Q-Learning for a Simple Board Game

OSKAR ARVIDSSON
and LINUS WALLGREN

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2010

Q-Learning for a Simple Board Game

O S K A R A R V I D S S O N
a n d L I N U S W A L L G R E N

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Johan Boye
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/arvidsson_oskar_OCH_wallgren_linus_K10047.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

This report applies the reinforcement learning algorithm Q-learning to the simple pen and pencil game dots & boxes, also known as La Pipopipette. The goal is to analyse how different parameters and opponents affect the performance of the Q-learning algorithm. Simple computer opponents play against the Q-learning algorithm and later Q-learning algorithms with different experience play against each other. Several different parameters are tried and the results are plotted on detailed graphs. The graphs show that the best results are gained against other Q-learning algorithms and a high discount factor. In the interval of 0.2 to 0.8, a discount factor of 0.8 showed the best results. In the same interval a learning rate of 0.2 showed the best long-term result while a higher learning rate, 0.8, gave the best short-term results. Especially when two Q-learning algorithms with different experiences played against each other, the best results were obtained with a learning rate of 0.8. The best results were shown when letting a Q-learning algorithm play against another Q-learning algorithm. A learning rate and discount factor of 0.8 showed the best results, however the data suggested even higher values of the discount factor might give even better results. In a dots & boxes game with a grid of three by three dots it requires about 1 million games to reach a good result.

Sammanfattning

Den här rapporten handlar om att applicera Q-learning, en belöningsbaserad inlärningsalgoritm, på spelet dots & boxes som ursprungligen kallades La Pipopipette. Målet är att analysera hur olika parametrar och motståndaren påverkar prestandan av Q-learning. Q-learningalgoritmen spelar både mot enkla datormotståndare och mot andra Q-learning algoritmer med andra erfarenheter. Flera parametrar testas och resultatet av alla tester presenteras i en mängd detaljerade grafer. Vad graferna visar är att de bästa resultaten fås när motståndaren också är en Q-learning algoritm samt när avdragsfaktorn är hög. När Q-learningalgoritmen spelar mot enkla datormotståndare visar det sig att inom intervallet 0.2 – 0.8 ger en learning rate på 0.2 samt en discount factor på 0.8 de bästa resultaten. Resultaten antydde även att ännu högre discount factors ger ännu bättre resultat. När två olika Q-learning algoritmer möter varandra blir resultatet som bäst när man även låter learning rate vara 0.8. Alla tester visar på att en miljon körningar är mer än tillräckligt för att göra algoritmen mycket bra när en spelplan på 3 gånger 3 prickar används.

Contents

1 Introduction

This report is part of the course DD143X, Degree Project in Computer Science, first level, which is given by the School of Computer Science and Communication at the Royal Institute of Technology.

The purpose of this project is to create a computer player utilising the Q-learning algorithm (Watkins, 1989) which will learn to play the traditional dots & boxes board game (Édouard Lucas, 1889). The computer player will then be evaluated with respect to efficiency and performance against several opponents with different strategies.

1.1 Background

Machine learning is a growing field with increasing importance. It is utilised when normal algorithms are too difficult and complex to develop and also because it has the ability to recognise complex pattern that otherwise might be overlooked. Machine learning methods can also be used when the task at hand is in a changing environment where adaptability is important in order to overcome new obstacles that might arise. (Mitchell, 2006)

An example of a field where machine learning has proven to be very effective is speech recognition. Humans find it easy to recognise and interpret speech, but find it hard to describe exactly how it is done. Much as a human learns how to understand a language, a computer can be taught the same, through use of machine learning methods. (Tebelskis, 1995)

Other examples of fields where machine learning is important are stock market analysis, medical diagnosis and face recognition. (Langley & Simon, 1995)

1.2 Purpose

Dots & boxes is a traditional board game which is normally played by two people at a time. Often one may not have anybody to play with, thus requiring a computer player acting opponent. To make the game enjoyable, the computer player should be appropriately skilled.

Using a simple reinforcement learning algorithm, called Q-learning, to create a computer player, the aim is to analyse the performance and efficiency of this player when faced against different opponents. The report looks at how these opponents affect and improve the rate of progress and end result of the Q-learning computer player.

1.3 Terminology

1.3.1 Dots & boxes

A traditional board game usually played with paper and pencil.

1.3.2 Reinforcement learning

Reinforcement learning is a subset of the machine learning techniques. These techniques learn through trial and error in search of an optimal solution.

1.3.3 Q-learning

Q-learning is a simple reinforcement learning algorithm.

1.3.4 Agent

An agent is a player in the game implementation, either a computer or an interface towards humans.

1.3.5 State

The state of the environment; the current game configuration.

1.3.6 Action

An action is a transition from one state to another, which is what a player does or can do in a specific state.

1.3.7 Feedback

An action generates feedback from the environment. The feedback can be either positive or negative. The feedback is used in the Q-learning algorithm for estimating how good an action is. The term reward is used for positive feedback and negative feedback is called punishment.

1.3.8 Learning rate

The learning rate is a parameter in the Q-learning algorithm, describing the relationship between old and new memories. A high value means that new memories take precedence over old memories and vice versa. A value of zero means that nothing will be learnt.

1.3.9 Discount factor

The discount factor is a parameter in the Q-learning algorithm, affecting the importance of future feedback. A high discount factor increases the importance of future feedback.

1.4 Theory

This section consists of more detailed descriptions on the various topics used and discussed throughout the report.

1.4.1 Dots & boxes

Dots & boxes is a simple game usually played with paper and pencil. It was first published by Édouard Lucas in 1889.

Typically the game board consists of nine squares, each with four dots on each side as shown in figure 1a. However game boards of arbitrary size can be formed. The game is mostly played by two players but there is no upper limit on the amount of players that can participate.

Between two adjacent dots a player can draw either a vertical or horizontal line if there isn't a line between them already. Each time four lines form a small box, a point is rewarded to the player who draw the last line in that box. When a player has finished a box, he must draw another line. The player with the most points, when all boxes in the grid have been completed, wins. If the game ends in a tie, the player who drew the first line loses.

A possible scenario of the first four turns in an instance of dots & boxes is given in figure 1 and described below.

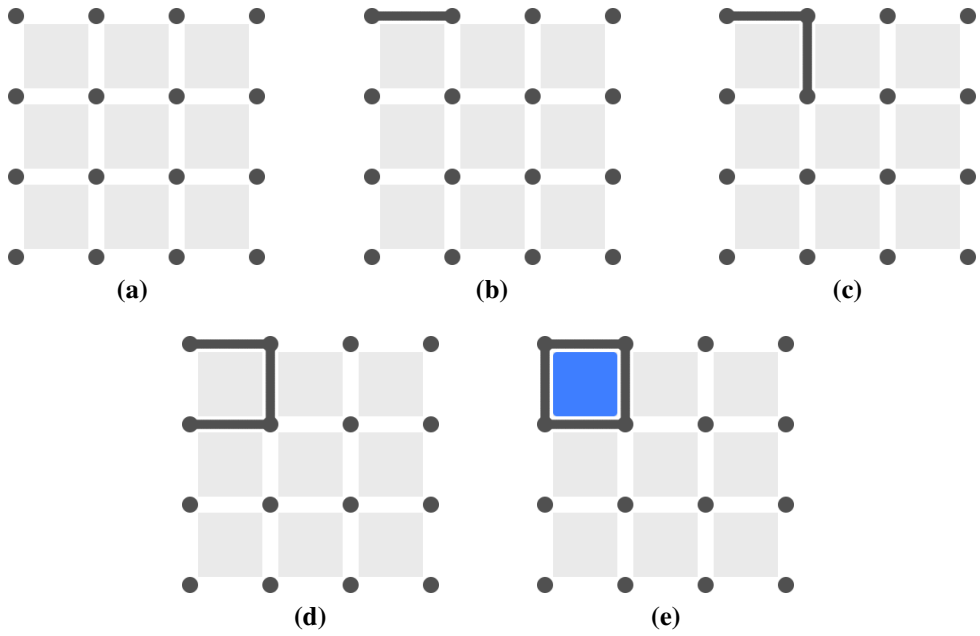

Figure 1: *A scenario of game play.*

1a) The game has not started and no lines have been drawn yet.

1b) The first player chooses to put a line in between two dots in the upper left corner of the grid.

1c) The second player chooses to add a line to the upper left box.

1d) The first player adds a line to the upper left box.

1e) The second player finishes the upper left box and receives one point.

Obviously, the first player should not have added the third line to the upper left box, as this practically gave the second player a point.

1.4.2 Q-learning

Q-learning is a reinforcement learning algorithm. The main advantage of this algorithm is that it is simple and thus easy to implement; the actual algorithm consists of only one line of code. Chris Watkins developed Q-learning back in 1989 when he combined several previous fields into what today is known as reinforcement learning. (Sutton & Barto, 1998; Watkins, 1989)

To understand Q-learning it is crucial to understand how a problem is represented. It is possible to divide almost all problems into several situations, so called states. For example in the dots & boxes game there is a state for each possible grid setup, meaning each different set of lines make up a different state. In each state a couple of actions can be taken, an action corresponds to what moves are legal in the game.

The simplest form of Q-learning stores a value for each state-action pair in a matrix or table. The fact that it requires one value for every state-action pair is one of the drawbacks of Q-learning; it requires a huge amount of memory. A possible solution to this problem is to use an artificial neural network (Hassoun, 1995), although that approach is not covered in this report.

In every state the algorithm visits it checks the best possible action it can take. It does this by first checking the Q-value of every state it has the possibility to get to in one step. It then takes the maximum of these future values and incorporates them into the current Q-value. When feedback is given, the only value that is updated is the Q-value corresponding to the state-action pair that gave the feedback in question. The algorithm for updating Q-values is shown in equation 1, where s_t and a_t corresponds to the state and action at a given time (Sutton & Barto, 1998).

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old Q-value}} + \underbrace{\alpha}_{\text{learning rate}} \times [\overbrace{\underbrace{r_{t+1}}_{\text{feedback}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1},a)}_{\text{max future Q-value}}}^{\text{expected discounted feedback}} - \overbrace{Q(s_t,a_t)}^{\text{old Q-value}}] \quad (1)$$

The fact that only one value is updated when feedback is given, gives Q-learning an interesting property. It takes a while for the feedback to propagate backwards through the matrix. The next time a certain state-action pair is evaluated, the value will be updated with regards to the states it can lead to directly. What this means is that in order for feedback to propagate backwards to the beginning, the same path need to be taken the same number of times the paths are long, each new iteration on the path propagating the effects backwards one step.

If the algorithm always took the best path it knew of it is very likely it will end up in a local maximum where it takes the same path all the time. Even though the path it takes is the best it knows of that does not mean there is no better path. To counter this it is necessary to let the algorithm sometimes take a new path and not the best one, in order for it to hopefully stumble upon a better solution.

If there is one, and only one, stable solution, that is which action is the best in each state, the Q-learning algorithm will converge towards that solution. Because of the back-propagation property of Q-learning, this however requires a large amount of visits to every possible state-action pair. (Watkins & Dayan, 1992)

There are two parameters to the algorithm, the learning rate α and the discount factor γ. These both affect the behavior of the algorithm in different ways.

The learning rate decides how much future actions should be taken into regard. A learning rate of zero would make the agent not learn anything new and a learning rate of one would mean that only the most recent information is considered.

The discount factor determines how much future feedback is taken into account by the algorithm. If the discount factor is zero, no regard is taken to what happens in the future. Conversely, when the discount factor approaches one a long term high reward will be prioritized.

2 Method

This section is about how the results achieved in this report were collected.

2.1 Agents

A game instance is played by two agents. The agents available are specified in this section.

2.1.1 Q-learning agent

The Q-learning agent uses the Q-learning algorithm to choose an action for a given state.

2.1.2 Random agent

The purpose of the random agent is to be unpredictable. It does always perform a random action upon any given state. This agent is used to train the Q-learning agent in as many states as possible, that is generate lots of training data.

2.1.3 Graphical agent

In order to interact with and play the game a graphical agent is used. It provides a graphical user interface for a human player to utilise. This gives an opportunity for the user to explore what actions the Q-learning agent chooses as well as the opportunity to test the Q-learning agent's performance against a human opponent.

The graphical agent provides a playing field window on which it displays the current state of the board. It does also track the points of both players.

2.1.4 Simple agent

The simple agent does always make the same move, independent of the opponent. It always chooses the minimum numbered line possible, according to figure 2.

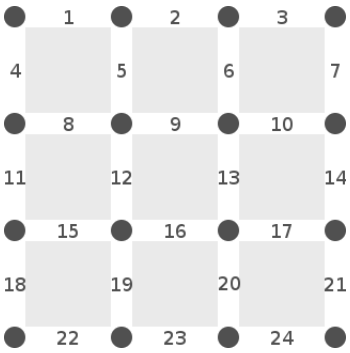

Figure 2: *How lines are numbered in the simple agent.*

The purpose of this agent is to give the Q-learning algorithm an easy and predictable opponent to play against, since it always does the same thing in the same position.

2.2 Problems

In this section, problems encountered during the writing of this report are discussed.

2.2.1 State space size

The states are defined as game configurations, where each line is unique and can be either drawn or not drawn. In a normal size dots & boxes game, the grid consists of 16 dots. This means a grid can contain up to 24 lines. The number of states is thus 2^{24} (about 16 million).

An action is defined as drawing a line. Thus there are 24 different actions that may be chosen. Not all lines may be drawn in each state however, as some lines already have been drawn. The mean number of lines possible to draw in a state is 12. Hence the number of state action pairs are $2^{24} * 12$ (about 200 million). In optimal conditions the number of turns needed to

walk through all these state-action pairs is therefore about 200 million. To compensate for the back-propagation property of the Q-learning algorithm, this process would have to be iterated a number of times. In total this would require a huge amount of games to be run. Therefore, a decision was made to decrease the size of the grid to 9 dots and 4 boxes. This greatly decreases the number of state-action pairs. The number of state-action pairs in a grid with 9 dots is only $2^{12} * 6$ (25 thousand), thus allowing more tests to be run in the time frame of this project.

2.2.2 Feedback policy

Two different ways of giving feedback to the Q-learning algorithm have been evaluated. Either positive or negative feedback is given only when the game ends or both when the game ends and when a box is completed and a point is awarded to the agent.

If rewards are given both when the game is won and when the agent finishes a box it must be emphasized that it is more important to win than to finish boxes. Therefore the reward for winning should be much greater than that of gaining a point by finishing a box. Likewise the punishment for losing a game needs to be more severe than that of losing a point.

To give a reward for a finished box is really only a primitive optimization technique since it speeds up back-propagation. However it assumes there is no strategy where losing boxes is good. This is not true, as shown in figure 3 and 4. In figure 4, the solid coloured player chooses to sacrifice a box to win the game, while in figure 3 the solid coloured player loses because of greediness.

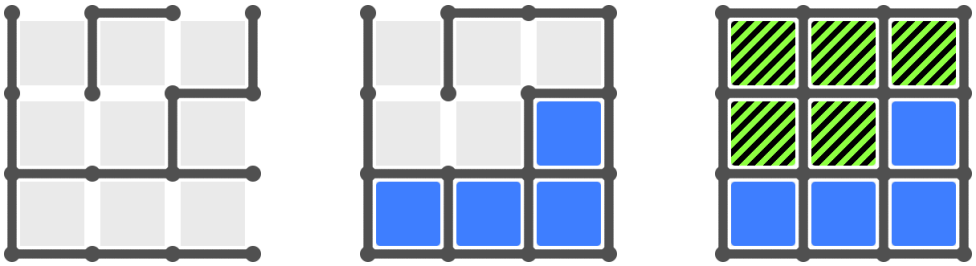

Figure 3: *Solid coloured player lose because all boxes possible are taken.*

In case a bigger grid had been chosen, it would have been a good compromise to speed up learning against not learning the most advanced strategies. Instead the decision was to make the grid smaller which leaves more time to teach the agent properly. Because of this, positive and negative feedback is only given when the game is either won or lost. What this in turn means is that in case the Q-learning agent cannot win it will give up and not try to take any more boxes. It has been seen as acceptable for the agent to have this behavior as the study will only be about the winning ratio for an agent.

In case the game ends in a draw, the rules of the game states that the player that did not start wins, as mentioned previously. This might pose a problem because the states do not include information about the starting order of the agents. This means that there might be state-action

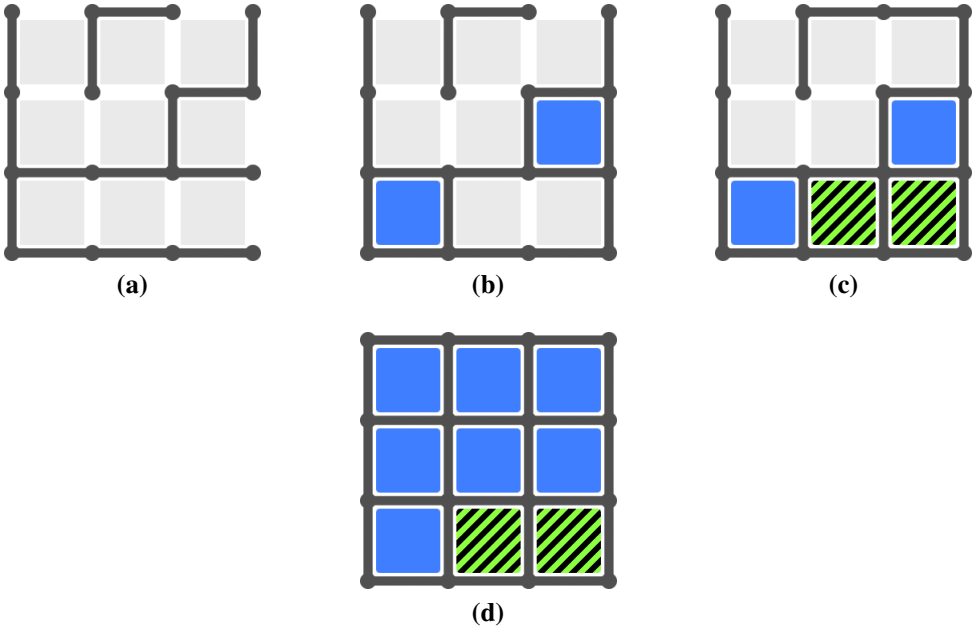

Figure 4: *Solid coloured player wins because two boxes are given away to stripe coloured player.*

pairs that will give a reward under certain circumstances but a punishment in other situations. We have chosen to ignore these special cases and trust the Q-learning algorithm to cope with them. This is an area where the implementation can be improved upon in the future.

2.2.3 Exploration behaviour

A problem with the Q-learning algorithm is how exploration of new paths is done. There are a couple of ways to execute this task. The simplest is sometimes to simply use a random action and hope it provides a better solution. Another way is to have a phase of exploration in the beginning of the agent's lifetime where all possible state-action pairs are explored.

The agent used in the implementation uses an exploration phase. This means that after the exploration phase, no further exploration is done. This could pose a problem since an agent trained against one opponent might not learn the most efficient behavior against another opponent because there is no real exploration being done. We have chosen to ignore this risk, trusting that the Q-learning algorithm will cope with it.

During the exploration phase each state-action pair is tested until it has a decent Q-value. Because of back-propagation this means that a state-action pair will be tested multiple times. This ensures that a reliable result is achieved.

2.3 Implementation

In a summary, the implementation is constituted of a bootstrapper, a game engine, a dots & boxes game representation and a number of agents. Additionally a table for storing and accessing the current Q-values is needed.

The main programming language used for this implementation is Java, however C is used to increase the performance of the state matrix.

The full implementation is available for download at http://www.csc.kth.se/~oskarar/dotsnboxes.tar.gz.

2.3.1 Bootstrapper

This component is responsible for parsing arguments from the user and then creates the agents and games requested. The bootstrapper then runs the games and waits for them to finish. When all games are finished, some cleanup is done, after which the application quits.

2.3.2 Game engine

The game engine begins by randomly picking the player to start. A game is run through repeatedly letting the agents take turn in a dots & boxes game simulation. Each turn the game engine requests an action from the active agent and the agent in question responds with the action of choice. The action is performed on the current dots & boxes game representation instance. Feedback is given right before the agent's next turn. Feedback is always zero whether the agent completed a box or not. When all lines in the grid have been set the round is considered completed. The winning agent then receives a reward of 10 points and the losing agent a punishment of -10. This process is repeated until the number of rounds requested by the user has been completed. A sketch of the process is shown in listing 1.

The current game state and score board is propagated to participating agents through the use of the observer pattern. Basically, the engine acts as an observable object and the agents as observers. The engine emits a signal to its observers when the game configuration changes.The signal can then be captured and interpreted by the observers. For example, this is utilised by the graphical agent to maintain the current scoreboard and graphical grid representation.

2.3.3 Dots & boxes game representation

The dots & boxes game representation is a component responsible for keeping track of the current state of the game. Basically this involves storing the lines of the grid that have been set, calculating the number of boxes created when a bar is set, and calculating a unique number for the current game configuration.

```
1   Agent current = firstAgent();
2
3   while not gameFinished(); do
4     if not isFirstGame(); do
5       current.giveFeedback(currentState(), 0);
6     end
7
8     boxesCreated = updateGrid(current.getAction(currentState()));
9
10    if boxesCreated == 0; do
11      current = nextAgent();
12    end
13  end
14
15  winner().giveFeedback(currentState, 10);
16  loser().giveFeedback(currentState, -10);
```

Listing 1: *Sketch over how game simulation is done.*

2.3.4 Q-learning agent

This agent consults a state matrix, a table holding Q-values, when an action is requested by the game engine. When feedback is given from the game engine, the state matrix is updated with a new Q-value for the current game configuration and the action chosen. A sketch over the procedure is shown in listing 2.

As described in 2.2.3, the exploration strategy explores all states in the beginning and when this is done, no further exploration is done.

```
1   function giveFeedback(currentState, feedback) :
2     Q(lastState, lastAction) :=
3       Q(lastState, lastAction)
4       + learningRate * (feedback
5         + discountFactor * maxQ(currentState)
6         - Q(lastState, lastAction));
7   end
```

Listing 2: *Sketch over how the Q-value for a state-action pair is updated.*

2.3.5 Graphical agent

This agent requests the user to choose which action should be used. It provides a graphical representation of the current game configuration and scoreboard. The state of the current game is updated through the use of the observer pattern.

2.3.6 State matrix

The state matrix is responsible for giving Q-learning agents the ability to access and modify Q-values. The current implementation uses a matrix for storing the Q-values.

The size of the matrix is calculated as shown in equation 2.

$$\text{number of states} \times max(\text{number of actions per state}) \tag{2}$$

As the number of actions per state is on average $\frac{1}{2} \times max(\text{number of actions per state})$, the matrix could be optimized to only use half that size, but this would also have required a more complex architecture for the state matrix. Even though halving the matrix saves space it might decrease performance due to increasing complexity.

The size of the matrix greatly increases with larger dots & boxes games, as described in 2.2.1. When creating games with 9 boxes, our Java implementation was not able to sufficiently handle the matrix. It was too slow and kept running out of heap space. Therefore, a C implementation was created to resolve these problems. The Java code uses Java Native Interface to utilise the C implementation.

2.4 Analysis

The results will be represented in graphs detailing how successful the Q-learning agent has been when playing against other agents. The success will be determined by the portion of games it has won. The tests will be made with different learning rate and discount factor parameters, in order to get a result of how important these values are for the performance of the Q-learning agent.

2.5 Tests

The tests consist of letting several agents play against each other in many different configurations. In figure 5 a name is given to each round. In the first column, the name corresponds to a Q-learning agent with the memory of the first player of the round with the name in question.

The opponents play 10 million games against each other and statistics are taken every thousand game. This gives the agents more than enough time to learn to play against their opponent and the sampling rate gives enough data to analyze and enables high detail results to be obtained.

For the agents without prior memory, different parameters for the algorithm are tried out. Because of limited computing power it is only possible to try a couple of different parameters. The values for both the learning rate and discount factor are 0.2, 0.4, 0.6 and 0.8. The parameters that show the best result are then used for 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 5: *Table showing the tests run*

3 Results

In this section results obtained from the tests executed during the writing of this report are presented.

3.1 Q1: Q-learning agent trained against a Random agent

This test was made with two agents, the first being a Q-learning agent without any previous memory and the second agent being an instance of the Random agent. The resulting Q-learning agent is called Q1. Ten million games were run and statistics collected every thousand game. The total win ratio for Q1 is shown in figure 8.

It is obvious from figure 6 that a learning rate of 0.2 is most successful while figure 7 shows that a higher discount factor gives better results.

3.2 Q2: Q-learning agent trained against another Q-learning agent

This test was made with two Q-learning agents without previous memory. Ten million games were run and statistics collected every thousand game. Q2 is the first of these two agents.

The results from this test, represented by figure 9, show that the two agent's win ratios are almost the same from the start to the end of the 10 million games. This is not surprising since the two agents have the same learning behaviour and almost equal experience. The cause of the small deviation the first few hundred thousand games is probably just chance.

3.3 Q3: Q-learning agent trained against a Simple agent

This test was made with two agents, the first being a Q-learning agent without any previous memories and the second agent being an instance of the Simple agent. Ten million games were run and statistics collected every thousand game. The resulting Q-learning agent is called Q3.

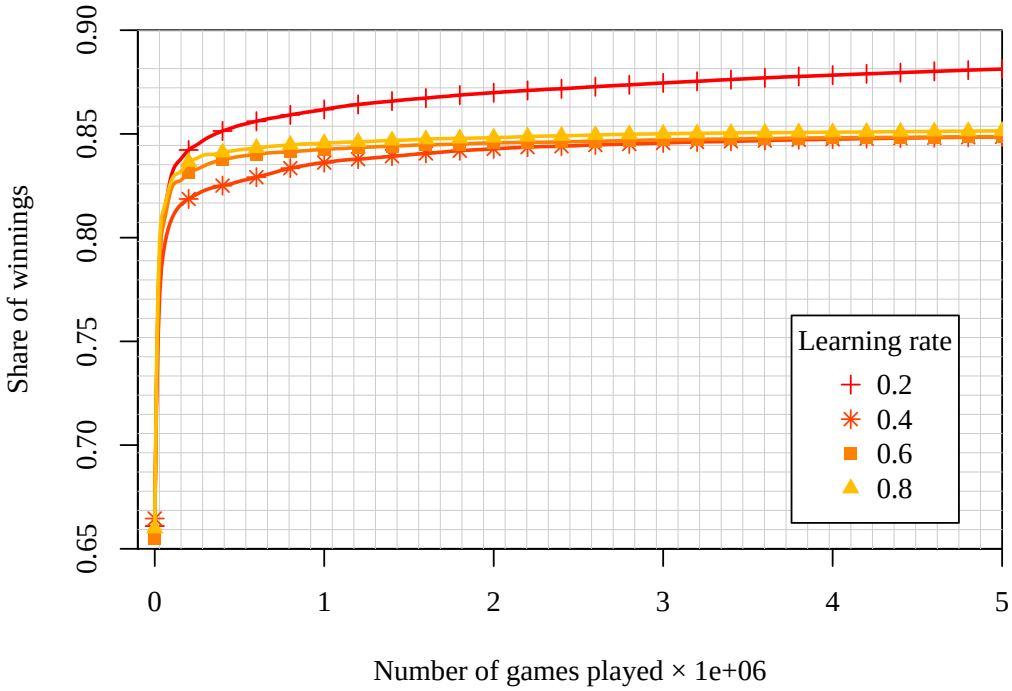

Figure 6: *Win ratio for Q1 with discount factor 0.2 over 5 million games.*

Not surprisingly it does not take long before the Q-learning agent understands how the Simple agent operates as seen in figure 10.

3.4 Q4: Q1 trained against a new Q-learning agent

This test was made with two agents, the first agent being the Q-learning agent Q1 and the second agent being a Q-learning agent without any previous memories. Ten million games were run and statistics collected every thousand game. Q4 is the result of letting Q1 extends its experience against a fresh Q-learning agent.

From figure 11 it is clear that the new Q-learning agent benefits from a high learning rate.

3.5 Q5: Q2 trained against a random agent

This test was made with two agents, the first agent being the Q-learning agent Q2 and the second agent being a Random agent. Ten million games were run and statistics collected every thousand game. Q5 is the result of letting Q2 extend its experience against the Random agent.

Results are shown in figure 12 and 13. In figure 12 it can be seen that Q2 has no problem against the Random agent and that lower learning rates are better.

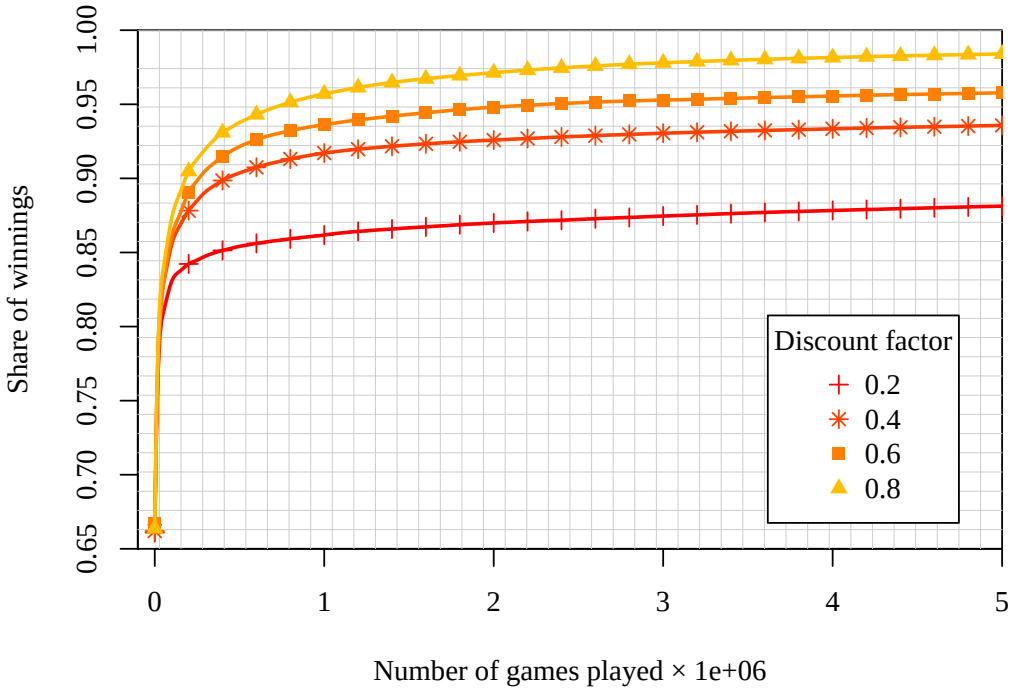

Figure 7: *Win ratio for Q1 with learning rate 0.2 over 5 million games.*

		Learning rate			
		0.2	0.4	0.6	0.8
Disc. fact.	0.2	0.888696	0.852646	0.850625	0.854005
	0.4	0.941035	0.938424	0.930991	0.918433
	0.6	0.965532	0.965026	0.962405	0.958045
	0.8	0.989537	0.987815	0.986991	0.979817

Figure 8: *Table of win ratio for Q1 over 10 million games.*

3.6 Q6: Q4 trained against 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. Ten million games were run and statistics collected every thousand game. Q6 is the result of letting Q4 extends its experience against Q5. Both Q4 and Q5 have been trained against a Q-learning agent without experience and against the Random agent, differentiated only by 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.

3.7 Q7: Q3 trained against Q1

This test was made with two agents, the first agent being the Q-learning agent Q3 and the second agent being the Q-learning agent Q1. Q3 has previously been trained against a Simple agent

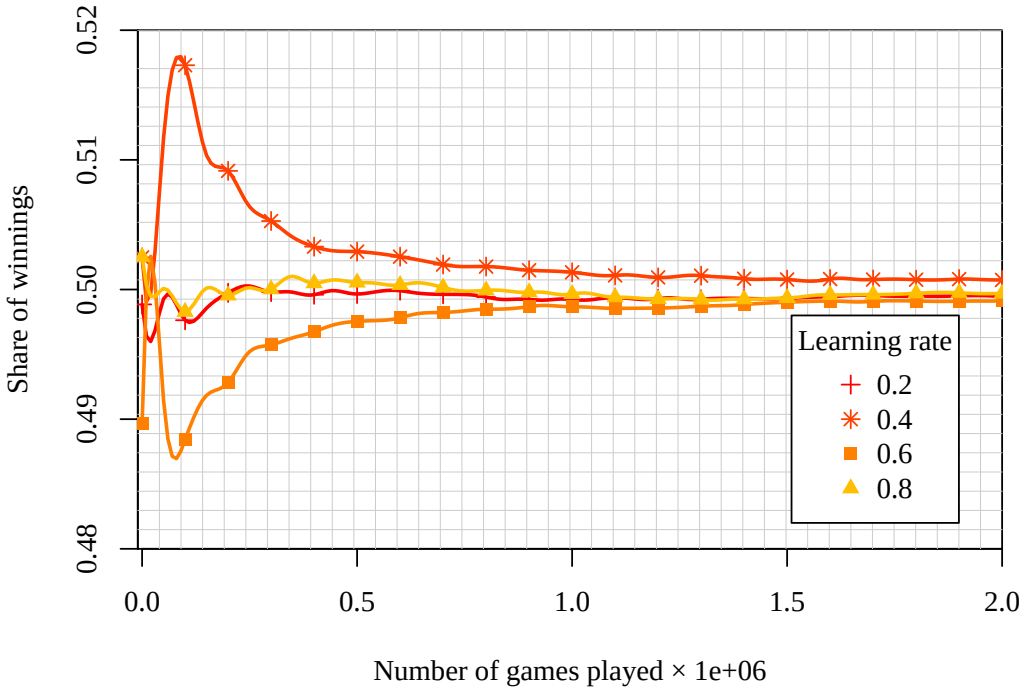

Figure 9: *Win ratio for Q2 when played against a Q-learning agent with discount factor 0.8 over 2 million games.*

while Q1 has been trained against a Random agent. Ten million games were run and statistics collected every thousand game. Q7 is the result of letting Q3 extends its experience against Q1.

Using the best parameters from both Q1 and Q3 it can be seen in figure 15 that Q1 is better in the beginning of the run.

3.8 Q8: Q1 trained against Q2

This test was made with two agents, the first agent being the Q-learning agent Q1 and the second agent being the Q-learning agent Q2. Q1 has previously been trained against a Random Agent while Q2 has been trained against a Q-learning agent. Ten million games were run and statistics collected every thousand game. Q8 is the result of letting Q1 extend its experience against Q2.

Using the best parameters from both Q1 and Q2 it can be seen in figure 16 that Q2 is better in the first 250 thousand games.

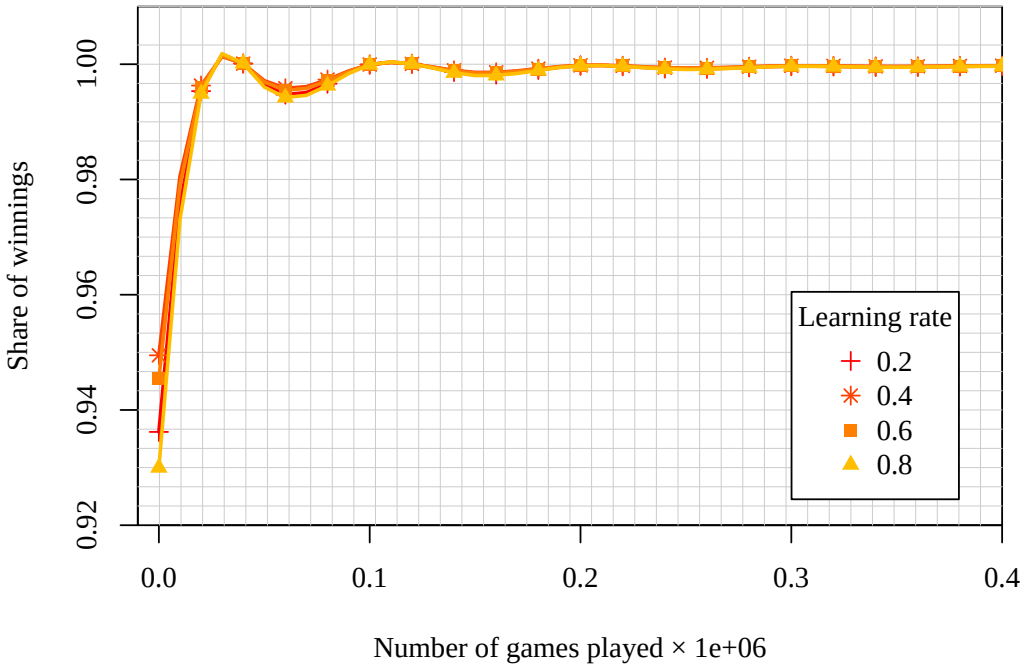

Figure 10: *Win ratio for Q3 with a learning rate of 0.8 over 400 thousand games.*

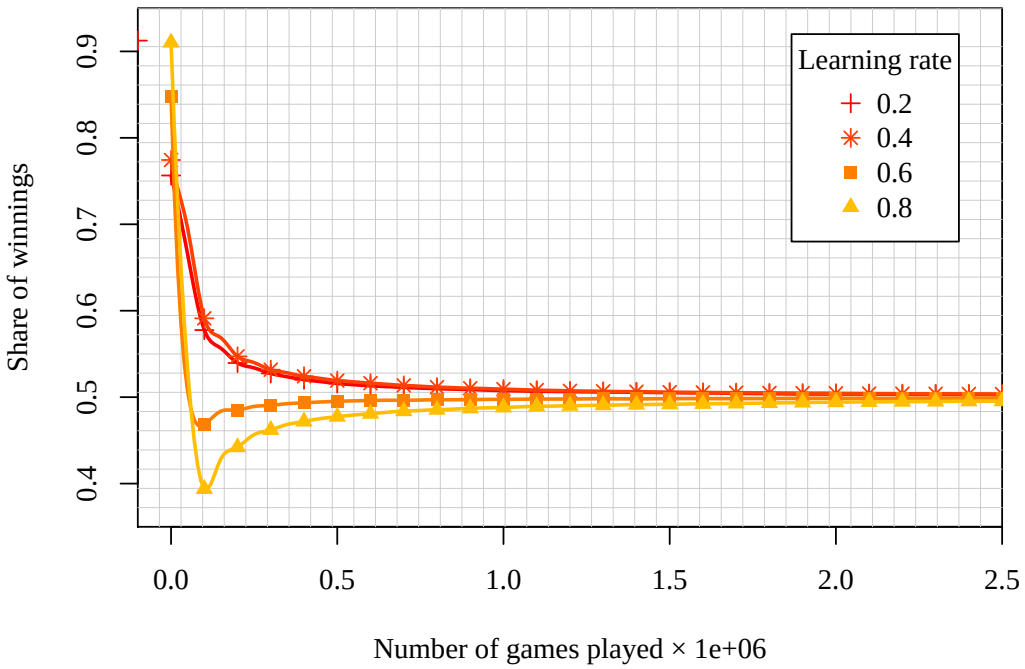

Figure 11: *Win ratio for Q4 with discount factor 0.2, based on Q1 with learning rate 0.2 and discount factor 0.4, over 2.5 million games.*

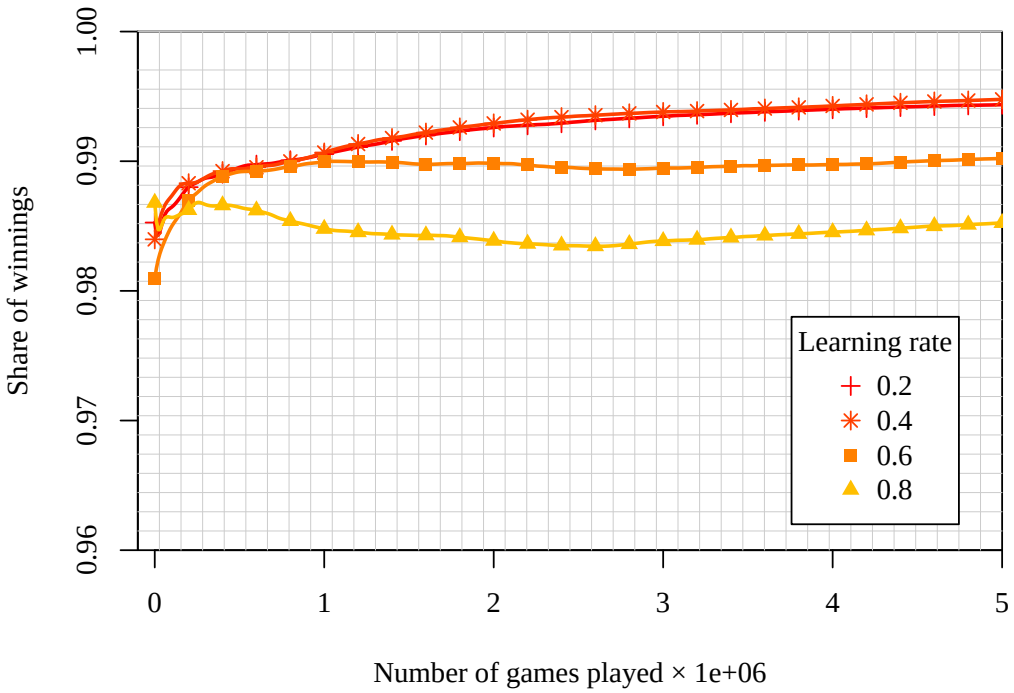

Figure 12: *Win ratio for Q5 with discount factor 0.8, based on Q2 with learning rate 0.2 and discount factor 0.8, over 5 million games.*

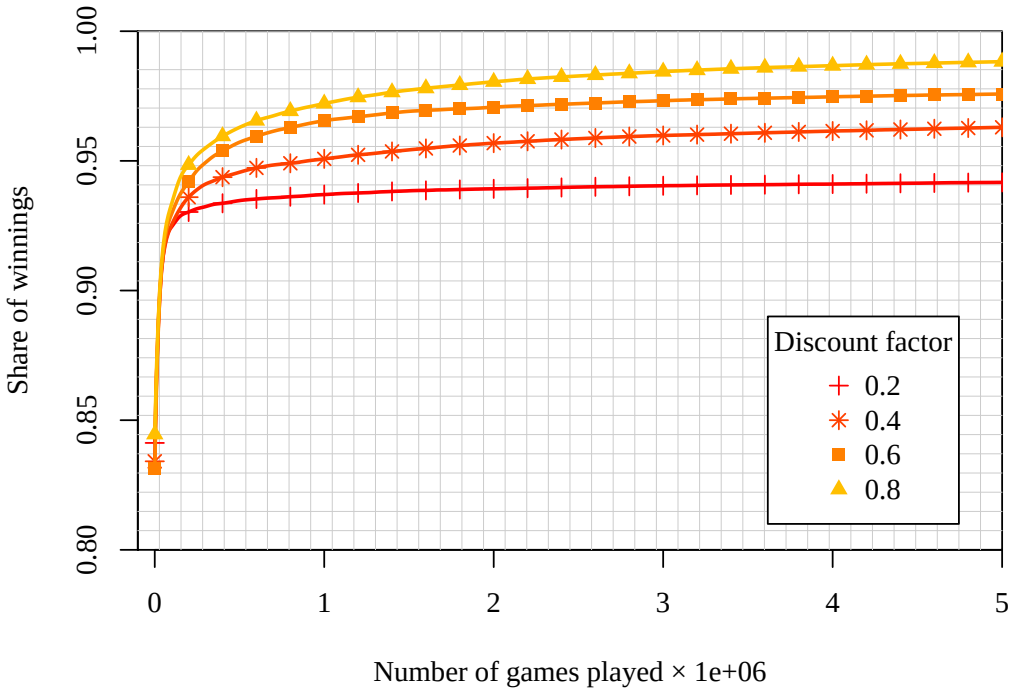

Figure 13: *Win ratio for Q5 with learning rate 0.2, based on Q2 with learning rate 0.6 and discount factor 0.8, over 5 million games.*

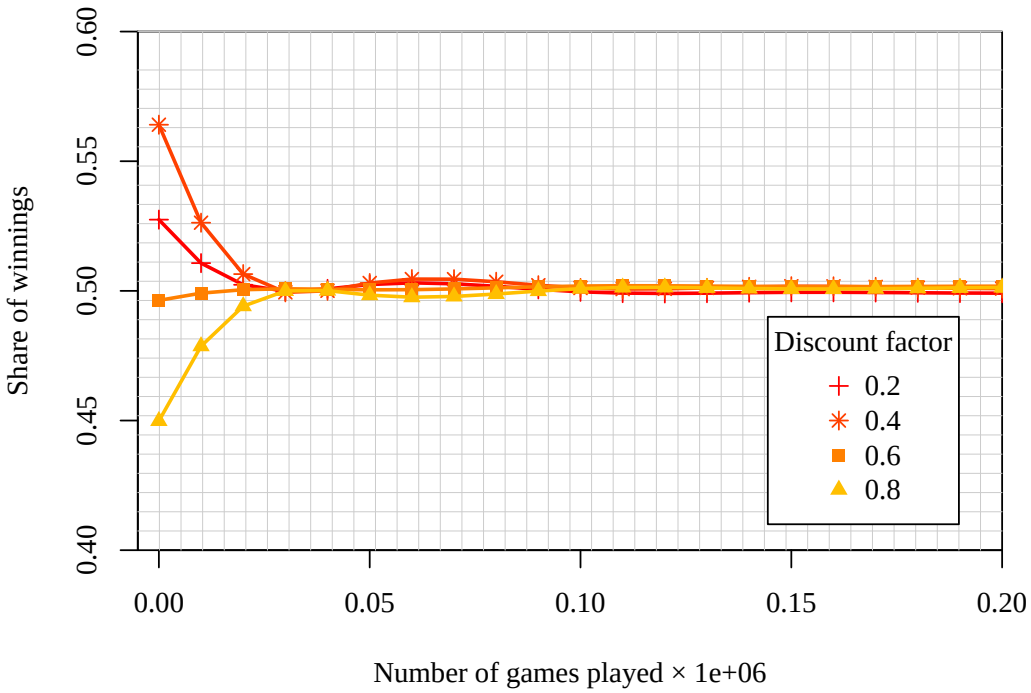

Figure 14: *Win ratio for Q6 with learning rate 0.4, based on Q4 with learning rate 0.6 and discount factor 0.8, over 200 thousand games.*

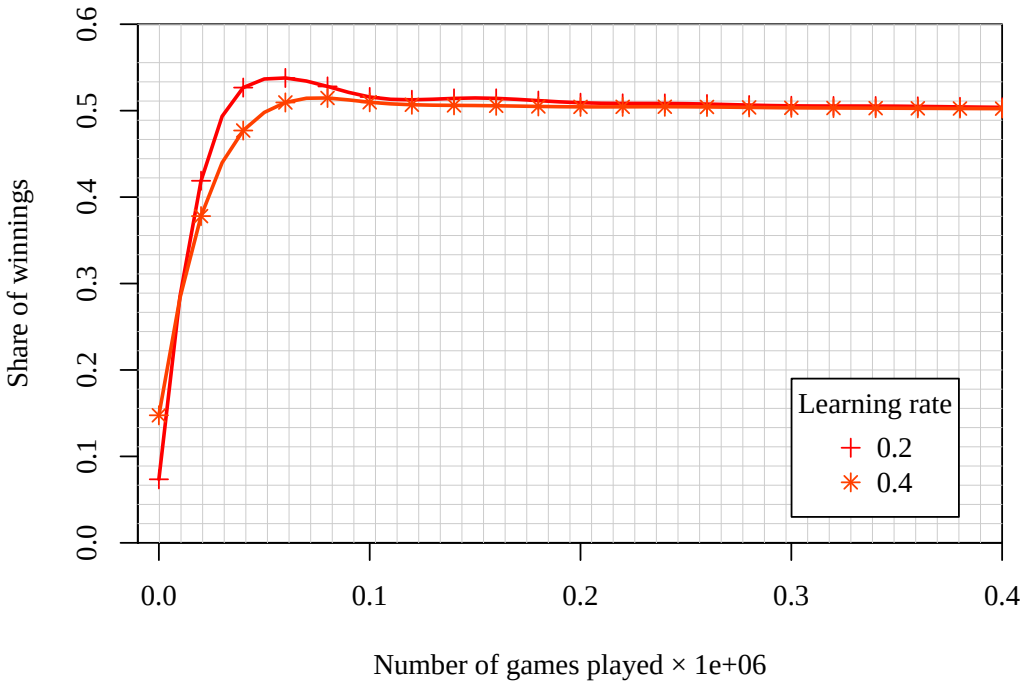

Figure 15: *Win ratio for Q7 with discount factor 0.8, based on Q1 with learning rate 0.2 and discount factor 0.8, over 400 thousand games.*

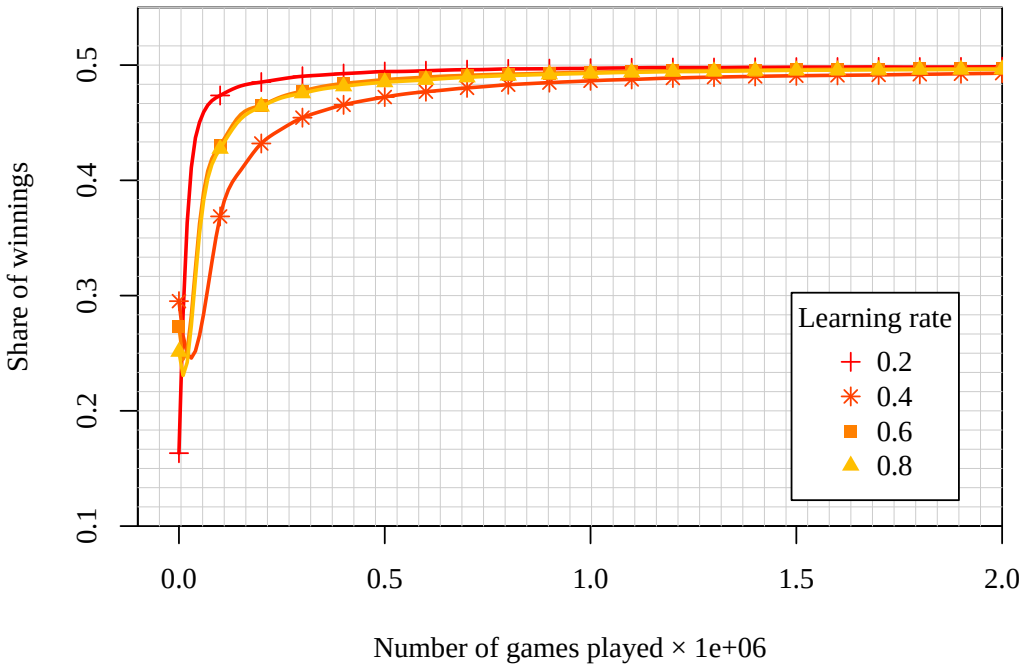

Figure 16: *Win ratio for Q8 with discount factor 0.2, based on Q1, over 2 million games.*

4 Discussion

In this section the results obtained from the tests are discussed and analysed.

4.1 Learning rate

The results from Q1 in figure 6 show that a higher learning rate gives a steeper slope in the beginning. In the end however, the lower learning rates shows a better result. That a high learning rate means a steeper learning curve in the beginning is not that surprising as that is the purpose of the parameter. It is however interesting to note that the end result often is better for lower learning rates. A probable reason behind this is that when using a higher learning rate old experience gets overwritten by new experience too soon and hinders learning, which means that the Q-learning agent doesn't have a good enough memory.

Figure 11 however, tells another story. When two Q-learning agents meet each other and one of the agents has more experience than the other it is more favourable to the inexperienced player to have a higher learning rate, especially in the beginning. The experience the experienced agent has tells him that certain strategies is good, however they might only work against the agent's previous opponent. This gives the new agent the opportunity to quickly learn and overcome the strategies the experienced agent has gained in previous runs.

4.2 Discount factor

The graphs in figures 7 and 13 indicate that a higher discount factor gives a better result. The graphs from test Q1 and especially figure 6 compared to figure 7 shows that the discount factor is more important than the learning rate when looking for a good result, both when looking for fast learning and high long term results.

To understand this, a deeper knowledge of how the discount factor affects the outcome of the Q-learning algorithm, is needed. Assume a learning rate approaching 1. Then the Q-learning algorithm can be simplified in the way shown in equation 3. The feedback policy used in this report only give feedback separated from 0 on transitions to final states. For non-final states this means that the expression can be further simplified, shown in the same equation.

$$\begin{aligned} Q(s_t,a_t) &= Q(s_t,a_t)+\alpha\times[r_{t+1}+\gamma\max Q(s_{t+1},a)-Q(s_t,a_t)] \\ &\approx Q(s_t,a_t)+r_{t+1}+\gamma\max Q(s_{t+1},a)-Q(s_t,a_t) \\ &= r_{t+1}+\gamma\max Q(s_{t+1},a) \\ &= \gamma\max Q(s_{t+1},a) \end{aligned} \tag{3}$$

Further, for final states, the Q value will have a fixed value of $\pm k$. For states adjacent to final states, this means that their Q-value will be $\pm\gamma\times k$. For states adjacent to these states, their

Q-values will be $\pm\gamma^2 \times k$, and so forth. This means that the Q-value of a state-action pair will be dependent on how many steps are needed to reach a final state. This argumentation could also be applied to configurations with other learning rates in the interval $(0,1)$.

However, this result is important because the number of steps needed to win will affect the Q-value in such a way that paths with fewer steps will weigh more than paths with more steps. As the connection is exponential, the lower the discount factor, the more will the Q-value be dependent on the size of the path.

With a discount factor of 0.2, the importance of the result is clear. In equation 4, the difference between winning a game in 4 and 6 steps with discount factor 0.2 and learning rate of 1 is shown.

$$\begin{aligned} Q(s_1,a_1) &= 0.2^4 \times k \\ Q(s_1,a_2) &= 0.2^6 \times k \end{aligned} \tag{4}$$

This means that the value from action a_1 will weigh 25 times more than the value from action a_2 when the Q-learning agent selects an action. The problem with this is that action a_2 may be a strategy for a guaranteed win while a_1 is not. In other words, with 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 behaviour from occurring.

4.3 Exploration phase

The figure 10 from test Q3 as well as figure 6 from test Q1 shows that it takes around 200–250 thousand runs in order for the Q-learning agent to complete its exploration phase.

A drawback with the exploration strategy chosen is that when the exploration phase is done it stops searching for new paths. So despite having an exploration strategy the Q-learning agent might still end up in a local maximum. Because of this it might be a good idea to try different exploration strategies in order to see how successful they are.

4.4 Performance

In figure 11 it can be seen that despite the fact that the agent already has run 10 million rounds against the Random agent it begins to lose against the new Q-learning agent. This might be a consequence of how the Random agent operates in comparison to the Q-learning agent. When the Random agent faces a situation where most actions lead to a loss and only a few to a win it is most likely the Random agent will lose. This will lead to the Q-learning agent thinking it is a good strategy to take that path. However later, when it meets a new Q-learning agent the old experienced agent takes the path that it thinks leads to a victory, but the new Q-learning

agent can quickly learn that it should take the actions that lead to a win, which gives the new Q-learning agent the upper hand.

The new Q-learning agent can even beat the experience agent before it has come out of its learning phase. A possible reason behind this is that the old Q-learning agent is sure its previous actions are good and will take the same paths until it no longer thinks they are.

It requires between 250k and 500k games in order for the Q-learning algorithm to reach an acceptable level of success by looking at figure 7, 11, 13 and especially figure 6. The Q-learning algorithm converges towards the optimal state matrix and as such, it takes longer and longer for improvements to show.

4.5 Conclusion

From all the tests run a few conclusions can be drawn. At first a higher discount factor always seems to give better results, independent of opponent, previous experience or learning rate. In fact it is probably a good idea to look at even higher discount factors when trying to implement a game like dots & boxes. Secondly, for a good end result a low learning rate is probably a good idea. A higher learning rate gives a much better result when training a new Q-learning agent against an already experienced agent.

It is interesting to note that even though Q1, with experience against the Random agent, shows promising results against Q3, with experience against the Simple agent, it is still not as good as Q2, which is trained against another Q-learning agent, as shown in figure 16. Likewise when looking at figure 11, the new Q-learning agent has no problem overcoming the already experienced Q1 for a while.

Considering the Q-learning algorithm begins to converge between 250k and 500k games it is not necessary to run more than about 1 million games in order to reach a good result without spending unnecessary computer resources.

The best combinations of parameters and opponents found during this research are a learning rate of 0.2 and a discount factor of 0.8 when training against another Q-learning agent with the same parameters as seen in figures 15, and 16.

References

Hassoun, Mohamad H. 1995. *Fundamentals of Artificial Neural Networks*. MIT Press.

Langley, Pat, & Simon, Herbert A. 1995. Applications of Machine Learning and Rule Induction. *Communications of the Association for Computing Macinery*.

Mitchell, Tom M. 2006 (July). *The Discipline of Machine Learning*. http://www.cs.cmu.edu/ tom/pubs/MachineLearning.pdf.

Sutton, Richard S., & Barto, Andrew G. 1998. *Reinforcement Learning: An introduction*. MIT PRess, Cambridge, MA.

Tebelskis, Joe. 1995 (May). *Speech Recognition using Neural Networks*. Ph.D. thesis, Carnegie Mellon University.

Watkins, Christopher John Cornish Hellaby. 1989. *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge.

Watkins, Christopher John Cornish Hellaby, & Dayan, Peter. 1992. Technical Note: Q-learning. *Machine Learning*.

Édouard Lucas. 1889. *La pipopipette*. Scanned version: http://edouardlucas.free.fr/pdf/oeuvres/Jeux%5F2.pdf.