Artificial Intelligence Assignment

November 2020

Dots and Boxes using Reinforcement Learning

At,

Birla Institute Of Technology and Science, Pilani



By

Archit Golatkar - 2018A4PS0572P

Sohan Sadhu - 2018A4PS1112P

**Table of Contents**

1. Introduction

1.1 Purpose

1.2 Dots and Boxes

1.3 Q – Learning

1. Terminology
2. Implementation

3.1 State Space

3.2 Reward Policy

3.3 Exploration/Exploitation

1. Tests
2. Results
3. Discussions
4. Problems Faced
5. Learning Outcomes
6. Drawbacks
7. Future Plans

Acknowledgments

We would like to thank **Prof. Navneet Goyal** Sir for his constant support throughout the project. This project would not have been possible without his invaluable guidance. This course gave us a very good opportunity to apply all that we have learned during the lectures. It gave us a very good insight into the field of Artificial Intelligence.

Archit Golatkar

Sohan Sadhu.

**1. INTRODUCTION**

This project was done as a term paper for the subject Artificial Intelligence (BITS F444 / CS F407) at BITS Pilani. Artificial intelligence is one of the highest evolving fields of our generation. The improvements in the sector of data science and high power computation is taking place at a very fast pace. With the onset of Industrial Revolution 4.0 it has become extremely necessary for the upcoming professionals to have a clear idea of concepts such as machine learning , deep learning and artificial intelligence in general. In this project, concepts of reinforcement learning such as q-learning was used. The entire coding was done in C++ using the standard libraries and the results obtained were exported to MS Excel , where all the graphs were plotted.

**1.1 PURPOSE**

The ultimate aim of our project is to train the computer using an agent so that it can master the traditional game of Dots and Boxes, which is a highly strategic game. In this project we use an agent called a Q-agent to play the game. To train the Q-agent, it plays against various opponents, which are also certain agents. This report discusses the entire training process and analyses the performance of the Q-agent against various opponents, with varying values of learning rate and discount factor.

**1.2 DOTS AND BOXES**

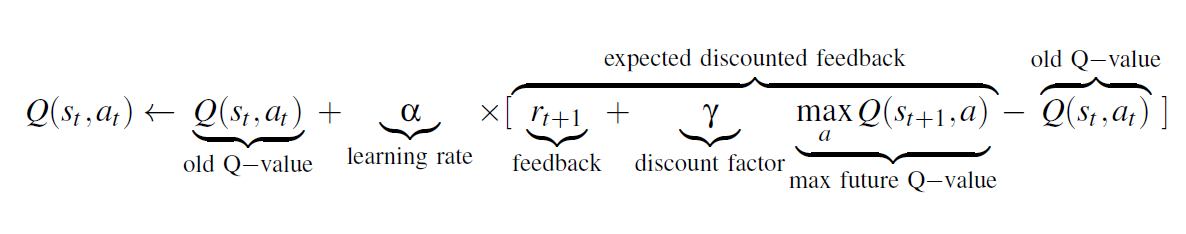
A traditional game, which is usually played on paper. It was introduced by Édouard Lucas in 1889. Board games such as chess or tic tac toe have fixed state space of 3x3 and 8x8, respectively. Unlike them, the state space of dots and boxes depends on the user's choice. It contains n x m dots (generally n x n) which are evenly spaced out in a form of a rectangular grid. In each move, a player draws a line to join two non-diagonal adjacent dots. If drawing a line creates a box in the game space, that player acquires the box and gets another move. This is a highly strategic game as sometimes a player needs to sacrifice a box so that in the future it may gain more boxes. The game ends when all possible lines are drawn. The winner is the player having the maximum number of boxes acquired. This project implements the q-learning algorithm which follows the rules of the game and gradually trains itself to master this game.

**1.3 Q-LEARNING**

Reinforcement learning is a paradigm of artificial intelligence which works on the basic principles of MDP. Using simple concepts like state, action, agents, environment, policy and reward, reinforcement learning can create wonders. Q-learning is a famous reinforcement learning algorithm. Its popularity is because of its easy implementation. Q-learning tries to learn a policy that maximizes the total expected reward.

At the core of Q-learning is the Q-table [state, action] which can be thought of like a table containing the probabilities of taking a particular action corresponding to a particular state. The Q-table provides the Q-learning agent a Q-value corresponding to a state action pair. A Q-value shows how good an action is for a particular state.

The Q-table gets updated following the below algorithm:



The Q-values are updated after each move of the learning agent till the game is over. Only one value is updated after one move. So for the feedback at later stages to propagate backwards to the beginning, it will require a large number of episodes. At the very beginning of the training process, the Q-agent has no idea of what its purpose is. Initially the agent will explore the states. For the first few episodes, the Q- agent will not learn much but as the number of episodes increases, it starts converging and learn the optimum Q-values.

The Exploration/Exploitation strategy decides whether the algorithm will reach a global or a local maxima. If the strategy is to always exploit the best possible action, then there are some chances that the algorithm converges to a local maxima. And if the strategy is to always explore, the algorithm might not converge to a solution. Therefore, a balance between exploration and exploitation must be maintained.

Learning rate and Discount factor are the two parameters that affect the Q-learning algorithm.

One of the few drawbacks of Q-learning is that it is memory intense. It stores the Q-values of all possible state action pairs in memory. This problem can be solved by implementing an artificial neural network to get the Q values.

**Q-Table :**

Let’s consider the following example for better understanding of Q- table. In the figure given below, we consider four dots and the possible lines that can be drawn (numbered 1 to 4).

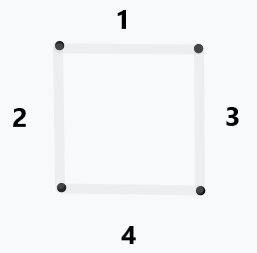


Fig 2

For every line, there can be two possibilities- either it's drawn or not. In the chart below, which can be thought of as a Q-table, the rows represent the number of possible states and the columns represent the actions. Note that, for each state not all the actions are possible. The actions which are taken before are marked by a **X.** The blank spaces are filled by the end of the training process. As mentioned earlier, the training process will be driven by the reward that the agent receives after it has accomplished a task. In the example given, the agent receives a reward whenever it completes the box. In the project. In the actual project, the agent receives a positive reward of 10 whenever it wins a box and a positive reward of 100 whenever it wins the game.

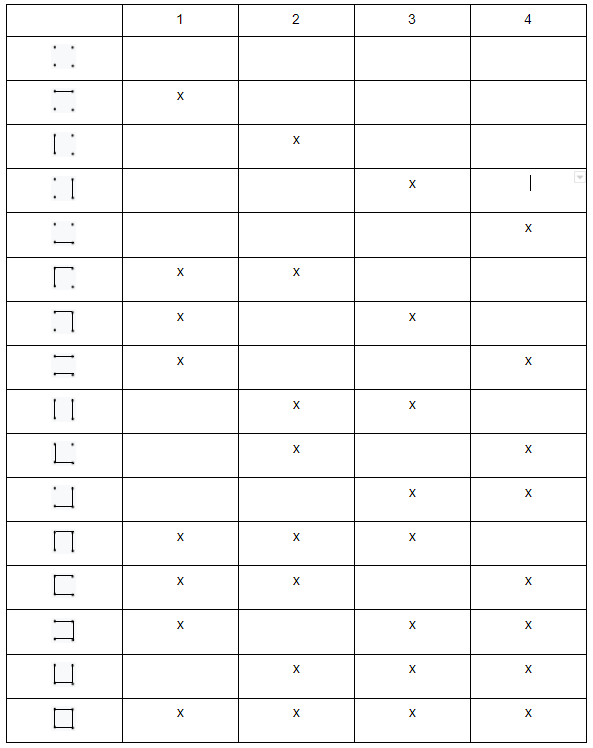


Fig 3 : State spaces for 4 dots (2x2)

**2. TERMINOLOGY**

1. **State space**: The collection of all possible states in a game. For the game containing 3 x 3 dots, there are 12 lines that can be drawn. So, for a state, line is either drawn or not resulting in a state space of 2^12 = 4096 entries.

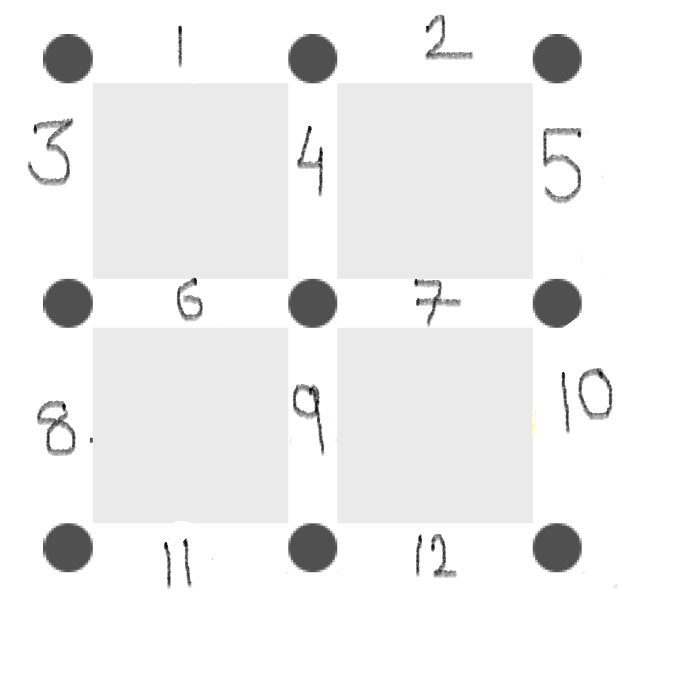


Fig1: Line numbering

1. **Action space**: The collection of all the actions that can be taken at a particular state is called action space. For the dots and boxes game, containing 3 x 3 dots the action space will include the line numbers as shown in fig1

1. **Agent**: An agent refers to an individual entity that is capable of taking a decision about what action to choose when subjected to an environment. In our case, the environment provides the agents the current state from which the agent chooses an action, i.e draw a line. The choice of action depends on some game rules and type of opponent agent.

Three types of opponent agents that we have implemented are explained below:

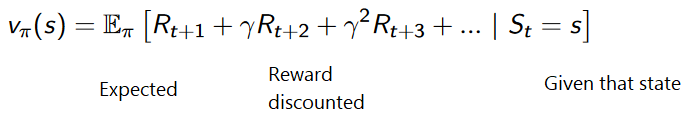
1. **Simple Agent**: This agent always takes an action of drawing the line with the smallest number from the available action space. I.e If available action space is {1,4,5,11}, a simple agent will select 1.
2. **Random Agent:** This agent takes the action of choosing a line randomly from the available action space. I.e the agent will choose anyone of (1,4,5,11) randomly.
3. **Q-Agent:** This agent uses the Q-learning algorithm to choose an action for a given state.

Environment only provides the agent with current state. It doesn’t provide any information about how the current state is reached i.e. it does not take into consideration the historical data. This property is called the **Markov property** which is the central idea of Reinforcement Learning.

1. **Learning Rate:** This is the factor in Q-learning that tells us how much priority is given to a new value. For example, learning rate of 1 means that the present value is replaced, while a learning rate of 0 means that the new value does not have any effect on the already present value. A high learning rate means that the learning process will be quick.

And a low learning rate means that the agent will take a longer time to learn a good policy.

1. **Discount factor:** The parameter in Q-learning that affects the importance of future rewards. A high discount factor increases the importance of future feedback. Here the learning agent will care more about the long term reward. Whereas a small discount factor implies that the importance of getting a reward at the current stage is better than getting a reward in future. (A dollar today is better than a dollar tomorrow)



Each reward in future will be discounted by a discount factor to the exponent of time step. As time progresses in a round, the importance of reward decreases.

1. **Exploration and Exploitation**: Exploration means that the agent will explore the possible states and not care about the optimal way to reach the goal. On other hand, exploitation means that the agent will use all provided information and exploit that to take the most rewarding action. We balance the exploration and exploitation by exploration quotient(𝛆).

We implemented a decaying 𝛆 greed approach where we slowly decay the epsilon overtime. [1,0.1,0.001].

1. **Reward:** This parameter is at the very heart of q-learning. The values of the q-table (which contains the state-action pair) is changed in accordance to the reward received by the agent from the environment for taking a certain action at a certain state.
2. **Policy:** It is the trained model which will tell the agent to choose the best action according to a given state

`

**3. Implementation**

* 1. **State space and Action Space**

As discussed earlier the possible number of states are huge and training a Q-agent in such cases is a problem pertaining to computation power. The q-table (Fig 3) for just 4 dots (from which only one box can be made) is also a very complex one. More number of dots would increase the state space exponentially. So, to reduce the state-space, we did our study on a 9 dots game (4 boxes).

For Q- table, instead of using a table in the form of an array, a hash map has been implemented. There are certain advantages of using the map data structure. Firstly, its memory allocation is dynamic which makes it computationally very efficient. Secondly, it assigns an unseen key to a value of zero when accessed first which is very suitable for initializing the q-values. But the problem encountered is that the q-table is like a function which takes in two inputs (current state and chosen action) and outputs a q value, which is non-implementable by the hash map. To make this work, the state action pair was considered instead, which is of string data type.

The example below will help us understand the methodology better.

We have used String data type to represent a state. So, the state will be of length 12 with each character as either '0' or '1'. Characters would contain '1' if that particular line is drawn, if not drawn it will be '0'. For example, the initial state will be "000000000000" when no line is drawn and the final state would be "111111111111" when all lines are drawn.

So state space will be a collection of states available. Ex. {"000000000000", "100000000000","010000000000",.......}

For action we have used integer datatype. So,from the above figure, an action will be a number corresponding to the line position. 1,2,3,....,12.

So action space will consist of all possible actions corresponding to a state.

Ex. For state "000000000000" action space would be {1,2,3,4,5,6,7,8,9,10,11,12}. And for state "100000000011" would be {2,3,4,5,6,7,8,9,10}

So the state-action pair will add the state and string form of action.

Ex. "100000000000" +toString(4) = "1000000000004".

So corresponding to state-action-pair Q value can be obtained. By Q\_table[state-action-pair],

* 1. **Reward Policy**

The reward policy had been set keeping in mind various factors. The reward is provided to the agent when (i) it scores a box, (ii) when it wins the game. We should keep in mind that the objective of the agent is to win the game and not just win the most easily accessible box. Many times it may happen that sacrificing a few boxes in the initial stages may help the player get the majority of the boxes in the future. It is obvious that the reward for winning the game should be much more than the reward for winning a box. It is also worth noticing that the maximum number of boxes that can be scored due to a move(single line drawn) is two. **So the reward on winning the game should be much greater than even twice the reward for winning a box.**

For this purpose, we have kept Reward for winning a box = 10, Reward for winning a game. = 100 and for losing a game = -100.

* 1. **Exploration v/s Exploitation**

Both the techniques are used in the q-learning algorithm. When the training process starts , the q-agent has no idea regarding the best moves. So in that case, it explores the state space by taking random actions. As the number of games played increases, the q-agent starts gaining knowledge about the possible actions (due to the feedback policies) and starts exploiting them so as to choose the optimum action.

The action chosen is highly dependent on the values of the q-table. The ability to update an existing q-value is dependent on two factors- learning rate and discount which have been explained before. There are various techniques that use both-exploration and exploitation such as softmax and epsilon greedy. We have used decaying epsilon ***𝜖***-greedy method where the exploration coefficient gradually decreases with the number of games played resulting in more exploitation attitude.

**4.**  **Tests:**

For learning, each of the combinations of agents played only 70,000 games due to time constraints. The results can be improved if trained on more games. Statistics were stored after every 1000 games. The tests are made with different combinations of learning rate and discount factor to show their importance on performance of Q-learning agent. For agents with no prior memory, few combinations of learning rate and discount factors are tried out. Due to lack of time only a couple of combinations have been tried. The parameters which showed the best results were used to train further in upcoming runs.

|  |  |  |
| --- | --- | --- |
| **First Player** | **Second Player** | **Name of Agent Produced** |
| Q - learning Agent | Random Agent | Q1 |
| Q - learning Agent | Q - learning Agent | Q2 |
| Q - learning Agent | Simple Agent | Q3 |
| Q1 | Q - learning Agent | Q4 |
| Q2 | Random Agent | Q5 |
| Q4 | Q5 | Q6 |
| Q3 | Q1 | Q7 |
| Q1 | Q2 | Q8 |

Figure 4 : Table showing the order in which tests were performed.

**5. Results:**

The results are represented by graphs showing how successful a Q -learning agent was against a particular agent. The success is interpreted by the share of winnings.

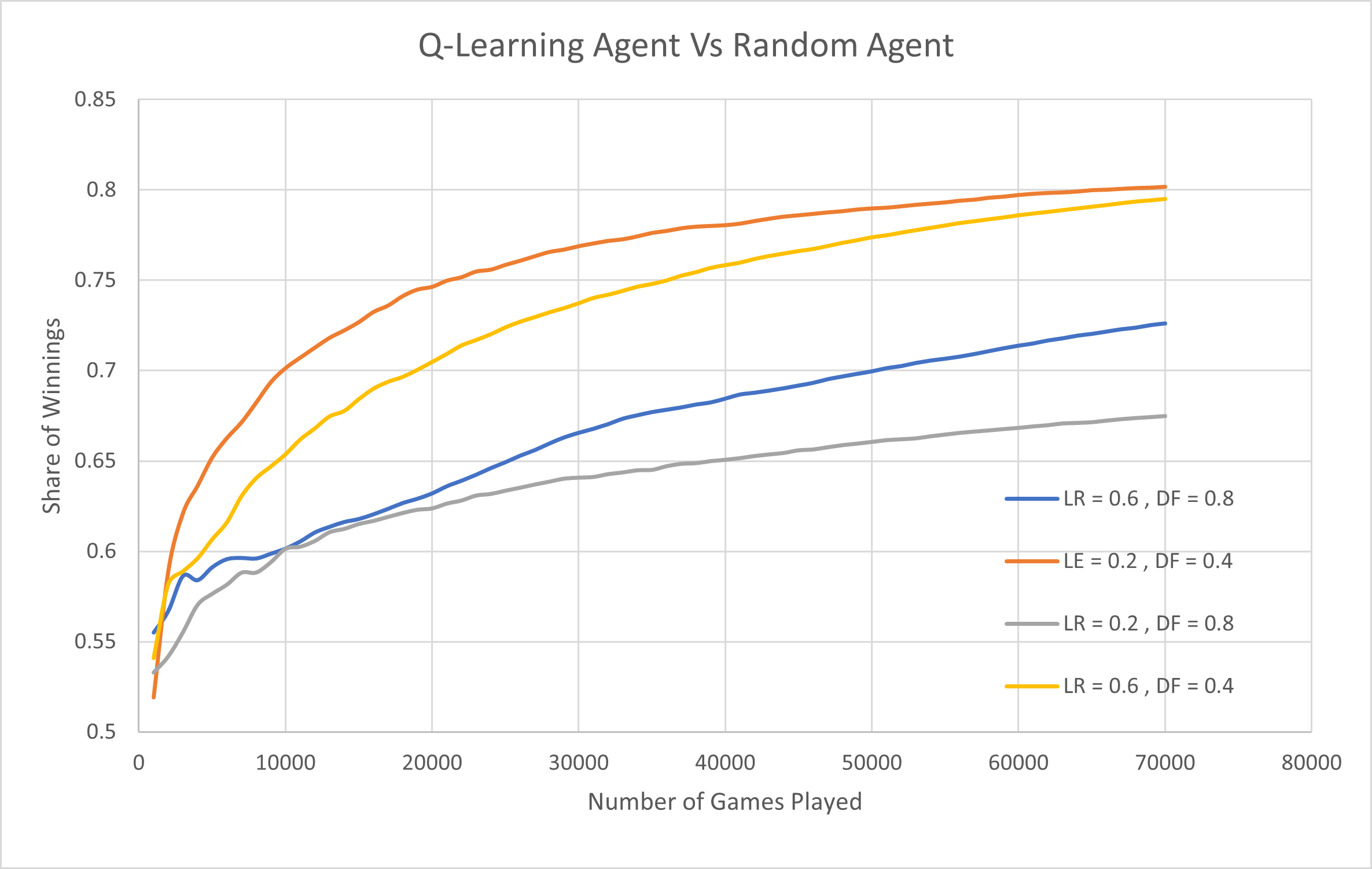
Share of Winnings (SOW) = Number of games won by Q agent / Number of games played.

Following are the results and discussion for each case.

1. **Q1 : Q-learning Agent vs Random Agent**

A Q-learning agent without any previous memory was played with Random Agent and the policy learned was stored as Q1.

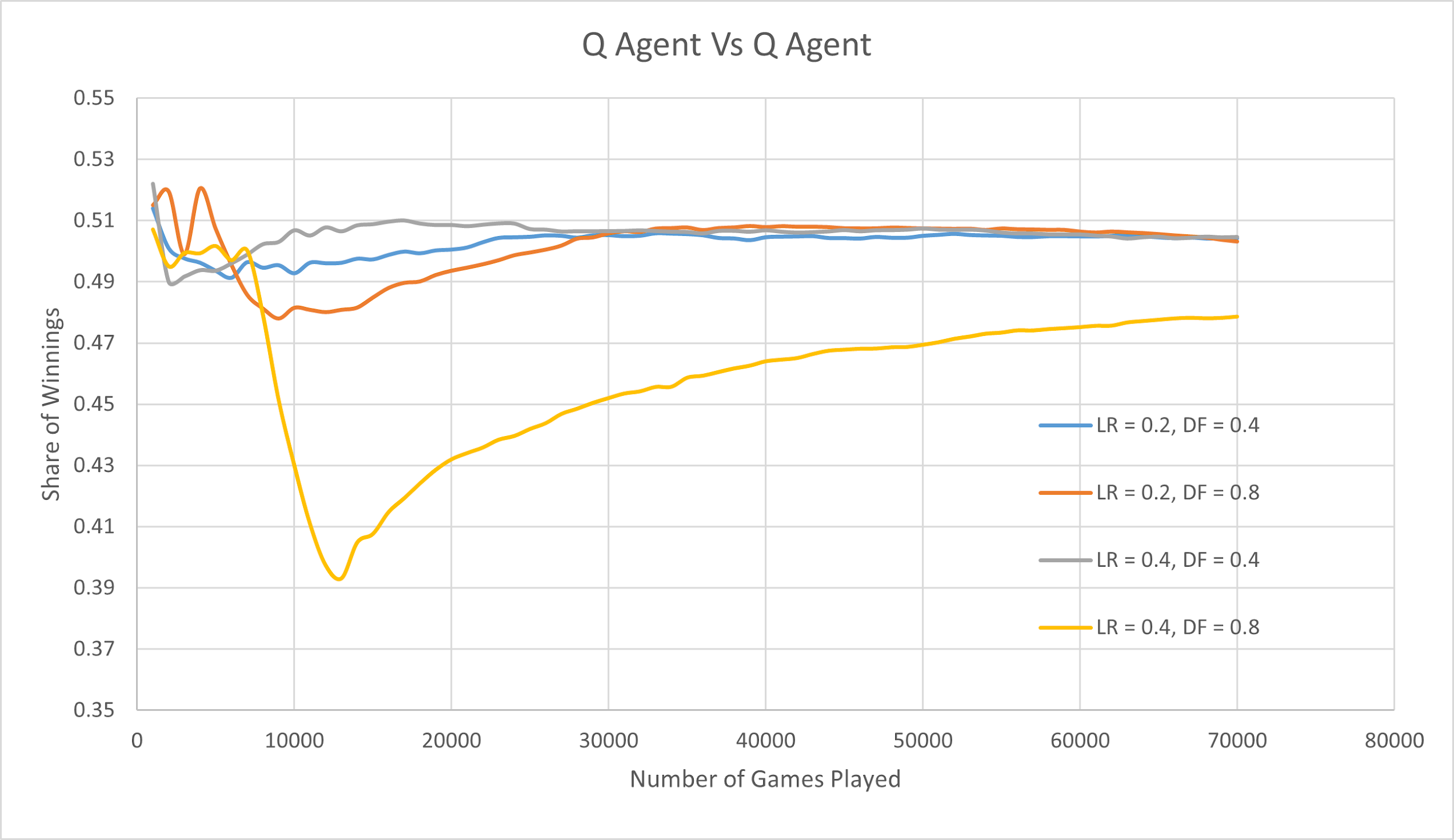
Figure 5 clearly shows that the learning rate of 0.2 and discount factor of 0.4 are most successful. However this result cannot be considered as final as the graphs are not yet converged after 70,000 rounds. Clearly, more games would have resulted in higher accuracy and proper interpretation of parameters(alpha and gamma).



*Figure 5: Q - learning agent Vs Random Agent*

1. **Q2 : Q-learning Agent vs Q-learning Agent**

This test was made with two Q-learning Agents without prior memory and the learned policy of the first q-agent was stored as Q2. The results in the figure shows that both of the agents have win ratio close to 0.5. Both agents are learning with the same learning rates and discount factors and almost equal experience. The cause of the small deviation in the first few thousand games is probably chance. Eventually with time , the win ratio would converge to 0.5 as both the players are improving at the same pace.

*Figure 6 : Q-agent vs Q-agent*

1. **Q3 : Q-learning Agent vs Simple Agent**

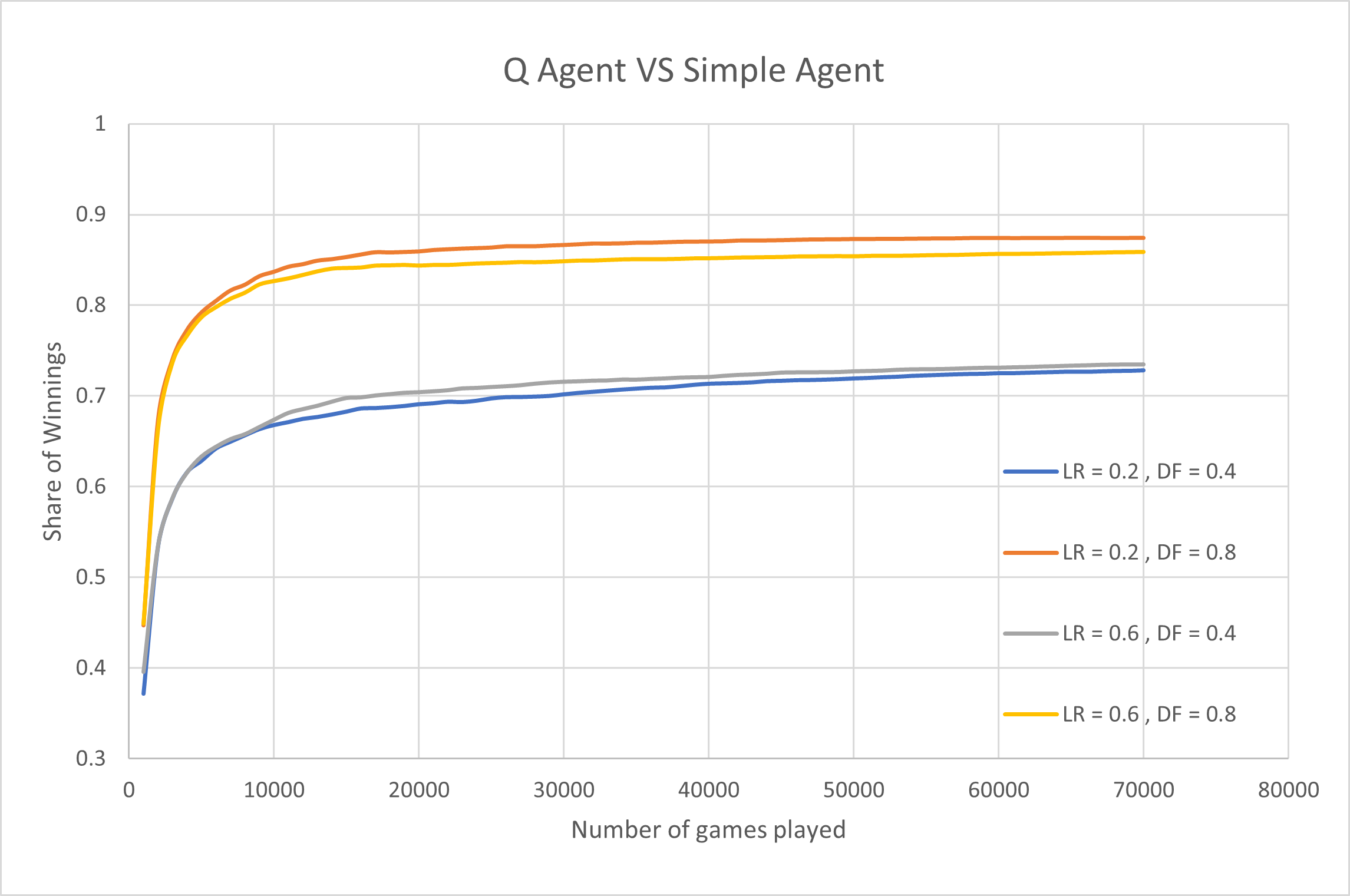
The test was made between a Q-learning agent with no previous memory and a Simple agent. The policy learned was saved as Q3.

The Q-learning agent does not take too long to understand the behavior of the simple agent and converges quickly. The figure clearly shows that a learning rate of 0.2 and discount factor of 0.8 showed best results.

Possible reasons:

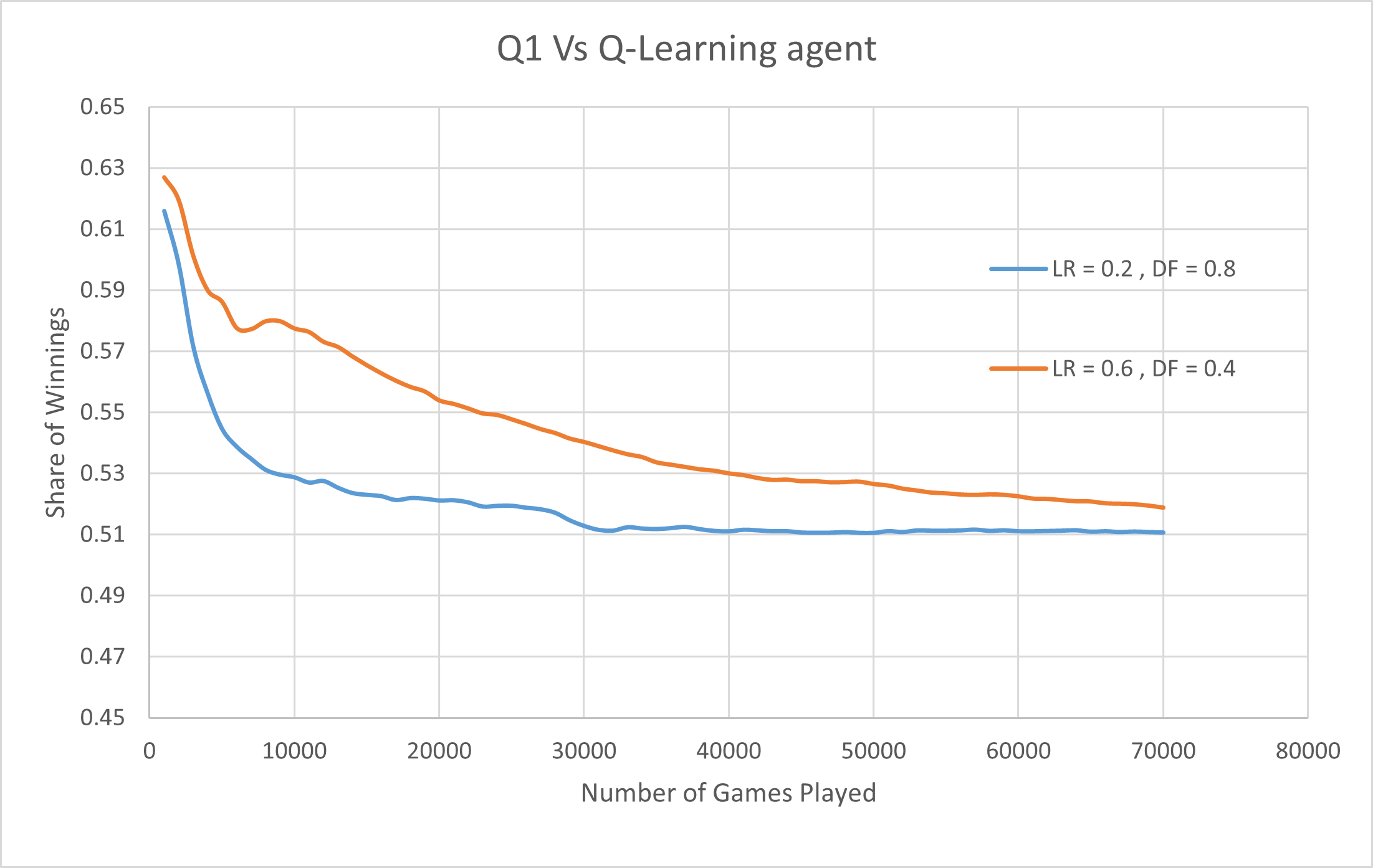
(i) This can be due to faults in our reward policy.

(ii)The Q-learning agent should logically converge to 1, but it's not because of our choice of exploration strategy. According to Decay Epsilon greedy method, the agent would still explore (with very small probability) in further rounds even though it had learned the best move. The results can be improved by changing the exploration strategy. (Keeping an exploration phase at the beginning and exploiting completely in later stages).//

*Figure 6 : Q-agent vs simple agent*

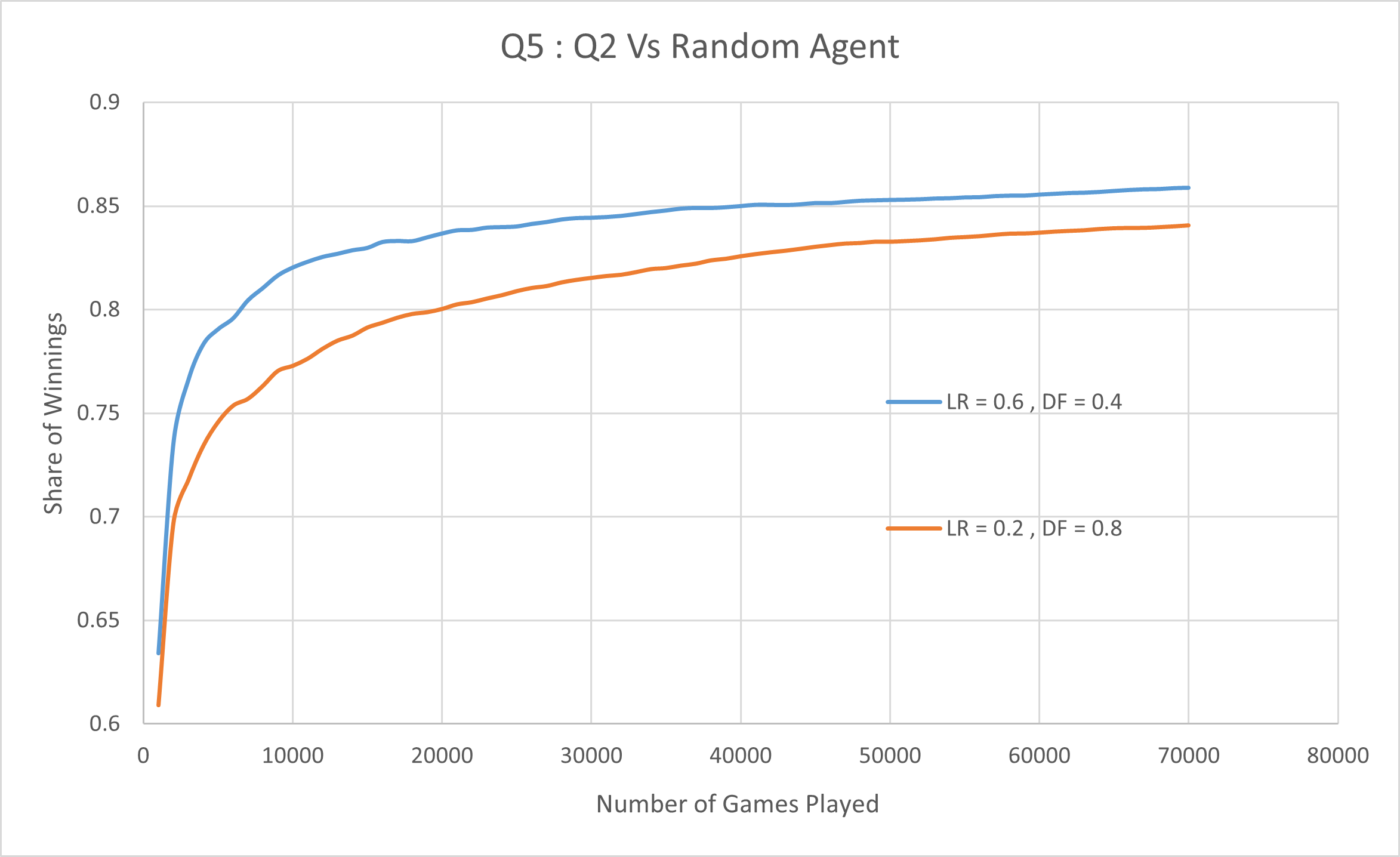
1. **Q4 : Q1 vs Q-Learning Agent**

In this experiment the Q1 agent is being trained against a Q-agent having no previous memory. Note that, Q1 is the agent formed after a Q-learning agent with no previous memory was trained against a random agent. We can observe in Fig 7 that the performance of the new Q-agent is improving with time, as it gets trained against Q1. Q4 is the result of letting Q1 extend its experience against a fresh Q-learning agent. The graph should converge at 0.5 SOW. It’s also observed that a higher learning rate favors the training process.

F*igure 7: Q1 vs Q-agent*

1. **Q5 : Q2 vs Random Agent**

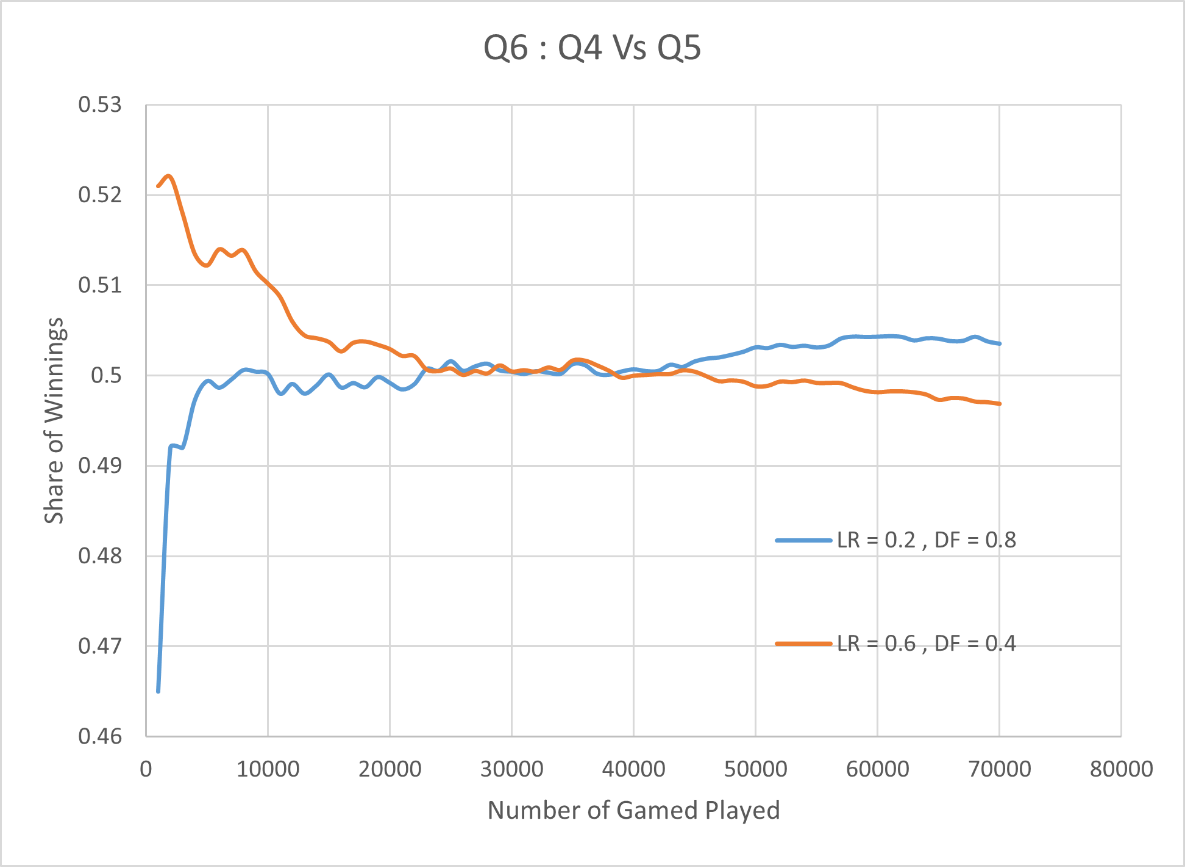
In this experiment , a previously trained Q-agent is trained against a random agent. It’s worth observing that the performance of the Q-agent has significantly improved from that of Q1. The Q-agent performs best for lower higher learning rates.



*Figure 8 : Q2 vs Random Agent*

1. **Q6 : Q4 vs Q5**

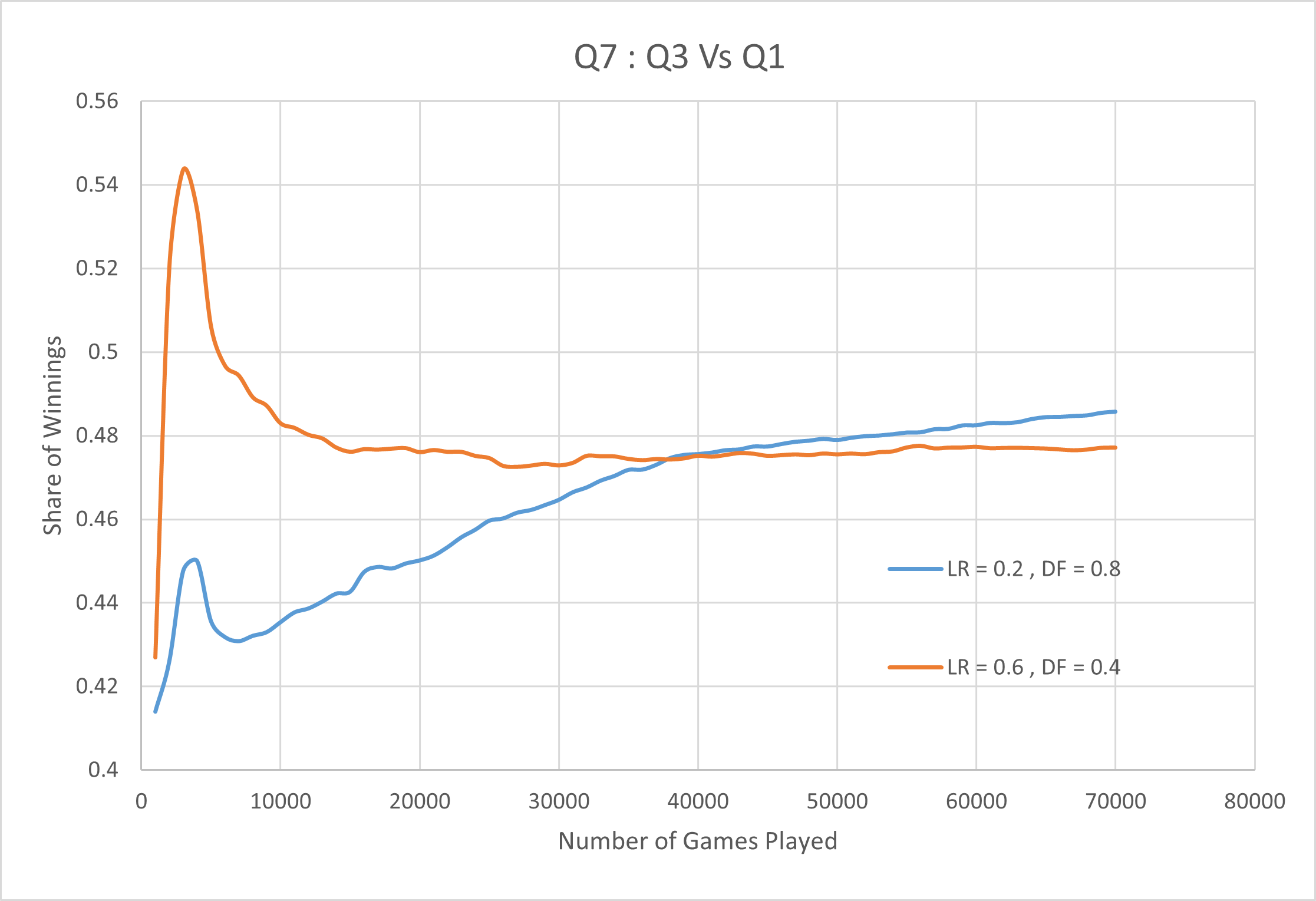
This test was made with two agents, the first agent being the Q-learning agent Q4 and the second agent being the Q-learning agent Q5.Both Q4 and Q5 have been trained against a Q-learning agent without experience and against the Random agent. The only difference is the order in which they have been trained against the two agents.In figure 14, it can be seen that there is not much difference between the two agents except a slight deviation in the beginning of the run.



*Figure 8 : Q4 vs Q5*

1. **Q7 : Q3 vs Q1**

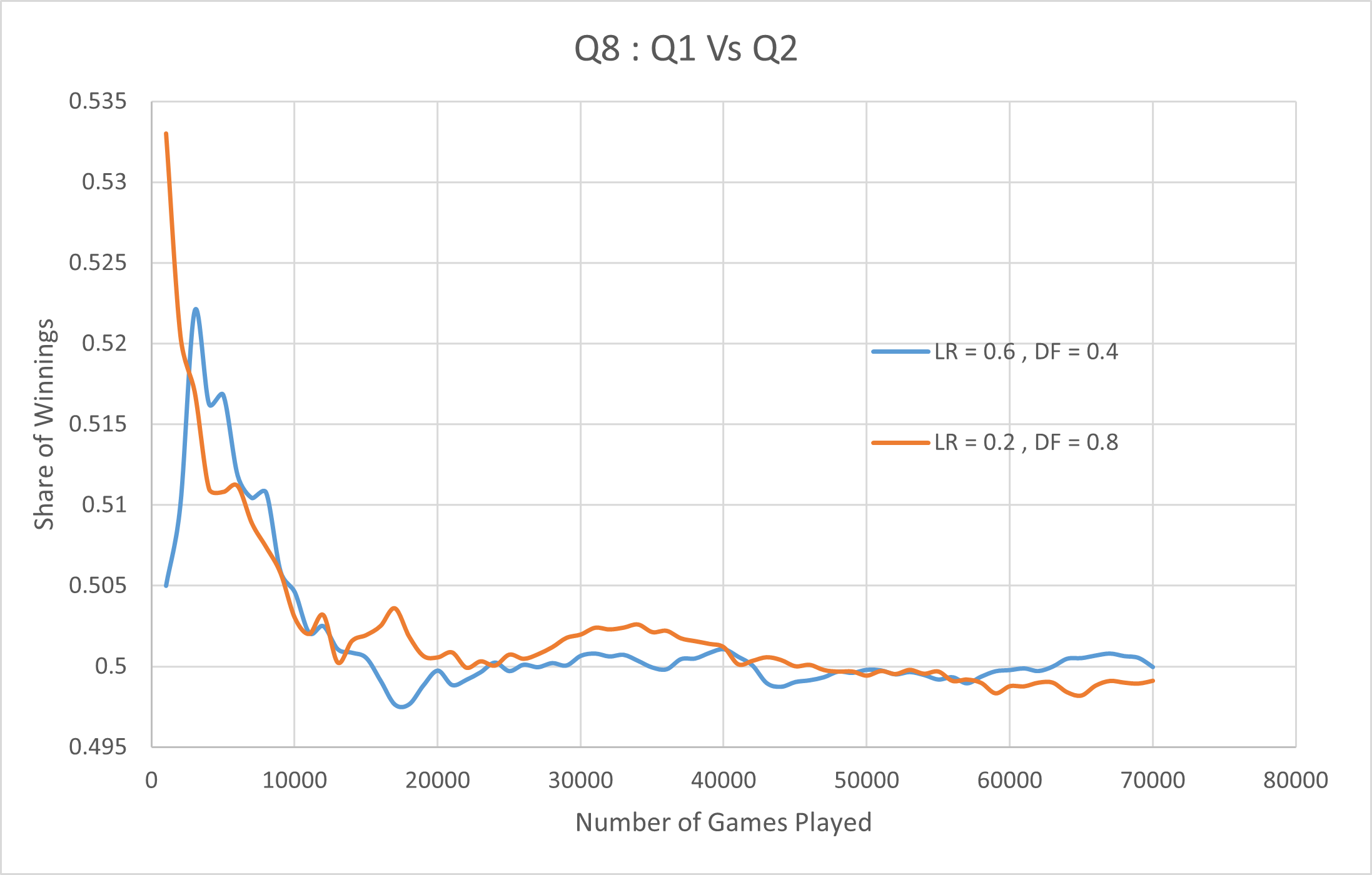
Both the agents here are Q-agents having previous memory. Q1 was previously played against a random agent and Q2 against a simple agent. We used the parameters that had the best performance for Q1 and Q3. We observe that both the agents are trained and as expected the SOW vs number of games converges to 0.5.



*Figure 9: Q3 vs Q1*

1. **Q8 : Q1 vs Q2**

The agents used in this experiment are both previously trained Q agents. Q1 has previously been trained against a Random Agent while Q2 has been trained against a Q-learning agent. We can observe that the SOW for Q1 converges to 0.5 which implies that both agents are becoming equally good at playing the game. The Q1 and Q2 parameters are chosen to be those which performed the best.



*Figure 10: Q1 vs Q2*

**6. Discussions**

This segment discusses the effect of the training parameters on the q-agent.

Learning Rate

A higher learning rate implies higher slope which is quite obvious as there is higher tendency to accept the new values. However , there is no direct correlation between the learning factor and the performance. According to Fig.6 (red and yellow lines) , we observe that a lower learning rate is more accurate in the long run. More data about Q4 would be beneficial as it is the agent formed between a previously trained q-agent and a new q-agent. This is so , since it would let us gauge whether accepting the results from the previously trained q-agent speeds up the training process or not.

Discount Factor

Looking at all the graphs, it would be quite reasonable to say that a higher value of discount factor results in better performance. Figure 6 is an example to prove the same. From theory, we know that discount factor is more important than learning rate as a training parameter. *At a low discount factor, the Q-learning agent is optimized for choosing a fast way to win before a more secure strategy. On the other hand, a high discount factor will partially prevent this behavior from occurring.*

**7. Problems Faced**

As students , we had limited knowledge on this project domain and so we had to follow the hit and trial method. This segment discusses all the problems we encountered and their solutions.

1. Choosing who takes the first action  
    Initially this didn't seem to be a decisive problem but later on getting the results of the q-table, we figured that many of the states were totally unexplored. Q agent always playing the first move would imply that the second move could never be played by the q-agent. This is quite a large problem as all the states having the number of lines drawn as odd would never be worked upon and q-table for those state action pairs would remain empty.

We solved this problem by randomly choosing an agent to play the first move.

1. Bonus moves when a player scores a box  
    As per the rules of the game , a player gets to play an additional move whenever it scores a box. Our initial code overlooked this. This rule was later implemented using do-while loops.
2. Exporting/Importing data from CSV to C++.  
   This problem was solved after understanding the concepts of file handling and csv files.
3. Lack of computational power and time constraints  
   We were able to perform only a certain number of games for training purposes as playing 1 lakh games took almost 3 hours of computing time..

**8. Learning Outcomes**

1. C++

This project helped us improve our coding skills and made us gain knowledge and insights about C++. We learned and implemented various data types and functions.

2. Excel/CSV handling

Through this project we learnt about the advantages of csv files. We used MS excel to convert the csv files to illustrative data which was used for commenting the training methods.

3. Problem Solving ability

This project demanded brainstorming sessions and good planning. Over the duration of the project, we gained more intuition behind the working of q-learning. Many a time, we spotted glitches in our code due to which the entire training process had to be redone. Even though we started off with a well planned structure , lack of experience was always a hindering factor.

4. Teamwork

This project definitely uplifted our teamwork spirit. Most of the tasks were done together by the team members. Time management and communication skills were also improved.

1. **Drawbacks**

1.1 Lack of computation power:

For many cases , we could not reach the converging values as computing time was very high. With higher computation power, better result accuracy could be expected.

1.2 Deviations from the expected results:

In few of the cases , the graphs plotted by us was different from what we would expect. Proper reasoning as to why such happened was discussed in the results section.

**10. Future Plans**

1. Deep Q-learning

We have plans to use a deep neural network instead of the hash map as the q-table. This would lower the memory requirements.

2. Implement a GUI and play with humans

Till now we have trained the q- agent by making it play against other agents. Next, we intend to design a GUI , so that the q- agent can train and play against human opponents.

3. Optimize out code

We will try to find out if any of the segments can be optimized for better performance.