Number and Presumed Quality of Features on Game Performance in Schnapsen*

Eric Zielinski, Cristiano Milanese, and Matteo Gambarutti

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam
{e.zielinski,c.milanese,m.gambarutti}@student.vu.nl

Abstract. This paper looks into how the number and presumed quality features impact the number of points earned for five machine learning bots based off of simple Monte Carlo sampling. Heterogenous datasets had to be created to test the impact of different quantities and presumed qualities of the bots. The performances are statistically tested to determine significance between bots that use a different quantity of features and are statistically tested to determine significance on the presumed quality of the features.

Keywords: Feature · Schnapsen · Machine learning · Monte Carlo · Quantity · Quality · Relevance.

1 Introduction

Professional board game players of some of the most complex games have met their match in recent years. AlphaGo and its variants have defeated reigning champions all over the world [9]. DeepBlue defeated the chess grandchampion some twenty years ago [4]. One could say that chess grandmasters are not so "grand" anymore. OpenAI Five defeated professional opponents in 1v1 environment in a game called Dota 2 [1]. In 2018, it was able to defeat amateur and semi-professional human teams, but it failed to win against a team of five professional human players. While it lost that match, this shows that game AI continues its march toward conquering more complex games, including games where it attempts to defeat multiple human opponents working as a team.

This research paper focuses on the feature set of the European cardgame called Schnapsen. It is a partially observable, deterministic, static, discrete and multi-agent game. Said another way, it is an adverserial game where each turn is determined by who won the previous turn, both players vie for the most points, and neither player knows the other player's hand in the beginning of the game.

2 Background Information

Schnapsen is a two-player adverserial-based card game with two phases and 20 cards consisting of Jacks, Queens, Kings, Tens and Aces; the points per card

* Supported by the Intelligent Systems teaching team

are 2, 3, 4, 10, 11 respectively. The trump suit follows the same hierarchy but a Jack of trump suit will beat the Ace of any other suit. Every game consists of X number of rounds. Every round consists of a maximum of 10 plays (20 available cards) in which each player uses a particular card in their hand. Points are distributed to one player after each play depending on which cards were played. Points are based on the combined value of both cards in-play, in addition to marriages: when a player leading has a matching suit of Queen and King - 20 points non-trump suit and 40 points trump suit. A player wins once they are the first to reach 66 points in the round (round points), but they may be beaten by the opponent if they fail to accurately keep track of their respective points.

In Phase 1, each player's hand consists of 5 cards. Each player's hand will continue to be replenished until there are no cards left in the stack, at which point Phase 2 commences. A player leading may perform a trump-Jack exchange, which means the player will obtain a higher-valued trump card.

Phase 2 introduces stricter rules. If the leading player plays suit X, the other play must player suit X if available, else they must play a trump card, else they play whatever they have left in their hand.

The winning player earns between 1 and 3 game points depending on how many round points the opponent had: 3 points if opponent had 0 points, 2 points if the opponent had less than 33 points and 1 if the opponent had between 33 and 65 points. The goal is to many rounds with as many points as possible to win the game until some threshold is reached.

2.1 Framework

We used an existing neural-network based framework that provided a game engine, a game-state object script and several helper functions. The classifier was a multi-layered perceptron ("MLP"). According to [2], it's sensitive to feature-scaling. In other words, it's best to include features with the same range of numbers, such as fitting between 0 and 1. In addition, the framework provided several bots that worked or were close to working right out of the box. The bot called rand played randomly; the bot called bully played its best cards first; the bot called rdeep used simple Monte Carlo sampling to peer several moves ahead and assigned a heuristics value based on the ratio of points earned by a player over total points earned in the entire round - this is the parent bot for all of the bots tested in this paper. The framework used the package called *scikit-learn*. Scikit gained popularity because of its wide range of state-of-the-art machine learning algorithms for both medium-scale supervised and unsupervised problems. Moreover, it focuses on bringing machine learning to non-specialists. Emphasis is put on ease of use, performance, documentation, and API consistency [8].

2.2 What are Features?

One of the most important elements of a good classifier is the selection of features [3]. Some say of feature selection is an art rather than science. Notwithstanding, it's important to select different features depending on the context, or what needs to be classified. A classifier with a feature Features can be numeric - such as ratios or integers, and they can consist of characters, such as a string.

2.3 Feature Engineering

Feature engineering encompasses feature extraction and selection in relation to machine learning. It is difficult to formulate a set of rules because both extraction and selection are subject to specific contexts. However, experts use certain conventions when dealing with these tasks. First, a feature should be simple. Second, it should be indepedent. Third, it should discriminative and independent to avoid biases or double counting features that are separate features but actually dependent in some way. Determining the best features to add is an NP-HARD problem [6], hence why some call feature selection an art rather than a science; however, advances in feature selection methods have led to automated selection in some cases [5] Finally, it might be tempting to add as many features as possible when first determining the right set of features. This method leads to what is called the Curse of Dimensionality [7].

2.4 Default Features

The feature set included data related to the current points of both players, both players' pending points, the trump suit, the phase, the remaining stock size, who's leading, whose turn it is and the opponents played card.

Some of the features were a ratio of several of these data points, such as player's points in relation to the total points earned in the round.

3 Research Question

The research question for this paper is: *Do the quantity and presumed quality of features in the feature set impact the performance (ie, earned points) of a machine-learning agent in the game of Schnapsen?* Before starting out, we hypothesized that the number of features wouldn't help performance in the context of Schnapsen, but we thought the relevance and quality of the features would. We determined the best way to answer the research question would be the tweak the number of features and the quality of the features based on industry convention. Perhaps we could confirm the old adage *Everything in Moderation* holds in the world of machine learning feature sets. Certainly some features are better than others, so perhaps we could find better ones.

4 Experiment

We decided to run a chi-square test for homogeneity using the proportions given by the tournament results, at first we set the null hypothesis to be the case in which all three win ratios were the same, while the alternative hypothesis was asserting the opposite. We decided to keep the significance level on 0.05 as it has been the default for previously seen statistic analysis. While the observed value marked as 'O' in chi-square formula would become the win ratio for that particular bot, the expected value E is evaluated by summing the overall game points assigned on the two matches that each bot A played (against B and C in the tournament) and then divide it by the number of matches, in order to have an expected value reflecting a fair amount of points gained by the bots during their turns to play. The difference of the two gives us the error, crucial for the test statistics to generate the probability of drawing those scores by chance, the sum of each addend provides the crossed result of these probabilites. Is indeed the evaluation of the test statistics that allowed us to build our experiment. Its methodology consisted of two parts: a qualitative and a quantitative part to test against the quantity of features and another set of tests comparing the quality of features of the five bots.

Each bot was modeled on rdeep. Description of bots:

Bot 1 - ml_rdeep is a machine learning bot with the default features mentioned in the previous section.

Bot 2 - ml_under_feat is a machine learning bot with five less features, for a total of five.

Bot 3 - ml_over_feat is a machine learning bot with two additional features, for a total of twelve.

Bot 4 - ml_final_points is a machine learning bot with a total of ten features - the same as ml_rdeep - but with one presumably unimportant feature replaced with a presumable more important one.

Bot 5 - ml_fine_tuned is a machine learning bot with a total of eight features.

The qualitative tests were structured in such a way to deduce which bots had less relevant features. Those tests included two of the bots that we presumed to have more relevant features while the third bot rotated between the other three bots ml_rdeep, under and over.

The quantitative tests used over and under bots to deduce whether the quantity of features, or lack thereof, had any impact on the performance.

Each of the parts contained three tournaments.

5 Results

5.1 The quantitative test

Table 1. Results of the experiment quantitative 1

Player	# Features	Points	Winner
ml_under_feat	5	286	
ml_rdeep	10	381	Winner
ml_over_feat	12	292	

Table 2. Results of the experiment quantitative 2

Player	# Features	Points	Winner
ml_final_points	10	374	
ml_over_feat	12	313	Winner
ml_under_feat	5	263	

Table 3. Results of the experiment quantitative 3

Player	# Features	Points	Winner
ml_fine_tuned_feat	8	379	Winner
ml_over_feat	12	282	
ml_under_feat	5	289	

5.2 The qualitative test

Table 4. Results of the experiment qualitative 1

Player	# Features	Points	Winner
ml_rdeep	10	325	
ml_final_points	10	282	
ml_fine_tuned_feat	8	344	Winner

Table 5. Results of the experiment qualitative 2

Player	# Features	Points	Winner
ml_under_feat	5	229	
ml_final_points	10	316	
ml_fine_tuned_feat	8	373	Winner

Table 6. Results of the experiment qualitative 3

Player	# Features	Points	Winner
ml_over_feat	12	293	
ml_final_points	10	353	
ml_fine_tuned_feat	8	390	Winner

5.3 Test statistics

All data are collected using the simple random method. Performing test statistic analysis:

$$\alpha = 0.05$$

$$H_0 : P_1 = P_2 = P_3$$
$$H_a : P_1 \neq P_2 \neq P_3$$

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E)^2}{E}$$

For the qualitative test:

$$\chi_1^2 = \frac{325 - 317.5}{317.5} + \frac{282 - 324.5}{324.5} + \frac{344 - 309}{309} = 9.602$$

$$\chi_2^2 = \frac{229 - 294}{294} + \frac{316 - 311.5}{311.5} + \frac{373 - 312.5}{312.5} = 26.145$$

$$\chi_3^2 = \frac{293 - 355.5}{355.5} + \frac{353 - 340.5}{340.5} + \frac{390 - 340}{340} = 18.8$$

For the quantitative test:

$$\chi_1^2 = \frac{286 - 307.5}{307.5} + \frac{381 - 319.5}{319.5} + \frac{292 - 332}{332} = 18.1$$

$$\chi_2^2 = \frac{374 - 323.5}{323.5} + \frac{313 - 329}{329} + \frac{263 - 297.5}{297.5} = 12.66$$

$$\chi_3^2 = \frac{379 - 323}{323} + \frac{282 - 329}{329} + \frac{289 - 298}{298} = 16.7$$

The significance level redirect us to the Chi-square table, on the line with degrees of freedom equal to 2.

$$(r - 1)(c - 1) = 2$$

The critical value ends up to be 5.991.

Because each of the values obtained are higher than the critical value from the table we can assess the rejection of the null hypothesis, therefore be sure that proportions are far enough from being equal and their difference did not occur by chance.

6 Findings

The values we obtained from the test statistics prove the lack of chance in our results. Scores obtained by each single bot in our tournaments, turned out to be somehow relevant (because rejecting the null hypothesis, which in our case was tested on homogeneity of win ratio points, does not say anything about the alternative one) and determined by the changes we made in their implementations. We can therefore infer from this first conclusion that the results yielded by the quantitative tests confirm that the number of features do not substancially impact the performances. The second phase of the assessment, again supported by the test statistics, concluded what was in principle the expected outcome: a bot with hand picked features outperforms the others, maintaining the amount of them roughly constant. In the future more tests would need to be run, hence more data would need to be generated in order to draw complete informations about the performances of the machine learning agents we are considering.

7 Conclusions

References

1. Openai, `https://openai.com/five/`
2. Scikit, `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`
3. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artificial intelligence **97**(1-2), 245–271 (1997)
4. Campbell, M., Hoane Jr, A.J., Hsu, F.h.: Deep blue. Artificial intelligence **134**(1-2), 57–83 (2002)
5. Cohen, S., Dror, G., Ruppin, E.: Feature selection via coalitional game theory. Neural Computation **19**(7), 1939–1961 (2007)
6. Domingos, P.: A few useful things to know about machine learning. Communications of the ACM **55**(10), 78–87 (2012)

7. Gheyas, I.A., Smith, L.S.: Feature subset selection in large dimensionality domains. Pattern recognition **43**(1), 5–13 (2010)
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of machine learning research **12**(Oct), 2825–2830 (2011)
9. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484 (2016)

Appendix A

Milestones

Milestone 1

Question 1) Which of the three default bots does the best? Add the output to your report.
The bot that performs best is rdeep.
Question 2) Have a look at the code: what strategy does the bully bot use?
Based on our observations of the code the bot seems to play the first trump card it finds in its hand, which as per the structure of the game engine, is going to be highest trump card. If none are present then the bot plays either the highest matching suit or any other.
Question 3) For our game, this strategy would be no good. Why not?
As it relates to Schnapsen, using the hill climbing heuristic means maximizing the points earned in the next round, which may require using the highest value trump suit card; using that card later on in the game may actually lead to a higher ratio than using it in the next move.
Question 4) If you wanted to provide scientific evidence that rdeep is better than rand, how would you go about it?
One way to get scientific evidence would be to run a hypothesis test on the proportion of wins for rdeep and rand from more than 30 games. Our claim would be that the proportion of rdeep winning against rand to be higher. The null hypothesis would be that the proportion of wins is rdeep = rand and the alternative rdeep ¿ rand.
Question 5) Add your implementation of get_move() and the result of a tournament against rand to your report.
This is our implementation of get_move():

```
    def get_move(self, state):
        # type: (State) -> tuple[int, int]
        """
        Function that gets called every turn. This is where to implement
↪   the strategies.
        Be sure to make a legal move. Illegal moves, like giving an
↪   index of a card you
        don't own or proposing an illegal mariage, will lose you the
↪   game.
                TODO: add some more explanation
        :param State state: An object representing the gamestate. This
↪   includes a link to
```

```
            the states of all the cards, the trick and the points.
        :return: A tuple of integers or a tuple of an integer and None,
            indicating a move; the first indicates the card played in
↪   the trick, the second a
            potential spouse.
        """

        # All legal moves
        moves = state.moves()

        for move in moves:
            if move[0] != None and move[1] != None:
                return move
            elif state.get_opponents_played_card():
                if state.get_opponents_played_card() % 5 == 1 or
                ↪   state.get_opponents_played_card() % 5 == 0:
                    if Deck.get_suit(move[0]) == state.get_trump_suit():
                        return move
            #Trump exchange
            elif move[0] == None and move[1] != None:
                return move

        return moves[0]
```

This is the result of a tournament between rand our bot:

```
Results:
    bot <bots.rand.rand.Bot object at 0x7fa75032dbe0>: 45 wins
    bot <bots.mybot.mybot.Bot object at 0x7fa75033d3c8>: 55 wins
```

Question 6) Two simple bits of code are missing (indicated with lines starting with #IMPLEMENT). Read the whole script carefully and finish the implementation of the Bot. Run the experiment. It should output a file experiment.pdf containing a heatmap. Add this heatmap to your report, and discuss briefly what it means.

The heat map shows that when player1's low move probability is 60% or higher, then it tends to win more games against player2, regardless of player2's low move probability.

Figure 1 shows the heatmap we got as output:

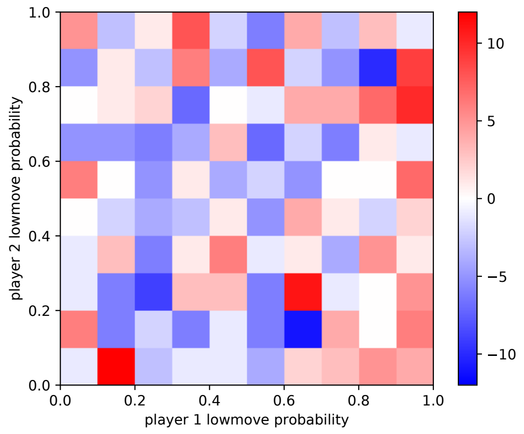

Fig. 1. Heatmap generated runnig the experiment.py file

Question 7) All you need to do to finish the minimax bot is to add one line of code on line 58. Take your time to really understand the minimax algorithm, recursion, and the rest of the code. Finish the bot, and let it play against rand (use flag to start in phase 2). Add the line you wrote and the results of the tournament to the appendix of your report.

```
58                  # IMPLEMENT: Add a recursive function call so that 'value'
                    ↪  will contain the
59                  # minimax value of 'next_state'
60                  value, m= self.value(next_state,depth = depth + 1)
```

Question 8) Once again, crucial parts of the implementation are missing. Finish the implementation of the alphabeta bot.
The script check_minimax.py lets you see if you implemented alphabeta and minimax correctly
What does it do? Run it and add the output to the appendix of your report.
The script compares the different move choices chosen by the minimax and alphabeta bots. It also compares the runtime.
This is the output we got:

```
Agreed.
Agreed.
Agreed.
Agreed.
Agreed.
Agreed.
Agreed.
Agreed.
Agreed.
Done. time Minimax: 0.01399842898050944, time Alphabeta:
↪  0.004466931025187175.
Alphabeta speedup: 3.1337911648015373
```

Question 9) What heuristic do these implementations use? Try to come up with a better one. Implement it and see how it does.
The default heuristic uses the proportion of total points of the player with the total points assigned to both players. We think adding pending points would improve the heuristic (like marriage pair).

Milestone 2

Question 1) Add a clause to the knowledge base to that it becomes unsatisfiable. Report the line of code you added.

```
kb.add_clause(~B,~C)
```

Question 2) Exercise 8 of this week's work session on logical agents contained the following knowledge base: $\neg A \implies B, B \implies A, A \implies (C \wedge D)$. Convert it to clause normal form, and write a script that creates this knowledge base. Print out its models and report them. As seen in the exercise, the knowledge base entails $A \wedge C \wedge D$. What does that say about the possible models for the knowledge base?

```
{A: True, D: True, C: True, B: False}
{A: True, D: True, C: True, B: True}
True
```

All possible models for the knowledge base will have to have A, C and D true, whereas B can be either True or False.
Question 3) What does this mean for the values of x and y? Are there any integer values for x and y that make these three constraints true?
It means that the values for x and y have to satisfy all three of those clauses. An integer value that would make these constraints true is the number 2.
Question 4) If we know that $x + y < 5$ must be false, as in the second model, we know that $x + y \geq 5$ must be true. Write each of these three models as three constraints that must be true.
(1) $[x = y] = True, [x + y > 2] = True, [x + y < 5] = True.$

(2) $[x = y] = True, [x + y > 2] = True, [x + y \geq 5] = True.$
(3) $[x = y] = True, [x + y \leq 2] = False, [x + y < 5] = True.$

Question 5) Write this down in propositional logic first (two sentences). Then convert this to clause normal form. Now create a knowledge base with the required clause and report which models are returned. The script test3.py does almost all the work for you. All you need to do is fill in the blanks.

Two sentences: S1 Since S2 is a sentence and S3 is a sentence then by definition S2 OR S3 is a sentence.
This is the CNF conversion:

(i) $(A \wedge B) \vee (C \wedge D)$
(ii) $((A \wedge B) \vee C) \wedge ((A \wedge B) \vee D)$
(iii) $(A \vee C) \wedge (B \vee C) \wedge (A \vee D) \wedge (B \vee D)$

S1: $x == y$
S2: $x + y > 2 \wedge x + y < 5$
S3: $x + y > -5 \wedge x + y < -2$
A: $x + y > 2$
B: $x + y < 5$
C: $x + y > -5$
D: $x + y < -2$

This is the output of test3.py:

```
{[x + (-y) == 0]: True, [x + y > 2]: False, [x + y > -5]: True, [x + y <
↪  -2]: True, [x + y < 5]: True}
{[x + (-y) == 0]: True, [x + y > 2]: True, [x + y < -2]: False, [x + y <
↪  5]: True, [x + y > -5]: True}
True
```

Question 6) Look at the code in test4.py. (The long list in the beginning is just the variable instantiation, the real modelling starts at line 50.) Extend the document with the knowledge for a strategy PlayAs, always playing an As first. Check whether you can do reasoning to check whether a card is entailed by the knowledge base or not.

The same reasoning can be applied to any listed state, a sentence is unsatisfiable if it is true in none of the models: in the PlayJack strategy, a unit clause J6 would find no true value in our model, hence considered to be unsatisfiable. Techniques like Modus Ponens and Resolution can be applied to demonstrate the entailment by refutation.

Question 7) Build a more complex logical strategies. For examples, you can define the notion of a cheap card, as being either a jack, king or queen, and devise a strategy that plays cheap card first. Test whether you can use logical reasoning to check whether the correctness of a move w.r.t. this strategy is entailed by your knowledge base.

We implemented the strategy ”Play cheap” taking into account the trump suit (otherwise the played card would not be cheap anymore). Considering the Jack’s

example, we have the following strategy (applied to C2): $PC2 \iff C2$, that once reduced in CNF we obtein $(PC2 \vee \neg C2) \wedge (\neg PC2 \vee C2)$.
We want to apply this strategy if and only if the current trump suit is not clubs, so we form the following statement: $(PC2 \iff C2) \iff \neg C$.
After the conversion to CNF, this is what we added to our knowledge base:

```
kb.add_clause(PC2, C2, ~C)
kb.add_clause(PC2, ~PC2, ~C)
kb.add_clause(~C2, C2, ~C)
kb.add_clause(~C2, ~PC2, ~C)
kb.add_clause(C, ~PC2, C2)
kb.add_clause(C, ~C2, PC2)
```

Once the game starts and we know the suit of the trump card, we can dinamically add it to our knowledge base:

```
# Example is the trump suit is Jack
kb.add_clause(C)
kb.add_clause(~D)
kb.add_clause(~H)
kb.add_clause(~S)
```

Note 1: The same concepts can be applied to the rest of the other cards.
Note 2: After few attempts and sessions with differents teaching assistants we were not able to use our knowledge base in order to let the bot use the above strategy. The results that will follow are based on the same concepts as above but they have been hardcoded.

Question 8) Replace the knowledge and strategy you modelled in test5.py into load.py. You might want to add some print statements to check whether kbbot now really follows your strategy. Provide an example game where you show that the strategy works.

```
   Start state: The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 0, pending: 0
The trump suit is: S
Player 1's hand: AC AD QH QS JS
Player 2's hand: KC QC QD JH 10S
There are 10 cards in the stock

player1: <bots.rand.rand.Bot object at 0x10961d198>
player2: <bots.kbbot.kbbot.Bot object at 0x10961d048>
*   Player 1 plays: JS
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 0, pending: 0
```

```
The trump suit is: S
Player 1's hand: AC AD QH QS JS
Player 2's hand: KC QC QD JH 10S
There are 10 cards in the stock
Player 1 has played card: J of S

Strategy Applied
*   Player 2 plays: QD
The game is in phase: 1
Player 1's points: 5, pending: 0
Player 2's points: 0, pending: 0
The trump suit is: S
Player 1's hand: AC AD 10D QH QS
Player 2's hand: KC QC JH AS 10S
There are 8 cards in the stock
```

Question 9) Produce a new knowledge based bot that allows you to play knowledge based strategies consequently. Compare the performance w.r.t. simpler knowledge based bot (with the individual strategies) and other bots.

```
Results:
    bot <bots.kbbot.kbbot.Bot object at 0x7f281cc4a8d0>: 66 wins
    bot <bots.rand.rand.Bot object at 0x7f281cc4a940>: 34 wins
```

Milestone 3

Question 1) Fill in the missing code (all the '???' lines) and run the training script. Run a tournament between rand, bully and ml. Show the code you wrote, and the result of the tournament.

```
            # IMPLEMENT: Add a function call so that 'value' will
            # contain the predicted value of 'next_state'
            # NOTE: This is different from the line in the
            ↪  minimax/alphabeta bot
            value = self.heuristic(next_state)
```

```
def features(state):
    # type: (State) -> tuple[float, ...]
    """
    Extract features from this state. Remember that every feature vector
↪   returned should have the same length.
```

```

    :param state: A state to be converted to a feature vector
    :return: A tuple of floats: a feature vector representing this
↪   state.
    """

    feature_set = []

    # Add player 1's points to feature set
    p1_points = state.get_points(1)

    # Add player 2's points to feature set
    p2_points = state.get_points(2)

    # Add player 1's pending points to feature set
    p1_pending_points = state.get_pending_points(1)

    # Add plauer 2's pending points to feature set
    p2_pending_points = state.get_pending_points(2)

    # Get trump suit
    trump_suit = state.get_trump_suit()

    # Add phase to feature set
    phase = state.get_phase()

    # Add stock size to feature set
    stock_size = state.get_stock_size()

    # Add leader to feature set
    leader = state.leader()

    # Add whose turn it is to feature set
    whose_turn = state.whose_turn()

    # Add opponent's played card to feature set
    opponents_played_card = state.get_opponents_played_card()
```

```
Results:
    bot <bots.rand.rand.Bot object at 0x10ff0a1d0>: 11 points
    bot <bots.bully.bully.Bot object at 0x10ff0a240>: 9 points
    bot <bots.ml.ml.Bot object at 0x10ff0a940>: 27 points
```

Question 2) Re-run the tournament. Does the machine learning bot do better? Show the output, and mention which bot was used for training.
We trainded two other bots observing kbbot and rdeep, respectivevely.
This is the result of a tournament played between the bots ml_kbbot (a machine learning based bot trainded observing a pure kbbot player), rand and bully.

```
Results:
    bot <bots.ml_kbbot.ml_kbbot.Bot object at 0x10cc67198>: 22 points
    bot <bots.rand.rand.Bot object at 0x1172186d8>: 7 points
    bot <bots.bully.bully.Bot object at 0x1172186a0>: 25 points
```

This is the result of a tournament played between ml_rdeep (a machine learning based bot trainded observing a pure rdeep player), rand and bully.

```
Results:
    bot <bots.ml_rdeep.ml_rdeep.Bot object at 0x10ea7b1d0>: 29 points
    bot <bots.rand.rand.Bot object at 0x11dfebb38>: 8 points
    bot <bots.bully.bully.Bot object at 0x11dfebf98>: 14 points
```

Question 3) Make three models: one by observing rand players, one by observing rdeep players, and one by observing one of the ml players you made earlier. Show the results in your appendix.

```
Results:
    bot <bots.ml.ml.Bot object at 0x10b3e0da0>: 21 points
    bot <bots.ml_rdeep.ml_rdeep.Bot object at 0x1176057b8>: 11 points
    bot <bots.ml_ml_rdeep.ml_ml_rdeep.Bot object at 0x10b3f0940>: 21
    ↪  points
```

Question 4) Add some simple features and show that the player improves.
We tried to add an extra feature and it was the number of trump cards.

```python
def get_num_trump_cards(trump_suit, hand):
    num_trump_cards = 0
    for card in hand:
        index = int(card / 5)
        if (index == 3 and trump_suit == "S") or (index == 2 and
        ↪  trump_suit == "H") or (index == 1 and trump_suit == "D") or
        ↪  (index == 0 and trump_suit == "C"):
            num_trump_cards += 1
    return num_trump_cards
    # Add the number of trump cards in player's hand
    num_trump_cards = get_num_trump_cards(trump_suit, state.hand())

    feature_set.append(num_trump_cards)
```

This is the result of a tournament between the new bot (with the number of trump cards as a new feature) and other good performing bots:

```
Results:
    bot <bots.rdeep.rdeep.Bot object at 0x100ec81d0>: 28 points
    bot <bots.ml_rdeep.ml_rdeep.Bot object at 0x100ed0588>: 33 points
    bot <bots.ml_ml_rdeep.ml_ml_rdeep.Bot object at 0x110442c18>: 22
    ↪  points
    bot <bots.ml_trump.ml_trump.Bot object at 0x10d27dba8>: 12 points
```

As we can see, the results are not really encouraging since the new bot could get only 12 points being the worst player in the tournament.
So we change our path and we tried to remove a feature rather than adding a new one. We decided to remove the *whose_turn* feature and we played another tournament.
Here the result:

```
Results:
    bot <bots.rdeep.rdeep.Bot object at 0x10e2a80b8>: 36 points
    bot <bots.ml_rdeep.ml_rdeep.Bot object at 0x10e2a8c88>: 32 points
    bot <bots.ml_ml_rdeep.ml_ml_rdeep.Bot object at 0x10e2c9400>: 29
    ↪  points
    bot <bots.ml_trump.ml_trump.Bot object at 0x11685d198>: 17 points
    bot <bots.ml_no_whose_turn.ml_no_whose_turn.Bot object at
    ↪  0x10e33a198>: 35 points
```

As we can see, we got a significant improovement and now our new bot could get 35 points being the second best player in the tournament. Again, the bot that takes into account the number of trump cards could get only 17 points being on more time the worst player of the tournament.