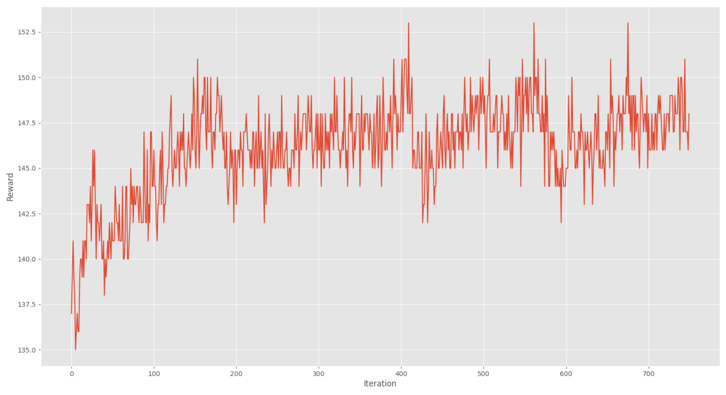

Fig. 1. Deep CFR algorithm performance achieved in the first 750 training iterations

After a total of 750 iterations, the maximum average reward achieved is 153. A trained agent's percentage of success depends on which player plays the card on the first turn. It was calculated on the basis of 10,000 games played against agents implementing a random strategy, for each of the situations. Somewhat similarly to the random case, player that plays the first hand has advantage. Specifically, first player won 51,5% of games, the second 39,4% of games and the third 41,5% games. Here, again, we see that the best "positions" to be in is the first and the last player of the first round.

Overall, trained agents definitely outperform these using random strategy. The greatest "progress" can be observed for the player that starts the game, it is about 13 percentage points.

The results of 10,000 simulations, in which *each* player implements a strategy calculated using the Deep CFR algorithm are similar to the baseline. The first player wins about 40.00% of games, the second one 29.14%, the third one 30.86%.

5 Conclusions

Obtained results differ from other card game focused AI in at least two aspects:

- Thousand Schnapsen is not popular all over the world, so no algorithm has been found in the literature that works at the level of a human expert,
- the considered algorithms, in theory, do not work for more than 2-player games.

Therefore, developing a solution that is more effective than the random strategy can be considered a success.

As part of the project, an abstraction of the game state was also developed, which can undoubtedly be used to implement other algorithms that deal with other card games, not explored in this work. Obviously, the obtained results open the way for further research in multiple directions.

References

1. Bergh, M.V.D., Hommelberg, A., Kosters, W., Spieksma, F.: Aspects of the cooperative card game hanabi. In: BNCAI (2016)
2. Brown, N., Lerer, A., Gross, S., Sandholm, T.: Deep counterfactual regret minimization. In: Proceedings of the 36th International Conference on Machine Learning. pp. 793–802 (2019)
3. Heinrich, J., Silver, D.: Deep reinforcement learning from self-play in imperfect-information games (03 2016)
4. Niklaus, J., Alberti, M., Pondenkandath, V., Ingold, R., Liwicki, M.: Survey of artificial intelligence for card games and its application to the swiss game jass (06 2019)
5. Osawa, H.: Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information. In: Ganzfried, S. (ed.) Computer Poker and Imperfect Information, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 26, 2015 (2015)
6. Risk, N., Szafron, D.: Using counterfactual regret minimization to create competitive multiplayer poker agents. pp. 159–166 (01 2010)
7. Robilliard, D., Fonlupt, C., Teytaud, F.: Monte-carlo tree search for the game of "7 wonders". pp. 64–77 (08 2014)
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018), `http://incompleteideas.net/book/the-book-2nd.html`
9. Ward, C., Cowling, P.: Monte carlo search applied to card selection in magic: The gathering. pp. 9 – 16 (10 2009). https://doi.org/10.1109/CIG.2009.5286501
10. Wisser, F.: Creating possible worlds using sims tables for the imperfect information card game schnapsen. vol. 2, pp. 7 – 10 (11 2010). https://doi.org/10.1109/ICTAI.2010.76
11. Wisser, F.: An expert-level card playing agent based on a variant of perfect information monte carlo sampling. In: Proceedings of the 24th International Conference on Artificial Intelligence. p. 125–131. IJCAI'15, AAAI Press (2015)
12. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Proceedings of the 20th International Conference on Neural Information Processing Systems. p. 1729–1736. NIPS'07, Curran Associates Inc., Red Hook, NY, USA (2007)