Machine learning training with a Neural Network: Comparing trained robots in Schnapsen[⋆]

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV, Amsterdam, The Netherlands

Abstract. The study aims to iteratively train a Schnapsen bot and observe whether the win rate is improving against RDeep compared to its previous generation. “The history of the interaction of machine learning and computer game-playing goes back to the earliest days of Artificial Intelligence” [10]. However the amount of research papers since are quite limited. Only a few papers have attempted to go deeper into card games using machine learning. This study intends to make that gap smaller. The methodology relies on various tournaments held between all the generations of the Schnapsen bot and against RDeep, a more advanced bot. The results reveal that Iterations do indeed have an effect on the win rate of the game bot and do improve it significantly. The implications that this paper holds is for the broader field of machine learning and the application of card games. Also it adds to the limited studies on Schnapsen.

Keywords: Machine Learning · Schnapsen · Autonomous Agents · Artificial Intelligence · Card Games · Intelligent Systems.

1 Introduction

Machine learning has been rapidly evolving, promoting more questions to be asked mainly about taking it to a next level and refining existing works. This paper is in pursuit of finding out whether it is possible to enhance an autonomous agent, such as an advanced Schnapsen bot, using iterative training. While various papers have explored general machine learning techniques like supervised learning and labeled data within it [4] or specific ones like autonomous agents [5], many papers have integrated the two topics in multiple different ways. Schnapsen being a complex and distinctive card game has made it possible for us to combine machine learning and autonomous agents. Despite there being a plethora of research and experiments on those two separately, papers that have attempted to combine both topics in the context of Schnapsen are quite limited. This paper seeks to close that gap by researching whether it is possible to use iterative learning to train a specific Schnapsen bot and improve it. By doing so the primary objective is to increase knowledge on the use of machine learning techniques in

[⋆] Project Intelligent Systems 2024

the digital card gaming domain, using Schnapsen as a case study. The aim is to explore the feasibility of enhancing an autonomous Schnapsen bot through iterative learning.

2 Background Information

2.1 Schnapsen

Schnapsen is a two-player card game consisting of twenty cards with five ranks of each suit, each rank being equal to a certain number of points: the Ace (worth eleven points), the Ten (worth ten points), the King (worth of four points), the Queen (worth three points) and the Jack (worth two points). The game consists of two phases and starts in phase one with five cards in each player's hand, which the opponent cannot see. All the remaining cards are put in a talon left face-down on the table, but one is taken out and placed face-up at the bottom of the talon. This card is a trump card and determines the trump suit for the entire game. This is the suit that is ranked higher than all other suits. The entire game is played in tricks. Both players place a card in the middle, of which the first one is considered the leader. When a trick is played, both players take a card from the top of the talon, with the leader being the first one to do so. The winner of a trick takes the cards of the trick played, is credited with the total number of points for those cards and is the leader of the next trick. When the stack is empty, phase two begins. Unlike phase one, both players basically (if they paid attention in the first phase) know the opponent's cards. A player wins a trick by placing either a higher-ranking card or a random card with the trump suit. In both phases there are two special moves: the trump exchange and declaring a marriage. When the leader of a trick has a Jack with the trump suit, this person may exchange the card that determined the trump suit with this card. This is known as the trump exchange. If a player has both the King and Queen of the same suit, they may declare a marriage, showing both cards to the opponent and placing one of the two into the trick. A normal marriage is worth twenty points, while a marriage of the trump suit is worth forty points and these points are obtained immediately after the declaration. There is also one special move which can be made in only phase one: a player can close the talon. If the talon is open after both players have drawn replacement cards for the previous trick, the leader of the upcoming trick may close the talon before they lead a card (so this includes the first trick and the last open-talon trick). They indicate this by placing the trump card face down on top of the talon. Closing the talon means that you claim you can win the deal with just the remaining cards in your hand: so after the talon is closed, nobody is allowed to refill their hand with a card from the talon. This means that once the talon is closed, the deal enters its second phase. If the person who closed the talon fails to win the deal, the opponent gets awarded with bonus game points, which is explained later. Different rules apply for placing cards in the two phases. In both phases, the leader of a trick can place the card he or she wants to, but the responding player is only allowed to do so in the first phase. In the second phase this player is obliged to follow the suit of the card placed by the leader. In addition, the responding player is obliged to win the trick if possible, but this has a lower priority. This means that the opponent must (if possible) play a winning card of the same suit, otherwise play a losing card of the same suit, otherwise play a winning trump card and if all this is not possible, play any other card. The game ends when there are no cards left or when one of the two players reaches sixty-six points.

Game points By finishing one game the winner can earn one to three game points. One player needs seven game points in total to win the whole game. To keep track of them the players can count down from seven to zero and whoever reaches zero or below wins the whole game. The game points can be won in two different ways: we look at if the talon is closed or not. Firstly, if the winner did not close the talon, he can get game points awarded in the following way: three points if the loser took no tricks, two points if the loser took tricks and marriages add to fewer than thirty-three trick points and one point if the loser took tricks and marriages add to at least thirty-three trick points. This means that if the loser declared a marriage at trick one but never won a trick, the deal is worth three game points. Secondly, when the talon is closed we can call the player that closed it the "closer" and the other one the "non-closer". Whenever the talon is closed, nobody can draw cards anymore and the trick points that can be won in the following last five tricks are in the hands of the players. If the closer manages to win the deal with sixty-six points, the game points will be awarded this way: three points if the non-closer had no tricks when the talon was closed, two points if non-closer took tricks and the marriages added to fewer than thirty-three trick points when the talon was closed and one point if non-closer took tricks and the marriages added to at least thirty-three trick points when the talon was closed. Otherwise, if the non-closer wins the deal, the game points are awarded to the non-closer in this manner: three points if the non-closer had no tricks at the time the talon was closed and two points in any other situation.

2.2 Machine Learning

Artificial intelligence can be used to develop, improve and advance games. Various techniques of AI are extensively used in current games, such as finite machines, agents, flocking and scripting [2]. A subfield of AI that has had various applications in games for years is Machine Learning (ML). As indicated by the authors of an article about the combination of machine learning and games, games, whether created for entertainment, simulation, or education, provide great opportunities for machine learning, through, among other things, the variety of possible virtual worlds and the subsequent ML-relevant problems that await the agents in those worlds [1].

2.3 Iterative Learning

The "learning" in machine learning is an iterative process, whereby analyzing data it generalizes, learns from it and in this way can carry out actions without explicit explanation or very specific data. This makes iterative learning an attractive approach for the bots used in the online version of the Schnapsen game described above.

2.4 Randbot

Randbot is one of the two consistent bots used in this research paper and is our baseline. This bot is an autonomous agent, specifically created for the game Schnapsen. Using the random module in Python, the Randbot implements a randomized decision-making strategy. This creates unpredictability in the game and makes this bot attractive for conducting research.

2.5 Rdeep-bot

RDeep is the second bot. Unlike the Randbot, the Rdeep-bot uses a clear strategy, namely the Monte Carlo tree search. From the five cards in his hand, an assumption is made about the possible scenarios with all possible combinations of cards. The bot selects random leaf nodes and continues its search by going back to the root node from the selected leaf nodes, to reevaluate the move and choose the most advantageous move between these end states, with RDeep playing the corresponding card in his hand.

3 Research Question

This study applies an iterative training methodology, exposing the Schnapsen bot to successive generations of training data generated from gameplay. The aim of using the iterative approach is to improve the bot's decision-making strategies by using the results of previous actions. By understanding a player's behavior, it is possible to make the bot behave in a specific way if that player is absent [3]. Through iterative training, the Schnapsen bot evolves by comprehending the behavior of its opponents. The study investigates whether this evolved bot demonstrates improved performance when playing against a completely different bot compared to its earlier generations. The older generations refer to the earlier bots that were created during the initial bot's training, which first plays against itself, before proposing a new machine learning bot which is then taught alongside the original Randbot. This paper aims to find an answer to the question: *To what extent will training with iterative learning of a Schnapsen bot enhance its win rate against the Rdeep bot compared to its previous generations?*

4 Experimental Set-up

4.1 Statistical Testing

For this experiment, there is a null hypothesis and an alternative hypothesis. The goal here is to accept the alternative hypothesis and reject the null hypothesis.

The null hypothesis (H0) is that iterative training has no significant influence on the win rate of the generations for the models playing against the Rdeep-bot (RDeep). The alternative hypothesis (H1) that iterative training does have significant influence on the win rate of the generations for the models playing against the Rdeep-bot. As the hypothesis suggests, the plan for the investigation is to assess the impact of iterative training on the win rates of ML-models derived from Randbot by competing against RDeep, a superior bot. The significance level is an a of 0.05. To evaluate the experiment, the key metric used is the winning rate, which is measured in game points and for the p-value calculations the used metric is the amount of games won. Since the ultimate goal of Schnapsen is to win, having the win rate as the key metric aligns with the main objective of the given setting. Game points are a chosen metric, because it evaluates the overall performance of the bot and shows the growth by how many points it accumulates. For the statistical testing, the first step is to collect all the relevant data, analyze them, and then evaluate whether the observed results provide enough evidence to reject the null hypothesis and accept the alternative hypothesis. By doing this, we can calculate the p-value which we use to reject the null hypothesis, since a treatment has no effect when, in fact, the null hypothesis is true [8]. The type of statistical testing that is being used is ANOVA.

4.2 Methodology

The primary objective of the experiment is to evaluate the difference in win rate measured in game points between RandBot and all the trained machine learning (ML) bots and the impact of iterative training on the models. Additionally, the aim is to inspect how the ML Bots perform in tournaments against each other and the Rdeep-bot. First of all, the Neural Network training model inside the mlbot.py file is changed in such a way that all of the iterations of the training phase of all bots are saved inside a separate folder named "ML rand iterations" located in the text file "iteration output". This is done for a better observation of all iterations per ML-bot. Because of this implementation, the Neural Network training model doesn't print out any of the iterations, which might make it seem that the training isn't working properly, when it actually does. Second of all, the function "create replay memory dataset" is added to the mlbot.py file, which is provided in a former course in the context of Machine learning training. The function helps us in keeping track of past games and helps in the training process of each machine learning model. After taking all of the measures mentioned above, the iterative training will begin with the creation of the first model. The training of the first ML Bot involves 1500 games between two RandBots. From this gameplay a dataset is derived which represents the foundation for training the first ML Bot. For the second ML Bot a third RandBot plays 1500 games against the first ML model and out of those games a new dataset is generated and added to the same text file as the first dataset. For the following ML Bots the two most recent bots play the games against each other from which the next ML Bot is created. The main parameters that were actively observed in each ML model are the win score, validation score and iterations in the training phase of each model. The replay memory dataset saves all the games into one file, contributing to the improvement, because it captures the progress of the gameplay more comprehensively. To do this, a function is created with the name "train rand model" and is implemented for training. All the games are saved to the same directory which boosts progress when saved together. For further implementations of games and tournaments between bots, a function is created that goes by the same tournaments which helps in automating and simplifying the process of generating new competitions. Additionally, this function contains pieces of code which represent the tournaments which were added from a previous course in which tournaments between Bots were used to test the performance of them. Furthermore, the function tournaments stores all the outcomes of the games in the directory "ML rand games" under different text files. Randbot will compete in 1000 games against RDeep to create a baseline for the other tournaments. Randbot being the predecessor to the ML models makes it the perfect for the baseline as it is the reference point that has not been influenced by any learning optimization. The opponent used to compete against the baseline in games is Rdeep. This bot is a very proficient player that is designed to make very strategic decisions as it evaluates the moves to be made in a deeper and more informed way. Moreover, it takes into account the immediate and future consequences of the decisions to be made. After training all seven ML models, a tournament will be held where all the ML bots play against each other. The intention of this experiment is to prove that the training process did in fact improve the win rate expressed through game points. After that, all ML models play in a tournament against Randbot, their precursor, to show that evolution has taken place. Rdeep was made to play tournaments against all seven of the ML models to get an idea of the relative performance that they have against Rdeep-bot compared to RandBot. Once all the information is gathered, it will be compared to see how much progress has occurred across the generations. To calculate the p-value an additional test will be done as you can not statistically analyze the tournaments with the ML bots against each other, since the game points are not stable values as their total can change by each round while the number of games won always has a stable total. The sum of the number of games to be won is equal to the number of games played. In order for the testing to be applicable, we take each ML bot, play it against RDeep for 1000 games each and take the amount of games won by each bot and calculate the p-value.

5 Results and Findings

5.1 Randbot vs Rdeep-bot

Before analyzing the other results of the Machine Learning models that were created, a baseline is established. Because all the created bots are trained descendants of the RandBot the first action that needs to be taken is to make RandBot play against another bot and later we can assess how much the Machine Learning bots improved from the gameplay of RandBot. As a result of this, RandBot played a thousand games against RDeep which ended with RandBot earning only 334 game points while RDeepBot earned 1609 game points (Fig. 1). Given these results we can clearly state that RDeep outperformed RandBot. Because of this our goal is to see through results if one or more of the ML models could outperform RandBot or even RDeepBot.

5.2 Win Rate and Validation Score in Neural Network iterations

Displayed in Figure 2 and Figure 3 are the win rate and validation score of each Machine Learning model before and after training with a Neural Network. The goal of this data collection and comparison was to find if there is any connection between the win rate and validation score given by the Neural Network at the beginning and end of each training, and the future performance in tournaments of each bot. If we were to follow the win rate at the end the order of the bots from most likely to least likely to win the most points in the following tournaments is ML Bot 7, ML Bot 2, ML Bot 6, ML Bot 3, ML Bot 4, ML Bot 5 and ML Bot 1. Otherwise, if we were to consider the validation score and create an array of bots from most likely to least likely to win the most game points, it would look like this: ML Bot 7, ML Bot 2, ML Bot 6, ML Bot 5, ML Bot 3, ML Bot 4 and ML Bot 1. Both lists from above are very similar and they propose that the best Machine Learning bot is the last one. At first this seems plausible since it is the bot trained on the biggest database that contains games between other already trained models. Nonetheless, this will be proven false in the actual tournaments.

5.3 Machine Learning Bots Tournament

Before comparing the performance of the ML Bots against other Bots each of the trained models need to be tested against each other. This was done through a tournament between all the ML models. Through the results of this experiment the prediction from the previous experiment (5.2) was proven to be false because as we can see in Figure 4 the models that performed the worst were ML1 and ML7, while ML2 performed the best. In addition to this we can say that the win rate curve (Fig. 5) spiked the most at the ML2 model, while it descended at the ML1 and ML7 model. It is observed that with more training the models don't get better, but they can get worse.

5.4 ML Bots vs RandBot tournament

While trying to find out if the ML models improved through training a tournament was held with the intention to find if they improved from the root bot. As it is observed in Figure 6 almost all Machine Learning bots outperformed or almost outperformed RandBot. The ones that performed the worst were the first and last generations of trained bots, which further proves the fact that training with Machine Learning can improve to a certain extent and more training can lead to a worse performance.

5.5 ML Bots vs RDeepBot tournament

As seen in the results displayed in Figure 7 three of the trained models surpassed RDeepBot and two other ML bots almost outran it. However, as previously analyzed, the bots that performed the worst were ML1 and ML7. Seeing that multiple ML Bots beat the RDeepBot, is a big improvement compared to the performance of the RandBot against the same competitor (5.1). Additionally, the win rate curve in Fig. 8 is similar to the one in Fig. 5, the only big change that can be seen is that ML Bot 5 was overall better than ML Bot 2. Furthermore, what came out of this last testing is that with the right amount of training a Machine Learning Bot (or multiple) can become proficient enough to beat RDeepBot. A drawback to this is that if there aren't enough generations or if there are too many generations of Bots they will not win against the RDeepBot.

6 Discussion

The test conducted in this paper is ANOVA as aforementioned in 4.1. ANOVA test is conducted when three or more groups are compared and analyzed whether there are any concrete differences. "if the means of any two groups are different from each other, the null hypothesis can be rejected."[9] In this case the comparison is about whether the game wins. First Randbot against RDeep and then each of the seven ML bots play against RDeep. The P-value was calculated using the difference in wins between when randbot played RDeep and when the MLbots played RDeep. The outcome was very significant as it is for every ML bot below the significance level of 0,05 with a p-value of 0,0001 for every ML bot [5.6, table 1] Given the low p-value the null hypothesis is rejected as it means that there is significant level between each ML bot playing against RDeep and Randbot against RDeep.

7 Conclusion

To conclude this research paper, by working with iterative training through Machine Learning, multiple robots were trained with the objective of improving their gameplay in Schnapsen from the baseline robot and comparing their gameplay with a more complex bot. Through surveying gameplay and the results, it was observed that the learning curve of each generation of Bots has at first an upward curve, but after a few generations it has a downward curve, in this paper's case that threshold was reached by the Machine Learning Bot 5. What this showed is that while training can be beneficial, it also has its own soft spot, which means that after generations the trained models become worse. However, most of the newly created bots managed to outperform their base robot and some of them outran RDeepBot, which confirms that training is beneficial.. The null hypothesis (H0) is that iterative training has no significant influence on the win rate of the generations for the models playing against the Rdeep-bot (RDeep). As aforementioned the null hypothesis was rejected as the p-value was way lower than the significance level of 0.05 with 0,0001 for each ML bot. The alternative hypothesis (H1) is that iterative training does have significant influence on the win rate of the generations for the models playing against the Rdeep-bot. Through the results it was proven that the iterative training does indeed have an impact on the win rate of the generations of the Schnapsen bot .

8 Future Work

After the experiments were concluded and the results collected, there were certain things that could be added as further improvements: additional experiments could be conducted or some variables that remained constant throughout the whole testing process could have been tested on. First and foremost, one of the suggested improvements was to also take into account the validation score when creating a new ML-bot. What this could mean is to create a new model based on the gameplay of two previously trained Machine learning lots that have the highest validation values and see what difference that has on actual games and tournaments compared to using the memory database of the two most current bots. Second, a more in-depth experimentation with more generations of ML-bots and more games per generation of replay memories would have been beneficial. This could be done to further validate findings or to generate new findings and obtain more relevant outcomes. Lastly, one of the variables that could have been tested on more is the amount of hidden layers of the Neural Network (NN). In this experiment, the NN constantly had only one hidden layer with thirty neurons, but it would have been compelling to examine the impact of a larger Neural Network on the ML-bot's training phase or to find the right number of hidden layers for our data. However, this could be challenging to do so, since the selection of hidden layers is a very difficult task as in some cases due to the number of hidden layers, conditions known as overfitting and underfitting may occur, which affect the efficiency and time complexity of the network badly. It is difficult to determine a good network topology solely from the number of inputs and outputs [6]. Even if one would try various amounts of hidden layers, they could end up with a similar conclusion as this paper [7], which states that three layers are enough for most neural networks.

References

1. Bowling, Michael, et al. ”Machine learning and games.” Machine learning 63.3 (2006): 211-215.
2. Sweetser, Penelope, and Janet Wiles. ”Current AI in games: A review.” Australian Journal of Intelligent Information Processing Systems 8.1 (2002): 24-42.
3. “—”. IEEE Conference Publication — IEEE Xplore, 1 oktober 2019, ieeexplore.ieee.org/abstract/document/8924850.
4. Florian, Razvan V. ”Autonomous artificial intelligent agents.” Center for Cognitive and Neural Studies (Coneural), Cluj-Napoca, Romania (2003).
5. THE FUNDAMENTALS OF MACHINE LEARNING `https://www.interactions.com/wp-content/uploads/2017/06/machine_learning_wp-5.pdf`
6. “Effects of hidden layers on the efficiency of neural networks”. IEEE Conference Publication — IEEE Xplore, 5 november 2020, ieeexplore.ieee.org/abstract/document/9318195.
7. Shen, Zuowei, Haizhao Yang, and Shijun Zhang. ”Neural network approximation: Three hidden layers are enough.” Neural Networks 141 (2021): 160-173.
8. Zahid, Maham. ”Inferential Statistics.” Handbook of Randomized Controlled Trials: 51.
9. Kim, Tae Kyun. ”Understanding one-way ANOVA using conceptual figures.” Korean journal of anesthesiology 70.1 (2017): 22-26.
10. Bowling, Michael, et al. ”Machine learning and games.” Machine learning 63.3 (2006): 211-215.

9 Appendices

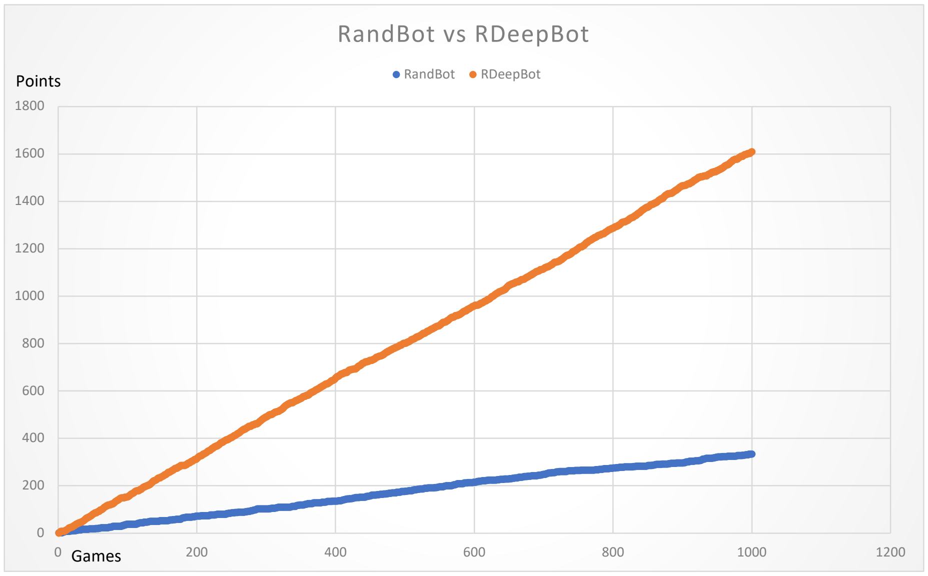
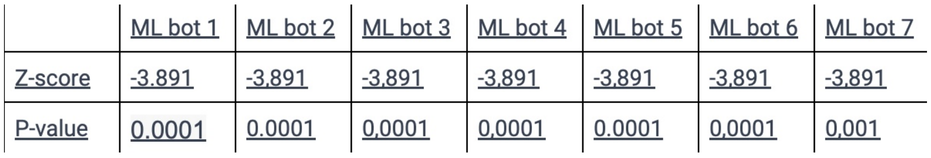


Fig. 1. Graph that contains data collected from a thousand games between RandBot and RDeepBot.



	ML bot 1	ML bot 2	ML bot 3	ML bot 4	ML bot 5	ML bot 6	ML bot 7
Z-score	-3.891	-3,891	-3,891	-3,891	-3,891	-3,891	-3,891
P-value	0.0001	0.0001	0,0001	0,0001	0.0001	0,0001	0,001

Table 1. Calculations of p-value.

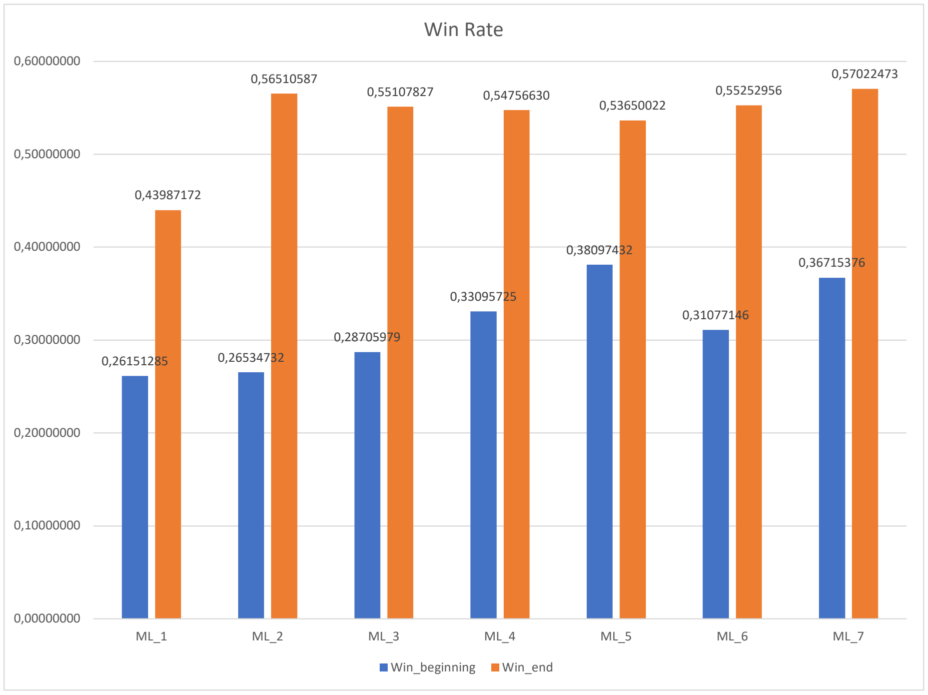


Fig. 2. Win rate chart which contains data about the win rate of each bot at the beginning and end of the training phase and stored in the directory "ML rand iterations/iteration output.txt."

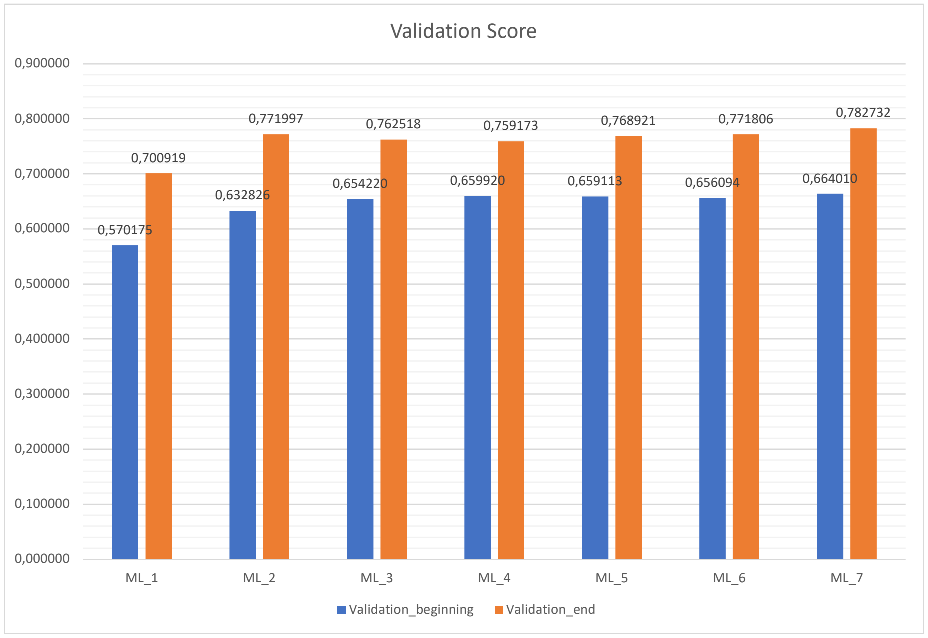


Fig. 3. Validation score chart which contains data about the validation of each bot at the beginning and end of the training phase and stored in the directory "ML rand iterations/iteration output.txt."

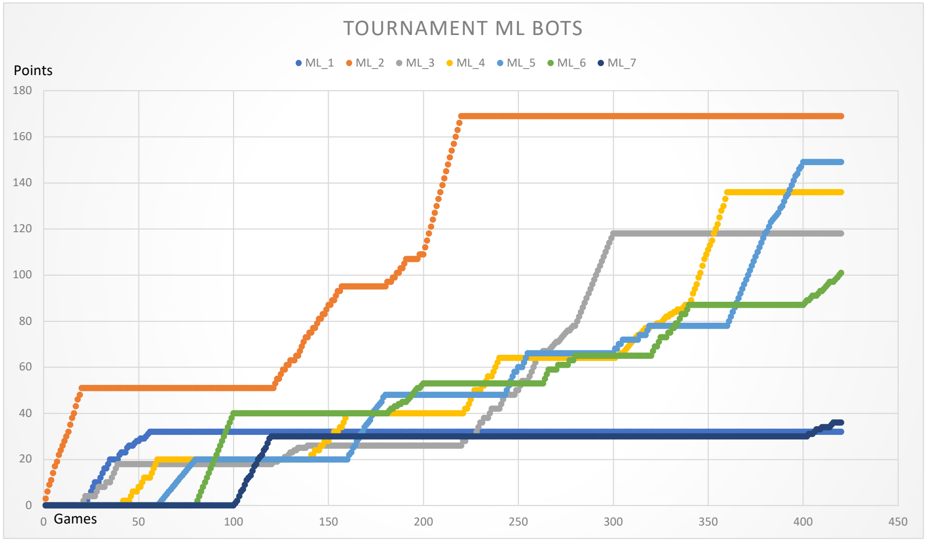


Fig. 4. Graph with the points won by each ML Bot in a tournament.

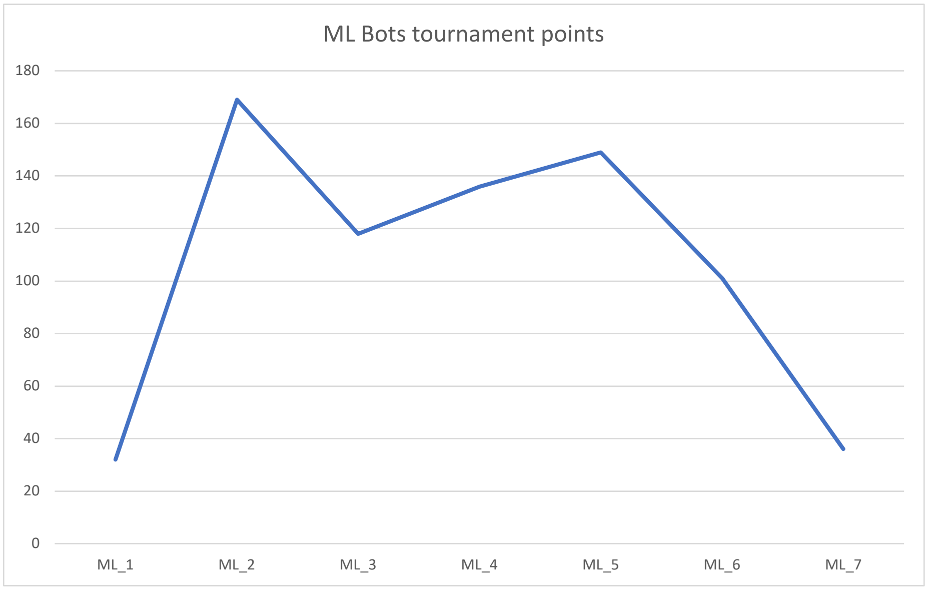


Fig. 5. Graph which represents the win rate curve based on the points won in the tournament between the Machine Learning models.

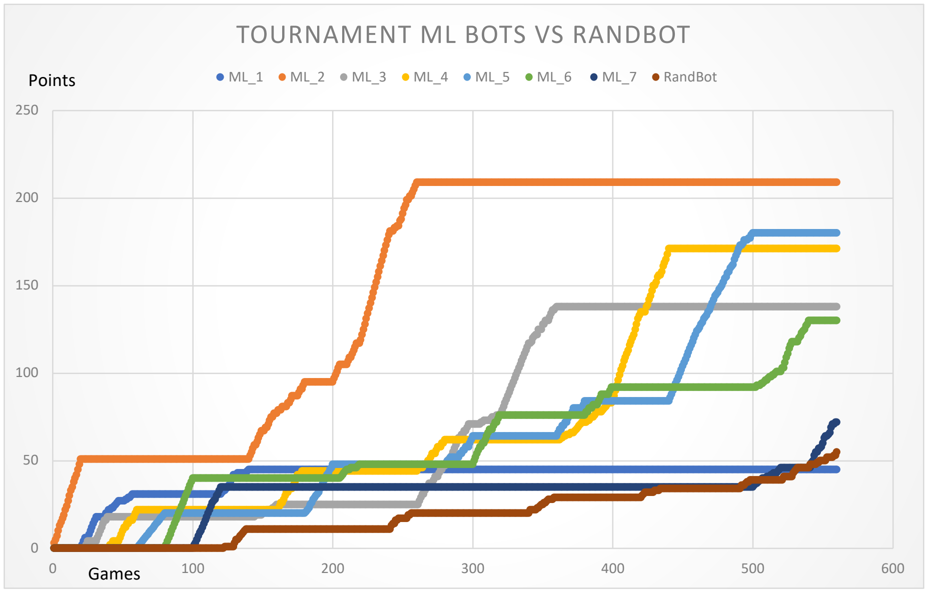


Fig. 6. Data of a tournament between all ML bots and RandBot represented in a graph.

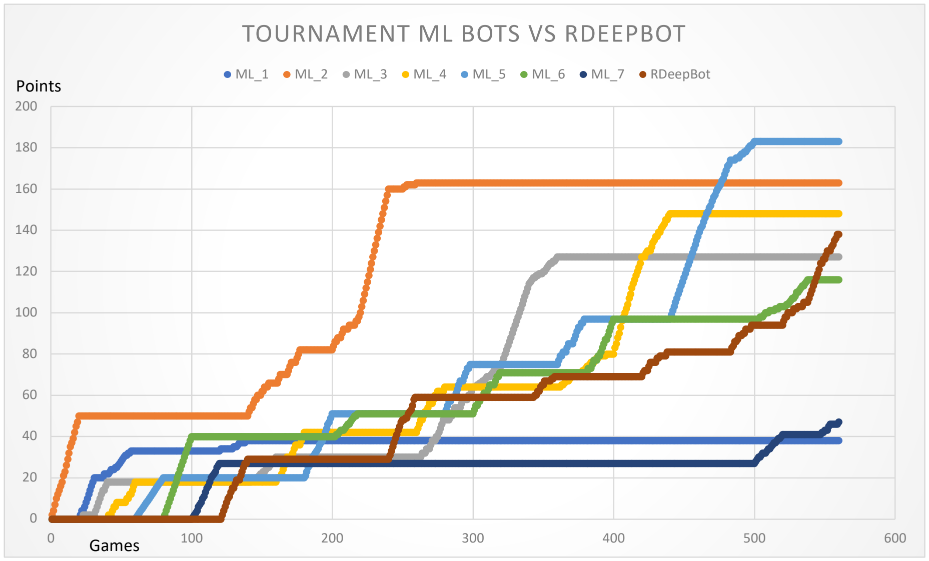


Fig. 7. The results of the tournament between the ML Bots and RDeepBot.

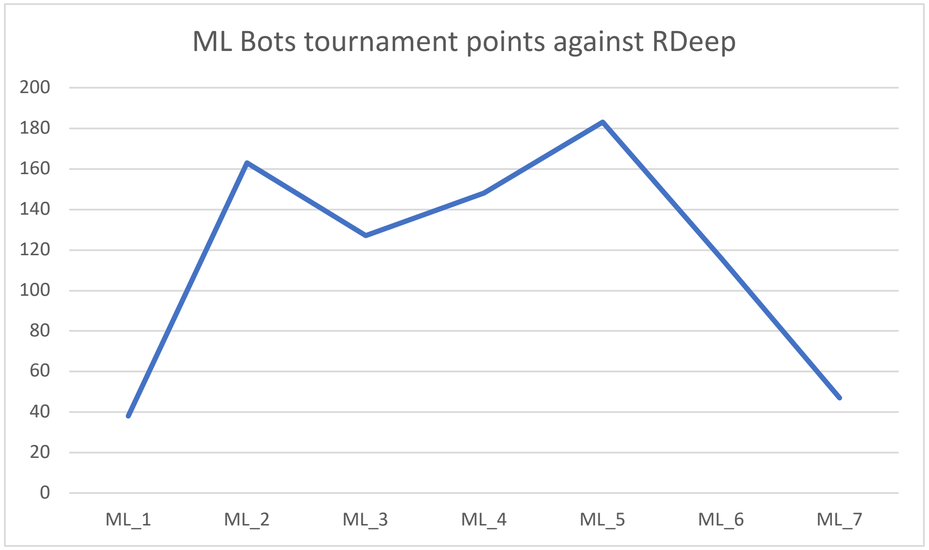
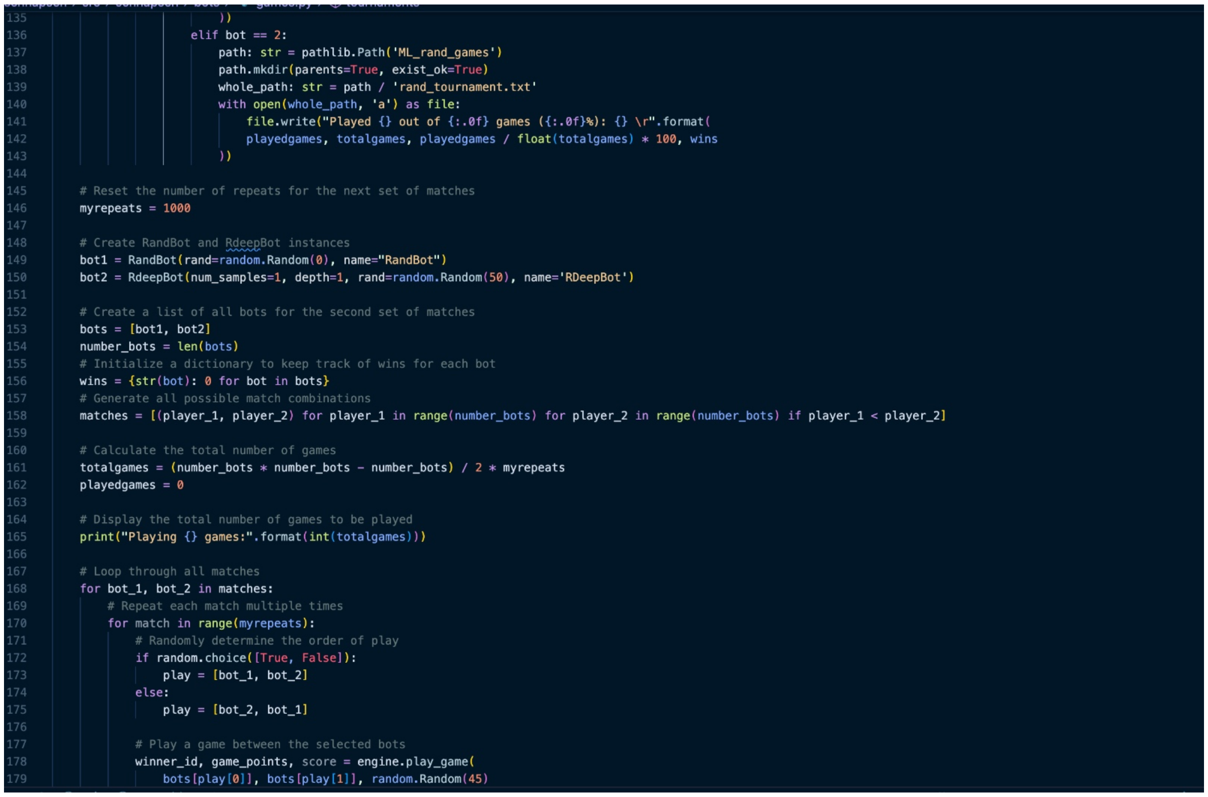
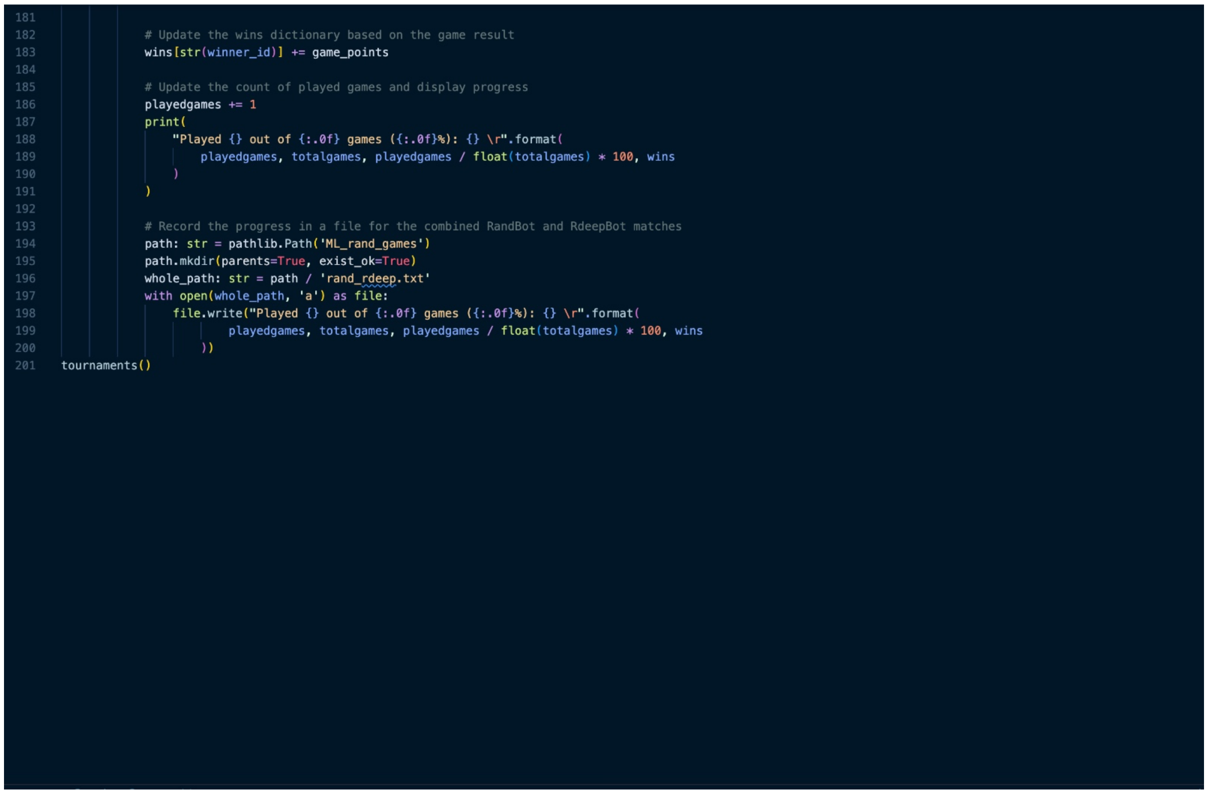


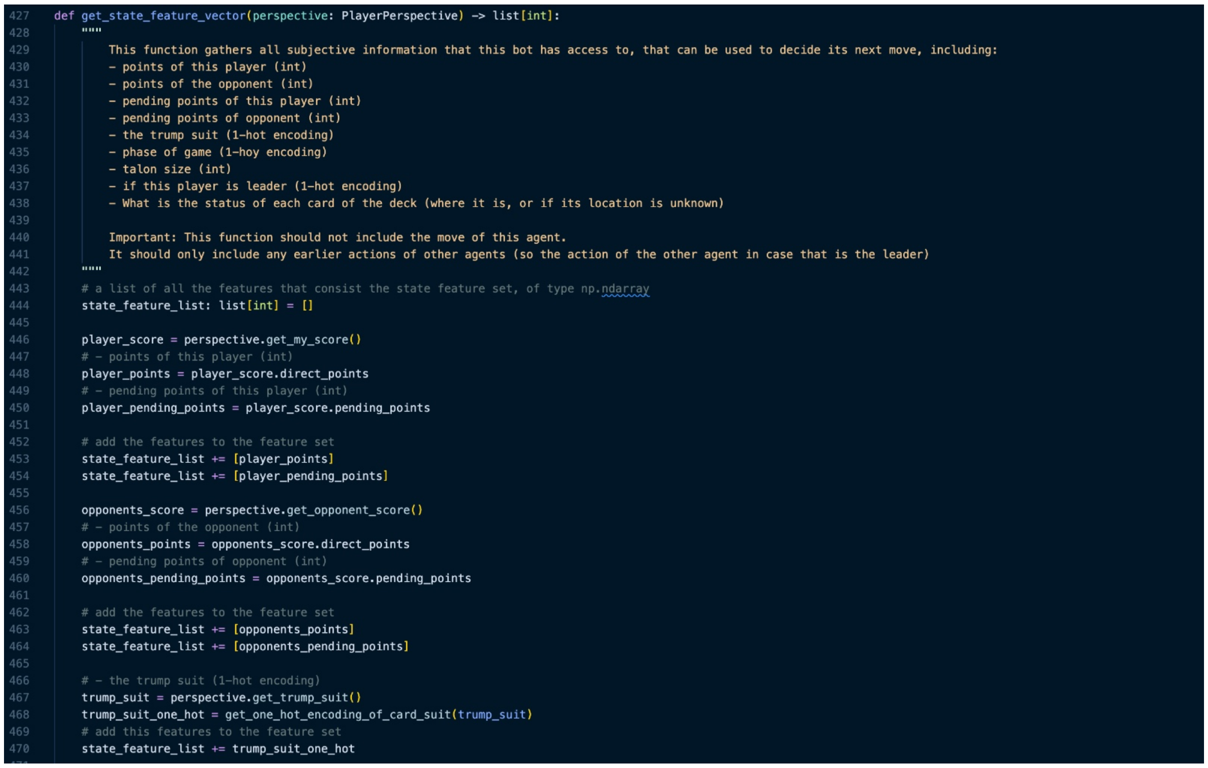
Fig. 8. Graph which represents the win rate curve based on the points won in the tournament between the Machine Learning models and RDeepBot.



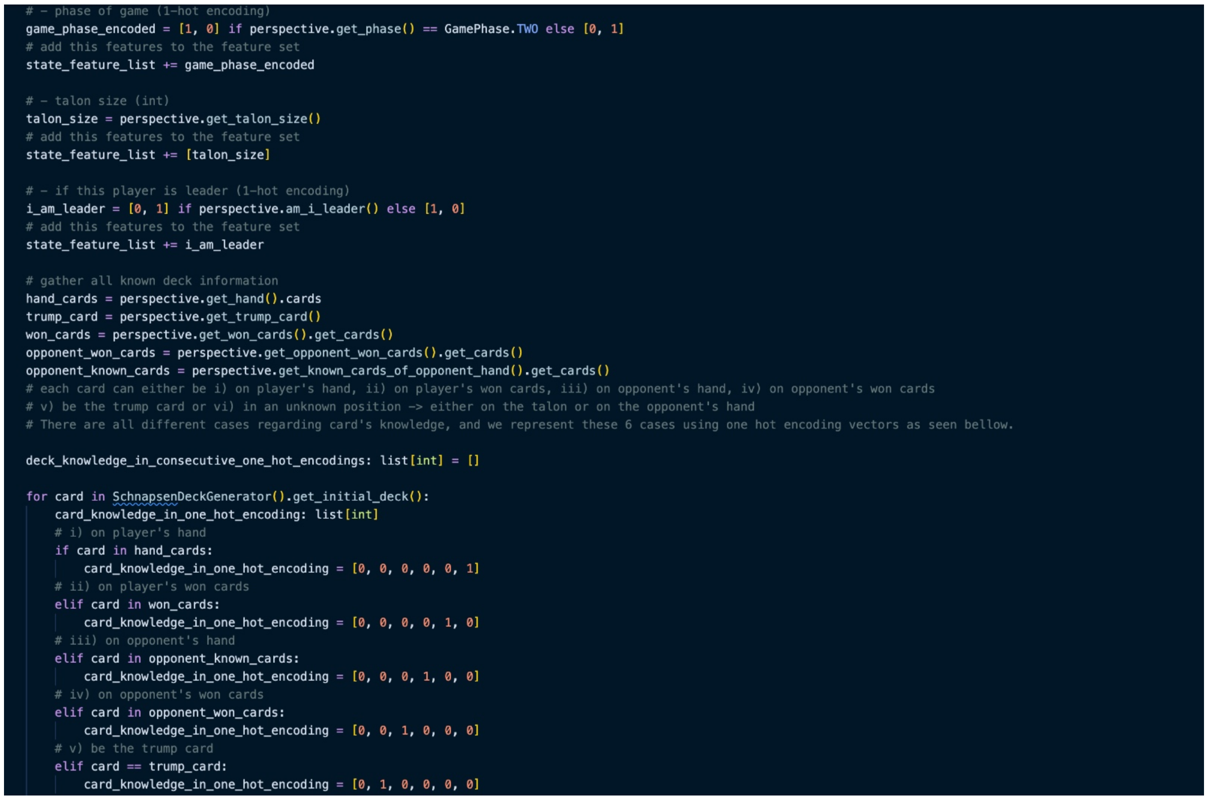
```
                    ))
                elif bot == 2:
                    path: str = pathlib.Path('ML_rand_games')
                    path.mkdir(parents=True, exist_ok=True)
                    whole_path: str = path / 'rand_tournament.txt'
                    with open(whole_path, 'a') as file:
                        file.write("Played {} out of {:.0f} games ({:.0f}%): {} \r".format(
                        playedgames, totalgames, playedgames / float(totalgames) * 100, wins
                    ))

    # Reset the number of repeats for the next set of matches
    myrepeats = 1000

    # Create RandBot and RdeepBot instances
    bot1 = RandBot(rand=random.Random(0), name="RandBot")
    bot2 = RdeepBot(num_samples=1, depth=1, rand=random.Random(50), name='RDeepBot')

    # Create a list of all bots for the second set of matches
    bots = [bot1, bot2]
    number_bots = len(bots)
    # Initialize a dictionary to keep track of wins for each bot
    wins = {str(bot): 0 for bot in bots}
    # Generate all possible match combinations
    matches = [(player_1, player_2) for player_1 in range(number_bots) for player_2 in range(number_bots) if player_1 < player_2]

    # Calculate the total number of games
    totalgames = (number_bots * number_bots - number_bots) / 2 * myrepeats
    playedgames = 0

    # Display the total number of games to be played
    print("Playing {} games:".format(int(totalgames)))

    # Loop through all matches
    for bot_1, bot_2 in matches:
        # Repeat each match multiple times
        for match in range(myrepeats):
            # Randomly determine the order of play
            if random.choice([True, False]):
                play = [bot_1, bot_2]
            else:
                play = [bot_2, bot_1]

            # Play a game between the selected bots
            winner_id, game_points, score = engine.play_game(
                bots[play[0]], bots[play[1]], random.Random(45)
```



```
            # Update the wins dictionary based on the game result
            wins[str(winner_id)] += game_points

            # Update the count of played games and display progress
            playedgames += 1
            print(
                "Played {} out of {:.0f} games ({:.0f}%): {} \r".format(
                    playedgames, totalgames, playedgames / float(totalgames) * 100, wins
                )
            )

            # Record the progress in a file for the combined RandBot and RdeepBot matches
            path: str = pathlib.Path('ML_rand_games')
            path.mkdir(parents=True, exist_ok=True)
            whole_path: str = path / 'rand_rdeep.txt'
            with open(whole_path, 'a') as file:
                file.write("Played {} out of {:.0f} games ({:.0f}%): {} \r".format(
                        playedgames, totalgames, playedgames / float(totalgames) * 100, wins
                    ))
tournaments()
```



```python
def get_state_feature_vector(perspective: PlayerPerspective) -> list[int]:
    """
        This function gathers all subjective information that this bot has access to, that can be used to decide its next move, including:
        - points of this player (int)
        - points of the opponent (int)
        - pending points of this player (int)
        - pending points of opponent (int)
        - the trump suit (1-hot encoding)
        - phase of game (1-hoy encoding)
        - talon size (int)
        - if this player is leader (1-hot encoding)
        - What is the status of each card of the deck (where it is, or if its location is unknown)

        Important: This function should not include the move of this agent.
        It should only include any earlier actions of other agents (so the action of the other agent in case that is the leader)
    """
    # a list of all the features that consist the state feature set, of type np.ndarray
    state_feature_list: list[int] = []

    player_score = perspective.get_my_score()
    # - points of this player (int)
    player_points = player_score.direct_points
    # - pending points of this player (int)
    player_pending_points = player_score.pending_points

    # add the features to the feature set
    state_feature_list += [player_points]
    state_feature_list += [player_pending_points]

    opponents_score = perspective.get_opponent_score()
    # - points of the opponent (int)
    opponents_points = opponents_score.direct_points
    # - pending points of opponent (int)
    opponents_pending_points = opponents_score.pending_points

    # add the features to the feature set
    state_feature_list += [opponents_points]
    state_feature_list += [opponents_pending_points]

    # - the trump suit (1-hot encoding)
    trump_suit = perspective.get_trump_suit()
    trump_suit_one_hot = get_one_hot_encoding_of_card_suit(trump_suit)
    # add this features to the feature set
    state_feature_list += trump_suit_one_hot
```



```
# - phase of game (1-hot encoding)
game_phase_encoded = [1, 0] if perspective.get_phase() == GamePhase.TWO else [0, 1]
# add this features to the feature set
state_feature_list += game_phase_encoded

# - talon size (int)
talon_size = perspective.get_talon_size()
# add this features to the feature set
state_feature_list += [talon_size]

# - if this player is leader (1-hot encoding)
i_am_leader = [0, 1] if perspective.am_i_leader() else [1, 0]
# add this features to the feature set
state_feature_list += i_am_leader

# gather all known deck information
hand_cards = perspective.get_hand().cards
trump_card = perspective.get_trump_card()
won_cards = perspective.get_won_cards().get_cards()
opponent_won_cards = perspective.get_opponent_won_cards().get_cards()
opponent_known_cards = perspective.get_known_cards_of_opponent_hand().get_cards()
# each card can either be i) on player's hand, ii) on player's won cards, iii) on opponent's hand, iv) on opponent's won cards
# v) be the trump card or vi) in an unknown position -> either on the talon or on the opponent's hand
# There are all different cases regarding card's knowledge, and we represent these 6 cases using one hot encoding vectors as seen bellow.

deck_knowledge_in_consecutive_one_hot_encodings: list[int] = []

for card in SchnapsenDeckGenerator().get_initial_deck():
    card_knowledge_in_one_hot_encoding: list[int]
    # i) on player's hand
    if card in hand_cards:
        card_knowledge_in_one_hot_encoding = [0, 0, 0, 0, 0, 1]
    # ii) on player's won cards
    elif card in won_cards:
        card_knowledge_in_one_hot_encoding = [0, 0, 0, 0, 1, 0]
    # iii) on opponent's hand
    elif card in opponent_known_cards:
        card_knowledge_in_one_hot_encoding = [0, 0, 0, 1, 0, 0]
    # iv) on opponent's won cards
    elif card in opponent_won_cards:
        card_knowledge_in_one_hot_encoding = [0, 0, 1, 0, 0, 0]
    # v) be the trump card
    elif card == trump_card:
        card_knowledge_in_one_hot_encoding = [0, 1, 0, 0, 0, 0]
```