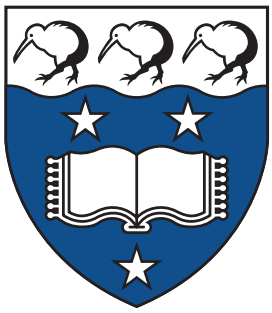
Integrating Reinforcement Learning into Strategy Games

by

Stefan Wender

Supervised by Ian Watson

The University of Auckland
Auckland, New Zealand



A Thesis submitted in fulfillment
of the requirements for the degree of

Master of Science in Computer Science

The University of Auckland, February 2009

Abstract

The present thesis describes the design and implementation of a machine learning agent based on four different reinforcement learning algorithms. The reinforcement learning agent is integrated into the commercial computer game *Civilization IV*. Civilization IV is a turn-based empire building game from the *Civilization* series. The reinforcement learning agent is applied to the city placement selection task. The city placement selection determines the founding sites for a player's cities.

The four reinforcement learning algorithms that are evaluated are the off-policy algorithms one-step Q-learning and Q(λ) and the on-policy algorithms one-step Sarsa and Sarsa(λ). The aim of the research presented in this thesis is the creation of an adaptive machine learning approach for a task which is originally performed by a complex deterministic script. Since the machine learning approach is nondeterministic, it results in a more challenging and dynamic computer AI.

The thesis presents an empirical evaluation of the performance of the reinforcement learning approach and compares the performance of the adaptive agent with the original deterministic game AI. The comparison shows that the reinforcement learning approach outperforms the deterministic game AI. Finally, the behaviour and performance of the reinforcement learning algorithms are elaborated on and the algorithms are further improved by analysing and revising their parameters. The performance of the four algorithms is compared using their identified optimal parameter settings and Sarsa(λ) is shown to perform best at the task of city site selection.

Acknowledgements

I am very grateful for the supervision and support of my advisor Ian Watson. Without his ideas and guidance this thesis would not have been possible.

My fellow computer science students Sebastian, Paul, Ali and Felix made sure that the time in the lab was never wasted, or at least never wasted alone. Thank you for many insightful and enlightening conversations.

The comments and time given by my readers Clare and Christine greatly improved and clarified this thesis.

I would especially like to thank my family who have supported me in every possible way for all my life, whatever I wanted to do, wherever I wanted to go. To my mum and dad, Susanne and Bernd, and to my brother and sister, Andreas and Christine, thank you for always being supportive.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction, Motivation and Objectives

Despite the fact that commercial computer games still suffer by and large from the shortcomings in artificial intelligence (AI) that John Laird mentioned in his keynote address at the AAAI 2000 conference (Laird and van Lent, 2001) there have been tremendous developments in the area of using commercial computer games as testbeds for AI research. This development has been made possible by substantial improvements in game technology in recent years.

While learning computer game AI in commercial games is still almost non-existent, the graphics in the games today are closer than ever to photorealism. Laird predicted this peak in graphics as a turning point. After reaching perfection in terms of graphics, he assumed, focus in computer game development would shift away from visuals and towards more captivating gameplay, including game AI.

Computer games have changed from being the hobby of a small group of people to being one of the mainstream entertainment industries. This transformation was already well underway when Laird held his keynote address. However, the market for computer games was still a lot smaller than today, when several publishers have turnovers of billions of dollars and single games like *Grand Theft Auto IV* or *World of Warcraft* yield hundreds of millions of dollars in return. The size of projects also implies that games have become far more complex and extensive than before. Increasing game complexity also affects the development process: The most recent commercial computer games go through a software life cycle that includes thorough quality assurance. The source code of a game is now developed by large teams of software engineers and thus has to be highly modular and of high quality to enable parallel work with several people working on the same code. However, since nearly all commercial games are closed source, the more recent trend of including extensive interfaces for modifications through users is very important for academics that want to use commercial games as testbeds.

Publishers have started to realize that the length of the lifetime of a game largely depends on how active the respective users, the *community* of that game, are involved with the game. Therefore the publishers have started to create more and more powerful tools for community

developers who want to extend and modify the game. What started with simple map editors in games like *Warcraft II* or *Doom* has now evolved through very capable scenario editors in games like *Bioware*'s *Baldurs Gate* series into entire scripting languages that allow the community to create content of their own for their favourite game. The scripting language *Lua* can be used to create powerful user interfaces in *World of Warcraft* and allows users to specify the complete AI of their computer-controlled servants in *Ragnarok Online*. Some developers even release the entire source code of their games, for example *Id Software* for both their *Quake* and *Doom* series. This is an important development for academics who want to use commercial games as testbeds for their research since the nature of their tasks usually requires profound changes to integrate custom applications. One sign of researchers using these new interfaces is that the number of projects using interactive computer games for AI research has grown tremendously (see Section 3.4 for numerous examples). And while some researchers used computer games as testbeds already a decade ago, they usually either developed their own games as test environments or used simple older games. Today more and more researchers apply machine learning techniques in recent commercial computer games.

Among the most popular commercial games, especially in terms of long-term gameplay, are the games of the *Civilization* series. These turn-based strategy games achieve their high replay value not through advanced adaptable AI techniques but through a high level of complexity in the later stages of the game. The early stages, however, are mostly deterministic and therefore not very challenging. This point will be elaborated on in Section 4.4. These characteristics as well as the large number of tasks involved in playing the game make Civilization games an ideal testbed for AI research. It is possible to replace one of the large number of tasks by a machine learning agent, thus making the AI less predictable and improving the overall gameplay experience.

This is one of the main incentives for integrating machine learning techniques into video games: the potential the techniques have to make these games more interesting in the long run through the creation of dynamic, human-like behaviour. The technique that probably resembles the human learning process most closely is Reinforcement Learning. Reinforcement learning in this thesis is used to do exactly what is described above: replace a deterministic task, thus making the game AI more adaptable.

There are three major objectives that this thesis tries to achieve. The first aim is the integration of machine learning into a professionally developed, complex computer game. The aim is not to create a machine learning agent that controls every aspect of the whole game. Since recent games are very intricate environments and Civilization games are especially renowned for their complexity, this would not be possible. However, an important task in the game will completely be taken over by reinforcement learning, namely the task of selecting the

best position for a player's cities. In order to perform this task efficiently, the reinforcement learning agent should show a clear tendency to improve.

The next objective of this thesis is to show that machine learning techniques can on the one hand provide a more versatile, adaptive computer game AI than standard techniques while on the other hand performing at least as well as the standard AI. Therefore the performance of the reinforcement learning agent will be compared to that of the current existing AI after reinforcement learning has successfully been integrated into the game. The results of this thesis in terms of these first two objectives have been published in (Wender and Watson, 2008) (see Appendix A). The second part of the empirical evaluation of this thesis further improves and extends the findings in the paper.

A third objective is to optimise the applied technique. Reinforcement learning includes a wide range of different algorithms that have different characteristics. Since some algorithms are better suited for certain tasks than others, it is important to evaluate their performance in the task of city site selection and select the algorithm that is suited the best for the task. The reinforcement learning algorithms contain parameters that can be used to put emphasis on different aspects of the reinforcement learning process. These parameters will be evaluated in order to find an optimal setting.

This chapter illustrated the motivation behind the topic of this thesis, the integration of reinforcement learning into a commercial computer game. It lists objectives of the thesis and their order of priority.
The following chapter will give an overview of the background of this thesis. It includes sections that elaborate on the historical background of machine learning in games and on the evolution of computer game AI. Reinforcement learning and its origins are explained in detail and research that is related to the topic of this thesis is mentioned. Finally, the reinforcement learning algorithms that will be used in this thesis are presented in detail and the general layout of a reinforcement learning environment as a Markov decision process is described.

Chapter 2

Thesis Outline

This chapter gives overview on the outline of this thesis. The different chapters that this thesis consists of are briefly mentioned and their content is explained.

The following chapter will give an overview on several topics that form the background of this thesis. On the one hand these are computer games in general and computer game AI in particular. On the other hand this is reinforcement learning, a set of unsupervised machine learning techniques. The 'Background' chapter contains information both on the historical background and development of these areas and on the research related to this thesis that has been done in these areas. Finally, the reinforcement learning algorithms that will be used throughout this thesis are explained in detail.

The chapter 'Testbed' explains the considerations behind the choice of Civilization IV as the testbed for this thesis. It then goes on to elaborate on the game mechanics in Civilization IV with focus on the existing game AI. Finally, the task of city site selection, that is taken over by the reinforcement learner is further illustrated and the present procedure behind this task is explained in detail.

The chapter 'Design and Implementation' contains information on how reinforcement learning is integrated into Civilization IV. At first the choices that are made when designing the reinforcement learning model are listed. The possible policies that can be used by the reinforcement learning algorithms are explained in detail and the deliberations behind choosing the ϵ-greedy policy are clarified. Then the integration of the algorithms into the actual game is presented. Finally, several improvements are listed that were made in order to speed up the convergence of the algorithms towards an optimal policy.

The subsequent chapter presents the empirical evaluation of the reinforcement learning algorithms that have been integrated in Civilization IV. It starts by elaborating on the reasoning behind the selection of the game settings and of the algorithm parameters. These settings are then tested in experimental runs and further adapted until the algorithms

prove that they are capable of developing efficient policies for selecting city sites. Then the algorithms' performance is compared compared with each others performance as well as with taht of the standard game AI. Through additional changes of the algorithm parameters the performance is further improved.

The chapter 'Discussion and Future Work' analyses the results of the experiments in the preceding chapter. The performance of the single algorithms is explained in detail. Furthermore possible extensions are listed and directions for future research are presented.

The conclusion sums up the results of this thesis. Furthermore, the overall contributions are mentioned and the initial objectives are compared with the actual results.

Chapter 3

Background

This chapter gives a brief overview of the background of machine learning in games as well as providing the background knowledge on the algorithms and techniques which are used throughout this thesis.

Its first part is a short summary of the development of artificial intelligence (AI) in classic board games. This section draws on surveys by Schaeffer and Fuernkranz (Schaeffer, 2000)(Fuernkranz, 2001) and summarises the research done for some of the most influential 'classic' games for AI research. The second part is about the evolution of AI in commercial games. It is mainly based on Tozour (2002) and also briefly touches on the differences between academic research and commercial applications. Finally the algorithms that are used in later parts of this thesis are explained in detail.

3.1 Machine Learning in Games

Arthur L. Samuel was one of the first people to pursue the application of machine learning to game AI. In 1947 he came up with the idea of creating a program that could play checkers. The two main papers describing his research, that lasted over the next three decades, are landmark papers for AI in general and for AI learning in particular. They introduce several techniques and ideas for AI learning which are still in use today in one way or another (Samuel, 1959) (Samuel, 1967). Samuel's program was based on the principle of learning the game without previously defined moves. This is in contrast to what became the common approach not only for checkers but also for chess and most other classical games where the computer programs have a certain amount of predefined game states in which they search for a solution.

In the late 1970s Samuel's program was defeated by a checkers program which was developed at Duke University (Truscott, 1978). This victory led to the assumption that the new program could match just about any human player. The assumption was indirectly proven wrong with the development of one of the most advanced checkers programs of the early 90s, *Chinook*, at the University of Alberta. Chinook is based on the now common principles of an opening book, an endgame database, previous knowledge, and extensive search through possible moves

(Schaeffer et al., 1992). Its evaluation function is not based on learning but has been tuned manually. Chinook won the 'world man-machine championship' in 1994 against the world champion checkers player Marion Tinsley and did not lose any game in the following years before retiring in 1997. In 2007 the developers stated that they computed the weak solution for checkers and thus Chinook cannot lose anymore (Schaeffer, 2007).

The history of computer AI playing checkers shows that even though Samuel started with a machine learning approach, research soon switched to optimising brute force searching programs which went on to become more effective and eventually unbeatable for human players. Despite this, there are also approaches that use machine learning algorithms to play checkers. Chellapilla and Fogel, for example, created a checkers program in the late 1990s, which did not use expert domain knowledge but learned playing the game through co-evolution (Chellapilla and Fogel, 1999).

One of the most researched areas in computer AI is the game of chess. Among the first papers to be published on the topic of computer chess was a paper by Shannon (1950). The author split algorithms applied to this problem into two types A and B. While both types are based on searching for possible moves, Type A algorithms do this by brute-force while Type B includes selective search. Type B algorithms search in a way that is similar to how human chess players tend to think. Subsequently Type A algorithms gained popularity due to their being easier to implement and to debug. Brute-force programs have managed to beat human grand masters for quite some time now; most well known is the series of duels between then world champion Gary Kasparov and chess programs made by IBMs Deep Blue team (*ChipTest*, *Deep Thought* and the famous *Deep Blue*) (Campbell et al., 2002).

In a way similar to checkers, considerable research has been done in the area of machine learning for chess. Probably the area most thoroughly studied is the induction of chess, that is the classification into *won/not won* from a given endgame. Quinlan (1983), for example, used the decision tree learning algorithm ID3 to acquire recognition rules. Muggleton (1990) applied DUCE, a machine learning algorithm that suggests high-level concepts to the user. The suggestions are based on recognized patterns in the rule database and the technique reduces the complexity of the rule base and generates concepts that are meaningful for domain-experts. Another machine learning technique which gained particular interest during the late 1980s is explanation-based learning (Mitchell et al., 1986). This approach was based on the assumption, that the domain theory can be utilised to find an explanation for a certain example. This example then can be generalized.

During the mid-90s a related approach called case-based reasoning (CBR) (Kolodner, 1992) was developed and applied to computer chess. Case-based reasoning is a technique which is quite similar to explanation-based learning and in principle 'Learning by Analogy' (Campbell et al., 2002). In order to solve a problem a CBR system looks for previously encountered

similar problems in its case-base. It then tries to adjust the solution to these known problems to fit the current one. The problem-solution pair it acquires in this way is then used to extend the existing case-base. MAPLE (Spohrer, 1985) is an early system that is based on CBR and learns whenever it makes a fatal mistake, that is when it reaches a point where all moves lead to a loss. CASTLE (Krulwich, 1993) is a modular (threat detection module and counterplanning module) system that also learns through mistakes. Whenever one of its components fails, it performs a self-diagnosis to find out which module failed and then uses explanation-based learning to extend its case-base by the missing case. Both CASTLE and MAPLE rely heavily on case-based planning (Hammond, 1989) which itself is based on explanation-based learning.

Another possibility to apply machine learning to computer chess programs is the design of the evaluation function. The evaluation function is the method that evaluates how good a certain move or a certain sequence of moves is. The tuning of this function has actually become the most promising direction for the application of machine learning to computer chess (Campbell et al., 2002). Tunstall-Pedoe (1991) used a genetic algorithm (GA) to optimize the evaluation function. The fitness of a certain parameter was determined by comparing the result of the function with the moves of a grandmaster. van Tiggelen and van den Herik (1991) came to the conclusion that genetic algorithms are too inefficient for use in a middle-game application and therefore used an artificial neural network (ANN or NN) instead. According to the authors this resulted in a more efficient and at the same time more accurate program. Schmidt (1994) was not satisfied with the results he got when he used a neural network and used temporal difference learning (TD learning) instead. The idea of TD learning had already been developed by Samuel (1959) for his checkers player but not really been in use again until Tesauro (1992) achieved amazing results with this in his backgammon program. In chess it is quite hard to evaluate effects of decisions made in the middle-game since the game is only decided at the end. TD learning addresses this problem by trying to minimise the differences between successive position evaluations. This means, for example, that if the program discovers after a series of evaluations that the assumed outcome is wrong, a previous assumption must have been wrong and the weights of the function have to be changed accordingly.

While checkers and chess have received tremendous attention and research, there are lots of other games for which computer players have been researched using plenty of different techniques. The aforementioned Tesauro created a backgammon player *TD-Gammon* (Tesauro, 1992) capable of beating human players. TD-Gammon is based on neural networks which are trained using TD learning. *Bill* is an Othello program written by Kai-Fu Lee and Sanjoy Mahajan that was among the best during the early 1990s. Besides using deep search and

extensive domain knowledge, it uses Bayesian learning in its evaluation function (Lee and Mahajan, 1990).

The game of poker offers several attributes that make it very interesting for AI research: it offers incomplete knowledge due to the hidden cards, it is played by several agents, and it uses concepts such as agent modeling and deception. Therefore, a lot of interesting research in the area of machine learning for poker has been produced. *Loki*, a program developed at the University of Alberta, uses explicit learning by observing its opponents and constructing models of these opponents. It then adapts to the play of these opponents (Billings et al., 1999). Korb and Nicholson (1999) produced a poker program that is based on Bayesian networks. The poker program of Dahl (2001) uses reinforcement learning (Russell and Norvig, 2003) to play the poker variation 'Texas Hold'em'. Rubin and Watson (2007) use case-based reasoning for their poker program *CASPER* which plays Texas Hold'em evenly against strong, adaptive competition. Other games that are popular with AI researchers include Go (Mueller, 2000), which has an even larger search space than chess, Scrabble (Sheppard, 2002) and Nine-Men-Morris (Gasser, 1996).

3.2 Evolution of Computer Game AI

Since the mid 1970s, computer games have been developed that allow a single player to compete with the program itself. Seminal games like *Pac-Man*, *Space Invaders* or *Donkey Kong* used, mostly due to restrictions of available resources, very simple techniques; these include finite-state machines, decision trees and production rule systems together with some random decision-making to add less predictable behavior. But while processing power increased in the following decade and games grew more complex and better-looking, the AI techniques remained by and large the same. Only in the 1990s were more complex techniques used. One reason for this was the success of strategy games such as MicroProse's *Civilization* or Blizzard's *WarCraft II* , since these games require AI as part of the central playing experience. Additionally, strategy games require a range of different AI techniques for unit-level AI as well as for overall strategic and tactical AI. First Person Shooters (FPS) are another game genre which led to the introduction of more complex AI into commercial games. While there have been games which belong to this genre since the early 1990s, they mostly tried to challenge the player by the sheer number of opponents or the amount of firepower he was facing. Significant progress was made in Valve's *Half-Life* which was praised for its tactical AI and Epic Games' *Unreal Tournament* which included bots that showed tactical behavior and scalability. One computer games genre which is practically based on AI is that of sim games/artificial life (A-Life) games.

Maxis' *SimCity* was one of the first games in this genre. Especially noteworthy for the complexity of its agents is *The Sims* which uses fuzzy state machines and A-Life technologies. The games of the *Creatures* series are basically A-Life simulators which make use of a wide range of AI techniques to simulate the evolution of the 'Norns' that populate the game. More recently Lionhead Studio's *Black&White* games have been built around a complex reinforcement learning approach that allows the player to train a 'creature'. These games show one of the most advanced game AIs to date.

While these games do use very advanced AI techniques, the most common technique still remains the simpler and thus easier to create and debug rule-based systems. This gap between academic research and the computer gaming industry has not gone unnoticed. Laird and van Lent (2001) state that while there is significant room for academic research in computer games, the computer games industry tends to go in its own direction. They then conclude that it is up to academic researchers to close this gap by using computer games as testbeds to develop AI methodologies which then can be used in commercial games. According to Nareyek, the academic community has so far failed to achieve this (Nareyek, 2004) (Nareyek, 2007). He states that apart from the usage of a few common techniques, in particular the A* algorithm for path finding, usually no academic research is used in commercial products. He also claims that common academic research goes into a direction that is all but uninteresting to the computer games industry.

On the other hand there have been quite a few attempts to bridge this gap between academia and the industry. Champandard (2003) uses the open-source framework FEAR (Flexible Embodied Animat 'Rchitecture) together with the commercial First Person Shooter *Quake 2* to evaluate several AI techniques proposed by the academic community. He succeeds at implementing these various techniques for different tasks of a non-player character (NPC), a so-called bot. Gold (2005) does research using a commercial game engine as testbed to develop a prototype game. He reaches the conclusion that it is indeed possible to mix commercial game development and research. Miikkulainen et al. (2006) argue that the AI techniques which are usually applied in modern games are in fact not appropriate at all for the purposes they are used for. On the other hand machine learning techniques which are not used at all in commercial products such as neuroevolution (neural networks in combination with genetic algorithms) are particularly well suited for computer games.

3.3 Origins of Reinforcement Learning

This section points out the early origins of reinforcement learning and is based on Sutton's and Barto's survey on reinforcement learning (Sutton and Barto, 1998). The specific reinforcement learning algorithms that are used in this thesis are described in Section 3.5.

The modern field of reinforcement learning was introduced in the late 1980s and evolved mainly from two different branches of research: optimal control, and learning by trial and error. A third branch which also played into the development to a smaller degree is TD learning. A short explanation for all three areas is given below.

'Optimal control' is a term which is used to describe the search for a controller that minimizes a specific variable in a dynamic system over time. One of the most prominent approaches to this problem was developed by Richard Bellman in the 1950s, the so-called dynamic programming (Bellman, 1957b). This technique is still used today to solve reinforcement learning problems. Bellman (1957a) also developed the notion of the Markovian decision process (MDP) which is the discrete stochastic version of the optimal control problem. Both dynamic programming and MDPs form a vital part of what is today known as reinforcement learning.

The other big branch of reinforcement learning, learning by trial and error, is based on the 'Law of Effect' in cognitive sciences: An action followed by a positive reward is remembered in this context and thus more likely to be performed again in the same situation. An action followed by a negative reward is remembered in a negative context and will be avoided (Thorndike, 1911). The 'Law of Effect' is selectional, i.e. it tries several different options and choses based on consequences. The law is also associative: the alternative found will be associated with a particular situation. One of the first attempts at teaching a computer program through trial and error learning was made by Farley and Clark (1954).

The third, quite recent branch which comes into play in modern reinforcement learning is TD learning. The first application of a form of TD learning, by analyzing the difference of a certain attribute between time-steps, was done by Samuel (1959). Witten (1977) was the first to integrate optimal control and trial and error learning. He also made a significant contribution to TD learning, an area which had received little attention since its first developments in the late 1950s.

The final step to create modern day reinforcement learning was the development of Q-learning by Watkins (1989). His work integrated all three previously described branches and extended the field of reinforcement learning in general. One of the first successful and noteworthy applications of these techniques was performed by Tesauro (1992), who brought further attention to the emerging field of reinforcement learning with his backgammon player TD-Gammon.

3.4 Related Work

This section is concerned with previous research on reinforcement learning in computer games. It is divided into subsections for research into different algorithms as well as their applications in computer game AI. Furthermore there is an explanation of other reinforcement learning

techniques such as motivated reinforcement learning (MRL) which address certain problem areas of reinforcement learning. The specific algorithms that are applied in this thesis are described in detail in Section 3.5. Also the previous usage of Civilization games as testbed for AI research is described.

3.4.1 Q-Learning

An important breakthrough in the history of reinforcement learning was the development of Q-learning (Watkins, 1989). Q-learning integrates, as stated in Section 3.3, different branches of previous research such as dynamic programming and trial-and-error learning into reinforcement learning.

Andrade et al. (2005) use a Q-learning algorithm to create an adaptive agent for a fighting game *Knock'em*. The agent is initially trained offline to be able to adapt quickly in an online environment. During the game it adjusts its play to the other players level of skill. The game AI thus always presents a challenge but remains beatable regardless of the players level of skill.

Smith et al. (2007) the authors introduce RETALIATE (for: Reinforced Tactic Learning in Agent-Team Environments), an online Q-learning technique that creates strategies for teams of computer agents in the commercial FPS game *Unreal Tournament*. The bots controlled by RETALIATE were able to adapt and devise a winning strategy against the inbuilt game AI in most cases in just one game.

Whiteson and Stone (2006) use a combination of Q-learning and the NEAT (NeuroEvolution of Augmented Topologies) algorithm to perform on-line evolutionary function approximation. They test this method with several standard reinforcement learning benchmarks and conclude that evolutionary function approximation can significantly improve standard temporal difference learning.

3.4.2 Sarsa

Graepel et al. (2004) use the Sarsa algorithm to compute strategies in the commercial fighting game *Tao Feng* to create a challenging computer opponent. Q-learning could not be used because of the lack of information about possible actions in any given state. Sarsa does not require this information. Stone et al. (2005) use the game *RoboCup Soccer Keepaway* as a testbed. In the game a number of agents (the keepers) try to keep the ball in their possession within a limited area while the opponents (the takers) try to capture the ball. The episodic semi-Markov decision process (SMDP) version of a Sarsa(λ) algorithm is used to compute the best strategy for the adaptive keepers against a set of predefined static takers. As a result

of empirical studies, the time it takes the takers to acquire the ball from the keepers rises constantly with number of episodes that were played.

McPartland and Gallagher (2008b) use a tabular Sarsa(λ) algorithm to control a bot in a first-person-shooter. Different techniques are combined with reinforcement learning (state machine, rule based and hierarchical reinforcement learning) to control the bot. Hierarchical and rule based reinforcement learning prove to be the most efficient combinations (McPartland and Gallagher, 2008a).

3.4.3 Reinforcement Learning and Case-Based Reasoning

Several publications explore a combination of case-based reasoning (CBR) and reinforcement learning for computer game AI.
Sharma et al. (2007) use a multi-layered agent to create strategies and tactics for an agent that competes in the real-time strategy (RTS) game MadRTS. One of the main aims of their research is to use transfer learning in an RTS environment. In order to achieve this, a hybrid case-based reasoning and reinforcement learning algorithm is used. CARL (CAse-Based Reinforcement Learner) consists of multiple layers, each of these layers being made up of three modules: a *planner*, a *controller* and a *learner*. The planner selects from the actions that are available in the state that the layer is currently in. The controller acts as an interface for perceptions/actions to lower layers and the learner modifies the data/features used by the planner.

For their experiments the authors use three layers, each consisting of the three modules described above. The top layer is hand coded to select the overall strategy and the bottom layer takes the orders from above and translates them into MadRTS specific orders. The middle layer contains a hybrid case-based reasoner and reinforcement learner. The hybrid CBR-RL Layer uses CBR as an instance-based function approximator to retrieve matching cases from the case-base while a TD reinforcement learning algorithm is used to revise the cases according to how the selected case performs. The performed experiments show that this architecture works well for the given task of controlling combat in the selected RTS game. Furthermore, transferring the gained knowledge to a slightly more complex task works as well: the agent adapts much faster if it uses previous cases than if learns from scratch.

Auslander et al. (2008) use the reinforcement learning algorithm RETALIATE developed by Smith et al. (2007). RETALIATE uses online Q-learning to create strategies for teams of agents in the FPS Unreal Tournament. As an extension the authors introduce techniques from CBR for the agent to adapt faster to a change in the strategy of its opponent. The resulting technique, CBRetaliate, tries to obtain a better matching case whenever the collected input reading shows that the opponent is outperforming it. The Q-table of the previously computed policy is part of the stored/retrieved cases among other recorded data. As a result

of the extension, the CBRetaliate agent is shown to significantly outperform the RETALIATE agent when it comes to sudden changes in the opponent's strategy.

Aha and Molineaux (2008) introduce the Continuous Action and State Space Learner (CASSL) which uses CBR and reinforcement learning in the continuous action/state space of the RTS game MadRTS. The integration of CBR and reinforcement learning has the additional benefit of showing causal relations between different states, something which CBR alone would not be able to do. The ensuing experiments show that this approach outperforms the same algorithm which descretizes the state space. CASSL maintains two case-bases, one for transitions, i.e. applying actions in certain states and another case-base for values, i.e. estimates of the value of certain states using the current policy. Each of these case-bases supports a cycle of case retrieval, reuse, revision and retention. At the beginning both case-bases are initialised to the empty set. By applying CASSL to a sequence of gameplay episodes the case-bases are subsequently filled with new cases and revises. A nearest neighbor metric is used to retrieve similar cases.

3.4.4 Neuroevolution

Neuroevolution algorithms use neural networks which are trained through genetic algorithms.There is a distinction between neuroevolution algorithms that only calculate the weights for the neural networks and algorithms that change the entire topology of the network.

Stanley et al. (2005) use a modification of the NEAT (NeuroEvolution of Augmented Topologies) algorithm (Stanley and Miikkulainen, 2002), rtNEAT (real-time Neuroevolution of Augmented Topologies) to build the Neuroevolving Robotic Operatives (NERO) game. In this game teams of virtual robots are trained using the inbuilt machine learning techniques to compete against other teams. A major contribution of rtNEAT is the fact that an evolutionary algorithm, which is usually designed to run off-line, is trained during the game. In the NERO game the player acts as an instructor for his team of robots which start the game without any skills at all. The player can only influence the environment and thus force the robots to perform certain actions.
Karpov et al. (2006) use NEAT to generate strategies for bots that play the FPS Unreal Tournament. The authors perform their experiments in the test environment TIELT (Testbed for Integrating and Evaluating Learning Techniques) (Aha and Molineaux, 2004) and also include an evaluation on the time and effort required to integrate a commercial game and use it as testbed for AI research. As a test case the task of navigating through a level in Unreal Tournament was used.

3.4.5 Dynamic Scripting

Spronck et al. (2006) developed an online learning technique they called 'dynamic scripting' . Dynamic scripting enables the computer game AI to adapt itself to the player, either in level of difficulty or in style of play. A rule base is used to create scripts which define the performance of the AI. For each computer agent there exists a rule base of manually defined domain specific rules. Each of these rules has a weight assigned to it that determines the likelihood for this rule of being chosen into the script. The script defines the agent's behavior during the game. At the beginning of a new evaluation cycle such a script is composed for every single agent according to the weights using a softmax procedure. After the end of a cycle the weights of the rules involved are updated in the rule base. If the outcome was a success, the weights for rules involved in the successful script are incremented, otherwise the weights are decremented. The overall sum of weights is kept constant; an increment of one weight results in all other weights being decremented. Dynamic scripting is comparable to actor-critic reinforcement learning and Monte Carlo control. The authors use the commercial role-playing game (RPG) Neverwinter Nights as a testbed for their research. They compose a scenario where two teams of equally strong computer characters fight each other. The characters on the one side are controlled by different versions of ingame AI rules. The characters on the other side are controlled by dynamic scripting. The result showed that the opponent AI which was based on dynamic scripting managed to adapt quite quickly (20-50 cycles on average) depending on the strength of the inbuilt game AI.

Ponsen et al. (2006) improve the basic dynamic scripting by eliminating the shortcoming of manually designed rule bases. The authors introduce the Evolutionary State-based Tactics Generator (ESTG), which creates tactics fully automatically. An evolutionary algorithm is used to fill the initial rule base with domain specific knowledge. A test against selected manually created scripts shows that dynamic scripting with automatically generated rule bases is able to adapt itself to changing strategies. After a certain amount of adaption, even the most advanced manual script was beaten by dynamic scripting.

3.4.6 Motivated Reinforcement Learning

Motivated reinforcement learning (MRL) is reinforcement learning for which the reward signal is computed through an additional motivation function. The motivation function uses domain independent rules to calculate the motivation signal. Thus the development of the agent relies more on environment and experiences rather than domain specific knowledge. The motivational function relies heavily on research in cognitive sciences.

Merrick and Maher (2007) target the development of NPCs for open-ended virtual worlds that adapt to changes in their environment. By adapting to changes in their environment,

the NPCs provide an ongoing interesting experience for the players. The adaptable NPCs are different to NPCs of recent commercial MMOGs (Massively Multiplayer Online Games) which are controlled by static scripts and do not adapt to changes in the game world. The authors describe an experiment in the game *Second Life*. The experiment uses a task which is defined as a MDP. Virtual sheep are created and controlled by an MRL algorithm. Two experiments show that the sheep adapt to changes in their previous environment.

Merrick and Maher (2006) also use a similar approach to create agents which act as support characters for the player. It is shown through empirical evaluation how a virtual agent that is controlled by an MRL algorithm can act as a vendor for goods produced by the player. The virtual agent is unsupervised and adapts to changes in its environment following a behavioral policy that has been previously defined. Merrick (2007) sums up previous research on creating adaptive NPCs. Furthermore, two different models of MRL that are used to control NPCs are compared.

3.4.7 Civilization Games as Testbed for Academic Research

Civilization is the name of a series of turn-based strategy games. Several variants of Civilization have been used in academic research because of the broad spectrum of problems involved in the games as well as the multitude of versions available, quite a few of them with open source code.

Ulam et al. (2004) use the Civilization variant *FreeCiv*, an open source version of the commercial game *Civilization II*, to show the effectiveness of model-based reflection and self adaption. The comparably small but important task of managing the construction of defensive units and buildings in a city is used as a testbed. Houk (2004) goes further and introduces an agent that plays the complete early expansion phase of FreeCiv. The Lisp module which was created to do this controls city management, unit management, and exploration.

Souto (2007) and Gundevia (2006) describe the integration of the commercial Civilization game *Call To Power 2* (CTP2) with the test environment TIELT (Aha and Molineaux, 2004). Their overall aim is transfer learning, but the test environment can also be used as an integrated testbed for future research using CTP2. TIELT integrated with CTP2 is used by Sánchez-Ruiz et al. (2007) as a testbed for adaptive game AI. In order to communicate with the game an ontology for the domain is developed and case-based planning in combination with CBR is used to create an adaptive AI.

Sánchez-Pelegrín et al. (2005) document the development of an AI module for the open source Civilization clone *C-Evo*. This variant, which is closest related to *Civilization II*, allows for the development of different AI modules which can then compete against each other. The authors develop an AI module based on CBR that can perform simple combat

actions. The modular architecture of C-Evo is then used to compare this module with a non-learning version of itself.

3.5 Algorithms

This section which is based on (Sutton and Barto, 1998) describes the general algorithms and concepts which are used throughout this thesis in detail. First the notions of the *Markov property* and *Markov decision processes* are explained. These are important since they are prerequisites for the efficient use of reinforcement learning techniques. The idea behind the reinforcement learning technique *temporal difference learning* is described afterwards. Both TD algorithms that are applied throughout this thesis are based on this technique. These specific algorithms, *Q-learning* and *Sarsa*, are elaborated afterwards. After explaining the simple, *one-step* versions of these algorithms, their more complex versions which include *eligibility traces* are shown in detail.

The algorithms described in this section are not yet altered in any way to adjust them for the specific area of application in this thesis. These alterations are described in Chapter 5.

3.5.1 The Markov Property

The agent in a reinforcement learning framework makes its decisions based on the information it gets about the environment at any one time, the so called *state*. If this state signal contains all the information of present and past sensations it is said to have the *Markov Property*. This does not mean that single actions leading up to the present state have to be observable, only the important parts for the current state have to be present. Mathematically the complete probability distribution for the response of an environment at time $t+1$ to an action taken at time t looks as follows

$$Pr\left\{s_{t+1} = s^{'}, r_{t+1} = r | s_t, a_t\right\}.$$

If an environment has the Markov property this also means that given the current state and action, it is possible to predict the next state and reward. Through iteration this allows to predict all future states. Furthermore choosing a policy based on a Markov state is just as effective as choosing a policy when knowing the complete history until that state. In reinforcement learning Markov states are important since decision are only made based on the current state. Even if a reinforcement learning environment does not have the Markov property, the standard algorithms which assume this property can still be applied. However these algorithms will usually only be as effective as far as the state signal resembles a Markov state.

3.5.2 Markov Decision Processes

If a reinforcement learning task presents the Markov property it is said to be a *Markov decision process* (*MDP*). If the task has a finite number of states and actions it is called a *finite MDP*. A specific finite MDP is defined by a quadruple $(S, A, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$.

In this quadruple

- S represents the state space,
- A represents the action space,
- $\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1} = s' | s_t = s, a_t = a\right\}$ represents the transition probabilities, i.e. the probability of reaching state $s\prime$ from s after taking action a.
- $\mathcal{R}^a_{ss'} = Pr\left\{s_{t+1} = s' | s_t = s, a_t = a\right\}$ represents the expected reward, given a current state s, an action a and the next state $s\prime$.

The MDP is used to maximize a cumulative reward by deriving an optimal policy π according to the given environment.

3.5.3 Temporal-Difference Learning

Temporal-difference (TD) learning is a reinforcement learning method which combines ideas from dynamic programming with Monte Carlo methods (Sutton, 1988). Like Monte Carlo methods it samples the environment to observe the current state without the need of a complete model of this environment. Both methods update their estimated value V of a visited state st based on the return after visiting the state while following a policy π. Just as for dynamic programming the estimation is thus based on previous estimates, the so-called 'bootstrapping'. The pseudo code for the simple TD algorithm looks as in Algorithm 1.

Initialise $V(s)$ arbitrarily and π to the policy to be evaluated
for *(each episode)* **do**
  Initialise s
  repeat for each step of episode
    $a \leftarrow$action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
    $s \leftarrow s'$
  until *s is terminal*

Algorithm 1: A Simple TD Algorithm for Estimating V^π

3.5.4 The Q-Learning Algorithm

Q-learning is an off-policy TD algorithm, i.e. it calculates the values of policies not only based on experimental results but also based on estimates about values of hypothetical actions, i.e. actions which have not actually been tried.
The formula for simple *one-step Q-learning* is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]. \tag{3.1}$$

Since Q-learning works independent of the policy being followed, the learned action value Q function directly approximates the optimal action-value function Q^*. In contrast to simple TD it does not assign a value to states, but to state-action pairs. The only prerequisite it has to finding the optimal policy with probability 1 is just that all states are visited infinite times, which is a basic requirement for all reinforcement learning methods that are guaranteed to find the optimal behavior.

The procedural form of the Q-learning algorithm can be seen in Algorithm 2.

Initialise $Q(s, a)$ arbitrarily
for *(each episode)* **do**
  Initialise s
  repeat for each step of episode
    Choose a from s using the policy derived from Q
    Take action a, observe r, s'
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]$
    $s \leftarrow s'$
  until s *is terminal*

Algorithm 2: Pseudocode for One-Step Q-Learning

3.5.5 The Sarsa Algorithm

Sarsa is an on-policy TD learning algorithm very similar to Q-learning (Rummery and Niranjan, 1994). The main difference is that it is not necessarily the action with the biggest reward that is used for the next state but the action according to the same policy that led to the present state. Sarsa stands for **S**tate-**A**ction-**R**eward-**S**tate-**A**ction, more specifically the quintuple referred to here is $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. This implies that to compute the update for a Q-value the following information is needed:

- The current state s_t.
- The chosen action a_t according to a policy π.
- The reward r_{t+1} gained from executing this action.

- The resulting new state s_{t+1}.
- The next a_{t+1} action that is chosen the policy π.

Sarsa was developed as an alternative to the off-policy Q-learning which always chooses the action yielding the maximum reward. Sarsa allows for a more controlled trade-off between exploitation (taking the highest yielding action) and exploration (picking random/ unknown actions). The update function for Q-values is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \tag{3.2}$$

This function makes use of every element in the quintuple described above. In its general form Sarsa looks as shown in Algorithm 3.

Initialise $Q(s, a)$ arbitrarily
for *(each episode)* **do**
    Initialise s
    Choose a from s using the policy derived from Q
    repeat for each step of episode
        Take action a, observe r, s'
        Choose a' from s' using the policy derived from Q
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    until *s is terminal*

Algorithm 3: Pseudocode for One-Step Sarsa

3.5.6 Eligibility Traces

Eligibility traces are a basic mechanism of reinforcement learning that is used to assign temporal credit. This means that it is not only the value for the most recently visited state or state-action pair that is updated. Value for states or state-action pairs that have been visited within a limited time in the past are also updated. The technique can be combined with any TD technique and it speeds up the learning process. As future chapters will show, assigning temporal credit is important for the task of city site selection that was chosen for this thesis.

TD(λ) is a popular TD algorithm that uses eligibility traces and was developed by Sutton (1988). It was used by Tesauro (1992) for his famous backgammon agent which learned to play on the same level as human players. The λ in TD(λ) is the so-called *trace-decay* parameter, i.e. the parameter which determines how far rewards propagate back through a series of states/actions. This parameter is used to compute the eligibility trace which is for state s at time t

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t. \end{cases}$$

To guarantee convergence towards the optimal solution, the limits for *lambda* are $0 < \lambda < 1$. This is quite obvious since lambda has to decay in order to make future rewards less important than present rewards. The pseudocode for TD(λ) can be seen in Algorithm 4.

Initialise $V(s)$ arbitrarily and $e(s) = 0$ for all $s \in S$
for *(each episode)* **do**
  Initialise s
  repeat for each step of episode
    $a \leftarrow$action given by π for s
    Take action a; observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    forall s **do**
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  until s *is terminal*

Algorithm 4: Pseudocode for TD(λ)

For both Q-learning and Sarsa eligibility traces are not used to learn state values $V_t(s)$ but rather values for state-action pairs $Q_t(s, a)$, just as in the one-step versions of these algorithms. The pseudocode for the eligibility trace version of Sarsa(λ) can be seen in Algorithm 5.

There are two different popular methods that combine Q-learning and eligibility traces. They are called Peng's Q(λ) (Peng and Williams, 1994) and Watkins's Q(λ) (Watkins, 1989) after the people that first proposed them. Empirically, it has been shown that Peng's Q(λ)

```
Initialise Q(s, a) arbitrarily and e(s) = 0 for all s ∈ S
for (each episode) do
    Initialise s, a
    repeat for each step of episode
        a ←action given by π for s
        Take action a; observe reward, r, and next state, s'
        Choose a' from s' using policy derived from Q
        δ ← r + γQ(s', a') − Q(s, a)
        e(s, a) ← e(s, a) + 1
        forall s, a do
            Q(s, a) ← Q(s, a) + αδe(s, a)
            e(s, a) ← γλe(s, a)
        s ← s'; a ← a'
    until s is terminal
```

Algorithm 5: Pseudocode for Sarsa(λ)

usually performs better and nearly as good as Sarsa(λ) (Sutton and Barto, 1998). However Peng's Q(λ) is far more complex to implement than Watkins's Q(λ), has not yet been proven to converge to the optimal function Q^*, and is basically a hybrid between Watkins's Q(λ) and Sarsa(λ). Therefore in this thesis only Watkins's Q(λ) is used as Q-learning algorithm with eligibility traces. The pseudocode for Watkins's Q(λ) can be seen in the figure for Algorithm 6.

```
Initialise Q(s, a) arbitrarily and e(s) = 0 for all s, a
for (each episode) do
    Initialise s, a
    repeat for each step of episode
        Take action a; observe reward, r, and next state, s'
        Choose a' from s' using policy derived from Q
        a* ← argmax_b Q(s', b) (if a' then a* ← a')
        δ ← r + γQ(s', a*) − Q(s, a)
        e(s, a) ← e(s, a) + 1
        forall s, a do
            Q(s, a) ← Q(s, a) + αδe(s, a)
            If a' = a*, then e(s, a) ← γλe(s, a)
            else e(s, a) ← 0
        s ← s'; a ← a'
    until s is terminal
```

Algorithm 6: Pseudocode for Watkins's Q(λ)

This chapter gave an overview of the historical background of machine learning in games in general and machine learning in computer games in particular. After illustrating the historical background of reinforcement learning, this chapter also looked at the use of reinforcement learning in computer games. It illustrated the related work for the chosen testbed, the com-

puter game Civilization IV. Finally, the specific reinforcement learning algorithms and their applications in computer games were explained.
The next chapter deals with the computer game that will serve as a testbed in this thesis. The selection of the testbed from a range of candidates will be elaborated on first. Then the game mechanics as well as the specific task for which reinforcement learning is used will be outlined.

Chapter 4

Testbed

This chapter gives an overview of the testbed that is used in this thesis, namely the computer game *Civilization IV*. At first a comparison of the four Civilization games that were considered as testbeds is given. The results of this comparison are the motive for using Civilization IV as a testbed. After selecting the testbed, a general introduction to the games of the Civilization series and their game mechanics is given. Then the way the existing Civilization IV AI works is illustrated before the task the reinforcement learning algorithms will perform is explained in detail.

4.1 Selection Criteria

As suggested in the previous section on related work, a general choice was made to use a game of the Civilization series as a testbed. The next section describes the selection of the Civilization game. The criteria according to which the game is chosen, are listed below.

- Code Availability: All important parts of the code have to be open source. These parts must be easily accessible. This is an exclusion criterion.
- Ease of Use: This criterion includes several subitems.
  - Quality of the Documentation: Even the smaller games contain an extensive code base. It is therefore important that information exists on how to use the code.
  - Quality of the Code: The quality of the code base is very important. The code should be well structured and be developed according to standards that make working in one part of the code similar to working in any other part.
  - Programming Language: The programming language should be as capable as possible, i.e. contain high-level functionality. While low-level programming languages offer more control over memory management and are usually faster, not speed but functionality is the main focus of this thesis. A potent development environment should exist for the programming language the code base is written in.

- Modularity: Only a small part of the existing AI will be replaced. It is important that this is easily possible and the rest of the existing AI remains completely functional.
- Quality of the Existing AI: Since only a small part of the overall AI will be replaced by reinforcement learning, the rest of the standard AI will still be used in experiments. It is therefore important that the existing AI is very capable and does not contain errors. These errors could make experimental results worthless.
- Age and Popularity: These two criteria are signs of the effort that went into the development of the game and of ongoing support. More recent programs usually underwent a more complex development and thus is also more likely to have higher quality code. If a commercial or open source game enjoys a high popularity the developers are more likely to release patches and bug fixes for it than for a game with low popularity. Since the results of young age and huge popularity are already covered by previous criteria, age and popularity are only used as indicators for these previous criteria.
- Reusable Results: Ideally there exists code or documentation from similar projects that can be reused.

4.2 Testbed Selection

Section 3.4.7 expands on related work that has been done on Civilization games. It lists several of these games of the Civilization series. All three of the games mentioned in that section have been used successfully as testbeds for academic research. A choice therefore had to be made between these three versions of Civilization and the most recent commercial version of the original game, Civilization IV. The criteria according to which this choice is made have been listed in the previous section. Table 4.1 lists some basic characteristics of the four games.

	Release	**Open Source (OS)/ Commercial**	**Programming Language**	**Target Group**
C-Evo	1996	Open Source	Delphi/Pascal	Very Advanced Players
FreeCiv	1999	Open Source	C	Medium to Advanced Players
Call to Power 2	2000	Originally Commercial Completely OS Now	C++	All Levels of Experience
Civilization IV	2005	Originally Commercial Partly OS Now	C++/Python	All Levels of Experience

Table 4.1: Comparison of Civilization Games

A first choice can be made between the inherently open source versions C-Evo and FreeCiv and the commercial versions CTP2 and Civilization IV which offer open source code but were not designed to be released to the public. The two open source games offer complete access to commented source code, they are smaller in scale and less complex than their commercial counterparts. Since they are easier to access and have been available for a longer time, they have been the preferred testbeds for research. Especially C-Evo offers great support for AI because it has its own interface for complete AI modules (Gerlach, 2008).

However, the code base of the two community projects is of a very different quality than the professionally developed commercial games. C-Evo's core has been developed by one person only. Consequently it has several shortcomings such as undocumented hidden dependencies and missing documentation for certain parts of the code. FreeCiv, on the other hand, was entirely developed as a community project. As a consequence, there are various inconsistencies in its code base. Its AI in particular has been poorly documented and developed according to the developers (Freeciv Project, 2008). For this thesis only a small part of the AI was to be replaced by reinforcement learning while the standard game AI controls all remaining tasks. The bad quality of its AI thus makes FreeCiv an unfavorable testbed.

A further disadvantage of the open source games is their programming languages. Since the open source games are older than the commercial ones, they have been developed in Delphi and C. The commercial games on the other hand both were developed using C++. The most convincing argument in favor of CTP2 and Civilization IV is the fact that they are commercial games: They have been developed by professional programmers, artists and designers with a massive budget, they went through a software life cycle that included thorough testing and reviewing and they have a huge user base. These two games represent the testbeds that John Laird referred to in his keynote address at the AAAI 2000 conference (Laird and van Lent, 2001) when he was talking about computer games being ideal testbeds for academic research.

The choice between CTP2 and Civilization IV is more complicated than between commercial and open source games. Both CTP2 and Civilization IV offer easily accessible interfaces for sophisticated user-developed modifications ('mods') besides their published source code. Furthermore, both games originally lacked documentation because the companies deleted all comments from the source code prior to releasing it as open source. They did so in order to avoid legal problems. All existing documentation has been added by the communities of the respective games.
CTP2 has the advantage of offering complete access to the source, in contrast to Civilization IV, which offers access only to the core. Thanks to prior work done by Souto (2007) and Gundevia (2006), it would be an option to use TIELT in conjunction with CTP2 instead

of working directly in the code. However, this integration with TIELT would offer only a limited set of commands. Since these limited commands do not allow to completely control the task of city site selection described in more detail in Section 4.5, it was decided not to pursue this approach. Another setback of the use of CTP2 is the amount of code lines that can be modified. If CTP2 is used, more than 2000 000 lines of code (LOC) are accessible and subject to possible change. Another problem of CTP2 is that the developer *Activision* ceased to support the game very soon after its initial release. Most of the large number of errors that remained in the code at that time have since been fixed by the community. This meant, however, that the development of other features like AI was neglected.

Civilization IV, on the other hand, is still in active development by the original developer *Firaxis*. Parts of the code developed by the community have actually been included in the commercial product (BetterAI Project, 2007). While the published source code for the core of Civilization IV contains about 100000 LOC with the AI code consisting of roughly 25000 LOC, a quick analysis of the code showed that the chosen task can actually be performed by replacing a single method. The information that is important for the task is contained in three to four classes. A further advantage of Civilization VI over CTP2 is its age. Civilization IV is about two computer game generations younger than CTP2 (Civilization IV's predecessor Civilization III was released one year after CTP2). The game thus contains more advanced features. Since the AI in Civilization IV is strong, the success of a competing agent using reinforcement learning for parts of its decision making process would be very impressive. For the reasons stated above, Civilization IV was chosen as testbed.

4.3 The Civilization Game

Civilization is the name of a series of turn-based strategy games in which the player has to lead a civilization of his choice from the beginnings (BC) to the present day. The games involve building and managing cities and armies, advancing the own empire through research and expansion as well as interacting with other, computer-controlled civilizations through means of diplomacy or war in a turn-based environment. The popularity of the original game has lead to a multitude of incarnations of the game, both commercial and open source. The variant of the Civilization game which will be used as a testbed in this thesis is *Civilization IV* (Firaxis Games, 2005). Civilization IV was developed by *Firaxis Games* and is the latest version in the commercial series of the original game. Large parts of its code base, including the part which controls the AI, have been released as open source. These parts can be modified to allow machine learning algorithms to take over tasks of the existing AI.

4.4 Existing Civilization IV Game AI

Since Civilization IV is a commercial game that aspires to provide enjoyable gameplay for all levels of skill, the existing computer AI is already quite sophisticated. It has furthermore been improved by the Better AI project (BetterAI Project, 2007). This is a community project that modified the code base of the existing game to improve overall AI performance. The performance of the computer AI players is therefore comparable to that of strong human players with equal handicaps and can, through a number of modifiers, be scaled up or down to several levels of difficulty. However, this scaling does not make the AI itself stronger or weaker. It merely influences the conditions under which the AI operates (the game rules) and makes them more or less favourable.

One of the AI's major shortcomings (from a machine learning point of view) is that the computer AI is almost completely deterministic: Only a very limited set of decisions and events in the game are controlled by randomness. The situations in which the AI's decisions are controlled by randomness are limited to combat situations. This shortcoming can however be used as an advantage when it comes to creating environments for empirical evaluation of modifications. The computer can provide a challenging opponent in these experiments but is still more or less completely normalized, i.e. deprived of randomness. Furthermore, an improvement of the Civilization IV AI would show that research can be used to create a bigger challenge for the player and thus offer a more enjoyable playing experience. Consequently, the improvement of the existing computer AI through reinforcement learning would demonstrate that scientific research can be used to improve commercial computer games.

4.5 City Placement Task

The most important asset in a game of Civilization IV are the cities. The three major resources a city produces are food (used for growth and upkeep of a city), commerce (used among others for research and income) and production (used to produce units and buildings). Furthermore, special bonuses that grant additional basic resources or other benefits like accelerated building speed can be gained. The playing field in Civilization IV is partitioned into 'plots' with each plot producing a certain amount of the resources mentioned above. A city has access only to the resources of a plot which is in a fixed shape of 21 plots surrounding the city (Figure 4.1).

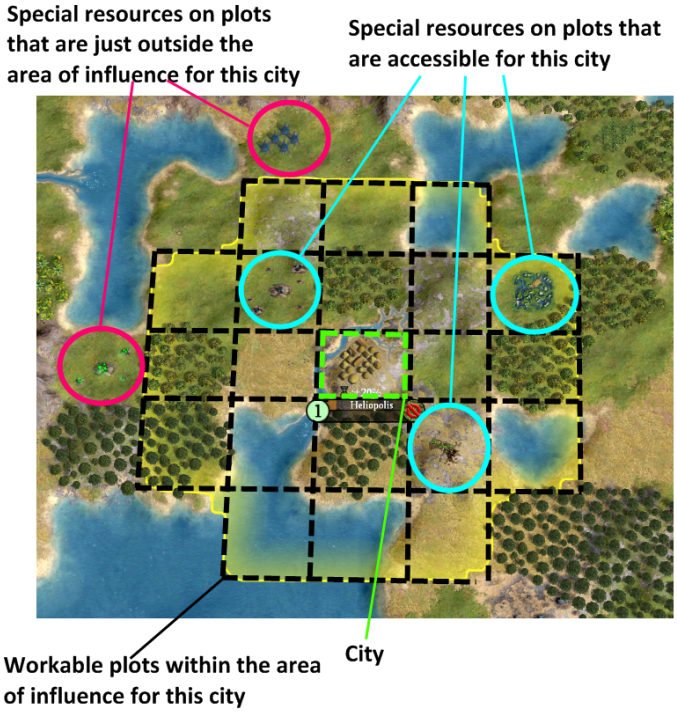


Figure 4.1: Civilization IV: Workable City Radius

In addition, the borders of an empire and thus the area of influence of its player are defined by the connected borders of the cities of that empire. Therefore the placement of cities is a crucial decision and influences the outcome of a game to a large degree. This is one of the main reasons why we chose to apply reinforcement learning to the city site selection task. Furthermore, city placement can be evaluated at a relatively early stage of the game since according to the game mechanics, the founding of cities (and thus the creation of the empires) takes place during the initial phase of the game. Depending on map size and number of players, it takes between one fourth and one half of the complete game time to settle a whole map. Once a whole map has been settled, the focus shifts from exploration and settlement to consolidation and combat. This allows to focus the experiments on the first third of the game, the settling phase. The first third of the game is also the fastest part of it because of the low number of active units in this phase of the game. It thus allows for a high number of runs in a relatively small period of time. Figure 4.2 shows an overview of an empire that consists of two small cities. The cities as well as the border of the whole empire are marked.

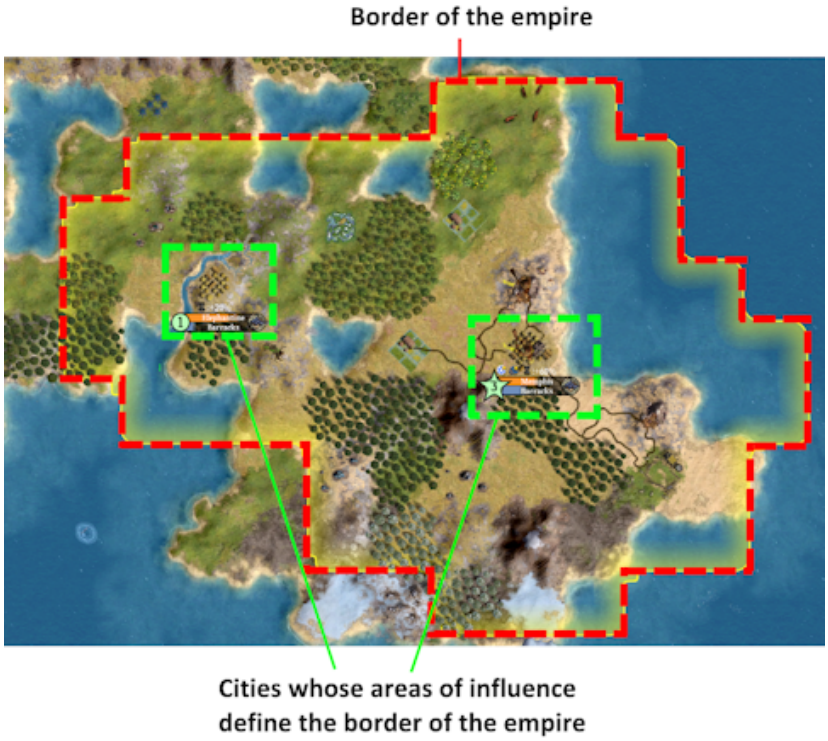


Figure 4.2: Civilization IV: The Border of an Empire

4.6 Existing Procedure for City Foundation

The existing procedure for building cities for a given civilization consists of several steps. However, it is strictly sequential and only takes into account information from the moment the plot is selected. This means, for instance, that possible locations for future cities or a high-level plan for the layout of the empire are not taken into consideration.

It is with respect to this characteristic that the adaptable reinforcement learning algorithms are to improve the existing AI. Since the reinforcement learning agent only replaces a small part of the process of founding a new city, it is important to understand how the existing process works. It is vital that the newly created process fits in seamlessly with the other parts of the game to enable meaningful experimental comparison between the existing process and the newly created process.

Creating Settlers

The founding of a new city is not determined by a direct decision of the AI that another city is needed but by the creation of a new settler unit in one of the cities. The creation of a new settler is a completely decentralized process. Every turn the best unit or building to be

produced in each city is determined by evaluating the needs of the active player in general and the needs of the specific city in particular.

Since the existing computer AI is programmed in a way that never lets a settler move around on its own, i.e. without a fighting unit as an escort, the fact that settlers are easily killed by any hostile fighting unit does not matter for the settlers controlled by computer AI. Consequently, the reinforcement learning process does not have to take into account lost settlers but can just assume that once the site for a new city has been chosen, that city will be built. Human players on the other hand often take the risk of sending a settler out on his own to save the time and resources it takes to build a defending escort.

Founding a City

Once a settler has been created, the computer AI computes a new command for this settler at every turn. The computation consists of a sequence of checks for possible actions. Depending on what the circumstances are the settler will usually be sent to a certain spot on the map where it will found a city. Since settlers cannot defend themselves, the computation also includes the check for a suitable escort. If such an escort is not available, the computer AI will order the settler to return to a safe place.

Other possible actions include loading the settler onto a transport ship if a much better founding spot has been discovered on a different continent or founding a city on the spot where the settler is located if the game has just begun and every turn that a city can be founded earlier is crucial.

The Selection of the Best Founding Sites

The most important method in the existing game AI with regard to this thesis is the method that determines the value of a certain plot when it comes to founding a city on this specific plot. The reinforcement learning approach will have to compete with the method the existing AI uses to determine the value of a certain plot. Therefore the present section will analyse the existing method of determining the value of a plot.

The existing method of selecting the best founding site computes an integer number for every plot. The number represents the value of founding a city on this plot by a given player. For the computation of the value a number of different factors are taken into account:

- Position of the plot in relation to water.
- Proximity of enemy settlements.
- Proximity of friendly (own) cities.
- Existing cities on the same continent (Team member's/Enemies/Own).

(a) Badly Distributed Cities

(b) Well Distributed Cities

Figure 4.3: City Distribution

- Characteristics of the enclosed workable tiles for the possible city:
  - Number of workable tiles.
  - Number of workable tiles owned by enemies or team members.
  - Number of workable tiles used by another one of the player's cities.
  - Yield of the workable tiles in terms of production, commerce, and especially food.
  - Possible gain in special resources.
  - Special resources that founding a city here will lock from access.

It is obvious from these characteristics that the selection of founding plots by the computer AI is completely static and based on the time of the calculation. No planning for future city placements or consideration of the bigger picture is done.
The actual effect of the use of reinforcement learning should therefore be twofold.

1. The already existing technique should be copied where it is useful. However, this should not be done by using a static evaluation method but through reinforcement learning. An example of a basic learnable strategy is optimising the distance between cities. Only one city can use a plot at any one time. Therefore, if the distance between cities is too small, their growth and the number of special resources they can access is limited (Figure 4.3).

2. Furthermore, the application of reinforcement learning should also lead to the development of new, improved strategies where the old method is lacking. This would include the founding of cities at sites that are neglected by the existing static evaluation method. For example sites that will normally be occupied by an opponent is not an obvious first choice as a founding site. However, settlement of some of these sites offers access to important resources and/or strategically important areas and can thus be decisive for a whole game.

The present chapter has given an overview of the chosen testbed, Civilization IV. The existing alternatives were pointed out and it was illustrated why Civilization IV was selected as a testbed. The game mechanics and the existing game AI have been described. Finally, the selected task, the selection of city sites, was elaborated on. The importance of this task as well as the method that is used by the standard game AI were explained.
The following chapter will focus on the integration of the algorithms and methodology (described in Chapter 3.5) into the game environment described in the present chapter. Both the design decision and considerations in terms of implementation will be explained in detail.

Chapter 5

Design and Implementation

This chapter describes the integration of reinforcement learning into Civilization IV. It furthermore elaborates on how the algorithms described in Section 3.5 have been adapted to the task described in Section 4.5. The process of adaption includes modifications to the game in order to produce a working system. Moreover, decisions on the design will be explained. Section 5.4 shows the adjustments that were made to speed up convergence and thus to counter the problem of a huge state space described in Section 5.2.1.

5.1 Policies

As stated in the section on algorithms (Section 3.5), while the overall aim of the agent is to find an optimal policy π^* for the given state space, it has to be provided with a standard policy π which it can use to pick the next action while learning this optimal policy. The policy π basically describes, given a state s, the probabilities of picking $a \in \mathcal{A}(s)$. That is, it describes the probabilities of being picked for all actions that are available in a state s. The policy provided can be one of several different types that have different levels of trade-off between *exploration* (picking a random action) and *exploitation* (picking the action that leads to the highest estimate). Depending on the purpose of an experiment, some policies are more useful than others. The selection of a preliminary policy is crucial since it determines the speed of the convergence. In theory, even with a completely random policy, convergence in the algorithms discussed is guaranteed if every state is visited infinite times and the reinforcement learning parameter are set to commonly used values. In practice however, convergence should of course be as fast as possible.

5.1.1 Greedy Policies

A greedy policy is a policy which always selects the action that maximises the return. It means that the agent will always pick the action with the best return according to the current knowledge of the agent. On the down side, a greedy policy completely ignores exploration,

i.e. no random and possibly better actions are picked. As for probabilities, a greedy policy implies that the action with the maximal estimated value is chosen with probability 1. If there are several actions that have the maximal value, it depends on the implementation if the first or last action with a maximal estimated value or a random action from all actions with maximal value is chosen.

5.1.2 ϵ-Soft Policies

ϵ-soft policies are all policies for which $\pi(s,a) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ for some $\epsilon > 0$ for all states and actions. This means that all states will eventually be visited. ϵ-soft policies are thus a superset of a whole range of policies, including ϵ-greedy policies which are described in the next section.

5.1.3 ϵ-Greedy Policies

One of the most commonly applied strategies in reinforcement learning is the ϵ-greedy strategy. The ϵ-greedy strategy implies that with probability $1-\epsilon, 0 < \epsilon < 1$ the agent will take an explorative action in $\epsilon * 100$ percent of all choices and an exploiting action in $(1-\epsilon) * 100$ percent of the cases. For the probabilities of the actions this implies that for all nongreedy actions a the probability of the action being picked is $\pi(s,a) = \frac{\epsilon}{|\mathcal{A}(s)|}$. For the greedy action the probability is the remainder, $1 - \frac{\epsilon}{|\mathcal{A}(s)|}$.

A special case of ϵ-greedy policies are *declining ϵ-greedy policies*. While the policy starts out as a standard ϵ-greedy policy, ϵ will slowly decline with the number of episodes until the policy is completely greedy.

5.1.4 Softmax Policies

A softmax policy can be used to counter the major drawback of ϵ-greedy methods, namely that when it comes to exploration, all actions are picked with equal probability. This strategy of picking actions implies that both the actions with the worst and with the second-best estimated value are equally likely to be picked, which is undesirable. To counter this drawback of ϵ-greedy methods, a parameter $\beta > 0$ is introduced into the probability equation. It is called *temperature*. This parameter determines how peaked the distribution is around the greedy action. The most widely used softmax method uses a Gibbs, or Boltzman, distribution:

$$\pi(s,a) = \frac{e^{\frac{Q(s,a)}{\beta}}}{\sum_{a' \in \mathcal{A}(s)} e^{\frac{Q(s,a')}{\beta}}}.$$

The higher the temperature is, the closer the actions get to being all equiprobable. On the other hand for $\beta \to 0$, the probability distribution approaches that of a greedy policy.

Depending on the area of application and human expertise, both policies, softmax and ϵ-greedy, have their advantages. However while both policies have exactly one parameter that has to be set, the effects of changes in ϵ are easy to track. A change of β, on the other hand, requires knowledge of the possible estimated values for the actions. For this reason ϵ-greedy and declining ϵ-greedy policies are used throughout the evaluation described in Chapter 6.

5.2 Reinforcement Learning Model

In order to apply reinforcement learning algorithms in an effective way, a model which contains the modules of an MDP described in Section 3.5.2 has to be defined. The model should furthermore fulfill the Markov property (Section 3.5.1) to enable efficient reinforcement learning. Therefore the reinforcement learning model for the city placement task consists of the quadruple $(S, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$: the set of states S, the set of possible actions $\mathcal{A}$, the transition probabilities $\mathcal{P}^a_{ss'}$ and the expected scalar reward signal $\mathcal{R}^a_{ss'}$. Ideally the chosen state and action space are finite. If they are finite, it implies that the MDP is a *finite MDP* which would simplify the reinforcement learning process. The following sections describe design and selection of the individual modules of the MDP.

5.2.1 States

A state $s \in S$ should contain all information at one point of the game that is important to the reinforcement learner, i.e. that the agent needs to make a decision. In order to have the Markov property, the state should also sum up all important information up to that point in time so that no previous information is required. For the task of city site selection, the important information at any one point in time is the information on all existing cities of the active player. The recorded values for each city are

- its position as (x, y) coordinates, i.e. the plot this specific city was built on,
- its rank in the founding sequence, i.e. when the city was founded, in relation to the other cities of this player.

The rank in the founding sequence is important to satisfy the Markov property: Without the rank, previous states would be required to obtain all necessary information. In general the rank of a city is crucial since a different order of founding can lead to very different results. Thus a state $s \in S$ can be described as a set of triples *(X-Coordinate, Y-Coordinate, Rank in the Founding Sequence)* with each triple representing one city.

This definition of a state suggests, that the set of all states S consists of all possible combinations of the *(X-Coordinate, Y-Coordinate, Rank in the Founding Sequence)* triples where cities can be built. They can be built on any plot $p \in P$ where P is all plots on the map. The resulting size of the state space is

$$|S| = \sum_{i=0}^{c} \frac{|P|!}{(|P| - i)!}, \tag{5.1}$$

with $c = |P|$ since every plot on the map could be a city.

Civilization IV offers several different standard map sizes for different numbers of players and terrain settings. It also offers the option to create custom map scripts. Table 5.1 shows the size of the standard maps in plots as well as the resulting number of possible states, depending on the number of cities that have been built.

	Duel Map (640 Plots)	Tiny Map (960 Plots)	Small Map (1664 Plots)	Standard Map (2560 Plots)	Large Map (4368 Plots)
1 City	640	960	1664	2560	4368
2 Cities	409600	921600	2768896	6553600	19079424
3 Cities	261326080	882894720	4601908480	16764113920	83300773920
4 Cities	1.66465E+11	8.44931E+11	7.64377E+12	4.28658E+13	3.63608E+14
5 Cities	1.05872E+14	8.07755E+14	1.26887E+16	1.09565E+17	1.58678E+18
6 Cities	6.72289E+16	7.71407E+17	2.10505E+19	2.79939E+20	6.92314E+21

Table 5.1: Relationship between the Map Size and the Number of Possible States

It is obvious from Table 5.1 that a brute force approach with experiments that use the biggest map size and run for a complete game with every player building a large number of cities, will not result in noteworthy coverage of the state space. As stated in Section 3.5 in the description of the algorithms, a convergence to the optimal policy π^* is only guaranteed for infinite visits to every single state. Convergence becomes more likely the more often all states are visited. The large number of states, especially for the bigger maps, makes it unlikely that all states are visited even once in the course of an experiment.

It is therefore necessary to keep the number of possible states to an absolute minimum and to optimise the algorithms. Measures taken to achieve this goal are described in Section 5.4.

5.2.2 Actions

The transition from one state to another is brought about by founding another city, thereby extending the existing set of *(X-Coordinate, Y-Coordinate, Rank in the Founding Sequence)* triples by another one. The set A of possible actions which can be taken when in a state $s \in S$ thus consists of founding a city on any of the plots ($p \in P \| p \notin s$), i.e. on any plot where there is no city of the active player yet.
This definition of the states and actions implies that the resulting state space will be like a graph with no cycles, i.e. a tree. Figure 5.1 shows a part of such a tree with the nodes representing the states and the branches representing the actions. Due to the structure of the state space, no state can be reached more than once in one episode.

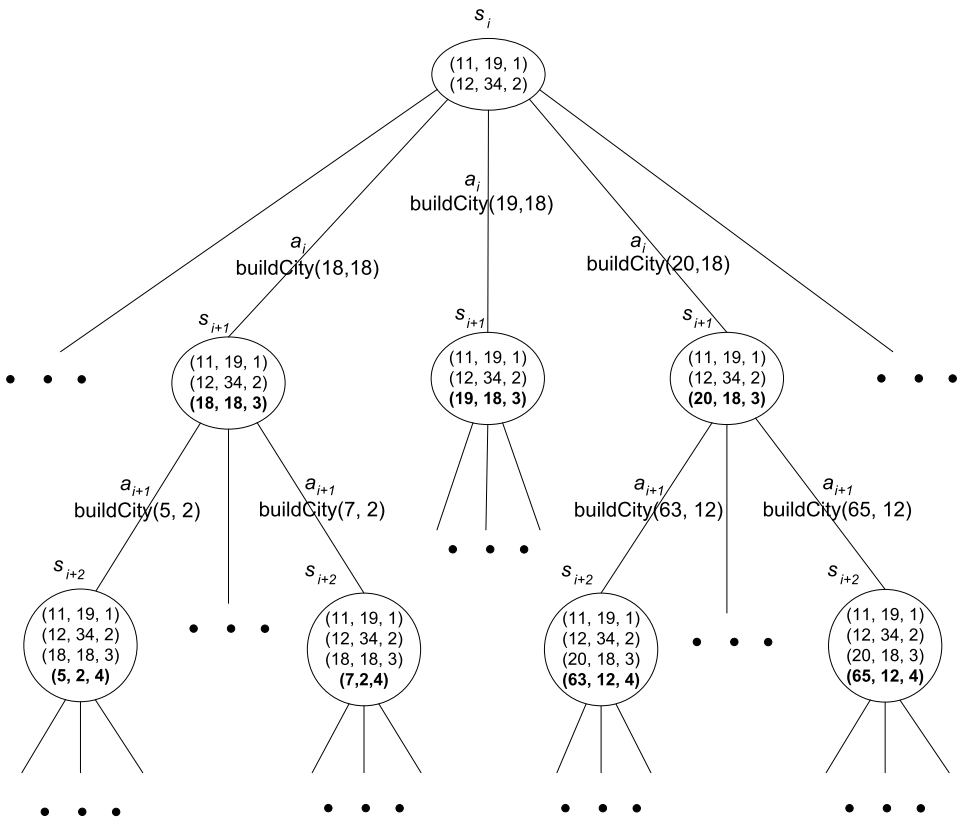


Figure 5.1: Excerpt of the State Space S including the Actions that lead to the Transition

The model above is a simplification of the game since it ignores cities which are lost to an enemy as well as cities which are won from an enemy. However, for the purpose of the present study, the simplification is acceptable because this approach focuses on the early expansion phase in the game during which attacks on cities are rare.

5.2.3 The Transition Probabilities

The algorithms that have been introduced in Section 3.5, Q-learning and Sarsa, compute a state-action function Q^π and not a state-value function V^π in contrast to standard TD-learning. Therefore no model of the environment, i.e. of the transition probabilities $\mathcal{P}^a_{ss'}$, is required.

5.2.4 The Reward Signal

Choosing the scalar reward signal $r_t \in \mathcal{R}^a_{ss'}$ is one of the most important decisions for the success of an agent that is based on reinforcement learning. The reward signal defines the goal for the agent and if it is chosen incorrectly, the agent will not be able to learn how to achieve its goal.

The Scalar Reward

The chosen reward signal for the task of selecting city sites is based on the score of a player. In the normal game this score is used to compare the performance of the players with each other and it is updated every turn for all players. The game score is determined by

- Population (in the cities),
- Territory (the cultural borders of the cities, see Figure 4.2),
- Technological advancement (developed with research output from the cities) and
- Wonders of the world (special buildings in the cities that provide particular benefits).

All the parts the game score is made up of are directly linked to the cities. The game score does not include any points for military strength since military units are always only a means to an end. A whole game can only be won through cities.

Attributing Rewards and the Problem of Delayed Reward

Since cities show their true potential only later in the game when they grow bigger and become more advanced, reward can only be attributed with delay. Another problem is the difficulty of attributing game score to a specific city since some information is not accessible to the agent. For instance the reward signal measures the direct contribution of a city to the score but it does not calculate the additional score if the city leads to the foundation of another city and thus indirectly adds to the player's total score. This problem arises because the decision to build a new city lies with a different part of the AI and is still controlled by the original game. Through the right choice of algorithm and parameters

for this algorithm (see Section 5.3), reinforcement learning should however be able to attribute the state-action pair which leads to this city being built an appropriately high Q-value.

Since cities are built only infrequently, states have to be decoupled from the normal game turns. Going from one game turn to the next is unlike the transition from one state to the next. A time step for the reinforcement learner is the time frame in which it has to choose an action and receives a reward after performing that action. It is defined as the time between founding one city and founding the next city. The update of the Q-value $Q(s_t, a_t)$ after taking an action a_t in state s_t happens immediately before executing the next action a_{t+1}, i.e. founding the next city. The selection of the appropriate plot for the foundation of a city, however, can happen several game turns before, with the settler unit moving to the chosen plot afterwards. The scalar value which represents the actual reward is computed by calculating the difference in game score between the founding turn of the last city and the founding turn of the city about to be founded. The difference is then divided by the number of game turns that have passed between the two foundations:

$$r \leftarrow \frac{(GameScore_{new} - GameScore_{old})}{(GameTurns_{new} - GameTurns_{old})}$$

Figure 5.2 shows the process of attributing reward for a computer agent that selects city sites by using the method presented above.

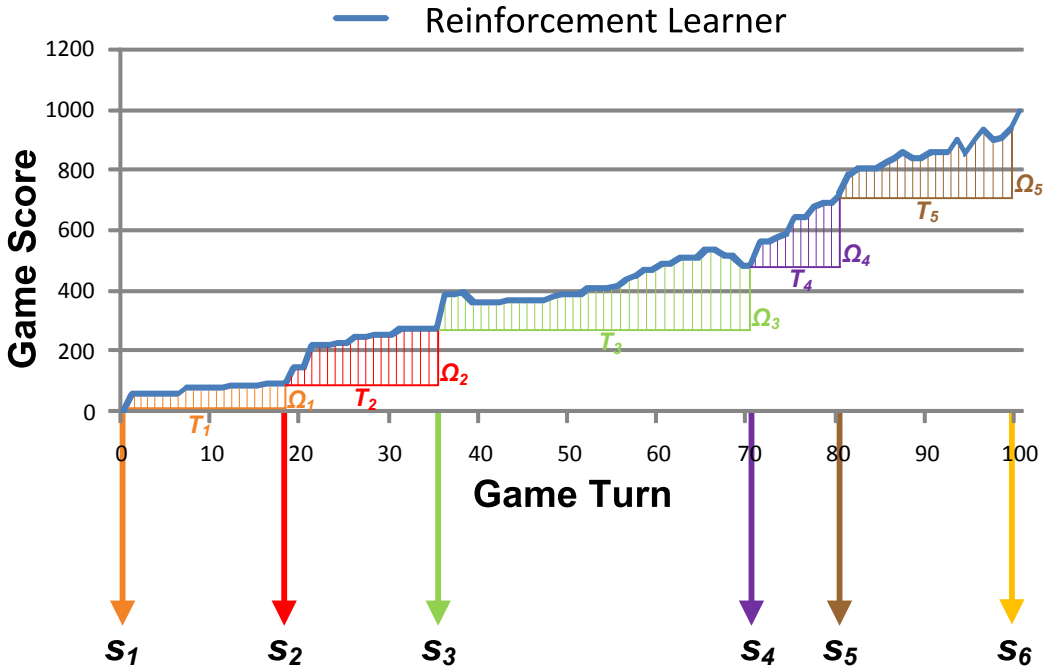


Figure 5.2: Computation of the Rewards

The downward arrows mark the game turns in which new cities are founded and the state is changed. The time it takes the settler unit to move to the designated founding site is ignored for the sake of a comprehensive view. If it was taken into account, it would not change the actual reward computation since the turns counted would remain the same with the state changing at a slightly earlier time. The ruled triangles mark the intervals which are taken into account for the computation of a reward. The x-axis represents the game turns T while the y-axis represents the game score Ω. The diagram shows that rewards are attributed in retrospect, i.e. after the next city has been founded.
by default the TD algorithms described in Section 3.5 propagate rewards backwards if they are run for several episodes. This feature is also very useful because it partly offsets the problem of delayed score gains described at the beginning of this section. Another mechanism to counter the problem of delayed reward even more effectively are eligibility traces (Section 3.5.6). Their integration is described in the Section 5.3.

5.3 Implementation of the Reinforcement Learning Algorithms

This section explains how the algorithms described in Chapter 3.5 are integrated into Civilization IV and how they are used to learn the task of city site selection.

Figure 5.3 shows a general view of how reinforcement learning is integrated with the game environment. The integration is independent of the algorithm and parameters used.

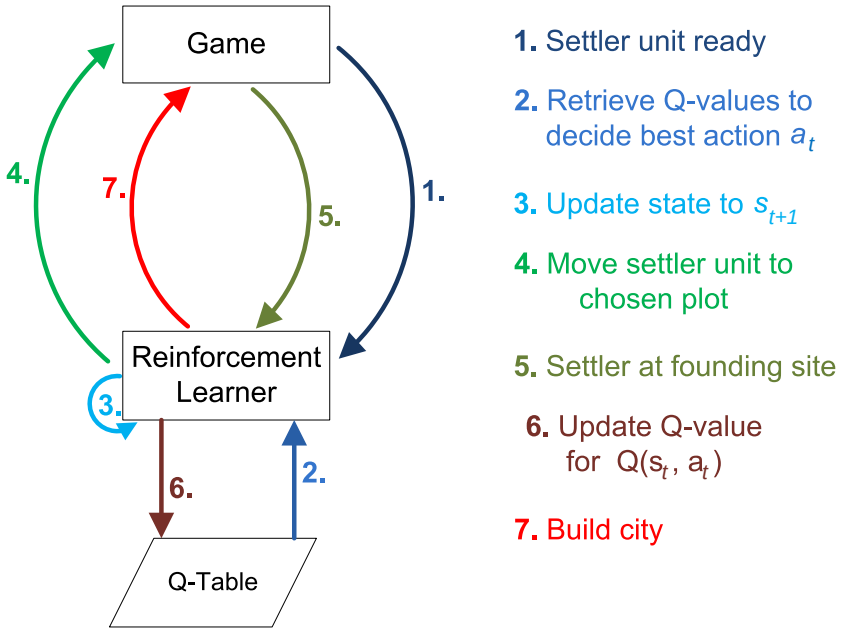


Figure 5.3: General View of the Algorithm Integration for City Site Selection

The diagram clearly shows that the algorithm works in an asynchronous way. While the change from state s_t to state s_{t+1} already occurs in step 3, the update of the estimated value for the state-action pair $Q(s_t, a_t)$ only occurs in step 6, after the game has created another city with the next settler unit.

The following sections illustrate how the algorithms and techniques described in Section 3.5 are adapted to the specific task at hand.

5.3.1 Integration of the Algorithms

The implementation of the two algorithms as well as the implementation of certain variations like eligibility traces and previously initialised states are similar for Q-learning and Sarsa. Internally, they use a 'linked tree' of state objects to store the single states. Due to the enormous size of the state space described previously, not all possible states are initialised

at the start of a game. The data structure that has been created to store all the necessary information of a single state is shown in Appendix C.

The reinforcement learning method completely replaces the method which is used to control the settler in the original game. Every time a settler becomes active for the agent that uses reinforcement learning, the reinforcement learning method is executed. Since settlers do not choose the preferred city site, travel there and build a city all in one turn, this newly created method can not only be a simple reinforcement learning method. It must also be able to move a settler that has already chosen the best city site and to create a city with the settler once it gets there. Furthermore it must be able to handle exceptions along the way, e.g. if the chosen city site becomes unavailable because an opponent settles there first or if the path is blocked by enemy territory.

The basic method or rather the basic set of methods into which the actual reinforcement learning methods are embedded is a set of if-else statements. The flow-diagram for this part of the logic can be seen in Figure 5.4.

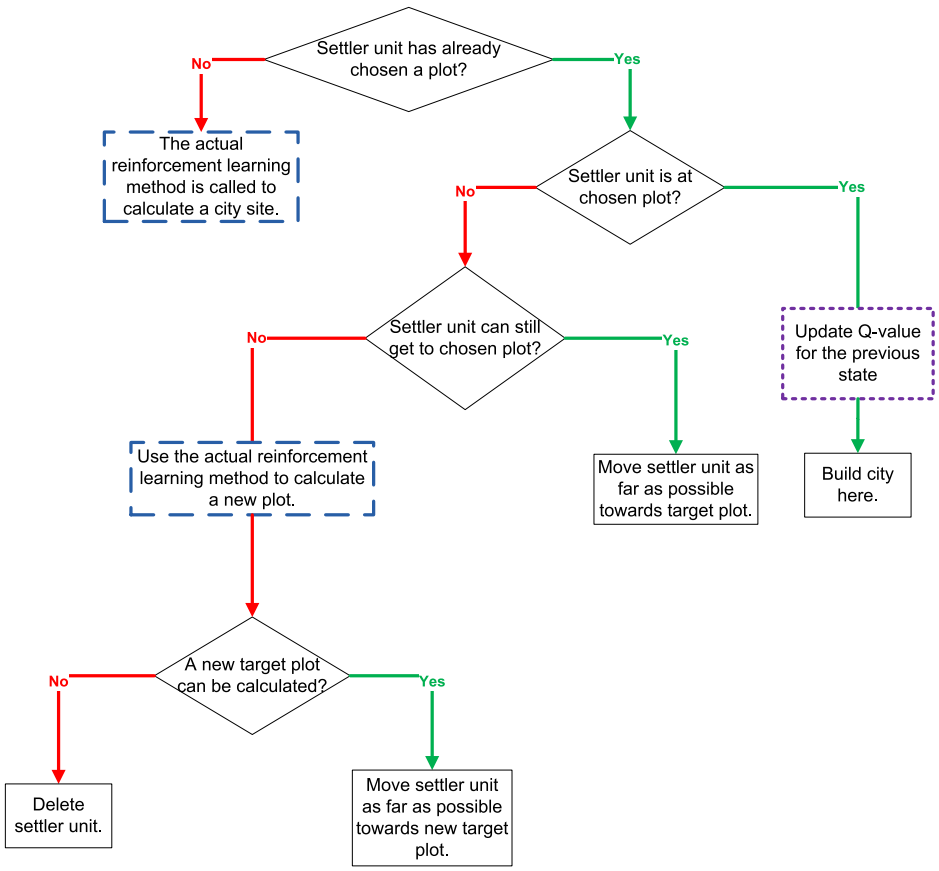


Figure 5.4: Flow of Events for City Site Selection

The actual algorithm is split into two parts. One part is the method that chooses the city site. The execution of this method is highlighted by the dashed border in the flow chart in Figure 5.4. The pseudocode for this method for one-step Q-learning can be seen in Listing 5.1.

```
if(possible states s' which follow taking action a ∈ A(s) are not yet initialised){
      Determine what plots meet the criteria for settlement.
      Create state objects for all of the chosen plots.
}

while(no city site selected)
{
      Compute random number α with 0 < α < 1;
      if(α < 1 − ε)
      {
            //follow exploitative policy
            Choose action a as the action with the highest estimated Q−value Q(s,a);
      }
      else
      {
            //follow explorative policy
            Compute random number φ with 0 ≤ φ < |A(s)|;
            Choose action a as the action at position φ in the array of A(s);
      }
      if(plot chosen through action a is reachable and settleable)
      {
            break;
      }
}
```

Listing 5.1: Pseudocode for the Implementation of the Plot-Selection Method for Q-Learning

The other part, which updates the value function $Q(s,a)$, is highlighted by the dotted border and the pseudocode for one-step Q-learning can be seen in Listing 5.2. This method is always executed when a new city is founded.

```
if(there is a previous state s_{t−1} for which Q(s_{t−1},a_{t−1}) can be updated)
{
      Determine number of turns T_new as T_new = T_now − T_lastcity which
            have passed between the founding of this city and the last;

      Determine the gained score Ω_new since the last city
            was founded: Ω_new = Ω_now − Ω_lastcity;

      Compute the reward r_t as the score gained per turn between founding
            this city and the last: r_t ← Ω_new / T_new;

      Update Q(s_{t−1},a_{t−1}) ← Q(s_{t−1},a_{t−1}) + α[r_t + γ max_a Q(s_t,a_{t−1}) − Q(s_{t−1},a_{t−1})];
}
```

Listing 5.2: Pseudocode for the Implementation of the Value-Update Method for Q-Learning

The pseudocode in Listings 5.1 and 5.2 shows Q-learning in particular. However, the methods for Sarsa look very similar. The major difference between Sarsa and Q-learning is that for Sarsa the following state and action have to be taken into account as well. This means that both methods, the one for choosing the city site and the one for updating the Q-values require one more state and one more action to be recorded previous to the execution of the method described above.

5.3.2 Implementation of Eligibility Traces

The introduction of eligibility traces (Section 3.5.6) into one-step Q-learning and one-step Sarsa is vital for the given task of selecting optimal city sites. Usually eligibility traces in reinforcement learning algorithms are used to speed up learning by propagating rewards back through a number of states that led up to the current state.

As stated in Section 5.2.4, an issue specific to the given task is that it is problematic to attribute game score to one city only. The approach taken uses the score gain per turn between one city founding and the next city founding,which, in a way, is inaccurate. The approach manages well to record the tangible benefits of improved territory and population, but it ignores most long-term effects that occur after the next city founding. However, these long-term effects are very important since cities develop to their full potential only after they have been allowed to grow for a number of turns. This shortcoming can to a certain degree be rectified by using eligibility traces. Eligibility traces assign temporal credit through propagating rewards back through the sequence of states and actions that have led up to the current state. eligibility traces therefore ensure, that cities are attributed with the score they help achieve in the long term.

In terms of implementation, instead of the one-step versions of Q-learning and Sarsa, variations of these algorithms which use eligibility traces are used: Watkins Q(γ) and Sarsa(γ) (Section 3.5.6).

Figure 5.5 illustrates again the method of attributing reward in retrospect, i.e. after the next city has been founded, introduced in Section 5.2.4. Based on Figure 5.5, Figure 5.6 compares the update of Q-values for the state-action pairs with a one-step Q-learning algorithm with that of Watkins' Q(λ).

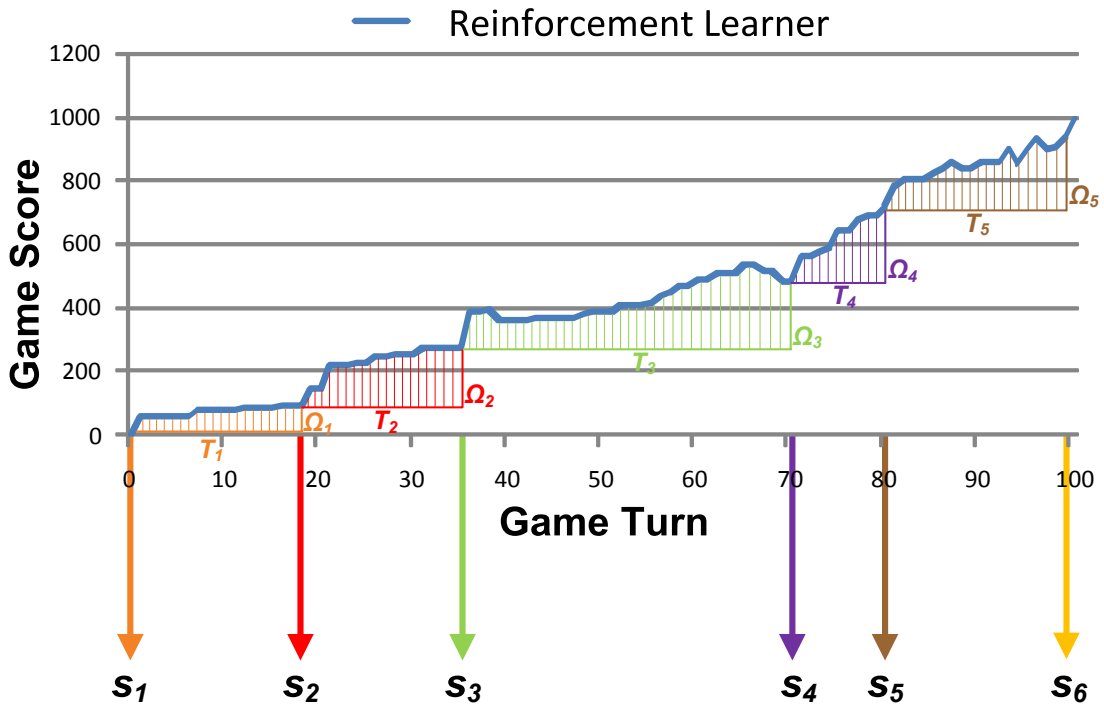


Figure 5.5: Computation of the Rewards

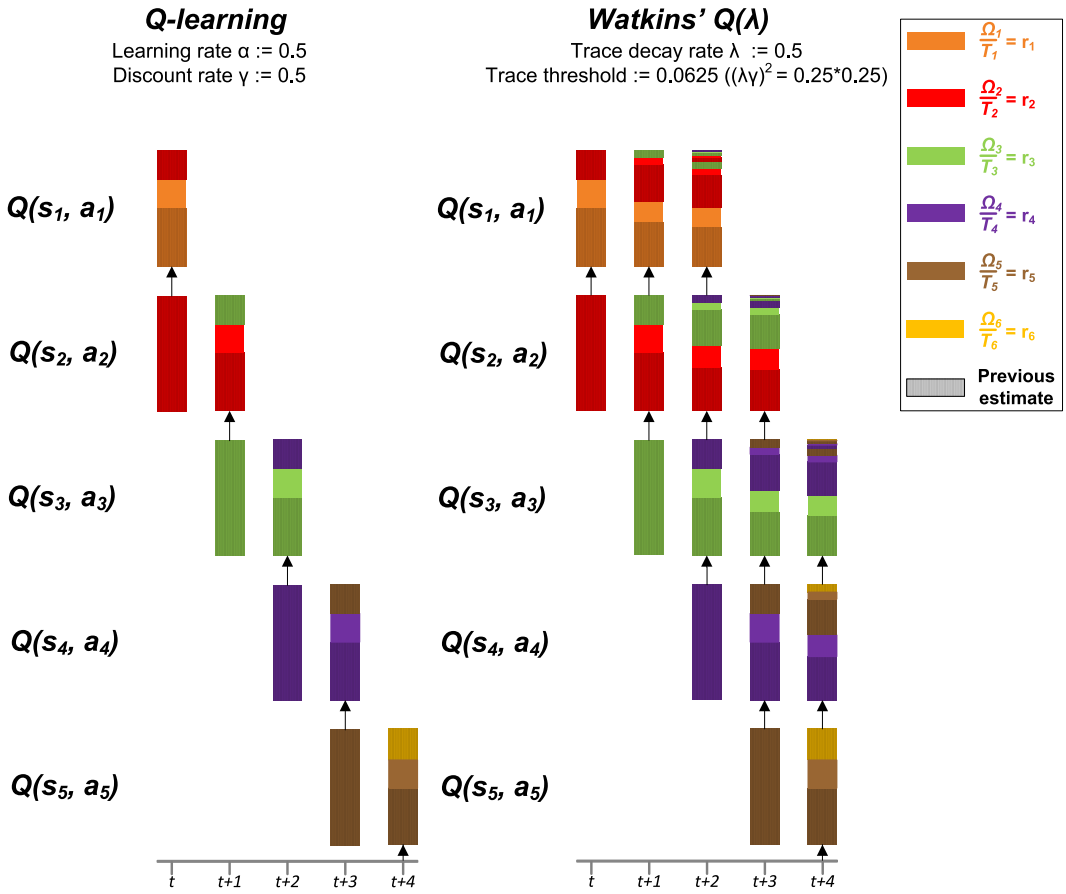


Figure 5.6: Comparison of Value Updates between One-Step Q-Learning and Watkins' Q(λ)

The diagram shows clearly that value updates through Q-learning (left side) are slower at propagating rewards of later states backwards. The trace threshold, i.e. the value underneath which the trace $e(s, a)$ will simply be counted as 0, is set to $(\lambda\gamma)^2$. This implies that rewards only propagate backwards two time steps, which is a relatively low value. Nevertheless, a noticeably faster convergence towards the estimated values of future state-action pairs can be observed.

For Watkins' Q(λ) it is also notable that, in order to make the propagation viable with Q(λ), the policy shown has to be the optimal policy since otherwise the backward propagation would stop at the first non-optimal choice.

5.4 Accelerating Convergence

This section discusses the modifications that were made to speed up the convergence of the reinforcement learning algorithms and thus to handle the problem of computability described in Section 5.2.1.

5.4.1 Reduction of the State Space

As shown in Table 5.1, the number of possible states grows exponentially with the map size. Therefore, simply reducing the map size and thereby the number of plots alone will not solve the problem, though this can contribute to solving the problem as well. A reduction in map size would efficiently reduce the number of possible states in earlier iterations of the algorithm, thus making these maps usable for evaluation.

Another option is to limit the plots that are considered for settling. Plots that can not be used to found a city are ignored.
This includes

- plots that are completely covered in water (either inland lakes or oceans) and
- plots that contain high mountains.

Ignoring these plots effectively reduces the number of plots on a map by two thirds on average. Since the reinforcement learning algorithms can not be influenced negatively or slowed down in any way through ignoring these plots, the exclusion described above is part of all performed experiments.

5.4.2 Pre-Initialised Q-Values

In order to accelerate convergence, previous knowledge can be used. Reinforcement learners generally do not require any previous knowledge of the environment and acquire all information through the learning process. However, previous information that is already available can be useful to speed up this acquisition process.
The knowledge available about the best city sites in a game of Civilization IV is computed by the standard city site selection method. This information is available to the standard game AI and can be used to initialise the Q-values for state-action pairs of the reinforcement learner. Usually these Q-values are all initialised to 0, which makes all actions equally likely to be picked. Consequently every state has to be visited in order to determine its usefulness for the exploitation part of the policy, often only to conclude that its usefulness is very low. If the states were instead initialised to the precomputed 'founding values' that are used by the standard game AI, these values could serve as indicators for the usefulness of a plot for

the reinforcement learner. This process would not speed up the guaranteed convergence to an optimal policy π^* but it would generate better performance earlier on.

However, this strategy has two side effects that have to be considered. One of the side effects is that the use of values computed by the standard AI could easily lead to a performance very similar to that of the standard game AI. To prevent this, only the values that are computed before the first cities are placed will be used to initialise both the first and later states. This strategy will also avoid making the founding values dependent on city locations.

The second side effect is that the founding values are in a completely different range than the values for rewards: Founding values usually range from 500 to 5000, whereas reward values, i.e. score gain per turn, will be somewhere between 4 and 50 depending on the stage of the game. Therefore, a pre-initialised Q-value will be changed completely after the particular state has been visited the first time. If Q-values were not normalised after the first visit to a state on that layer in the state-space, they keep the higher pre-initialised founding values. To avoid this, all Q-values are normalised relative to the computed reward of the first state-action pair to make the Q-value of all other states on the same layer of the state space comparable. Possible negative effects of the initialisation with founding values computed by the standard AI are evaluated in in the empirical evaluation in Section 6.

Besides the two techniques that are used in order to speed up convergence and reduce run-time, several other decisions with the goal of speeding up convergence were made when setting up the experiments to evaluate the algorithms. These decisions are all elaborated on in the section on experimental setup in the following chapter.

The present chapter has described how the reinforcement learning algorithms from section 3.5 were adapted to Civilization IV. The implementation of the different parts of the Markov model from section 3.5.2 in the game environment has been illustrated in detail as has the integration of the algorithm into the overall system. The issues related to including this machine learning technique into a commercial game environment have been described as well as the measures taken to resolve these issues.

The following chapter will present experiments that are performed using the different reinforcement learning algorithms implemented in Civilization IV. The experiments are executed using multiple sets of parameters on each algorithm. Each of the performed experiments serves a certain purpose that is elaborated on in the appropriate section.

Chapter 6

Evaluation

This chapter describes the experiments that were run using the algorithms and the environment described in the previous chapters.

In the first section the experimental setup as well as the measures that are taken to improve reliability of the experiments are described. The following section illustrates how the optimal settings for the test runs are determined by analysing preliminary experimental results. Subsequently the shortcomings in the experimental setup that can be identified from the results are eliminated. The third section elaborates on the performance of the four different algorithms after a working setup has been found. The final section of this chapter uses the knowledge acquired in previous sections to compare the performance of the four variations of reinforcement learning algorithms with the approach of the standard game AI.

6.1 Experimental Setup

This section gives a description of the parameters which are used in the algorithms as well as the settings which are chosen within the game for the subsequent experiments.

6.1.1 Algorithm Parameters

As stated in Section 5.3, several different parameters can be set for the algorithms. The parameters that are shared by all algorithms are the following ones:

- The learning rate α. The learning rate $0 < \alpha < 1$ determines how much of the previous estimate will be replaced with the new estimate. A value of 0 will prevent the agent from learning anything while a value of 1 will completely replace any previous values.

- The discount factor γ. The discount factor $0 < \gamma < 1$ determines the weight of future rewards. For a value of 0 future rewards are ignored. A value of 1 will result in the agent considering current and future reward as equally important.

- The exploration rate ϵ. In the ϵ-greedy policy which is used the exploration rate determines the ratio between exploration and exploitation (see Section 5.1).

The parameters which are specific to the algorithms that use eligibility traces, Q(λ) and Sarsa(λ), are the trace decay rate λ and a threshold Λ for trace decay. The trace decay rate $0 < \lambda < 1$ determines how much of a reward propagates backwards to Q-values of state-action pairs leading up to the current state. For the value 0, a future reward is only propagated backwards once, namely to the Q-value of the state-action pair directly preceding the current one. For the value 1, all Q-values leading up to the current one will receive the full reward.

The threshold Λ determines up to what value traces are taken into consideration. When an eligibility trace is too small, the according Q-value will be ignored when updating future estimates.

The actual values that the different parameters are set to are different from experiment to experiment. They are given in the according sections.

6.1.2 Game Settings

Civilization IV offers a multitude of settings to customize the game environment. Most of the settings serve the purpose to offer interesting game play for players of different skill levels or with different preferences when it comes to the style of playing. When running experiments, however, it is not the players' needs that have to be taken into account. Rather, the settings have to create an environment for experiments that provide reliable data. The requirements for experiments to be carried out successfully are discussed below.

The game environment has to be *fast*. This is important because the more quickly one game is over, the more games can be played in the course of one experiment. In order to shorten the time it takes to play one game, the game is run with the lowest settings possible in terms of graphics. Unfortunately, the parts of the game which have been released as open-source do not include the code for the graphical interface. Therefore it is not possible to deactivate the graphical front-end completely.

Furthermore, it is important to find the optimal length of a game. A game becomes increasingly slow as history progresses, because the players acquire more cities and units as well as more complex cities and units. Therefore, a low number of turns is beneficial for processing speed and game time. Also, as stated in Section 6.1, state space increases exponentially as numbers of cities increase.
However, in order to obtain *meaningful* results, a certain number of cities has to be built. A minimum number of cities is also necessary to analyse how well rewards propagate backwards

when using eligibility traces. The possible number of cities also depends on the map size and the number of players. To obtain the maximum number of cities per player while keeping the map size small, the experiments performed are duels between the reinforcement learning agent and one standard computer AI. If only two players are in the game, both of them will be able to found more cities than with a higher number of players. For two players the smallest possible map size, **Duel**, is used. Despite the small map size both players can still have up to five cities per player if they play optimally, i.e. greedy. The type of the map used is called **Pangaea**. For a Pangaea type map, all land surface is linked in one big continent. Other possible map types include islands. These are not used in the experiments because of the ships that as a matter of transporting settler units would have to be considered by the reinforcement learner. Thanks to the small map size and the fact that it only consists of one continent, the number of turns in the experiments can be limited to the first **110 turns** [1] and still encompass the complete settling phase. The settling phase is the phase during which the area is divided between the players through their city founding. As for the length of experiments, several test runs showed that for 110 turns played per game a viable number of episodes would be 1500. One episode is one game that lasts 110 turns. After 110 turns the environment is automatically reset and the game starts from the beginning again.

The limiting factor in terms of episodes is not time but memory, since all states are kept in memory by the program (for a description of the state object class see Appendix C). The memory needed for state objects created in 1500 episodes of length 110 turns amounts to about one gigabyte. An alternative approach was tested as well: States were stored to and retrieved from a database instead of being kept in memory. However, this approach was too slow to be used while running the experiments in real-time.

[1] a complete game lasts up to 400 turns

Table 6.1 lists the settings that were chosen to make experiments fast yet meaningful.

Setting	*Value*
Map Size	Duel
Plot Number	640
Plot Number after Pre-Selection	214
Map Type	Pangaea (One Continent)
Number of Turns	110 (4000BC - 375BC)
Number of Episodes	1500
Number of Opponents	1
Barbarians	Off
Leader	Hatsheput, Egyptian
Difficulty	Noble (4/9)

Table 6.1: Game Settings

In order to obtain reliable results, *normalisation* of the experiments is even more important than running them at the least amount of time possible. Normalisation is the process of making results comparable. Among other things, normalisation implies that the random factor is minimised. As stated in the description of the standard game AI in Section 4.4, Civilization only contains few random events. **Barbarians** are one of these few random events. Barbarians are basically another player who gets new units every turn and uses these to attack all other players. Since the barbarians turn up randomly outside the field of view of the players, they were switched off completely for the experiments.

When setting up the experiments, another prerequisite to make results comparable is that the conditions for each player in the game have to be equal. The first setting that has to be adjusted is the general level of difficulty. The difficulty setting can be used to control a set of basic values (e.g. the cost for creating and maintaining units and buildings or researching technologies) by which the AI and the human players have to play. By adjusting these values it can be made harder or easier for the human player to compete against the computer. Since the reinforcement learning agent replaces the human player, this setting has to be set to **Noble**, the fourth from the bottom out of nine levels of difficulty, which means that the computer AI and the reinforcement learning agent play by exactly the same set of rules.

Every player in Civilization IV represents a leader that the game assigns certain 'traits of character' by the game. These traits determine the player's style of play. For example, computer players can be aggressive or non-aggressive and they can focus on different priorities, e.g. expansion or research. In order to normalise experiments, the same leaders have to be chosen for all games. Furthermore, choice of leaders can influence the number of cities that are built during the first 110 turns since players expand faster if a leader that favours expansion is

chosen. To this end the leader **Hatshepsut of Egypt** was chosen for both players. Her traits of character allow cities to extend their borders fast. As an opponent she is not aggressive but focuses on extending her empire.

6.2 Parameter Selection for the Reinforcement Learning Algorithms

This section is concerned with preliminary experiments that identify a number of problem areas. In order to be able to obtain meaningful results in section 6.4 where the reinforcement learning algorithms are compared with the standard AI, it is important to know how the algorithm parameters have to be configured. Several problems occur when the wrong set of parameters for the algorithms is chosen. The problems and their possible solutions are discussed in the following sections.

6.2.1 Aim of Parameter Selection

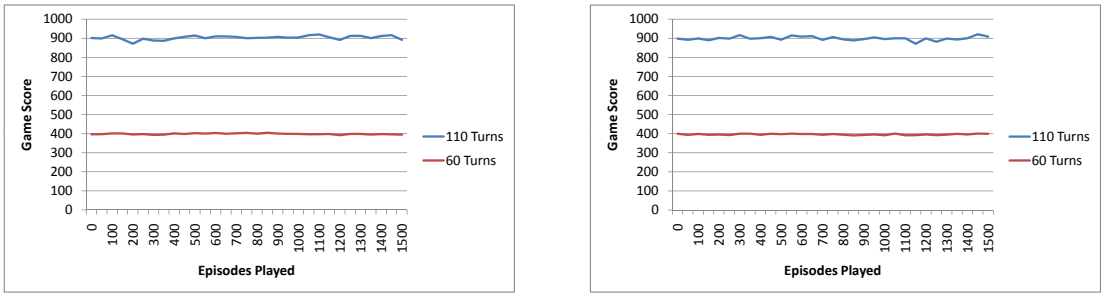
The aim of this section is to identify the settings that are most appropriate for obtaining meaningful results in future tests. In order to approximate these settings, the following method is used. A number of tests with different settings are run and evaluated. The main criterion of evaluation is the ability of the algorithms to learn the task of selecting city sites through reinforcement learning. Game score is used as a reward signal. Therefore, a clear growth of game score over time is expected to indicate a successful learning process. Since an ϵ-greedy policy is used throughout this section, the markedness of the increase in game score is determined by the value of ϵ. The increase in game score is triggered by the detection of better possible actions that can then be pursued when picking according to a greedy policy. Nevertheless, a growth in game score should be observable for any $\epsilon < 1$, i.e. for any policy that contains some degree of greedy selection.

6.2.2 Preliminary Test Runs

The first runs were carried out with parameters set to common values as described by Sutton and Barto (1998). The parameters are displayed in table 6.2. The values were slightly adjusted to match the given task since its focus on future reward is stronger than usual (higher discount rate γ) and its exploration rate ϵ is slightly higher than common because the large basic state space has to be taken into account.

Learning Rate α	0.2
Discount Rate γ	0.6
Exploration Rate ϵ	0.3
Trace Decay Rate λ (for Sarsa(λ) and Q(λ))	0.9

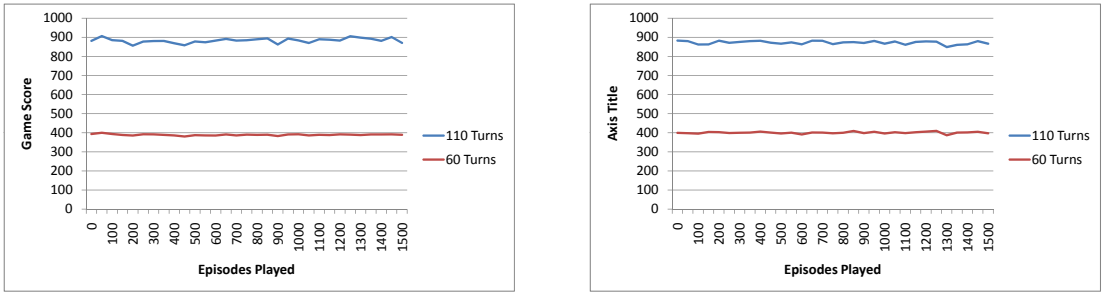
Table 6.2: Parameter Settings for Preliminary Runs



(a) One-Step Q-Learning

(b) One-Step Sarsa

Figure 6.1: Results for Preliminary Tests using the One-Step Versions of the Algorithms



(a) Q(λ)

(b) Sarsa(λ)

Figure 6.2: Results for Preliminary Tests using the Eligibility Trace Versions of the Algorithms

The results of test runs for 1500 episodes can be seen in Figures 6.1 and 6.2. The scores are averages over 50 episodes to account for the large random proportion caused by exploration. The different lines in the diagrams account for scores after 60 turns and after 110 turns. Each algorithm was run 10 times for 1500 episodes with the diagrams displaying the average of these runs. An example of the raw data that underlies these and subsequent diagrams is shown in Appendix B. Appendix B shows 1000 episodes of the results of a test run in Section 6.4.

The diagrams in Figures 6.1 and 6.2 show that there is no observable tendency towards a better policy that leads to a higher score. Furthermore, all algorithms seem to have comparable performance and the only thing distinguishing the results of the test runs from the results of a completely random policy is the comparably high score. There are several reasons for this behaviour.

The underlying reason is the size of the state space as described in Section 5.2.1. Since for the chosen game settings the number of cities averages about four per game, there are about two billion possible states at the fourth level of the tree that maps the state space (Formula 5.1 and Figure 5.1). While the size of the state space was expected to keep the agent from learning the optimal policy, the lack of an observable improvement has a different cause.

The reason for the lack of learning lies in the mechanics of the reinforcement learning algorithms as well as in the chosen values for the parameters. All Q-values are initialised at 0. This means that even for a greedy pick, the first selection will always be random. No matter how bad this first selection is, if a greedy pick is made at the same point of the state space in the next episode(a greedy pick happens in 80% of all cases due to ϵ being 0.2), the action to settle the plot that was previously picked at random will be picked again. Through the defined learning rate an algorithm will always replace about 20% of the old estimate by a combination of immediate and future rewards. Therefore, the Q-value for this state-action pair will not reach its full value at once but will reach it within a short time because of the 80% greedy selection policy. The problem of this policy is that in order to achieve the highest Q-value, randomly picked actions that lead to the founding of cities on better plots would have to be picked as many times - randomly - as the action that was picked first. Due to the bad ratio between possible states and episodes run, this is unlikely to happen. Consequently, the agent that uses reinforcement learning algorithms with the values chosen above will very likely pick a random policy and stick to it applying only minor variations. Figure 6.9 illustrates this behaviour.

The four algorithms only show minor differences in their results. Sarsa is slightly less influenced by the problem described here because it is an on-policy algorithm. It therefore has

a random component when selecting actions for the future-reward-component of the algorithm (Formula 3.2). Both Sarsa(λ) and Q(λ) slightly aggravate the problem described above, since they increase the approximation of the Q-value after it has been selected through propagating rewards backwards. However, the difference between algorithms is marginal.

There are several solutions to this problem. The first option is to change the parameters of the algorithm. The first step is to set the learning rate α to 1. This effectively means that only the estimate computed at a given point in time matters, i.e. no previous estimates will be taken into account. This can be combined with setting the discount rate γ to 0 in order to completely ignore the future rewards that could also influence Q-values.
The obvious downside of this approach is that it makes the whole learning process very volatile since no bootstrapping takes place anymore. Single outliers in terms of reward could influence the outcome of a complete experiment. Moreover, if future rewards are ignored, one of the key components of the TD algorithms is taken away.

The second option is changing the Q-values initialisation as described in Section 5.4.2. Instead of initialising them all to 0, they are set to the value of the plot that is settled in the action leading up to the respective state. This value is estimated by the standard game AI at the beginning of a game. This estimation is supposed to provide the reinforcement learner with some basic ideas as to what plots are desirable to found a city on, i.e. what actions should be selected when following a greedy policy.
The possible drawback of this approach is, that the developed policies might be very similar to those of the standard game AI. Furthermore, the problem it tries to solve might continue to exist albeit on a smaller scale: the reinforcement learning agent is still likely to pursue most often the policy it picks first . To counter this behaviour, an attenuated version of the first approach could be used. The learning rate α could be set very high and the discount rate γ very low.

A third possible approach would be to increase ϵ, i.e. the exploration rate. A higher exploration rate would increase randomly selected actions and thus the chances of finding and properly estimating the value of a rewarding action. However, it would also increase the time it takes to find good policies, which is undesirable.

Another very basic approach would be to reduce the number of turns, thus liming the number of cities that are built and thereby reduce the size of the state space. If the number of turns is limited, the underlying problem of inadequate state space coverage would also be resolved. Reducing the number of turns consequently seems to be the most promising strategy.

The following three sections show the results for tests applying the first two strategies as well as the last one. The strategies are always combined with an increased exploration rate

ϵ. An increased exploration rate will lead to a slightly lower average game score but also to a faster traversal of the complete initial levels of the tree that represents the state space.

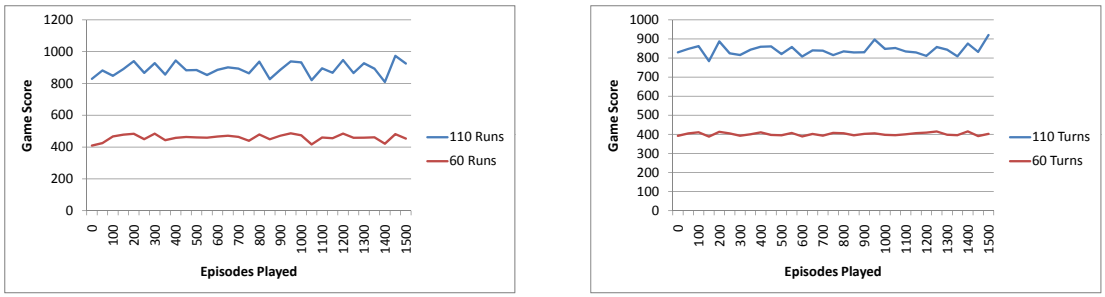
6.2.3 Test Runs with Full Discount

In this section the results of the experiments run according to the first option to solve the problem of the agent never switching policies are illustrated. The parameters are set to ignore previous estimates and to not take into account future rewards. The according algorithm parameters are listed in table 6.3.

Learning Rate α	1
Discount Rate γ	0
Exploration Rate ϵ	0.4
Trace Decay Rate λ (for Sarsa(λ) and Q(λ))	0.8

Table 6.3: Parameter Settings for Full Discount Runs

The results of the test run for these settings are displayed in the diagrams in figures 6.3 and 6.4.

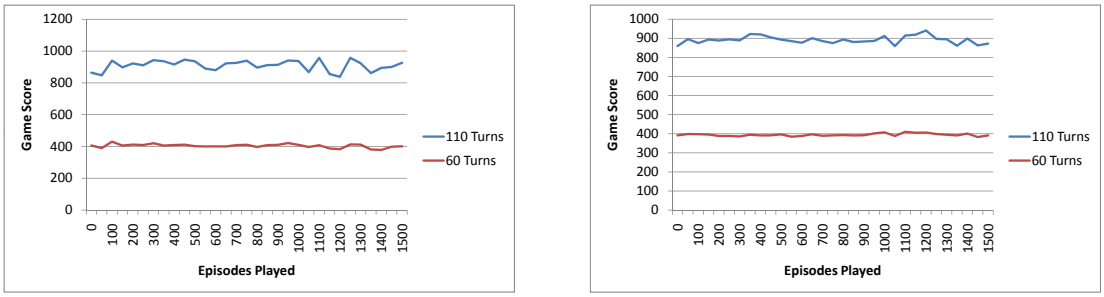


(a) Q-Learning

(b) Sarsa

Figure 6.3: Results for Fully Discounted One-Step Versions of the Algorithms

As expected for the given parameter settings, the diagrams of the results testify to a higher degree of variation in game score although the results are averages over a period of time and a number of runs. This is especially noticeable for the algorithms based on Q-learning, both for the one-step version and Q(λ). Usually the Q-learning algorithms should exhibit a more consistent performance than Sarsa since they are off-policy while Sarsa is on-policy. For the applied ϵ-greedy policy therefore Sarsa has an additional source of randomness (the



(a) Q(λ) (b) Sarsa(λ)

Figure 6.4: Results for Fully Discounted Eligibility Trace Versions of the Algorithms

40% random actions when selecting future actions) whereas Q-learning always chooses the best estimate according to a greedy policy. However, in the current setup no difference can be observed since future rewards are ignored completely. Since none of the four algorithms shows any noticeable trend of improving the game score, the chosen parameter settings are deemed to be unsuccessful in overcoming the problem described in the previous section.

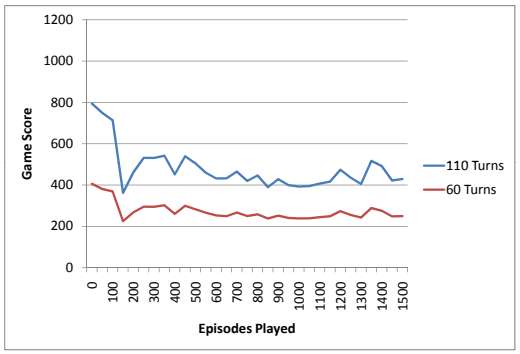
6.2.4 Test Runs with Pre-Initialised Q-Values

This section examines the results of tests run with the settings described in Section 5.4.2. The Q-values for all state-action pairs are initialised to the values that the standard AI attributes to them prior to the start of the game, i.e. before any city is built. The values are recorded before the first city is founded and then used to initialise Q-values whenever new states are created. The values have to be calculated before any city is built, because they afterwards become dependent on the location of enemy and own cities. The parameter settings for the tests run for this section are displayed in table 6.4.

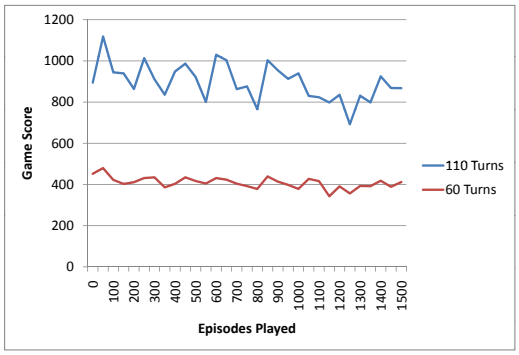
Learning Rate α	0.2
Discount Rate γ	0.6
Exploration Rate ϵ	0.4
Trace Decay Rate λ (for Sarsa(λ) and Q(λ))	0.8
Initial Q-Values $Q(s, a)$	Initial Values of Standard AI

Table 6.4: Parameter Settings for Tests with Pre-Initialised Q-Values

The results of the test runs are displayed in figures 6.5 and 6.6.

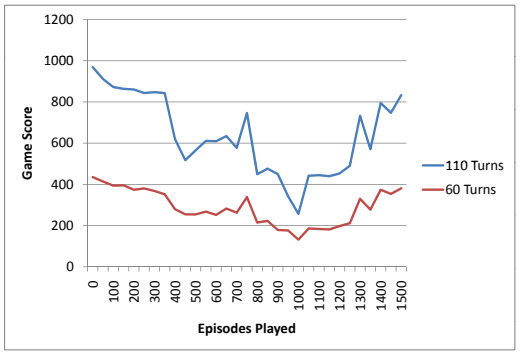


(a) Q-Learning

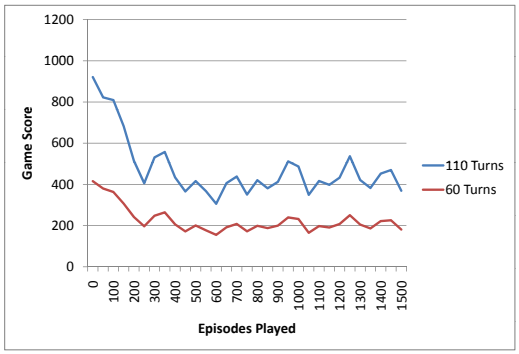


(b) Sarsa

Figure 6.5: Results for Pre-Initialised Q-Values using the One-Step Versions of the Algorithms



(a) Q(λ)



(b) Sarsa(λ)

Figure 6.6: Results for Pre-Initialised Q-Values using the Eligibility Trace Versions of the Algorithms

As the diagrams show, the use of pre-initialised Q-values does not have the desired effect of leading to a reliable convergence towards a higher score. In fact, the opposite is the case. After an initial high the score becomes either very volatile or declines to an average score that is far worse than the initial score.

On the upside it is noteworthy that initially all algorithms start with a high average when compared to previous test runs. This indicates that in the first few episodes the algorithms perform exceptionally well. This performance implies that using pre-initialised states can

speed up convergence. However, the reinforcement learning agent fails to maintain the strong performance in the first few episodes and ultimately uses a policy which is worse than if Q-values are not set previously.

As the test results presented above show, pre-initialised Q-values alone are not sufficient to solve the general problem of finding an appropriate policy. Therefore the next section will consider the approach of reducing the number of turns played to obtain a better state space coverage.

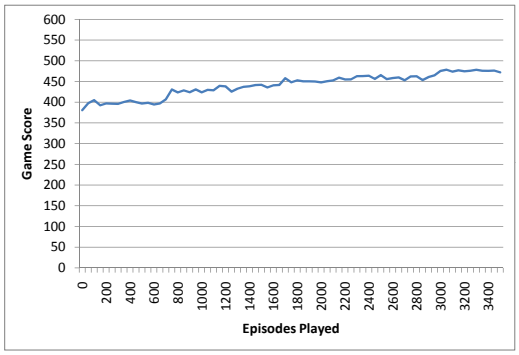
6.2.5 Test Runs with a Reduced Number of Turns

Since both a variation of core algorithm parameters and the use of pre-initialised Q-values did not produce the desired results, the approach of running experiments for a reduced number of turns was taken. This approach automatically leads to less cities being founded, and thus limits the state space. Furthermore, since this approach only takes into account the early game, it tremendously speeds up the test runs and makes possible experiments with more than double the number of episodes. On the other hand it also leads to the players having lower scores on average. Consequently, it makes trends in the development of average score less noticeable. The number of turns chosen for the tests was 60. Test runs of length 60 turns limit the observed phase in the game to half of the initial settling phase. The initial settling phase is the phase during which the players divide the continent between them through founding cities. The expected number of cities for games of length 60 turns is between two and three. The number of episodes that is played per experiment was set to 3500, i.e. more than double the number of episodes of previous experiments. Table 6.5 shows a summary of these parameter settings.

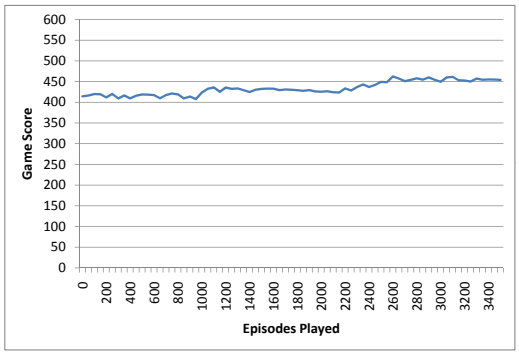
Learning Rate α	0.2
Discount Rate γ	0.6
Exploration Rate ϵ	0.4
Trace Decay Rate λ (for Sarsa(λ) and Q(λ))	0.8
Initial Q-Values $Q(s, a)$	0
Number of Turns	60 (4000BC - 1600BC)
Number of Episodes	3500

Table 6.5: Parameter Settings for Test Runs with Reduced Length

Figures 6.7 and 6.8 show the results of those test runs.

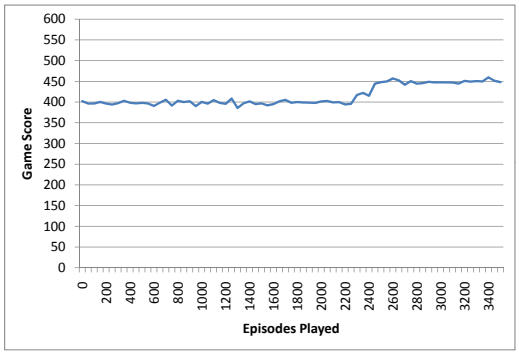


(a) Q-Learning

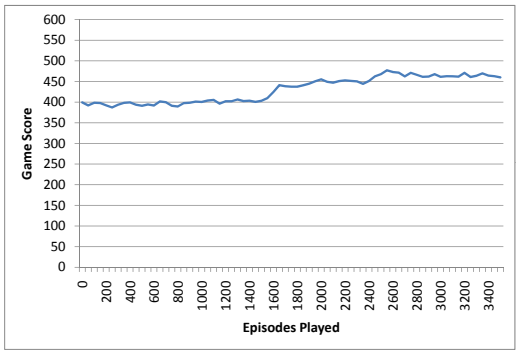


(b) Sarsa

Figure 6.7: Results for Short Runs using the One-Step Versions of the Algorithms



(a) Q(λ)



(b) Sarsa(λ)

Figure 6.8: Results for Short Runs using the Eligibility Trace Versions of the Algorithms

The diagrams show a clear tendency towards improving the game score. They therefore improve the used policies for selecting city sites. All four algorithms exhibit a tendency of improving the average score over time. However, their behaviour differs in the way and speed of improving game score. The diagrams display the results of 10 separate runs over 3500 episodes each.

The performance of the four algorithms in this experiment is exemplary of how the algorithms perform at the task of city site selection in general. Therefore the next section compares the performance of the four different algorithms and elaborates on how the results can be interpreted.

6.3 Comparison of Reinforcement Learning Algorithm Performance

This section elaborates on the results of the test runs of length 60 turns that are shown in Figure 6.7 and Figure 6.8. The focus will be on measurable and observable features like the speed of convergence towards higher scores, the average score throughout the experiments and the range of variations of the score during the run of an experiment.

One-step Q-learning shows a consistent performance. Despite the high percentage of 40% random decisions the average score rises by nearly 25% during the run of the experiment. There are several distinct steep rises of the average score. They mark the points where one of the contributing runs learned a noticeably improved greedy policy. The improvement thus is not completely linear, but there is a continuous improvement over time.

Sarsa improves as well but it mostly does so through distinct rises at certain times. Between these distinct improvements the average score declines since unlike Q-learning Sarsa does not pick the maximum possible future reward but strictly follows its ϵ-greedy policy, i.e. it sometimes picks future rewards at random. In this particular case this policy leads to a lower score on average. The following section will explore if this is also the case when the agent tries to play optimally.

Both of the algorithms that use eligibility traces, Q(λ) and Sarsa(λ), exhibit an even more distinctive version of the same behaviour as Sarsa. Their major gains are distinctive rises in average score. Unlike the one-step versions of the algorithms they show neither growth nor decline between thee steep gains but only a volatile up and down around an average value. It is noteworthy that the steep inclines (two in the case of Sarsa(λ), one nearly continuous incline in the case of Q(λ)) are markedly steep despite the diagrams in Figure 6.8 being an average of 10 runs. The inclines mark the discovery of a better greedy policy. Their markedness in a diagram that shows an average over multiple runs implies that they always happen at nearly the exact same point in time. This is due to the fact that apart from the random exploration that Sarsa contains, the algorithms always behave in the way described in Section 6.2.2. They pick a random policy at first and then slowly improve this policy. The steep inclines in average score mark the points where such an improvement takes place.

A major characteristic of the performance of an algorithm is the number of episodes it takes for the algorithm to overcome this initial random policy and replace it with a better policy. An important factor influencing this characteristic is the quality of the initially chosen random policy. However, it can be ignored in the test runs since all algorithms seem to be performing at roughly the same levels initially (around a game score of 400 on average).

The results of the phenomenon of algorithms being stuck with their initial policy have been discussed in Section 6.2.2. If all actions are attributed with the same reward at the beginning, it depends on the implementation which one is chosen. Since one aim of the test runs was to compare algorithms actions were not picked at random from all actions leading to the same reward but the last one that is found was picked. When comparing the Q-values of available actions in a state s, the most recent action a with a $Q(s, a)$ that ties for the highest value will thus replace a previous action of the same value.

The problem of algorithms converging towards a non-optimal policy at the beginning is illustrated in Figure 6.9. The figure shows the sequence of events that is described above with some example reward values. At the bottom of the figure the probability of being chosen is given for the different actions. With the current ϵ-greedy policy the probability for picking an action $a \in \mathcal{A}$ that is available in the current state is

- $\frac{\epsilon}{|\mathcal{A}|}$ for picking a random action, i.e. performing an explorative move and
- $(1 - \epsilon)$ for picking the greedy action, i.e. performing an exploitative move.

In the current experiments $\epsilon = 0.4$. The number of possible actions is equal to the number of possible plots that can be settled, i.e. 214 for the initial level of the state space on the chosen map. For the actual probabilities this means that any random action has the probability $\frac{0.4}{214} \approx 0.0019$ to be picked. The greedy action is picked with probability 0.6 which means that the greedy action is picked approximately $0.6/0.0019 = 316$ times as often as any other action. This leads to the conclusion that the estimate for the random action which is picked first will approximate the actual reward it receives very quickly, regardless of the value of the learning rate α is.

When computing a Q-value in a simplified version of the update function for Q-learning where both future and current rewards are considered as one variable r this means that

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * (r - Q(s_t, a_t)).$$

If the values are initialised at 0 and a value update is performed not only once but n times, the Q-value will be

$$Q(s_t, a_t) = \alpha * r * \sum_{i=0}^{n} (1 - \alpha)^i. \tag{6.1}$$

This is a simplification of the usual case in which the reward signal varies between different value updates. However this simplification allows to calculate how many episodes it will take to replace the action a' that was selected first and where the according Q-value approximated

the reward. As an example it is assumed that there is a different action a_* that yields 1.5 times the reward r of the action $a^{'}$ that was chosen first. This is a comparably large value if the action that was chosen first does not perform very bad. The currently used learning rate is $\alpha = 0.2$. The wanted value is the number of times n this action a^* has to be chosen until the according Q-value $Q(s_t, a_t^*)$ becomes bigger than $Q(s_t, a_t^{'})$.

$$0.2 * (1.5 * r) * \sum_{i=0}^{n} (0.8)^i > r$$
$$\Rightarrow \sum_{i=0}^{n} (0.8)^i > \frac{10}{3}$$

This equation can be resolved for $n \geq 4$, i.e. including the update for $n = 0$, $Q(s_t, a_t^*)$ would have to be updated 5 times before $Q(s_t, a_t^*) > Q(s_t, a_t^{'})$. Considering the number of possible actions $|\mathcal{A}| = 214$ and the exploration rate of $\epsilon = 0.4$, this would mean that on average $\frac{214}{0.4} * 5 = 2675$ episodes would be required to replace the first, non-optimal action $a^{'}$ with a^* when following a greedy policy. This is despite the fact that a^* yields a much higher reward and despite ignoring future reward, which would further aggravate the problem.
The calculations above further illustrate why the initial test runs did not produce any viable results. However, the calculation in equation 6.1 also shows how improvements to the algorithm's performance could be possible. If the value of the learning rate α was changed, this would directly influence the number of episodes that are required until a non-optimal action is replaced by an action that yields a higher reward. Through choosing a higher learning rate it should thus be possible to speed up convergence towards the optimal policy. This approach is evaluated in Section 6.5.

A comparison of the algorithms by speed of convergence shows that Q-learning is the most continuous while not reaching the highest value. The average score of the Q-learning algorithm starts to improve from the very beginning. Its curve only shows minor steep improvements. One-step Sarsa is the algorithm second to overcome the initial random policy. However, it does not manage to achieve the same average score level as one-step Q-learning. This observation can be attributed to a second significant policy improvement that Q-learning achieves while Sarsa uses a policy that has only improved once. Both of the algorithms that use eligibility traces also show two clearly marked policy improvements. For Sarsa(λ) these improvements happen roughly after one third and after two thirds of the number of episodes played. For Q(λ) both improvements happen within a very short time after about two thirds of the total number of episodes. The results section presented in this demonstrate that the settings used in this section allow the reinforcement learning agent to effectively learn better policies. The next section describes the comparison of the performance of the reinforcement learning algorithms with the standard game AI.

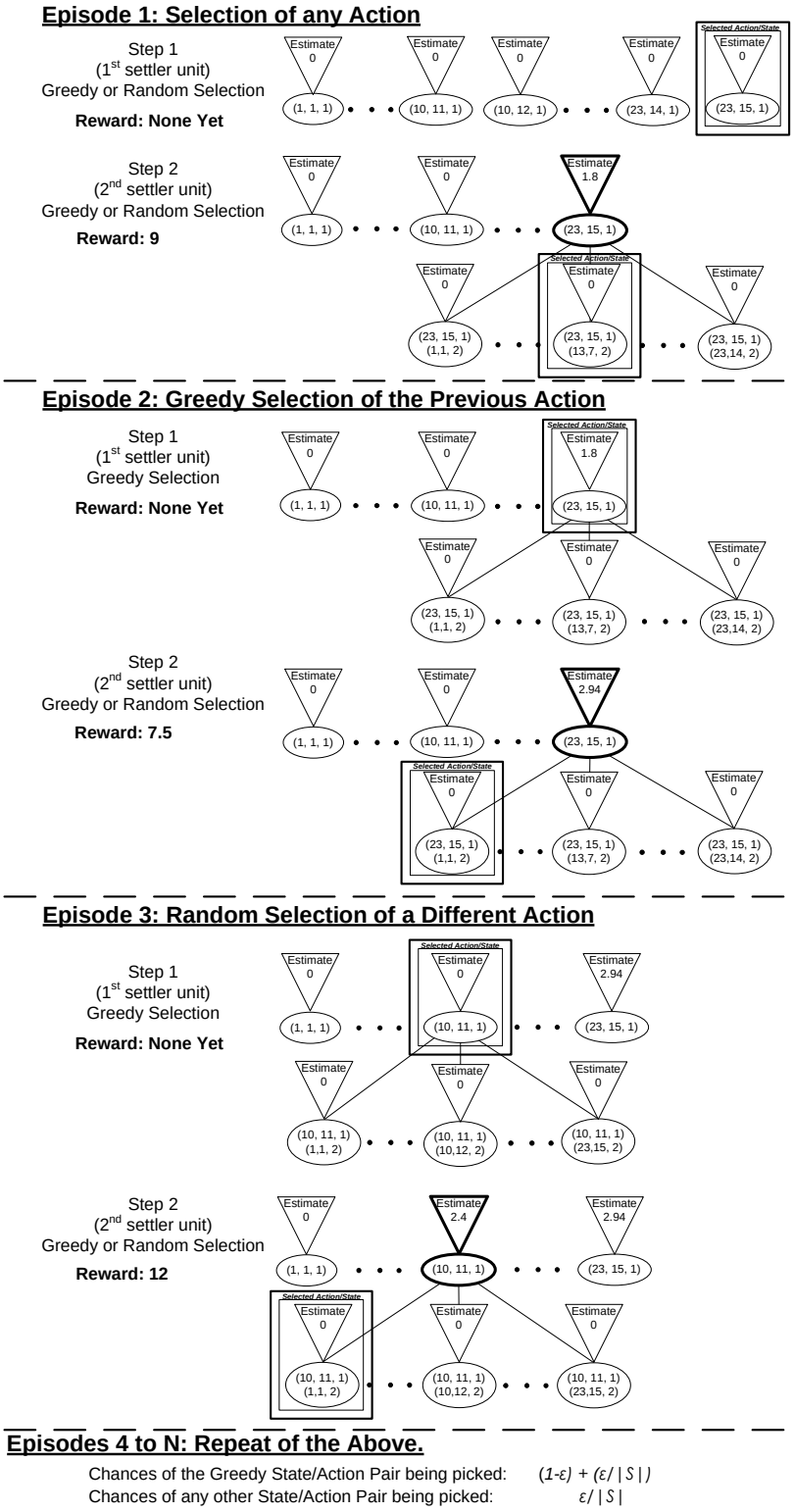


Figure 6.9: Illustration of Early Algorithm Conversion

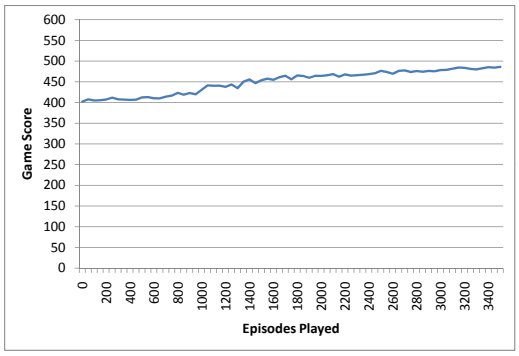
6.4 Comparison of the Standard Game AI with Reinforcement Learning

In this section the performance of the reinforcement learning algorithms is compared with the performance of the standard game AI. As demonstrated in the previous section, all four reinforcement algorithms implemented are able to learn new policies for city site selection. However, their performance was skewed because some actions were selected at random. Consequently, random selections are eliminated to enable reliable comparison. In order to achieve elimination of random selection, a declining ϵ-greedy policy (Section 5.1)instead of the standard ϵ-greedy policy is used to pick actions. The reinforcement learning agent starts at an exploration rate ϵ of 60%. The exploration rate is subsequently reduced by 0.6/3500 in every episode, i.e. after 3500 episodes the policy is completely greedy. Table 6.6 shows the settings for these test runs.

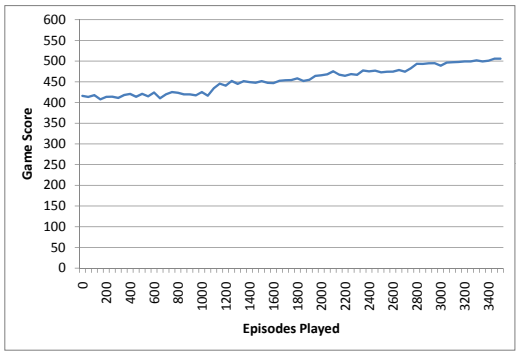
Learning Rate α	0.2
Discount Rate γ	0.6
Exploration Rate ϵ	Between 0.6 and 0.0 Declining by 0.6/3500 (0.017%) per Episode
Trace Decay Rate λ (for Sarsa(λ) and Q(λ))	0.8
Initial Q-Values $Q(s, a)$	0
Number of Turns	60 (4000BC - 1600BC)
Number of Episodes	3500

Table 6.6: Parameter Settings for Comparison of Reinforcement Learning with the Standard AI

The results of running the four algorithms individually with these settings are shown in Figure 6.10 and Figure 6.11.

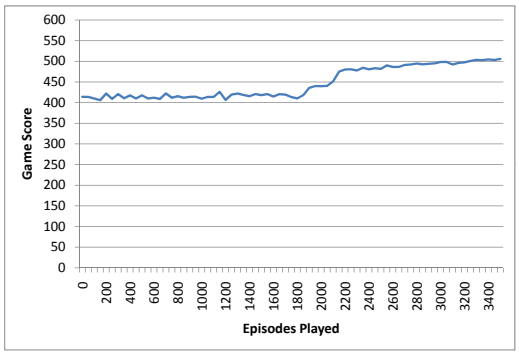


(a) Q-Learning

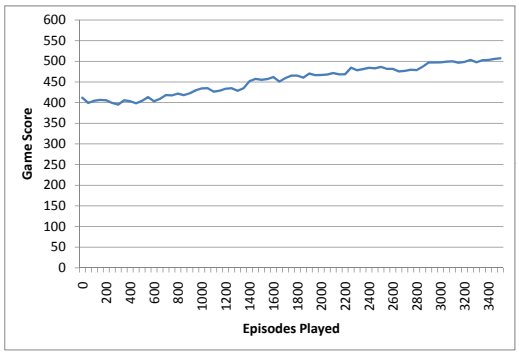


(b) Sarsa

Figure 6.10: Results for Declining ϵ-Greedy using the One-Step Versions of the Algorithms



(a) Q(λ)



(b) Sarsa(λ)

Figure 6.11: Results for Declining ϵ-Greedy using the Eligibility Trace Versions of the Algorithms

As expected the average score is even higher than the average score in the previous test runs in Section 6.2.5. The random selection of actions is continually reduced as the number of episodes increases. Thus the number of greedy selections increases. The curve of the average game score shows a slight continuous growth for all four algorithms. Apart from the additional growth, the algorithms exhibit the characteristics that have been described in Section 6.3. For the one-step algorithms Q-learning shows a very consistent growth with nearly no observable

steep increases in average game score while Sarsa has a slightly less marked growth but has two distinctive increases. Both of the algorithms that use eligibility traces show two strong increases in average game score. For Q(λ) the increases happen in close succession whereas for Sarsa(λ) they are more distributed and hidden by a more consistent increase over time. All four algorithms improve by about 25% over 3500 episodes.

The standard game AI always picks its city sites according to a greedy policy. If there is no random event involved, standard game AI will always perform the same actions. As a result, if two players controlled by the standard AI are playing each other and there are no crucial battles (one of the few random events in Civilization IV) or attacks by randomly spawned barbarians (switched off for the tests carried out), the course of the game will be exactly the same for every single episode.

To obtain the average score of the standard game AI, it has to play under exactly the same conditions as the reinforcement learning agent. If the standard AI plays under exactly the same conditions as the reinforcement learning agent, the experiment shows that for games with length 60 turns the AI will always gain the same score in every single game. Figure 6.12 shows the comparison of the performance of the standard AI with the results of the reinforcement learning algorithms.

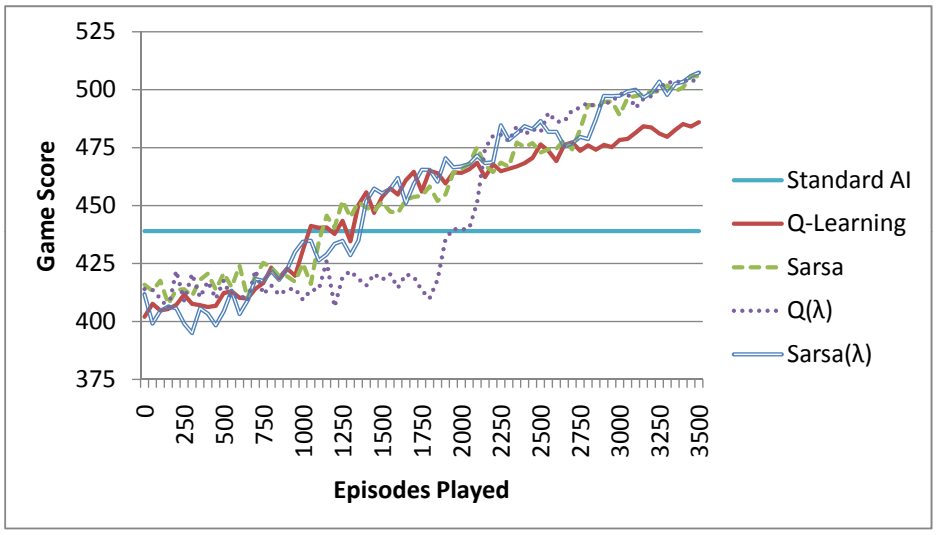


Figure 6.12: Performance Comparison between Standard AI and Reinforcement Learning Algorithms

The difference between the achieved game score of the reinforcement learning algorithms and the standard AI is remarkable. The reinforcement learning algorithms start at an average score of about 400 and improve to an average score of about 500 with Q-learning achieving slightly less.

The standard game AI, which follows a greedy policy from the very beginning, achieves a score of 439. The reinforcement learning algorithms thus perform between 10% and 20% (depending on the algorithm) better than the standard AI after learning for 3500 episodes. However, this does not imply that the reinforcement learning algorithms have found the optimal policy already. As shown in the previous section, the current parameter settings might enable the reinforcement learning algorithms to learn better policies, but there is evidence that they still can be improved.

6.5 Optimising the Learning Rate α

A change to the algorithm parameters that can possibly improve the results is an increase of the learning rate α. This change is suggested by Equation (6.1) in Section 6.3. Experiments were performed with $\alpha = 0.5$ and $\alpha = 0.8$. All other settings, including the declining ϵ-greedy policy, remained the same. The results of these experiments are shown in figures 6.13 to 6.16. Each diagram compares the results for the different settings of α using one of the reinforcement learning algorithms.

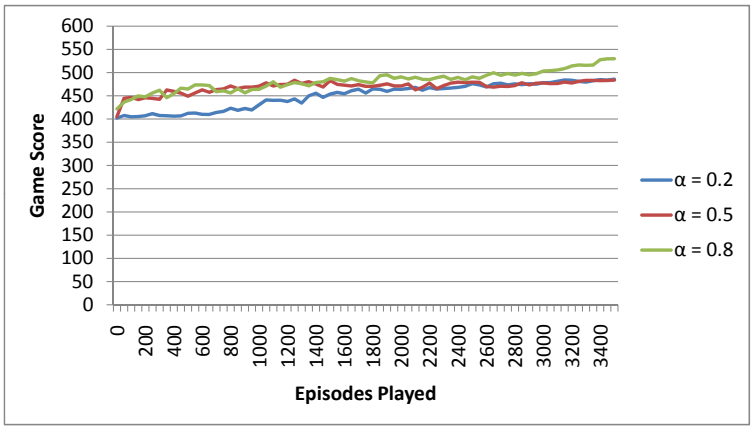


Figure 6.13: Comparison of Different Learning Rates using One-Step Q-Learning

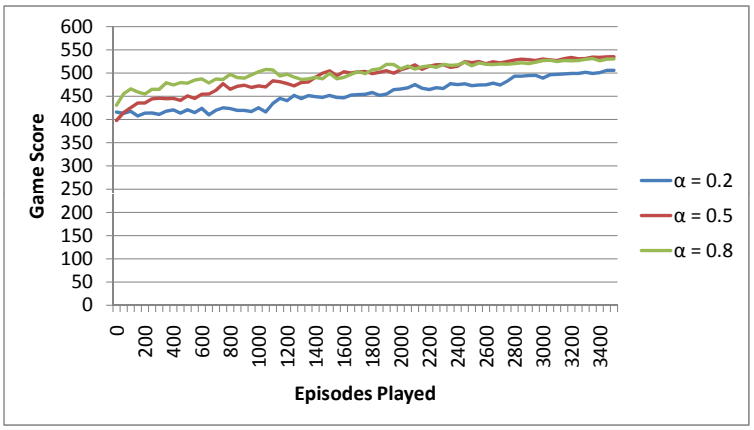


Figure 6.14: Comparison of Different Learning Rates using One-Step Sarsa

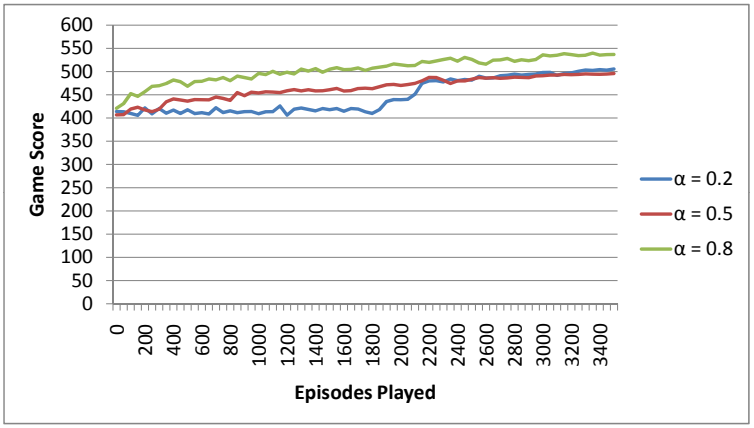


Figure 6.15: Comparison of Different Learning Rates using Q(λ)

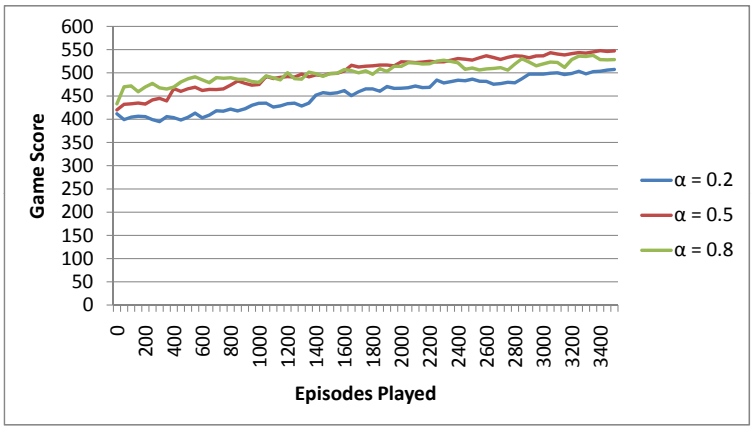


Figure 6.16: Comparison of Different Learning Rates using Sarsa(λ)

The diagrams exhibit several interesting features when comparing the results for a learning rate of 0.2 with those of a learning rate of 0.5. None of the algorithms performs worse with increased learning rate compared to the previous lower α when they are judged according to the quality of the learned policy at the very end of a test run.

Both algorithms that are based on Q-learning end up with a similar final score for both settings of α. The Sarsa-based algorithms on the other hand manage to improve their performance by up to 50 score points, i.e. 10% of their overall average score. As predicted by the findings in Section 6.3, the score improves the most during the first 500 episodes of the experimental runs. While running with a setting of $\alpha = 0.2$ all four algorithms show no improvements or very little improvements in this phase. With a setting of $\alpha = 0.5$ however, there is a steady increase in average score right from the start. After the first 1000 episodes, the performance of the algorithms that run with $\alpha = 0.5$ is comparable to those running with a setting of $\alpha = 0.2$. It is noteworthy that despite the increased learning rate all four algorithms show variations that are less marked than before. This behaviour is striking since the Q-values were expected to fluctuate more due to larger changes in the values at every update.

The results for a setting of $\alpha = 0.8$ differ again between the algorithms that are based on Q-learning and those that are based on Sarsa. The average performance of one-step Sarsa and Sarsa(λ) is nearly identical for $\alpha = 0.5$ and $\alpha = 0.8$ in terms of game score. However, the average game score fluctuates a lot more for $\alpha = 0.8$. Also the score that is achieved when following the final policy that is found, i.e. the greedy policy after 3500 episodes, is slightly worse for $\alpha = 0.8$ for both algorithms. The fluctuation with $\alpha = 0.8$ is also worse than before for one-step Q-learning and Q(λ). However, it is less strong than for the Sarsa-based algorithms. Furthermore, the algorithms based on Q-learning do perform far better for $\alpha = 0.8$. Both one-step Q-learning and Q(λ) perform up to 50 score points, i.e. 10% of their overall average score better with $\alpha = 0.8$ than with $\alpha = 0.5$. This increase of 10% is similar to what the Sarsa-based algorihms gained by increasing α from 0.2 to 0.5. Consequently, on-policy and off-policy algorithms require different settings for α in order to perform optimally.

This chapter started by elaborating on the experimental settings for the game environment as well as for the parameters for the different reinforcement learning algorithms. These settings were then tested and refined in preliminary test runs. The results of these test runs were analysed and different improvements to the settings were made in order to improve the learning process for the algorithms. Once settings were found that showed promising learning capabilities, the performance of the four reinforcement learning algorithms was compared to the performance of the other reinforcement learning algorithms. In the final section the performance of the reinforcement learning algorithms was compared to that of the standard game AI.

In the following chapter the results of the experimental runs and of the whole thesis will be discussed. Furthermore, possible future work building on the results of the research in this thesis will be discussed.

Chapter 7

Discussion and Future Work

This chapter elaborates on the findings of the experiments discussed in the previous chapter. Furthermore, the general results of this thesis are discussed and possible improvements are mentioned.

The empirical evaluation carried out in Chapter 6 shows the potential of applying reinforcement learning to select the best plots for founding cities in Civilization IV. Several adjustments to the preliminary settings had to be made before obtaining satisfactory results. They were necessary because of the specific requirements of the chosen task and especially of the chosen test environment. Particularly the memory constraints of Civilization IV as well as the decreasing game speed in the later phases of the game make long-term evaluations time consuming and impractical. Although this can partly be compensated for by choosing the appropriate game settings, the attempts at finding settings that make it possible to evaluate the complete settling phase were not successful.

The application of fully discounted reinforcement learning algorithms that completely ignore previous estimates was not successful. Failure of fully discounted reinforcement learning algorithms was to be expected because it ignores one of the core elements of TD learning. No bootstrapping takes place in the application, i.e. previous estimates are no longer used to create new estimates. Therefore, the reinforcement learning agent loses its 'memory'.

The second attempt at adapting the settings for test runs by using pre-initialised Q-values did not produce an improvement in developing new policies either. However it is promising for future applications: The idea of using existing domain knowledge to initialise the Q-values of state-action pairs can be used if the values that are used are adjusted to better fit the purpose. For instance the current implementation does not consider an opponent's territory because the values that are used are computed before anybody has claimed any territory. One possible approach would be to not completely rely on the pre-initialised value when picking the next founding plot, but to only use these values when there is a draw between two plots with equal value.

The ensuing decision to reduce the number of turns that are played per episode makes the learning process easier to observe. The reduction in game reduces the state space because fewer cities are built. It also limits the number of possible actions of both the reinforcement learning agent and the standard game AI. The reduction in average game score means that it is easier to distinguish between a real improvement and a random event. It thus facilitates evaluation of the true performance of an algorithm because it reduces the 'noise' hiding the true performance of the algorithms. In the first three experiments the episodes are 110 turns long and the noise manifests itself in the strong variations in average game score. The variations also occur in the 60-turn episodes but they are far less distinctive.

Although the reduction in length led to very good results, the extension of the time span analysed is still part of future improvements. Since the ultimate goal is to create a reinforcement learning agent that can play the complete settling phase, both the length of the games and the number of episodes used for the experiments will have to be extended in the long run. However, it would not be useful to extend the learning process to a whole game. In later stages only few cities are founded but there is a large number of cities that is won or lost through combat. The fluctuating number of cities would obscure the actual score that is gained by newly founded cities and make learning according to the current reward signal all but impossible. With the length of one episode adjusted to 60 turns, all four reinforcement learning algorithms used improved their policies for selecting the best plots to found a city on by up to 40% over the standard game AI. Considering the short playing time, this improvement is very remarkable. As stated in Section 5.2.4 where the different components of the game score are described, a higher score can result in a number of improvements for the reinforcement learning agent. Since the difference in score is big in the case of these experiments, the reinforcement learning agent will have a number of advantages over his opponent.

- The reinforcement learning agent has more population or more cities than its opponent.
- The reinforcement learning agent has a larger territory than its opponent. This means that the reinforcement learning agent has access to more special resources that increase the yield in production, food and commerce.
- The reinforcement learning agent has more advanced technologies than its opponent, leading to a number of benefits such as stronger combat units and higher yield from plots.
- The reinforcement learning agent has more Wonders of the World (special, very expensive buildings) that can have numerous effects that further benefit the reinforcement learning agent's empire.

Figure 7.1 shows a summary of the best results of the four algorithms with their optimal settings from the empirical evaluation.

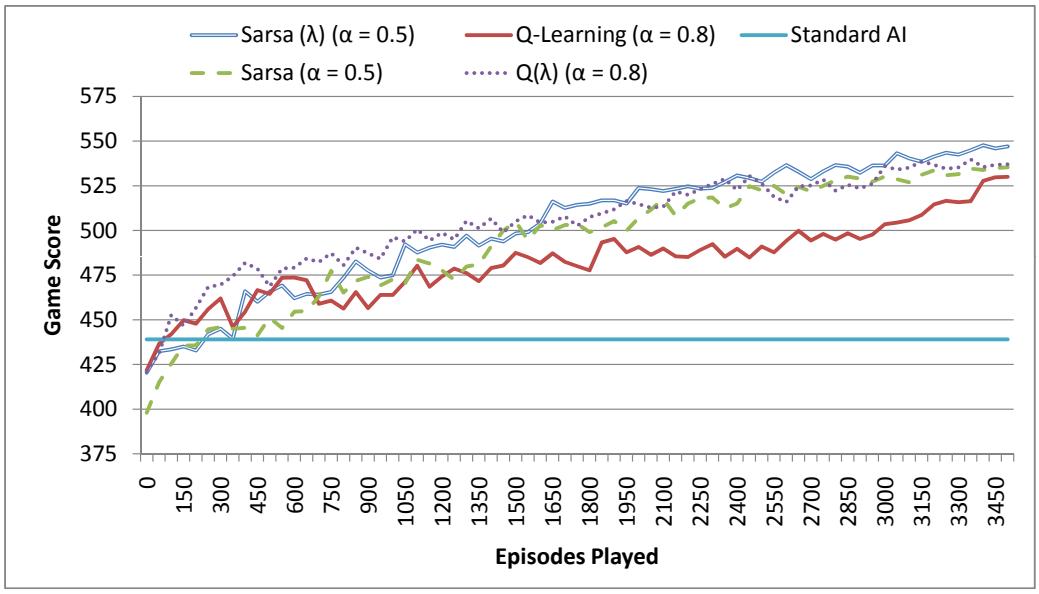


Figure 7.1: Performance Comparison between Standard AI and Reinforcement Learning Algorithms with Optimal Settings

Both algorithms that use eligibility traces show a better performance than their one-step counterparts. This was to be expected due to the nature of the task. City site selection relies heavily on reward propagation. Due to this heavy reliance, even in games that only last 60 turns the improvement through eligibility traces becomes apparent. The comparison also shows that the on-policy Sarsa algorithms perform better than the off-policy Q-learning algorithms. One-step Q-learning is the weakest algorithm in terms of final score and also exhibits the strongest fluctuations in its average score during the learning process. One-step Sarsa performs at about the same level as Q(λ) but Q(λ) has a stronger and more stable performance earlier during the experiment. Sarsa(λ) shows the strongest performance by a margin of about 3% when compared to Q(λ) and one-step Sarsa. Since Sarsa(λ), just like Q(λ), also exhibits a consistent growth, with the current settings Sarsa(λ) is the best algorithm for the task of city site selection in Civilization IV.

Through a number of changes to the experimental setup further improvements could be achieved. The first setting to be changed is the aforementioned extension of the number of episodes. In several experiments the increase in average game score can be attributed to a large degree to two distinct rises at two points in time. This principally means that the greedy policy that an agent pursues was improved greatly at those two points in time. However, it is unlikely that two improvements already lead to the optimal policy π^*. Therefore, given a suitable number of episodes, it can be assumed that there will be additional improvement of the score.
The second setting that could be changed in order to obtain better results is the number of turns an episode takes. Since cities need time to develop their full potential, the gap in score between good and mediocre cities widens with the number of turns played.

The mediocre performance of the standard game AI and especially the complete lack of variation in its performance were additional interesting discoveries of the experimental runs. These findings confirm a general premise of this thesis: Despite its apparently sophisticated and critically acclaimed AI (BetterAI Project, 2007) Civilization IV has a very simplistic and mostly static AI. If there is no human player to add randomness to the gameplay, the game is close to being completely deterministic. In the final experiment the reinforcement learning agent gradually approximates a completely greedy and thus deterministic policy. If the reinforcement learning agent were to be integrated into the game the exploration rate ϵ would have to be prevented from reaching 0 in order to keep the game more interesting and versatile. However, for the experiments carried out in this thesis, using a completely greedy policy does make sense since the optimal performances of the algorithms and the standard AI were compared.

Possible extensions and additions to the reinforcement learning agent in Civilization IV can be divided into four categories. These categories are the improvement of the four algorithms which have been used, the evaluation of other reinforcement learning approaches, and the combination of other machine learning techniques with reinforcement learning. Moreover, the task in Civilization IV that is solved through reinforcement learning can be extended.

The results of the empirical evaluation show that the applied reinforcement learning methods have the potential to outperform the static standard AI. However, learning becomes computationally unfeasible when crossing a certain threshold of game complexity, either in terms of game turns or of map plots. Consequently, the algorithms have to be optimised further to extend the range of tasks for which the reinforcement learning agent can be used, i.e. for longer games or larger maps. One way to optimise the algorithms is to speed up the learning process to accelerate the convergence towards the optimal action-value function. The two algorithms that use eligibility traces are a first step into this direction.

They could further improved by changing the standard policy that is used to select actions from an ϵ-greedy policy to a softmax policy (Section 5.1.4). If the policy is changed to a softmax policy some issues arise. Softmax policies are a double-edged sword. On the one hand they can remedy the problem that actions that eventually yield higher rewards are not picked since another random action was picked first and now has become part of the greedy policy (see Section 6.2.2). On the other hand a softmax policy could aggravate the exact same problem since it less often choses actions that have not yet been yet. Consequently, a softmax policy only improves the algorithm if an action that yields a higher reward than the current greedy action has already been picked at least once. In spite of this problem, softmax policies should be evaluated and their performance compared to that of the currently used ϵ-greedy policy.

Apart from the extension of the existing methods, future research can be carried out in the field of other reinforcement learning techniques: Monte Carlo methods or pure dynamic programming can be evaluated as to how well they are suited for the city placement selection task (Sutton and Barto, 1998).

The most promising technique when it comes to solving the problem of dimensionality would be the use of function approximation. This form of generalisation is used in several of the publications on reinforcement learning and it is particularly suitable to handle problems of big state spaces. Function approximation was considered at the very beginning of this thesis. However, function approximators are problematic when it comes to convergence (Sutton, 2000). Therefore, a table-based approach was chosen to prove the feasibility of using reinforcements learning algorithms to select city sites. Another major drawback of function approximation is its dependency on an accurately chosen representation (Sutton and Barto, 1998). Based on the knowledge gained in this thesis and particularly in the empirical evaluation, function approximation could now be attempted.

An additional field for future research on reinforcement learning using Civilization IV as a test bed is the combination of other machine learning methods with reinforcement learning. One particularly promising method is Case-Based Reasoning (CBR). The application of CBR to the plot selection task would allow to solve the problem of experience learned on one map being useless on another map because of the different topology. A combination of reinforcement learning and CBR has successfully been applied to a similar problem before. Auslander et al. (2008) solve the problem of transferring knowledge acquired through reinforcement learning from one type of behaviour of the opponents to a different behaviour in the FPS Unreal Tournament. Their algorithm CBRetaliate manages a case-base of policies for a number of agents that have been learned through Q-learning. The case-base is used to speed

up the process of creating winning strategies through reinforcement learning in a changing environment.

A possibility to improve the game play experience would be the application of Motivated Reinforcement Learning (MRL). One of the game mechanics in Civilization IV is the aforementioned 'character traits' (Section 6.1.2) of the different computer AIs. Certain players by definition have advantages in certain areas of the game for example in the fields of expansion, finance or combat. These advantages are hard coded numbers and are meant to express certain character traits like aggressiveness or expansionism. Instead of hard coded numbers the agents could use a reinforcement learning agent which is rewarded by a motivational function as described by Merrick and Maher (2007). This could lead to more interesting, human-like behaviour for AI players.

Another possibility to build on the research carried out for this thesis is to extend the task for which these machine learning techniques are used. For the purpose of the present thesis, the task of the reinforcement learning agent has been limited to the choice of site for building a city. For future development, the task could be extended to include the choice of the optimal number of cities and when to build them.

This chapter elaborated on the general results of the evaluation. The results of the experiments experiments that were performed were discussed and the implications of their outcome was illustrated. Possible future extensions were discussed for the test runs as well as for the algorithms and the testbed integration. Furthermore, the application of other reinforcement learning techniques for the overall task was considered as was the integration of reinforcement learning with other machine learning techniques.

The next chapter will summarize the results of the thesis and draw a conclusion on its achievements, success and and possible improvements of reinforcement learning agents in Civilization IV.

Chapter 8

Conclusion

This section concludes the thesis. The premises as well as the design and implementation of the reinforcement learning agent are summed up first. Then the results and contributions of this thesis are summarized.

This thesis has presented the integration of reinforcement learning into the commercial computer game Civilization IV. Civilization IV was chosen as a testbed because it is accessible, recent and critically acclaimed as a commercial product. The task that was chosen for the integrated reinforcement learning algorithms is the selection of the plots on the map that are used to found cities on. The particular task was selected for being comparably easy to observe while at the same time being crucial to the game. The reinforcement learning model, i.e. the states, actions and the reward signal are defined to contain the Markov property. They constitute a finite Markov decision process.

Four different reinforcement learning algorithms have been adapted to the task of city site selection and have been integrated into the game environment. All four of them are based on TD learning. Q-learning and Sarsa are one-step versions while Q(λ) and Sarsa(λ) work with eligibility traces. By using Q-learning and Sarsa, both on-policy and off-policy algorithms are tested.

After establishing proper settings for the test runs, algorithms were evaluated empirically. The results of the empirical evaluation were promising. All four algorithms that were used manage to learn city placement strategies for a specific map. After undergoing a training process, the reinforcement learning agent outperforms the standard game AI in short matches by up to 40%, depending on the algorithm used. The standard AI is comparatively weak in its general performance. Moreover, it behaved in a completely deterministic way, which is one of the major drawbacks that machine learning can improve upon in commercial games. The integration of the reinforcement learning agent into Civilization IV thus extends the otherwise deterministic early game by a non-deterministic, adaptable component that enhances the game-playing experience.

The findings of the initial experiments also point out possible shortcomings of the algorithms. They thus lay the groundwork for future work. Several possible extensions that are mentioned in Chapter 7 could further improve the performance of the algorithms and extend the reinforcement learning agent's capabilities and responsibilities.

Another contribution of this thesis to the field of computer game AI research is the use of the commercial computer game Civilization IV as a testbed for AI research. Civilization IV has great potential and can be used for other machine learning experiments in the future because of the diversity of the tasks involved in the game and the relative ease with which these deterministic tasks can be taken over by learning agents. The findings of experiments that were run with Q-learning were published in (Wender and Watson, 2008))(see Appendix A). The second part of the empirical evaluation of this thesis is based on the findings in the paper and further improved and extended the results by using different algorithms and better settings.

As a general conclusion it can be said that the objectives that were defined in the beginning have been achieved. The objectives were to integrate the reinforcement learning agent into Civilization IV and to find settings for the agent that enable it to perform at or above the level of the standard game AI. Furthermore these settings were to be optimised to allow the reinforcement learning agent to perform at the highest level possible.

Since all these objectives have been achieved, the application of reinforcement learning to select city sites in Civilization IV has been successful.

Appendix A

Paper Presented at the 2008 IEEE Symposium on Computational Intelligence and Games (CIG'08)

Using Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV

Stefan Wender, Ian Watson

***Abstract*— This paper describes the design and implementation of a reinforcement learner based on Q-Learning. This adaptive agent is applied to the city placement selection task in the commercial computer game *Civilization IV*. The city placement selection determines the founding sites for the cities in this turn-based empire building game from the *Civilization* series. Our aim is the creation of an adaptive machine learning approach for a task which is originally performed by a complex deterministic script. This machine learning approach results in a more challenging and dynamic computer AI. We present the preliminary findings on the performance of our reinforcement learning approach and we make a comparison between the performance of the adaptive agent and the original static game AI. Both the comparison and the performance measurements show encouraging results. Furthermore the behaviour and performance of the learning algorithm are elaborated and ways of extending our work are discussed.**

I. Introduction

One of the main incentives for integrating machine learning techniques into video games is the ability of those techniques to make those games more interesting in the long run through the creation of dynamic, human-like behaviour [1]. Among the most captivating games, especially in terms of long-term gameplay, are the games of the *Civilization* series. However these turn-based strategy games achieve their high replay value not through advanced adaptable AI techniques but through a high level of complexity in the later stages of the game. The early stages however are, as we will also see in this paper, mostly deterministic and therefore not very challenging. These characteristics as well as the large number of tasks involved in playing the game make *Civilization* games an ideal test bed where one of those many tasks can be replaced by a machine learning agent, thus making the AI less predictable and improving the overall game play experience.

II. Related Work

The machine learning method we chose for our task is Reinforcement Learning (RL) [2], a technique which allows us to create an adaptive agent that will learn unsupervised while playing the game. More specifically the Q-Learning algorithm as introduced by [3] will be used to demonstrate the applicability of reinforcement learning in the commercial video game *Civilization IV*.

Because of the broad spectrum of problems involved in Civilization video games as well as the multitude of versions of the games that are available, several of them with open source code, multiple variants of the game have been used in academic research. Perhaps most popular as a test bed is the Civilization variant *FreeCiv*, an open source version of the commercial game *Civilization II*. *FreeCiv* has been used to show the effectiveness of model-based reflection and self adaption [4]. Furthermore an agent for *FreeCiv* has been developed that plays the complete early expansion phase of the game [5].

The development of an AI module that is based on Case-Based Reasoning (CBR) for the open source Civilization clone *C-Evo* is documented in [6]. *C-Evo* is an open source variant of Civilization, which is closest related to *Civilization II* and allows for the development of different AI modules which can compete against each other.

More directly related to this paper is research which uses the commercial Civilization games as a test bed. The commercial Civilization game *Call To Power II* (CTP2) is used as a test bed for an adaptive game AI in [7]. In order to communicate with the game an ontology for the domain is developed and case-based planning in combination with CBR is used to create an adaptive AI. CTP2 has also been integrated with the test environment TIELT [8], thus preparing a test bed for future research using CTP2.

In [9] a Q-Learning algorithm is used to create an adaptive agent for the fighting game *Knock'em*. The agent is initially trained offline to be able to adapt quickly to the opponent in an online environment. RETALIATE (**Re**inforced **Ta**ctic **L**earning **i**n **A**gent-**T**am **E**nvironments), an online Q-Learning algorithm that creates strategies for teams of computer agents in the commercial First Person Shooter (FPS) game *Unreal Tournament* is introduced in [10]. This approach is extended in [11], where the authors use CBR in order to get the original RETALIATE algorithm to adapt more quickly to changes in the environment.

III. Civilization IV as Test Bed for Research in Computer Game AI

The variant of the Civilization game which will be used as a test bed in this paper is *Civilization IV*. *Civilization IV* is the latest title in the commercial series of the original game. Large parts of its code base, including the part which controls the AI, have been released as open source. Also the existing computer AI is already quite sophisticated and thus can provide a challenging opponent in empirical experiments. Furthermore an improvement of the existing AI would show that research from academia can be used to create a bigger challenge and thus offer a more enjoyable playing experience which will in the end lead to better games in general.

Stefan Wender and Ian Watson are with The University of Auckland, Department of Computer Science, Auckland, New Zealand; e-mail: swen011@aucklanduni.ac.nz || ian@cs.auckland.ac.nz

However the use of *Civilization IV* as a test bed for research also bears a challenge. The code base that was released as open source consists of more than 100000 lines of code, which mostly are very sparingly commented. Since only parts of the source code have been released, several major functions have to be emulated, most importantly the automatic restarting of a game which is crucial when running tests that are supposed to last for several thousand games.

A. The Game

Civilization is the name of a series of turn-based strategy games. In these games the player has to lead a civilization of his choice from the beginnings BC to the present day. It involves building and managing cities and armies, advancing the own empire through research and expansion as well as interacting with other, computer-controlled civilizations through means of diplomacy or war in a turn-based environment. The popularity of the original game has lead to a multitude of incarnations of the game, both commercial and open source.

B. The City Placement Task

The most important asset in a game of Civilization IV are the cities. The three major resources a city produces are food (used for growth and upkeep of a city), commerce (used among others for research and income) and production (used to produce units and buildings). Furthermore special bonuses which grant additional basic resources or other benefits like accelerated building speed can be gained. The playing field in Civilization IV is partitioned into "plots" with each plot producing a certain amount of the resources mentioned above. A city can gain access only to the resources of a plot which is in a fix shape of 21 plots surrounding the city: Figure 1.

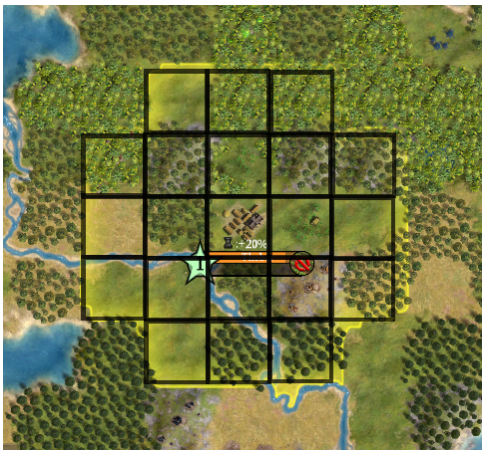


Fig. 1. Civilization IV: Workable City Radius

In addition the borders of an empire and thus the area of influence of its player are defined by the summarized borders of the cities of that empire. Therefore the placement of cities is a crucial decision and influences the outcome of a game to a large degree. This is the reason why we chose to apply RL to the city site selection task.

Cities are founded by mobile settler units which can be exchanged for a city on the plot they are located on. The standard AI uses a strictly sequential method of determining which are the best sites for building new cities. Each plot on the map is assigned a "founding value" which is based on numerous attributes like the position of the plot in relation to water, proximity of enemy settlements, proximity of friendly cities and of course the resources that can be gained from plots. We replace this sequential computation of founding values with reinforcement learning.

At the current stage of our research the task of the reinforcement learner is limited to choosing the best location for an existing settler. The decision of when to build a settler is still made by the original game AI.

IV. Reinforcement Learning Model

Reinforcement learning is an unsupervised machine learning technique, in which an agent tries to maximise the reward signal [2]. This agent tries to find an optimal policy, i.e. a mapping from states to the probabilities of taking possible actions in order to gain the maximum possible reward. The reinforcement learning model for the city placement task consisting of the set of states S, possible actions A and the scalar reward signal r is defined as follows:

A. States

A state $s \in S$ contains the coordinates of all existing cities of the active player. The other important information besides the position of a city is when this city was created. This information is crucial since a different order of founding can lead to very different results. Therefore, in order to satisfy the Markov property (i.e. any state is as well qualified to predict future states as a complete history of all past sensations up to this state would be) and thus for the defined environment to represent a Markov Decision Processes (MDP) each plot also contains the information when, in relation to the other cities, this city was founded.

A state $s \in S$ can be described as a set of triples *(X-Coordinate, Y-Coordinate, Rank in the Founding Sequence)* and each triple is representing one city. This definition of the states means that the resulting model will be a graph with no cycles, i.e. a tree. Figure 2 shows a part of such a tree with the nodes representing the states and branches representing the actions. Because of this structure of the state space, no state can be reached more than once in one episode.

The set of all states S consists therefore of all possible combinations of the *(X-Coordinate, Y-Coordinate, Rank in the Founding Sequence)* triples where cities can be built on any plot $p \in P$.The resulting size of the state space is

$$|S| = \sum_{i=0}^{c} \frac{|P|!}{(|P|-i)!} \tag{1}$$

With $c = |P|$ since every plot on the map could be a city.

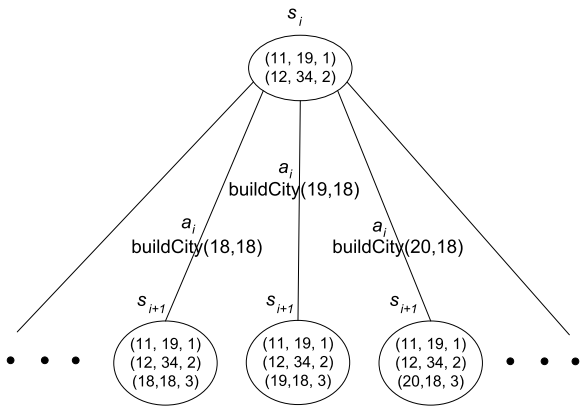


Fig. 2. Excerpt of the State Space S

B. Actions

The set A of possible actions which can be taken when in a state $s \in S$ consists of founding a city on any of the plots ($p \in P \| p \notin s$), i.e. any plot where there is no city of the active player yet. Since the map size of a game varies and for an average sized map there are $|P| = 2560$ plots already, this results in a very large state space. One measure we took to reduce this size significantly is to ignore ocean plots, as cities can only be founded on land. This reduces the number of possible plots to about one third of the map size.

C. Rewards

The reward signal is based on the score of a player. In the original game this score is used to compare the performance of the players with each other and it is updated every turn for all players. The game score consists of points for population in the cities, territory (the cultural borders of the cities added up) and technological advancement (developed with research output from the cities). Therefore, all the parts the game score is made up of are connected to the cities. A time step, i.e. the time frame in which the reinforcement learner has to choose an action and receives a reward after performing that action, is defined as the time between founding one city and founding the next city. The update of the Q-value $Q(s_i, a_i)$ after taking an action a_i in state s_i happens immediately before executing the next action a_{i+1}, i.e. founding the next city. The selection of the appropriate plot for this foundation however can happen several game turns before that, with the settler unit moving to the chosen plot afterwards. The scalar value which represents the actual reward is computed by calculating the difference in game score between the founding turn of the last city and the founding turn of this city. The difference is then divided by the number of game turns that have passed between the two foundations:

$$r \leftarrow \frac{(GameScore_{new} - GameScore_{old})}{(GameTurns_{new} - GameTurns_{old})}$$

V. Algorithm

The top-level algorithm which controls the overall city site selection task can be seen in Figure 3.

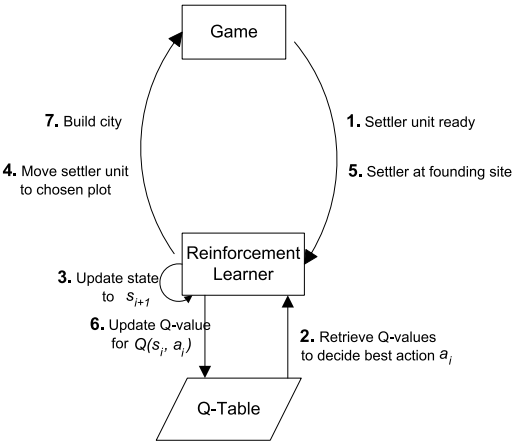


Fig. 3. Top-level Algorithm for City Site Selection

The actual Q-Learning algorithm which is based on *One-step Q-Learning* as described in [2] is shown below.

Initialise $Q(s, a)$ to 0
  Repeat for each episode:
    Initialise s
    Repeat for each step in this episode:
      Determine possible actions A_i in s_i
      Choose action $a_i \in A_i$ as
        I) $a_i \leftarrow max\ Q(s_i, a_i)$ with probability 1-ϵ
        OR
        II) $a_i \leftarrow random\ a_i \in A_i$ with probability ϵ
      Send settler unit to plot chosen in a_i
      $r \leftarrow gameScoreGainPerTurn()$
      $Q(s_i, a_i)$
        $\leftarrow Q(s_i, a_i)$
          $+\alpha \left[r + \gamma * max_{a_{i+1}} Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i)\right]$
      $s_i \leftarrow s_{i+1}$
    until the number of steps is X

The setting of a fixed number of turns X is motivated by the game mechanics. The size of X is directly related to the size of the map and the number of opponents. The size is usually defined in a way that after X turns, most of the map has been divided between the different factions and the main focus in the game shifts from expansion to consolidation of acquired territory and conquest of enemy terrain and thus away from the city placement task.

The discount factor γ in our experiments, which are presented in section VI, is set to 0.1. The reason for this rather low value lies in the large number of possible actions in each state. All $Q(s, a)$ values are initialised to zero and the reward signal is always positive. Therefore, actions which are pursued in earlier episodes are likely to be called

disproportionally more often if a policy different to complete randomness is pursued. The learning rate α is set to 0.2 which proved to be high enough to lead to relatively fast convergence while being low enough to protect the Q-values from anomalies in the reward signal. Both γ and α values have also been tested in trial runs and proven to work best for our purposes with the values given above.

Since Q-Learning is by definition an *off-policy* algorithm, the learned action-value function Q converges with probability one to the optimal action-value function Q^* even following a completely random policy. However, this requires visiting every single state infinite times, which is computationally not feasible. The algorithm uses an ϵ-greedy policy. Its general behavior can be modified by altering ϵ, depending on whether the main aim is learning or performing optimally, i.e. if the focus should be on exploration or exploitation.

It is noteworthy that all Q-values show a dependency on the starting position on the map of the respective player. Furthermore, since the usability of a plot as founding site for a city completely depends on the geography of the surroundings, the Q-values obviously are correlated to the map which is used in the game. As soon as the geography changes, the Q-values have to be computed from scratch. This dependency and how we intend to deal with it is further elaborated in section VII.

VI. Empirical Evaluation

We ran several experiments to compare our adaptive city plot selection algorithm to the existing deterministic sequential approach. In order to be able to compare the growing effectiveness of greater coverage over the state space, several experiments with differing numbers of turns as well as different settings for the ϵ-greedy policy were run. Except for the method of choosing the best spot for its cities, the original game AI was left unchanged.

The standard setup is one computer player that uses experience gained through Q-Learning against two other computer players which use the standard method of determining founding plots. The map used is the same in every game, as well as the starting positions. All players have exactly the same "character traits" (a game mechanic which leads to advantages in certain parts of the game like research or combat) so they are starting under exactly the same premises. The size of the chosen map is the second smallest in the game and the map consists of about 300 plots which can be settled. According to Equation (1) this leads to the number of possible states

$$|S| = \sum_{i=0}^{c} \frac{300!}{(300-i)!}.$$

c is in this case equal to the number of cities that are expected to be built in the given number of turns. This basically means that not the complete state-tree will be traversed but only the tree up to a depth equal to the maximum number of cities. The number of game turns differed between the experiments and was decided according to the goal of the respective test. Due to the low number of actions which are taken in one episode, a large number of episodes had to be played for every setup to get meaningful results.

One of our main aims was to find out how the reinforcement learner performed compared to the standard game AI. To achieve an adequate coverage of the state space, which is necessary to reach comparable results to the static but quite sophisticated standard game AI, the number of game turns was set to 50 (12.5 % of the maximum length of a game). This seems rather short but since the map on which the game is played is small, the game phase during which the players found new cities is usually very short as well. After 50 turns on average about half of the map has been occupied by the three players.

The limitation of the single episodes to a length of 50 turns leads to an expected maximum number of cities of 2 for the reinforcement player. This means that the possible states are limited to about

$$\sum_{i=0}^{2} \frac{300!}{(300-i)!} \approx 90000.$$

Since despite the low number of cities the high branching factor leads to this large state space, convergence is not guaranteed, even though the Q-Learning algorithm is usually able to cope through the ϵ-greedy policy which pursues the maximum rewards with probability $1-\epsilon$. ϵ was initialised at 0.9, i.e. in 90% of all cases our algorithm would pick a random action while selecting the action with the highest $Q(s,a)$ value in the remaining 10%. This ratio was subsequently slowly reverted, that means after 3000 played episodes only 10% of all actions would be random while 90% were picked according to the highest $Q(s,a)$ value. This is necessary to draw a meaningful comparison between the reinforcement learner and the standard AI when both try to play optimal or close to optimal.
3050 episodes of length 50 turns were played by the computer AI. After each episode the score of the players was recorded. For the final evaluation, the score of the RL player was averaged across the last 50 episodes to even out the anomalies which occur because of the explorative policy. As a reference value, the same experiment was performed with a standard AI player instead of the reinforcement player, i.e. three standard computer AI players compete against each other on the same map with the same premises as in the previous test. Figure 4 shows the results of both experiments.

The first thing that attracts attention is the performance of the standard computer AI under these circumstances, which results in the same score for every single game. This is due to the fact that there is no randomness in the decisions of the computer AI when it comes to city placement. There are very few non-deterministic decisions in *Civilization IV*, most of them in the combat algorithms, but those do not have any effect until later in the game when players fight against each other. Therefore, for the first 50 turns, the three standard AIs always performed the exact same actions.

The diagram also shows that the reinforcement learner,

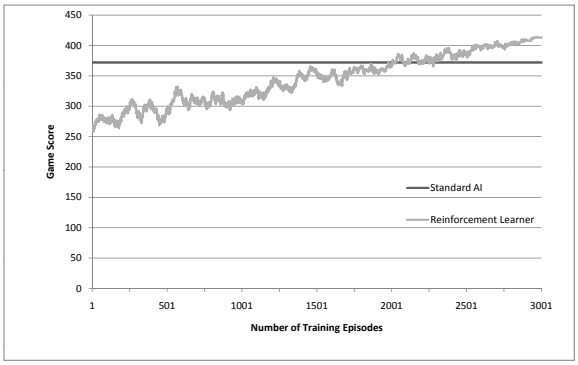


Fig. 4. Average Score Comparison between Reinforcement Learner and Standard AI for Games with Length 50 Turns

while initially inferior in average score, ends up beating the average score of the standard AI. On the downside it is noteworthy that it took the reinforcement learner more than 2000 episodes of the game to reach that point. Since the number of the episodes played is still a lot smaller than the state space for the placement of two cities, there also remains room for improvement by finding even better policies.

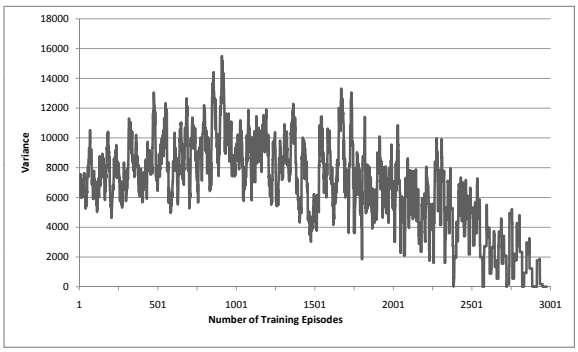


Fig. 5. Variance in the Average Score: From Explorative to Exploitative Policy by decreasing ϵ

Figure 5 shows the variance of the average of the scores for the reinforcement learner. Its decline illustrates the change of policy with increasing number of episodes.

Since the reinforcement learner performed very well for short games and a large number of episodes, another experiment was conducted to evaluate the convergence with less episodes and more turns. The number of turns was doubled to 100 turns per episode while the number of episodes was reduced to 1500. The evaluation happened in the same way as in the previous experiment through recording the game score and averaging the score for 50 episodes. As an addition the score was not only measured at the end of an episode, i.e. after 100 turns but also after 50 turns like in the previous experiment. Furthermore we performed another set of tests in the same environment with an AI that follows a completely random policy. This means that $\epsilon = 1$, which results in the player always picking actions at random.

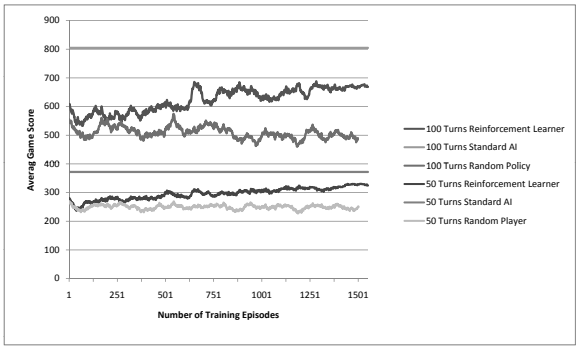


Fig. 6. Average Score Comparison for Games of Length 100 Turns

Figure 6 shows the results of these tests. As in the previous test run with only 50 turns, the standard AI achieves the same score for every single game at 50 turns. According to the diagram it also seems as if this is the case at 100 turns. However the standard AI achieves not the same score in every game, but alternates between two different scores which ultimately results in the same average over 50 episodes. This alternation suggests that a probabilistic decision is made during the second 50 turns. At the beginning when the RL agent has no experience yet, both RL and random agent have about the same average score. But while the score for the player following a completely random policy shows as expected no sign of long term growth or decline, the average score of the reinforcement learner improves with the growing number of episodes played.

The score for the reinforcement learner at 50 turns shows the same upward tendency as the score for the RL agent in the previous experiment (Figure 5). The average score is still lower than that of the standard AI because of the smaller number of episodes played. If growth of the average score continues, this would very likely change within the next 1000 episodes. The average score for the reinforcement learner after 100 turns shows the same tendency as the score at 50 turns even though the gap between the score. However the gap between the average score for the standard AI and the reinforcement learner is much bigger at 100 turns than at 50 turns. This can be explained through the significant difference in size of the state spaces for 50 turns and 100 turns. While during 50 turns players will get a maximum of two cities, 100 turns will allow building up to five cities which multiplies the number of possible states by nearly 300^3. Therefore it is remarkable that there is already a visible increase in the average score at 100 turns. This also means that there is potentially a huge margin to be gained over the standard AI through optimising the Q-values.

VII. Future Work

As previously stated, this paper presents the first results of a work in progress, the application of reinforcement learning to tasks in Civilization IV. Several extensions and additions are planned for the near future and can be divided into

three categories. These categories are the improvement of the Q-Learning algorithm which has been used throughout this paper, the evaluation of other reinforcement learning algorithms and the combination of other machine learning techniques with RL. Furthermore the usage of *Civilization IV* as a test bed for computer game AI research can be extended.

The results of the empirical evaluation show that while the applied reinforcement learning method has potential to outperform the static standard AI, in its current state learning becomes computationally unfeasible when crossing a certain threshold of game complexity, either in matters of game turns or map plots. Therefore the optimisation of the Q-Learning algorithm is crucial to extend the tasks for which it can be used, i.e. longer games or larger maps. One way to to do this is by speeding up the learning process and as a result accelerating the convergence towards the optimal action-value function. This can for instance be achieved by using eligibility traces and thus having a multi-step Q-Learning algorithm instead of the current one-step Q-Learning. Another way to improve the speed of convergence is the initialisation. At the moment all $Q(s, a)$ values are initialised to 0 at the beginning of the algorithm. This means that every state has to be visited in order to determine its usefulness for the exploitation part of the policy, often only to conclude that its usefulness is very low. If the states were instead initialised to the precomputed "founding values" that are used by the standard game AI, these values could serve as indicators about the usefulness of a plot for the reinforcement learner. This would not speed up the guaranteed convergence to an optimal policy π^* but generate better performance earlier on, resulting in more challenging gameplay.

Besides extending the existing method, other reinforcement learning algorithms and techniques such as the on-policy temporal-difference algorithm SARSA or Monte Carlo methods will be evaluated as to how well they are suited for the city placement selection task [2].

Another field for future research on RL using Civilization IV as a test bed is the combination of other machine learning methods with RL. One particularly promising method is Case-Based Reasoning (CBR). The application of CBR to the plot selection task would allow to resolve the previously mentioned problem with learned experience on one map being useless on another map because of the different topology.

Furthermore the application of Motivated Reinforcement Learning (MRL) could improve the game play experience. One of the game mechanics in Civilization IV are "character traits" of the different computer AIs. Certain players have by definition advantages in certain areas of the game like expansion, finance or combat. These advantages are hard coded numbers and are meant to express certain character traits like aggressiveness or expansionism. If those agents would instead use a reinforcement learner which gets his rewards through a motivational function as described in [12], this could lead to very interesting behaviour for AI players.

Also the task for which these machine learning techniques are used can be extended. While at the moment the task only consists of determining where a city should be, in the future this could also include the choice if the city is needed at all, i.e. the optimal number of cities and when to build them.

VIII. Conclusions

This paper presents the design and implementation of a reinforcement learner which is used to perform a city site selection task in the turn-based strategy game *Civilization IV*. The Q-Learning algorithm which was used, manages to learn city placement strategies for specific maps. After sufficient training the RL agent outperforms the standard game AI in short matches. The reinforcement learner also shows promising results for longer and more complex games. The findings from these experiments on the possible shortcomings of the algorithm lay the groundwork for future work. Furthermore the usage of the commercial video game *Civilization IV* as a test bed for AI research demonstrates great potential because of the diversity of the tasks involved in Civilization and the relative ease with which these deterministic tasks can be taken over by machine learning agents. The integration of our RL agent into *Civilization IV* extends the otherwise deterministic early game by a non-deterministic, adaptable component which enhances the game-playing experience.

References

[1] J. Laird and M. van Lent, "Human-level AI's Killer Application: Interactive Computer Games," *AI Magazine*, vol. Summer 2001, pp. 1171–1178, 2001.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[3] C. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, University of Cambridge, England, 1989.

[4] P. Ulam, A. Goel, and J. Jones, "Reflection in Action: Model-Based Self-Adaptation in Game Playing Agents," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence American Association for Artificial Intelligence (AAAI)*, 2004.

[5] P. A. Houk, "A Strategic Game Playing Agent for FreeCiv," Northwestern University, Evanston, IL, Tech. Rep. NWU-CS-04-29, 2004.

[6] R. Sánchez-Pelegrín, M. Gómez-Martín, and B. Díaz-Agud, "A CBR Module for a Strategy Videogame," in *1st Workshop on Computer Gaming and Simulation Environments, at 6th Int. Conference on Case-Based Reasoning (ICCBR)*, D. Aha and D. Wilson, Eds., 2005.

[7] A. Sánchez-Ruiz, S. Lee-Urban, H. Muñoz-Avila, B. Díaz-Agudoy, and P. González-Caleroy, "Game AI for a Turn-Based Strategy Game with Plan Adaptation and Ontology-based Retrieval," in *Proceedings of the ICAPS 2007 Workshop on Planning in Games*, 2007.

[8] D. W. Aha and M. Molineaux, "Integrating Learning in Interactive Gaming Simulators," Intelligent Decision Aids Group; Navy Center for Applied Research in Artificial Intelligence, Tech. Rep., 2004.

[9] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Automatic computer game balancing: a reinforcement learning approach," in *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, 2005.

[10] M. Smith, S. Lee-Urban, and H. Muñoz-Avila, "RETALIATE: Learning Winning Policies in First-Person Shooter Games," in *Proceedings of the Seventeenth Innovative Applications of Artifcial Intelligence Conference (IAAI-07)*, 2007.

[11] B. Auslander, S. Lee-Urban, C. Hogg, and H. Muñoz-Avila, "Recognizing the Enemy: Combining Reinforcement Learning with Strategy Selection using Case-Based Reasoning," in *Advances in Case-Based Reasoning: 9th European Conference, ECCBR 2008, Trier, Germany, September, 2008, Proceedings*, K.-D. Althoff, R. Bergmann, M. Minor, and A. Hanft, Eds. Springer, 2008.

[12] K. E. Merrick and M. L. Maher, "Motivated Reinforcement Learning for Adaptive Characters in Open-Ended Simulation Games," in *ACE '07: Proceedings of the International Conference on Advances in Computer Entertainment Technology*. New York, NY, USA: ACM, 2007, pp. 127–134.

Appendix B

Results for 1000 Episodes of Length 60 Turns using Sarsa with Declining ϵ-Greedy Policy

0	595	476	392	469	476	413	439	451	469	447	**125**	475	420	478	406	292	445	412	375	392	403	**250**	481	392	478	407	553	469	481	459	435	478	**375**	503	314	470	342	369	490	460	459	458	485
1	493	419	420	405	426	426	290	405	417	462	**126**	432	431	456	406	546	401	431	472	449	390	**251**	439	371	499	406	510	490	438	459	391	478	**376**	353	413	478	421	552	490	448	308	458	312
2	441	420	479	435	400	392	397	215	406	361	**127**	363	378	438	301	402	445	349	488	435	262	**252**	531	420	478	366	384	469	460	459	392	478	**377**	439	335	387	350	510	490	481	459	458	478
3	458	371	438	406	481	478	425	370	445	397	**128**	432	216	421	392	539	459	481	459	362	368	**253**	531	371	387	406	546	445	495	459	456	384	**378**	531	371	484	421	510	490	481	478	402	405
4	463	442	421	529	400	363	390	386	435	397	**129**	482	413	429	400	460	490	293	278	392	390	**254**	356	420	478	412	469	497	481	411	435	485	**379**	531	335	478	227	510	490	481	459	458	485
5	465	420	492	406	354	398	335	392	476	397	**130**	439	484	456	406	539	490	453	459	399	478	**255**	439	371	478	406	539	445	481	459	326	427	**380**	441	413	478	457	510	490	481	430	484	347
6	465	441	456	383	407	363	397	343	435	493	**131**	410	371	471	267	510	490	386	459	406	493	**256**	385	420	478	406	405	357	481	442	399	478	**381**	531	464	434	313	510	490	481	459	458	478
7	465	407	471	392	491	420	393	370	435	397	**132**	481	424	456	370	510	490	481	459	494	485	**257**	439	371	478	406	552	445	481	459	456	500	**382**	256	371	478	414	377	298	481	461	455	468
8	278	261	456	490	400	363	397	378	435	458	**133**	378	413	411	443	510	490	412	459	435	478	**258**	434	420	478	406	510	232	481	385	406	485	**383**	439	471	251	387	510	490	481	459	406	478
9	465	413	456	406	262	439	465	370	494	361	**134**	439	379	421	406	510	490	453	459	346	478	**259**	439	403	471	421	510	490	481	459	428	330	**384**	304	420	471	421	478	376	481	375	308	478
10	465	407	450	456	378	403	397	497	435	397	**135**	302	413	336	437	510	490	311	459	406	471	**260**	397	371	471	392	510	269	451	414	406	485	**385**	531	483	442	414	510	445	385	472	392	478
11	465	371	329	406	405	260	495	392	399	397	**136**	439	349	450	406	382	490	495	459	414	485	**261**	481	469	442	346	510	503	481	459	464	471	**386**	531	371	478	450	283	500	481	424	401	478
12	465	298	456	437	442	403	397	360	435	480	**137**	382	420	197	450	539	490	375	459	435	431	**262**	351	420	478	406	510	407	481	339	458	478	**387**	531	420	443	450	552	490	469	459	435	478
13	465	413	384	406	403	335	404	370	399	397	**138**	531	493	450	392	386	490	481	355	399	478	**263**	439	456	449	450	382	445	481	459	468	413	**388**	531	371	507	421	466	490	481	461	442	478
14	465	457	421	445	400	410	397	392	435	176	**139**	531	371	471	439	552	196	481	472	435	439	**264**	272	420	478	392	510	465	282	225	435	478	**389**	531	420	327	401	510	490	453	459	399	469
15	571	371	373	406	462	346	397	392	328	368	**140**	531	401	456	412	320	503	481	428	278	478	**265**	432	394	327	215	320	497	488	459	399	322	**390**	531	371	513	457	437	449	481	327	320	478
16	458	510	421	297	400	410	397	392	435	260	**141**	531	420	337	227	517	355	481	459	435	478	**266**	433	371	513	457	517	450	412	414	406	485	**391**	364	420	448	421	539	445	434	459	435	500
17	372	413	442	406	404	503	397	190	205	390	**142**	462	371	405	406	362	445	481	404	527	478	**267**	439	379	448	425	193	490	481	459	435	478	**392**	432	371	471	414	369	353	467	358	347	485
18	471	446	479	385	407	490	397	399	392	438	**143**	439	420	421	412	517	441	481	466	435	478	**268**	327	413	471	421	517	490	328	308	399	478	**393**	426	420	478	414	510	490	368	459	441	330
19	449	378	462	399	372	490	397	256	435	397	**144**	462	371	405	406	383	490	481	406	314	478	**269**	481	450	478	414	233	445	488	459	435	478	**394**	439	431	507	414	402	272	460	459	234	485
20	465	379	450	421	413	490	397	370	399	354	**145**	503	420	421	406	552	442	481	459	441	478	**270**	429	407	478	414	539	497	481	459	399	478	**395**	375	371	447	414	517	503	495	459	435	225
21	365	378	370	406	520	490	397	423	435	390	**146**	354	371	405	357	261	490	481	260	435	384	**271**	531	440	478	335	328	445	481	459	435	478	**396**	439	252	478	414	539	416	481	459	378	478
22	465	444	456	397	400	490	397	392	399	253	**147**	439	408	421	406	539	437	481	459	399	498	**272**	531	413	478	443	510	503	445	459	399	431	**397**	233	413	478	414	510	490	312	459	435	448
23	389	413	421	406	469	490	397	272	435	368	**148**	531	407	405	492	254	445	410	298	435	478	**273**	531	348	478	389	551	490	481	459	435	485	**398**	439	393	478	414	510	372	488	459	304	478
24	458	355	405	328	400	490	495	399	493	393	**149**	531	470	421	406	517	250	488	459	399	478	**274**	531	413	478	421	510	490	350	459	346	433	**399**	531	470	478	414	510	490	387	459	441	322
25	279	413	421	406	322	490	397	422	399	397	**150**	432	371	405	517	358	490	480	499	435	478	**275**	345	421	478	414	377	490	481	459	435	485	**400**	577	400	478	368	510	319	488	459	435	485
26	458	261	405	406	400	347	490	392	440	396	**151**	439	380	421	392	546	326	481	459	399	478	**276**	481	378	478	414	517	490	415	459	343	468	**401**	243	470	478	414	510	445	442	459	399	444
27	367	442	421	406	400	490	397	254	435	390	**152**	407	413	405	354	377	445	481	329	435	412	**277**	400	283	478	415	539	490	460	459	435	485	**402**	535	285	478	291	510	376	481	459	435	478
28	471	366	494	406	400	405	481	399	450	397	**153**	439	475	421	406	517	327	481	459	372	478	**278**	439	413	405	457	510	490	478	459	448	440	**403**	338	470	478	421	270	445	481	406	399	323
29	345	378	456	406	400	503	397	488	399	397	**154**	449	420	456	360	395	503	481	262	441	431	**279**	453	371	507	421	423	441	481	450	441	485	**404**	542	422	478	313	539	503	481	459	435	485
30	465	389	441	406	400	544	449	392	486	397	**155**	439	376	456	406	510	393	481	459	479	478	**280**	439	420	353	414	517	445	384	466	372	431	**405**	577	470	478	421	412	490	481	260	399	478
31	460	413	456	450	400	445	397	384	406	450	**156**	406	407	434	176	423	490	481	449	399	384	**281**	400	371	507	414	539	426	488	260	392	485	**406**	577	315	344	401	539	490	481	459	435	478
32	458	371	451	392	400	440	485	399	349	361	**157**	439	421	405	406	517	468	481	459	205	485	**282**	439	420	471	414	510	490	481	459	440	186	**407**	577	470	507	414	465	490	481	298	432	478
33	288	420	456	408	400	445	397	478	392	300	**158**	531	442	429	329	539	445	363	430	441	427	**283**	476	371	507	414	510	486	481	401	406	485	**408**	577	433	464	376	539	490	481	459	435	478
34	465	371	458	406	386	403	479	392	449	361	**159**	531	446	421	406	510	310	488	459	383	478	**284**	531	420	378	414	510	445	481	459	382	449	**409**	577	512	478	457	385	490	481	392	472	478
35	450	420	456	403	400	490	397	458	435	375	**160**	531	371	422	406	510	503	253	488	392	427	**285**	432	371	478	414	510	439	481	283	392	485	**410**	344	462	574	394	510	490	460	459	399	504
36	458	371	524	406	240	407	311	370	428	361	**161**	400	421	450	406	444	442	481	459	186	498	**286**	531	437	405	414	476	490	481	466	308	372	**411**	599	512	478	421	491	490	481	329	399	478
37	465	420	456	176	442	490	397	392	406	473	**162**	439	413	470	354	552	490	481	459	392	402	**287**	272	371	478	414	539	490	481	459	458	485	**412**	356	437	499	415	510	490	434	459	406	386
38	465	371	421	406	363	490	319	392	497	397	**163**	495	393	456	406	254	432	481	459	410	485	**288**	503	262	478	522	470	445	495	459	472	441	**413**	535	512	478	457	471	469	467	462	523	478
39	465	235	405	377	400	445	390	392	399	368	**164**	439	371	451	204	517	445	403	459	399	478	**289**	559	413	478	421	539	422	481	413	458	485	**414**	250	512	440	420	510	490	437	459	399	439
40	465	413	451	406	481	250	207	392	439	361	**165**	442	399	450	406	411	249	488	459	378	478	**290**	531	437	335	397	518	445	481	472	458	167	**415**	599	512	478	457	450	379	488	430	354	478
41	465	284	450	250	481	490	390	392	399	260	**166**	531	442	403	417	517	490	293	459	435	478	**291**	439	371	478	421	510	393	271	449	458	478	**416**	577	512	183	457	517	490	550	466	392	396
42	465	407	433	406	354	490	335	368	289	390	**167**	429	411	456	406	228	490	453	459	361	478	**292**	531	450	233	271	469	490	488	459	458	485	**417**	584	512	507	457	458	393	481	430	456	485
43	465	383	450	346	539	490	397	406	435	410	**168**	439	371	411	412	517	490	393	459	406	478	**293**	531	442	507	457	510	451	363	458	458	478	**418**	432	512	299	421	517	490	242	459	421	265
44	465	442	446	370	460	470	365	487	328	390	**169**	471	420	405	406	491	490	481	459	435	340	**294**	531	407	276	354	466	445	488	459	458	478	**419**	577	512	507	414	539	410	488	450	404	478
45	465	445	456	448	539	445	397	370	399	290	**170**	432	371	421	406	510	490	481	459	399	478	**295**	531	371	471	457	510	495	357	227	458	478	**420**	437	512	251	414	510	445	495	459	450	430
46	271	371	254	406	369	445	397	488	435	390	**171**	438	420	405	406	424	490	481	319	435	433	**296**	432	374	437	512	386	445	481	459	458	478	**421**	584	384	471	414	510	437	481	375	437	478
47	465	446	421	233	474	490	397	392	399	306	**172**	531	371	529	406	539	490	481	459	399	478	**297**	531	413	478	421	539	392	427	385	435	478	**422**	400	505	413	414	510	490	375	472	456	478
48	497	413	336	370	440	450	397	283	435	397	**173**	233	420	479	464	443	490	481	448	435	403	**298**	467	393	501	414	404	445	481	459	458	478	**423**	535	564	507	414	510	347	488	395	421	478
49	471	291	450	447	517	503	397	370	399	427	**174**	481	498	384	406	517	435	481	459	399	478	**299**	531	371	507	414	517	503	536	298	266	432	**424**	577	512	401	414	510	490	394	459	421	478
50	411	413	456	406	444	503	397	377	435	397	**175**	439	420	421	456	481	490	481	413	435	440	**300**	298	378	235	397	443	490	488	459	458	485	**425**	577	475	471	414	510	282	481	459	421	478
51	458	357	421	412	510	490	397	428	399	377	**176**	531	413	491	406	517	516	481	459	352	478	**301**	439	413	471	457	539	420	345	225	458	354	**426**	577	470	482	414	510	503	424	459	421	478
52	465	407	405	406	488	297	397	426	435	390	**177**	531	371	456	269	472	445	367	428	406	428	**302**	474	305	478	420	386	503	481	459	458	485	**427**	577	270	478	271	517	356	460	459	421	396
53	465	527	421	406	539	490	397	399	505	320	**178**	327	427	421	406	539	405	481	459	486	485	**303**	439	413	478	457	510	522	478	459	329	447	**428**	577	470	478	457	510	503	495	459	421	485
54	326	413	405	406	411	490	397	260	435	390	**179**	439	413	405	419	494	503	345	461	435	504	**304**	388	371	478	425	510	445	488	459	458	485	**429**	419	457	478	354	517	411	481	459	421	269
55	465	392	421	406	517	490	497	399	428	397	**180**	519	432	421	406	539	356	481	459	355	478	**305**	481	420	478	421	510	446	481	459	390	378	**430**	542	505	478	457	443	490	481	459	421	485
56	475	378	405	433	539	490	397	376	435	397	**181**	439	420	405	445	510	503	386	403	435	473	**306**	444	371	478	407	510	503	481	459	458	485	**431**	332	495	478	487	517	381	481	459	421	319
57	458	347	421	406	510	490	390	399	372	397	**182**	367	469	421	406	510	494	488	459	504	485	**307**	439	420	478	457	510	503	481	459	408	378	**432**	535	512	478	421	292	490	481	365	421	478
58	491	413	405	455	510	490	481	449	435	478	**183**	481	371	405	304	510	490	358	251	435	431	**308**	411	371	478	421	510	490	481	459	458	498	**433**	278	451	478	420	510	459	392	459	421	365
59	465	371	456	412	216	490	487	428	468	397	**184**	382	298	421	406	510	410	481	459	495	485	**309**	439	420	478	414	299	490	481	459	463	338	**434**	599	470	478	457	479	503	481	291	463	478
60	360	420	450	469	539	490	453	348	399	513	**185**	439	413	363	422	510	445	413	341	435	431	**310**	531	371	478	414	552	490	481	459	458	478	**435**	400	441	478	458	510	445	298	459	421	375
61	465	371	355	406	261	490	488	428	383	390	**186**	350	357	450	406	510	394	495	459	363	478	**311**	531	442	356	414	363	490	481	459	458	471	**436**	584	505	478	457	350	503	481	440	391	485
62	351	420	421	401	539	490	481	428	406	421	**187**	439	407	263	498	510	490	386	346	399	485	**312**	531	371	478	414	552	490	434	423	458	478	**437**	382	402	424	420	546	490	433	459	450	386
63	465	371	234	412	539	490	375	428	478	390	**188**	467	401	450	406	443	419	431	459	371	478	**313**	531	327	418	414	320	490	488	459	466	487	**438**	599	470	478	457	520	490	488	458	430	478
64	338	420	450	478	510	445	481	428	392	450	**189**	531	413	418	412	539	445	460	329	435	478	**314**	531	413	507	414	517	490	471	236	458	485	**439**	400	440	328	364	517	490	457	459	456	478
65	465	371	494	392	355	465	419	428	371	397	**190**	458	436	450	406	320	440	481	459	399	260	**315**	372	448	328	436	412	490	481	459	410	207	**440**	584	470	507	457	366	490	495	505	356	478
66	380	392	479	319	517	445	431	428	435	478	**191**	439	371	376	406	539	490	338	459	435	478	**316**	439	413	507	421	546	490	471	290	458	485	**441**	417	512	412	421	517	490	475	459	450	478
67	458	420	430	412	227	362	278	443	360	361	**192**	428	401	405	406	444	459	488	459	535	424	**317**	420	449	412	355	261	490	488	459	307	478	**442**	577	512	471	414	539	490	488	503	444	478
68	370	205	405	383	552	503	431	435	435	441	**193**	481	420	431	406	539	445	449	459	435	478	**318**	531	407	471	421	539	404	360	398	458	478	**443**	432	512	465	319	510	490	488	459	456	478
69	465	442	460	406	465	516	478	439	398	361	**194**	497	443	421	399	405	357	488	459	370	233	**319**	400	391	478	479	299	445	488	459	458	478	**444**	577	512	478	457	510	490	481	418	429	365
70	465	350	456	433	539	445	495	399	435	478	**195**	432	378	449	406	552	490	487	459	435	485	**320**	481	371	507	457	517	375	271	452	458	431	**445**	577	512	420	421	510	490	474	459	456	485
71	465	371	361	406	485	458	356	480	242	397	**196**	439	525	450	372	517	358	453	459	417	338	**321**	473	279	535	468	338	445	488	459	458	478	**446**	577	512	478	414	510	479	481	413	495	319
72	465	517	421	349	539	445	488	399	435	571	**197**	531	407	386	399	510	445	488	459	435	485	**322**	439	413	507	450	546	394	372	490	458	365	**447**	577	512	500	414	510	490	434	459	456	478
73	465	371	364	412	453	385	318	234	448	397	**198**	531	420	450	466	462	497	481	459	435	372	**323**	442	408	285	469	470	490	488	459	458	485	**448**	577	512	478	414	510	419	488	428	493	360
74	465	383	421	354	510	503	481	399	435	397	**199**	439	371	454	412	517	445	481	459	399	478	**324**	531	371	471	421	517	475	424	430	176	319	**449**	577	512	450	414	471	445	471	459	450	478
75	465	371	451	399	480	437	442	346	399	397	**200**	531	420	421	359	510	529	481	432	435	401	**325**	391	440	353	510	327	445	488	459	458	485	**450**	307	380	478	414	510	503	481	420	485	442
76	465	442	421	412	510	490	438	399	435	397	**201**	401	371	458	406	517	490	481	459	399	478	**326**	439	442	513	421	517	440	441	385	516	310	**451**	542	470	335	414	244	490	481	459	456	485
77	467	371	458	406	443	432	179	423	399	397	**202**	439	420	450	430	245	490	481	459	435	478	**327**	468	205	376	452	539	490	460	459	458	485	**452**	400	408	478	414	517	495	481	459	421	381
78	458	471	450	406	539	445	488	428	466	397	**203**	478	371	216	406	517	490	481	459	464	478	**328**	531	413	478	457	510	459	495	403	364	345	**453**	599	512	183	414	441	503	481	459	421	478
79	398	420	421	386	441	289	282	402	435	428	**204**	531	420	405	468	340	490	312	332	435	478	**329**	463	270	234	450	510	445	481	459	458	485	**454**	254	369	507	414	517	269	481	459	421	403
80	458	469	405	406	539	503	488	428	464	390	**205**	356	177	421	412	552	490	481	459	456	478	**330**	439	413	478	421	510	497	481	420	439	403	**455**	599	470	499	414	309	490	481	459	421	485
81	444	420	303	421	510	449	481	392	435	449	**206**	439	413	405	472	510	490	443	243	435	478	**331**	531	473	276	413	510	445	481	459	458	485	**456**	435	382	478	468	517	253	481	459	421	347
82	465	515	421	406	510	490	481	428	485	390	**207**	367	481	421	412	510	468	488	459	436	478	**332**	531	371	471	457	510	503	481	392	261	219	**457**	584	512	431	421	421	490	481	459	421	478
83	319	407	357	429	510	298	481	403	435	375	**208**	481	413	405	426	510	490	346	369	435	478	**333**	531	420	437	396	510	490	481	459	458	478	**458**	535	499	478	469	517	376	481	459	421	485
84	465	378	456	406	510	445	481	399	401	361	**209**	419	360	421	392	510	427	488	459	448	413	**334**	484	371	478	457	467	394	481	332	420	262	**459**	577	512	478	421	327	445	481	336	421	478
85	525	413	421	466	510	285	481	428	435	468	**210**	439	413	405	406	510	503	255	439	399	478	**335**	432	420	478	421	539	445	481	459	458	485	**460**	260	376	478	459	517	535	357	459	421	478
86	465	411	405	406	369	503	401	428	456	361	**211**	350	270	421	412	510	437	488	459	474	427	**336**	471	371	478	414	510	363	481	459	413	421	**461**	535	512	478	421	538	490	481	479	421	478
87	445	413	421	464	546	459	481	428	406	368	**212**	439	413	405	406	510	490	440	474	435	478	**337**	531	420	478	414	517	490	307	459	493	485	**462**	486	205	478	361	539	490	481	459	421	478
88	439	371	405	406	443	503	412	215	446	361	**213**	531	428	421	406	207	461	481	459	458	431	**338**	528	371	478	414	244	490	481	459	464	465	**463**	584	512	478	457	380	490	481	465	285	478
89	428	420	421	456	517	312	431	399	392	438	**214**	531	420	405	406	517	497	438	440	399	478	**339**	439	420	478	414	517	490	320	459	493	478	**464**	233	263	478	401	510	544	347	466	456	478
90	439	371	405	406	393	490	448	470	458	397	**215**	531	475	421	406	467	282	431	459	464	400	**340**	412	382	478	414	539	490	488	459	410	419	**465**	535	505	478	457	510	445	488	449	421	478
91	330	420	421	308	517	319	481	428	406	444	**216**	531	371	465	406	539	503	463	458	435	485	**341**	439	378	478	414	510	490	363	459	458	485	**466**	577	414	478	450	510	503	206	459	421	478
92	481	371	405	392	261	445	469	349	474	390	**217**	531	430	456	276	520	468	495	459	505	197	**342**	447	417	478	414	510	490	460	459	420	478	**467**	577	505	478	457	510	490	488	486	483	260
93	479	233	421	479	517	376	481	399	435	290	**218**	497	420	500	406	517	490	444	505	435	478	**343**	531	378	478	414	517	490	419	459	458	478	**468**	577	495	393	396	510	490	497	466	421	485
94	531	413	296	406	466	503	412	407	410	390	**219**	531	420	478	232	385	384	488	459	399	382	**344**	350	392	327	253	539	490	460	459	458	478	**469**	577	512	513	457	510	490	467	526	474	478
95	344	321	421	387	510	503	453	428	435	272	**220**	497	371	433	406	517	490	431	459	435	478	**345**	439	407	471	457	510	490	485	434	464	478	**470**	577	457	376	435	553	490	488	459	456	478
96	481	407	314	406	466	490	378	283	435	397	**221**	531	433	478	298	329	272	481	459	399	176	**346**	484	320	471	538	431	490	481	459	458	478	**471**	528	470	478	450	510	490	488	336	418	290
97	439	427	450	464	539	490	488	399	399	397	**222**	391	420	373	370	510	503	251	459	435	478	**347**	432	413	471	421	510	490	419	375	458	372	**472**	584	499	327	414	437	490	278	459	450	485
98	531	413	529	406	369	490	311	389	435	397	**223**	503	252	507	528	489	459	431	459	399	328	**348**	354	400	442	412	462	490	481	472	458	485	**473**	497	512	471	414	517	490	488	479	456	355
99	531	464	479	406	510	490	481	428	399	397	**224**	256	413	256	406	539	503	393	459	435	478	**349**	503	420	478	421	539	432	392	459	458	365	**474**	542	512	187	414	539	490	481	459	421	485
100	531	371	493	406	451	490	375	169	435	397	**225**	439	371	513	441	414	312	481	459	399	478	**350**	439	257	478	454	438	445	481	459	458	485	**475**	577	512	471	414	510	282	481	459	465	553
101	531	609	479	406	517	490	481	428	399	397	**226**	278	420	500	412	546	490	481	459	420	478	**351**	531	413	478	421	510	426	372	236	458	311	**476**	577	512	257	414	393	503	481	459	450	485
102	549	371	609	406	559	420	176	336	435	260	**227**	503	371	507	466	349	196	481	391	392	478	**352**	531	334	327	368	510	490	488	459	458	498	**477**	350	512	507	414	552	449	481	459	251	478
103	531	470	479	406	510	490	481	399	253	361	**228**	360	420	257	406	510	503	481	472	378	478	**353**	531	371	513	443	510	465	481	290	389	480	**478**	535	512	357	414	261	490	481	459	450	478
104	428	371	470	272	421	307	307	347	435	321	**229**	481	371	507	243	510	354	481	318	441	478	**354**	531	380	448	320	510	445	481	459	458	478	**479**	460	512	478	414	517	336	481	459	456	478
105	439	440	456	406	539	490	481	399	343	361	**230**	419	420	403	406	517	503	481	459	468	478	**355**	571	413	471	450	233	384	481	379	550	478	**480**	577	512	465	414	366	490	481	459	421	478
106	243	442	396	406	510	490	442	428	435	462	**231**	439	371	478	412	235	243	481	197	399	478	**356**	531	360	478	278	539	503	481	459	458	478	**481**	345	419	478	449	517	357	481	459	404	478
107	503	428	405	406	510	445	481	428	500	390	**232**	531	408	363	406	552	490	481	466	354	452	**357**	450	413	507	457	309	272	481	461	471	478	**482**	599	512	535	457	320	490	481	375	443	487
108	457	420	363	365	510	260	448	428	435	461	**233**	531	371	478	406	510	452	513	272	441	478	**358**	531	477	391	421	517	503	481	459	458	478	**483**	542	360	507	458	539	459	387	472	456	485
109	531	392	456	406	510	490	481	428	434	397	**234**	531	384	447	406	510	445	481	459	433	381	**359**	457	371	478	421	421	401	481	461	436	412	**484**	577	512	478	457	350	503	481	488	421	424
110	356	413	398	426	510	253	481	428	435	397	**235**	531	378	478	406	510	470	557	413	399	498	**360**	531	420	478	351	517	445	481	459	458	478	**485**	577	440	478	443	546	442	428	459	421	485
111	439	257	421	392	392	490	481	422	428	397	**236**	531	373	258	336	510	503	481	472	342	410	**361**	456	371	478	421	327	372	481	430	425	403	**486**	577	512	478	421	549	503	460	236	421	354
112	463	378	431	364	539	372	481	428	406	427	**237**	428	413	507	406	510	503	481	335	435	478	**362**	531	420	478	461	517	490	219	459	458	485	**487**	577	460	478	361	510	312	385	459	392	485
113	439	371	405	392	407	445	481	451	304	361	**238**	439	483	347	392	298	490	481	472	360	349	**363**	356	371	478	450	538	319	488	459	385	478	**488**	577	470	478	457	510	445	488	290	450	425
114	351	420	347	506	539	469	481	428	435	397	**239**	304	371	513	406	517	490	424	420	435	478	**364**	439	420	376	414	539	445	236	459	493	478	**489**	577	338	478	493	510	468	360	459	421	485
115	481	609	450	406	327	490	481	488	346	397	**240**	531	425	478	458	350	490	488	466	398	422	**365**	497	371	478	414	442	347	488	459	447	478	**490**	450	470	478	421	510	490	488	350	421	284
116	435	413	456	470	510	469	481	428	435	393	**241**	584	371	478	406	546	490	361	459	435	485	**366**	531	420	478	414	510	503	471	459	458	478	**491**	577	506	478	374	510	234	374	459	421	485
117	531	446	421	412	290	445	338	458	407	390	**242**	439	440	478	400	452	490	481	459	435	282	**367**	531	473	478	414	380	423	481	459	457	478	**492**	515	470	478	457	510	490	481	432	469	347
118	531	413	405	387	517	468	438	428	435	397	**243**	346	378	478	406	539	490	425	459	399	485	**368**	531	371	478	414	539	445	356	459	493	284	**493**	577	477	478	421	551	253	310	459	456	485
119	531	371	421	406	392	490	431	449	435	397	**244**	439	393	478	267	423	433	495	459	435	348	**369**	531	378	478	414	386	452	481	459	458	498	**494**	449	512	478	414	510	490	488	346	410	334
120	531	420	405	333	539	432	431	399	399	397	**245**	468	407	478	406	517	445	243	459	399	478	**370**	531	413	478	414	510	445	481	459	458	198	**495**	577	442	478	244	377	328	214	459	450	485
121	441	371	421	406	564	503	363	436	435	340	**246**	531	320	478	429	255	346	488	459	435	478	**371**	531	279	478	332	357	479	481	459	458	478	**496**	356	512	478	457	517	503	488	401	378	478
122	531	443	405	412	510	176	431	399	399	368	**247**	468	371	478	406	510	490	461	459	399	478	**372**	233	378	478	457	517	445	497	459	458	284	**497**	535	216	478	310	284	420	226	459	450	478
123	476	407	421	406	401	490	448	377	435	474	**248**	531	290	478	314	436	372	495	459	435	478	**373**	439	421	478	282	424	503	488	459	458	478	**498**	272	470	478	414	546	490	488	414	339	447
124	531	471	450	406	510	291	481	459	232	397	**249**	272	378	471	399	539	445	481	459	430	478	**374**	300	378	478	457	539	490	438	291	458	339	**499**	542	412	499	414	423	490	481	459	450	485

500	447	512	478	414	517	490	481	459	375	396
501	577	512	403	414	453	490	481	459	456	478
502	410	512	507	414	510	490	481	459	375	478
503	599	512	382	414	480	490	481	459	450	478
504	577	512	507	414	510	490	481	459	421	478
505	577	512	327	414	443	490	481	459	421	478
506	577	512	471	414	539	490	501	459	474	478
507	577	512	405	414	378	490	488	459	456	449
508	577	591	478	477	539	517	307	459	447	478
509	346	470	353	450	429	503	481	459	456	465
510	535	343	513	504	517	529	348	459	464	478
511	468	512	478	421	510	490	481	291	450	420
512	577	309	478	415	510	499	474	472	456	478
513	577	505	234	421	510	445	481	528	421	369
514	577	335	478	439	510	503	434	459	421	485
515	330	470	357	457	510	490	481	398	421	440
516	599	472	478	432	510	375	497	459	421	478
517	278	505	465	421	510	503	495	459	421	440
518	599	471	507	254	404	272	451	459	421	485
519	344	470	478	457	517	503	481	355	421	396
520	535	460	478	270	472	416	176	472	392	485
521	577	470	478	450	539	490	488	440	450	375
522	577	434	478	443	386	442	481	459	421	485
523	577	512	478	421	510	503	495	470	528	478
524	577	484	478	414	393	312	481	466	456	478
525	577	512	478	414	539	445	481	505	347	478
526	406	512	478	414	292	468	481	498	450	478
527	577	512	478	414	539	490	481	511	254	478
528	564	512	478	414	431	490	481	498	450	478
529	577	512	478	414	510	490	481	511	469	478
530	233	512	355	414	357	490	481	498	456	443
531	535	512	471	414	517	490	481	511	395	478
532	365	512	406	414	539	490	481	498	450	416
533	535	409	478	443	510	490	450	511	439	478
534	243	512	262	450	510	490	481	469	450	207
535	535	459	513	379	510	490	225	498	385	485
536	282	512	499	457	510	490	488	430	450	384
537	599	233	478	393	510	425	434	505	267	498
538	577	470	403	450	510	490	488	403	450	422
539	577	450	507	387	328	379	342	505	289	478
540	577	505	428	421	510	490	481	243	450	401
541	577	524	478	437	344	393	435	498	456	478
542	577	470	427	457	517	490	488	413	421	478
543	577	460	478	421	384	401	345	498	421	478
544	577	512	431	414	517	445	488	429	421	478
545	432	504	478	207	395	437	431	511	421	478
546	577	512	290	457	510	490	488	404	421	478
547	476	437	471	421	423	490	481	498	421	478
548	542	512	478	414	517	503	481	372	421	478
549	542	432	478	443	412	282	481	505	421	356
550	577	470	478	421	510	497	481	512	421	478
551	450	499	478	414	480	355	481	498	421	427
552	599	512	478	414	510	445	481	511	455	478
553	535	512	478	414	489	269	481	498	456	403
554	577	512	478	414	539	490	481	511	426	485
555	467	512	478	414	510	486	481	498	450	321
556	584	512	478	414	510	445	385	511	350	478
557	584	512	478	414	510	488	481	498	450	311
558	584	512	478	472	510	490	295	511	416	478
559	584	512	485	421	510	490	488	384	450	325
560	584	233	478	247	510	490	410	498	378	485
561	584	470	405	443	510	490	460	505	450	313
562	584	334	507	442	466	490	341	498	325	485
563	584	512	478	421	510	490	488	452	456	275
564	584	459	478	414	510	490	167	511	499	485
565	584	512	404	414	510	490	488	383	456	478
566	319	273	471	414	510	490	404	469	481	478
567	584	505	373	414	510	490	488	291	456	478
568	478	469	507	414	510	278	392	505	275	478
569	605	512	206	283	320	490	481	498	450	478
570	490	297	513	450	510	490	269	511	479	356
571	584	476	478	376	539	490	488	176	456	498
572	447	392	478	421	510	490	481	498	421	421
573	599	505	430	329	264	490	481	511	421	478
574	439	450	471	457	517	490	481	498	421	490
575	584	505	453	481	409	490	481	272	421	478
576	438	470	478	450	510	490	481	498	421	449
577	605	512	450	330	510	490	481	511	421	498
578	535	512	507	421	517	490	481	498	421	480
579	577	512	478	384	363	487	400	511	445	478
580	419	512	478	457	517	490	488	498	450	478
581	584	512	478	469	205	445	457	511	346	478
582	502	512	478	421	552	490	495	498	450	478
583	542	512	478	412	376	404	430	511	478	478
584	577	512	478	421	552	490	460	347	456	478
585	577	334	478	373	510	280	495	498	410	254
586	577	512	478	450	510	490	481	551	450	478
587	577	397	478	199	510	375	480	511	455	419
588	577	470	478	457	510	503	467	498	450	485
589	442	393	478	234	510	487	448	511	363	478
590	577	470	355	457	539	490	460	420	450	478
591	577	483	471	421	510	404	384	498	392	478
592	577	512	536	414	251	445	481	433	450	360
593	577	512	471	414	517	443	391	498	352	478
594	577	512	478	414	340	503	460	317	320	469
595	577	396	478	414	552	439	374	440	456	478
596	423	470	456	414	510	490	348	505	308	378
597	584	353	478	414	510	487	481	451	450	485
598	392	509	427	414	240	445	308	498	395	467
599	535	476	478	414	552	263	460	511	450	478
600	460	499	273	294	299	503	468	498	503	338
601	577	512	444	414	517	490	488	383	456	478
602	298	553	507	452	539	465	312	469	441	485
603	599	470	478	421	510	445	488	461	456	478
604	519	355	478	435	354	375	440	498	449	458
605	535	505	411	421	539	445	481	404	456	485
606	367	475	471	506	431	372	481	505	466	478
607	599	512	258	450	539	445	481	260	456	478
608	577	441	507	430	510	413	481	505	421	478
609	577	476	478	414	510	503	481	498	421	478
610	577	267	478	357	510	506	481	511	421	478
611	577	505	473	421	510	445	481	498	421	478
612	577	512	478	492	510	437	481	511	421	478
613	332	512	347	421	510	503	481	498	421	410
614	535	512	478	410	510	234	481	511	421	478
615	298	512	431	457	443	490	451	498	423	349
616	584	512	471	421	546	225	481	394	456	478
617	419	512	478	414	539	490	390	498	275	318
618	535	512	478	414	510	269	460	298	450	498
619	388	512	478	414	460	503	469	498	291	442
620	599	512	478	414	539	420	481	291	450	485
621	577	431	478	414	481	490	481	505	449	478
622	577	470	478	414	517	355	481	372	456	478
623	450	469	478	414	542	445	347	505	457	431
624	577	470	478	414	510	450	488	376	456	478
625	577	393	478	232	421	490	206	505	550	478
626	577	476	392	457	552	490	488	365	421	478
627	577	384	478	451	366	490	526	505	469	478
628	577	512	427	421	517	490	495	511	498	478
629	577	283	478	279	539	490	501	498	431	478
630	460	470	251	457	510	490	488	395	498	478
631	542	392	471	472	510	490	481	505	498	478
632	282	512	353	457	510	490	481	498	498	478
633	599	335	513	375	510	490	481	511	498	478
634	605	512	478	457	433	490	481	498	498	368
635	577	177	478	364	552	499	481	511	498	478
636	304	470	234	457	464	445	481	498	498	498
637	584	510	478	421	517	503	481	511	498	478
638	243	512	357	414	414	490	307	498	498	368
639	599	512	478	437	510	404	481	511	498	478
640	307	512	465	457	356	503	414	498	392	410
641	542	512	478	421	517	285	495	375	475	478
642	474	512	457	414	360	503	431	476	327	432
643	535	512	507	414	517	407	488	327	498	485
644	577	512	413	414	510	490	457	498	278	310
645	577	512	484	414	517	361	495	284	475	478
646	577	512	478	414	193	490	439	498	363	485
647	577	512	478	414	517	289	467	398	462	478
648	577	512	478	414	233	490	425	498	358	478
649	564	512	478	414	539	354	495	402	498	478
650	577	479	478	414	510	503	279	498	465	478
651	233	505	478	414	510	334	495	461	498	478
652	535	476	478	451	510	490	310	498	493	478
653	304	512	478	421	510	423	481	391	498	478
654	599	460	478	294	510	490	491	505	352	478
655	243	470	478	421	350	308	488	476	498	207
656	535	447	478	481	546	490	345	505	498	485
657	338	470	431	457	520	490	481	343	498	391
658	542	366	478	254	517	490	481	505	498	478
659	577	512	430	414	385	490	481	498	498	460
660	577	456	471	442	517	490	481	511	498	485
661	276	476	450	457	358	490	481	498	498	413
662	599	379	507	384	510	490	481	511	498	485
663	577	470	464	414	509	490	481	498	498	442
664	577	205	471	413	539	490	481	511	498	485
665	577	505	427	421	518	490	432	498	497	478
666	577	470	478	387	552	490	488	373	446	478
667	577	512	402	457	414	445	403	498	499	478
668	419	512	507	421	539	468	481	457	498	478
669	535	512	290	414	489	490	293	505	420	478
670	233	512	471	414	510	493	481	401	446	478
671	599	512	478	414	510	445	458	511	384	478
672	397	512	478	414	510	499	481	443	498	431
673	599	512	234	414	510	445	462	505	346	478
674	291	512	478	414	510	448	481	253	498	400
675	584	457	478	414	510	445	497	505	383	478
676	278	505	478	414	510	319	495	347	462	434
677	599	473	478	464	510	503	451	498	334	485
678	356	470	478	421	510	457	481	270	469	405
679	535	422	478	479	510	490	481	498	232	485
680	328	470	478	421	429	407	481	225	462	444
681	584	499	478	414	517	490	481	505	469	478
682	584	512	478	414	380	361	481	498	498	478
683	584	476	478	404	510	490	481	511	498	478
684	584	505	478	457	404	289	481	498	498	478
685	584	421	478	381	539	490	481	511	498	478
686	584	512	446	457	379	490	481	498	498	478
687	584	347	478	432	517	490	481	511	498	478
688	584	470	379	457	320	490	226	498	532	478
689	584	292	507	349	539	490	488	548	475	490
690	584	470	329	414	382	490	247	498	426	478
691	490	512	513	254	510	490	488	450	475	461
692	584	512	197	457	320	490	448	498	273	478
693	475	512	513	421	517	490	495	375	498	396
694	584	512	478	414	539	490	405	469	527	485
695	439	512	478	414	510	419	481	494	498	388
696	577	512	442	414	510	445	481	505	332	478
697	319	512	478	414	510	477	495	488	498	260
698	605	512	475	414	510	445	393	505	456	478
699	456	512	478	414	510	411	488	249	498	347
700	535	506	448	414	510	445	481	498	498	478
701	406	470	471	414	327	357	481	404	498	478
702	584	477	392	326	510	445	481	505	498	478
703	354	512	478	421	245	468	481	426	498	478
704	584	464	478	408	539	490	481	498	498	478
705	289	512	478	421	409	319	481	404	498	478
706	599	432	478	408	510	490	481	498	498	478
707	577	512	478	421	371	362	419	372	498	478
708	577	458	478	399	510	503	488	505	498	226
709	577	470	478	421	401	499	347	498	521	478
710	577	427	478	358	510	445	488	511	498	326
711	577	470	478	421	463	503	226	498	608	478
712	429	480	478	426	539	490	488	511	550	463
713	584	505	471	421	402	355	497	498	408	478
714	490	296	499	387	539	503	490	511	550	167
715	584	470	355	421	510	503	391	498	395	478
716	444	512	470	414	510	490	483	511	492	347
717	584	512	263	414	510	490	319	498	468	478
718	233	512	505	414	510	490	526	443	550	485
719	535	512	494	414	510	490	278	469	506	478
720	599	512	470	414	510	490	512	439	608	478
721	577	512	434	414	510	490	490	505	550	478
722	442	512	470	414	322	490	526	516	550	478
723	577	512	372	414	510	490	490	511	550	478
724	429	512	470	414	364	490	526	423	550	478
725	584	214	355	414	517	490	490	498	550	431
726	584	470	470	414	356	329	526	461	289	478
727	584	449	370	254	517	445	490	498	550	244
728	584	505	470	457	494	328	282	401	327	485
729	584	358	499	479	510	490	483	505	608	497
730	584	470	499	421	556	392	253	437	321	478
731	584	429	499	414	510	490	512	498	544	368
732	584	505	499	414	372	419	341	511	395	478
733	271	376	499	403	517	445	506	498	550	501
734	535	476	499	457	329	512	483	511	355	478
735	321	488	499	450	510	445	490	498	608	478
736	584	512	334	421	510	411	312	511	454	478
737	571	525	499	319	510	445	512	498	608	478
738	577	476	345	457	510	386	469	511	304	478
739	365	512	470	505	510	445	490	169	550	478
740	535	512	354	421	510	439	340	498	550	478
741	372	512	470	414	348	490	519	260	550	478
742	584	512	349	414	539	319	490	505	550	473
743	480	512	499	414	424	490	526	498	550	478
744	577	512	471	414	510	362	490	511	550	422
745	577	512	499	414	233	503	526	500	459	478
746	577	512	356	414	539	490	490	498	608	401
747	577	512	492	414	328	490	526	197	463	478
748	577	512	499	414	510	490	490	476	550	461
749	577	512	499	414	551	490	473	505	356	478
750	356	286	499	349	510	490	490	498	608	508
751	535	470	499	421	205	490	345	433	521	478
752	367	512	499	279	552	490	512	498	550	501
753	599	512	499	457	510	490	468	511	378	478
754	419	512	499	461	510	490	526	498	550	478
755	535	512	462	421	510	490	448	511	550	478
756	495	205	499	361	510	490	519	498	550	478
757	599	512	340	457	510	337	461	511	550	478
758	341	270	492	491	510	445	490	498	550	478
759	577	470	477	450	510	503	522	511	550	393
760	298	459	499	387	299	490	526	498	400	485
761	599	512	451	421	552	420	488	511	550	420
762	577	273	492	437	453	490	526	422	429	478
763	577	505	505	457	539	404	490	498	608	401
764	577	469	499	359	442	445	526	451	437	478
765	577	512	524	457	510	449	490	498	550	438
766	577	512	499	421	386	503	526	320	464	498
767	382	512	424	414	510	375	490	498	550	377
768	599	512	470	414	329	445	526	308	475	485
769	341	512	499	414	510	235	490	505	608	475
770	599	512	499	414	510	445	476	346	347	478
771	457	458	499	414	510	471	512	498	544	478
772	577	512	499	414	510	445	448	319	557	478
773	298	507	499	414	510	503	490	505	608	478
774	584	512	499	414	510	490	412	498	550	478
775	419	421	499	521	510	490	483	511	550	478
776	535	505	471	450	510	490	419	498	550	478
777	388	434	492	427	510	490	512	511	550	478
778	599	470	456	421	510	490	329	498	550	478
779	605	456	470	472	261	490	519	511	421	478
780	577	470	456	421	539	490	365	498	608	269
781	577	484	499	265	292	490	490	511	350	485
782	577	512	398	457	539	490	451	498	608	485
783	577	417	492	423	279	490	519	478	410	478
784	577	505	273	450	517	269	490	505	608	480
785	577	379	505	355	393	497	526	498	330	478
786	300	470	402	414	517	392	490	511	550	478
787	599	512	499	449	359	490	526	317	511	478
788	365	512	499	457	539	298	490	511	608	386
789	577	512	499	416	466	490	526	476	450	478
790	468	512	499	421	510	278	490	505	544	421
791	577	512	499	414	510	490	442	343	504	485
792	355	512	499	414	539	421	512	505	608	478
793	535	512	499	414	510	445	460	437	550	478
794	304	512	499	414	510	256	526	505	550	478
795	584	512	456	414	510	490	319	373	550	478
796	233	512	470	414	510	439	512	505	550	478
797	599	512	451	414	510	445	490	498	550	478
798	577	408	492	414	510	426	526	511	443	478
799	577	512	469	414	510	503	483	498	608	300
800	577	393	499	348	510	393	526	511	381	485
801	577	512	391	414	510	490	424	498	608	393
802	577	423	470	197	376	490	506	511	401	485
803	233	470	499	414	552	490	490	498	550	478
804	599	421	499	368	463	490	526	511	213	478
805	282	505	441	414	539	490	490	498	550	471
806	535	428	499	404	254	490	526	443	402	485
807	434	476	440	457	517	490	490	469	608	434
808	584	412	505	381	361	490	526	191	426	485
809	383	512	499	457	539	490	490	476	550	449
810	584	524	499	479	536	490	226	347	455	478
811	542	553	499	457	510	490	512	476	550	478
812	577	553	499	483	510	490	483	243	550	478
813	429	553	499	457	510	365	481	505	550	478
814	584	553	499	442	510	445	422	225	550	478
815	584	553	499	421	510	262	495	505	550	478
816	584	553	235	414	510	490	405	291	550	478
817	584	553	470	414	510	407	488	505	368	478
818	584	553	296	414	510	445	410	498	544	254
819	584	553	470	414	299	465	488	511	332	478
820	584	553	373	414	517	497	429	498	550	463
821	584	553	492	414	510	269	481	511	375	478
822	584	553	279	414	517	497	481	498	544	422
823	584	553	492	414	354	490	481	511	289	478
824	584	553	393	414	517	445	481	498	550	420
825	584	553	492	410	539	497	481	511	260	498
826	605	553	359	457	510	499	481	498	550	421
827	584	524	499	234	442	442	481	225	550	478
828	577	553	473	457	510	521	481	505	550	478
829	584	382	499	307	469	499	481	493	550	478
830	403	553	546	457	510	528	481	498	550	478
831	535	267	499	391	510	499	362	423	550	478
832	278	546	499	457	510	499	488	498	455	478
833	599	244	499	260	510	557	226	414	608	478
834	360	546	499	457	510	557	488	498	411	478
835	599	466	499	404	510	557	384	393	550	206
836	453	553	499	457	510	445	481	505	321	485
837	535	422	499	326	510	503	384	451	544	483
838	577	553	499	421	416	490	481	498	486	498
839	577	401	263	414	510	490	427	511	550	553
840	577	553	505	414	558	337	481	498	477	485
841	577	421	418	414	510	445	419	511	550	520
842	577	546	492	414	395	484	488	498	455	478
843	282	553	405	414	510	445	481	511	492	478
844	535	553	492	414	292	290	481	498	608	478
845	535	553	347	414	546	490	481	511	550	478
846	577	553	470	414	402	573	481	419	550	478
847	319	553	487	414	539	445	481	505	550	478
848	535	553	470	414	309	319	481	411	550	306
849	381	553	536	414	552	503	481	498	356	478
850	605	553	470	515	394	457	282	450	608	364
851	326	553	499	421	539	490	488	498	487	478
852	535	553	499	573	510	407	481	349	550	397
853	391	553	499	421	510	490	481	505	359	478
854	599	553	499	475	510	490	335	487	550	260
855	584	553	499	421	510	490	488	498	550	485
856	584	348	499	417	510	490	362	478	608	496
857	584	553	499	421	510	490	488	505	407	478
858	584	488	357	439	510	490	319	498	608	319
859	584	553	499	457	385	490	460	511	439	478
860	584	506	488	487	510	490	382	498	608	478
861	584	553	470	421	420	490	488	511	550	478
862	584	498	354	414	510	490	481	498	550	478
863	584	553	470	414	520	437	481	511	550	478
864	584	442	470	414	517	490	481	498	550	478
865	584	553	499	414	385	490	481	511	550	478
866	384	479	449	414	517	445	481	498	360	478
867	535	553	499	414	329	490	481	511	544	377
868	364	438	328	414	510	490	481	498	528	478
869	577	553	492	443	480	282	481	383	608	365
870	250	427	448	421	539	503	481	469	482	478
871	599	553	470	472	510	468	399	505	544	424
872	613	553	499	421	510	490	495	498	499	485
873	584	553	499	426	510	384	452	375	608	347
874	420	553	499	421	510	490	460	476	378	485
875	584	553	499	412	510	328	357	505	492	425
876	393	553	499	421	424	503	481	498	384	485
877	535	553	499	414	510	494	262	461	550	278
878	577	553	499	414	299	503	488	511	550	485
879	577	553	564	368	539	488	358	498	550	478
880	577	553	492	443	480	503	481	511	550	478
881	577	553	499	497	510	503	307	498	563	478
882	577	553	499	421	344	490	481	511	550	478
883	389	553	487	414	517	490	386	498	233	478
884	577	553	470	414	384	490	460	511	544	478
885	403	448	511	414	517	490	495	498	320	478
886	535	553	499	414	395	490	481	511	608	368
887	498	458	309	414	510	490	481	498	516	485
888	542	549	492	414	423	490	481	511	608	284
889	495	374	453	414	517	490	481	498	359	498
890	584	553	499	443	539	490	481	511	492	218
891	457	440	499	421	510	490	481	498	608	478
892	542	553	499	414	510	400	481	490	550	485
893	577	386	499	414	510	445	481	498	550	478
894	577	553	499	420	510	430	526	431	550	368
895	577	327	499	414	510	503	495	498	550	478
896	577	546	499	414	510	346	439	413	550	485
897	577	405	499	414	309	490	481	498	550	478
898	480	553	362	235	552	430	356	391	550	478
899	584	413	470	450	423	490	481	505	608	478
900	429	553	397	402	552	493	438	321	446	478
901	577	553	470	421	262	445	488	498	550	478
902	384	553	438	539	517	503	438	508	360	478
903	535	553	499	421	489	490	495	511	544	478
904	437	553	398	414	539	461	311	498	499	478
905	584	553	492	414	392	497	481	511	608	397
906	462	553	318	414	552	282	481	498	379	485
907	535	553	505	414	450	503	481	511	492	422
908	577	553	458	414	517	468	481	498	550	478
909	577	553	470	414	488	490	481	511	550	295
910	577	470	499	414	510	384	481	498	550	485
911	577	553	499	414	510	490	481	452	550	348
912	577	467	499	414	510	490	481	498	550	478
913	535	553	499	414	510	490	481	508	329	284
914	577	369	499	414	510	490	481	498	537	498
915	442	553	499	247	510	490	425	391	450	198
916	577	477	499	443	510	490	495	505	608	478
917	429	553	453	451	510	490	393	479	493	485
918	584	509	499	421	488	490	488	498	492	478
919	584	549	334	384	510	490	282	544	464	478
920	584	299	470	414	309	490	460	511	608	478
921	403	553	373	443	517	243	348	422	225	478
922	535	438	470	421	415	490	481	505	544	478
923	469	553	406	458	510	464	298	498	516	478
924	542	553	470	457	452	490	481	511	608	262
925	577	553	551	421	539	260	205	498	544	478
926	577	553	499	414	459	490	488	511	550	395
927	577	553	430	414	539	461	481	498	608	478
928	577	553	470	414	510	490	481	511	544	444
929	577	553	499	414	510	297	481	498	550	485
930	491	553	499	414	510	490	481	511	608	403
931	535	553	499	414	510	385	481	498	544	485
932	476	553	499	414	510	503	481	472	346	265
933	577	553	499	414	510	487	481	498	608	478
934	401	553	499	414	510	490	481	254	401	384
935	584	315	499	414	510	451	481	498	544	485
936	460	549	309	414	510	445	345	488	421	478
937	584	387	492	414	405	503	481	498	608	478
938	384	546	401	529	539	490	386	384	373	478
939	535	446	470	421	376	490	488	498	608	478
940	577	553	257	397	510	490	363	411	507	478
941	577	465	492	421	480	490	488	498	608	478
942	577	553	460	291	510	490	385	354	522	478
943	577	437	470	421	364	490	481	498	550	478
944	577	546	450	413	517	490	427	442	608	478
945	535	412	492	457	251	490	481	498	544	207
946	577	553	457	356	517	490	536	511	550	485
947	577	355	492	457	455	490	488	498	608	480
948	577	549	501	253	510	432	481	511	544	478
949	351	517	492	443	510	445	481	498	550	504
950	599	553	499	414	510	497	481	511	608	478
951	569	553	499	414	510	445	481	498	355	406
952	584	553	499	414	510	514	481	511	544	478
953	448	553	499	414	510	503	481	498	404	478
954	599	553	499	414	510	160	481	511	537	478
955	577	553	499	414	510	503	481	243	289	478
956	577	553	499	414	354	326	481	498	544	478
957	577	553	524	414	539	490	414	511	260	478
958	577	553	470	414	320	179	488	498	544	478
959	577	553	502	353	539	490	291	225	341	478
960	577	553	492	457	471	490	481	505	544	396
961	577	553	499	445	510	490	374	384	468	478
962	395	553	499	457	327	490	488	476	537	431
963	584	553	340	463	552	490	328	505	550	485
964	335	427	492	421	328	490	488	498	608	499
965	584	553	499	341	510	490	490	511	544	478
966	360	493	499	414	344	490	444	498	550	533
967	599	553	451	462	517	490	481	511	608	478
968	356	524	492	421	539	490	453	498	544	454
969	535	553	499	470	510	490	453	511	550	478
970	243	398	499	421	510	490	453	498	378	471
971	535	553	499	414	510	399	571	511	544	471
972	577	290	499	414	510	445	571	498	319	471
973	577	546	499	414	510	407	571	511	544	471
974	577	384	499	414	510	503	478	498	499	471
975	577	517	499	414	440	479	571	511	608	471
976	577	335	383	414	517	490	441	443	430	471
977	406	553	505	414	292	475	485	469	537	328
978	584	372	376	414	510	445	189	191	415	506
979	319	553	499	414	465	440	471	476	608	497
980	535	553	516	392	539	490	347	347	544	471
981	319	553	499	414	309	459	471	476	550	454
982	535	553	370	468	552	445	321	336	608	471
983	271	553	470	421	491	468	507	505	544	471
984	599	553	244	487	510	445	431	225	550	471
985	412	553	505	421	262	503	478	505	608	253
986	535	553	499	459	517	490	262	498	544	513
987	577	553	499	421	510	490	478	511	439	253
988	577	553	499	247	510	490	478	498	544	471
989	577	553	499	443	510	490	478	511	441	471
990	577	553	499	423	510	490	478	498	608	471
991	577	285	499	421	510	490	478	511	426	471
992	440	546	499	414	510	490	478	498	544	471
993	535	327	327	414	510	490	478	511	463	471
994	319	546	470	414	461	490	478	498	550	471
995	605	392	392	414	510	490	397	370	412	471
996	278	549	505	414	549	357	571	498	544	207
997	535	327	327	414	510	490	385	336	608	506
998	410	553	470	414	466	356	485	469	544	233
999	599	412	418	414	539	445	236	462	550	506

Appendix C

RLState class

```
class CvRLState
{

public:
     struct CvCityPos
     {
          int       iX;
          int       iY;
          int       iCityRank;
     };

protected:

     //the existing cities in this state
     CvCityPos* m_paCities;
     //number of cities of the current player in this state
     int m_iNumCities;
     //number of states that succeed this state directly
     int m_iNumSubStates;
     //the current Q-value
     float m_fStateVal;
     //the preceding state
     CvRLState* m_pPrevState;
     //the following states
     CvRLState* m_paSubStates;
     //when was the last city founded
     int m_iFoundTurn;
     //does this state use pre-initialised values
     bool m_bPreInit;
     //what was the game score before the last city was founded
     int m_iPrevScoreScore;
     //if the current algorithm uses eligibility traces, its value is stored here
     float m_fEligibility;
};
```

Listing C.1: CvRLState.h

References

Aha, D. W. and Molineaux, M. (2004). Integrating Learning in Interactive Gaming Simulators. Technical report, Intelligent Decision Aids Group; Navy Center for Applied Research in Artificial Intelligence.

Aha, D. W. and Molineaux, M. (2008). Learning Continuous Action Models in a Real-Time Strategy Environment.

Andrade, G., Ramalho, G., Santana, H., and Corruble, V. (2005). Automatic Computer Game Balancing: A Reinforcement Learning Approach. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1111–1112, New York, NY, USA. ACM.

Auslander, B., Lee-Urban, S., Hogg, C., and Muñoz-Avila, H. (2008). Recognizing the Enemy: Combining Reinforcement Learning with Strategy Selection using Case-Based Reasoning. In *Proceedings of the 9th European Conference on Advances in Case-Based Reasoning (ECCBR-08)*. Springer.

Bellman, R. (1957a). A Markov Decision Process. *Journal of Mathematical Mechanics*, 6:679–684.

Bellman, R. (1957b). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

BetterAI Project (2007). Civilization IV Better AI. URL: `http://sourceforge.net/projects/civ4betterai/` [last checked: 20/02/2009].

Billings, D., Pena, L., Schaeffer, J., and Szafron, D. (1999). Using Probabilistic Knowledge and Simulation to Play Poker. In *Proceedings of the National Conference on Artificial Intelligence*.

Campbell, M., Jr., A. H., and Hsu, F.-H. (2002). Deep Blue. *Artificial Intelligence*, 134 no. 1-2:57–83.

Champandard, A. (2003). *AI Game Development: Synthetic Creatures with Learning and Reactive Behavior*. New Riders Games.

Chellapilla, K. and Fogel, D. (1999). Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge. *IEEE Trans. Neural Networks*, 10(6):1382–1391.

Dahl, F. A. (2001). A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold'em Poker. In *Proceedings of the 12th European Conference on Machine Learning*. Springer-Verlag.

Farley, B. and Clark, W. (Sep 1954). Simulation of Self-Organizing Systems by Digital Computer. *Information Theory, IEEE Transactions on*, 4(4):76–84.

Firaxis Games (2005). Civilization IV. URL: `http://www.2kgames.com/civ4/home.htm` [last checked: 20/02/2009].

Freeciv Project (2008). FreeCiv. URL: `http://freeciv.wikia.com/` [last checked: 20/02/2009].

Fuernkranz, J. (2001). *Machine Learning in Games: A Survey*, pages 11–59. Nova Biomedical.

Gasser, R. (1996). *Solving Nine MenŠs Morris*, pages 101–113. Cambridge University Press, Cambridge, MA.

Gerlach, S. (2008). C-evo: Empire Building Game. URL: `http://c-evo.org/` [last checked: 20/02/2009].

Gold, A. (2005). Academic AI and Video Games: A Case Study of Incorporating Innovative Academic Research into a Video Game Prototype. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*. IEEE.

Graepel, T., Herbrich, R., and Gold, J. (2004). Learning to Fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*.

Gundevia, U. (2006). Integrating War Game Simulations with AI Testbeds: Integrating Call To Power 2 with TIELT. Master's thesis, Lehigh University.

Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.

Houk, P. A. (2004). A Strategic Game Playing Agent for FreeCiv. Technical Report NWU-CS-04-29, Northwestern University, Evanston, IL.

Karpov, I., D'Silva, T., Varrichio, C., Stanley, K., and Miikkulainen, R. (May 2006). Integration and Evaluation of Exploration-Based Learning in Games. *Computational Intelligence and Games, 2006 IEEE Symposium on*, 1:39–44.

Kolodner, J. (1992). *Case-Based Reasoning*. Morgan Kaufmann.

Korb, K. B. and Nicholson, A. (1999). Bayesian Poker. In *UAI'99 - Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence*, pages 343–350.

Krulwich, B. L. (1993). *Flexible Learning in a Multi-Component Planning System*. PhD thesis, The institute for the Learning Sciences, Northwestern University, Evanston, IL.

Laird, J. and van Lent, M. (2001). Human-level AI's Killer Application: Interactive Computer Games. *AI Magazine*, Summer 2001:1171–1178.

Lee, K.-F. and Mahajan, S. (1990). The Development of a World Class Othello Program. *Artificial Intelligence*, 43(1):21–36.

McPartland, M. and Gallagher, M. (2008a). Creating a Multi-Purpose First Person Shooter Bot with Reinforcement Learning. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games (CIG'08) (to appear)*, pages 143–150.

McPartland, M. and Gallagher, M. (2008b). Learning to be a Bot: Reinforcement Learning in Shooter Games. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, Stanford, California. AAAI, AAAI Press.

Merrick, K. E. (2007). Modeling Motivation for Adaptive Nonplayer Characters in Dynamic Computer Game Worlds. *Comput. Entertain.*, 5(4):1–32.

Merrick, K. E. and Maher, M. L. (2006). Motivated Reinforcement Learning for Non-Player Characters in Persistent Computer Game Worlds. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 3, New York, NY, USA. ACM.

Merrick, K. E. and Maher, M. L. (2007). Motivated Reinforcement Learning for Adaptive Characters in Open-Ended Simulation Games. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pages 127–134, New York, NY, USA. ACM.

Miikkulainen, R., Bryant, B., Cornelius, R., Karpov, I., Stanley, K., and Yong, C. H. (2006). *Computational Intelligence in Games*. IEEE Computational Intelligence Society, Piscataway, NJ.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-Bbased Generalization: A Unifying View. *Machine Learning*, 1(1):47–80.

Mueller, M. (2000). *Generalized Thermography: A New Approach to Evaluation in Computer Go*, pages 203–219. Universiteit Maastricht, Maastricht.

Muggleton, S. (1990). *Inductive Acquisition of Expert Knowledge*. Turing Institute Press, Addison Wesley.

Nareyek, A. (2004). Computer Games - Boon or Bane for AI Research? *Künstliche Intelligenz*, 18(1):43–44.

Nareyek, A. (2007). Game AI is Dead. Long Live Game AI! *Intelligent Systems*, 22(1):9–11.

Peng, J. and Williams, R. J. (1994). Incremental Multi-Step Q-Learning. In *Machine Learning*, pages 226–232. Morgan Kaufmann.

Ponsen, M., Muñoz-Avila, H., Spronck, P., and Aha, D. (2006). Automatically Generating Game Tactics through Evolutionary Learning. *AI Magazine*.

Quinlan, J. R. (1983). *Learning Efficient Classification Procedures*, pages 463–482. Tioga, Palo Alto.

Rubin, J. and Watson, I. (2007). Investigating the Effectiveness of Applying Case-Based Reasoning to the Game of Texas Hold'em. In *Proc. of the 20th. Florida Artificial Intelligence Research Society Conference (FLAIRS)*. AAAI Press.

Rummery, G. A. and Niranjan, M. (1994). On-Line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department. URL: `citeseer.ist.psu.edu/rummery94line.html`.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):211–229.

Samuel, A. L. (1967). Some Studies in Machine Learning Using the Game of Checkers.ii - Recent Progress. *IBM Journal of Research and Development*, 11(6):601–617.

Sánchez-Pelegrín, R., Gómez-Martín, M. A., and Díaz-Agud, B. (2005). A CBR Module for a Strategy Videogame. In Aha, D. and Wilson, D., editors, *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)*.

Sánchez-Ruiz, A., Lee-Urban, S., Muñoz-Avila, H., Díaz-Agudoy, B., and González-Caleroy, P. (2007). Game AI for a Turn-Based Strategy Game with Plan Adaptation and Ontology-based Retrieval. In *Proceedings of the ICAPS 2007 Workshop on Planning in Games*.

Schaeffer, J. (2000). The Games Computer (and People) Play. *Academic Press*, 50:189–266.

Schaeffer, J. (2007). Checkmate for Checkers. URL: `http://www.nature.com/news/2007/070719/full/news070716-13.html` [last checked: 20/02/2009].

Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., and Szafron, D. (1992). A World Championship Caliber Checkers Program. *Artificial Intelligence*, 53 no. 2-3:273–290.

Schmidt, M. (1994). Temporal-Difference Learning and Chess. Technical report, University of Aarhus, Aarhus, Denmark.

Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41:265–275.

Sharma, M., Holmes, M., Santamaría, J. C., Irani, A., Jr., C. L. I., and Ram, A. (2007). Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In Veloso, M. M., editor, *IJCAI*, pages 1041–1046. URL: `http://dblp.uni-trier.de/db/conf/ijcai/ijcai2007.html#SharmaHSIIR07`.

Sheppard, B. (2002). World-Championship-Caliber Scrabble. *Artificial Intelligence*, 134:241–275.

Smith, M., Lee-Urban, S., and Muñoz-Avila, H. (2007). RETALIATE: Learning Winning Policies in First-Person Shooter Games. In *Proceedings of the Seventeenth Innovative Applications of Artifcial Intelligence Conference (IAAI-07)*, pages 1801–1806. AAAI Press.

Souto, J. H. (2007). A Turn-Based Strategy Game Testbed for Artificial Intelligence. Master's thesis, Lehigh University.

Spohrer, J. (1985). Learning Plans through Experience: A First Pass in the Chess Domain. In *Intelligent Robots and Computer Vision, Volume 579 of Proceedings of the SPIE - The International Society of Optical Engineering*, pages 518–527.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., and Postma, E. (2006). Adaptive Game AI with Dynamic Scripting. *Machine Learning*, 63(3):217–248.

Stanley, K., Bryant, B., and Miikkulainen, R. (Dec. 2005). Real-Time Neuroevolution in the NERO Video Game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653–668.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10:99–127.

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement Learning for RoboCup Soccer Keepaway. *Adaptive Behavior*, 13(3):165–188.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (2000). The Right Way to do Reinforcement Learning with Function Approximation. Talk at Neural Information Processing Systems 2000 (NIPS00).

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tesauro, G. (1992). Temporal Difference Learning of Backgammon Strategy. In *Proceedings of the 9th International Conference on Machine Learning 8*, pages 451–457.

Thorndike, E. (1911). *Animal Intelligence*. Hafner, Darien.

Tozour, P. (2002). *The Evolution of Game AI*, pages 3–15. Charles River Media, Hingham, MA.

Truscott, T. (1978). The Duke Checkers Program.

Tunstall-Pedoe, W. (1991). Genetic Algorithms Optimizing Evaluation Functions. *ICCA Journal*, 14(3):119–128.

Ulam, P., Goel, A., and Jones, J. (2004). Reflection in Action: Model-Based Self-Adaptation in Game Playing Agents. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence American Association for Artificial Intelligence (AAAI)*.

van Tiggelen, A. and van den Herik, H. J. (1991). *ALEXS: An Optimization Approach for the Endgame KNNKP(h)*, pages 161–177. Ellis Horwood, Chichester.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England.

Wender, S. and Watson, I. (2008). Using Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games (CIG'08) (to appear)*, pages 372–377.

Whiteson, S. and Stone, P. (2006). Evolutionary Function Approximation for Reinforcement Learning. *J. Mach. Learn. Res.*, 7:877–917.

Witten, I. H. (1977). An Adaptive Optimal Controller for Discrete-Time Markov Environments. *Information and Control*, 34:286–295.

Abbreviations

A-life	Artificial Life
AI	Artificial Intelligence
CBR	Case-Based Reasoning
CTP2	Call to Power 2
FPS	First-Person Shooter
GA	Genetic Algorithm
MDP	Markov Decision Process
MMOG	Massively Multiplayer Online Game
MRL	Motivated Reinforcement Learning
NEAT	NeuroEvolution of Augmented Topologies
NERO	Neuroevolving Robotic Operatives
NPC	Non-Player Character
RETALIATE	Reinforced Tactic Learning in Agent-Team Environments
RL	Reinforcement Learning
RPG	Role-Playing Game
RTS	Real-Time Strategy
SMDP	Semi-Markov Decision Process
TD	Temporal Difference
TIELT	Testbed for Integrating and Evaluating Learning Techniques