b
UNIVERSITÄT
BERN

Data Science Project

JassBot
Project Report

June 15, 2023

Raphael Ziegler
raphael.ziegler@gmail.com

Abstract

This project report presents the JassBot, a data science project focused on training an intelligent agent to play the Swiss card game Jass. The project aims to train a reinforcement learning algorithm to learn and implement an optimal game strategy. The project uses the two-player Handjass variant and excludes elements such as Marriage and Melds for simplicity. The report explores different reinforcement learning algorithms, including Monte Carlo Control, SARSA(λ), and Value Approximation. The game environment, agents, actions, and various card encoding methods are described. The report concludes by analyzing the results obtained from different encoding approaches.

While macro patterns are observed, no apparent, detailed patterns emerge. The findings indicate the potential for using reinforcement learning in developing game strategies and suggest avenues for future enhancements to the JassBot project.

Contents

1 Project Objectives

For this project, a JassBot is created and trained to play against itself. For this purpose, a reinforcement learning algorithm will be trained to find the best game strategy. The baseline will be the random pick of a playable card for comparison.

1.1 Deck [1]

Jass is played with a deck of 36 cards (Ace, King, Queen, Jack, 10, 9, 8, 7, 6) Swiss-French or Swiss-German cards (Ass, König, Ober, Under, Banner (= 10), 9, 8, 7, 6). The Swiss-German cards use Swiss suits, a variant of German suits, and have a distinctive design.

The game is traditionally played with Swiss-suited cards east of the Brünig-Napf-Reuss line [2] and with French cards in western Switzerland.
The Swiss suits are Rosen (roses), Eicheln (acorns), Schellen (bells), and Schilten (shields)

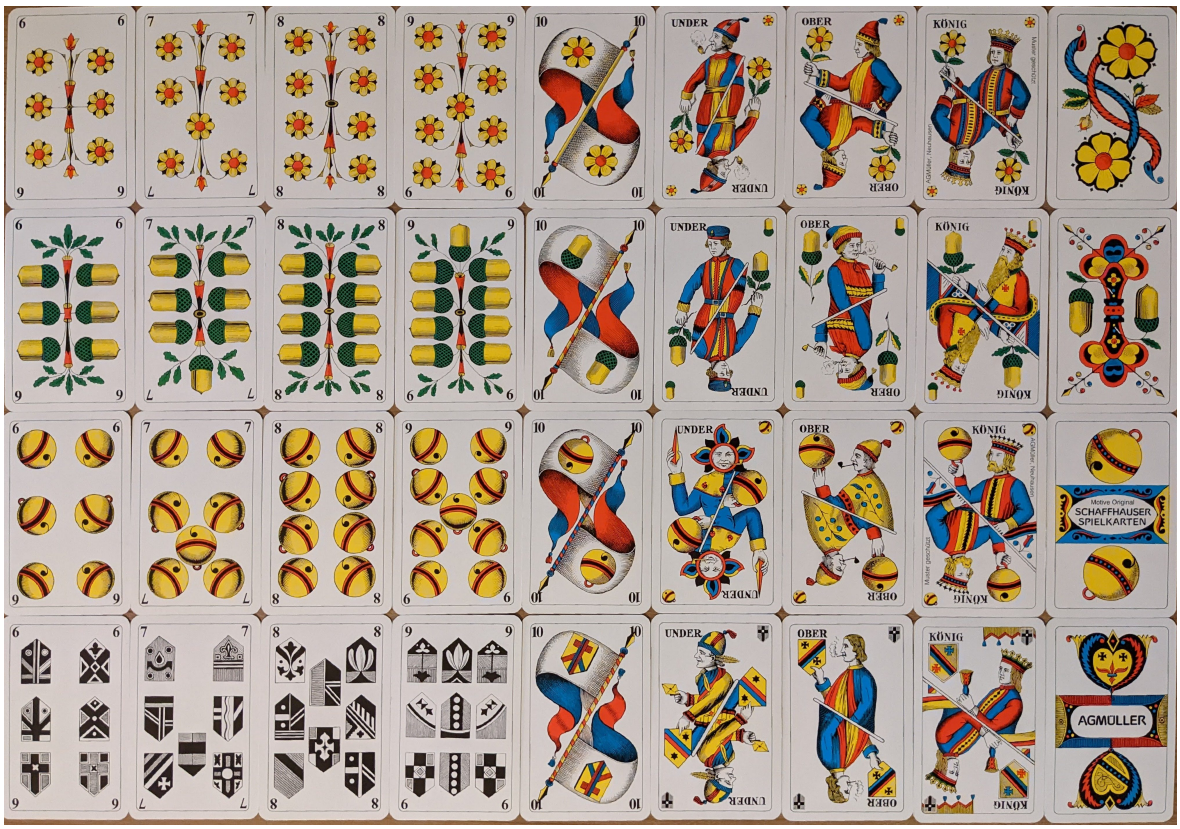


FIGURE 1.1: Swiss German Cards

1.2 Game variants

Jassen is the collective term for different game variants with different numbers of players. In most variants, each player receives nine cards (hand). The most popular variant is the Schieber, where two teams of 2 players play against each other until one team reaches the points needed to win. Another well-known game variant is the Differenzler (known from the TV show Samstagsjass). Here four players play against each other; before each deal, each player announces how many points he intends to make in this round. The difference to the actual score is noted as penalty points and added at each round. The player with the least penalty points wins. Another game variant is the Handjass, which can be played in pairs, threes, or fours. In each round, the player who takes the most points scores a stroke, and anyone who takes fewer than 26 points gets a Null - sometimes known as a Herdöpfel (potato) - which is worth minus one stroke. The player who achieves a net score of 7 strokes wins and retires from the game: the loser is the last player who fails to reach seven strokes. [3]

1.3 Rules [1]

Jass is a game of points scored for three features: Stöck, Wiis, and Stich, respectively, "marriages, melds, tricks.

1.3.1 Tricks

The trump Jack, also called Puur, counts 20 and is the highest card in the game. The trump Nine, or Nell, Näll, is the second-best card. Plain suit numerals below 10 count nothing. The total value of all counters in the pack is 152, that is, 62 in trumps plus 30 in each plain suit. Winning the last trick scores an additional 5 points. Hence the total possible for the third scoring feature, "tricks," usually is 157 points.

The rank of the cards, from highest to lowest, and their values in card points are shown in the following table:

TABLE 1.1: Card Values - trump games

Card Values											
Plain suit rank			A	K	O/Q	U/J	10	9	8	7	6
Value	20	14	11	4	3		10		0	0	0
Trump suit rank	J/U	9	A	K	O/Q	U/J	10	9	8	7	6

The no-trumps game called Obenabe and Undenufe, in which the ranks are reversed, are shown in the following table:

TABLE 1.2: Card Values - no-trump games

Card Values																		
Obenabe										Undenufe								
Rank	A	K	O/Q	U/J	10	9	8	7	6	6	7	8	9	10	U/J	O/Q	K	A
Value	11	4	3	2	10	0	8	0	0	11	0	8	0	10	2	3	4	0

1.3.2 Marriage

Marriage (Stöck): A marriage is the holding in one hand of the König and Ober (King and Queen) of trump. Its holder claims it upon the second of them to a trick. Its score of 20 is recorded as if made before those for melds and tricks, even though it is not revealed until after melds have been declared.

1.3.3 Melds

Meld (Wys or Weis): A meld is a suit sequence of three or more cards or a quartet of Aces, Kings, Queens, or Jacks scoring as follows:

- Four Jacks: scores 200
- Four 9's: scores 150
- Five or more in suit sequence: scores 100
- Four A, K, Q, 10: scores 100
- Four in suit sequence: scores 50
- Three in suit sequence: scores 20

A card may not be used in two melds at once, though the trump King or Queen may belong to a meld in addition to being married; that is, a player holding four Kings and a sequence of four to the Ace or King would count only 100 for Kings, not also 50 for the sequence.

1.4 Chosen Game and Rules for this Project

For this project, a two-player Handjass was used, hoping to minimize the game's complexity. It was also decided not to use Marriage and Melds, as these scores are obtained randomly and do not contribute to learning a game strategy. A further decision was not to implement the five scores for the last-round winner to minimize the programming effort. If this project finds a successful game strategy, the remaining game elements can be added in a later version.

2 Descriptive Statistics

The number of different possible hands is:

$$N = \binom{36}{9} = 94'143'280$$

Using all combinations in a reasonable time for training will not be feasible because the computational cost can not be satisfied with the hardware at hand. 1'000'000 hands were generated to be used for the descriptive statistics.

As expected, the cards are completely random, and no pattern can be seen. (Figure 2.1)

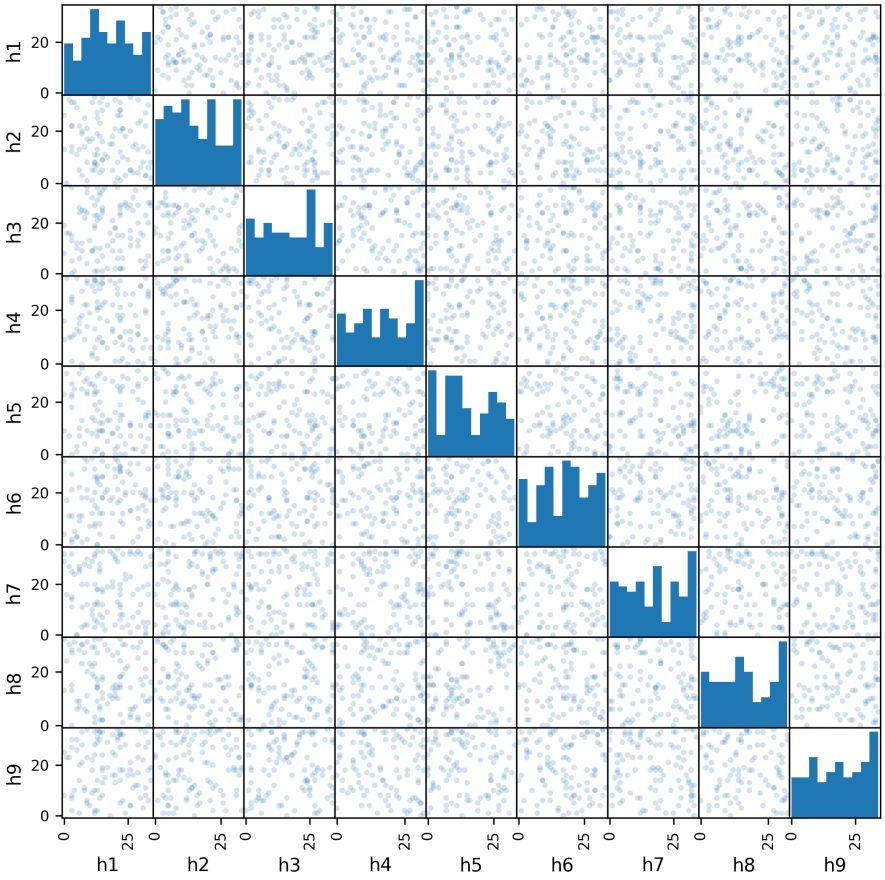


FIGURE 2.1: Scatter Matrix of 1'000'000 hands

3 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents should take actions in an environment to maximize the cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. [4]

For this project, It was looked for a well-documented RL example so it can be customized for the project's needs. The choice fell on the article "Playing Cards with Reinforcement Learning" [5], which describes Reinforcement Learning and the principles behind it; also, the Jupyter Notebook is provided [6]. The article uses the game Easy 21 (a variant of BlackJack) to explain the basics of reinforcement learning and three approaches for estimating the best playing strategy.

The goal of the following algorithms is to learn the Q value function. The Q value is the estimated potential reward in a given game situation.

3.1 Monte Carlo Control (MCC)

Monte Carlo means that we are going to play game sequences/episodes (play a round of Jass). The **Control** part means that we are going to find the optimal policy (best action to pick to maximize our winning chances). After each round, the Q value gets updated. If the played game was a win (resp. loss), the values of the state-action pair should be increased (resp. decreased)

3.2 SARSA(λ)

SARSA is the acronym of State, Action, Revard, State', Action', which is related to how the algorithm updates the Q value function. The λ is a model's parameter called the trace factor, scaling from 0 to 1. If λ=1, we have the classic MCC approach; if λ=0, it is a Temporal-Difference learning update. This means we adjust our current Q value to match a future Q value estimation better.

3.3 Value Approximation (VA)

Value Approximation means that we will approximate the Q value function rather than compute it for all the possible state-action pairs.

3.4 Adjustments for the JassBot

For the JassBot, some adjustments to the Easy 21 example were necessary. First, a "new" game engine (environment) had to be written to interact with the RL algorithms. The number of possible actions was kept the same because, in Jass (and Easy 21), you can only take two actions (take the trick or not take the trick). The code for the RL algorithms had to be

adapted only slightly and could be copied as far as possible from the example. The value approximation was not pursued further since different typical game situations must be represented. Because of the more than 90 million possible hand combinations, hand coding of significant game situations did not seem purposeful.

3.4.1 Game Environment

The game environment implements the Jass mechanisms. Here the deck of cards and the two players are created.
A game lasts until one player has reached 7 points. A game consists of several rounds (also called episodes). A round consists of 9 cards per player (All nine cards in the hand are played). Different functions are implemented (see 3.4.2) to decide which card to play.

3.4.2 Agents and Actions

An AGENT represents the two players, and this agent can choose different functions for selecting a card to play, depending on the situation.

For the RL of the first player, two actions are available (take: take the trick or leave: leave the cards on the table/let the other player win the cards).

- take will always play the strongest card possible
- leave will always play the weakest card possible

For the second player, no RL is used, here the strategy can be defined once in the code, which will always be used (e.g., random, probabilistic, card selection based on the previously learned Q values)

3.4.3 Card encoding

With different encodings, it was examined whether the learning success can be improved if more information is encoded in the numbers.
Several methods have been implemented for encoding the cards/game states:

- simple: Each card has a fixed number (the Trump cards have the highest numbers).
- linear numbering: from weak (0) to strong (26)
- card strength: Different cards can have the same value
- digits1: see Figure 3.1a
- digits2: see Figure 3.1b

```
"""
first digit:            second digit:
------------            -------------
1 if Rose               0 if card 6
2 if Eichel             1 if card 7
3 if Schellen           2 if card 8
4 if Schilten           3 if card 9
                        4 if card Banner
                        5 if card Under
                        6 if card Ober
                        7 if card König
                        8 if card Ass

third digit:
------------
0 if card suit an played card suit doesn't march
1 if played card is None (we play first)
2 if card suit and played suit match
3 if card suit is trump suit and trump is played
4 if card suit is trump suit
```

(A) Digits1

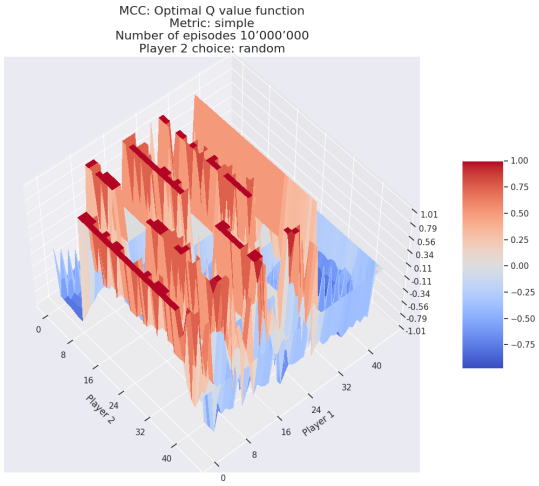
```
"""
first digit:
------------
1 if Rose
2 if Eichel
3 if Schellen
4 if Schilten

second digit:
-------------
0 if card 6
1 if card 7
2 if card 8
3 if card 9
4 if card Banner
5 if card Under
6 if card Ober
7 if card König
8 if card Ass

third digit:
------------
0 if we play first
1 if no trump was played
2 if trump was played

fourth digit:
-------------
0 if player 2 takes the trick
1 if take the trick
2 if play suit
3 if discard
4 if we played first
"""
```

(B) Digits2

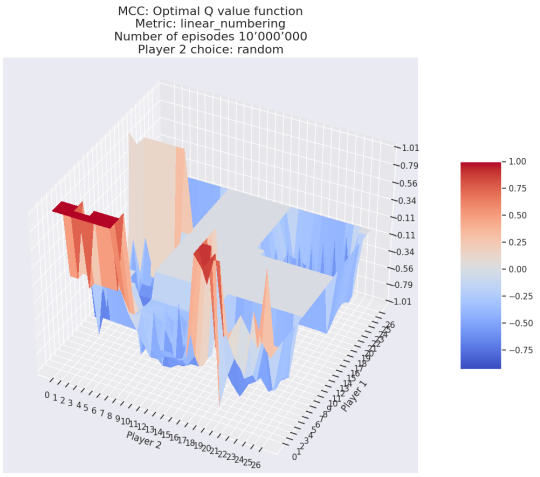
FIGURE 3.1: Card encoding

4 Results

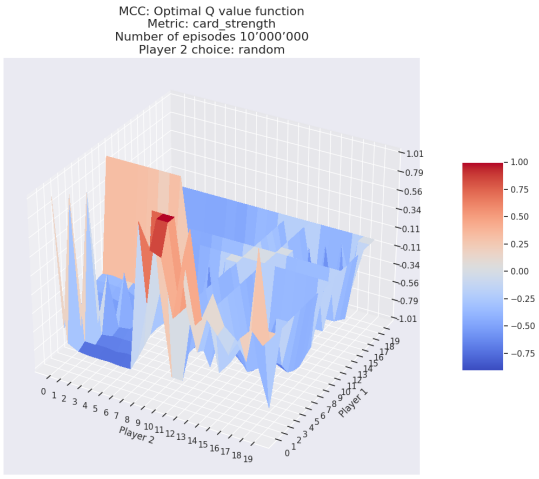
The results of the different card encodings (MCC Figure 4.1 and SARSA Figure 4.2) show patterns in the macro range, but no clear pattern can be seen in detail. The macro pattern is explained by the fact that certain card combinations cannot occur and therefore have zero value. As expected, the different card encodings produce different patterns.



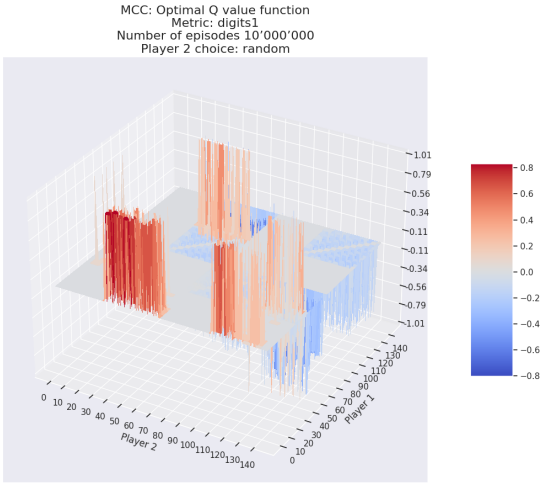
(A) Simple



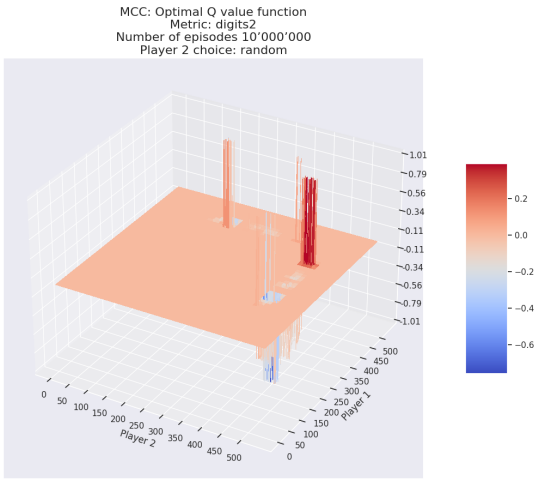
(B) Linear Numbering



(C) Card Strength

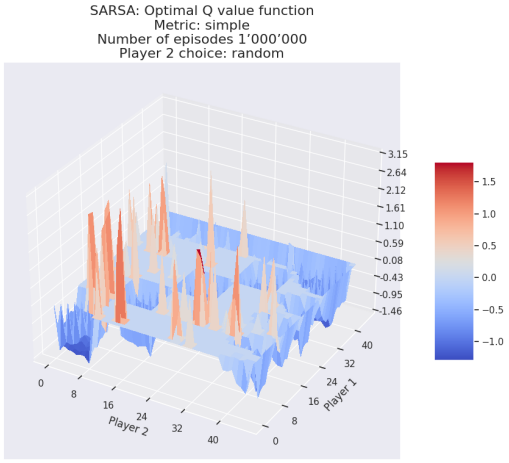


(D) Digits1

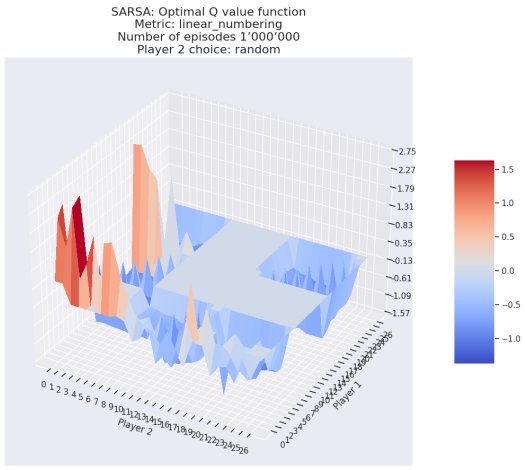


(E) Digits2

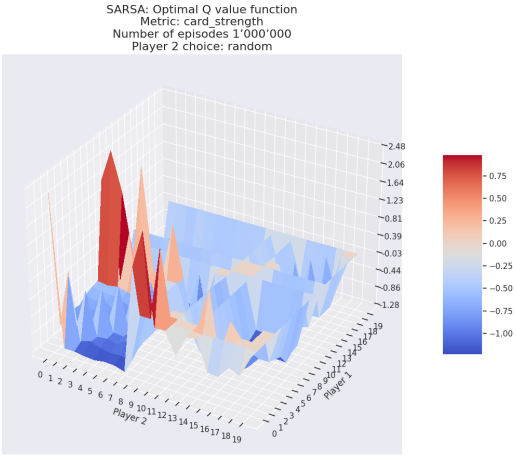
FIGURE 4.1: Monte Carlo Control: Q value function with different card encodings



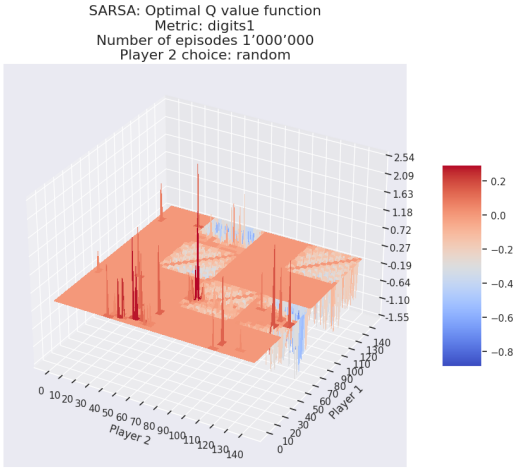
(A) Simple



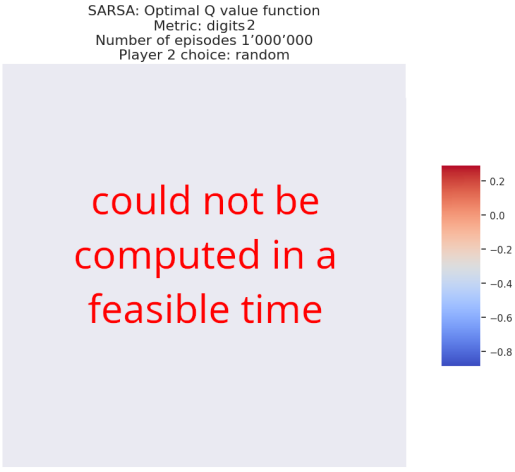
(B) Linear Numbering



(C) Card Strength



(D) Digits1



(E) Digits2

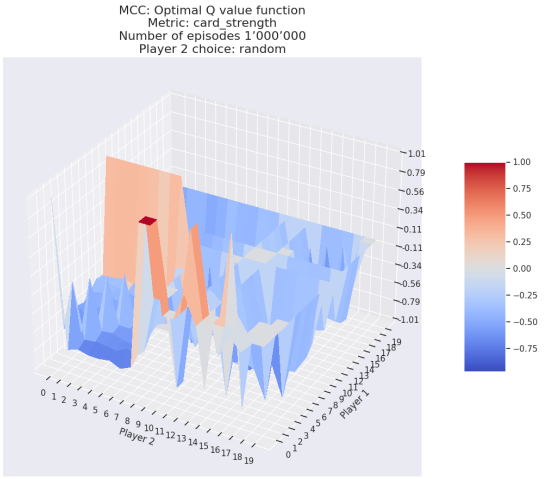
FIGURE 4.2: SARSA: Q value function with different card encodings

The two RL algorithms require vastly different execution times. MCC can execute more episodes in less time than SARSA. For the MCC plots, 10 million episodes were used; this

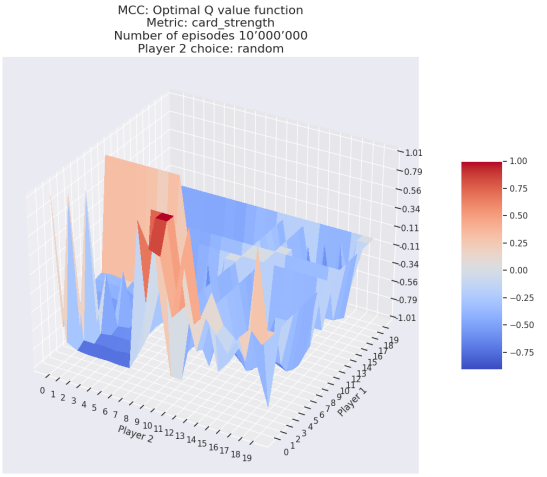
was not practical for SARSA since it requires many times more computation time, and the card encoding affects the computation times.

The two RL algorithms reveal similar patterns; MCC shows a more uniform Q value function due to the larger number of episodes. However, it can be seen that the SARSA algorithm shows a similar tendency. See Appendix B MCC and SARSA plots side by side with the same card encoding.

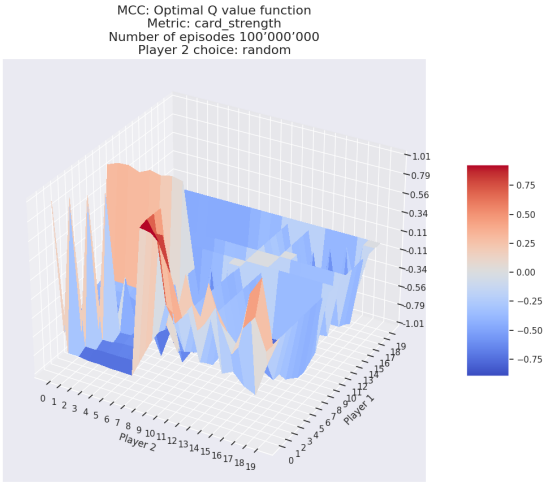
The development of the Q value function in dependence on the different numbers of episodes is shown in Figure 4.3. The greater the number of episodes, the more apparent the pattern. At 100 million episodes, a clear pattern can be seen.



(A) Card Strength with 1′ 000′ 000 episodes



(B) Card Strength with 10′ 000′ 000 episodes

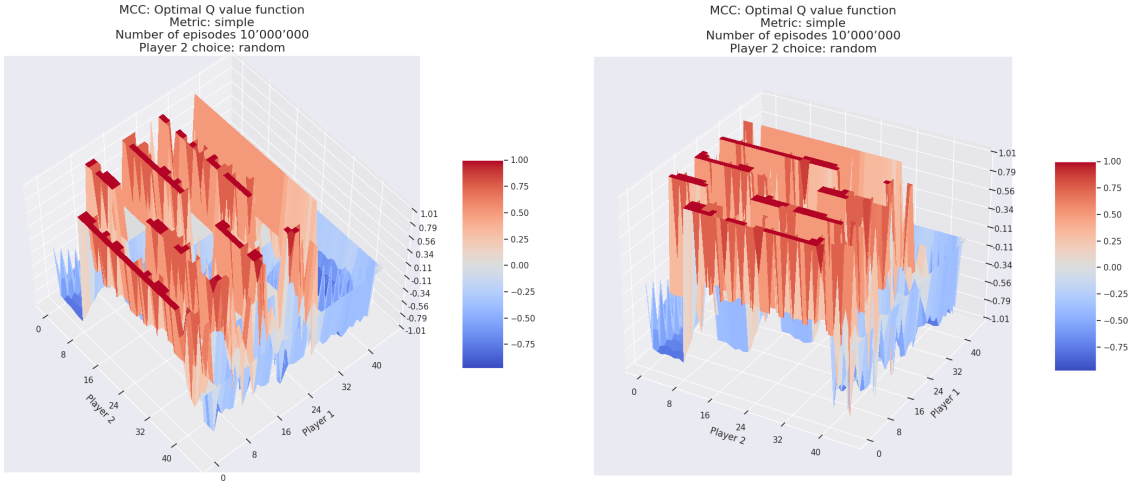


(C) Card Strength with 100′ 000′ 000 episodes

FIGURE 4.3: Monte Carlo Control: Evolution of Q value function with more episodes

A comparison with a Q value function where the agent can only randomly select a card shows the same pattern (4.4), confirming that the RL can not demonstrate learning success. This result confirms the "independent" test performed on a second game engine (Jupyter Notebook: dashboard.ipynb, see Github [7]). The test lets two agents play against each

other; AGENT 1 uses the learned Q-value function, and AGENT 2 randomly chooses a card. After a million games, AGENT 1 (Q value function) is in the lead with 51%.



(A) RL with options take or leave

(B) RL with option random

FIGURE 4.4: Comparison: MCC with different decision options

5 Conclusion

The two ML algorithms (MCC, SARSA) could not achieve any meaningful learning success, or more precisely, the learned Q value function does only achieve a 1% better result than a randomly played card.

The fact that not all possible hand variations were played through could also have contributed to the poor performance. The RL needed more opportunities to learn all the variants.

The considerations to encode the game states were only partially successful.

- One possible reason is that "only" two actions are available (take, leave). However, the action cannot be guaranteed to be executed (e.g., a take action cannot have a card in hand to take the trick).
- The changing of the dealer (the winner of the last round plays the first card in the current round) leads to the fact that the actions do not reflect the actual state of the game because the player who deals can only partially judge whether an action (take/leave) is possible.
- If the desired action was impossible, a playable card was randomly selected. This should have been considered a learning termination (terminal).

The example of Easy 21 was not transversally for Jass, which allows more game states. It should be possible to consider more than two game states:

- Consideration of trump suit
- Who plays the first card in a round (for the first and second players, there are two different strategies to learn).
- Does the player have the cards to decide if he can take or leave? Only if both possibilities exist can something be learned about the game strategy (take/leave).

The example shown in Figure 4.3c with 100 million episodes suggests we have been working with too few episodes. The illustrated case with the card encoding card strength is unsuitable with the current environment for the independent test since several cards can have the same value.
If the project is pursued further, either optimization must be found in the current Python code to speed up the execution time or more powerful hardware is needed.

Acknowledgements

I thank everyone who accompanied me during the CAS Applied Data Science.

Special thanks go to Sigve Haug and Mykhailo Vladymyrov, with whom we often had exciting conversations about all and everything and, of course, also in professional matters.

Furthermore, I would like to thank all the other lecturers/referees who always actively supported us and patiently answered our questions.

Another thank goes to my fellow students; seeing and exchanging ideas with them was always delightful.

Last but not least, I would like to thank Matyas Amrouche for writing the article *"Playing Cards with Reinforcement Learning"* [6] and providing the JupyterNotebook [5].

A Github Repository

Mentioning of essential files:

python

For running the code, the minimum required Python version is 3.10

Easy 21

- Easy 21 Jupyter Notebook

jassbot

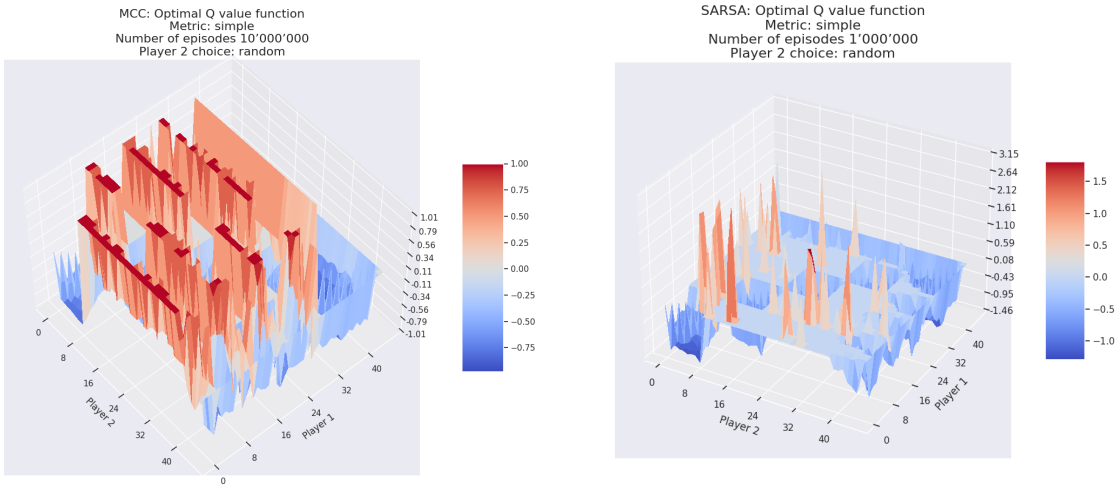
- config.py: Configure everything in this file (different card encodings), and for ease of use, start the RL and plot the Q value function.
- env.py: Jass environment for interaction with the RL algorithms
- ai_functionality.py: logic for deciding what card to play
- function.py: functions for card encoding
- plotting.py: functionality for the different plots
- monte_carlo_control.py: Monte Carlo Control functionality
- sarsa.py: SARSA functionality
- dashbord.ipynd: Jupyter Notebook to determine if we learned something (testing) - configuration made in the config.py file
- game_engine.py: Jass game-engine (environment) for the testing

Easy 21

- LaTeX source files

B MCC and SARSA plots side by side with the same card coding

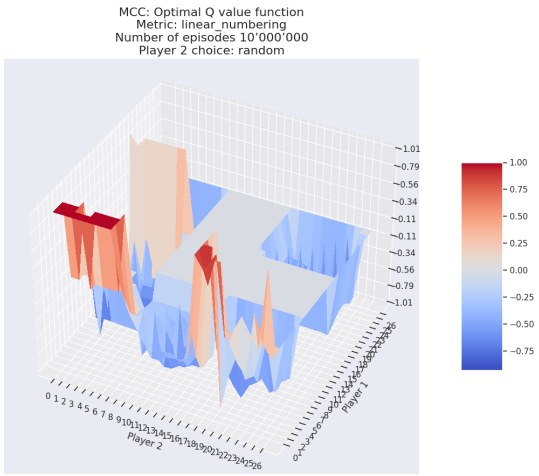
- MCC with 10′ 000′ 000 episodes
- SARSA with 1′ 000′ 000 episodes



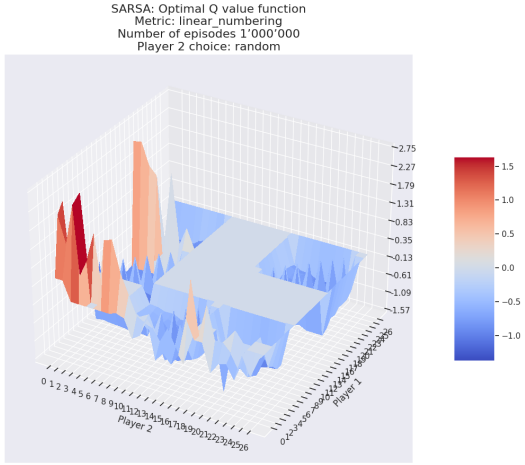
(A) MCC simple

(B) SARSA simple

FIGURE B.1: MCC and SARSA card encoding: simple

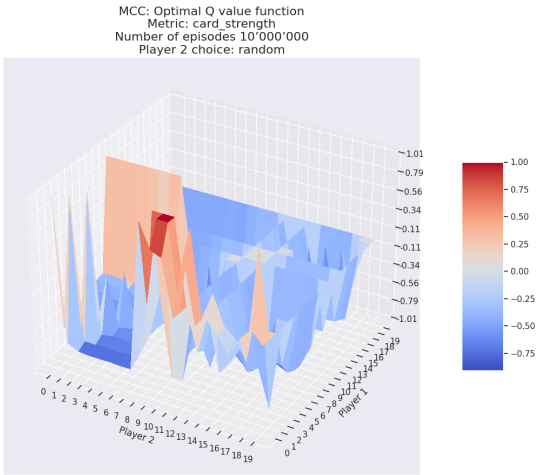


(A) MCC linear numbering

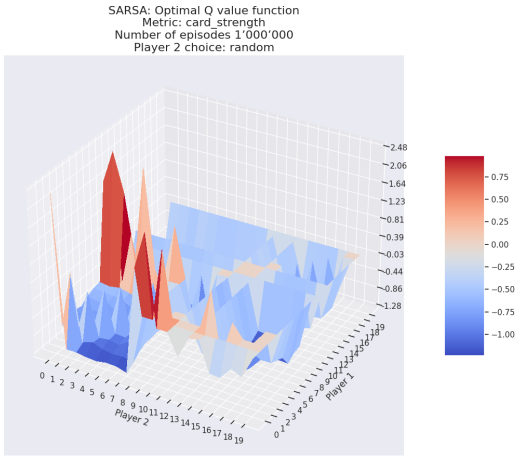


(B) SARSA linear numbering

FIGURE B.2: MCC and SARSA card encoding: linear numbering

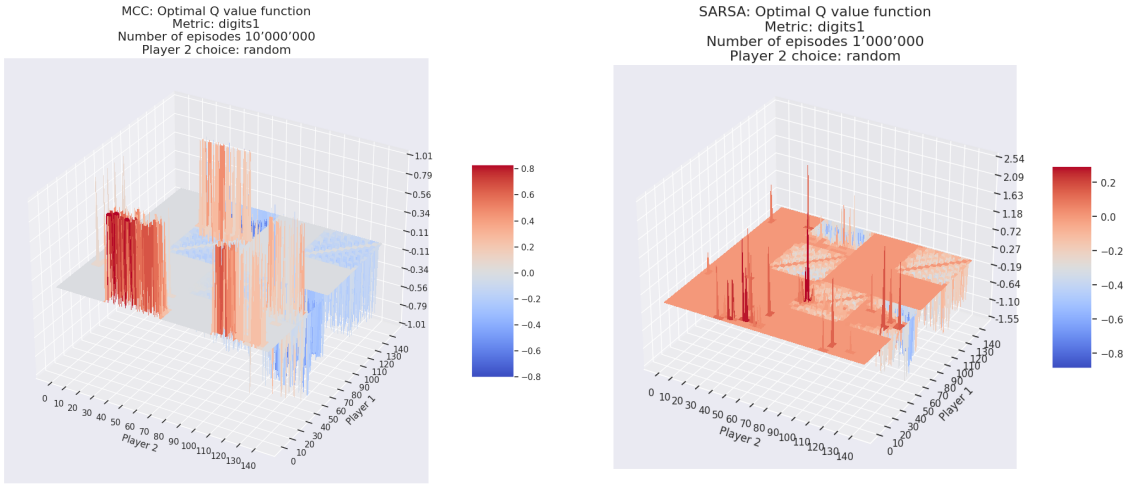


(A) MCC card strength



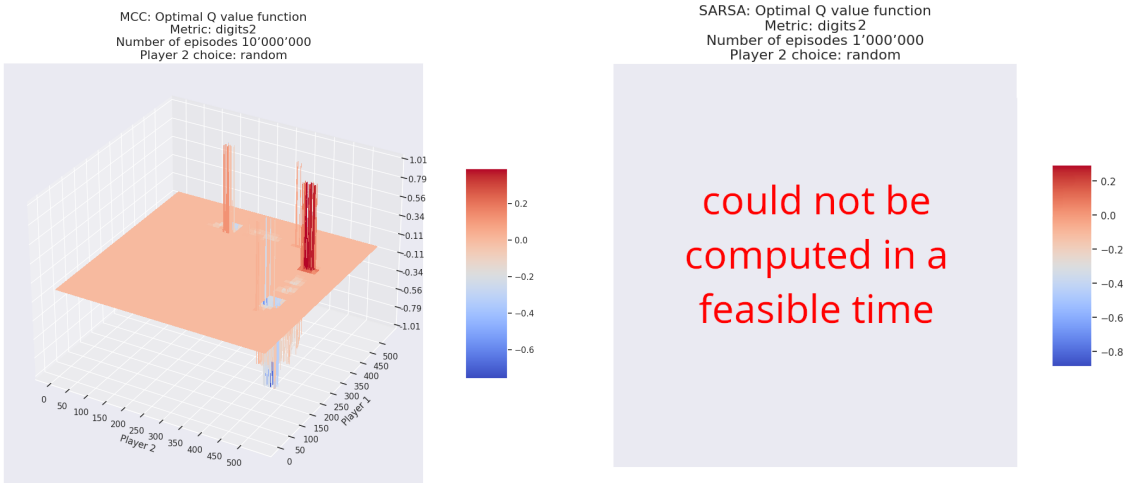
(B) SARSA card strength

FIGURE B.3: MCC and SARSA card encoding: card strength



(A) MCC digits1 (B) SARSA digits1

FIGURE B.4: MCC and SARSA card encoding: digits1



(A) MCC digits2 (B) SARSA digits2

FIGURE B.5: MCC and SARSA card encoding: digits2

Reference

[1] Jass. *in Wikipedia*. URL: https://en.wikipedia.org/wiki/Jass (visited on 05/24/2023).

[2] Brünig-Napf-Reuss line. *in Wikipedia*. URL: https://en.wikipedia.org/wiki/Br%C3%BCnig-Napf-Reuss_line (visited on 03/14/2023).

[3] Handjass. URL: https://www.pagat.com/jass/handjass.html (visited on 05/28/2023).

[4] Reinforcement learning. *in Wikipedia*. URL: https://en.wikipedia.org/wiki/Reinforcement_learning (visited on 05/18/2023).

[5] Matyas Amrouche. *Playing cards with Reinforcement Learning*. URL: https://towardsdatascience.com/playing-cards-with-reinforcement-learning-1-3-c2dbabcf1df0 (visited on 03/27/2023).

[6] Matyas Amrouche. *Github/Easy 21*. URL: https://github.com/Matyyas/Easy21 (visited on 03/27/2023).

[7] Raphael Ziegler. *JassBot*. URL: https://github.com/raphaelziegler/CAS-in-Applied-Data-Science/tree/main/final_project/python (visited on 06/15/2023).

List of Figures

List of Tables

List of Abbreviations

ML	Machine Learning
RL	Reinforcement Learning
MCC	Monte Carlo Control
SARSA	State, Action, Revard, State’, Action’
VA	Value Approximation