

L-Università  
ta' Malta

**Coursework Title:** Blackjack Environment  
**Study Unit Name:** Reinforcement Learning  
**Study Unit Code:** ARI2204  
**Year:** 2024  
**Lecturer:** Dr. Josef Bajada

**Full Team Names and ID:**

Gianluca Aquilina: 348904L

Jacob Zammit: 320004L

## **Introduction**

This report will describe the implementation and the results of the reinforcement learning agents using different policies to develop a strategy to play Blackjack against the dealer. Our aim is to compare these algorithms to see their effectiveness.

## **Blackjack Environment**

### **Deck Creator**

First, we need to define the cards being used in Blackjack in order to use them for the game. We started implementing the environment by defining the '*Card*' class which represents the individual card and creating a constructor with 2 attributes:

1. **rank** – the number/letter on the card.
2. **suit** – suit of the card

Afterwards, we defined the '*Deck*' class which represents a deck of playing cards and in it, contains a constructor with an attribute, *cards*, which is the list that is holding *Card* objects representing a deck of cards, and a method, *reset()*, which its purpose is to shuffle the deck of cards by combining the rank and suit as a *Card* object and the shuffling the cards using the *random* module. Another method added in the class is *draw\_card*, which basically pops the last card from the *cards* list.

### **Blackjack Round**

Next, we need to create the actual round of the game of blackjack between the player and the dealer with the ability to interact with the game. The class '*BlackjackRound*' starts off with its constructor which initializes a new deck and creates lists dedicated for the player's and the dealer's cards; *player\_cards* and *dealer\_cards* respectively. This class is then followed up by five methods to enable interaction with the dealer in blackjack:

- a) **start** – used to start a new round of the blackjack. It starts by resetting and shuffling the deck and draws 2 cards for the player and 1 card for the dealer from the newly shuffled deck
- b) **hit** – if the player/agent decides to hit, then the player receives another card by appending the *player\_cards* list to increase the number of cards.

- c) **stand** – if the player/agent decides to stand, first, it checks if the sum of the dealer's cards is less than 17 as the dealer policy specifies that the dealer must stand if he has a higher sum than 17. If it is less than 17, then the dealer receives another card by appending the *dealer\_cards* list to increase the number of cards
- d) **get\_sum** – returns the total value of the cards. We, first, declare 2 variables, *total*, the total value of the cards, and *has\_ace*, which is acting as a Boolean flag to see if it has an ace or not. Next, it will go through the cards and adds them up. If the rank is either Jack, Queen, or King, then they add 10 to the total and if the rank is an Ace, then the total is added by 11 and the Boolean flag turns into True as an Ace is found. If, after adding them up, the total number of the card ranks are bigger than 21 and one of the cards is an Ace, then the total variable is decreased by 10 as, now the Ace is equivalent to 1 since, if it stayed equivalent to 11, then it will go over 21 hence losing the game.
- e) **get\_outcome** – it returns the outcome of the game. The method gets the sum of both set of player and dealer's cards using the *get\_sum* method mentioned above, and it returns 'Loss' when either the total number of the card ranks for the player is greater than 21 or, if the player decides not to risk and stands, the total number of ranks of the dealer's cards is greater than the sum of the player's cards. It returns 'Win' if either the total number of the card ranks for the dealer is greater than 21 or, if the player decides not to risk and stands, the total number of ranks of the dealer's cards is smaller than the sum of the player's cards. Last case, is a draw, since the last case for this scenario will be that both the dealer and the player have the same sum of card ranks.

## **Reinforcement Learning Agent Implementation**

The code has 2 methods that are needed for the RL algorithms implementation:

- a. **choose\_action** : the agent chooses an action based on the current state and the epsilon (the exploration rate) which are its parameters. So, firstly, it checks if the state is in the *q-values* dictionary and if it isn't, then it chooses a random action; either hit or stand. With the probability of the epsilon, it uses exploration by choosing a random action, whilst with the probability of 1 – epsilon, it uses exploitation by picking the action with the highest q-value for the current state.
- b. **generate\_agent\_state**: its purpose is to generate the state representation for the agent. It takes 3 parameters; *player\_sum*, the sum of the player's ranks, *dealer\_card*, the card that the dealer starts with, *has\_ace*, the Boolean flag to check if the player has an Ace, and these are displayed as a tuple which represents the state.

## **Monte Carlo On-Policy Control**

Our first algorithm going to be used is the Monte Carlo On-Policy. This algorithm uses the episodes of experience to estimate the best policy and action-value function. It uses the  $\epsilon$ -Greedy policy to guarantee exploration while converging towards the best policy. With this algorithm, there are 2 variants:

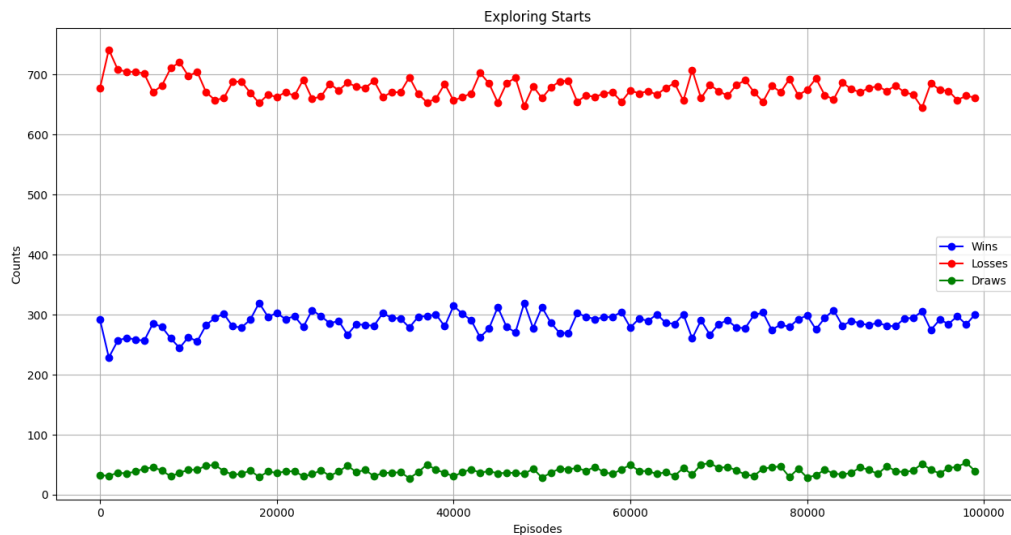
1. Exploring Starts: this variant is used to ensure that all state-action pairs are explored by each episode starting with a randomly chosen state-action pair and makes sure that the state-action pairs have a non-zero probability of being visited. As the number of episodes increases, all the state-action pairs would be selected many times.
2. Non-Exploring Starts: this variant means that at the start of each episode, the agent starts from an initial state and follows the policy to generate the episode. It relies more on the greedy policy for exploration.

The first implementation for the Monte Carlo On-Policy is the exploring starts. It starts by calling the *BlackjackRound* class to start the round. Afterwards, we generate the state of the agent based on the sum of the player's cards, the card of the dealer and whether the player has an ace. If the player's sum is between 12 and 20, then a random action is chosen, hence using the exploring starts variant. If otherwise, the action is chosen based on the  $\epsilon$ -greedy policy. The action and states for that current episode is stored in their lists respectively. If the random action is 'HIT', the player gets another card and if the player's sum is then over 21, the episode ends with a reward of -1 since the player lost. If the random action is 'STAND', it calls the *stand* method and then, gets the sum of both the player and dealer's cards. The rewards list will be appended by 1 if the player wins, -1 if the dealer wins, and 0 if they both got the same number. The current state is updated after every action and the next action is chosen using the  $\epsilon$ -greedy policy to balance exploration and exploitation.

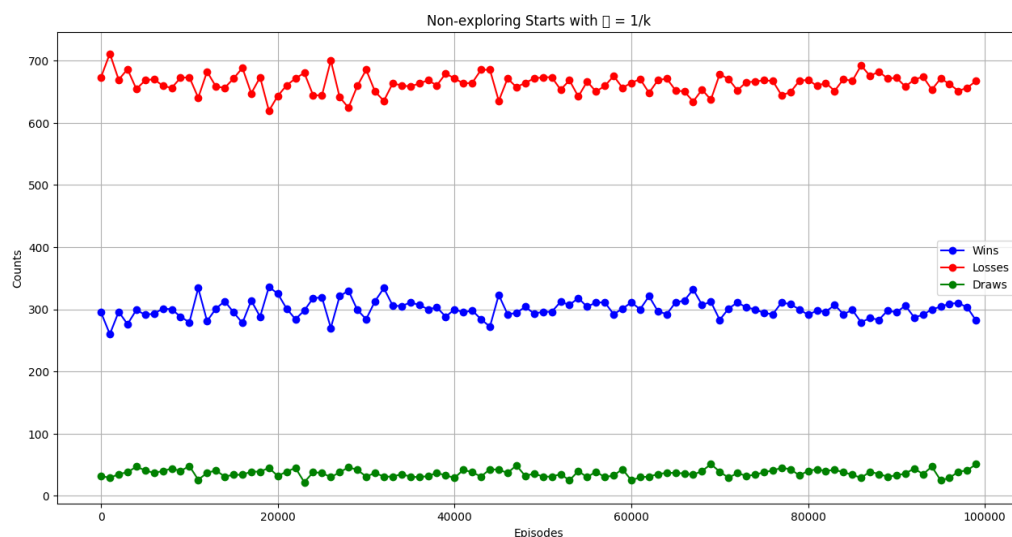
The second implementation is the non-exploring starts. It has a very similar implementation like the above but, unlike the exploring starts, the episode starts from the initial state of the Blackjack game rather than from a state-action pair. Then the agent uses the greedy policy to choose the action throughout the episode and to balance exploration and exploitation. The reward system is the same as above.

## **Statistics**

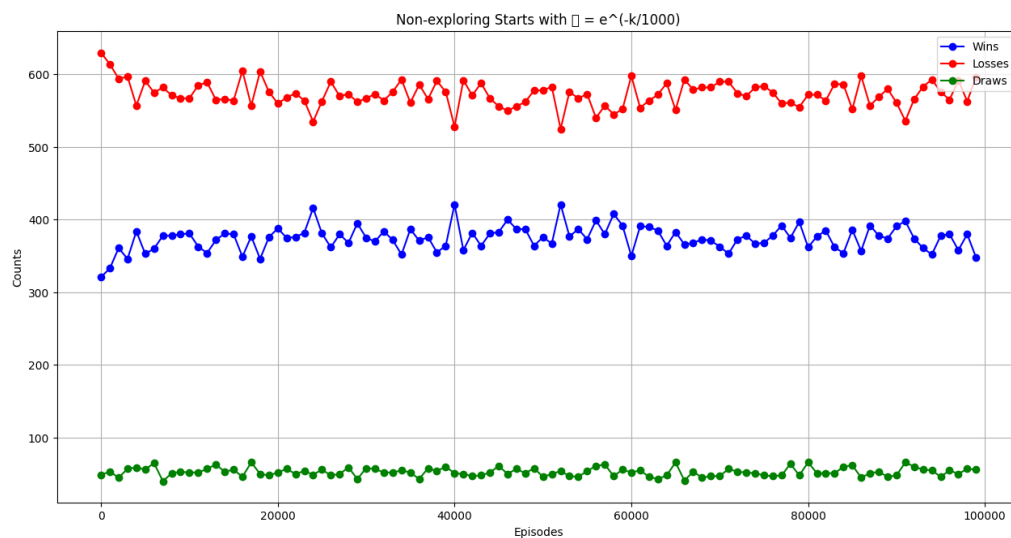
Let's start analysing the results from the total wins, draws and losses for each algorithm configuration



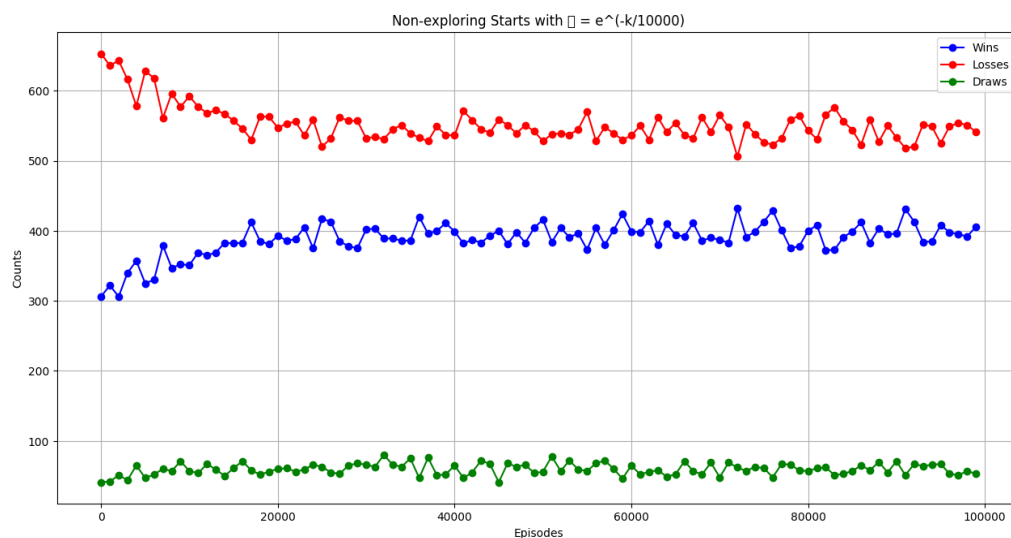
When you look at this graph, you notice that it doesn't show significant improvement in the number of wins as it is very consistent. Although exploring starts, make sure that the agent explores a wider range of states, it may not be as effective and a big performance difference compared to the non-exploring starts algorithms.



With this epsilon decay, this method explores more but as the number of episodes increase, it starts to exploit the policy more. Similar to the exploring starts, it is very consistent as the number of wins is always in the 300s and it is not improving significantly.

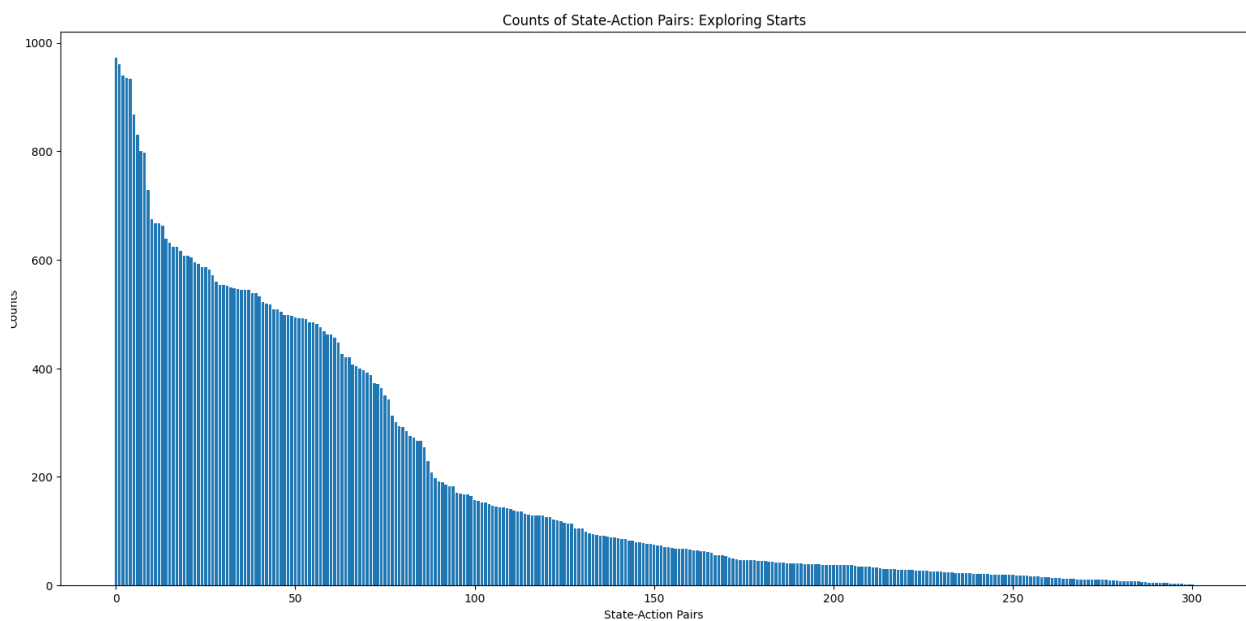


In this graph, you can see a better improvement in the number of wins and a decrease in the number of losses, which means that the policy, the more the number of episodes, continues to improve. This also shows a better balance between exploration and exploitation.

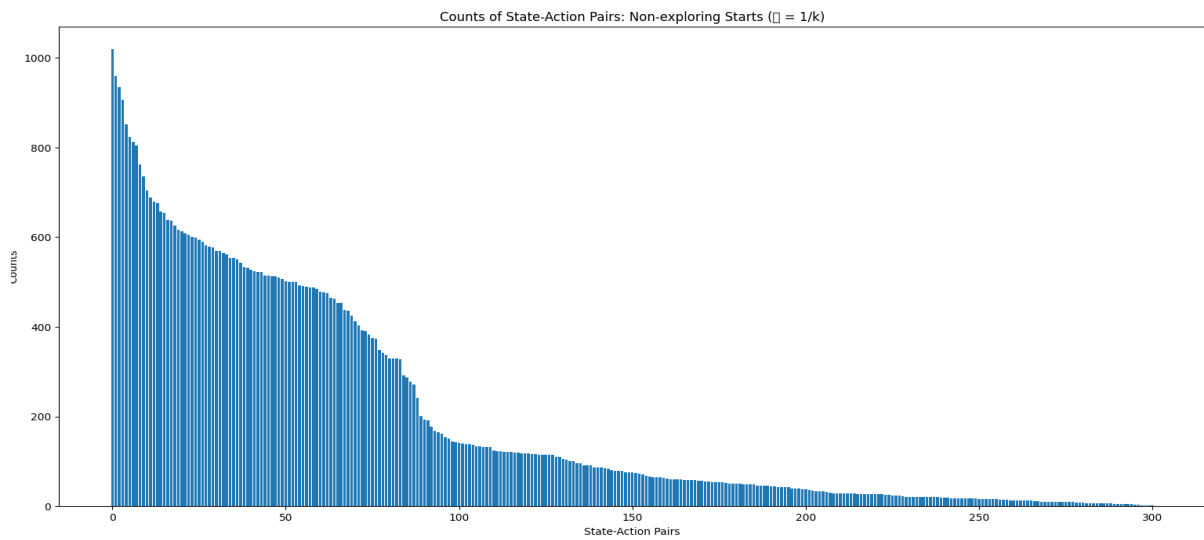


Here you can see that , since the epsilon is slower, therefore there will be more exploration for a longer period which allows the policy to improve even further. From this, compared with the others, shows that if you maintain the exploration for longer, a more optimal policy is discovered. In conclusion, we can say that configurations that have a slowed decay in exploration, tends to perform better in terms of balancing the exploration and exploitation and being more effective.

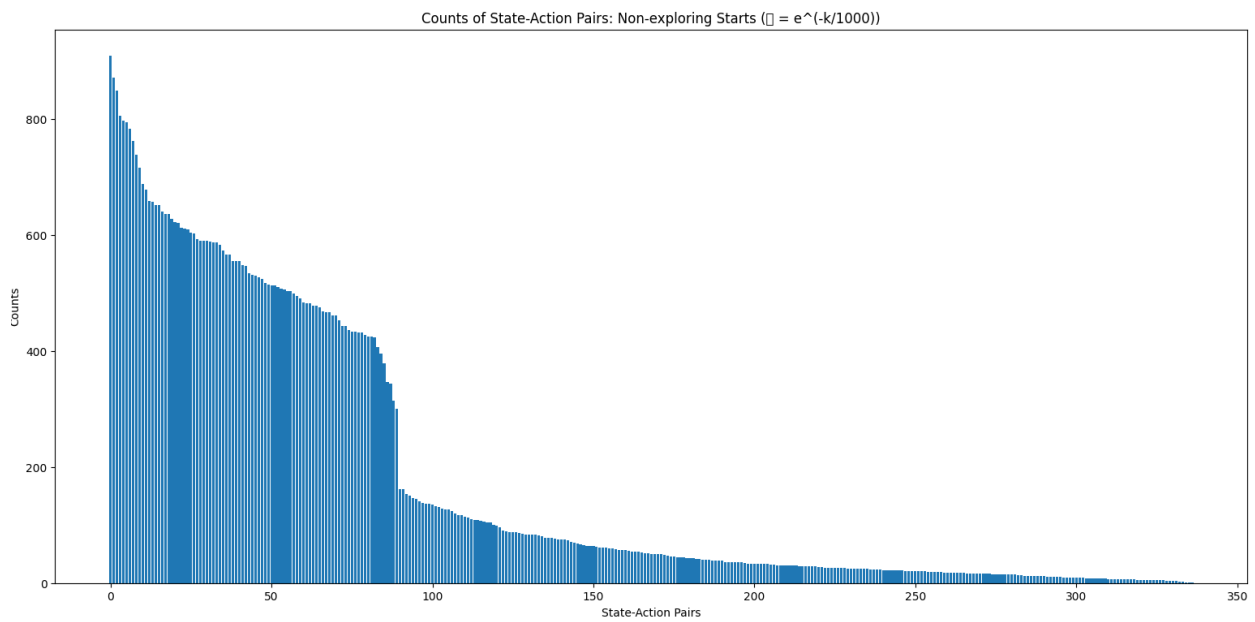
Now, let's look at the counts of each state-action pair by highest count first for each algorithm configuration:



From this analysis, the configuration visits quite a broad state space, as the agent is visiting evenly each state-action pair, as you can see by the gradual decrease of count. There will of course be pairs which are visited more than others but, it explores a wider range of pairs.

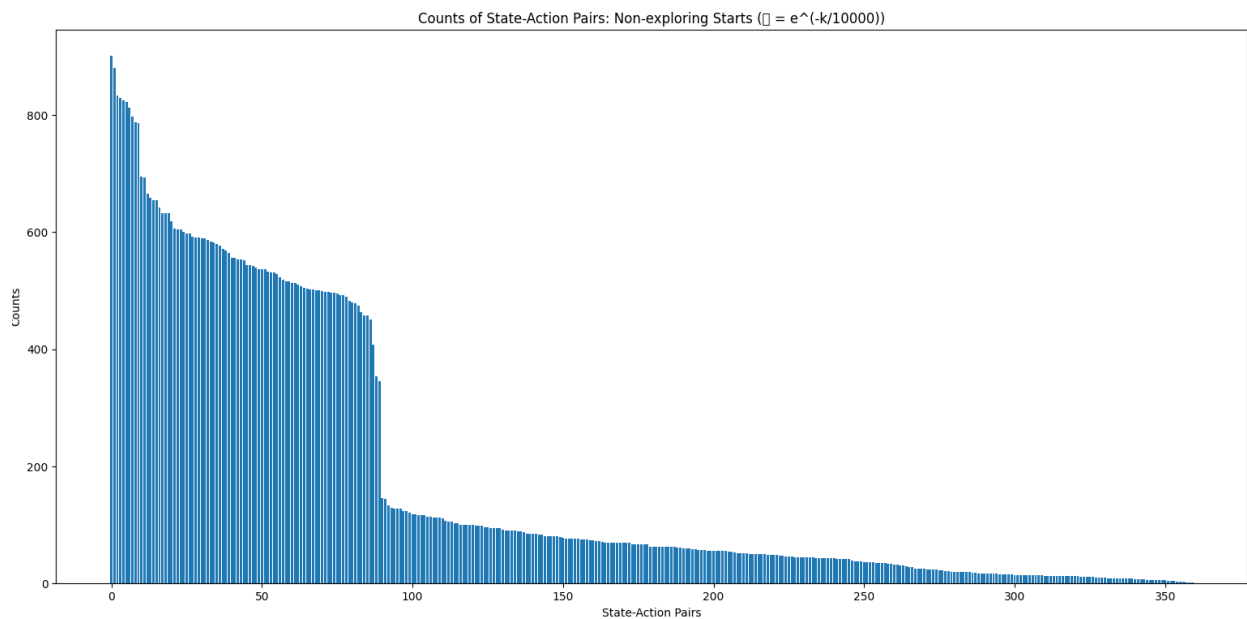


In this graph, you can see a slight sharper decline as the pairs become less frequent and a much higher count of the first few visited state-action pairs. This indicates that there is more a specific exploration.



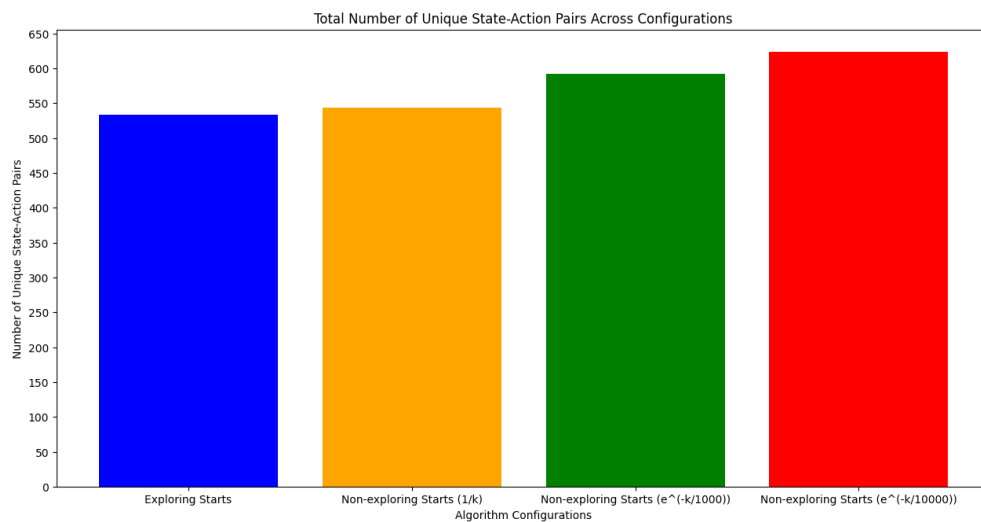
This is similar to the latter explanation but there is a slight less steepness of decline, which means that there was a broader range of state-action pairs, This suggest that there was a bit more balance between exploration and exploitation.





This graph achieves the broadest state space among the non-exploring starts as it visits a wider range of state-action pairs more evenly. In conclusion, both exploring starts and the non-exploring start with the slowest decay result in a broader state space coverage, which can mean that the policy and the performance starts to improve in the long run.

Let's now have a look at the total number of unique state-action pairs for each configuration:



From this graph you can see that non-exploring starts visit more unique state-action pairs than the exploring starts, and, the slower the decaying exploration rates, the more unique state-action pairs are discovered. This indicates that the exploration will get more extensive as the epsilon goes slower.

Finally let's have a look at the strategy tables formed for all configurations:

Strategy Table with Ace as 11 - Exploring Starts											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	H	S	H	S	S	S	S	H	S	S	
19	S	S	S	H	S	S	H	S	S	S	
18	S	S	H	H	S	S	S	S	S	H	
17	S	S	S	S	S	S	H	H	H	H	
16	H	H	H	S	H	S	S	H	H	H	
15	S	S	S	H	S	H	H	S	H	S	
14	S	H	H	H	S	H	H	H	H	H	
13	S	S	S	H	S	H	H	H	H	H	
12	S	S	H	S	H	H	H	S	H	S	
Strategy Table without Ace as 11 - Exploring Starts											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	H	H	S	H	H	H	S	S	H	S	
19	H	H	S	H	H	S	S	S	S	S	
18	H	S	S	H	S	S	S	H	S	H	
17	S	S	S	S	S	S	S	H	S	S	
16	H	H	H	S	H	H	H	S	S	H	
15	S	H	H	H	S	S	S	S	S	S	
14	H	S	S	S	S	H	S	S	S	H	
13	S	S	S	H	S	H	S	H	S	H	
12	S	H	S	S	S	S	S	S	S	S	

Both tables show that the agent prefers to stand with higher sums across all dealer cards and for lower sums, it is recommended to hit to increase the player's total. However, in the second table, it is being more conservative as it is standing on sums of 13 and higher when the dealer shows lower cards.

Strategy Table with Ace as 11 - Non-exploring Starts with $\epsilon = 1/k$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	H	S	S	H	S	H	
19	H	H	H	S	S	S	S	S	S	H	
18	H	S	S	S	H	S	S	H	H	H	
17	S	H	H	S	S	H	S	H	S	S	
16	S	S	H	S	S	H	S	H	H	S	
15	H	S	S	S	H	H	H	H	H	S	
14	H	H	H	H	S	S	H	H	S	H	
13	H	S	S	H	S	H	H	H	S	H	
12	S	H	H	H	S	S	S	S	H	S	
Strategy Table without Ace as 11 - Non-exploring Starts with $\epsilon = 1/k$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	H	S	H	H	H	H	H	H	S	H	
19	S	S	S	S	S	S	S	H	S	S	
18	S	S	S	H	S	S	S	S	S	S	
17	S	H	H	S	S	H	H	S	H	H	
16	S	H	H	S	H	S	S	H	H	H	
15	S	S	S	H	H	S	S	S	H	H	
14	S	H	H	H	S	S	S	S	S	H	
13	H	H	S	S	S	H	S	H	H	S	
12	S	S	H	S	S	H	S	H	H	S	

Similar to the latter, there is a clear stand on higher sums and hits on lower sums. In the first table, there is slight more variations due to probably more focused exploration and on the second table, there is a more defined strategy that of exploring starts.

Strategy Table with Ace as 11 - Non-exploring Starts with $\epsilon = e^{(-k/1000)}$											
Player	Sum	2	3	4	5	6	7	8	9	10	A
-----											
20		S	H	S	S	S	S	S	S	S	H
19		S	S	H	S	H	S	H	S	S	S
18		S	S	S	S	H	S	H	S	S	S
17		S	H	S	S	H	H	S	S	H	H
16		H	S	S	S	H	S	H	H	H	S
15		S	H	H	S	S	H	H	H	H	H
14		S	S	H	H	H	H	S	H	H	H
13		S	H	H	S	H	S	H	H	H	S
12		S	H	S	S	H	S	S	S	S	S
Strategy Table without Ace as 11 - Non-exploring Starts with $\epsilon = e^{(-k/1000)}$											
Player	Sum	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	H	S	S
18		S	S	S	S	S	S	S	H	S	S
17		S	H	S	S	S	S	S	S	S	S
16		S	S	S	S	S	S	H	H	S	S
15		S	S	S	S	S	S	S	S	H	H
14		S	S	H	H	S	S	S	S	S	H
13		S	S	S	S	S	S	S	S	S	S
12		S	S	H	S	S	H	S	S	S	S

Both tables look slightly more consistent and defined than the other 2 configurations  
 wi=hich means that the exploration and exploitations is starting to balance more. It, like  
 it should, stands on higher sums and hits on lower sums

Strategy Table with Ace as 11 - Non-exploring Starts with $\epsilon = e^{(-k/10000)}$											
Player	Sum	2	3	4	5	6	7	8	9	10	A
-----											
20		S	S	S	H	S	H	S	S	S	S
19		S	S	S	S	S	S	S	S	S	H
18		S	H	S	S	S	H	S	S	S	S
17		H	H	S	S	S	S	H	H	H	H
16		S	S	H	H	S	S	S	H	H	H
15		S	S	S	S	S	H	H	H	H	S
14		S	S	S	S	S	H	H	H	H	H
13		S	H	S	S	S	H	H	S	H	S
12		H	S	S	S	S	S	S	S	S	S
Strategy Table without Ace as 11 - Non-exploring Starts with $\epsilon = e^{(-k/10000)}$											
Player	Sum	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	H	S
18		S	S	S	S	S	S	S	S	S	S
17		S	S	S	H	S	S	S	S	H	S
16		H	S	S	S	S	S	S	S	S	H
15		S	S	S	S	S	S	S	H	S	H
14		S	S	S	S	S	S	S	S	S	S
13		S	S	S	H	S	S	S	H	S	H
12		S	S	S	S	S	H	S	S	S	S

These last tables have a very consistent strategy, due to prolonged exploration since there is a slow epsilon decay. In conclusion, explorations that has a longer epsilon decay leads to deep learning and a more effective policy.

## **Sarsa On-Policy Control**

The second algorithm implemented is the Sarsa On-Policy. This algorithm operates by choosing an action following the current epsilon-greedy policy and updates its Q-values accordingly. [1] It uses an epsilon-greedy policy.

The implementation includes 4 different configurations of epsilon;

Configuration 1 which has epsilon set to 0.1

Configuration 2 where epsilon is equal to 1/the current episode

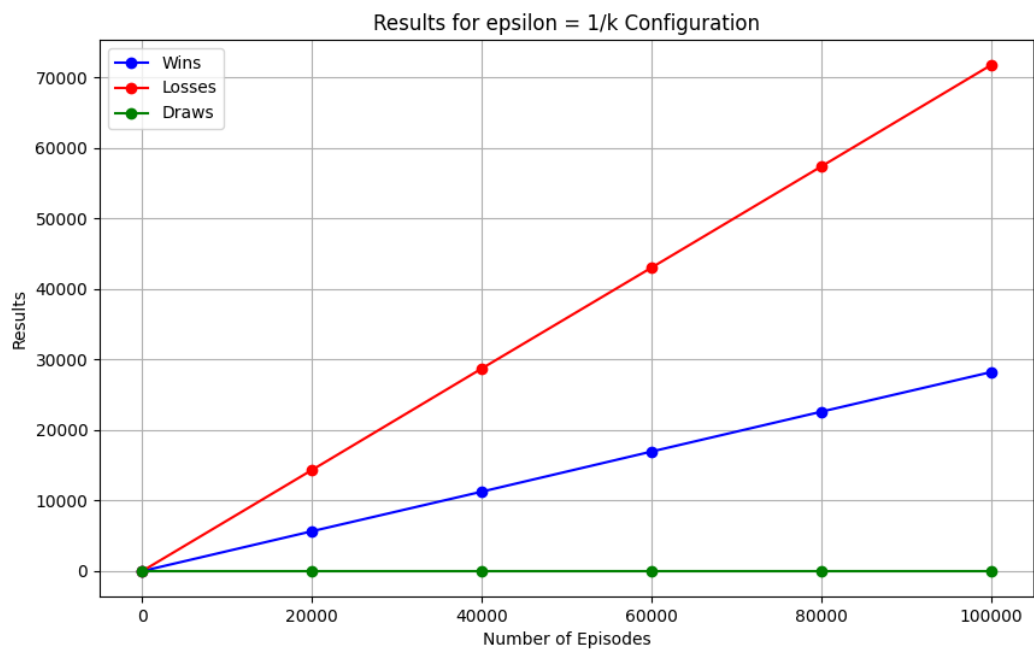
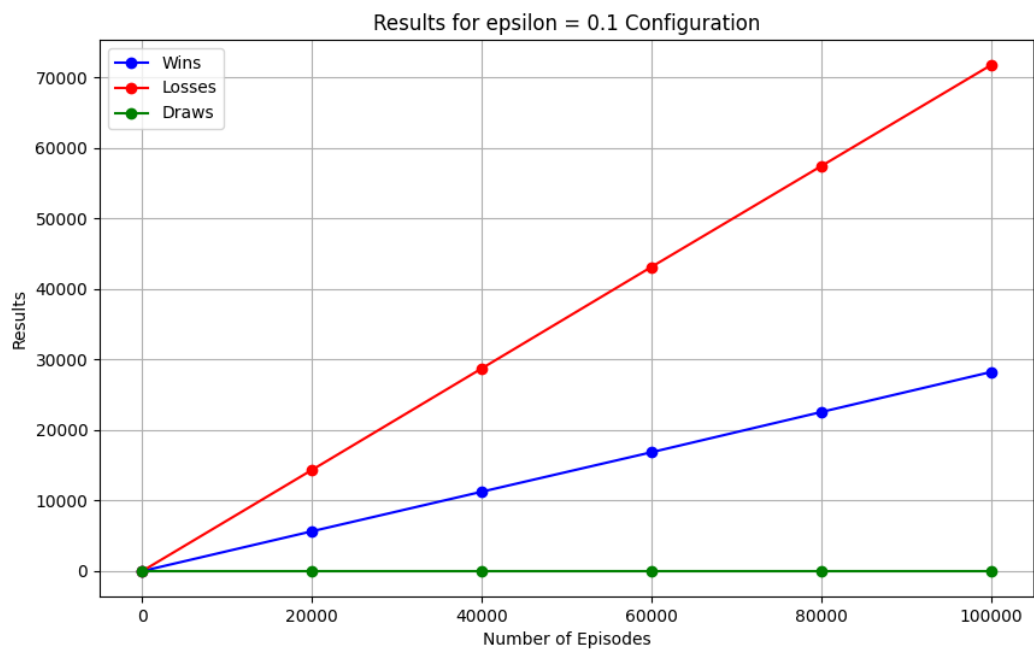
Configuration 3 where epsilon is defined as  $e$  to the power of  $-k/1000$ , where  $k$  is the current episode

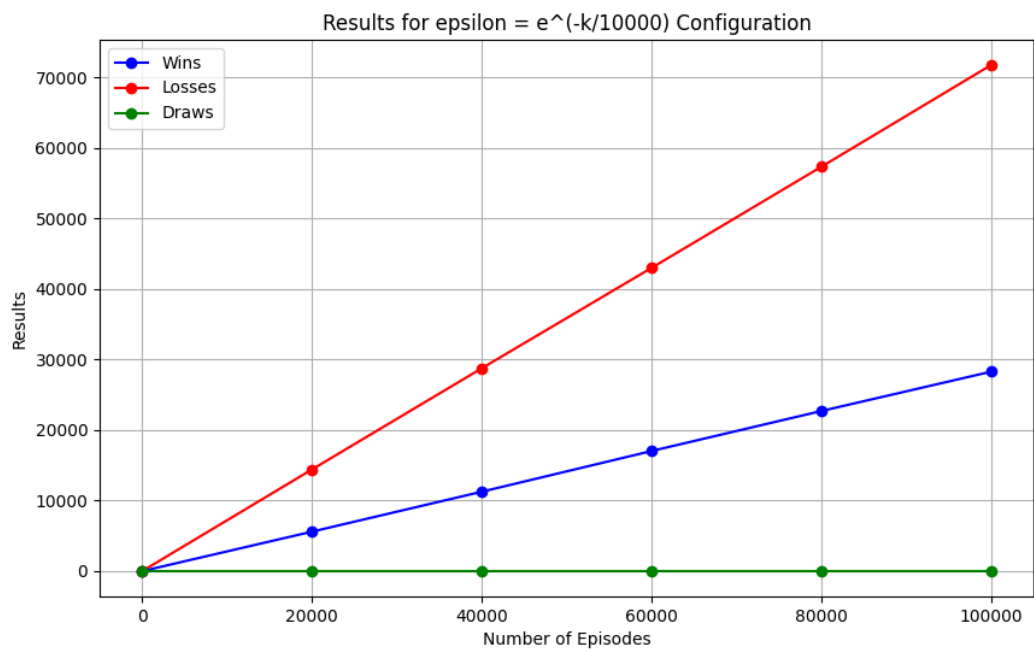
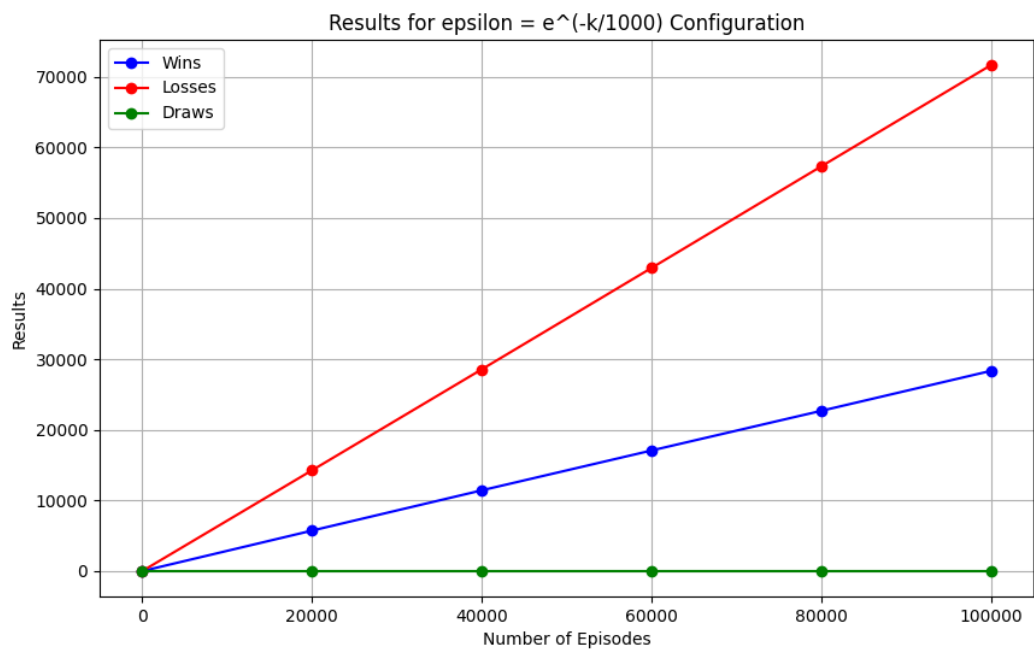
Configuration 4 where epsilon is defined as  $e$  to the power of  $-k/10000$ , where  $k$  is the current episode

Each configuration is run 100000 times, meaning 100000 games of blackjack were played for each configuration.

### **Statistics**

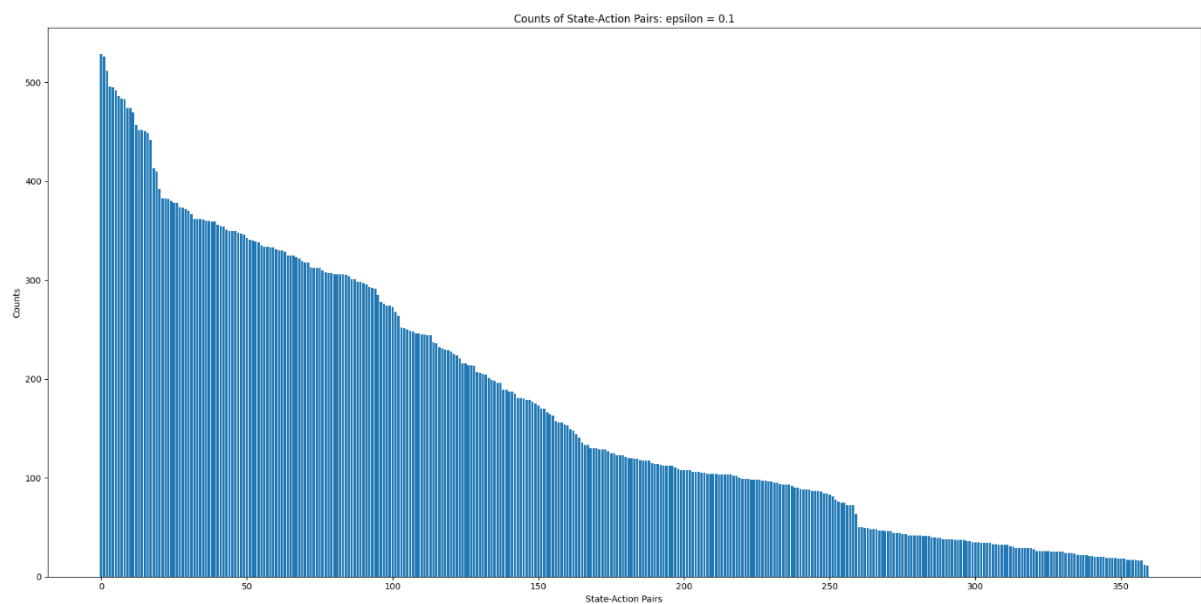
Every 1000 episodes, the amount of wins, losses, and draws obtained by each configuration are stored, and in the end, they are plotted on one-line-graphs. Here are the results for each configuration:



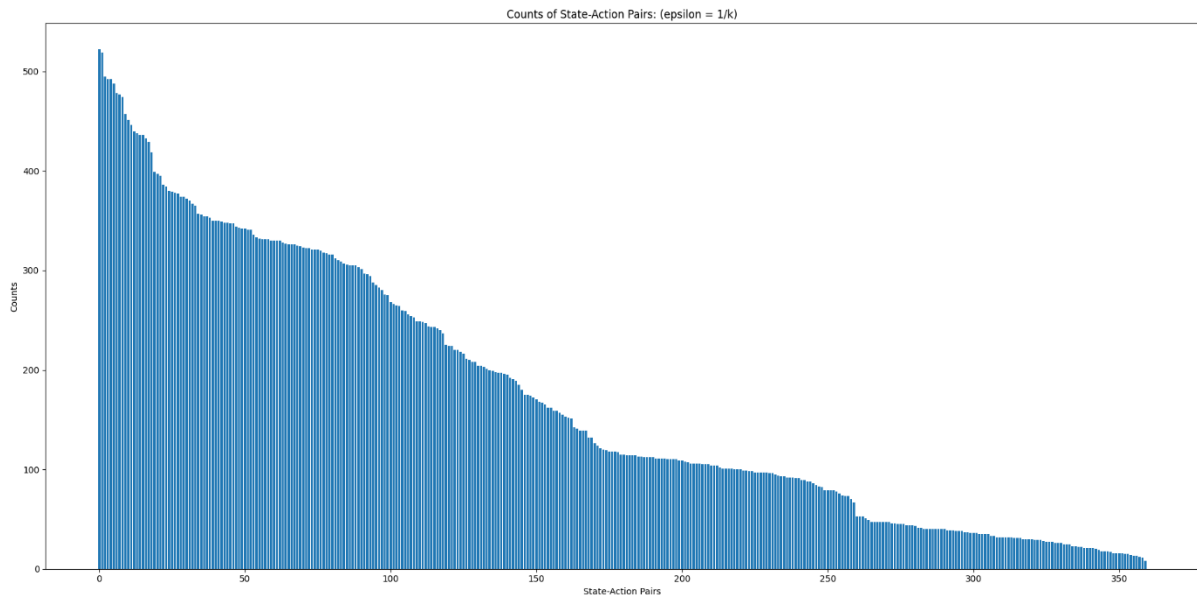




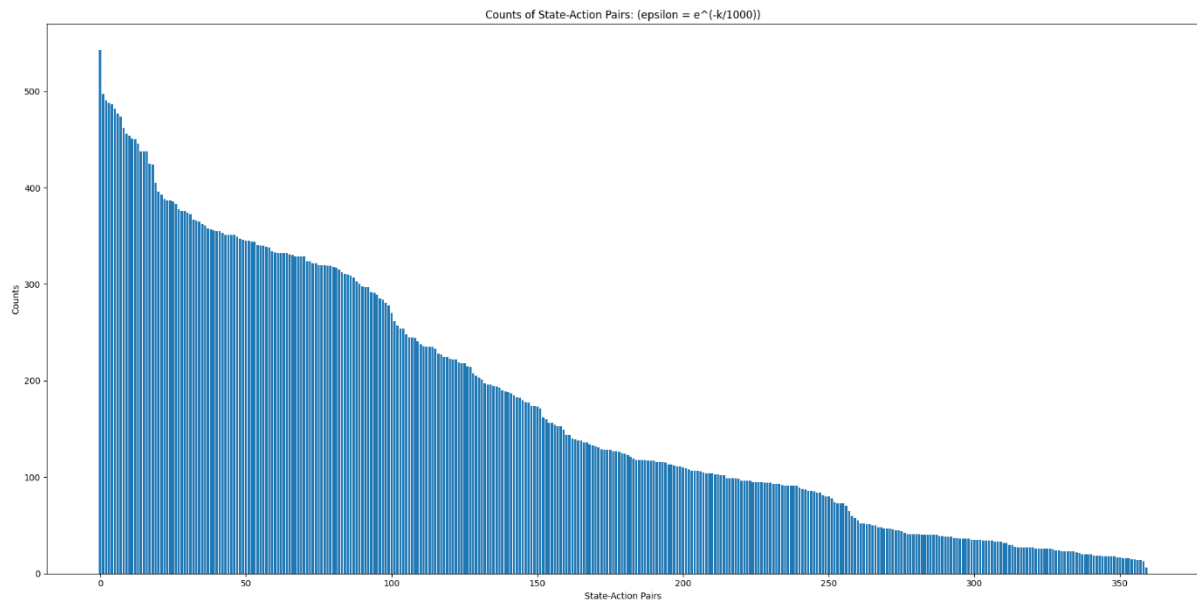
The counts of each unique state-action pair were tracked for each configuration, and these were plotted on bar charts, where the x-axis has the different state-action pairs, and the y-axis has the counts of these pairs. The bar charts are sorted by highest count first. Below are these bar charts for all four configurations of epsilon for Sarsa, and some discussion about the results.

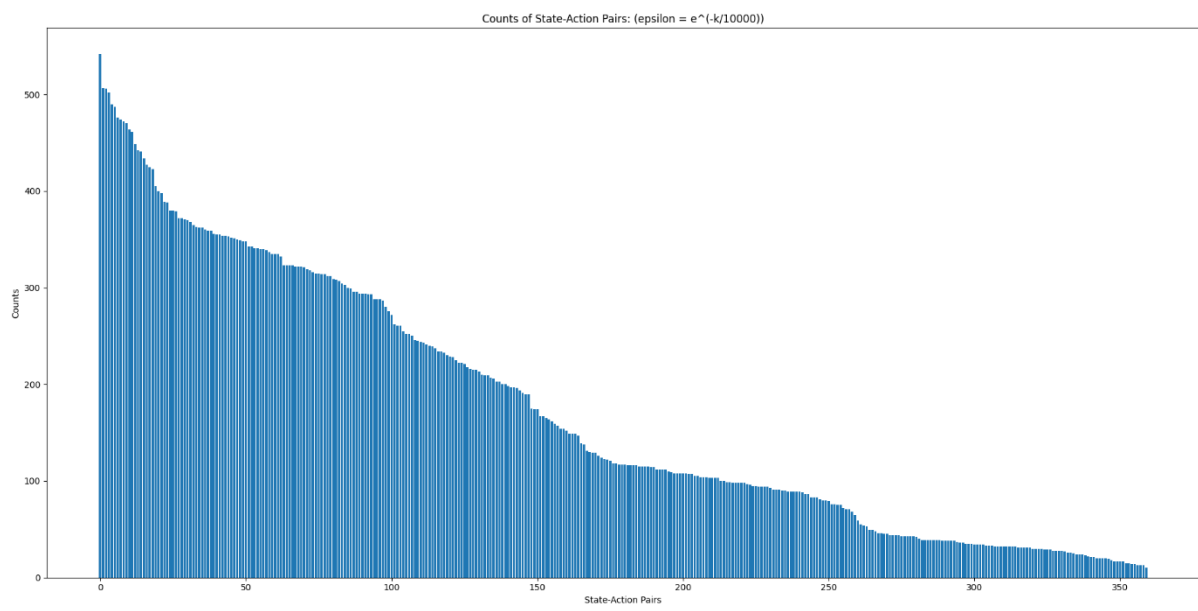


For configuration 1, where epsilon is always being set to 0.1, the chart has the least steep curve out of the four. This means the agent is visiting a wide range of pairs.



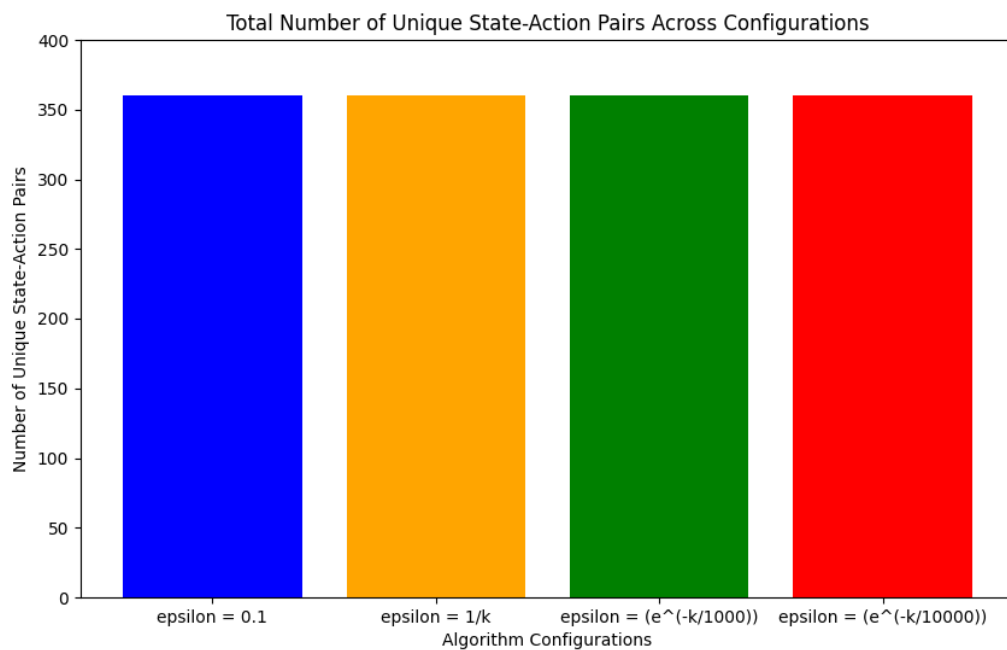
In configuration 2, where epsilon is  $1/k$  where  $k$  is the episode number, the agent still explores a lot.





Configurations 3 and 4 are less steep, indicating the agent started choosing smart actions quicker.

The total number of unique state-action pairs was plotted on a bar chart, in the figure below:



All configurations visited a similar number of unique state-action pairs.

Finally, Blackjack strategy tables were generated. Two tables were generated for each configuration, one where the agent is counting its ace as 11, and one where it is not counting its ace as 11.

Here are the strategy tables that were generated by the program:

Configuration 1:

Strategy Table with Ace as 11 - epsilon = 0.1											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	H	S	S	H	S	S	
19	S	H	S	S	S	S	H	S	S	S	
18	H	S	S	S	S	H	S	S	S	H	
17	S	S	S	S	S	S	S	S	S	S	
16	S	S	H	S	S	H	H	S	S	S	
15	S	H	S	H	S	S	H	H	S	S	
14	S	S	S	H	S	S	H	S	S	S	
13	S	S	H	H	S	S	H	S	S	S	
12	S	S	S	S	S	S	S	S	S	S	
Strategy Table without Ace as 11 - epsilon = 0.1											
Player Sum	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	H	S	
16	S	S	S	S	S	S	S	S	S	H	
15	S	S	S	S	S	S	S	S	H	S	
14	S	S	H	S	S	S	S	S	S	S	
13	S	S	S	S	S	S	S	S	H	S	
12	S	S	S	S	S	S	S	S	S	H	

Configuration 2:

Strategy Table with Ace as 11 - epsilon = 1/k											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S	H
18	S	S	H	H	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	S	S	S	S	S	S
15	H	S	S	S	S	S	S	S	S	H	S
14	S	S	S	S	S	S	H	S	S	S	S
13	S	S	S	S	S	H	S	S	S	S	S
12	S	S	S	S	S	S	S	S	S	S	S
Strategy Table without Ace as 11 - epsilon = 1/k											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	S	S	S	H	S	S
15	S	S	S	S	S	S	H	S	S	S	S
14	S	S	S	S	S	S	S	S	S	S	S
13	S	S	S	S	S	S	S	S	S	S	S
12	S	S	S	S	S	S	S	S	H	S	S

Configuration 3:

Strategy Table with Ace as 11 - epsilon = $e^{(-k/1000)}$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	H	S	S	S	S	S	S	S	S	S
19	S	S	S	H	H	S	S	S	S	S	S
18	S	H	S	S	S	S	S	S	H	S	S
17	S	S	S	S	S	S	S	H	S	S	S
16	S	H	S	S	H	H	S	S	S	S	S
15	H	S	S	S	S	S	H	S	H	S	S
14	S	S	H	S	S	S	S	S	S	S	S
13	S	S	H	S	H	H	S	S	H	S	S
12	S	S	S	S	S	S	S	S	S	S	S
Strategy Table without Ace as 11 - epsilon = $e^{(-k/1000)}$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	H	S	S	S	S	S
16	S	S	S	S	S	S	S	S	S	S	S
15	S	S	S	S	S	S	S	S	S	S	S
14	S	S	S	S	S	S	S	S	S	H	S
13	S	S	S	S	S	S	S	S	S	H	S
12	S	S	S	S	S	S	S	S	S	S	S

Configuration 4:

Strategy Table with Ace as 11 - epsilon = $e^{(-k/10000)}$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	H	S	S	S	S	S	
19	S	S	S	S	S	S	S	S	S	S	
18	S	S	S	S	S	S	S	H	S	S	
17	S	S	S	S	S	S	H	S	S	S	
16	S	S	H	S	S	H	H	H	S	S	
15	S	S	S	S	H	S	H	H	S	S	
14	S	H	S	S	S	S	S	S	S	S	
13	S	H	H	S	S	S	S	S	S	H	
12	S	S	S	S	S	S	S	S	S	S	
Strategy Table without Ace as 11 - epsilon = $e^{(-k/10000)}$											
Player Sum	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	H	S	
17	S	S	S	S	S	S	S	S	S	H	
16	S	S	S	S	S	S	H	S	S	S	
15	S	S	S	S	S	S	S	S	H	S	
14	S	S	S	S	S	S	S	S	S	S	
13	S	S	S	S	S	S	S	S	S	S	
12	S	S	S	S	S	S	S	S	S	S	

## Q-Learning Off-Policy Control

The third algorithm implemented is the Q-learning Off-Policy Control. This algorithm updates the q-values using the formula:

$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma(\max_{a'} Q(s',a') - Q(s,a))]$  where  $s$  is the current state,  $a$  is the action taken,  $r$  is the reward received after taking action  $a$  in state  $s$ ,  $s'$  is the new state after action,  $a'$  is any possible action from the new state  $s'$ ,  $\alpha$  is the learning rate ( $0 < \alpha \leq 1$ ) and  $\gamma$  is the discount factor ( $0 \leq \gamma < 1$ ). [2]

Q-learning manages exploration and exploitation, and converges to the optimal policy which gives maximum expected reward [2].

The Q-learning algorithm was implemented for Blackjack with an epsilon-greedy policy.

It was ran using 4 configurations of epsilon:

Configuration 1 which has epsilon set to 0.1

Configuration 2 where epsilon is equal to  $1/\text{the current episode}$

Configuration 3 where epsilon is defined as  $e$  to the power of  $-k/1000$ , where  $k$  is the current episode

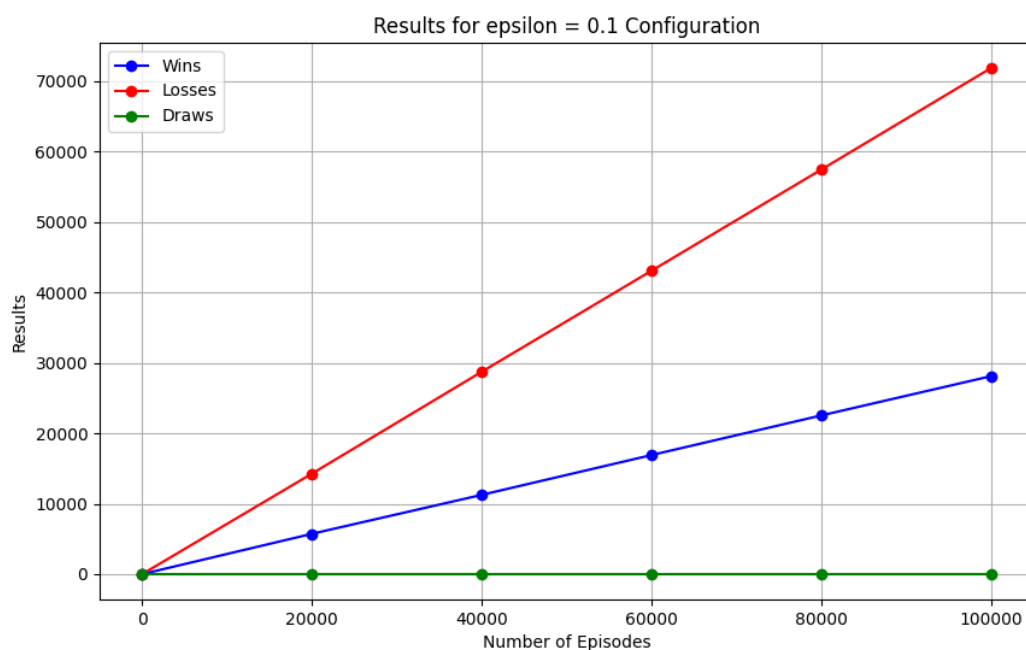
Configuration 4 where epsilon is defined as  $e$  to the power of  $-k/10000$ , where  $k$  is the current episode

Each of these configurations were run 100000 times, and certain statistics were gathered which will be discussed in the next section.

## **Statistics**

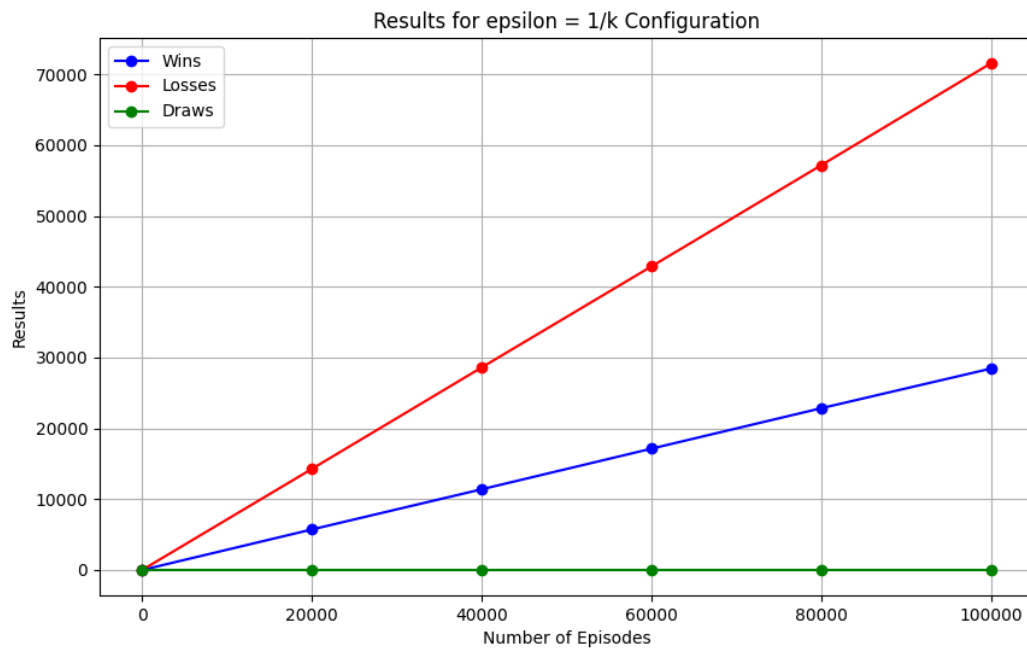
Here are the statistics of wins, draws, and losses obtained by each configuration:

Configuration 1:

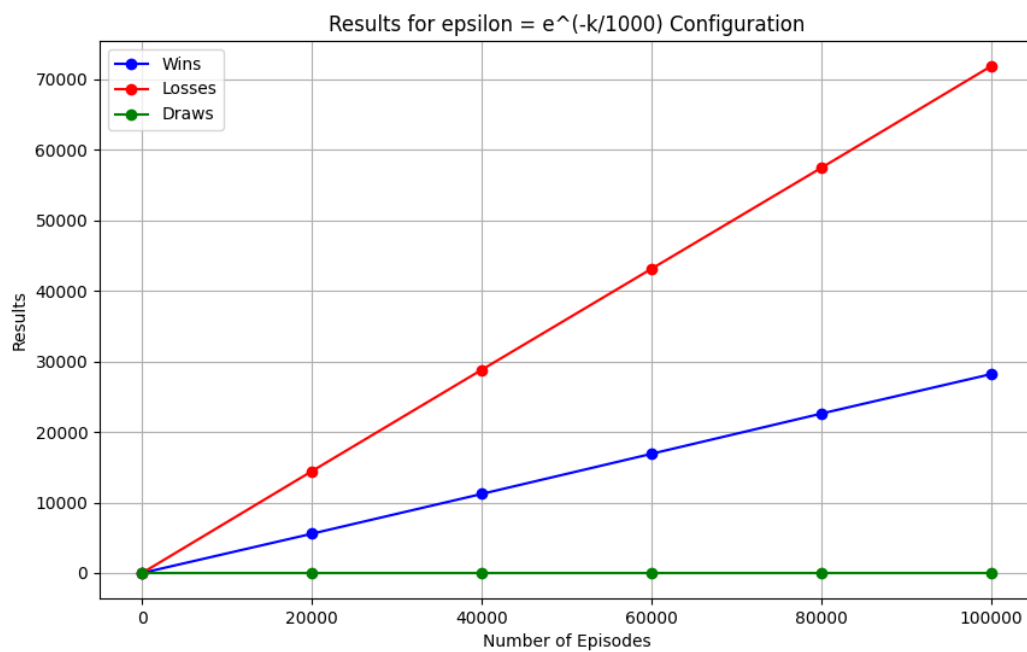




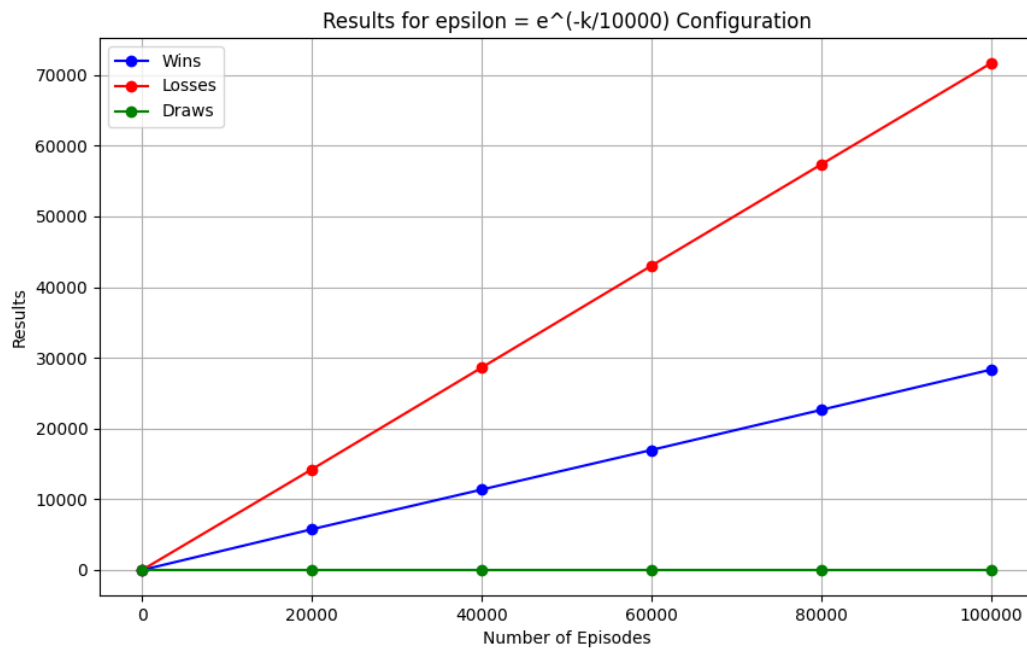
## Configuration 2:



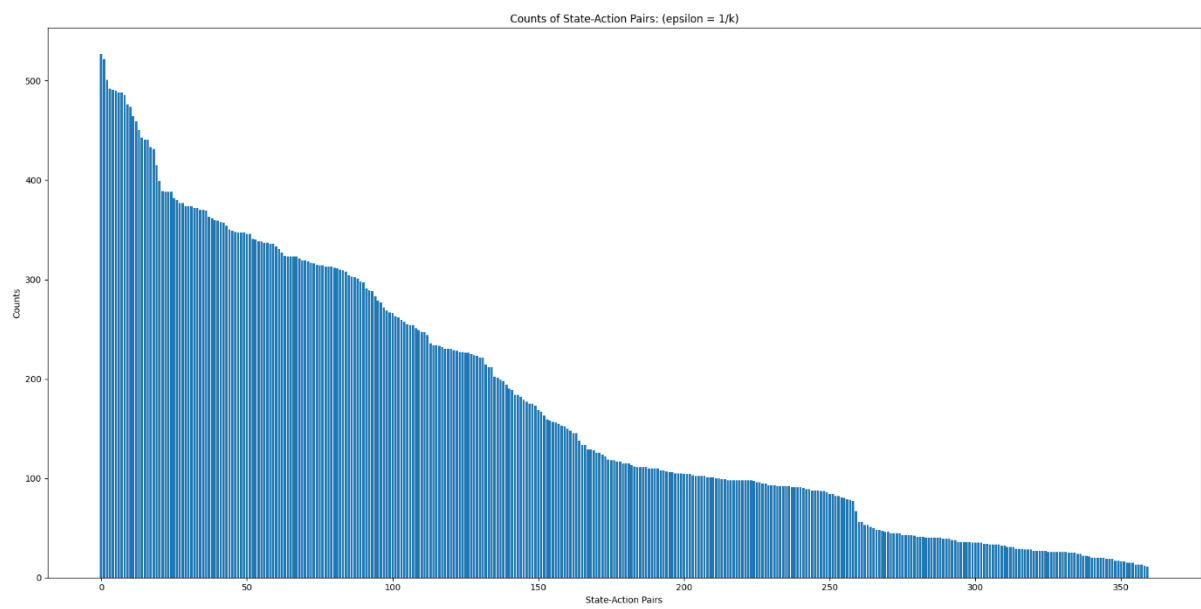
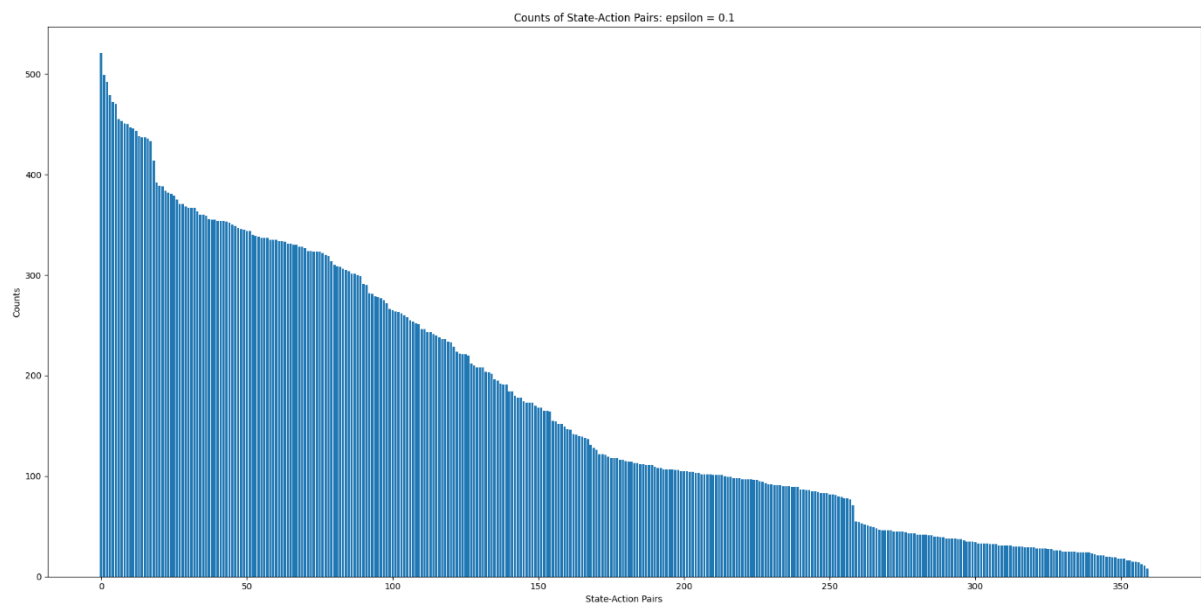
## Configuration 3:

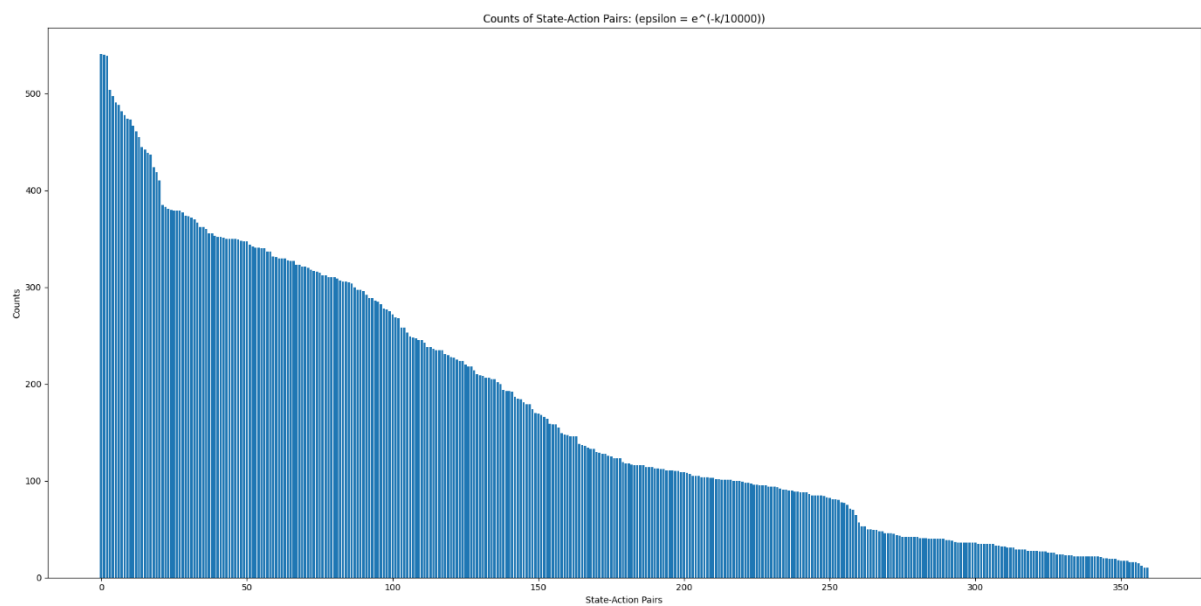
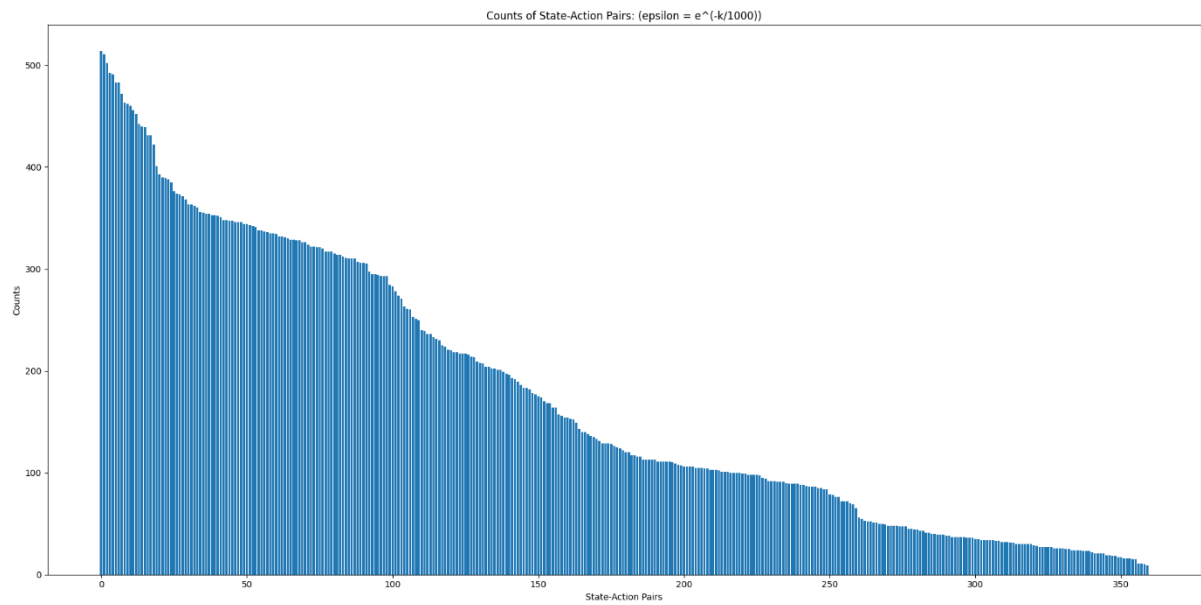


## Configuration 4:

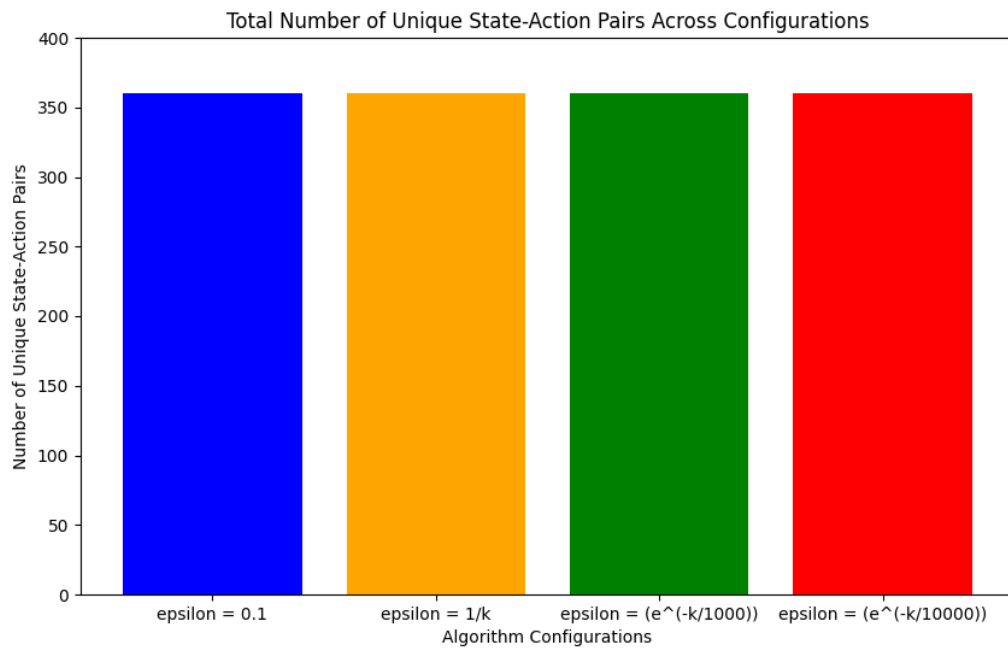


The counts of each unique state-action pair were tracked for each configuration, and these were plotted on bar charts, where the x-axis has the different state-action pairs, and the y-axis has the counts of these pairs. The bar charts are sorted by highest count first. Below are these bar charts for all four configurations of epsilon for Q-learning.





The total number of unique state-action pairs was plotted on a bar chart, in the figure below:



Finally, Blackjack strategy tables were generated by the program , two tables for each configuration, one where the agent is counting its ace as 11, and one where it is not counting its ace as 11.

Here are the strategy tables below:

Configuration 1:

```

Strategy Table with Ace as 11 - epsilon = 0.1
Player Sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A
-----
20 | S | S | S | S | S | S | S | S | S | S | S
19 | S | S | S | S | S | S | S | H | S | S | S
18 | S | S | S | S | S | S | S | S | H | S | S
17 | S | S | S | H | S | S | S | H | S | S | S
16 | S | H | S | S | S | S | S | S | S | H | S
15 | H | S | S | S | S | S | S | S | S | S | S
14 | S | H | S | S | S | S | S | S | S | S | S
13 | S | S | S | S | S | S | S | S | S | S | S
12 | S | S | S | S | S | S | S | S | S | S | H
Strategy Table without Ace as 11 - epsilon = 0.1
Player Sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A
-----

```

```

20 | S | S | S | S | S | S | S | S | S | S | S
19 | S | S | S | S | S | S | S | S | S | H | S
18 | S | S | S | S | S | S | S | S | S | S | S
17 | S | S | S | S | S | S | S | S | S | S | S
16 | S | S | S | S | S | S | H | S | S | S | H
15 | S | S | S | S | S | S | S | S | S | S | S
14 | S | S | S | S | S | S | S | S | S | S | S
13 | S | S | S | S | S | S | S | S | H | S | S
12 | S | S | S | S | S | S | S | S | S | S | S

```

Configuration 2:

```

Strategy Table with Ace as 11 - epsilon = 1/k
Player Sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A
-----
20 | S | S | S | S | S | S | S | S | S | S | S
19 | S | S | S | S | S | S | H | S | H | S | S
18 | S | S | S | S | S | S | S | S | S | S | S
17 | S | S | S | S | H | S | S | S | S | S | S
16 | S | S | S | H | S | S | H | H | S | S | H
15 | H | H | S | S | H | S | S | S | S | S | S
14 | S | S | S | H | S | S | S | S | S | S | H
13 | S | S | S | S | S | S | S | S | S | H | S
12 | S | S | S | S | S | S | S | S | S | S | S
Strategy Table without Ace as 11 - epsilon = 1/k
Player Sum | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A
-----
20 | S | S | S | S | S | S | S | S | S | S | S
19 | S | S | S | S | S | S | S | S | S | S | S
18 | S | S | S | S | S | S | S | S | S | S | S
17 | S | S | S | S | S | S | H | S | S | S | S
16 | S | S | S | S | S | S | S | S | S | S | S
15 | S | S | S | S | S | S | S | S | S | S | H
14 | S | S | S | S | S | S | S | S | S | S | S
13 | S | S | S | S | S | S | S | S | S | S | S
12 | S | S | S | S | S | S | S | S | S | S | S

```

Configuration 3:

Strategy Table with Ace as 11 - $\epsilon = e^{(-k/1000)}$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	H	S	S	S	S	S	
19	S	S	S	S	S	S	S	S	S	S	
18	S	S	S	S	S	H	S	S	H	S	
17	S	H	S	S	S	S	H	S	S	S	
16	S	S	S	S	S	S	S	S	S	S	
15	S	H	H	H	S	S	S	S	H	S	
14	S	H	S	H	S	S	S	S	S	H	
13	S	H	H	H	S	S	S	S	S	S	
12	S	S	S	S	S	S	S	S	S	S	
Strategy Table without Ace as 11 - $\epsilon = e^{(-k/1000)}$											
Player Sum	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	H	S	S	S	H	S	S	
15	S	S	S	S	S	S	S	S	S	H	
14	S	S	S	S	S	S	S	S	S	S	
13	S	S	S	S	S	S	S	H	S	S	
12	S	S	H	S	S	S	S	S	H	S	

Configuration 4:

Strategy Table with Ace as 11 - $\epsilon = e^{(-k/10000)}$											
Player Sum	2	3	4	5	6	7	8	9	10	A	
20	S	S	S	S	H	S	S	S	S	S	
19	S	S	S	S	H	S	S	S	S	S	
18	S	S	S	S	S	H	S	S	S	S	
17	S	S	S	S	S	S	S	H	S	H	
16	H	H	H	S	S	S	S	S	H	S	
15	S	S	S	H	S	H	S	S	S	H	
14	S	S	S	S	S	S	S	S	S	H	
13	S	S	S	S	H	H	S	S	S	S	
12	S	S	S	S	S	H	S	S	S	S	
Strategy Table without Ace as 11 - $\epsilon = e^{(-k/10000)}$											
Player Sum	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	S	S	S	S	S	
15	S	S	S	S	S	S	S	H	S	S	
14	S	S	S	S	S	S	S	S	S	S	
13	S	S	S	S	S	S	S	S	S	S	
12	S	S	S	S	S	S	S	S	H	S	

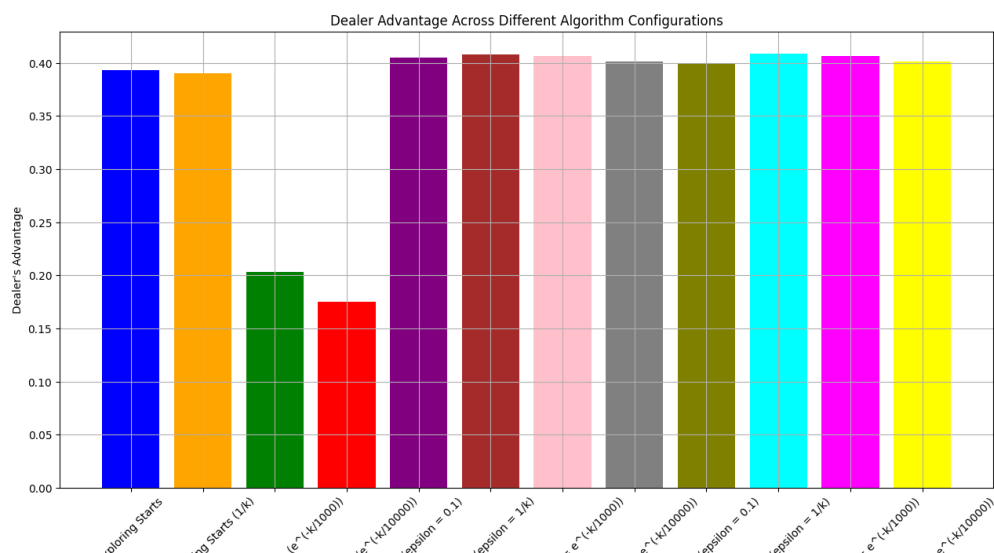
In configurations 3 and 4, the best action is to stand more often than in configurations 1 and 2.

References:

[1]: <https://builtin.com/machine-learning/sarsa>

[2]: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning>

## **Conclusion**



In conclusion, you can see that the Monte-Carlo Non-Exploring Start ( $e^{-k/10000}$ ) can minimise the most the dealer's advantage (the code is in the Monte-Carlo section). This result is possible as the epsilon decay is quite slow, so therefore, the agent has more time to explore the state-action pairs. In conclusion, the balance between exploration and exploitation influences the performance of the RL algorithms as if there is a lot of exploration, the agent will take longer trying out new actions instead of actually using the best strategy but, if there is a lot of exploitation, it can miss the better strategies, hence not producing the finest policies because it didn't have enough time to look at more state-action pairs. This balance is done by using the epsilon-greedy strategy which uses the probability of exploration to affect the balance of the two.



# FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

## Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We\*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our\* work, except where acknowledged and referenced.

I / We\* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

\* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Gianluca Aquilina

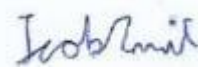
\_\_\_\_\_  
Student Name



\_\_\_\_\_  
Signature

Jacob Zammit

\_\_\_\_\_  
Student Name



\_\_\_\_\_  
Signature

ARI2204

\_\_\_\_\_  
Course Code

Blackjack Environment

\_\_\_\_\_  
Title of work submitted

24/05/2024

\_\_\_\_\_  
Date

## **Distribution Of Work**

### **Gianluca Aquilina 348904L**

- Implementation of the blackjack framework (card, deck and blackjack round)
- Implementation of Monte Carlo On-Policy Control
- Documentation part of Blackjack environment, reinforcement agent implementation and Monte-Carlo and with statistics
- Implementation of dealer chart with some arrangements to all 3 algorithms to make it work
- Documentation of conclusion

### **Jacob Zammit 320004L**

- Implementation of Sarsa On-Policy Control
- Implementation of Q-Learning Off-Policy Control
- Documentation of Sarsa and Q-Learning

Gianluca Aquilina

---

Student Name



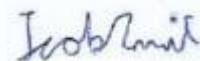
---

Signature

Jacob Zammit

---

Student Name



---

Signature