University of Malta

Faculty of Information and Communication Technology

ARI2204 – Reinforcement Learning

Blackjack RL Coursework 2021-2022

Faculty of ICT

Andrew Darmanin (95602L)

Edward Sciberras (274402L)

Scholastic year: 2021/22

Table of Contents

Overview and implementation of the RL algorithms developed

In this section, the implementation designs of every algorithm that was constructed will be discussed. However, before discussing the RL methods, firstly an understanding of the simulated BlackJack environment must be achieved.

Simulating a game of Blackjack

In this simplified version of Blackjack the game mechanics are the following:

- At the beginning of the round, the deck is restored and shuffled.
- The player gets two cards and the dealer gets dealt a card.
- The player has to decide to HIT or STAND. (Here is where the model decides)
    - If the players card's sum is more than 21, the player has lost and the game ends
    - Else, ask the player to HIT or STAND again.
- When the player chooses to STAND, the dealer policy is executed.
    - Dealer policy:
        - 1. HIT - if the sum of dealer cards is less than 17.
        - 2. STAND otherwise (the sum is greater or equal to 17).
- After the dealer plays, the game winner is announced. (Reward is given)

A more RL oriented look at the above game mechanics:

The following is valid for all the RL methods.

The state will be s = (PlayerSum, DealerCard, hasAce?). Therefore the State-Action pairs will be Q = (s, HIT/STAND). The player sum is between 12 and 21 while the dealer's card is between 2 and 11 (11 being an Ace), and hasAce? is a boolean variable determining if the player has an Ace which can turn into a 1.

For exploration reasons it is important to know the number of times an action $a \in \mathcal{A}$ was selected in a state $s \in$ S. Hence, the counts of every state action pair were stored also.

The Reward given is the following:

- If the outcome of the round is a win, the reward is +1,
- if the outcome is a loss, the reward is -1,

- otherwise in case of a draw the reward is 0.

Therefore the reward is given only after the game winner is identified. Hence after the final state is reached.

How are the State-Action Values and Counts stored?

The State-Action values and their respective counts are stored in a dictionary named "StateActionValues". The keys of this dictionary are tuples in the form of: (s, a), where s is the state (for example, (12,3,True)) and a is the action (HIT/STAND). This dictionary will return a list containing two elements. The 1st being the actual Q value expected by taking action a in state s and the 2nd element being the number of times action a was chosen in state s. Example: StateActionValues[((12,3,True),STAND)] = [-0.3853, 218]. -0.3853 being the expected return if Stand is picked and 218 being the number of times the model chose Stand from that state. By making use of a dictionary, the look up speed for every state action pair is very fast and hence the algorithms can adjust these values efficiently. It is important to note that this dictionary is initialised before every model is called by calling InitialiseQvalues().

InitialiseQvalues() sets every StateActionValues (that will be needed) Qvalue and counts to 0.

RL methods

In the next part of this section, how every model was created will be discussed.

The three RL methods that were constructed (each having 4 variations) are Monte Carlo, SARSA and Q-Learning (SARSAMAX). The first two are On-policy while the last is Off-policy. All of the models fall under Model Free Learning, hence they are suitable for learning to play BlackJack. This is because it is impossible to know the underlying Markov Decision Process (MDP) for a game of Blackjack because of the "randomness" of the deck (therefore transitions). Model Free learning models can learn without any knowledge about the underlying MDP [1].

Monte Carlo

MC methods learn from experience by looking at complete episodes. Therefore the state-action values are updated after the episode is finished (final state is reached). More specifically, it uses a ϵ-Greedy policy to improve. After every episode the Q values (V(S)) and Ncounts (N(S)) used need to be updates following the following equations:

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

The ϵ-Greedy Policy is the following

- With probability ϵ choose a random action.
- With probability $1 - \epsilon$, choose the best action

In MC there is the need to keep track of the current episode, currentEpi will do just this. Hence every time an action is chosen from a state it is added to currentEpi by currentEPi.append((state,a)). Every time an action needs to be taken, MC(state,currentEpi,Emode) is called. This function will make a choice depending on the ϵ-Greedy policy (stochastic).
Before the function returns the reward, the Q-values and counts are adjusted by calling updateQvalues(currentEpi,result).
Two versions of MC were built: playBlackjackMC() and playBlackjackMCExplore(). The only difference being that in playBlackjackMCExplore() the action taken from the first state in an episode is chosen randomly. This is done by a = random.randint(STAND, HIT) for only the first state-action in the episode. The rest of the episode uses ϵ-Greedy policy with ϵ = 1/k.

SARSA (State–action–reward–state–action)

The sarsa model is similar to the MC one with the differences being the following.

- SARSA is a temporal difference method hence it learns from every time step.
- Perform the Q-values update in every timestep and the formula for the update is the following:

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

where alpha is the step size $\alpha = \frac{1}{N(s,a)+1}$

For Sarsa function playBlackjackSarsa(episode,Emode) is called. Again Emode parameter will handle the configuration of the ϵ-Greedy policy. The difference to MC being that it updates the values at every timestep and uses sarsaUpdateValues() to do so. Hence it only keeps track of the previous Q and the current Q. It is expected that SARSA learns faster than MC [2].

Q-learning (SARSAMAX)

It is an Off policy method. Hence it uses another policy to select the next action from a given state. The reference policy being a configuration of SARSA ϵ-Greedy policy. The main difference here is that in Q-learning the best/greedy action is chosen. Q-Learning is biassed towards the highest values and hence exploration is limited [3]. In the implementation, Q-learning mode first learns Sarsa's policy then switches to Q-learning (calling function qLearning()) where it uses the Sarsa policy but always chooses the best possible action.

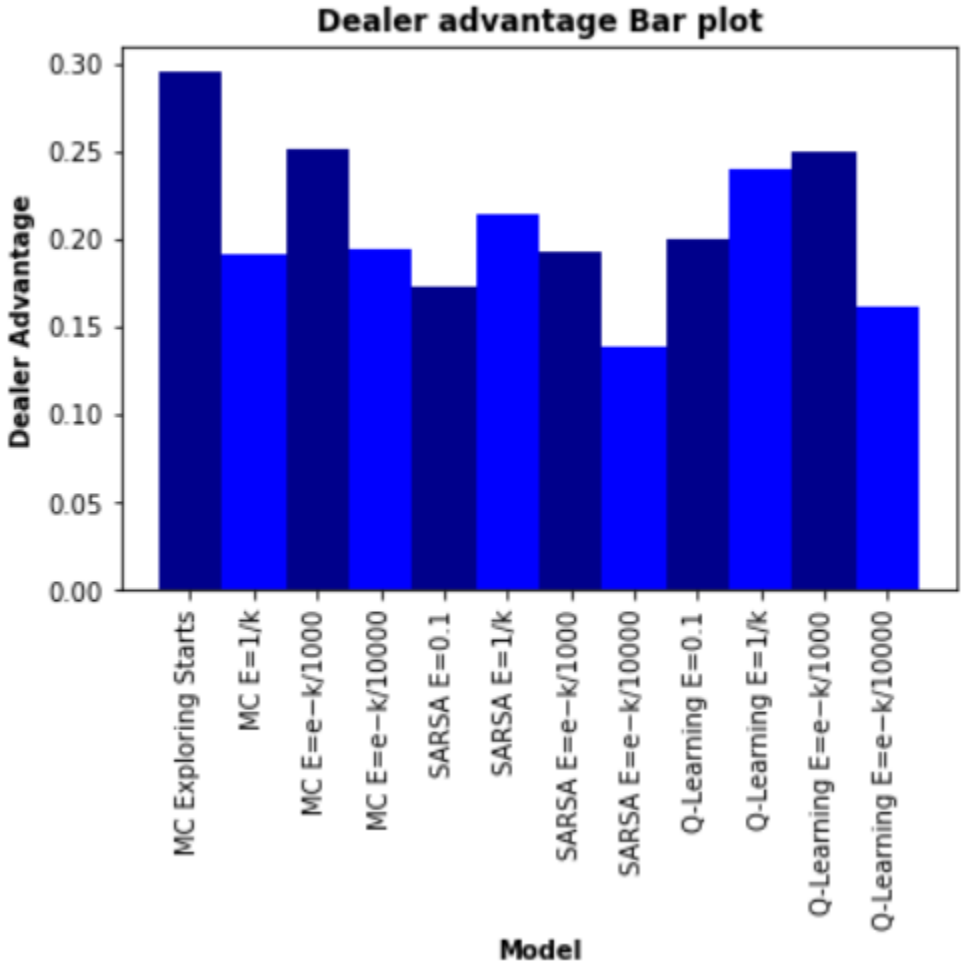
After implementing these models, proper testing (mostly input-output testing) was performed to ensure that they work as intended. Finally, various plots and tables were extracted from the results of the models. These will be used to analyse the RL models and compare them to each other. Moreover, the trade off between exploration and exploitation will be highlighted.

Results and observations

The table below displays the mean number of wins, draws and losses over the last 100,000 episodes for each policy obtained using each algorithm configuration.

Model	Win (mean)	Draw (mean)	Lose (mean)	Dealer advantage
MC Exploring Starts	0.33067	0.0623	0.60703	0.29472
MC $\epsilon = 1/k$	0.37408	0.07563	0.55029	0.19062
MC $\epsilon = e^{-k/1000}$	0.34916	0.06786	0.58298	0.25084
MC $\epsilon = e^{-k/10000}$	0.37537	0.0694	0.55523	0.19327
SARSA $\epsilon = 0.1$	0.38455	0.07045	0.545	0.17261
SARSA $\epsilon = 1/k$	0.36333	0.07573	0.56094	0.21380
SARSA $\epsilon = e^{-k/1000}$	0.37275	0.07696	0.55029	0.19234
SARSA $\epsilon = e^{-k/10000}$	0.39376	0.08582	0.52042	0.13855
Q-Learning $\epsilon = 0.1$	0.37075	0.07382	0.55544	0.19940
Q-Learning $\epsilon = 1/k$	0.35321	0.07054	0.57626	0.23997
Q-Learning $\epsilon = e^{-k/1000}$	0.355	0.05422	0.59079	0.24930
Q-Learning $\epsilon = e^{-k/10000}$	0.38888	0.07338	0.53775	0.16065

Dealer Advantage plot



Blackjack Strategy tables for every model's configurations

Monte Carlo models:

MC Explore

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	H	S	S	S	S	H	H	H
17	H	H	H	H	H	H	H	H	H	H
16	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H
14	H	H	H	H	H	H	H	H	H	H
13	H	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	H	H	H	H	H	H

MC $\epsilon = 1/k$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	H	S	S	S	H	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
18	H	H	S	S	S	H	H	H	H	S
17	S	H	H	S	H	H	H	S	H	H
16	H	H	H	S	S	H	H	H	H	H
15	H	H	H	H	S	H	H	H	H	H
14	H	S	H	H	H	H	H	H	H	H
13	H	H	H	H	H	S	H	H	H	H
12	H	H	S	S	H	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	S	S	S	H	S	S	H	S	H
18	H	H	S	S	H	S	S	H	H	H
17	H	H	H	H	H	S	H	H	S	H
16	S	H	H	S	S	H	H	H	H	H
15	S	S	S	S	H	H	H	S	H	H
14	H	S	S	H	H	S	H	H	H	H
13	H	H	H	S	H	H	H	H	H	H
12	H	S	H	H	H	S	H	H	H	H

MC $\epsilon = e(-k/1000)$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	H	H	S	S	S	H
19	S	S	H	S	H	H	H	S	S	H
18	S	S	S	S	H	S	S	S	H	S
17	H	S	H	H	H	S	H	S	H	H
16	H	S	S	H	S	H	H	H	H	H
15	S	H	H	H	S	H	H	H	H	H
14	S	H	H	H	H	H	H	H	H	H
13	S	H	H	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	H	S	S	S	S	H	S
19	S	S	H	H	S	S	S	S	H	S
18	S	S	H	S	S	S	S	H	S	S
17	H	S	S	S	H	S	S	S	H	H
16	S	H	H	H	H	H	H	H	H	H
15	H	S	H	H	H	H	H	H	H	H
14	H	S	H	S	S	S	H	H	S	H
13	S	H	S	H	H	H	H	H	H	H
12	S	H	H	H	H	H	H	H	H	H

MC $\epsilon = e(-k/10000)$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	H	S	H	S	S	S	S	S	S	S
19	H	S	S	S	S	H	S	S	S	H
18	S	S	H	H	H	H	S	H	H	H
17	S	H	H	H	S	S	H	S	H	H
16	S	H	H	H	H	S	H	H	H	H
15	H	H	S	S	H	H	H	S	H	H
14	H	H	H	S	S	H	H	H	H	H
13	H	S	H	S	H	H	H	H	H	H
12	S	H	H	H	H	S	S	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	H	S	S	S	S	H
19	H	S	S	S	S	S	S	S	S	S
18	S	H	H	H	S	S	H	S	S	H
17	H	S	S	H	H	H	H	H	H	H
16	H	S	H	S	S	H	H	H	H	H
15	S	H	S	S	H	H	H	H	H	H
14	S	S	S	S	H	H	H	H	S	H
13	H	H	S	H	H	H	H	H	H	H
12	H	H	H	H	H	H	H	S	S	H

Sarsa models:

SARSA ϵ = 0.1

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	S	S	S	S	S	S	S	S	H
18	H	S	S	S	S	S	S	H	H	S
17	H	H	S	H	H	S	S	H	H	H
16	S	H	H	S	H	H	H	H	H	H
15	H	H	S	H	H	H	S	H	H	H
14	H	S	H	H	S	H	H	S	H	H
13	H	H	H	S	S	H	S	S	H	H
12	H	H	S	H	S	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	H	S	S	H	H	S
18	S	S	H	S	S	H	S	H	S	H
17	S	S	H	H	S	H	H	H	H	H
16	S	H	H	S	S	S	H	H	S	H
15	S	S	S	H	H	H	H	H	H	H
14	H	H	S	H	S	H	H	H	H	H
13	S	H	S	H	H	H	H	H	H	H
12	S	H	S	H	S	S	S	H	S	H

SARSA $\epsilon = 1/k$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	H	S	S	S
19	S	S	S	H	H	S	S	S	H	H
18	S	S	S	S	S	S	S	H	H	S
17	S	H	S	H	S	H	H	H	H	S
16	H	H	S	S	S	S	S	S	H	H
15	H	S	H	H	H	H	H	H	H	H
14	S	H	S	H	H	H	S	S	S	H
13	H	H	H	S	S	H	H	H	H	H
12	H	H	S	H	H	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	S	S	S	H	H	S	S	H	H
18	H	H	H	S	H	S	S	S	S	H
17	S	S	S	S	H	S	H	H	H	H
16	H	H	H	H	S	S	H	H	S	H
15	S	H	H	S	H	S	H	H	H	H
14	S	S	H	S	H	H	H	S	H	H
13	S	S	H	H	H	H	H	H	H	H
12	H	H	H	H	S	S	H	H	H	S

SARSA $\epsilon = e^{(-k/1000)}$
With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	H	S	S	S	S	S	S	S
19	H	S	S	S	S	S	S	S	S	H
18	H	S	H	H	S	S	S	S	H	H
17	S	H	H	S	S	H	S	H	H	H
16	H	H	H	S	S	S	H	H	H	H
15	S	S	H	H	S	H	S	H	H	H
14	S	H	S	H	S	H	S	H	H	H
13	H	H	H	H	S	S	S	H	H	H
12	H	S	S	H	S	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	H	S	S	S	S	S	S	S
19	H	S	S	H	S	H	S	S	H	S
18	S	S	S	S	S	S	S	H	H	S
17	H	S	S	S	S	S	H	S	H	H
16	S	S	S	H	S	H	H	S	H	H
15	H	S	H	S	H	H	H	H	H	H
14	S	H	S	H	S	S	H	S	H	H
13	H	S	S	S	H	S	H	H	H	H
12	H	S	H	H	H	S	H	H	H	H

SARSA $\epsilon = e^{(-k/10000)}$
With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	H	S	S	S	S	S
19	S	H	S	S	S	S	S	H	H	H
18	H	S	S	S	S	S	H	S	H	H
17	S	H	S	H	H	S	H	H	S	S
16	H	H	S	H	H	H	S	H	S	H
15	S	H	S	S	H	H	H	S	H	H
14	H	H	S	S	H	H	H	H	H	H
13	S	H	S	H	S	S	H	S	H	H
12	H	H	S	H	H	H	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	H	S	S	S	S	S	S
19	S	H	S	H	S	S	S	S	S	H
18	S	S	S	S	S	S	S	H	S	S
17	S	S	S	S	S	H	H	S	S	H
16	H	S	H	H	S	H	H	H	H	H
15	S	H	H	S	S	S	H	H	H	H
14	H	S	H	H	H	H	S	H	H	H
13	H	S	S	S	H	H	S	H	H	H
12	H	H	H	S	S	H	H	H	H	H

Q-Learning models:

Q-Learning ϵ = 0.1
With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	H	S	S	S	S	H	S	S	S
19	S	H	S	S	S	S	S	S	H	H
18	S	S	S	S	S	S	H	S	H	S
17	S	S	H	S	H	S	H	S	H	H
16	H	S	H	S	H	S	H	H	H	S
15	H	H	S	S	S	H	H	H	H	H
14	H	S	H	S	H	H	H	H	H	H
13	H	S	S	H	H	H	H	H	S	H
12	H	S	S	S	S	S	H	H	H	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	H	S	S	S	H	H	S	S	S	H
18	S	S	S	H	S	H	S	H	H	S
17	H	S	S	S	S	S	H	H	H	H
16	H	H	H	H	H	S	S	H	S	H
15	H	H	H	S	S	S	H	H	H	H
14	H	H	S	S	H	H	H	H	S	S
13	H	H	S	H	S	H	H	H	H	H
12	H	S	S	H	H	H	H	H	S	H

Q-Learning $\epsilon = 1/k$
With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	H	S	S	S	S	S	S	S	H
18	S	S	H	S	H	S	S	H	H	S
17	H	H	S	S	S	S	H	H	H	H
16	S	H	S	S	H	H	H	H	H	S
15	H	H	S	S	H	H	H	S	H	H
14	S	S	S	H	H	H	S	S	H	S
13	H	S	H	S	S	H	H	S	H	H
12	H	S	S	H	S	H	H	H	H	S

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	H	S	S	S	S	S	S	H	S	S
19	S	H	S	S	S	S	S	S	H	S
18	S	H	H	S	H	S	H	H	H	H
17	H	S	S	S	S	S	H	H	H	H
16	S	H	S	H	S	H	S	S	H	S
15	H	H	H	S	H	H	H	H	H	S
14	H	S	H	S	H	S	H	H	H	H
13	H	H	H	H	S	S	H	H	H	H
12	S	S	H	H	H	S	S	H	H	S

Q-Learning $\epsilon = e^{(-k/1000)}$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	S	S	H	S	S	S	S	H	H
18	S	S	H	H	S	H	S	H	H	H
17	S	S	H	H	S	H	H	H	S	H
16	H	H	S	S	H	H	H	H	H	H
15	S	H	S	S	H	H	S	H	H	S
14	S	H	H	H	H	H	H	H	S	H
13	H	S	H	H	H	H	H	H	H	H
12	H	H	S	H	H	H	H	H	S	H

Without a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	H	S
19	H	H	S	H	S	S	S	S	S	H
18	S	S	H	H	H	S	S	H	H	S
17	H	H	S	H	H	H	H	S	H	S
16	S	S	S	S	H	S	S	S	H	H
15	S	S	H	H	H	H	H	H	H	S
14	H	H	H	S	S	H	H	H	H	H
13	H	H	H	S	S	H	H	S	S	H
12	H	H	S	S	H	S	H	H	H	S

Q-Learning $\epsilon = e^{(-k/10000)}$

With a usable Ace:

	2	3	4	5	6	7	8	9	10	A
20	S	S	H	S	S	S	H	H	S	S
19	H	S	S	H	S	S	S	H	S	H
18	S	S	S	S	S	S	S	S	H	S
17	S	S	S	S	H	S	S	S	H	H
16	S	H	H	H	H	H	S	H	H	H
15	H	H	H	H	H	H	H	H	H	H
14	H	H	H	S	S	H	S	H	H	H
13	S	H	H	H	S	H	S	H	H	H
12	H	H	S	H	H	H	S	H	H	S

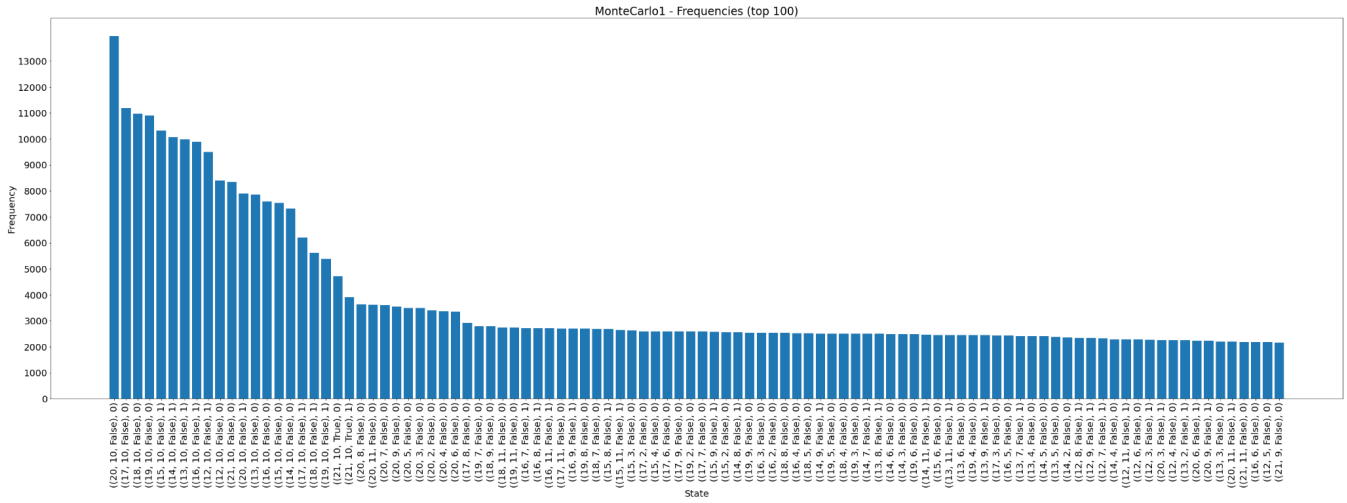
Without a usable Ace:

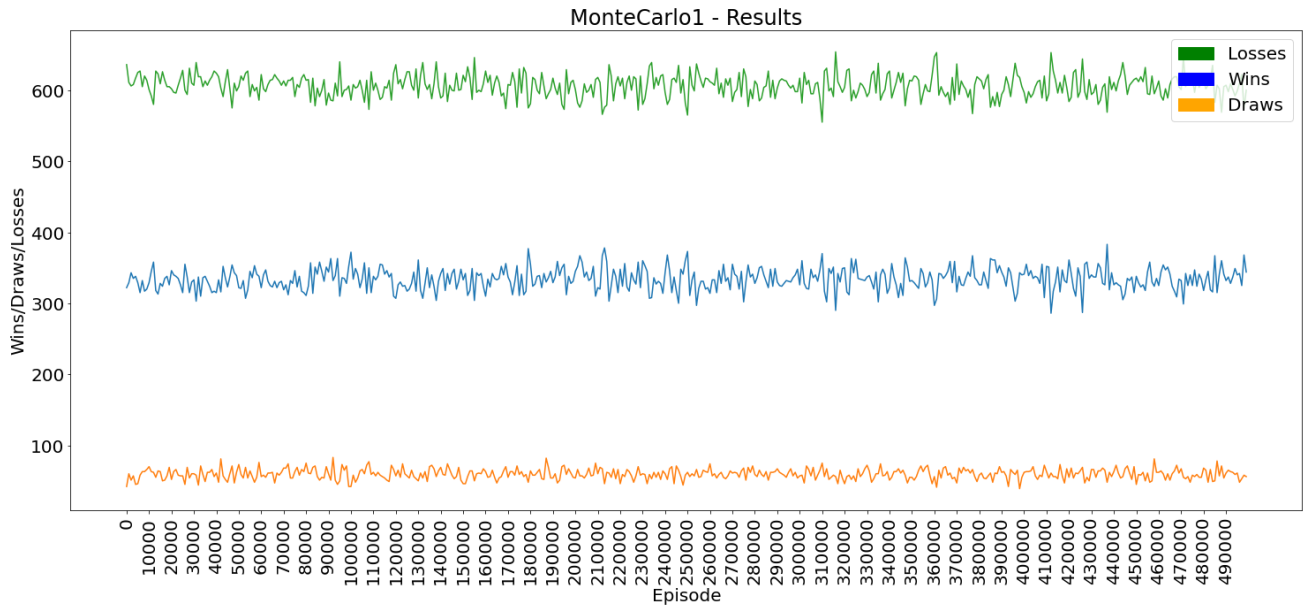
	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	H
19	S	H	S	S	H	S	H	S	S	S
18	S	H	S	S	S	S	S	S	S	H
17	H	S	H	H	S	H	H	H	H	H
16	S	H	H	S	H	S	H	H	S	H
15	S	H	S	H	H	H	S	H	H	H
14	H	H	H	S	S	H	H	H	H	H
13	H	S	S	H	H	H	S	H	H	H
12	S	H	S	S	S	H	H	H	S	H

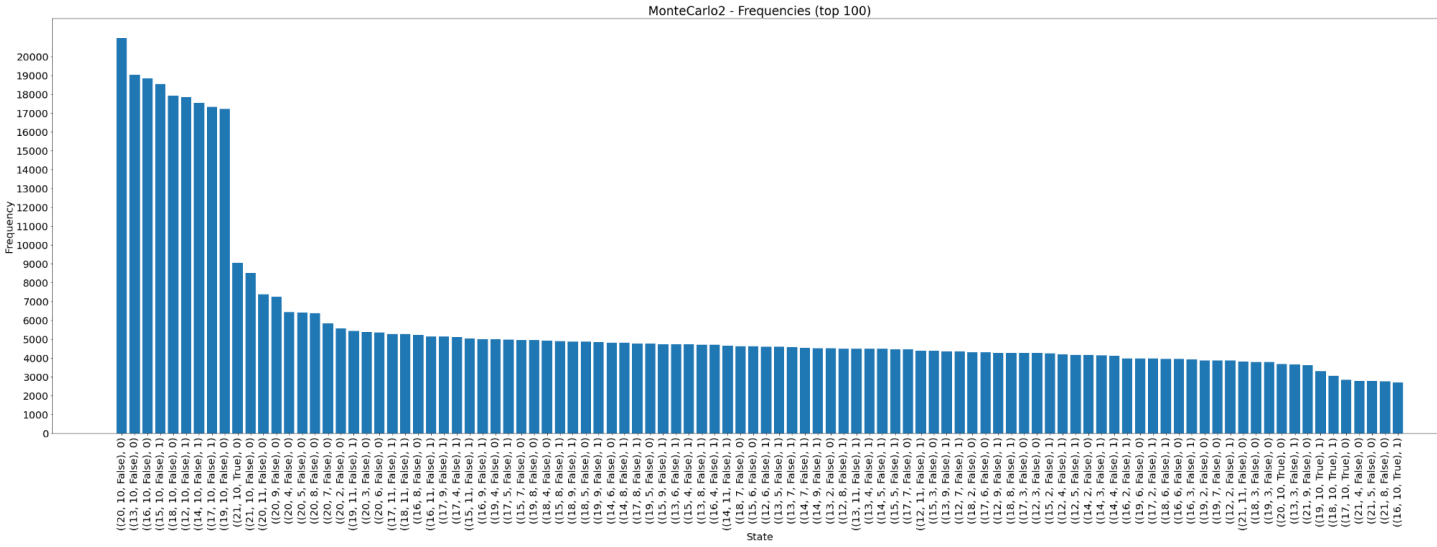
Performance of the RL methods

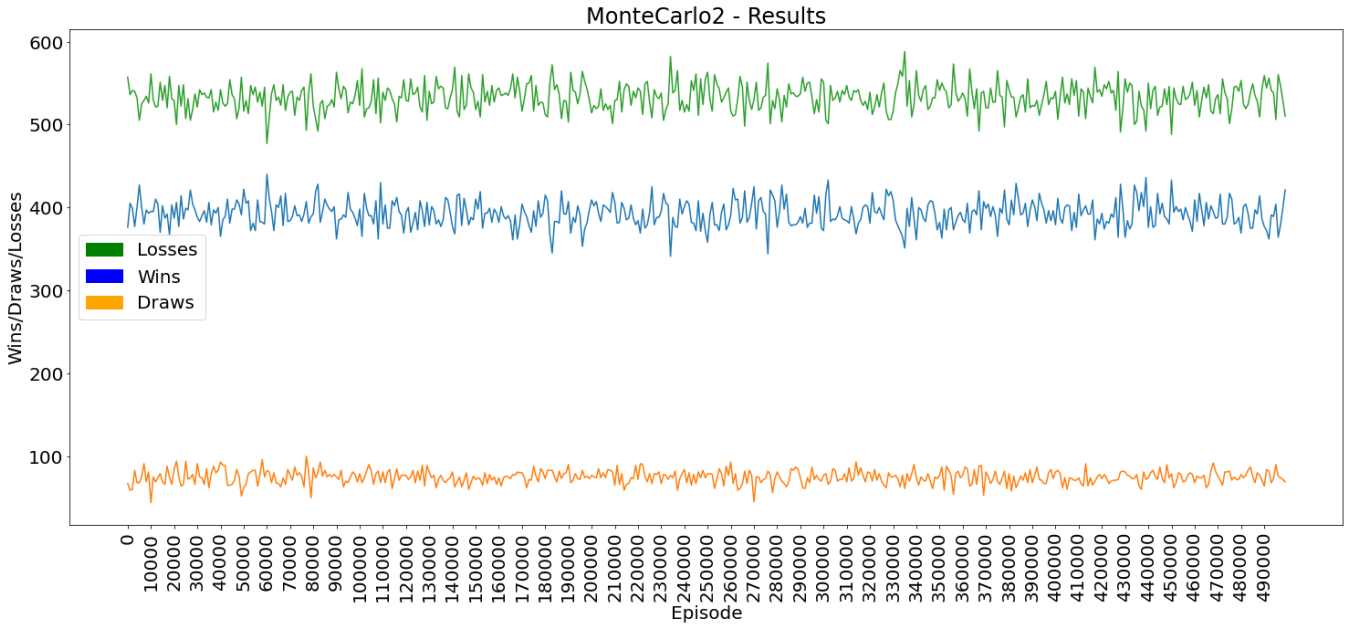
Including:

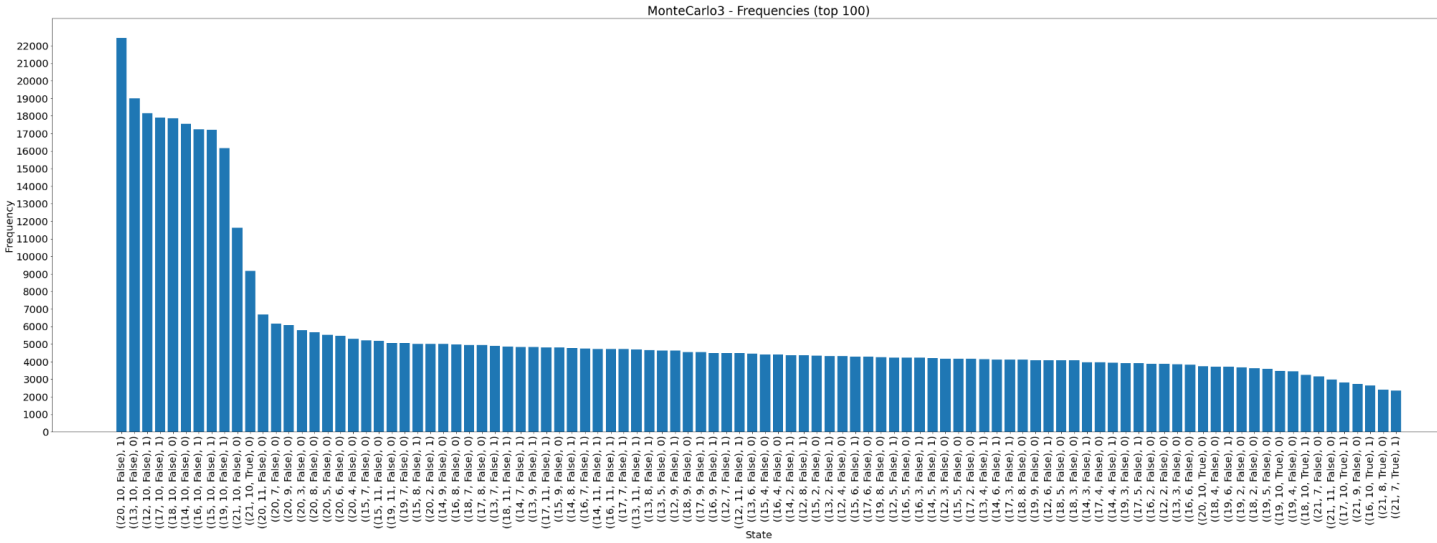
- Frequency of the top 100 state action pairs
- Win, draws and loses over the 500,000 episodes
- States not explored

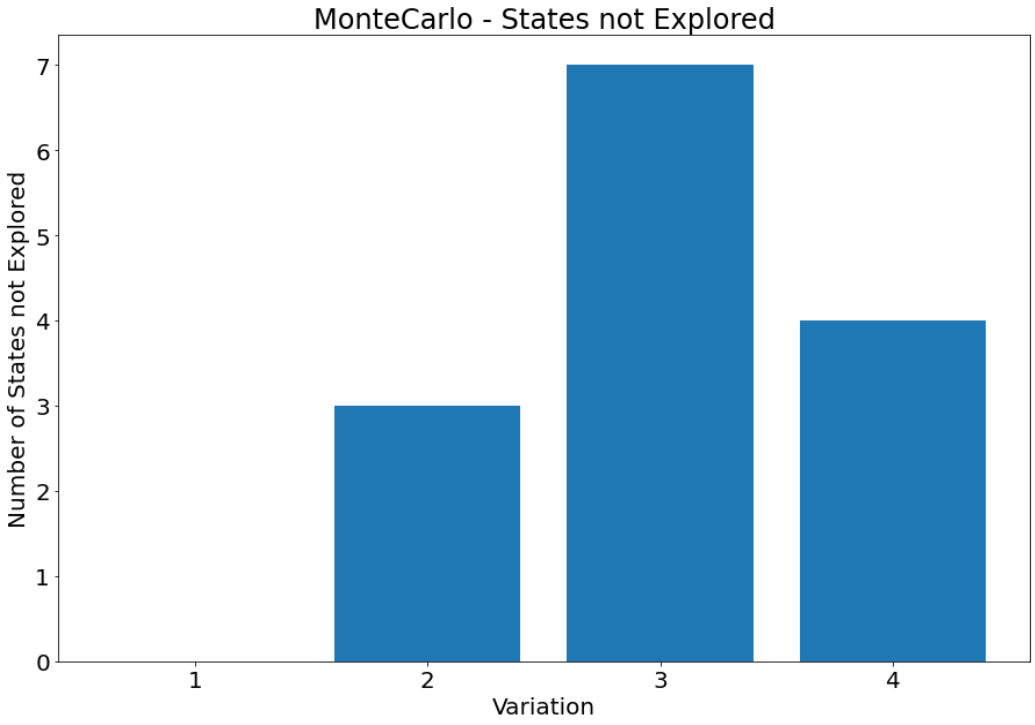


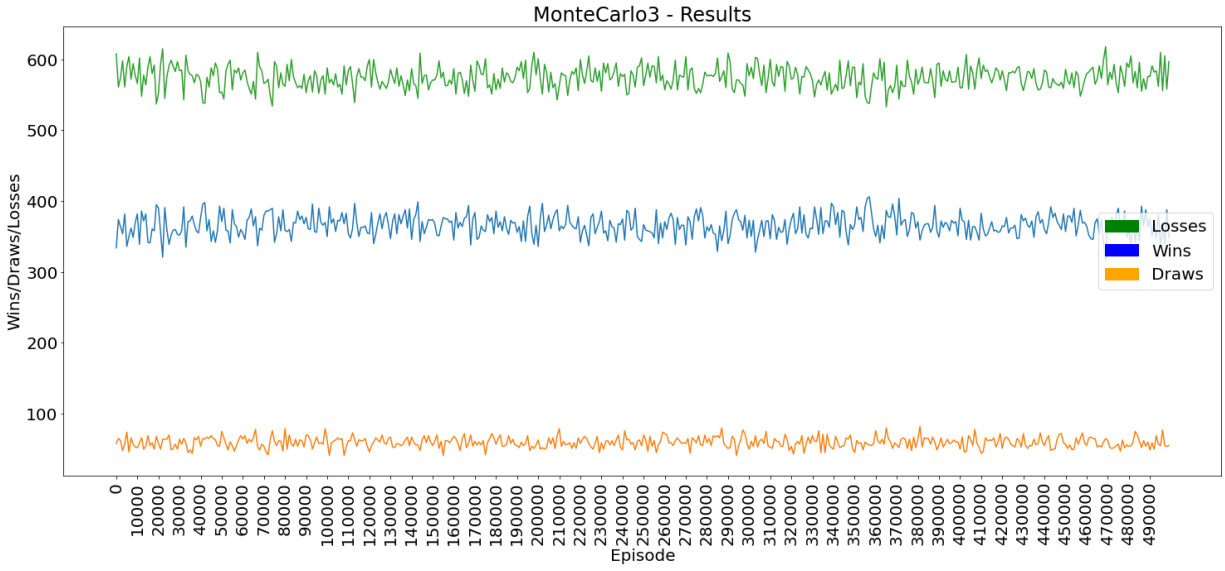


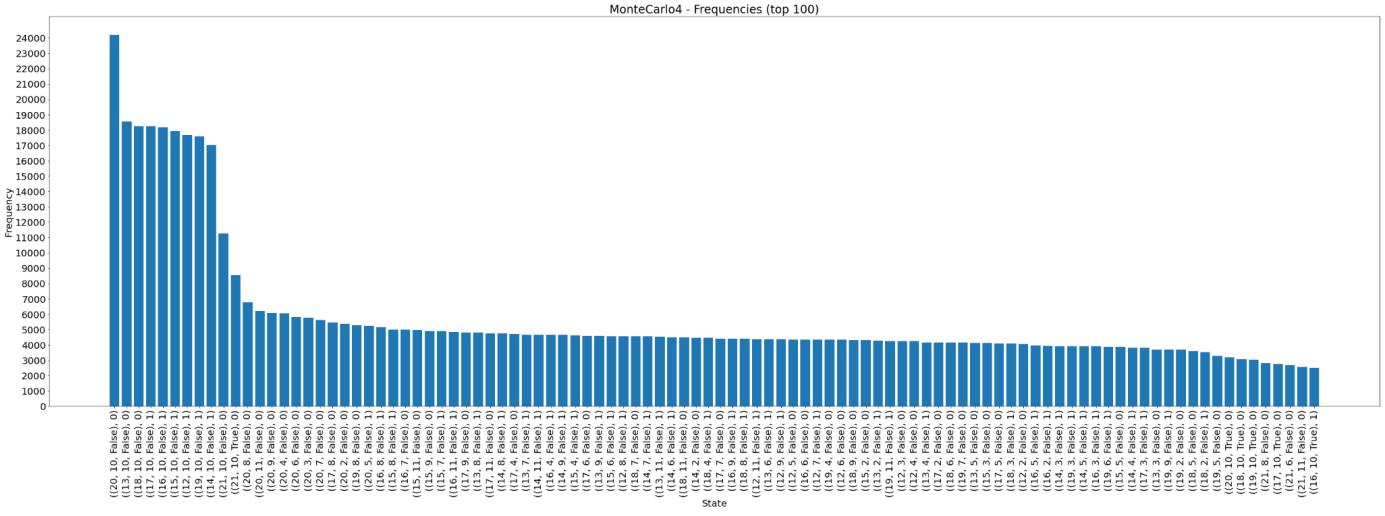

MonteCarlo2 - Frequencies (top 100)
Frequency
State

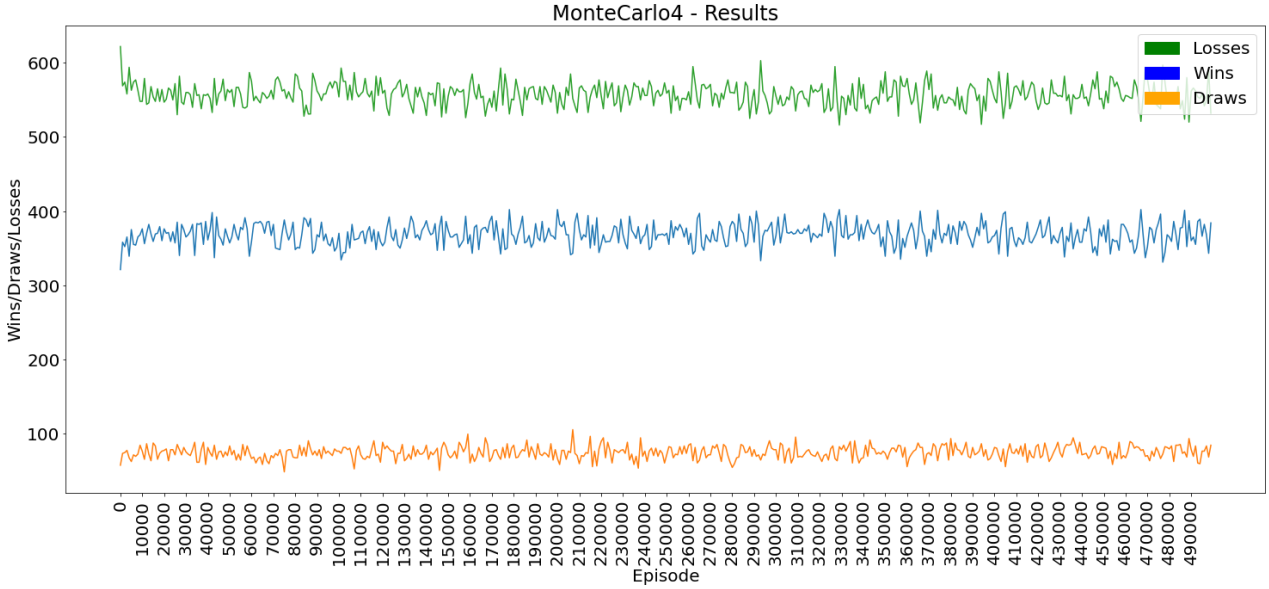

MonteCarlo2 - Results
Wins/Draws/Losses
Losses
Wins
Draws
Episode

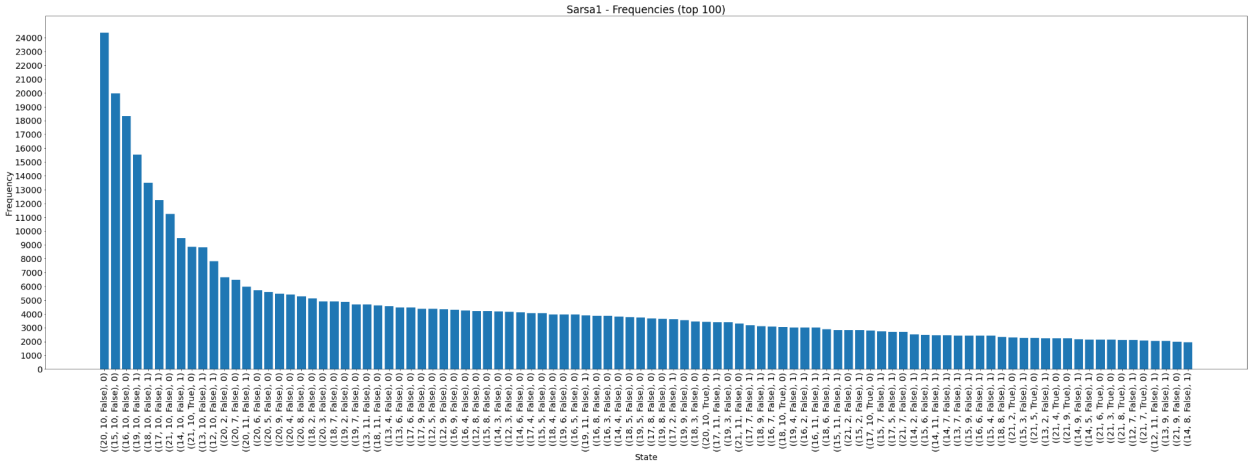

MonteCarlo3 - Frequencies (top 100)
Frequency
State

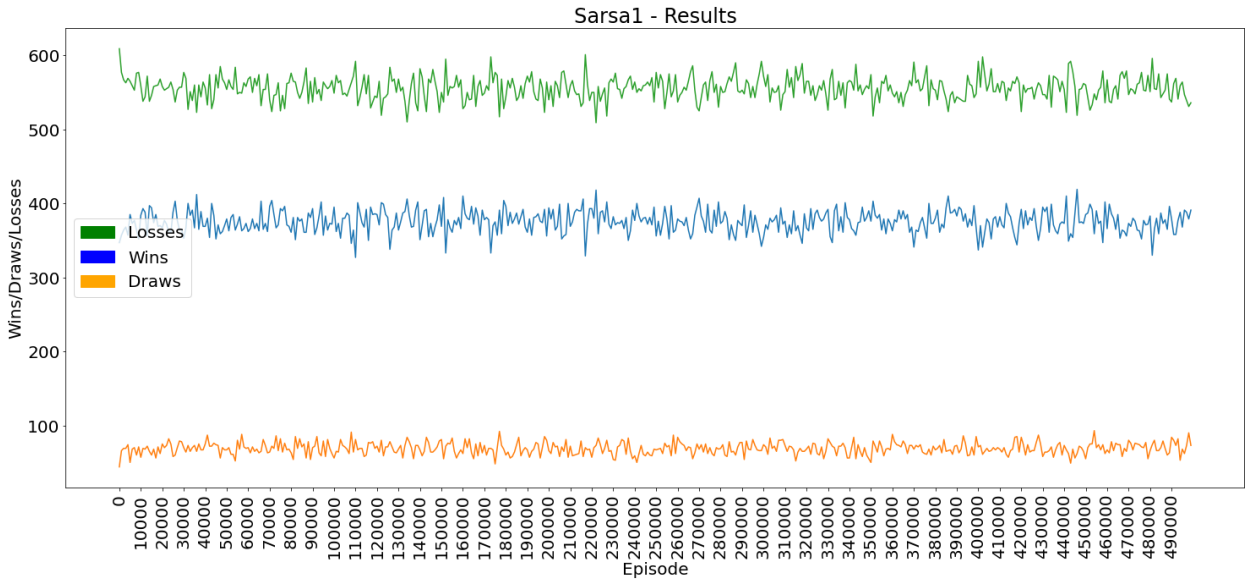

MonteCarlo - States not Explored
Number of States not Explored
Variation

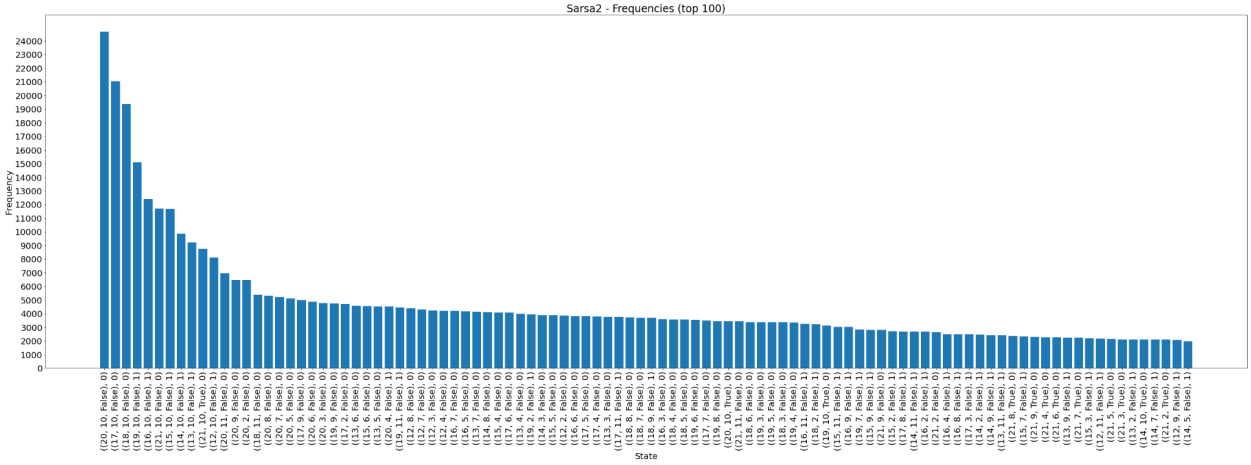

MonteCarlo3 - Results
Wins/Draws/Losses
Episode
Losses
Wins
Draws

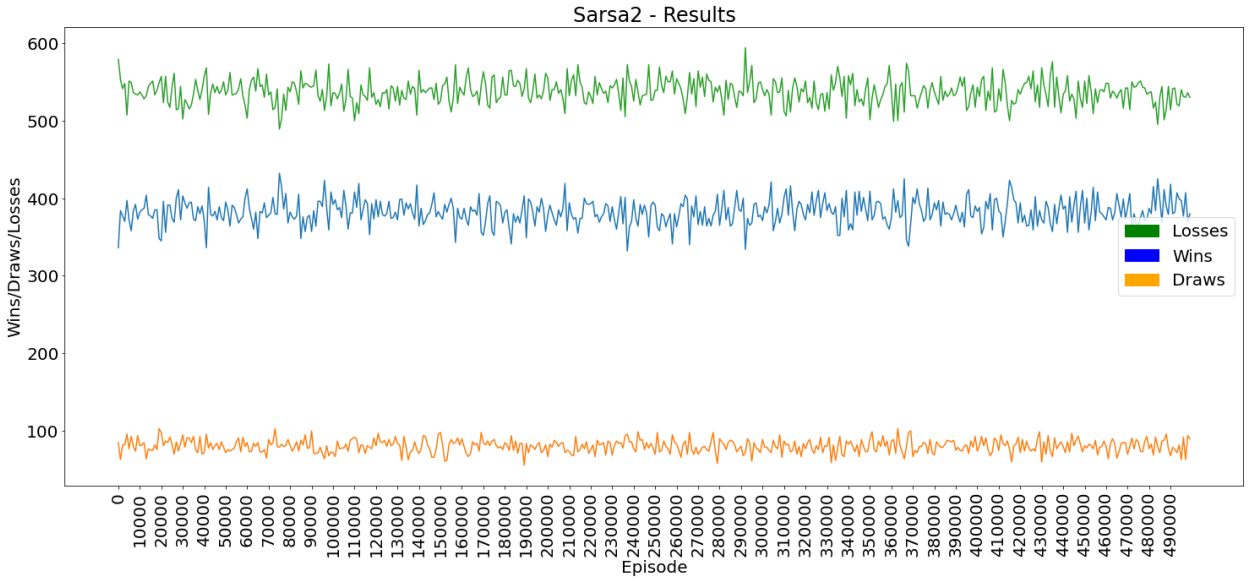

MonteCarlo4 - Frequencies (top 100)
Frequency
State

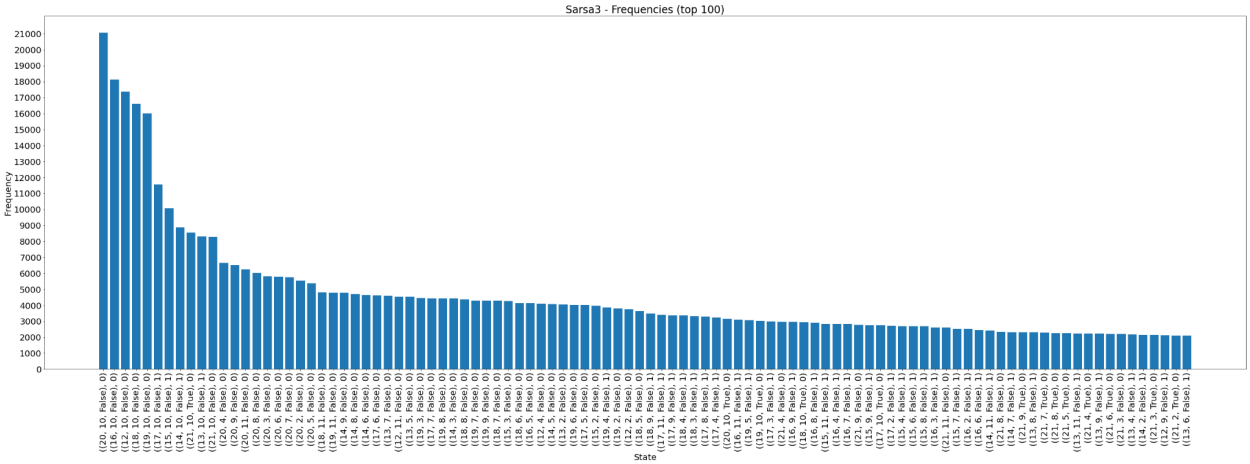

MonteCarlo4 - Results
Wins/Draws/Losses
Episode
Losses
Wins
Draws

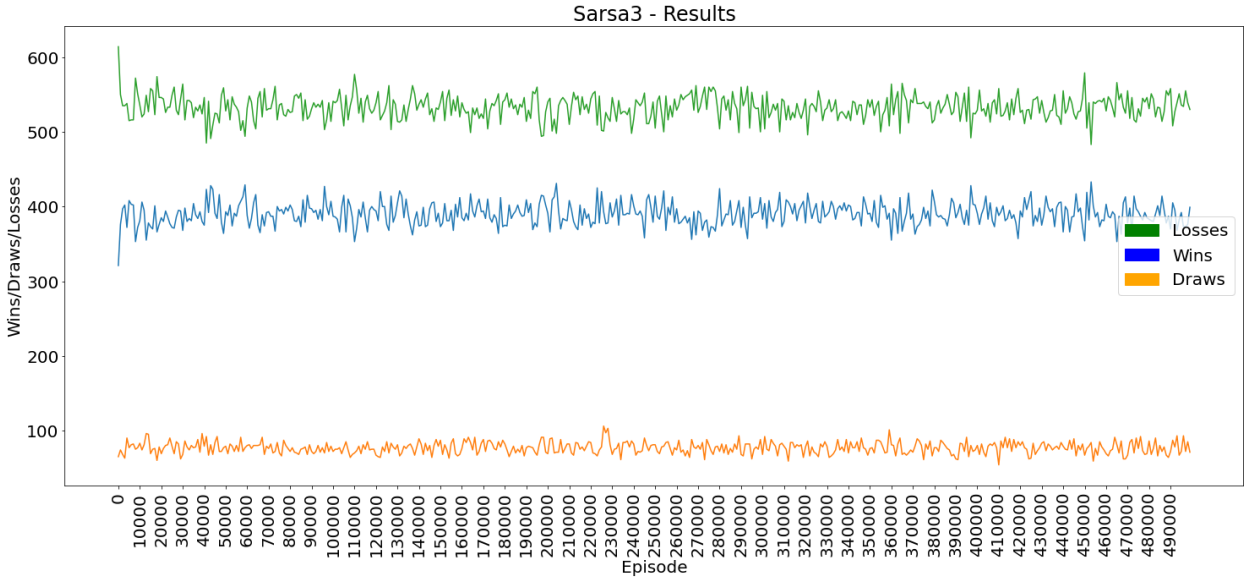

Sarsa1 - Frequencies (top 100)
Frequency
State

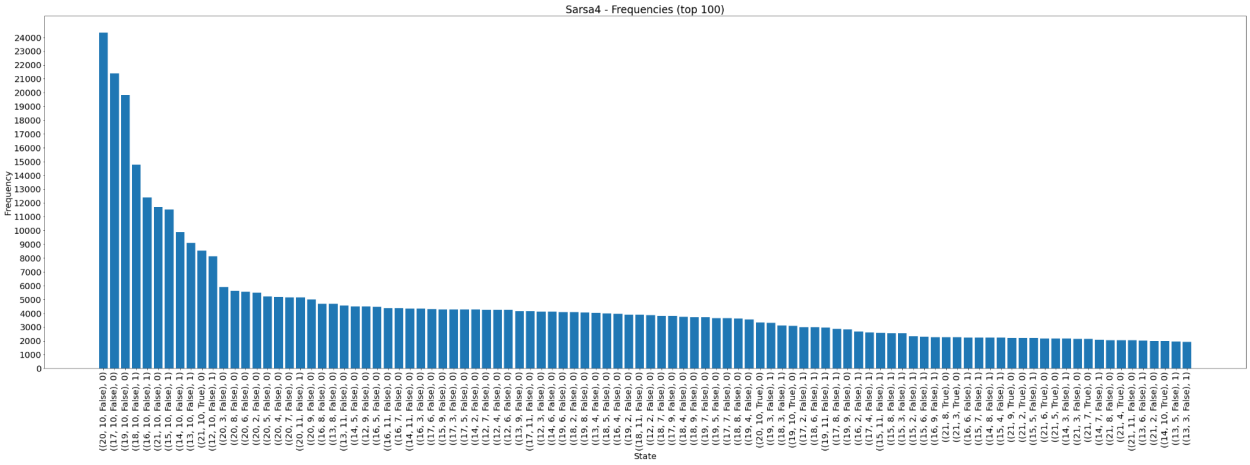

Sarsa1 - Results
Wins/Draws/Losses
Losses
Wins
Draws
Episode

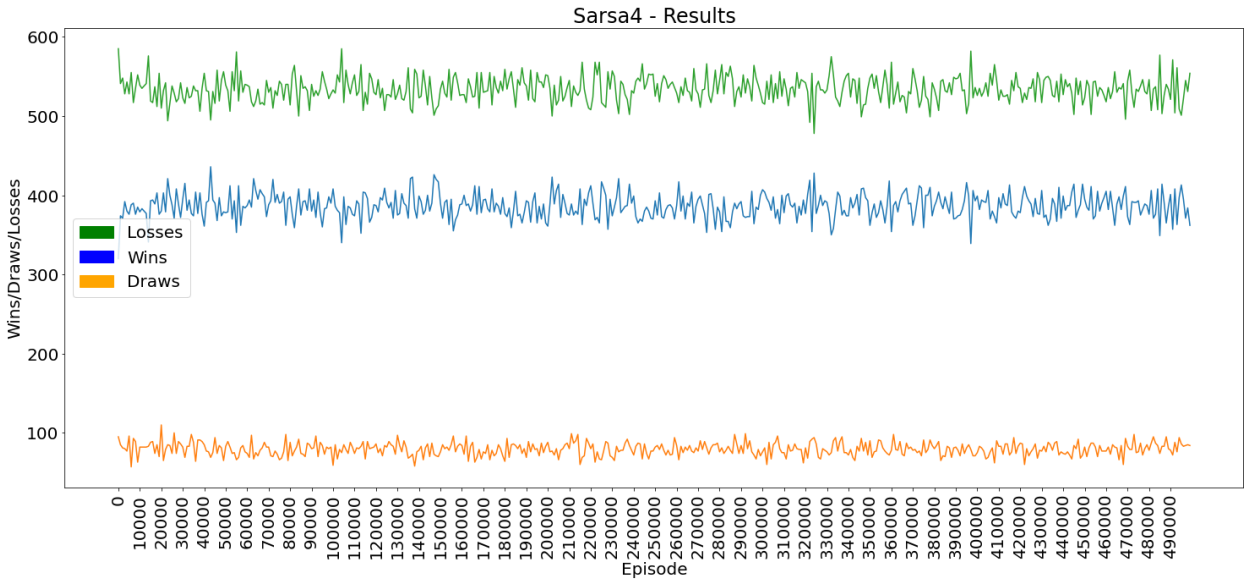

Sarsa2 - Frequencies (top 100)
Frequency
State

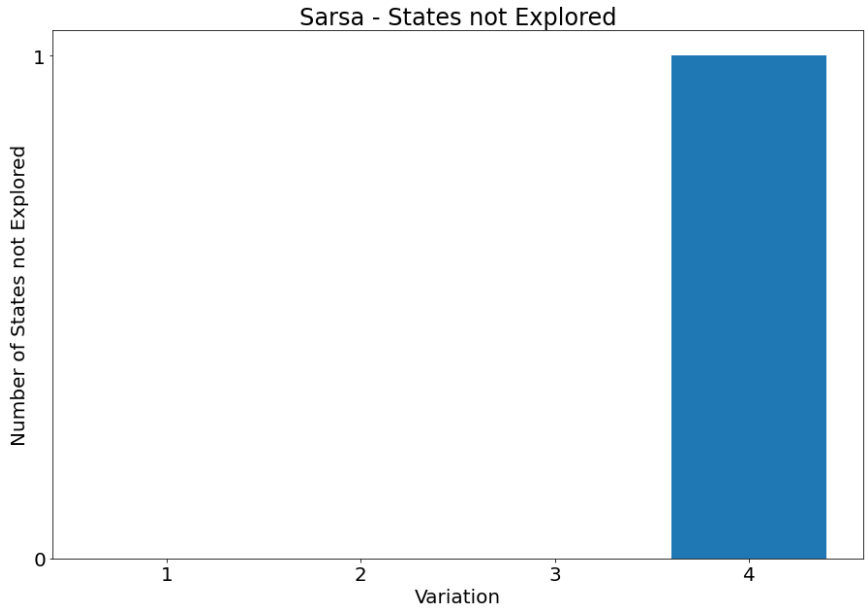

Sarsa2 - Results
Wins/Draws/Losses
Episode
Losses
Wins
Draws

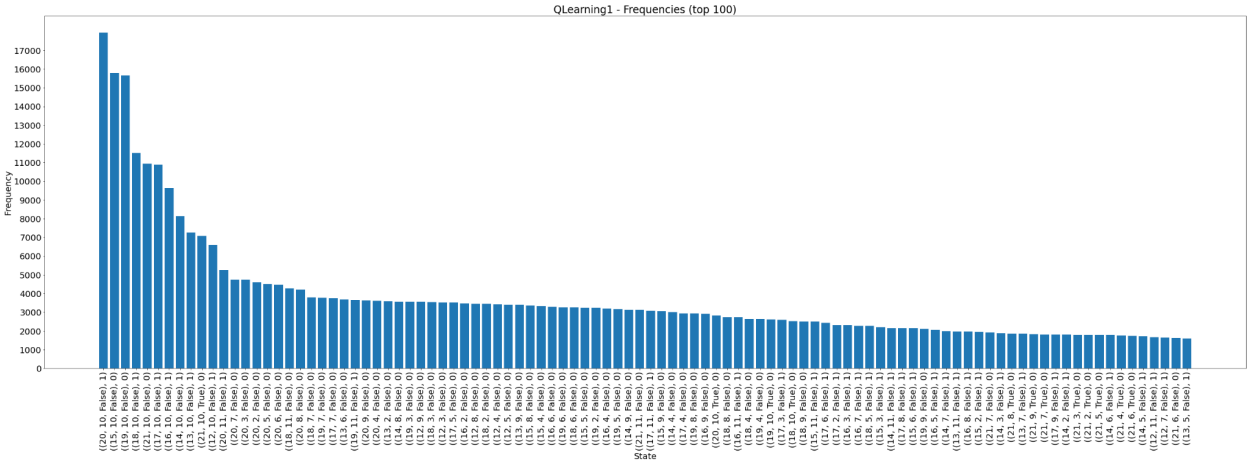

Sarsa3 - Frequencies (top 100)
Frequency
State

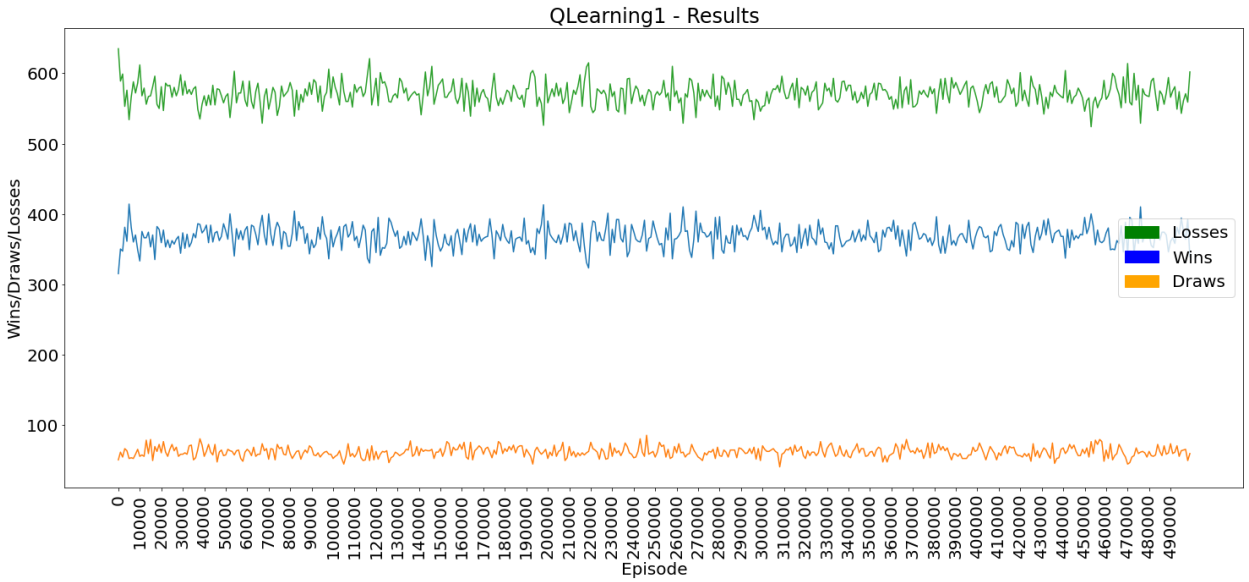

Sarsa3 - Results
Wins/Draws/Losses
Episode
Losses
Wins
Draws

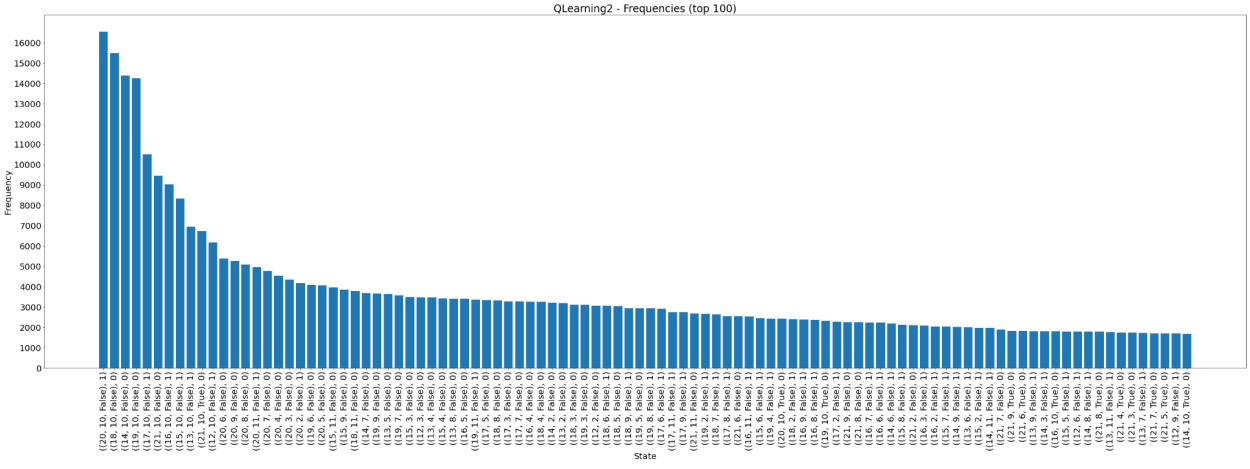

Sarsa4 - Frequencies (top 100)
Frequency
State

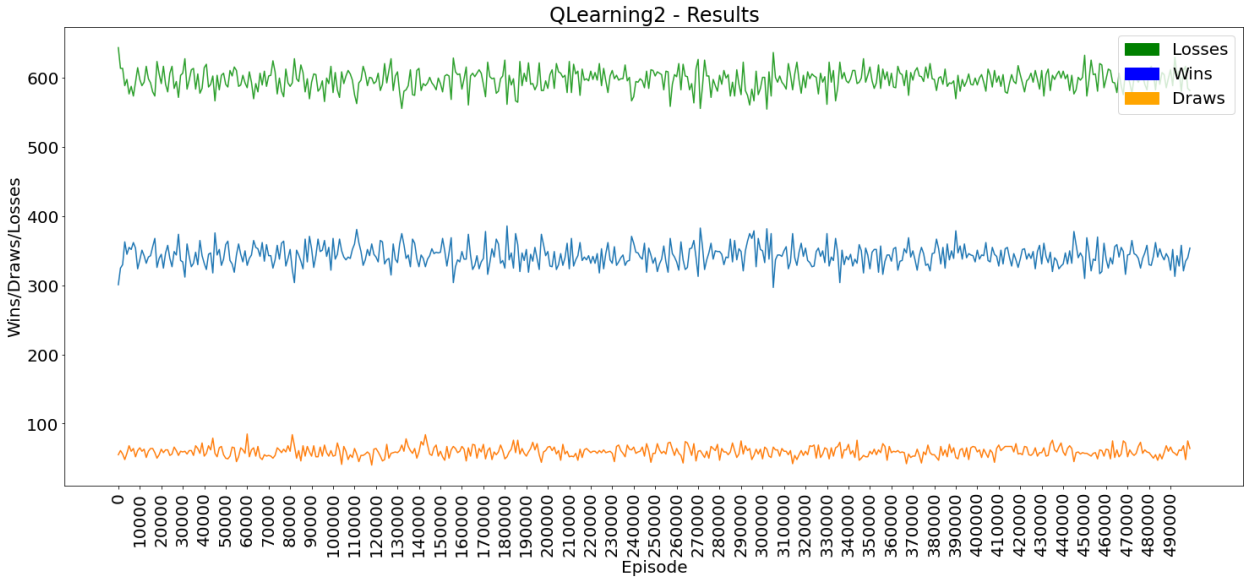

Sarsa4 - Results
Wins/Draws/Losses
Losses
Wins
Draws
Episode

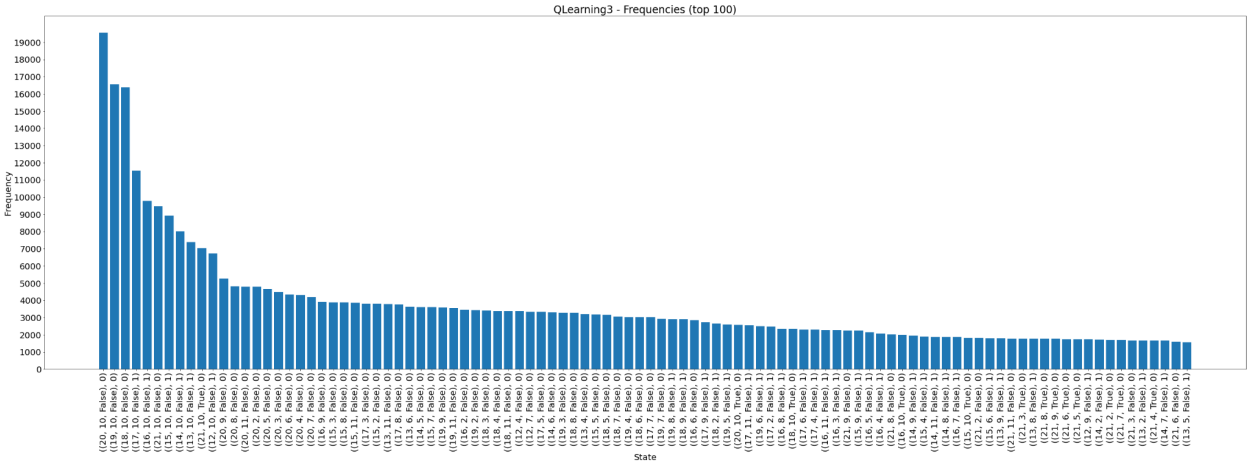

Sarsa - States not Explored
Number of States not Explored
Variation

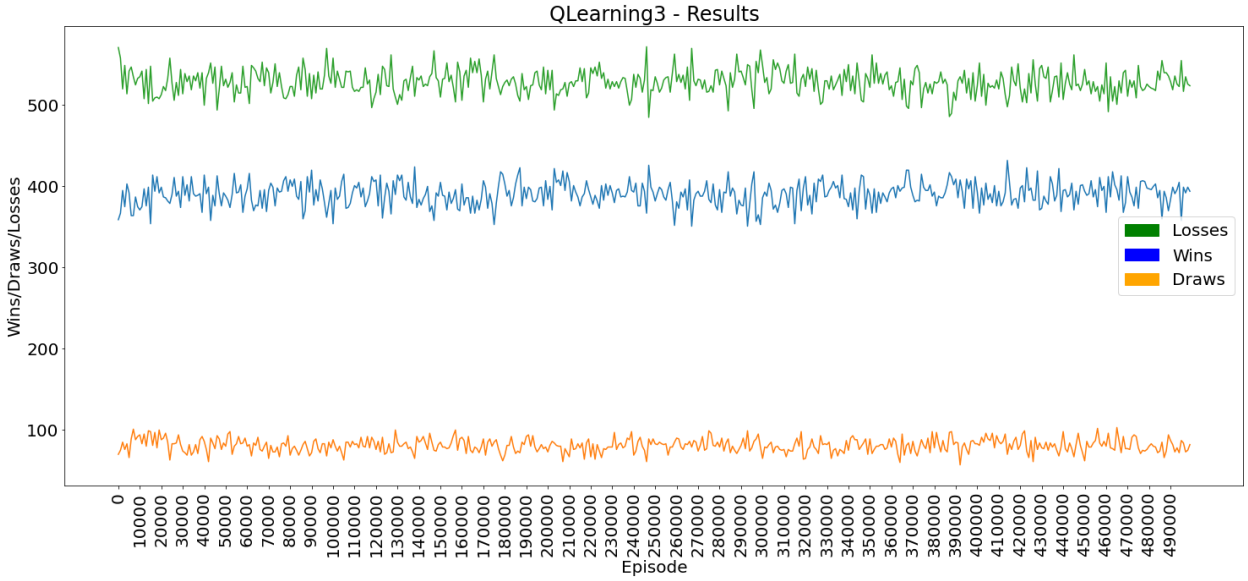

QLearning1 - Frequencies (top 100)
Frequency
State

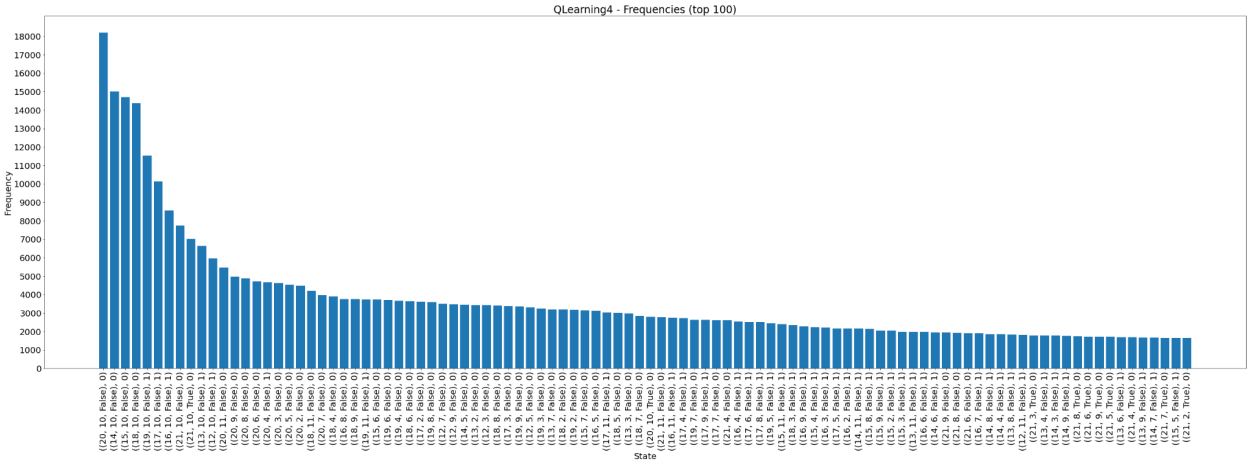

QLearning1 - Results
Wins/Draws/Losses
Episode
Losses
Wins
Draws

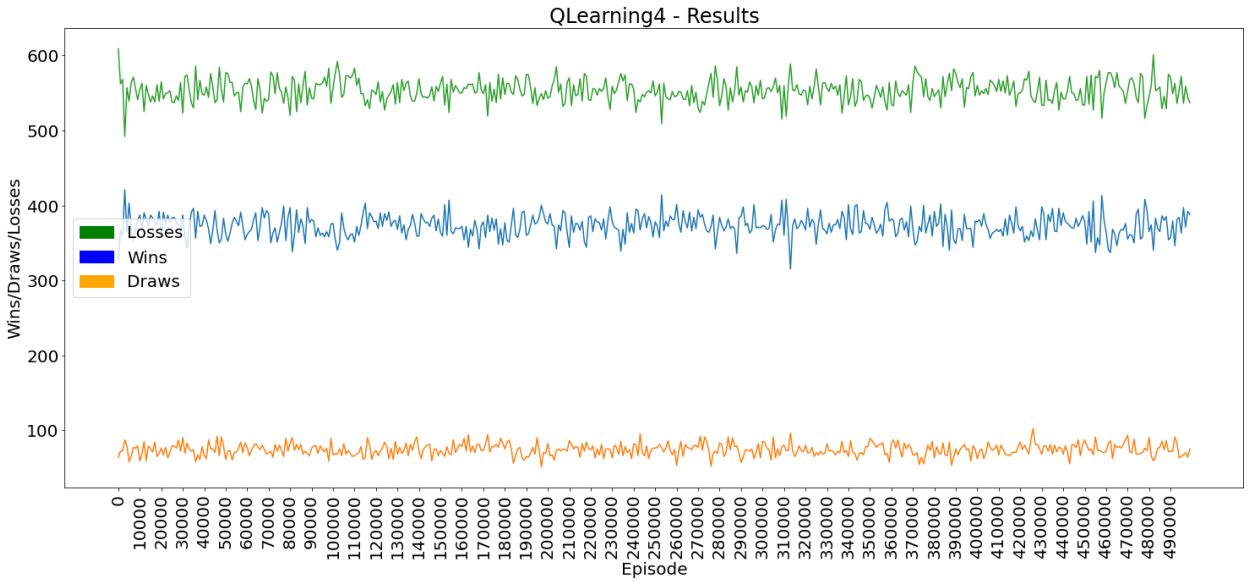

QLearning2 - Frequencies (top 100)
Frequency
State

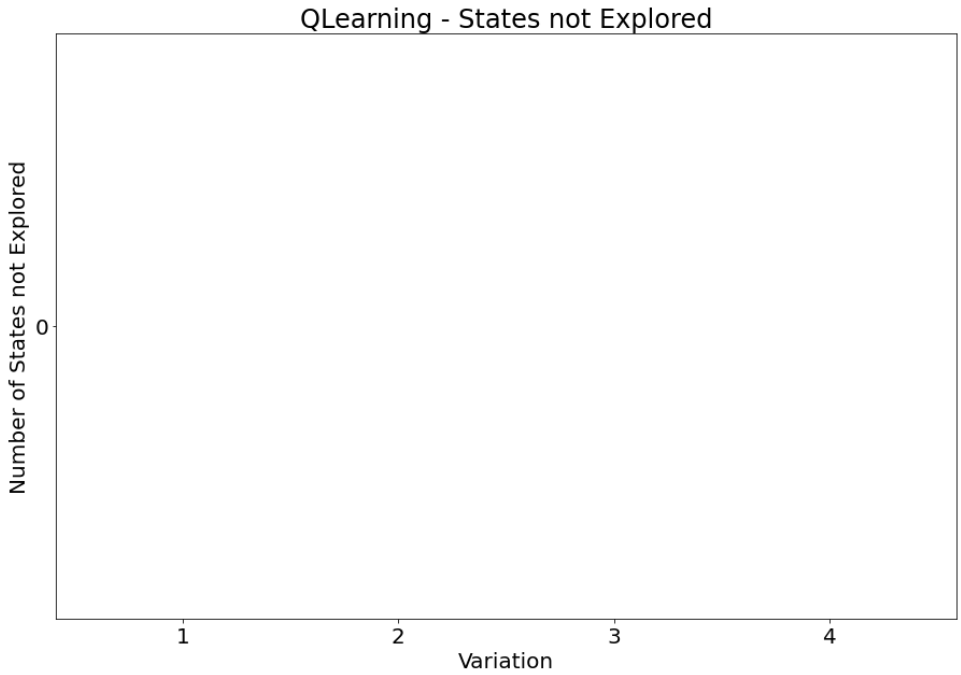

QLearning2 - Results
Losses
Wins
Draws
Wins/Draws/Losses
Episode


QLearning3 - Frequencies (top 100)
Frequency
State


QLearning3 - Results
Losses
Wins
Draws
Wins/Draws/Losses
Episode


QLearning4 - Frequencies (top 100)
Frequency
State


QLearning4 - Results
Losses
Wins
Draws
Wins/Draws/Losses
Episode


QLearning - States not Explored
Number of States not Explored
Variation

Discussion - The differences between each algorithm and algorithm configuration, and how exploration and exploitation had an effect on the performance of each algorithm.

This section will be dedicated to discussing the difference between each algorithm, the configurations of the algorithms (different epsilon values) as well as the performance contrast between the multiple set ups.

Algorithms

Starting off with Monte Carlo, this algorithm is the most basic one as it lacks the step size that the other algorithms include. Looking at the results of this concept, it is clear to see that the performance is the worst when it comes to winning when compared to the other algorithms. This low performance could be attributed to the fact that Monte Carlo does not update Q values until the end of an episode unlike the other two algorithms that update Q values after each state change. Hence learning is slower than TD learning models. The advantages of MC methods are that the estimates are unbiased due to actually finding the reward before updating the Q values. A particular MC variety which performed poorly compared to the rest is MC exploring starts. This makes sense since in this environment the only actions are HIT and STAND. Hence when choosing a random action at the start of the episode, there is a 50/50 odds that the method will choose either one, which can easily be a very bad option. Consider the case where the sum is 20 (cards being 10 and 10) and the method chooses HIT or the case where the sum is 13 and the method chooses STAND. The dealer has the biggest advantage when playing with this algorithm.

The Sarsa algorithm can be seen as an improvement on the previous model as not only does it include the step size feature, but also updates the state action values after each time step. In fact a slight enchantment in the winning percentages is observed. When compared to the other methods, Sarsa seemed to be quite an exploitative algorithm. This can be seen in its graphs in which the top state action pairs have a much larger frequency when compared to the other algorithms. One can speculate that this is due to the fact that Sarsa's Q values are inflated due to not getting the real reward every time an update is made. Sarsa also had overall a good performance level. In fact, Sarsa with $\epsilon = e^{-k/10000}$ managed to minimise the dealer advantage the most. (Dealer advantage being 0.13855). This is not a surprise, since this configuration manages to explore more state action pairs at the start and later on, as k grows, it exploits the policy. Resulting in a decent trade-off between exploration and exploitation.

The Q Learning models are off-policy models. Hence a Sarsa policy with ϵ greedy was used as a reference. However when Q learning is active it will always choose the best action according to the policy. Hence Q learning tends to exploit much more than explore. With this being said, it did not have the highest win percentages on average with the four configurations. However this might be due to the fact that the sample size that the algorithm had to go off was not huge, meaning that there could have been mistakes that were not ironed out with the low number of repetitions. It could be the case that it did not converge to the optimal policy. The performance was slightly behind that of Sarsa in this case.

Configurations

Beginning this discussion with the Monte Carlo simulation, it is quite clear to see that the exploring starts configuration had an extremely high amount of exploration with the randomised first choice, however this did not produce good results.

With regards to the four possible different configurations of the other two algorithms, there is a pattern that can be noticed by looking at the graphs and the numbers. Since they did not have a configuration of exploring starts it is easier to compare between them. In general, the two configurations that had the highest win percentage were the $\epsilon = e^{-k/10000}$ and the $\epsilon = 0.1$. This could be attributed to some factors. Regarding the $\epsilon = 0.1$ it could be that the model has a high exploration and keeps on exploring at a consistent rate, no matter the episode number. Thus, it has a higher chance of finding new states and moves that might be better than the current ideal move. For instance, if there is a state that has a high percentage of winning if it is hit on, but in this small chance that it fails, the algorithm is much less likely to go down that path again and it would start to favour a state that might not be the correct one. The optimal policy is reached by exploring many state action pairs and converging to the correct expected Q values. The $\epsilon = e^{-k/10000}$ configuration might have a good performance level as the more episodes go on, the less likely it is to explore and more likely it is to choose the ideal option and exploit. At the same time it takes more episodes exploring compared to $\epsilon = e^{-k/1000}$ configuration which starts to exploit the policy quicker. $e^{-k/10000}$ might be the best explore to exploit trade off configuration.

From the graphs of each algorithm, it is easy to observe that the $\epsilon = 1/k$ configurations are much more likely to exploit as opposed to explore. This can be seen with the frequency counts of the most likely states, as they have a much higher count than their counterparts.

Looking at the graphs that show the state action pairs that were not explored could not really tell you much about the algorithms or configurations. The values are so low and there is little correlation to them. The only piece of information that might be correct is that when the first configuration was run (the configuration with the most exploring incentive) there were no empty states in each of the methods. However, if all the algorithms were run again, there could have been completely different results in this specific piece of data. Once again, this can be attributed to pure chance. However given a configuration that keeps exploring (like ϵ = 0.1) and many episodes to run, the chances are that all states will be entered and explored.

Finally, looking at the BlackJack Strategy sheets, which displays what one should play, according to the model, given the current state. It is quite evident that the models converged to a good enough policy after 500,000 episodes. This can be concluded by analysing how the model chooses to Hit when the sum is a small number and chooses to Stand when the sum is a larger number. However a few exceptions can be noted, more specifically the choice to stand with a sum of 12 when having an ace. This for a reasonable player seems to be the bad option and would rather Hit, however some of the policies converged to Stand. One reason for this might be due to the rareness of the state (for the sum to be 12 with an ace, the hand must be double aces) hence the policy hasn't had enough experience with that state to be certain and converge to the ideal solution. This can be said for many other states. With that said the policies seem to Hit more when the user still has a playable ace, this is expected and correct (since the chance of losing is lower).

All in all, it is important to keep in mind that these results are not an exhaustive list or an end all be all. In fact almost every time the algorithms are run a slight variation of the graphs is observed. This is due to the very large sample state of a game of Blackjack and also the order of the episodes have an impact on the resulting policies (the wins and loses at the start of the training loop have a greater final effect). Moreover five hundred thousand episodes is not a lot in the grand scheme of things and all methods would benefit from an increased number of runs. Some of them might even converge to the optimal policy if allowed to run for a longer time.

Citations and references

[1] Wunderlich, K., Smittenaar, P. & Dolan, R. J. Dopamine enhances model-based over model-free choice behavior. Neuron 75, 418–424 (2012).
[2] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction. Pg 113-162 link: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
[3] B. Jang, M. Kim, G. Harerimana and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," in *IEEE Access*, vol. 7, pp. 133653-133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

Distribution of Work

It was made sure that every person contributed to about 50 marks.

Andrew Darmanin:

- Environment development
- Monte Carlo
- Q-learning
- Blackjack Strategy table
- Dealer advantage plot
- Report (contributing to about 1,400 words)

Edward Sciberras:

- Environment development
- Ace Implementation
- Sarsa
- Plots and Analysis of Performance
- Report (contributing to about 1,400 words)

Signatures: