
# Changing the Random Behavior of a Q-Learning Agent Over Time

PETER BOSTRÖM, ANNA MARIA MODÉE

Bachelor's Thesis in Computer Science

KTH CSC, Stockholm Sweden 2011, course DD143X 6hp

E-mail: pbos@kth.se – ammodee@kth.se

Supervisor: Johan Boye
Examiner: Mads Dam

# Abstract

Q-learning is a Reinforcement learning technique where an AI agent learns from experiences. This technique is commonly used together with the so-called $\epsilon$-greedy policy. The goal of this thesis was to determine how a few different random-behavior policies could affect learning rate of the Q-learning agent. To test this our agent played a reduced instance of the board game Blokus on a 5 by 5 board, primarily against a random-playing opponent. During these tests two different policies were tested that both started with the agent preferring random moves and gradually moving over to trusting its previous experiences. Both new policies introduced were able to converge to a close to 100% win rate. The study showed to be inconclusive however as the game instance was very limited and with our implementation the agent was able to beat it without any random behavior at all. Their similar performance to a relatively non-exploring strategy along with theoretical motivation for their use indicate that further research on them is motivated.

# Referat

Q-learning är en belöningsbaserad inlärningsteknik där en AI-agent lär sig genom erfarenheter. Den här tekniken förekommer vanligen ihop med en policy som kallas $\epsilon$-greedy. Målet med det här arbetet var att bestämma hur olika policies påverkade inlärningsgraden för den Q-learningbaserade agenten. För att testa agenten spelades en mindre instans av brädspelet Blokus på ett 5 gånger 5-bräde, först och främst mot en motspelare som lade sina brickor fullständigt slumpmässigt. Under testerna undersöktes två olika policies som båda startade med att agenten föredrog planlösa drag för att sedan gradvis gå över till att lita mer och mer på sina tidigare erfarenheter. Båda nya policies gjorde att agentens beteende konvergerade till nära 100% vinstfrekvens. Studierna visade sig dock vara ofullständiga på grund av att spelinstansen var väldigt begränsad och att agenten i vår implementation klarade av instansen utan något slumpat beteende över huvud taget. Resultatet gav dock att båda nya policies hade liknande prestanda på denna instans. Tillsammans med teoretiska argument för de nya policiernas användning indikerar detta på att fortsatt undersökning inom området är motiverad.

# Contents

# Chapter 1

# Introduction

Machine learning is a sub-domain of Artificial Intelligence. The machine learning discipline is focusing on designing algorithms that allows an *agent*, also called learner, to evolve behaviors based on the agent's experience. Hence the name machine learning. While collecting data, the agent tries to draw conclusions e.g. to recognize patterns and underlying characteristics for example. One of the goals in machine learning is for the agent to automatically see these patterns and act in an intelligent way based on this knowledge. A common problem when implementing an agent is the vast space of possible "valid" behaviors, or the set of all possible behaviors given all possible inputs.

There are many approaches on how to design these agents, *reinforcement learning* being one of them, and some are especially useful in particular fields. The applications for machine learning are many; computer vision, search engines, bioinformatics, stock market analysis, classifying DNA sequences, speech and handwriting recognition, game playing, and robot locomotion just to name a few[1].

## 1.1 Motivation

Q-learning is a well-known algorithm within *reinforcement learning*, first introduced in 1989 by Watkins[2] in his PhD.D thesis at Cambridge University. There are numerous studies on Q-learning being used for reinforcement learning of, among other applications, several different games; both multi-player, such as Reversi and tic-tac-toe, and single-player games, with one example being Tetris. For competitive multi-player games the goal is to beat the other player, e.g. to win. In single-player games the goal is to win against either an artificial opponent, your own skill, time or chance[3].

Q-learning's effectiveness on a game is restricted by both the number of states a game can be in, the so-called state space, and whether game states can be repeated or not. Consider chess without the threefold-repetition rule, or any game which is not guaranteed to end in a finite number of moves. In these cases the agent is not guaranteed to ever finish a game and will therefore not have learned anything.

An agent will also not have learned much until a "fair amount" of this state space has been explored, because it will not have learned the characteristics of the game. Thus using Q-learning in conjunction with a very complex game will be much less effective and require an insurmountable number of runs. As the algorithm also has to know and store values for actions taken from each state, storage space for the Q-learning algorithm also becomes an issue.

Agents using Q-learning are commonly learning through a so-called $\epsilon$-greedy policy. That means there is a chance of $\epsilon$ for the agent to take a random move. This policy is used to ensure that the agent performs the necessary exploration of the state space. With this common policy the chance for random behavior is a fixed value. This thesis was performed with the motivation by that exploration is assumed to be more important in the beginning but gradually less important when the agent has explored a significant amount of available states.

The study will be performed using a strategy board game called Blokus. Blokus is relatively easy to implement, easy to learn and yet hold the possibility for highly complex strategies. Though more complex game than both Reversi and tic-tac-toe the game finishes in a limited amount of moves. Both being more complex and having a limited amount of moves make the game ideal for this thesis' focus. It enables the possibility for the agent to develop advanced strategies and at the same time it is easy to implement different opponents. An important factor for choosing Blokus is that reductions can be made to the game while still having a similar game, should it prove necessary.

## 1.2 Initial Questions

This study was performed with the intention of determining how well a Q-learning agent will perform when using different policies for the agent's random behavior. As such there were questions regarding the feasibility of use and what impact of the random behavior of the agent will affect the algorithm's effectiveness.

- Is it feasible to use the algorithm on the game Blokus?
- How does the agent's opponent affect the agent's ability to learn?
- How does the adjustment of the chance of random behavior over time affect the agent's learning rate?

The answers to these questions will be based on several parameters. Is the state space small enough to be storable in memory? Will the study have to be done on a reduced version of the game? Can the random behavior be remodeled to improve the agent's learning rate? Which parameters are optional for the agent to learn Blokus as well as possible? These are the questions this report aims to answer.

## 1.3 Summary of Results

The results of this study are inconclusive. Even though most of the policies tested seems very rewarding, the game impact made a single misplaced tile play a very big role in the outcome of the game. This caused the $\epsilon$-greedy policy to to perform very bad when $\epsilon$ values were other than 0. When $\epsilon$ was 0, the agent performed equal to the best configurations of the other policies.

During testing, all opponents except the random agent were defeated almost instantly. For this reason, most of the presented test results are the performance of the Q-learning agent when playing against the random agent.

The policies other than $\epsilon$-greedy were able to reach a win rate of 100% relatively fast, usually within 50000 played games. For some configurations however, the win rate fluctuated over time. This was thought to be when the Q-learning agent found a special case where it would lose or play a draw and develop a new strategy which does not lead to this case.

## 1.4 Outline

The thesis report is divided into six chapters. After the introduction in the current chapter, Chapters 2 and 3 handles the necessary background and discuss previous work and research. Chapter 4 describes how the problems introduced in Section 1.2 were approached in detail and declares assumptions and constraints applied to the problem to set the scope of this thesis. The following chapter, Chapter 5, contain results of the study. The final chapter, Chapter 6, contains conclusions and thoughts.

For a complete list of acronyms and terms used in this report please see Appendix A. The complete set of graphs over the results can be found in Appendix B.

# Chapter 2

# Background

The purpose of this chapter is to provide insight and context to this project. First follows an explanation of the game used in the study, Blokus. Later, in Section 2.2 the concept of *machine learning* and some algorithms within its domain is described.

## 2.1 The Game

Blokus is a strategy game for two to four players. Each player have their own set of 21 tiles in one color. The game is played on a square board of 20 by 20 squares. The tiles are the free polyominoes (sequence $A000105$ in OEIS[4]) of order one to five, constructed from one to five square pieces, see Figure 2.1.

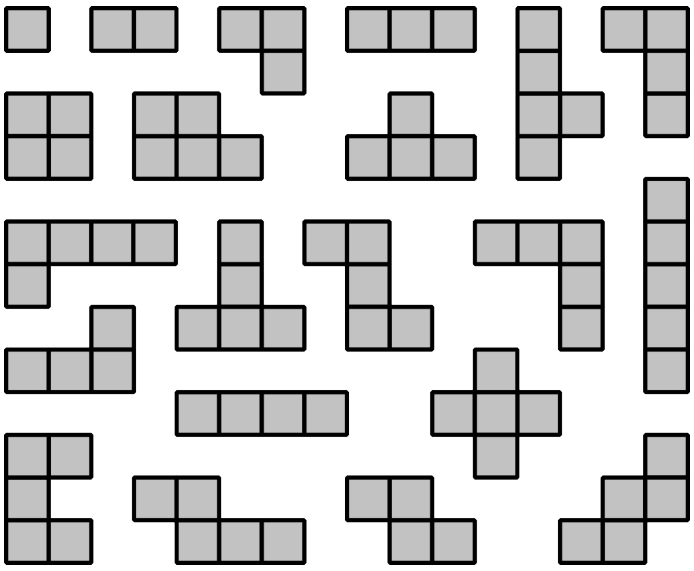

**Figure 2.1.** The 21 tiles used in Blokus

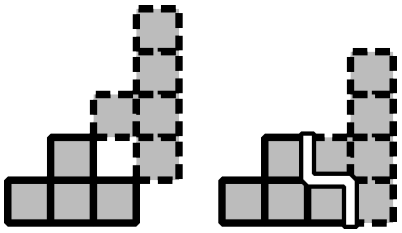

**Figure 2.2.** The correct and wrong way to place a new tile

Each player start by placing a tile so that one of its squares is placed in the player's starting corner. The players then take turns placing tiles. A new tile must be placed corner to corner a tile of the same color. A tile may not be placed edge-to-edge with an existing tile of the same color, see Figure 2.2.

The goal is to cover the board with as many squares as possible while preventing others from doing so. This means that a tile with more squares, though harder to place, is more rewarding. If a player is unable to place a tile, the turn goes on to the next player until no more tiles can be placed. At the end of the game, the player occupying the most squares win.

The starting player is presumed to be at at least a slight advantage, comparable to first-move advantage in chess[5].

## 2.2 Machine Learning

Learning is a very general concept of the ability to change and acquire new knowledge based on information. Machine learning concerns algorithms that allow computer programs to evolve similarly. A definition of machine learning is: "A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$."[1]. The focus of machine learning lies on experience and how the agent improves with this experience. There are many different approaches for how this is done, and this Section describes the arguably most common approaches.

### 2.2.1 Q-Learning

In reinforcement learning agents interact with an environment by performing *actions*, going from one *state* to another. Most methods are based on an estimation of the reward for actions, by guessing the value of the states the actions will lead to. When performing an action from a state the agent observes the reward given as well as the estimated value of newly available states.

The reinforcement learning technique Q-learning belongs to the class of learning methods called *temporal difference learning*[6]. Temporal-difference methods com-

pares the expected reward to the actual reward after each performed action and then improves the estimated state value based on what the agent has learned from the actual reward given. Thus the algorithm learns from experience.

Q-learning makes use of the so-called Q-function, also called action-value function[7], which takes a pair of a state $s$ and action $a$, and returns the estimated total future reward when performing action $a$ from state $s$. Each such possible pair is assigned a value, called the Q-value, and together they form the values of the Q-function.

The key is to compare the expected reward to the reward given by the environment when doing the action. After performing an action $a$ from the state $s$, the environment will move the agent to a new state $s'$ and give the agent a reward $r$. This enables a better estimation of $Q(s, a)$. The two estimations are:

$$\begin{aligned} Q(s,a)_1 &= Q(s,a) \\ Q(s,a)_2 &= r + \gamma max Q(s', a') \end{aligned} \tag{2.1}$$

The variable $\gamma$, called *discount factor*, affects the importance of future rewards. The discount factor is always given a value $0 \leq \gamma < 1$. A $\gamma$ value of 0 makes the agent consider only the latest reward given, while a value close to 1 makes the new experiences slowly add up over time, working for a long-term reward instead. The new and old Q-value can be weighted together for a new estimate. This forms the formula used in the update step in the actual Q-learning algorithm.

$$Q(s,a) \leftarrow Q(s,a) + \eta[r + \gamma max Q(s', a') - Q(s,a)] \tag{2.2}$$

The variable $\eta$, the *learning rate* of the algorithm, decide how much of $Q(s, a)_2$ in equation (2.1) will affect the new value. The range is $0 < \eta \leq 1$. If the value is 1, the equation would be equal to $Q(s, a)_2$ and only take the newest estimate into account. A value close to 0 makes the algorithm learn from new experiences very slowly as the new estimate is not taken into consideration as much.

The last important part of the Q-learning algorithm is the type of *policy* the agent will follow. A goal for the agent is to perform as good as possible, thus finding a policy which gives the greatest reward. There is always at least one such optimal policy[8]. The so called *greedy policy* is following the currently best path of actions. During learning however, for the values to converge into good estimates it is required that the agent visits all available states to gain information about them. This makes it necessary for the agent to explore other actions than the estimated optimal ones.

The *$\epsilon$-greedy policy* takes a random action with the probability of $\epsilon$ and takes the estimated optimal, or greedy, action with a probability of $1 - \epsilon$ each step.

The Q-learning algorithm can be written as follows:

**Algorithm 1** The Q-Learning Algorithm[6]

For all pairs ($s$,$a$) initialize $Q(s, a)$ arbitrarily
Initialize $s$
**while** not done **do**
  Choose $a$ based on $Q(s, a)$ using the $\epsilon$-greedy policy
  Take the action $a$ and observe the reward $r$ and next state $s'$
  *Update* $Q(s, a)$ using (2.2)
  Store $s'$ in $s$
**end while**

### 2.2.2 Artificial Neural Networks

Another approach within machine learning that is outside the scope of this study is Artificial Neural Networks (usually referred to as ANN:s). Inspired by biological neural networks, ANN:s consist of a group of connected *artificial neurons*. They are often used to find complex patterns and relationships[9]. ANN:s are capable of *learning*, and can be given a specific task to solve, a set of observations and a class of functions. The ANN will then solve it optimally when considering the *cost function* $C$.

To find the best solution, the cost function measures how far away a particular solution is from an optimal solution. By iterating over the solution space, the learning algorithm finds the solution with the smallest cost.

### 2.2.3 Different Kinds of Learning

In machine learning algorithms can be classified depending on how the learning is done. The categories are as follows[8]:

- *Supervised learning* is when the algorithm is given a set of examples to map correct respond depending on input. ANN:s are a perfect example of this.
- *Unsupervised learning* algorithms are not given the correct responses to input, but the task is to categorize similar inputs together and find what they have in common.
- *Reinforcement learning* is something in between supervised and unsupervised learning. The reward given after each action is a hint of what the correct response might be, the task is to find the way to maximize the cumulative reward[7]. The Q-learning algorithm used in this project is an example of a reinforcement-learning algorithm.
- *Evolutionary learning* is inspired by biological evolution where the most fit versions survive and form a new, better generation.

# Chapter 3

# Previous Research

Previous research has been done both on the importance of the different variables in the Q-learning algorithm[10], as well as the quality of the algorithm itself[11][12].

## 3.1 Learning Rate and Discount Factor

Independently of the policy being followed, the Q-learning algorithm is capable of estimating the Q-values correctly[6]. Even so, it is important to consider the learning rate and discount factor in implementation. Arvidsson and Wallgren[10] found out that for a simple two-player game with a small board, a learning rate and discount factor of 0.8 showed the best results. They claim that some tests indicated that even higher values for the discount factor might be better.

These tests however were all performed without giving the agent any intermediate rewards. Intermediate rewards make the agent prefer short-term rewards to begin with and gives the agent work toward an initially greedy strategy.

## 3.2 Strengths and Weaknesses in Q-Learning

The nature of Q-learning has been a popular topic of research, as have its strengths and weaknesses. Fairly recent research by Forsberg and Mattsson[12] states that Q-learning is unable to recognize patterns and can not see similarities in strategies to predict them.

It is possible to implement Q-learning agents using ANN:s, shown by Björkqvist and Wester in their bachelor's thesis[11]. Their implementation, using 64 ANN:s for their Reversi agent, one ANN for each move, was able to detect patterns. To summarize, the agent improved during time, against agents with different policies.

This implies, along with the result from Forsberg and Mattsson, that the implementation and not only the algorithm is of most importance for the result.

# Chapter 4

# Analysis

This chapter outlines choices made in the implementation, the approach used and how results were analyzed. The constraints from the original problem are presented in Section 4.1. The methods used are explained in Section 4.2. Section 4.3 describes the agents, both the Q-learning agent as well as other agents that were ones. Finally, Section 4.4 presents methods used for measuring performance with different configurations.

## 4.1 Constraints

To scale down the problem, it was necessary to put constraints on the implementation. This was done to get useful results while keeping the complexity of the implementation as well as the scope of the study at a reasonable level.

The constraints placed on the game are as follows:

- A full-size board is too expensive.
- A full set of tiles is too expensive.
- The agent will always play first.

The constraints placed on the agent are as follows:

- The learning rate and discount factor, variables $\eta$ and $\gamma$, are not changed.
- The agent will not detect and merge states from different move orders that are identical.

The board size and the number of tiles available are two interrelated quantities. If the number of available tiles drastically decreased and the board size was kept the same all players would have more than enough room for all of their available tiles. This would lead to a lot of, if not only, draws. Keeping all 21 tiles is simply not feasible as state space becomes ridiculously large. To find a lower limit to

this state space a single player was considered. He plays alone instead of against someone and is always able to place all tiles available but only in one way. This crude approximation would give a state space of 21! different states, between two and three times more than the number of addressable bytes in a 64-bit computer. Approaching this original problem with Q-learning was clearly not feasible and the number of tiles has to be reduced. To still have a game where players could block the other player the board size was reduced as well.

Making the agent always play first was to simplify analysis of gameplay results. The choice of keeping $\eta$ and $\gamma$ unchanged is simply because they are outside the scope of this report.

The data structure used to represent state space is a tree where each node represents a state and states which can be reached from this node are represented as children. When the agent reach a state for the first time, new state node children for that state are created, eliminating "impossible" states from taking space in the data structure.

Not merging identical states was done out of simplicity. In retrospect this could have been done using a hash table.

## 4.2 Method

The size of the board was limited to 5 by 5 and the number of tiles to 4 (all polyominoes of order 3). There was a positive reward of one given for each square on a tile that the agent places and a negative of one for each square on a tile that the opponent places. Winning will give a positive reward equal to twice the amount of squares on a player's starting tiles. Losing will give a negative reward of the same quantity. Ending up in a draw will give the agent a negative reward of half of what it receives when losing. This means that the agent will prefer draws over losing and winning above all.

For all studies $\eta$ and $\gamma$ values of 0.8 were used, as suggested by the previously mentioned thesis by Arvidsson and Wallgren[10]. It should be pointed out that these values were suggested for agents which unlike ours used no intermediate rewards. Finding optimal values for $\eta$ and $\gamma$ was outside the scope of the report and these values were used either way.

The different policies for random behavior use different functions to change a corresponding $\epsilon$ during learning. Epsilon-greedy uses a fixed value while other change based on the number of played games and one policy also depend on the number of played tiles in the current game.

All methods were modeled after that there is no learning phase. The $\epsilon$ policy for the agent will be used for its entire lifespan.

To measure the performance of the Q-learning agent, the number of wins, losses and draws were collected during the learning period. The Q-learning agent learned playing 2 player games against all opponents, including itself, using different policies. For each opponent, the Q-learning agent would start knowing nothing about the

game. Each $100^{th}$ game, total number of wins, draws and losses were saved. This was done until for 100000 games for each test. Over time it was possible to see how the agent progressed and how many games were necessary for a good policy to develop.

### 4.2.1 First Policy – Fixed Epsilon-Greedy Policy

A fixed $\epsilon$ value is a common method used in the $\epsilon$-greedy policy. Other methods were compared to this textbook policy to see if they were rewarding to use.

Low values for $\epsilon$ means that the agent will not explore other options very often. A high value means that the agent will explore other options and learn from them but instead not use what it has previously learned. The values 0, 0.2, 0.5 and 0.8 were chosen as reasonable values to compare for $\epsilon$. It should be noted that $\epsilon = 0$ means deterministic behavior.

### 4.2.2 Second Policy – Epsilon-Decreasing Policy

A decreasing $\epsilon$ value is motivated by that when the agent starts playing it will know nothing about the game. Therefore completely random actions are appropriate to have an exploratory behavior at the beginning. As the agent learns more about the game it will be more capable of estimating which move is the best. Thus a decreasing $\epsilon$ value seems appropriate.

The following formula for a decreasing $\epsilon$ was investigated as a second policy:

$$\epsilon = \frac{c}{c + |runs|} \tag{4.1}$$

Epsilon will decrease with the number of finished games and the function will converge to 0. Thus the agent's policy will converge to *deterministic* behavior. The rate at which $\epsilon$ decreases will depend on which constant $c$ is chosen. This $c$ was given values 1, 16, 100, 200, 500, 1000 and 2000.

### 4.2.3 Third Policy – Depth-Dependent Decreasing Epsilon

State space branches out from the first state (empty board) as most states have options they will give birth to at least a few new states. Therefore on a certain *depth* in the acyclic state-space graph, in our implementation a tree, more states are expected. That is, when more pieces have been placed the board have more possible states. States on a level further down are thus expected to be less visited. Therefore when visiting a state further down in the graph it should be more probable to select a random move as it is less likely that it has been done before.

A policy where the $\epsilon$ converge with different rates depending on their depth takes account for this. The following formula for a decreasing $\epsilon$ was investigated as a third policy:

$$\epsilon = \frac{c^{performed\_moves}}{c^{performed\_moves} + |runs|} \tag{4.2}$$

It is important to note that games with repeated states don not have this acyclic-graph behavior. In those "depth" is not clearly defined. This policy was indented for use in problems where *depth* is clearly defined.

In the studied implementation *depth* was defined as the $n^{th}$ tile to be placed from that player. This limits depth to the total number of tiles per player. With 4 available tiles and a $c$ value of 5 has $c^{performed\_moves}$ ranging between $5^1$ and $5^4$, that is 5 and 625, which correspond to the $c$ value in the previous policy.

$c$ values of 1.5, 2, 3, 4 and 5 were studied.

## 4.3 Agents

The implementation holds two kind of agents; the main agent using Q-learning and opponent agents with static strategies.

### 4.3.1 Q-Learning Agent

The Q-learning agent is the one whose learning rate using different random-behavior probabilities is the focus of this research. When encountering previously unvisited states their Q-values are initially set to 0. Another approach that was not used is to initialize them to random values.

Except for the different new random-behavior policies that change during training the agent acts just according to the Q-learning algorithm.

### 4.3.2 Other Agents

There were three agents with different policies working as opponents to the Q-learning agent.

- The *simple agent* always choose the first action, which places the smallest tile.
- The *greedy agent* always choose the last action, which places the largest tile with the idea to gain as much area as possible.
- The *random agent* always choose actions arbitrarily.

## 4.4 Performance Measurement

The simple and greedy agents were primarily used to assert that our Q-learning agent reliably beat agents using a fixed strategy.

The opponent used in the $\epsilon$-evaluating studies was a random agent. This ensured that our agent learned characteristics of the game and not how to beat a single player. A random player is also the hardest player for a Q-learning agent to reliably beat.

To be able to evaluate how good the agent performed a reference run was performed where the random agent got to play against itself. This was done to measure

a possible first-move advantage to have appropriate reference values. Because of this first-move advantage if the agent won as often as it lost it would be presumed to be performing worse than its opponents. An average of the win/draw/loss ratio was used to be able to be able to draw conclusions from the other results.

A final test was performed to assert the agent's performance against itself.

# Chapter 5

# Results

This chapter contains the result from the study using the policies described in Chapter 4. Each Section here holds the results from different policies. Section 5.1 contains measurements using a fixed epsilon-greedy policy. Next, in Section 5.2, the results using epsilon-decreasing policy is presented. Finally are the results from the third policy, depth-dependent decreasing epsilon, in Section 5.3. For the complete set of graphs available, including the ones presented here, please refer to Appendix B.

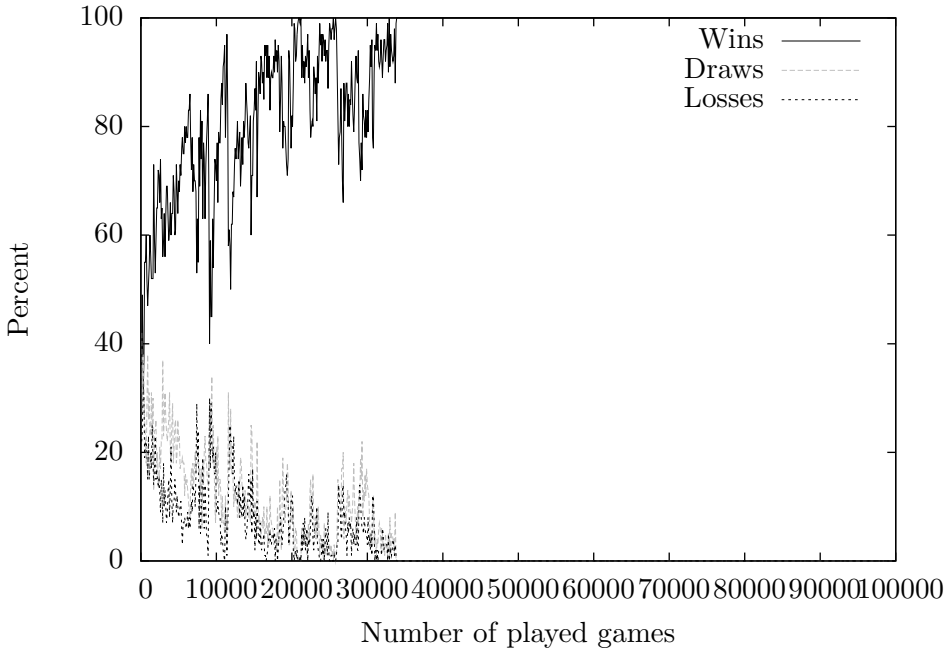


**Figure 5.1.** Results for the Q-learning agent playing against the random agent with a fixed $\epsilon$ of 0

## 5.1 Fixed Epsilon-Greedy Policy

The focus has been on the tests where the random agent, presented in Section 4.3.2, was the opponent. This is because both the simple and greedy agent were defeated after less than 1000 games for all configurations of this policy. For $\epsilon = 0.2$ a perfect win rate of 100% is reached in around 1000 played games against both static strategies. The configuration of $\epsilon = 0$ reaches a perfect win rate before 100 games for the greedy agent and 200 games for the simple agent.

The test on the first policy against the random agent which performed best is presented in Figure 5.1. It can be seen that just before 35000 played games is where the policy reach a 100% win rate.

None of the tested non-zero $\epsilon$ values converged to a 100% win rate.

**Average performance for epsilon-greedy policy, $\epsilon = 0$**

Wins 93.252% Draws 4.015% Losses 2.733%

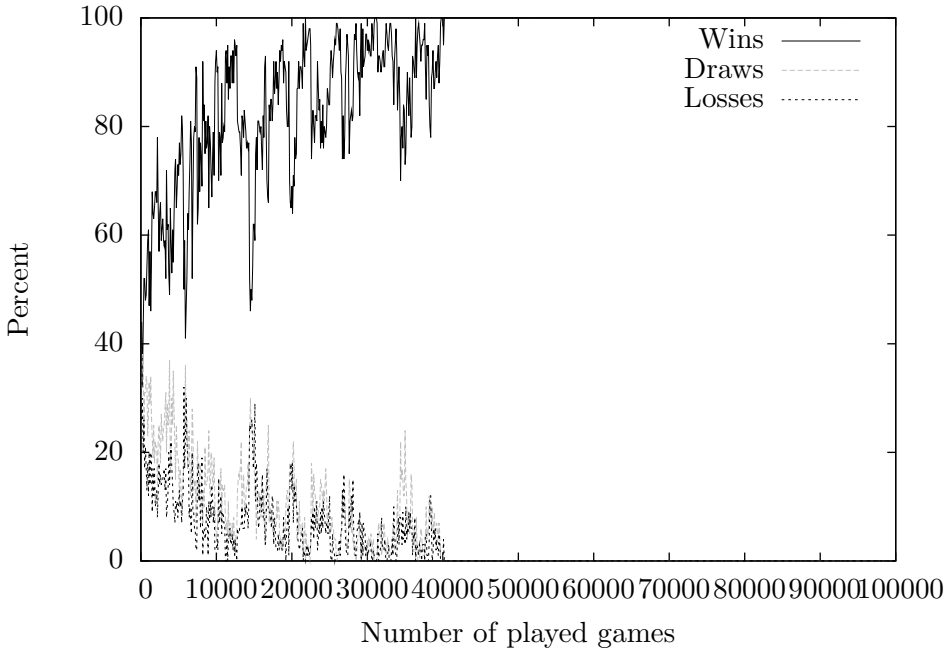


**Figure 5.2.** Results for the Q-learning agent playing against the random agent with the configuration of $\epsilon = 1/(1 + |runs|)$, using the epsilon-decreasing policy

## 5.2 Epsilon-Decreasing Policy

For the Epsilon-Decreasing Policy both the simple and greedy agent are defeated quickly compared to the random agent, this time under less than 300 games for all the configurations. Because of this, the focus was on the test with the random agent being the opponent. For a full list of graphs versus the random agent using this policy see Appendix B.3.

In Figure 5.2 the configuration giving the best performance for the Q-learning agent is displayed. The win rate of 100% in this test is reached first after around 40000 played games.

Draw rate is almost always higher than the loss rate.

**Average performance for epsilon-decreasing strategy,** $\epsilon = 1/(1 + |runs|)$

Wins 92.634% Draws 4.407% Losses 2.959%

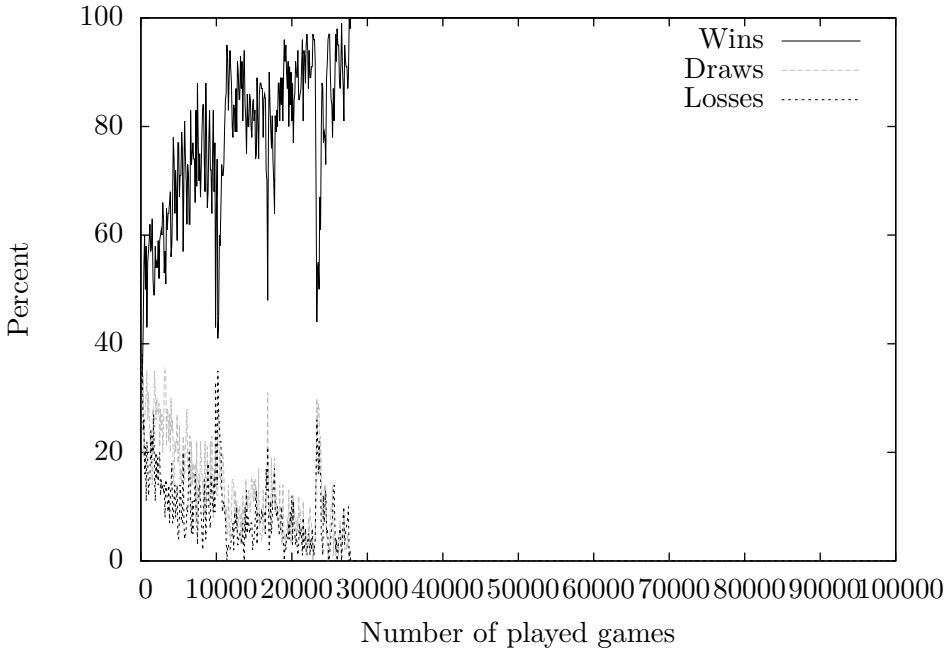


**Figure 5.3.** Performance of the Q-learning agent playing against the random agent with configuration $\epsilon = 4^{performed\_moves}/(4^{performed\_moves} + |runs|)$ using the depth-dependent decreasing-epsilon policy.

## 5.3 Depth-Dependent Decreasing Epsilon

The opponent most difficult to defeat was by far the random agent. This is the reason the test with the random agent playing the opponent were in focus. It took around 400 games on the best configuration of $\epsilon = 4^{performed\_moves}/(4^{performed\_moves} + |runs|)$ for the Q-learning agent to get a 100% win rate over both the greedy agent and the simple agent.

The same configuration against a random agent took under 30000 played games for the Q-learning agent to reach a 100% win rate, here displayed in Figure 5.3.

The Depth-Dependent Decreasing Epsilon policy had the best average win rate as well as being the fastest to converge to a 100% win rate.

**Average performance for depth-dependent decreasing epsilon,**
$\epsilon = 4^{performed\_moves}/(4^{performed\_moves} + |runs|)$

Wins 93.706% Draws 3.711% Losses 2.583%

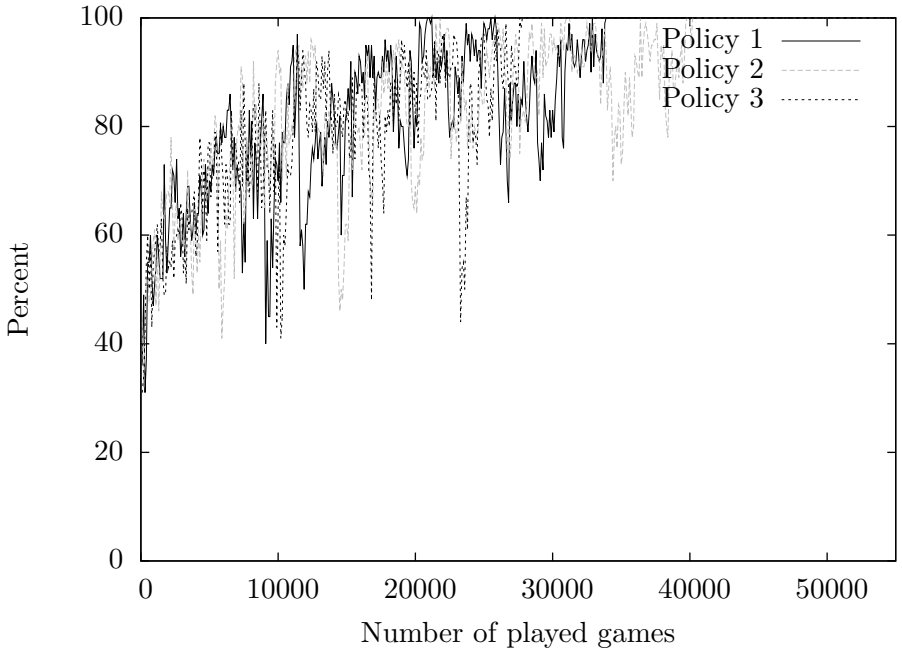


**Figure 5.4.** Win rate for the Q-learning agent for each policy using the parameters giving the best test results.

## 5.4 Comparison

To compare the policies different performances, both average win rate and win rate over time are presented here. Presented data are from the test results of the Q-learning agent versus random agent.

Figure 5.4 compares the best results for each policy, displaying the win rate for each over the number of games played. They can be seen individually in Figure 5.1, 5.2 and 5.3. Please note that the x-axis only range up to 55000 played games, compared to the individual figures which ranges from 0 to 100000.

The first policy maintains 100% win rate after almost 35000 played games. The second policy maintains 100% after about 40000 games and the third after almost 30000 games.

### Average win rate over the best test result for each policy

Wins first policy: 93.252% Wins second policy: 92.634% Wins third policy: 93.706%

# Chapter 6

# Conclusions

This chapter contains conclusions in Section 6.1 drawn from the results achieved in Chapter 5 and the theoretical background and constraints presented in Chapter 4. There are also some thoughts on how the game instance have affected the results in Section 6.2 and some suggestions on further investigations are given in Section 6.3.

## 6.1 Policy Analysis

This thesis' aim was to design new policies for Q-learning and to evaluate them. In this section is an analysis of how worthwhile each policy is as well as reasoning on why the results look like they do.

### 6.1.1 Fixed Epsilon-Greedy Policy

The fact that our $\epsilon$-greedy policy where $\epsilon$ was 0 worked at all is implementation specific. All values were initially zero when unexplored. When a state leads to a loss it eventually gets a negative value, meaning that unexplored options will be explored as soon as it gets known that the current strategy leads to a loss or draw.

It does not necessarily lead to the agent finding the path to maximized reward as it will only change strategies if it loses and be happy with any result where the agent wins. This is why an $\epsilon$ of 0 is not textbook Q-learning. Q-learning is supposed to eventually explore the entire state space and converge to a maximum. In this instance the agent can find a reliable way to win regardless of the opponent's moves. In other instances it is possible that the agent might have had to settle for a draw occasionally or even to lose.

For the sake of completeness another example of epsilon-greedy should be discussed as well. Because when using an $\epsilon$ of 0.2 results were less successful. See Figure B.3 in Appendix B. Using this value seems to lead to the agent converging to winning only around 75-80% of the time. This corresponds to the agent performing random moves 20% of the time.

From this a conclusion can be drawn that this particular game or instance of the game is not forgiving of bad moves. Because doing the correct move 80% of the time isn't enough to win reliably even though the agent has the first-player advantage.

Higher $\epsilon$ values will make the agent learn faster, but ignore what it has learned far too often. We can't be sure that this $\epsilon$ value is optimal but it seems to work well.

### 6.1.2 Epsilon-Decreasing Policy

The Epsilon-Decreasing policy seems very rewarding. It should work in all types of games which have a finite number of moves, as the more an agent explores the less it needs to explore later.

It's probable that 1 is the optimal constant $c$ in the formula (4.1), it's just the best one found. 4 for one might have been even better. Finding optimal values is not important as it is very game dependent as well as game-instance dependent. The game Blokus itself is largely outside the scope of the report.

Some of the graphs, presented in Appendix B.3, seem to reach an optimal win rate and win for a few ten thousand games but then start losing for a while. This is assumed to be that after a strategy has seemed good for a while the opponent finds a case where the player actually loses. This means that new strategies have to be explored again. After gradually figuring one out the agent goes back to a good win rate. This would probably keep on happening if even more games are played than the ones displayed in the graphs.

The graphs with a very high constant $c$, see formula 4.1 in Section 4.2, maintain a portion of random behavior for a very long time. This is believed to be better for other games which take a lot longer to learn.

### 6.1.3 Depth-Dependent Decreasing Epsilon

This strategy focuses on explorations that are relatively untested further down the tree rather than questioning moves done near the root. These early moves will mostly to be tested early in the game and presumed to still be good. It tries to perform minor optimizations in estimated optimal strategy by questioning later moves rather than early ones.

A few of these graphs show that after a while the agent start losing or playing draws. Just like in the previous policy this is believed to be because the agent finds out a flaw in its current strategy and has to consider new earlier moves and relearn.

The constant $c$, see formula 4.2 in Section 4.2, in this policy is believed to correlate to the average branching factor of the game. Thus it is believed to be most effective in games where the moves have a fairly constant branching factor excluding repeated states.

### 6.1.4 General Conclusions

Overall it's not clear that when a policy hits a 100% win rate it will stay that way, as shown in the graphs. The agent might learn of some special cases where it would lose and try to re-learn a better, or safer, strategy.

All results are highly dependent on the game being played. This is probably a result of the few number of tiles the players have. One game is over so quickly, critical moves have a very high impact on the result.

## 6.2 Game Impact

Unlike the original game and a lot of other games, in this instance of Blokus a single misplaced tile quite often leads to disastrous results. From manually playing this instance it has been noted that a player can actually be locked out from the game in as few as two moves by the starting player.

This large focus on placing every single tile correctly makes $\epsilon$-greedy perform extra bad. To exaggerate, choosing random tile placement 20% of the time 20% of the moves could be missteps that would lead to a loss. The situation when using the $\epsilon$-greedy is similar. This explains why the agent performs so bad in our case though using a common random-behavior policy. In the original version of Blokus these missteps would likely cause small fluctuations in score which the agent could learn to overcome by dominating the random agent.

## 6.3 Further Research

Unfortunately this study of new policies showed inconclusive results. As there's theoretical reasoning behind them, see Section 4.2, further research has potential to uncover their uses.

In particular research in the following areas are suggested:

- *Board size and number of tiles.* Two constraints of this study were the use of a 5 by 5 board instead of the original 20 by 20 board and the use of 4 tiles instead of 21. Reasoning suggest that a larger board would be more forgiving for misplaced tiles, which would affect the results for the policies. The impact of the number of tiles used was not investigated either but probably had some effect as well. These are both points that could be investigated in further research is motivated.

- *Different games/applications.* This same study could be conducted on another problem than the board game Blokus. There's no reason to keep them both connected.

- *Maximizing reward.* Instead of measuring wins, draws and losses, accumulated reward could have been evaluated instead. This would have given different

results and graphs. Had that been done the new policies would likely defeat the $\epsilon$-greedy with $\epsilon = 0$, because it is likely to get stuck in a local maximum.

- *Merging states.* Another constraint of this study was to not implement the possibility of detecting and merging identical states in the tree data structure. It is unclear how much impact this would have had on the learning rate and storage of state space.

# Bibliography


[1] Wikipedia, "Machine Learning." http://en.wikipedia.org/wiki/Machine_learning, 2011-03-07.

[2] Chris Watkins, *Learning from a Delayed Reward*. PhD thesis, Cambridge, 1989.

[3] Wikipedia, "Single-Player Games." http://en.wikipedia.org/wiki/Game#Single-player_games, 2011-03-21.

[4] Neil J. A. Sloane, "A000105: Number of polyominoes (or square animals) with n cells." On-Line Encyclopedia of Integer Sequences – http://oeis.org/A000105, 2009.

[5] Wikipedia, "First-Move Advantage in Chess." http://en.wikipedia.org/wiki/First-move_advantage_in_chess, 2011-03-21.

[6] Örjan Ekeberg, "Lab 4: Reinforcement Learning." KTH, DD2341: Machine Learning – http://www.csc.kth.se/utbildning/kth/kurser/DD2431/ml10/rl.pdf, 2010.

[7] Richard S. Sutton & Andrew G. Barto, *Reinforcement Learning: An Introduction*, pp. 3–16, 52–83, 89–109. The MIT Press, 1998.

[8] Stephen Marsland, *Machine Learning: An Algorithmic Perspective*, pp. 6–7, 293–312. Chapman & Hall/CRC, 2009.

[9] Wikipedia, "Artificial Neural Network." http://en.wikipedia.org/wiki/Artificial_neural_network, 2011-03-20.

[10] Oskar Arvidsson & Linus Wallgren, "Q-Learning for a Simple Board Game," Bachelor's thesis, KTH/CSC, 2010.

[11] Klas Björkqvist & Johan Wester, "Självlärande Othello-spelare," Bachelor's thesis, KTH/CSC, 2010.

[12] John Forsberg & Lukas Mattsson, "Artificial Intelligence using the QLearning Algorithm," Bachelor's thesis, KTH/CSC, 2010.

# Appendix A

# Terms and Acronyms

**Agent** Also called learner. A program implemented to learn and interact.

**AI** Artificial Intelligence.

**ANN** Artificial Neural Network, model used to process information.

**Consumer** Refers to a theoretical piece of code which makes use of the mutability data collected.

**Reinforcement learning** Is an approach in Machine Learning concerned with how an agent ought to act in an environment so as to maximize the cumulative reward.

**Policy** The policy for choosing the next action. Commonly used policies are; greedy, $\epsilon$-greedy, Boltzmann distribution and random.

**State value** Each state in the current policy has a value of estimated future reward.

**Temporal difference** A class of learning methods which improve the estimated state values in every time step. For more information, see [7] page 133.

$\epsilon$ The probability of random action in the $\epsilon$-greedy policy. Pronounced epsilon.

# Appendix B

# Graphs

The first graph shows the results of a random agent playing against another random agent, where the results of the one playing the first move are displayed. Last is the performance of a Q-learning agent playing against another Q-learning agent, using the depth-dependent decreasing-epsilon policy. All other graphs are the performances of the main Q-learning agent playing against the random agent. For a description of the random agent, please refer to Section 4.3.2. Under each figure the configurations for the policy used are given.

## B.1 Random Run

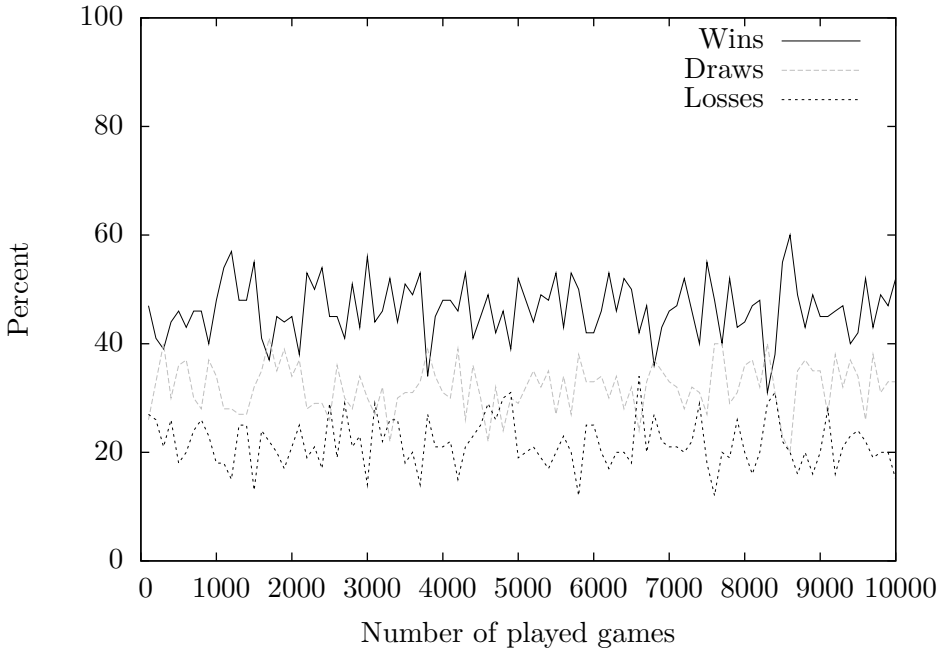


**Figure B.1.** Random run

### Random Average

Wins: 46.45% Draws: 31.98% Losses: 21.57%

## B.2 Fixed Epsilon-Greedy Policy

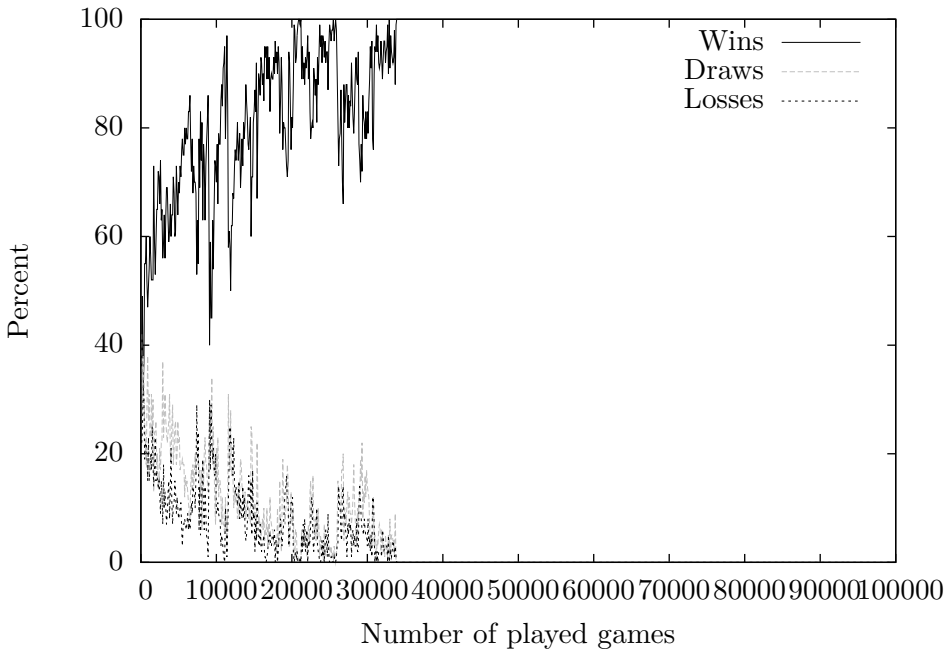


**Figure B.2.** $\epsilon = 0$

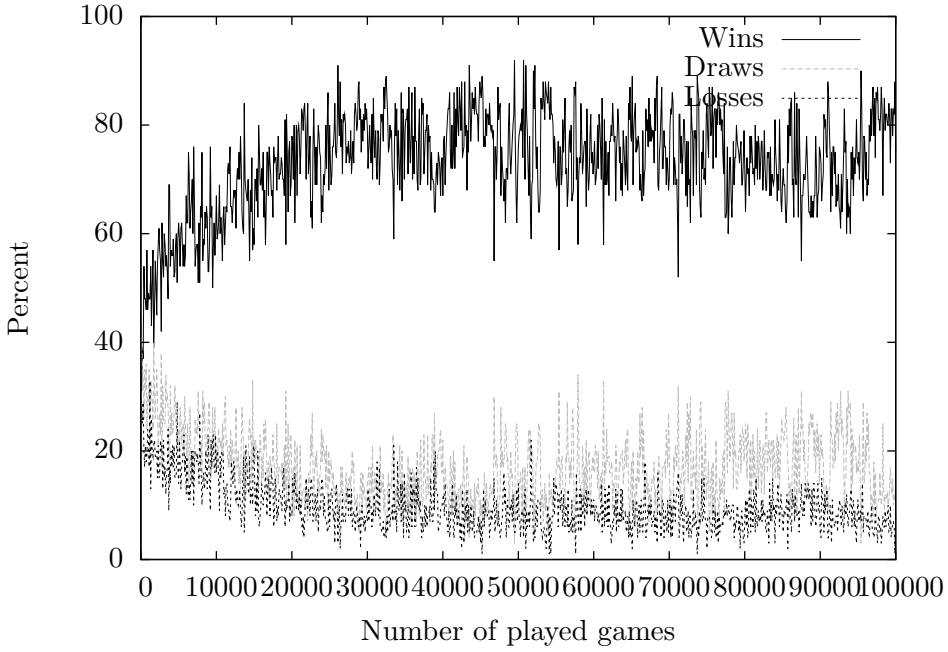


**Figure B.3.** $\epsilon = 0.2$

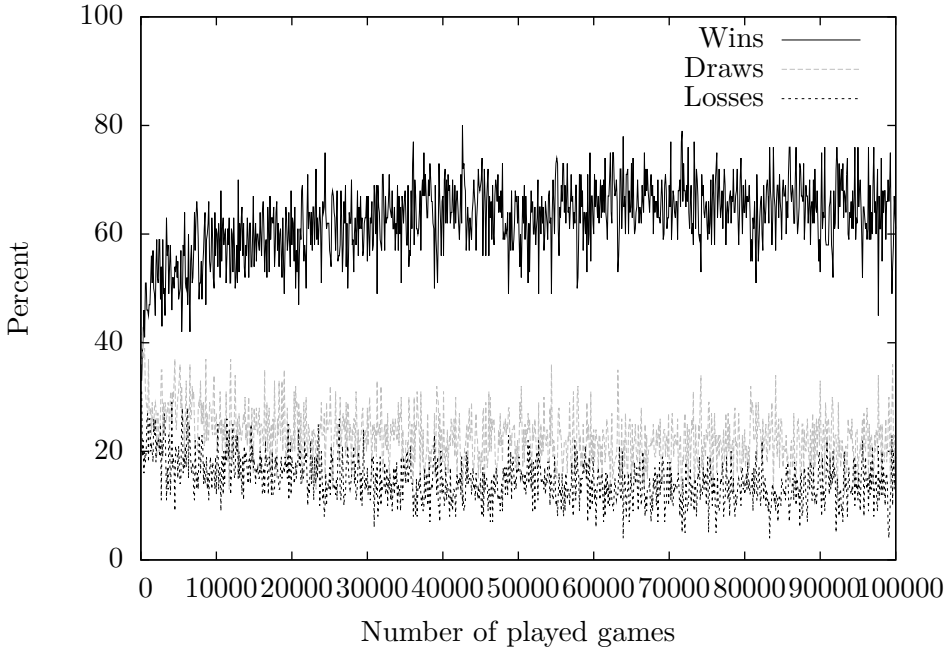


**Figure B.4.** $\epsilon = 0.5$

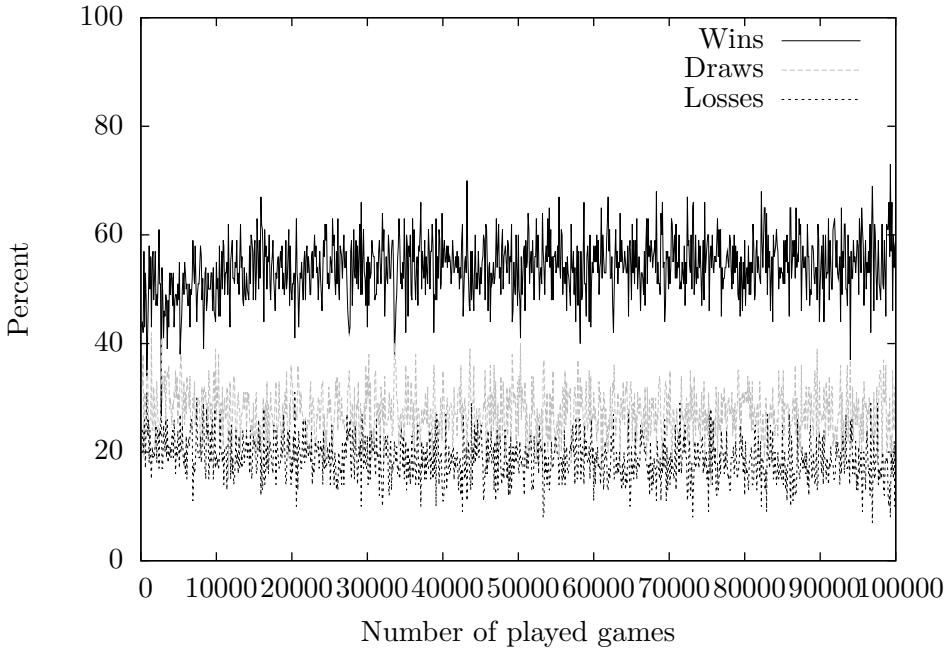


**Figure B.5.** $\epsilon = 0.8$

## B.3 Epsilon-Decreasing Policy

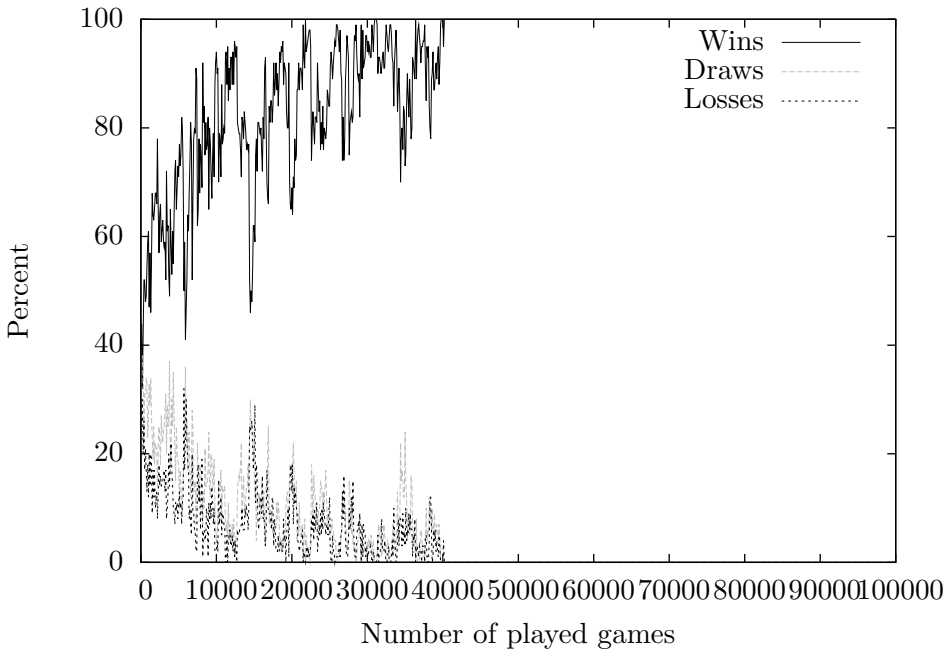


**Figure B.6.** $\epsilon = 1/(1 + |\text{runs}|)$

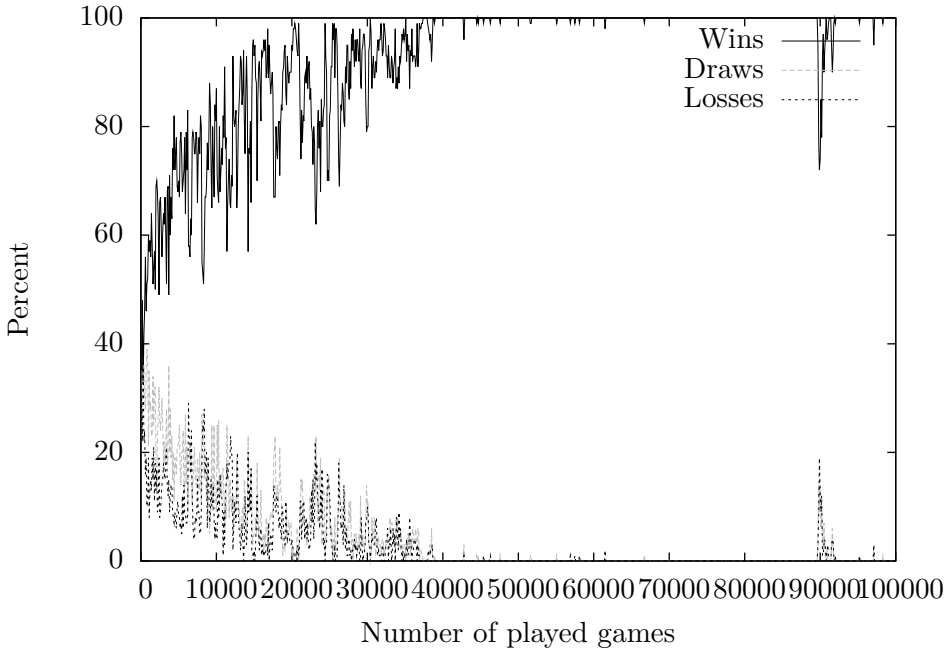


**Figure B.7.** $\epsilon = 16/(16 + |\text{runs}|))$

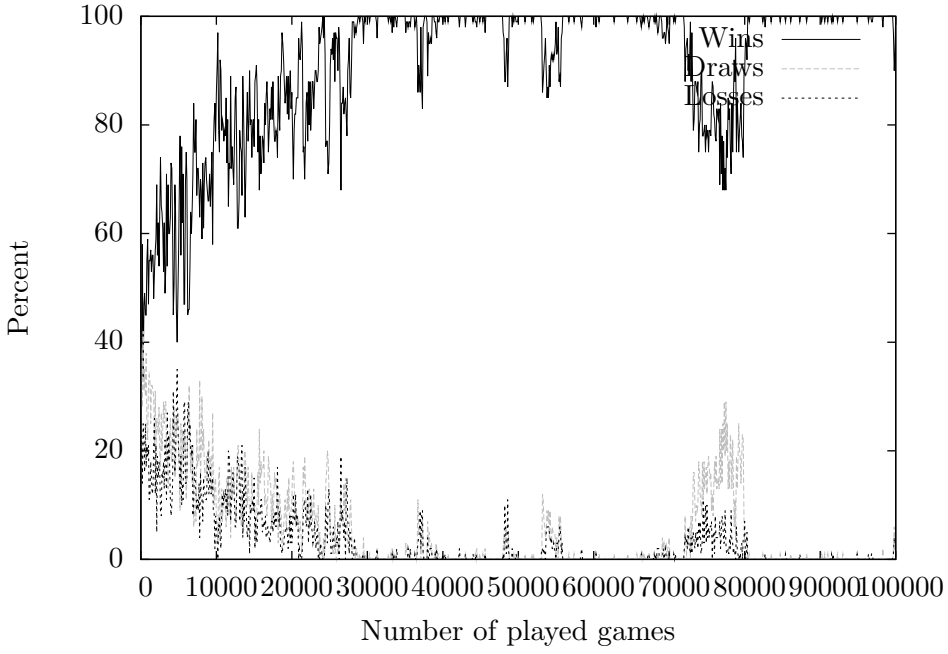


**Figure B.8.** $\epsilon = 100/(100 + |\text{runs}|)$

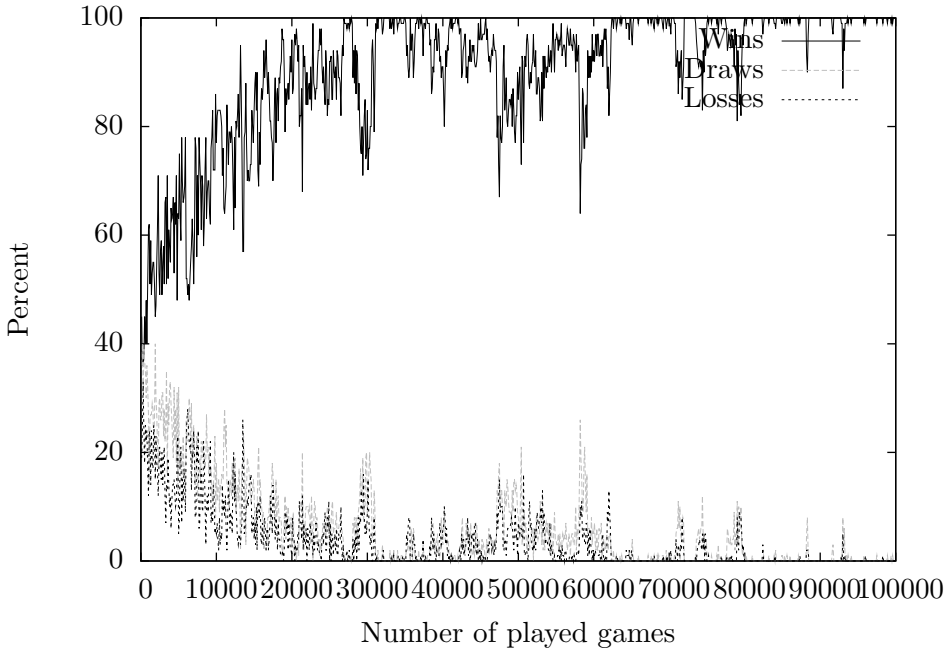


**Figure B.9.** $\epsilon = 200/(200 + |\text{runs}|)$

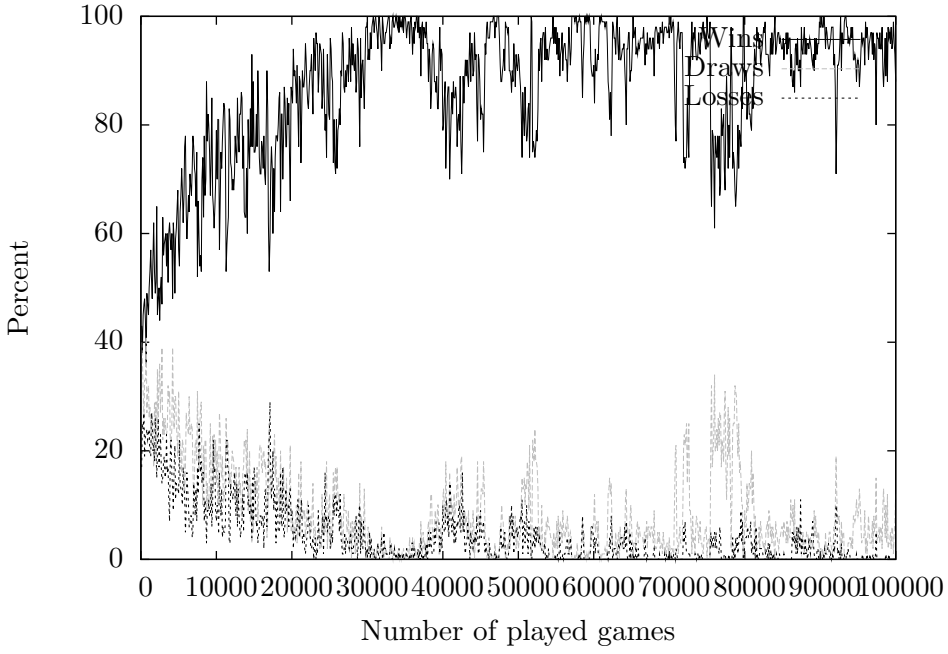


**Figure B.10.** $\epsilon = 500/(500 + |\text{runs}|)$

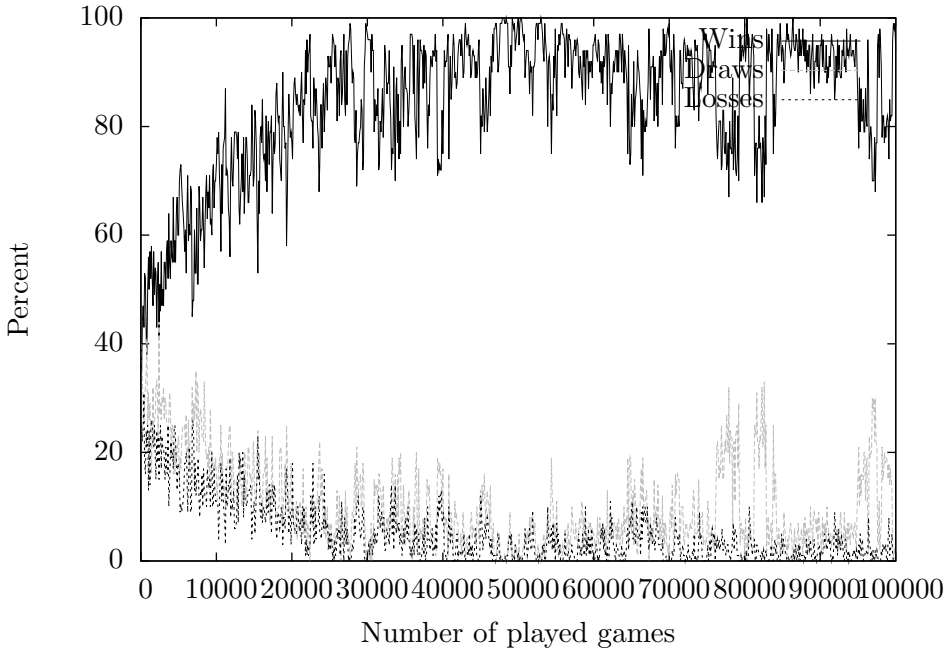


**Figure B.11.** $\epsilon = 1000/(1000 + |\text{runs}|)$

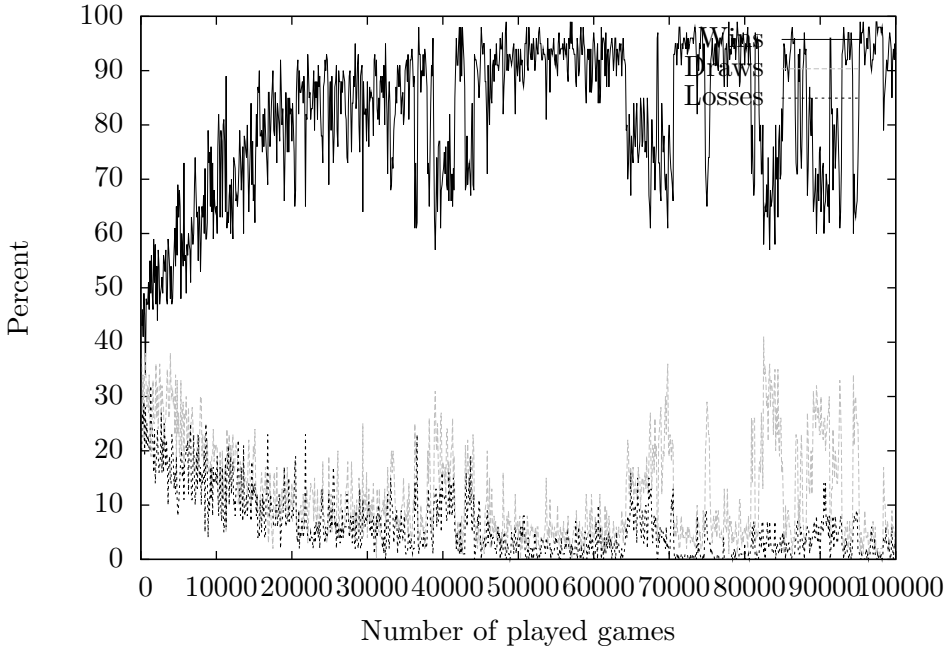


**Figure B.12.** $\epsilon = 2000/(2000+\text{runs})$

## B.4 Depth-Dependent Decreasing-Epsilon

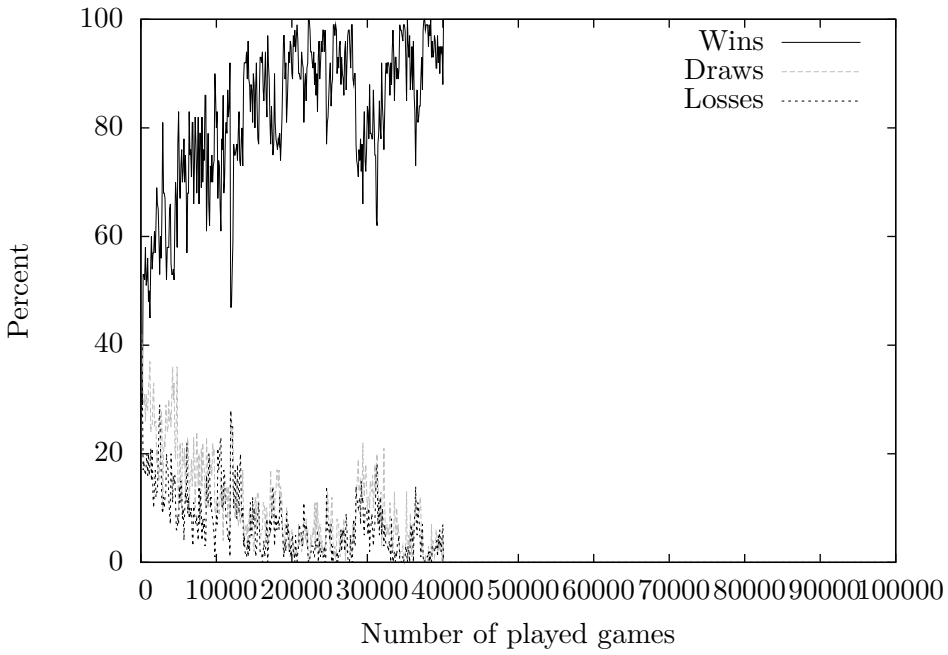


**Figure B.13.** $\epsilon = 1.5^{performed\_moves}/(1.5^{performed\_moves} + |runs|)$

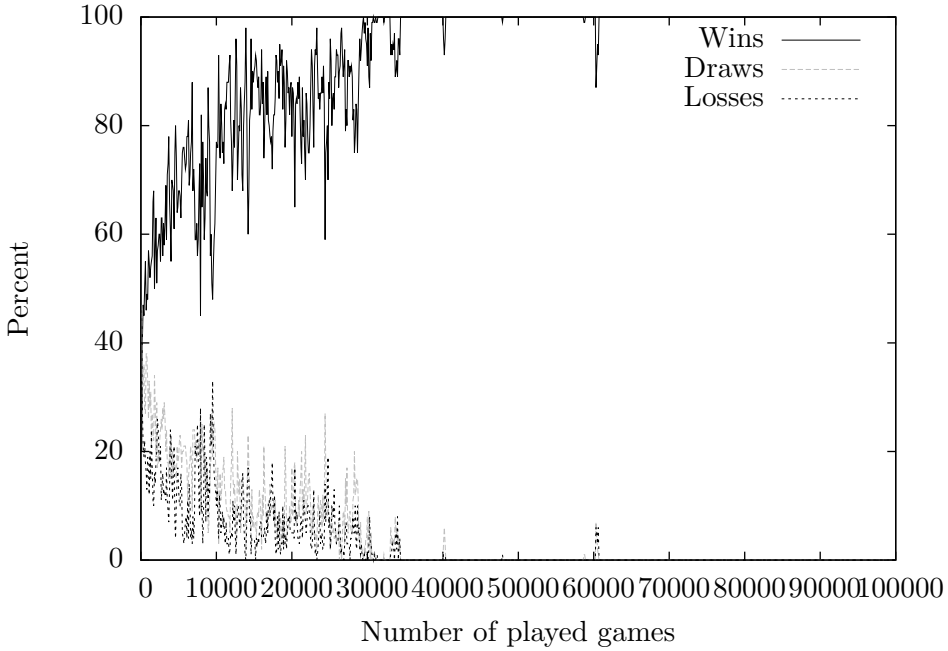


**Figure B.14.** $\epsilon = 2^{performed\_moves}/(2^{performed\_moves} + |runs|)$

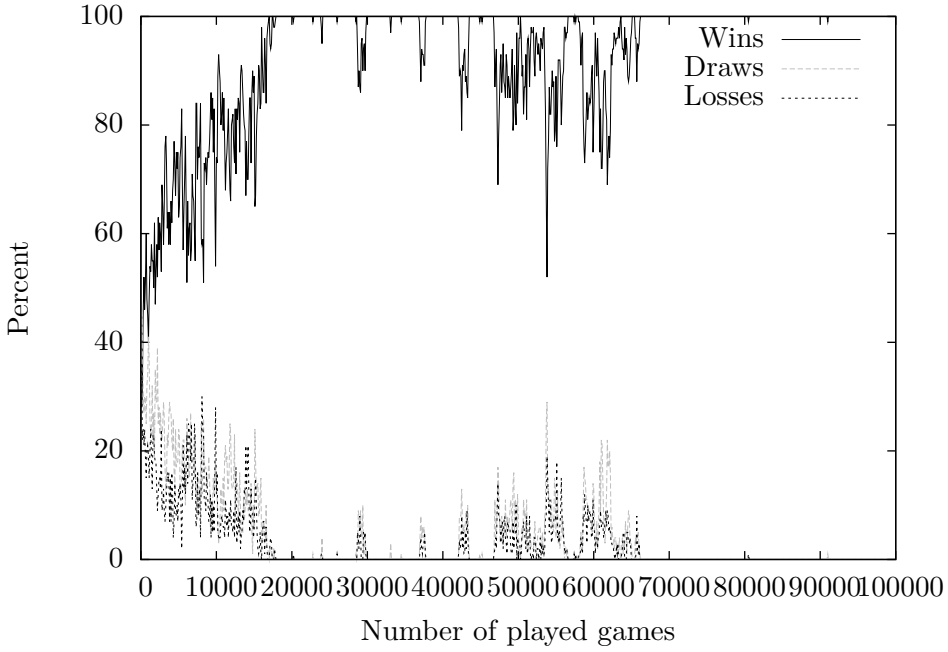


**Figure B.15.** $\epsilon = 3^{performed\_moves}/(3^{performed\_moves} + |runs|)$

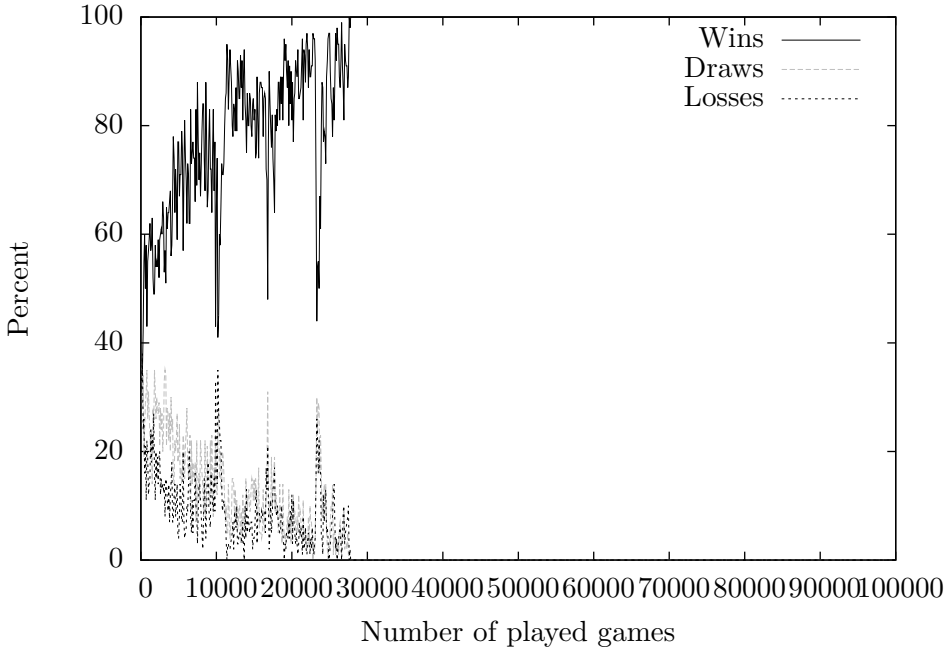


**Figure B.16.** $\epsilon = (4^{performed\_moves}/(4^{performed\_moves} + |runs|)$

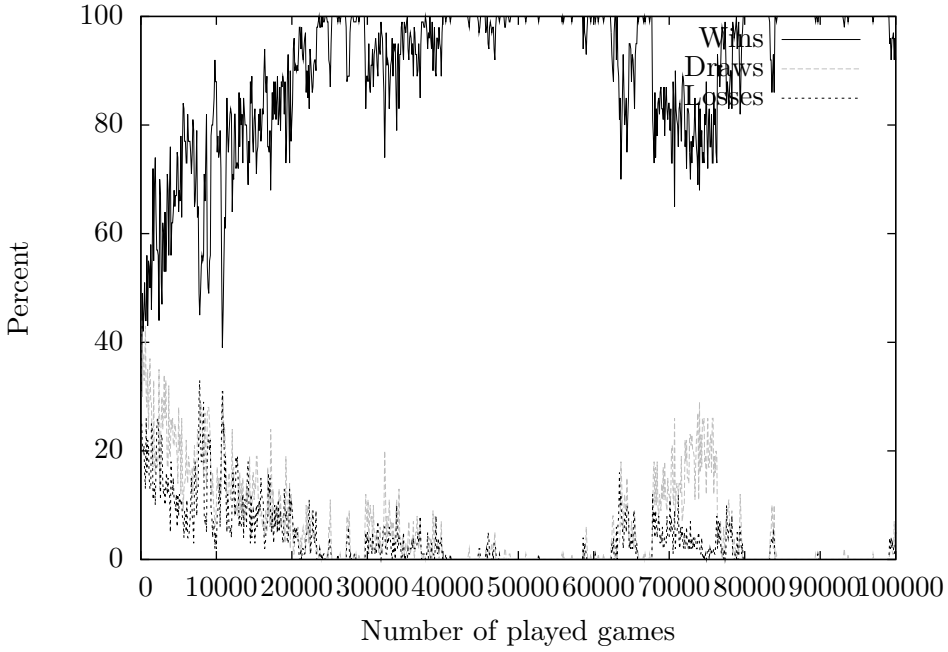


**Figure B.17.** $\epsilon = (5^{performed\_moves}/(5^{performed\_moves} + |runs|)$

### B.4.1 Agent vs. Agent – Depth-Dependent Decreasing-Epsilon Policy

Due to the short time it took for the agent to beat itself, please note the x-axis only range from 0 to 10000 played games.

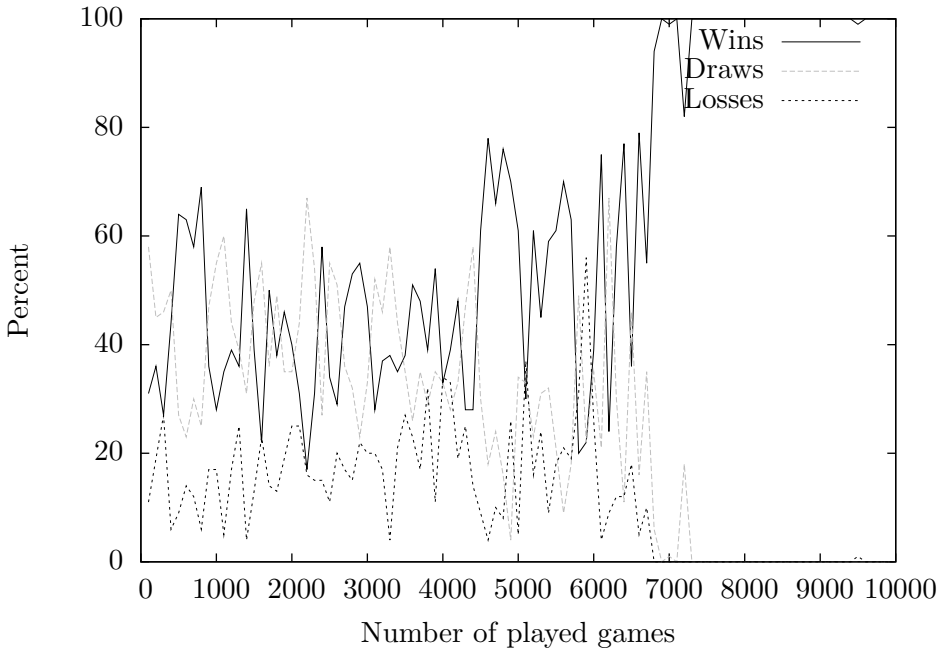


**Figure B.18.** $\epsilon = (1.5^{performed\_moves}/(1.5^{performed\_moves} + |runs|)$