# Effect of Safe, Risky and Neutral Strategies on Win Ratios In Games

Raihan Karim Toyesh Chakravorty

Vrije Universiteit Amsterdam

**Abstract.** Knowledge bases make up a big part of Artificial Intelligence, and so do researches concerning it. Games using knowledge bases are very common nowadays. Winning mostly depends on strategies; the better the strategy, the higher the chance of winning. Some strategies are to play it safe, while others focus on going all in for the big reward. Our research focuses on determining which strategy proves to be more advantageous. To perform the research, we make use of the card game *Schnapsen* and use bots as our playing agents.

We consider three basic types of strategies. The first one approaches by taking safe stances, the second moves with risky stances (both of which are simple bots) and the third plays neutrally, that is sometimes safe stances, and the other times risky stances. Comparing their winning percentages will be a good indicator of their effectiveness. Knowledge bases and games have a good connection. They improve the players' performance and so, our neutral bot is based on the use of a knowledge base.

**Keywords:** Knowledge Base · Artificial Intelligence · Intelligent Agents · Schnapsen · Bots · Cheap · Expensive · Neutral · Strategy.

## 1 Introduction

### 1.1 Adversarial Games For Research

Since the early 90s, card and board games has proved significant in the field of computer science. Adversarial games like Chess and Go have helped AI develop a lot in the past few years, and have helped us achieve many milestones throughout the journey. In the 90s, one of the biggest milestones of AI was accomplished. An AI program that could play chess was developed. People believed that chess was so difficult that one needed intelligence to win it. AI researchers had taken it as a goal and in 1997, *Deep Blue* became the first machine (AI) to defeat the (human) Chess World Champion. [1] Developing game-playing programs became a major area in the field of Artificial Intelligence since then. For example, *AlphaGo* which plays Go, a game similar to Chess but having much more possible moves and states, was created. It defeated the Go World Champion in 2017.

### 1.2 Schnapsen For Research

People often have distinct opinions whether games containing chances, like *Backgammon* and *Poker*, require skills or not. Some say games like these don't involve any skills, while others disagree. [2] So, to make an experiment without any bias towards opinions, we chose *Schnapsen*, a game that has a moderately good balance of chance and skill.

Games like GO and Chess have very large state spaces, which makes using them more complex. [3] Schnapsen, on the other hand is quite manageable. Even though it has a big state space of around 10 to the power of 20, it's way smaller than Chess', which has 10 to the power of 46. [4] It also has some really good rules and strategies to be experimented with.

Moreover, Schnapsen has imperfect information in the first phase of the game, as opponent's cards are unknown and unpredictable. This isn't the case for Chess and Go. Hence, Schnapsen is more convenient to use as most real life problems involve imperfect information. Experimenting with it will thus give solutions that are more easily implementable in the real world. [5]

### 1.3 Expectations

We expect the Neutral bot to be far superior than the Cheap bot and the Expensive bot. Comparing these two, we think that the Expensive bot will be the worst of the lot.

## 2 Background Information

### 2.1 Schnapsen and It's Rules

Schnapsen is a trick-taking card game which originated in Austria. It is similar to its German cousin, *66*. The main aim of this game is to get 66 points by winning tricks as quickly as possible. Here, trick refers to one round in the game.

Some basic rules of this game are as follows:

- Only the Jacks, Queens, Kings, Tens, and Aces are used.

- Jacks, Queens, Kings, Tens and Aces are worth 2, 3, 4, 10 and 11 points respectively.

- To win a trick, you must play a more valuable card of the same suit or play a card from the Trump suit. If you don't have a card to win the trick, you must play the least valuable card so as to give the least points to the opponent.

- If you have the Trump Jack, then at the beginning of the trick, you can exchange it for the more valuable trump card that's placed face up under the talon.

- If you have both the King and Queen of a same suit, and you play them at the beginning of a trick, you can claim marriage points (30 points if it's a non-trump marriage, and 40 points if it's a trump one).

- Once the talon (deck of cards) is finished, following suit becomes compulsory. Only if you don't have a card of the same suit can you play a different card (including Trump cards).

- There are 2 ways to win the game:
  - Your overall point is greater than or equal to 66
  - You win the last trick

- After winning the game, the player gets points on the basis of the following:
  - If the opponent has not scored any card points, the player gets 3 game points.
  - If the opponent has scored less than 33 card points, the player gets 2 game points.
  - If the opponent has scored 33 card points or more, the player gets 1 game point.

## 2.2 Schnapsen Strategies

There is no perfect strategy to win a game of Schnapsen. This is because it is an imperfect information game. This means that we do not know what moves the opponent can make and can only guess them.

However, some of the strategies that can be used to play optimally, and win a game if possible, are as follows:

- If you can make a Trump Jack exchange, do it as soon as possible. Having a more valuable trump card is better than having a less valuable one.

- Declaring marriage moves as soon as possible is ideal. This is because keeping the marriage cards in hand and waiting for the perfect moment to play them may cost us the marriage. It is very difficult to maintain the marriage cards in our hand as we can only hold 5 cards at a time. Also, if we lose a trick, our chance to play the marriage card goes away.

- If the opponent plays a card, and you can win the trick using a card of the same suit, do it. It allows you to move first the next turn, which is in turn helpful if you want to play a marriage or do a Trump Jack exchange. Also, it gives you some card points which counts towards the final points.

- Winning a trick with a higher value card of the same suit is more ideal than winning with a less valuable card. This is because you may lose the more valuable card to a trump later on in the game.

- If the opponent plays a less valuable card like Jack, Queen or King, and you can't win without using a trump card, it is best to lose that round and discard the least valuable card (unless you only have Aces and Tens). However, if the opponent's card is an Ace or Ten, it is worthwhile to use your Trump card to win that trick (preferably the cheapest one), as it is very valuable.

- When it is our chance to move first, and the game is still in phase 1, that is the talon is open, it is best to play a cheap card as we do not know what cards the opponent has. For example, if we play a non-trump Ace and the opponent has a trump card, we lose a really valuable card to him.

### 2.3 Bots and Knowledge Base

For our safe and risky bots, the strategy is simply. They choose the cheapest (and hence the safest) or the most expensive (and hence the riskiest) move available to them respectively. The Neutral bot, on the other hand is a bit more complex. It selects the most optimal move rather than playing the cheapest/most expensive card.

As mentioned previously, although not as big as Chess, the state space of Schnapsen is very large and it would take quite long to search for the most optimal move exhaustively. Therefore, by using a knowledge base of strategies, we can make decisions faster and more easily. The agent then chooses the optimal move based on the strategies. [6]

The KB will consist of some basic strategies to play optimally and not loose much points. This also includes the point scoring ones such as marriages[ further details in section **2.2**]. We could have implemented KB in the other two bots as well, but keeping the code simple was key to having a simple strategy of playing cheap/most expensive card first. It also makes the code clearer and easy to interpret.

### 2.4 Previous Researches and Findings

There hasn't been much research done with Schnapsen and it's strategies as it is a relatively smaller challenge compared to Chess and Go. But researchers have tried many different strategies and techniques to improve a performance of intelligent agents playing card games. There has been a good attempt with having a KB of heuristic values found using evaluation function on the states and moves. They also have attempted to use evaluation function using fuzzy values assigned to the game states in the search tree. [6]

A good number of research papers have discussed about the technique of abstracting the game and then using Equilibrium Approximation Affects (EEA) to find strategies which will be then mapped back to the real game. [7] There have also been researches involving Perfect Information Monte Carlo Sampling (PIMC) mainly due to the fact that it has no pre-calculation phase and has previously produced some really competitive game playing AI agents.[8] Although, one paper pointed out it's drawback of naively invading the imperfect information elements of the game tree. But, they also showed how how the performance of PIMC-based players can be improved by training individual card inferences using supervised learning. [8]

## 3 Research Motive

### 3.1 Motivation and Inspiration

Strategies are a very important part of games, especially when designing game playing agents. There has been a lot of work done to find strategies that optimizes the performance of the player as different strategies affect the game-play of the players differently. The type and nature of strategies is also another important factor that affects the performance of the player by easing the problems to different levels. [9] Therefore, we decided to research how different types of strategies affect performance in game.

### 3.2 Research Question and Plan

Now, we come to our research question: *'Which strategy (bot) works the best and results in a better performance in game?'*. By the term '*best*' we mainly refer to the win ratios in the sense that whichever bot wins the most games has the better strategy. However, we slightly refer to the points as well. To answer this, we have the three bots as mentioned before. All of them play optimally in their own sense.

### 3.3 Hypothesis and Prediction

Our hypothesis surrounds the fact that the neutral bot will perform better than other the two custom bots. This is because this bot doesn't have only one strategy, and changes its play style depending on the game's stage. It sometimes plays the cheap cards while the other times, it plays the more expensive cards. Also, it has a KB which allows it to meld marriages and hence, we expect it to perform better.

The other two custom bots should perform worse compared to the Neutral bot. Amongst these two, we expect our Cheap first bot (safe playing) to perform better than the risky bot. This is because as the later plays riskily, it has a higher chance of losing more points than the Cheap first bot.

## 4 Experimental Setup

### 4.1 Experimental Information

In this experiment, we are trying to understand how different strategies and features affect the win rate of a game (Schnapsen in our case). To determine this, we conducted a number of Round Robin tournaments between a number of bots.

To conduct the tournament, we used the tournament.py python code provided to us. Apart from this, we also used the three pre-built bots rand, bully and rdeep provided to us. We compared these to the three bots we created as follows:

### 4.2 Cheap Bot

This bot follows a really simple strategy. It just focuses on playing the cheapest card in its hand first. It can also perform a Trump Jack exchange. The idea of creating this bot was to compare its win rate to the neutral bot's win rate, which is described below. They have one common strategy, and that is playing the cheapest card first. This is a really important strategy in Schnapsen, and we wanted to check how much other strategies/features have an effect on the win rate when added to this strategy.

### 4.3 Expensive Bot

This bot, just like the previous bot, follows a really simple strategy. It just returns the most valuable card in its hand as a move. We also implemented Trump Jack exchange just to make it similar to the Cheapest bot. The idea behind this bot was to check how much effect does playing the cheapest card have on the win rate as compared to playing the most expensive card. In short, we can call the Expensive bot the opposite of the Cheapest bot.

### 4.4 Neutral Bot

This bot, unlike the previous two, has an advanced strategy. It can perform Trump Jack exchange, meld marriages using a KB, decide when to play a cheap card and when to play an expensive one. This is our main bot, which, according to us, plays optimally and has the best strategies. Also, the win rate of this bot as compared to the other ones is extremely important as it falls in the middle ground, and thus paints the perfect picture about how important strategies and features are.

As for the knowledge base, it is a very simple knowledge base which just checks if there is a King and Queen of the same suit available to the bot. If the condition is satisfied, it plays the cheaper of the two, that is the queen, to meld a marriage.

Apart from these three bots, as mentioned above, we also used the three bots given to us. Their strategies are as follows:

### 4.5 Rand

This is the most basic bot of all of the bots. It has no strategy whatsoever, and simply plays a random card from its hand. Winning a game depends purely on luck for this bot.

### 4.6 Bully

This bot is a very basic representation of the Neutral bot that we created. It uses similar features and strategies but falls short when compared to the ones used in the Neutral bot. The main reason for a comparison with bully was to find out how much affect does just a few different features have on the win rate (just like the Cheapest bot).

### 4.7 Rdeep

This is the flagship bot. Comparing our bots with this bot gives us a good idea about how optimally do the bots play. If the number win rate of our bot is greater than rdeep, then it's an extremely good bot which always plays optimally with excellent strategies. If the win rate is a bit lower than rdeep's, it's a good bot with good strategies, and it plays optimally most of the time. However, if the win rate is a lot lesser than rdeep's, the strategy of that bot isn't great, and it does not play optimally.

## 5 Results and Findings

We ran 4 tournaments of 60 games each and one tournament of 150 games. The results are represented in the form of pie charts. Each of our 3 custom bots had a tournament with the bots provided to us. There was one tournament between our three custom bots. Finally, we had an overall tournament between all the 6 bots.

The results of the tournaments conducted were almost as expected. However, there were a few results which were rather interesting. We will go into the findings of each tournament individually.

## 5.1 Neutral vs Pre-Built Bots

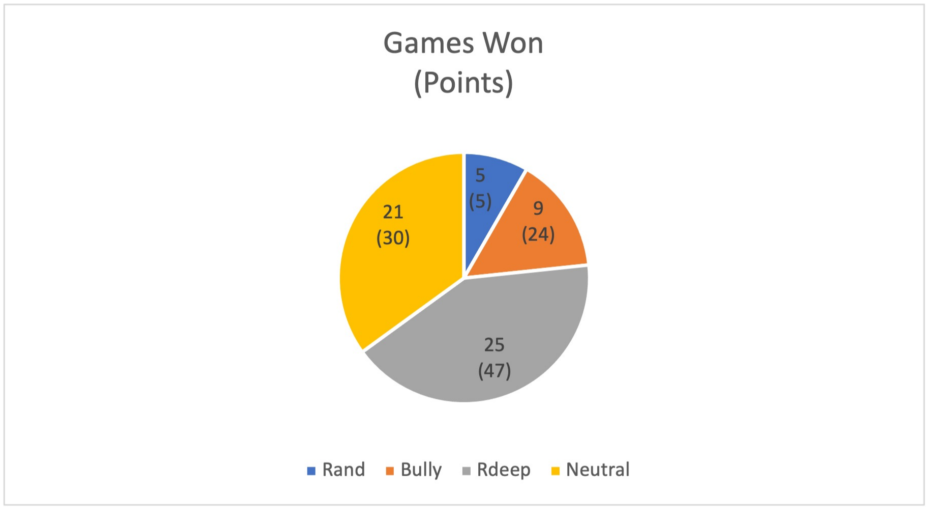

**Fig. 1.** Tournament results of Neutral vs Rand vs Bully vs Rdeep

Our Neutral bot gave very impressive results. It won much more games than Rand and Bully. Rdeep was ahead of our bot by a small margin. Getting this close to Rdeep was a really good performance for us. But in terms of points, Rdeep scored much more than Neutral. The total points scored by Bully was close to Neutral's score even though Bully won less games. This is due to Neutral's play safe strategy as opposed to Bully's play expensive strategy when other strategies have failed.

## 5.2 Cheap vs Given Bots

Although not as good as Neutral, Cheap performed better than we expected. It won more games than Rand, and also a good number of games more than

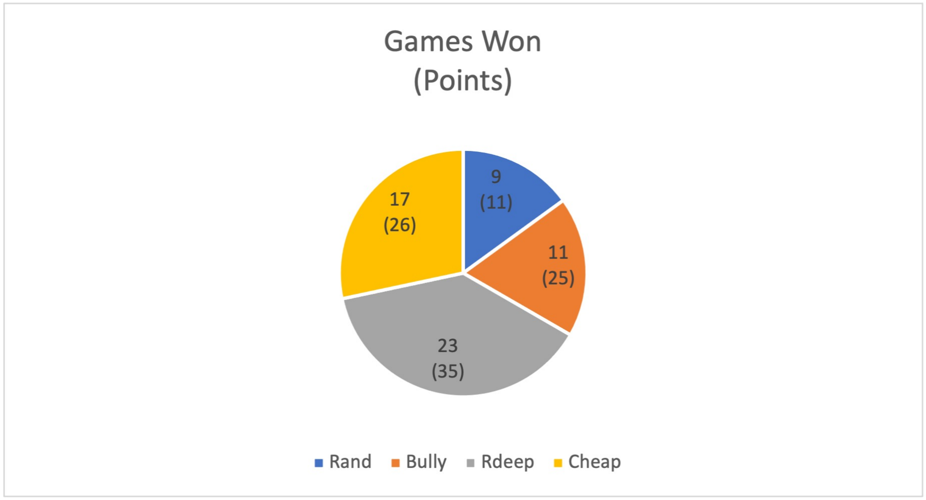

**Fig. 2.** Tournament results of Cheap vs Rand vs Bully vs Rdeep

Bully, which we didn't expect. Playing safely is what helped it win more, thus proving that it's a better strategy than playing expensive (Bully's last resort). However, this strategy costed Cheap to score low points. Its points were almost equal to Bully's (just a 1 point difference). Rdeep won much more games, but Cheap succeeded in losing with fewer points difference as compared to Neutral even though it didn't win many games.

### 5.3 Expensive vs Pre-Built Bots

As expected, Expensive performed the worst making Neutral the best performer amongst the three, with Cheap in the middle. Expensive won almost as many games as Rand, performing just slightly better (1 game more). Even the points were similar, with Expensive scoring slightly more. Both Bully and Rdeep won much more than Expensive, scoring a lot more points.

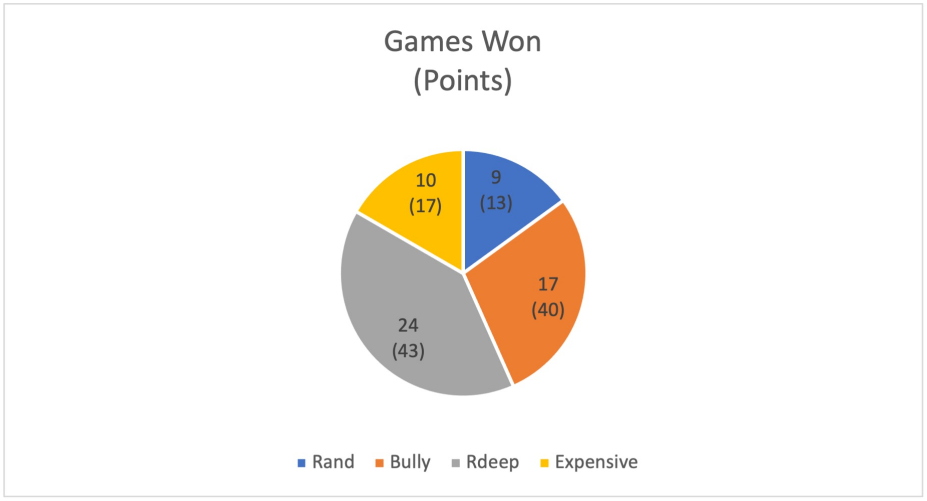

**Fig. 3.** Tournament results of Expensive vs Rand vs Bully vs Rdeep

### 5.4 The three different strategy bots against each other

Now that we have found out how our three custom bots performs compared to the pre-built bots, let's play them against each other !

This tournament's results again match with our prediction perfectly. Neutral performs best by scoring much more than the other two bots and winning many more games as well. Cheap and Expensive have almost similar performance in terms of win rate as Cheap won just one more game than expensive. This is most probably the result of the safe play of Cheap. However, Expensive here scored much more points than Cheap even though it won one game less. This wasn't something we mentioned in our hypothesis, but we did realize its reason after seeing the tournament results. The fact that it scores more when it wins a trick as it plays the most expensive card was something we overlooked before. So, it scores more due to playing aggressively, but loses more games due to the risk it takes.

### 5.5 All bots against each other

We also ran a tournament between all the six bots to get an overview of the performances. Rdeep performed the best with most wins and points as usual,

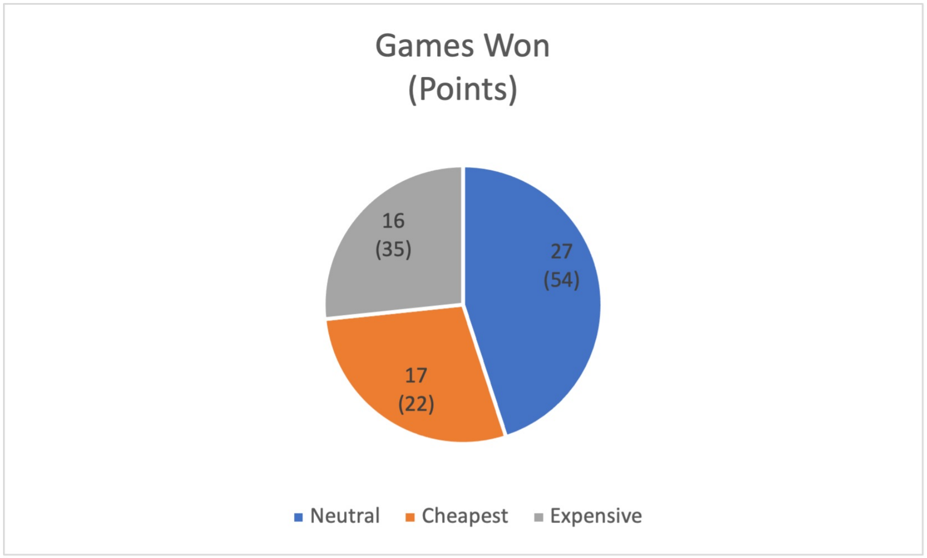

**Fig. 4.** Tournament results of Rand vs Bully vs Rdeep vs Neutral vs Cheap vs Expensive

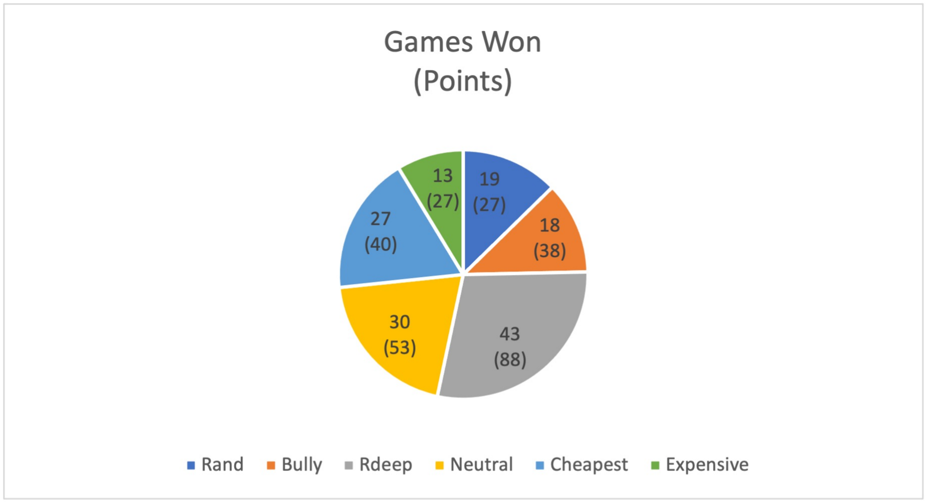

**Fig. 5.** Tournament results of Neutral vs Cheap vs Expensive

followed by Neutral which scored pretty good as well but won just a few games more than Cheap which was quite surprising. Cheap acted different here by winning just three games less than Neutral and scoring higher than all the others, except Neutral and Rdeep. This led us to to and interesting observation discussed later. Bully again was a lot behind Cheap in terms of win, but scored just slightly less than Cheap. Surprisingly, expensive won the least games with a good difference with Rand(the second lowest) which we assume to be happening due to the risky play. But,it still managed to score as much as Rand.

### 5.6 Interesting Observation

As mentioned before, Cheap's good performance made us have a more thorough look at the game play which led us to an interesting observation. That is, Neutral's game-play is a lot similar o Cheap's.And we also figured out that the reason is that when none of the other strategies of Neutral are applicable, it plays the cheapest card to give away the least cards which is same as Cheap's strategy ! But it is the other point scoring strategies that is helping Neutral play more neutrally and better than Cheap. So this observation leads us to the conclusion that playing is safe is far more better than expensive strategy as it's helping our neutral bot as well.

## 6 Conclusion

To sum up, our hypothesis turned out to be correct. The bot playing with the neutral strategies indeed won most of the games and points, which makes the neutral strategy the best of the three. The bot with the risky strategy won the least games and scored least due to losing more games as well, but did score really good in some games despite losing due to the fact that winning any trick did let it score the most points, which is something that we had not predicted. Although Cheap's performance matched our hypothesis, it outperformed our prediction. It won much more games compared to what we expected as a result of the safe play, but did score less in many games. So this research shows that using safe strategies, which chooses the cheaper moves in this case, results in a higher win ratio compared to aggressive or risky strategies, which was choosing the more expensive cards here. But playing safe along with playing aggressive at the right chances increases the winning ratios even more (as done by our Neutral bot).

### 6.1 Future Scope

Lots of researches are going on on game playing agents and strategies to be used, along with some on the nature of strategies as well. More in-depth researches still needs to be done about the nature and type of strategies to optimize player performance in games. Our experiment had simple strategies in the bots, but this idea could be further experimented with agents having a KB of a number of risky, safe and neutral strategies to make the bots even stronger and tried on more complex games to see if that has any changes on the current findings or if any new observation comes up.

## References

1. Games, computers, and artificial intelligence; J Schaeffer; Artificial Intelligence 134(1-2):1; Elsevier 2002; 0004-3702
2. Borm, P., van der Genugten, B. On a relative measure of skill for games with chance elements. Top 9, 91–114 (2001). https://doi.org/10.1007/BF02579073
3. Wisser, Florian. "Evaluating the Performance of Presumed Payoff Perfect Information Monte Carlo Sampling Against Optimal Strategies." Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. 2016.
4. Wisser, Florian. "An expert-level card playing agent based on a variant of perfect information Monte Carlo sampling." Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.
5. Billings D., Papp D., Schaeffer J., Szafron D. (1998) Poker as a testbed for AI research. In: Mercer R.E., Neufeld E. (eds) Advances in Artificial Intelligence. Canadian AI 1998. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 1418. Springer, Berlin, Heidelberg. https://doi-org.vu-nl.idm.oclc.org/10.1007/3-540-64575-6$_5$3
6. Haufe, Sebastian, et al. "Knowledge-based general game playing." KI-Künstliche Intelligenz 25.1 (2011): 25-33.
7. Sandholm, T. (2015). Solving imperfect-information games: The smallest common poker game, two-player limit Texas Hold'em, is essentially solved. Science, 347(6218), new series, 122-123. Retrieved January 27, 2021, from http://www.jstor.org/stable/24745713
8. Solinas, C., Rebstock, D., Buro, M. (2019). Improving Search with Supervised Learning in Trick-Based Card Games. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 1158-1165. https://doi.org/10.1609/aaai.v33i01.33011158
9. Baldwin, R., Cantey, W., Maisel, H., McDermott, J. (1956). The Optimum Strategy in Blackjack. Journal of the American Statistical Association, 51(275), 429-439. doi:10.2307/2281431

## 7 Appendices

### 7.1 Neutral Bot

```
from api import State, Deck
import random
from ..mybot import load_1
from .kb import KB, Boolean


class Bot:

    def __init__(self):
        pass

    def get_move(self, state):
        moves = state.moves()

        random.shuffle(moves)

        for move in moves:
            if move[0] is None and state.get_opponents_played_card() is None:
                return move

        for move in moves:
            if move[1] is not None and state.get_opponents_played_card() is None:
                if not self.kb_consistent(state, move):
                    return move

        if state.get_opponents_played_card() is not None:
            moves_trump_suit = []
            moves_same_suit = []

            for move in moves:
                if state.get_trump_suit() == Deck.get_suit(move[0]):
                    moves_trump_suit.append(move)

                if Deck.get_suit(move[0]) == \
                Deck.get_suit(state.get_opponents_played_card()):
                    moves_same_suit.append(move)

            if len(moves_trump_suit) > 0 and state.get_opponents_played_card() % 5 < 2:
                return return_cheapest(moves_trump_suit)

            elif len(moves_same_suit) > 0:
                cheapest_same_suit = return_cheapest(moves_same_suit)
                expensive_same_suit = return_expensive(moves_same_suit)

                if state.get_opponents_played_card() % 5 > expensive_same_suit[0] % 5 \
                and cheapest_same_suit[0] % 5 > 1:
                    return cheapest_same_suit

                elif state.get_opponents_played_card() % 5 <= cheapest_same_suit[0] % 5:
                    return expensive_same_suit

                else:
                    return_cheapest(moves)
        return return_cheapest(moves)


    def kb_consistent(self, state, move):
    # type: (State, move) -> bool

        kb = KB()

        load_1.general_information(kb)
        load_1.strategy_knowledge(kb)

        index = move[0]

        variable_string = "pm" + str(index)
        strategy_variable = Boolean(variable_string)

        kb.add_clause(~strategy_variable)

        return kb.satisfiable()


def return_cheapest(moves):
    cheapest = moves[0]

    for move in moves:
        if cheapest[0] % 5 <= move[0] % 5:
            cheapest = move

    return cheapest


def return_expensive(moves):
    expensive = moves[0]

    for move in moves:
        if expensive[0] % 5 > move[0] % 5:
            expensive = move

    return expensive
```

### 7.2 Cheap Bot

```
import random

class Bot:
    def __init__(self):
        pass

    def get_move(self, state):
        moves = state.moves()

        random.shuffle(moves)

        cheapest = moves[0]

        for move in moves:
            if move[0] is None:
                return move

        for move in moves:
            if cheapest[0] % 5 <= move[0] % 5 and move[0] is not None:
                cheapest = move

        return cheapest
```

### 7.3 Expensive Bot

```
import random

class Bot:

    def __init__(self):
        pass

    def get_move(self, state):
        moves = state.moves()

        random.shuffle(moves)

        expensive = moves[0]

        for move in moves:
            if move[0] is None:
                return move

        for move in moves:
            if expensive[0] % 5 >= move[0] % 5 and move[0] is not None:
                expensive = move

        return expensive
```

### 7.4 Knowledge Base

```
from bots.mybot.kb import Boolean

M0 = Boolean('m0')
M1 = Boolean('m1')
M2 = Boolean('m2')
M3 = Boolean('m3')
M4 = Boolean('m4')
M5 = Boolean('m5')
M6 = Boolean('m6')
M7 = Boolean('m7')
M8 = Boolean('m8')
M9 = Boolean('m9')
M10 = Boolean('m10')
M11 = Boolean('m11')
M12 = Boolean('m12')
M13 = Boolean('m13')
M14 = Boolean('m14')
M15 = Boolean('m15')
M16 = Boolean('m16')
M17 = Boolean('m17')
M18 = Boolean('m18')
M19 = Boolean('m19')

PM0 = Boolean('pm0')
PM1 = Boolean('pm1')
PM2 = Boolean('pm2')
PM3 = Boolean('pm3')
PM4 = Boolean('pm4')
PM5 = Boolean('pm5')
PM6 = Boolean('pm6')
PM7 = Boolean('pm7')
PM8 = Boolean('pm8')
PM9 = Boolean('pm9')
PM10 = Boolean('pm10')
PM11 = Boolean('pm11')
PM12 = Boolean('pm12')
PM13 = Boolean('pm13')
PM14 = Boolean('pm14')
PM15 = Boolean('pm15')
PM16 = Boolean('pm16')
PM17 = Boolean('pm17')
PM18 = Boolean('pm18')
PM19 = Boolean('pm19')
```

```
def general_information(kb):
    kb.add_clause(M2)
    kb.add_clause(M3)
    kb.add_clause(M7)
    kb.add_clause(M8)
    kb.add_clause(M12)
    kb.add_clause(M13)
    kb.add_clause(M17)
    kb.add_clause(M18)


def strategy_knowledge(kb):
    kb.add_clause(~PM3, M2)
    kb.add_clause(~PM8, M7)
    kb.add_clause(~PM13, M12)
    kb.add_clause(~PM18, M17)
    kb.add_clause(~PM3, M3)
    kb.add_clause(~PM8, M8)
    kb.add_clause(~PM13, M13)
    kb.add_clause(~PM18, M18)
    kb.add_clause(~M2, ~M3, PM3)
    kb.add_clause(~M7, ~M8, PM8)
    kb.add_clause(~M12, ~M13, PM13)
    kb.add_clause(~M17, ~M18, PM18)
```