Rapid Adaptation of Video Game AI

Rapid Adaptation of Video Game AI

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Tilburg,
op gezag van de rector magnificus,
prof. dr. Ph. Eijlander,
in het openbaar te verdedigen ten overstaan van een
door het college voor promoties aangewezen commissie
in de aula van de Universiteit
op woensdag 3 maart 2010 om 16:15 uur

door

Sander Cornelus Johannes Bakkes
geboren op 27 januari 1980 te Swalmen

Netherlands Organisation for Scientific Research

The research reported in this thesis has been funded by NWO, the Netherlands Organization for Scientific Research, in the framework of the Rapid Online Learning for Entertainment Computing (ROLEC) project, with grant number 612.066.406.

The research reported in this thesis is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, with grant number BSIK03024.

SIKS Dissertation Series No. 2010-06
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

TiCC Ph.D. Series No. 11

ISBN 978-90-8559-926-5

Cover design by Job Pereboom, Studio Konstruktief

Preface

In the early 2000s, the video-game industry has grown to surpass the Hollywood movie industry in revenues. In recent years, video gaming has become a mass-market entertainment industry on a par with TV, movies, and music. Modern video games have become increasingly realistic in their visual and auditory presentation. However, the artificial intelligence (AI) of video games has not reached a high degree of realism yet. A major disadvantage of current game AI is its inability to adapt adequately to game circumstances. The disadvantage can be resolved by endowing game AI with adaptive behaviour, i.e., the ability to adapt to game circumstances, generally by means of learning adequate game behaviour while the game is in progress.

Current approaches to adaptive game AI typically require numerous trials to learn effective behaviour in online gameplay (i.e., game adaptation is not rapid). In addition, game developers are concerned that applying adaptive game AI may result in uncontrollable and unpredictable behaviour (i.e., game adaptation is not reliable). These characteristics hamper the incorporation of adaptive game AI in actual, commercially released video games. The general goal of our research is to investigate to what extent it is possible to establish game AI capable of adapting *rapidly* and *reliably* to game circumstances. Our findings, that are discussed in this thesis, may be used by game developers for incorporation in actual video games.

This thesis would not have been possible without the assistance and encouragement of a large group of people. I was most fortunate that my supervisor, Jaap van den Herik, allowed me to pursue my goals and I owe many thanks to his support and commitment. I highly appreciate his comments and advise on my scientific writings, as well as his efforts in arranging additional funding for finalising the research. A special gratitude goes out to my daily advisor, Pieter Spronck, for supporting me so very dedicatedly, for his research inspiration, and for the charm he brings to academia.

Moreover, I would like to thank my colleagues at Tilburg University and Maastricht University. The supportive staff of both universities deserve my appreciation for their help in many respects. I would especially like to mention Peter Geurtz for technical assistance in the past, and Joke Hellemons, Leen Jacobs, and Dennis Tantradjaja for their ongoing efforts in different areas of support. The research atmosphere that I experienced within the university walls is one that I happily look back upon. I consider myself lucky to have worked in an international environment, and secretly enjoyed the daily dosage of slightly unorthodox behaviour and thoughts that it brought about. For that, I acknowledge gratefully all direct colleagues, and roommates over the years. I would especially like to thank Steven de Jong, Guillaume Chaslot, Alessandro Rossi, Sander Spek, Andreas Rüdel, Sophia Katrenko, and Anne Bolders for their ongoing assistance, hospitality, and friendship.

During my research, I had the pleasure to supervise and cooperate with several students. I explicitly thank Laurens van der Blom, Philip Kerbusch, Bart Mehlkop, and Frederik Schadd, whose research efforts and their results have been integrated in the thesis. In addition, I thank Tom Nowell and Alexander Seizinger for their generous assistance in developing AI for the open-source SPRING game environment.

Outwith the academia, I found the support of individuals and organisations for expressing myself in the form of photography and DJ'ing. Here I emphasise the collective of photographers 'Nxtlvl', and the DJ's of 'Loki'. I acknowledge in particular the persons behind social gatherings in various (improvised) settings, and in the 'Ipanema', 'Zondag', and 'Landbouwbelang', for bringing together the many people that I will miss when obligations will change my life after the thesis defence.

In conclusion to these acknowledgements, I would like to thank friends from my hometown Swalmen and my brother Milan, with whom it was a pleasure to grow up. Ich bedank mien leef oma veur 't mich mitgaeve van häör wies laeveslesse die ouch op vandaag nag geljig zin; en mien eljers, veur häör ónveurwaardelikke sjteun en vertroewe. Finalmente, gostaria de expressar a minha felicidade por receber o amor da Cândida.

Sander Bakkes
March 3, 2010

Contents

Preface i

Contents iii

1 Introduction **1**

1.1 Games 1

1.2 Game AI 3

1.2.1 Goals 3

1.2.2 Game developers' approach to game AI 4

1.2.3 Academic approach to game AI 5

1.3 Problem statement and research questions 6

1.4 Research methodology 7

1.5 Thesis structure 8

2 Background **11**

2.1 Video games 11

2.1.1 History 11

2.1.2 Game genres 13

2.1.3 Game studies 14

2.1.4 State of the industry 15

2.2 Research into game AI 16

2.2.1 Entertainment and game AI 16

2.2.2 Non-adaptive game AI 17

2.2.3 Adaptive game AI 19

2.3 Chapter summary 20

3 Incremental adaptive game AI **21**

3.1 What is incremental adaptive game AI? 22

3.2 Incremental adaptive game AI with TEAM 23

3.2.1 Team AI 23

3.2.2 The TEAM adaptation mechanism 24

3.2.3 TEAM vs. Quake III CTF team AI 27

3.2.4 Discussion of TEAM 29

3.3 Incremental adaptive game AI with TEAM2 32

3.3.1 The TEAM2 adaptation mechanism 32

3.3.2 TEAM2 vs. QUAKE III CTF team AI 33

3.3.3 Discussion of TEAM2 36

3.4 Practical applicability 38

3.5 Chapter conclusions 39

4 Case-based adaptive game AI **41**
4.1 What is case-based adaptive game AI? 41
4.1.1 Case-based reasoning 42
4.1.2 Approach 42
4.1.3 Contributions and limitations of the approach 44
4.2 Case-based adaptive game AI for map adaptation 45
4.2.1 Approach to map adaptation 45
4.2.2 Experiment with map adaptation 47
4.2.3 Discussion of the results 55
4.3 Case-based adaptive game AI in a simulated game 55
4.3.1 Simulated video game 56
4.3.2 Gathering domain knowledge 57
4.3.3 Exploiting domain knowledge 59
4.3.4 Experiments with exploiting domain knowledge 62
4.3.5 Discussion of the results 63
4.4 Towards case-based adaptation in video games 64
4.5 Chapter summary 65

5 The evaluation function **67**
5.1 Evaluation functions 68
5.1.1 Evaluation functions in classic games 68
5.1.2 Evaluation functions in video games 69
5.1.3 Establishing an evaluation function 71
5.1.4 Exploiting game knowledge 73
5.2 An evaluation function for SPRING 73
5.2.1 The game environment 73
5.2.2 Towards a case base of game knowledge 76
5.2.3 Our evaluation function 77
5.2.4 Phase of the game 78
5.2.5 Material strength 78
5.2.6 Commander safety 79
5.3 Validating the evaluation function 79
5.3.1 Experimental setup 80
5.3.2 Two measures for performance assessment 82
5.3.3 Measured performance 82
5.3.4 Discussion of the results 86
5.4 Chapter conclusions 87

6 The adaptation mechanism **89**
6.1 Adaptation mechanisms 89
6.1.1 Rapid adaptation 90
6.1.2 Reliable adaptation 91
6.2 An adaptation mechanism for SPRING 93
6.2.1 General adaptation procedure 93

6.2.2 Game indexing . . . 94
6.2.3 Clustering of observations . . . 94
6.2.4 Initialisation of game AI . . . 95
6.2.5 Similarity matching . . . 95
6.2.6 Online strategy selection . . . 96
6.3 Validating the adaptation mechanism . . . 97
6.3.1 Experimental setup . . . 97
6.3.2 Performance assessment . . . 101
6.3.3 Exp1: Results of game adaptation . . . 101
6.3.4 Exp2: Results of difficulty scaling . . . 104
6.3.5 Discussion of the results . . . 105
6.4 Chapter conclusions . . . 106

7 Opponent modelling 109
7.1 Opponent modelling . . . 109
7.1.1 Opponent modelling in classic games . . . 110
7.1.2 Opponent modelling in video games . . . 111
7.2 Establishing opponent models in SPRING . . . 113
7.2.1 Approach to establishing opponent models . . . 113
7.2.2 Implementation of the approach . . . 114
7.2.3 Experiments with the implementation . . . 116
7.2.4 Discussion of the results . . . 122
7.3 Exploiting opponent models in SPRING . . . 123
7.3.1 Approach to exploiting opponent models . . . 123
7.3.2 An experiment with exploiting opponent models . . . 125
7.3.3 Discussion of the results . . . 127
7.4 Chapter conclusion . . . 127

8 Integrating the three components 129
8.1 How to integrate the three components . . . 129
8.2 Incorporating opponent modelling . . . 130
8.2.1 Establishing opponent models . . . 131
8.2.2 Exploiting opponent models . . . 132
8.3 Experiments with case-based adaptive game AI . . . 134
8.3.1 Experimental setup . . . 134
8.3.2 Performance assessment . . . 134
8.3.3 Generated opponent models . . . 135
8.3.4 Results of game adaptation . . . 135
8.3.5 Discussion of the results . . . 138
8.4 Practical applicability . . . 139
8.4.1 Scalability . . . 139
8.4.2 Dealing with imperfect information . . . 140
8.4.3 Generalisation to different games . . . 141
8.4.4 Acceptance by game developers . . . 142

8.5 Chapter conclusions 143

9 Conclusions **145**
9.1 Answers to the research questions 145
9.2 Answer to the problem statement 147
9.3 Future research 148

References **151**

Appendices **171**

A Game environments **171**
A.1 Quake III CTF 171
A.2 SPRING 172

B Features of game observations in SPRING **177**

C Evaluation function for SPRING **179**

D Parameters of strategic behaviour in SPRING **183**

Summary **185**

Samenvatting **189**

Curriculum vitae **193**

Publications **195**

SIKS dissertation series **197**

TiCC Ph.D. series **205**

Video games don't affect kids, I mean if Pac-Man affected us as kids, we'd all be running around in darkened rooms, munching pills and listening to repetitive music.

Marcus Brigstocke (b. 1973)

1 Introduction

Over the last decades, modern video games have become increasingly realistic in their visual and auditory presentation. However, artificial intelligence (AI) in games has not reached a high degree of realism yet. Game AI is typically based on non-adaptive techniques (Millington, 2006). A major disadvantage of non-adaptive game AI is that once a weakness is discovered, nothing stops the human player from exploiting the discovery. The disadvantage can be resolved by endowing game AI with adaptive behaviour, i.e., the ability to adapt to game circumstances, typically by means of learning adequate game behaviour while the game is in progress. In practice, adaptive game AI in video games is seldom implemented because currently it requires numerous trials to learn effective behaviour in online gameplay (i.e., game adaptation is not rapid). In addition, game developers are concerned that applying adaptive game AI may result in uncontrollable and unpredictable behaviour (i.e., game adaptation is not reliable). The general goal of the thesis is to investigate to what extent it is possible to establish game AI capable of adapting *rapidly* and *reliably* to game circumstances.

Chapter 1 provides the motivation of the thesis research. In the chapter we first discuss different types of game environments (Section 1.1).[1] Subsequently, we describe the current state of game AI (Section 1.2). Next, we formulate the problem statement that guides the research, along with five research questions (Section 1.3). Then, we describe the research methodology adopted in our research (Section 1.4). Finally, we outline the structure of the thesis (Section 1.5).

1.1 Games

Abt (1970) defines a game as an activity among two or more independent decision makers seeking to achieve their objectives in some limiting context. In this definition, the decision maker is the *player* of the game, and the context is the *environment* with accompanying rules in which the game is played. By extension, competition for a single player, i.e., a puzzle, may also be regarded as a game. This thesis concerns computer-game environments for two

[1] The plural "we" is used in the thesis, in recognition of the fact that research is an inherently collaborative effort.

or more players. The environments can roughly be divided into two groups, namely 'classic games', and 'video games'. Classic games, such as chess, checkers, and poker, are typically played on a game board, or with game cards. Video games, such as COMMAND & CONQUER, HALF-LIFE, and WORLD OF WARCRAFT, we define as game environments requiring the interaction with a user interface that is provided by a video device.[2]

With the advance of AI techniques, games are no longer solely played by humans, but are also played by computer-controlled players. For classic games, the goal of the computer-controlled player typically is to exhibit the most challenging behaviour. In many classic games, computer-controlled players have been able to outplay human professionals, such as the World Champion in checkers (Schaeffer *et al.*, 1996) and chess (Hsu, 2002). Recently, professional players have been outplayed in the game of Go (Lee *et al.*, 2009), which is considered one of the most complex classic games in existence. In addition, computer programs have been able to solve challenging games such as Connect Four (Allis, 1988, 1994; Allen, 1989) and checkers (Schaeffer *et al.*, 2007).

The AI techniques developed to achieve these excellent results, however, cannot directly be applied to video games, since video games differ characteristically from classic games (Spronck, 2005a). In previous work, Spronck (2005a) discusses in detail numerous differences. Below we single out the five differences that are most relevant from the perspective of AI research.

Game-theoretical classification: Game theory distinguishes between perfect and imperfect information games, as well as between deterministic and stochastic games (Koller and Pfeffer, 1997; Halck and Dahl, 1999). In perfect information games complete information of the game environment is available, while in imperfect information games part of the game environment is hidden from the player. In deterministic games no element of chance is present, while in stochastic games chance plays a prominent role. In general, classic games deal with much or even perfect information and are highly deterministic (Spronck, 2005a), while video games involve environments in which a large percentage of information is hidden, at least initially, and are highly stochastic (Buro, 2004; Chan *et al.*, 2004).

Origin of complexity: The complexity of classic games arises from the interaction of a few simple, transparent rules (Spronck, 2005a). The complexity of a video game arises from the interaction of large numbers of in-game objects and locations, controlled by complex, opaque rules (Fairclough *et al.*, 2001; Nareyek, 2002; Buro, 2004).

Pacing: Classic games usually progress at a slow pace (Nareyek, 2002), and typically require each player to respond to the game environment in a turn-based fashion. Video games, with the notable exception of turn-based strategy games, usually progress at a fast pace (Nareyek, 2002), and typically require each player to respond to the game environment in a real-time fashion.

[2] We note that, strictly speaking, video games are *computer programs* that present a game environment. To distinguish between computer programs (such as KNIGHTCAP) and classic games (such as chess), in the thesis we indicate computer programs with small capitals.

Player skills: Classic games foremost require the human player to use one's intellectual skills. Video games require the human player to employ a wide variety of skills. In addition to intellectual skills, video games may require the human player to use one's timing skills, imagination, reflexes, sensory abilities, emotions, and even ethical insights (Spronck, 2005a).

Goal: The goal of classic games is to provide a challenging play for the human player. The goal of video games is to provide an entertaining play for the human player (Tozour, 2002a; Chan *et al.*, 2004; Lidén, 2004). In this context, entertainment is a loosely defined, multifaceted notion that may refer to, for instance, engaging gameplay, appropriate challenge, and intelligent narrative.

As a result of the challenges provided by these differences, the field of AI research has extended over the past decade to encompass video games (Woodcock, 1999). So far, many challenges in the field of video game AI are not addressed to general satisfaction, even though research in classic game AI has led to fruitful results. These challenges are, among others, pathfinding, spatial and temporal reasoning, and decision making under high uncertainty (Buro and Furtak, 2003).

This thesis investigates game AI of modern video games. We note that the research has little overlap with research in classic game AI. Therefore, the term 'game' will henceforth be used to refer to 'video game'.

1.2 Game AI

Game AI is defined as the decision-making algorithms of game characters, that determine the character's behaviour (Wright and Marshall, 2000; Allen *et al.*, 2001; Fairclough *et al.*, 2001; Nareyek, 2002). Early games, such as PONG and PAC-MAN, presented relatively simple game environments where only rudimentary game AI was required. Modern games, such as ASSASSIN'S CREED and FAR CRY 2, present complex and detailed game environments that set high requirements to the game AI. Figure 1.1 gives an example of the visual realism that is presented by modern video games.

In this section, we first describe the goals that game AI aspires to for providing entertaining game AI (1.2.1). Second, we discuss the approach that game developers take to establish game AI (1.2.2). Third, we discuss the approach that academics take to establish game AI (1.2.3).

1.2.1 Goals

There are numerous goals that game AI aspires to for providing entertaining game AI. Spronck (2005a) distinguishes between seven goals. The seven goals are summarised below, arranged according to increasing difficulty.

No obvious cheating: Game AI should obtain its effectiveness without cheating obviously, for instance, without executing actions that are in principle unavailable.

Figure 1.1: Game environment of the modern video game Far Cry 2.

Unpredictable behaviour: The behaviour exhibited by the game AI should be unpredictable to the human player.

No obvious inferior behaviour: Game AI should not exhibit obvious inferior behaviour.

Using the environment: Game AI should be able to utilise and exploit intelligently features of the game environment.

Self-correction: Game AI should be able to correct its behaviour to avoid repeating mistakes.

Creativity: Game AI should be able to generate novel solutions to unforeseen game circumstances.

Human-like behaviour: The behaviour exhibited by game AI should be indistinguishable to the in-game behaviour exhibited by human players.

Of these seven goals, only the first four have been (partially) addressed by game developers. The remaining goals are foremost subject to academic research.

1.2.2 Game developers' approach to game AI

Game developers use the term 'game AI' in a relatively broad sense, to refer to techniques that control all facets of the behaviour of game characters. Typically, for them the term refers to low-level facets of game-character behaviour, such as aiming, maneuvering, and character animation. It may even refer to the generation of random numbers (Rabin, 2004a).

In modern games, the intelligent behaviour of game characters is often established on the basis of tricks and cheats (Laursen and Nielsen, 2005). For instance, in the game HALF-LIFE only two of the opponent characters were allowed to attack the player at any given time. The result was that when a non-attacking opponent character was ready to attack, an already attacking opponent character was chosen and instructed to run for cover. It was found that human players confronted with the scenario did not notice that only two opponent characters were attacking at the same time (Laursen and Nielsen, 2005). In an enhanced scenario, opponent characters running for cover would yell "Cover me!", or "Flanking!". Human players were surprisingly overwhelmed by the illusion of collaborative teamwork that was presented by the enhanced scenario (Laursen and Nielsen, 2005), and assumed intelligence where none existed (Lidén, 2004).

Tricks and cheats may indeed be instrumental in establishing the so-called illusion of intelligence (Crawford, 1984). Research has shown that players generally appreciate game AI which maintains the illusion that the AI is really intelligent (Scott, 2002). However, merely applying tricks and cheats is far from adequate for upholding the illusion of intelligence. Especially in the increasingly realistic game environments that are typically presented by modern games, an inadequacy of the game AI will be particularly apparent.

Over the last decade, game developers have increasingly attempted to compete by offering a better gameplay experience (Tozour, 2002a; Graepel *et al.*, 2004). As game AI is an essential element of gameplay, game developers increasingly emphasise the quality of their game AI. However, to establish better game AI, game developers need help from the academic community (Laird, 2000; Rabin, 2004b). This is discussed in the next subsection.

1.2.3 Academic approach to game AI

Academics use the term 'game AI' in a relatively strict sense, to refer to techniques that control the *intelligent* behaviour of game characters. Typically, the term refers to relatively complex behaviour of game characters, such as long-term planning, anticipating opponent behaviour, and reasoning over observations.

Rule-based systems are a common technique for establishing such intelligent behaviour, usually in the form of scripts (Nareyek, 2002; Tozour, 2002b). The advantage of the use of scripts is that they are user friendly, and that the behaviour resulting from its implementation can be predicted straightforwardly (Tozour, 2002b; Tomlinson, 2003). However, stemming from the complexity of modern game environments, the disadvantage of the use of scripts is that they tend to be extensive and complex (Brockington and Darrah, 2002), and prone to contain design flaws and programming mistakes (Nareyek, 2002).

A similar but more advanced approach to game AI is goal-oriented behaviour (GOB). Goal-oriented behaviour is a blanket term that covers any decision-making technique that is explicitly seeking to fulfil a game character's internal goals or desires (Millington, 2006). Typically, it is implemented as a high-level decision maker to execute predefined scripts. Though the approach can give surprisingly sensible results, it generally fails to take account of timings and the side effects that certain decisions may have (Millington, 2006).

Alternatively, intelligent behaviour can be established automatically, by incorporating machine learning techniques into the game AI. Generally, the focus is on the fifth and the sixth goal listed in Subsection 1.2.1: 'self-correction' and 'creativity'; i.e., its focus is on adaptive game AI. Adaptive game AI is game AI that is capable of adapting to changing circumstances, typically by means of learning adequate game behaviour while the game is in progress, i.e., online learning. Incorporating the capability of online learning in video games is particularly challenging, because current machine learning techniques are not ideally suited for integration in video-game environments. Generally, these techniques require numerous learning trials, make numerous mistakes before obtaining successful behaviour, and are computationally expensive (Spronck, 2005a). Therefore, game researchers often opt to design special purpose mechanisms for adaptation of game AI, which has yielded good results in previous research (Demasi and Cruz, 2002; Graepel *et al.*, 2004; Spronck, 2005a). Still, a recurring characteristic of adaptive game AI is its difficulty with establishing rapid adaptation of game AI. The reason is that current approaches to adaptive game AI require either (1) a high quality of the domain knowledge used (which generally is unavailable to the AI), or (2) a large number of trials to learn effective behaviour online (which is highly undesirable in an actual video game).

In addition, game developers and their publishers are distrustful of academic game AI (Fairclough *et al.*, 2001). Personal communication with game developers indicated that this stems predominantly from a perceived lack of reliability of academic game AI. For instance, game developers are concerned that when a player deliberately provides adaptive game AI with 'bad' learning examples, as a result it will exhibit inferior behaviour (Woodcock, 2002; Charles and Livingstone, 2004). To ensure that AI techniques developed within the academia are incorporated in the video games for which they are designed, it is vital that researchers focus on the reliability of their techniques, and preferably incorporate means to enforce their techniques to operate within strictly defined margins of reliability.

1.3 Problem statement and research questions

The contents of the three subsections of Section 1.2 lead us straightforwardly to our problem statement. Subsection 1.2.1 described seven goals that game AI aspires to for providing entertaining game AI. We stated that only the four most uncomplicated goals have been (partially) addressed by game developers. Subsection 1.2.2 indicated that to establish better game AI, game developers need help from the academic community. Subsection 1.2.3 indicated that the quality of game AI can be enhanced by incorporating the ability of self-correction and artificial creativity. These two goals are called 'adaptive game AI'. For adaptive game AI to be suitable for use in practice, in an actual, complex video game, the game requires the ability to adapt rapidly and reliably to game circumstances. Consequently, our problem statement reads as follows.

> **Problem statement:** *To what extent can adaptive game AI be created with the ability to adapt rapidly and reliably to game circumstances?*

Below, as a guideline to answer the problem statement, we formulate five research questions. A common approach to implement adaptive game AI focusses on the online learning of adequate game behaviour in an incremental fashion. We refer to this approach as 'incremental adaptive game AI'. Incremental adaptive game AI has obtained good results in previous research, and may be suitable to establish game AI with the ability of rapid and reliable adaptation. The first research question therefore reads as follows.

> **Research question 1:** *To what extent is incremental adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?*

Though improved results may be expected when using incremental adaptive game AI, the approach is not specifically designed to establish rapid and reliable adaptation of game AI. To our knowledge, an approach that specifically focusses on rapid and reliable adaptation of game AI does not exist. Therefore, we choose to design such an approach. In our research, we assume that three main components are required to establish rapid and reliable adaptation of game AI, namely (1) an evaluation function, (2) a mechanism for online adaptation of game AI, and (3) opponent modelling. The second, third, and fourth research question therefore read as follows.

> **Research question 2:** *To what extent can a suitable evaluation function for a complex video game be established?*

> **Research question 3:** *To what extent can a mechanism be employed to provide online adaptation of game AI?*

> **Research question 4:** *To what extent can models of the opponent player be established and exploited in a complex video game?*

Game developers will consider using an alternative approach to game AI when they are convinced of its qualitative effectiveness. The remainder of the thesis focusses primarily on designing and demonstrating such an alternative approach. We refer to our approach as 'case-based adaptive game AI'. Case-based adaptive game AI incorporates the three aforementioned components. In our research, we investigate case-based adaptive game AI as a *proof of concept*. A convincing illustration of case-based adaptive game AI, is to demonstrate it in the environment for which it is designed. The fifth research question therefore reads as follows.

> **Research question 5:** *To what extent is case-based adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?*

The answers to these five research questions will allow us to formulate an answer to the problem statement.

1.4 Research methodology

A research methodology is an essential element of every research project (Weigand *et al.*, 2005). Within our research we use three methodologies, namely (1) literature research, (2)

	RQ1	RQ2	RQ3	RQ4	RQ5
Literature research	×	×	×	×	
Software experiments	×	×	×	×	×
Applicability analysis	×				×

Table 1.1: Research methodologies used for answering the research questions.

software experiments, and (3) applicability analysis. The software experiments concern a (quantitative) empirical evaluation. We discuss the adopted methodology per research question (summarised in Table 1.1).

To answer research question 1, we will establish incremental adaptive game AI that is aimed at providing rapid and reliable adaptation to game circumstances. To this end, first we will perform a literature research to select a successful implementation of incremental adaptive game AI. Second, we will adapt the selected implementation for the purpose of providing rapid and reliable adaptation, and will subsequently test the implementation in software experiments performed in an actual video game. Third, we will provide an analysis of the practical applicability of incremental adaptive game AI.

We introduce case-based adaptive game AI as an alternative to incremental adaptive game AI. The three main components of case-based adaptive game AI are investigated according to research question 2, 3, and 4.

To answer research question 2, we will establish an evaluation function based on insights obtained from the literature. Subsequently, we will test the evaluation function using software experiments performed in an actual video game.

To answer research question 3, first we will perform a literature research to identify a suitable mechanism for online adaptation of game AI. Second, we will perform software experiments to validate the adopted mechanism in an actual video game.

To answer research question 4, we will perform a literature research to identify a suitable method for establishing and exploiting models of the opponent player in video games. Subsequently, we will perform software experiments to validate the adopted method in an actual video game.

To answer research question 5, first we will perform software experiments that test a complete implementation of case-based adaptive game AI in an actual video game. Second, we will provide an analysis of the practical applicability of case-based adaptive game AI.

1.5 Thesis structure

In the thesis we study rapid adaptation of video game AI. In our study, we consider that for rapid adaptation to be accepted by game developers, game adaptation needs to be reliable.

The problem statement and research questions are introduced in Chapter 1. There we also describe the adopted research methodology and the structure of the thesis. Table 1.2 summarises which research questions are addressed in the chapters of the thesis.

	RQ1	RQ2	RQ3	RQ4	RQ5	PS
Chapter 3	×					
Chapter 5		×				
Chapter 6			×			
Chapter 7				×		
Chapter 8					×	
Chapter 9	×	×	×	×	×	×

Table 1.2: Problem statement and research questions addressed by the different chapters.

In Chapter 2 we provide the background information relevant to the topic of the thesis. Subsequently, in Chapter 3, we investigate incremental adaptive game AI (in accordance with research question 1). In Chapter 4 we describe case-based adaptive game AI, and perform preliminary experiments to test the concept of the approach. Then, in Chapter 5 through Chapter 7 we discuss each component of case-based adaptive game AI in detail.

In Chapter 5 we discuss how to establish a suitable evaluation function for a video game (in accordance with research question 2). In Chapter 6 we discuss the mechanism that we employ for online adaptation of game AI (in accordance with research question 3). In Chapter 7 we discuss how to establish and exploit models of the opponent player (in accordance with research question 4). In Chapter 8 we report on experiments that integrate the three main components of case-based adaptive game AI. The experiments test a complete implementation of case-based adaptive game AI in an actual video game (in accordance with research question 5). In Chapter 9 we answer the research questions, we translate the answers to an answer to the problem statement, and finally, we give recommendations for future research.

2 Background

Game AI research started as a mathematical subject mostly focussed on search algorithms. With the rising popularity of video games, some researchers shifted their attention towards creating human-like characters in game environments. From this point on, game AI research expanded into additional disciplines, e.g., computer science, social sciences, and arts. The focus of the present research is on creating AI that can be employed successfully in the present-day, complex video games.

Chapter 2 provides background information in support of the present research. In the chapter we first discuss background information on the topic of video games (Section 2.1). Subsequently, we give an overview of previous research on the application of AI techniques in video games (Section 2.2). Finally, we provide a summary of the chapter (Section 2.3).

2.1 Video games

This section provides a concise overview of scientific research into video games. First, we will present a short history of video games (2.1.1). Second, different game genres are discussed (2.1.2). Third, three approaches to scientific studies into video games are described (2.1.3) Finally, we discuss the state of the video-game industry (2.1.4).

2.1.1 History

In 1948, the researchers Goldsmith, Grove, and Mann received a patent for their Cathode-Ray Tube Amusement Device (Goldsmith *et al.*, 1948). The proposed device would be a missile simulator which was inspired by radar displays from World War II. The device would use eight vacuum tubes to simulate a missile firing at a target, and would use knobs to adjust the curve and speed of the missile. This became known as the very first concept for a video game.

The first game to use a graphical display in practice, was the game TENNIS FOR TWO.[1] This video game was created in the year 1958, and was a predecessor of PONG, one of the most widely recognized video games as well as one of the first.

Game aficionados had to wait till the year 1962, when the first video game became commercially available in the form of SPACEWAR!. The game SPACEWAR!, illustrated in Figure 2.1, was developed at MIT, which had in its possession the then state-of-the-art DEC PDP-1 computer. Steve Russell, a fellow at MIT's Hingham Institute, decided that he had to do something exciting with it (Baratz, 2001). Brand (1974) quotes Russell who said:

> *"It had... a cathode-ray tube display... Here was this display that could do all sorts of good things! So we started talking about it, figuring what would be interesting displays. We decided that probably you could make a two-dimensional maneuvering sort of thing, and decided that naturally the obvious thing to do was spaceships."*

The game created by Russell inspired the engineers Bushnell and Dabney to build 1,500 arcade machines on which a clone of SPACEWAR! could be played. Despite the project being a commercial failure, the engineers persevered in their goal to bring video games to a mainstream audience, and in the year 1972 they established the video-game company Atari. Though the first video-game console had already been released by Magnavox, it was Atari who, by pioneering in arcade machines, home video-game consoles, and home computers, helped define the video-game industry from the 1970s to the mid 1980s.

As the capabilities of computers in terms of processing power advanced, so did the complexity of video games. In the 1980s a team consisting of only a couple of people could create a top-rated game (Spronck, 2005a). Currently, game-development teams consist of hundreds of people, with development costs of a top-rated game of around twenty million dollars (Richtel, 2008), and in rare cases as high as one-hundred million dollars (Rogers, 2005).[2] For a long time the processing power of computers was mainly invested into creating better graphics (Spronck, 2005a). When in the late 1990s specialised 3D video cards became available and widespread, its use freed up processing power for other gameplay features, such as game AI (Tozour, 2002a).

Throughout the years, video games have become so realistic that computer-controlled characters in these games are expected to behave realistically (Laird and Van Lent, 2001). Still, game AI has not yet reached a high degree of realism (Fenley, 2002; Millington, 2006). In the last few years, however, the developers of video games have increasingly emphasised the importance of high-quality game AI, and incorporated game-AI programming as an important activity in game development (Spronck, 2005a).

[1] We note that before the release of the first video game, computers were already able to represent and play several *classic* games. A notable example is the 1952 TIC-TAC-TOE computer program that was created by A.S. Douglas as part of his Ph.D. research (Douglas, 1954).

[2] In the thesis, the term 'dollar' is used as a reference to the official currency of the United States of America.

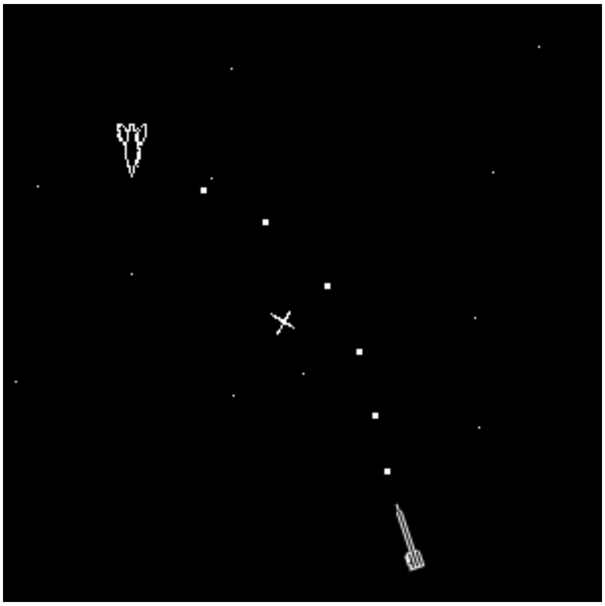


Figure 2.1: SPACEWAR!, one of the earliest known video games.

2.1.2 Game genres

Video games can be categorised into different genres. Due to a general lack of genres commonly agreed-upon or criteria for the definition of genres, the classification of games is not always consistent or systematic, and sometimes is seemingly arbitrary between sources. For instance, Fairclough *et al.* (2001) distinguish 'action games', 'adventure games', 'role-playing games', and 'strategy games'. To these four genres, Schaeffer (2001) adds 'god games' and 'sports games', and Spronck (2005a) adds 'simulation games' and 'puzzle games'.

Though by no means an exhaustive list can be established, my view is that by addition of 'simulation games' to the selection by Fairclough *et al.* (2001), the vast majority of games can be categorised. This leads to the categorisation of games into the following five genres: action games, adventure games, role-playing games, simulation games, and strategy games. We discuss the different genres below.

Action: Action games are games that foremost challenge players' physical reaction speed and precision. The four main types of action games are *fighting games* (such as STREET FIGHTER), *platform games* (such as SUPER MARIO BROS.), *shooter games* (such as DUCK HUNT), and *sports games* (such as FIFA SOCCER).

Adventure: Adventure games focus on narrative challenges, and typically require players to interact socially with game characters, explore the game world, and solve problems.

The two main types of adventure games are *text-based adventures* or *interactive fiction* (such as ZORK), and *graphical adventures* (such as MONKEY ISLAND).

Role-playing: Computer role-playing games (CRPGs) are derived from traditional, i.e., tabletop, role-playing games, especially DUNGEONS & DRAGONS. Role-playing games focus on narrative challenges, and typically require the player to assume the role of a game character which evolves over the course of the game. Gameplay in this genre is focussed on solving challenges, called 'quests'. The two main types of computer role-playing games are *single-player role-playing games* (such as BALDUR'S GATE), and *massively multiplayer online role-playing games* (MMORPGs) (such as WORLD OF WARCRAFT).

Simulation: Simulation games intend to simulate parts of the real world, or of a fictional world. The objective of the game depends on the world being simulated, and the position taken by players within this world. The two main types of simulation games are *god games* (such as SIMCITY), and *vehicle simulations* (such as FLIGHT SIMULATOR).

Strategy: Strategy games primarily focus on players' decision-making capabilities, and typically are played in an environment where chance does not play a prominent role in the outcome of the game. The three main types of strategy games are *puzzle games* (such as LEMMINGS), *turn-based strategy games* (such as CIVILIZATION), and *real-time strategy games* (such as COMMAND & CONQUER).

In the thesis, we focus on real-time strategy games, as well as on action games (particularly, first-person shooter games). One should note that games may be developed as so-called hybrids, combining elements that are characteristic to one or more game genres. An example of a hybrid game is THE LEGEND OF ZELDA, which has elements of action, adventure, and role-playing. In fact many games are developed as a combination of several genres (Spronck, 2005a), a phenomenon that can be explained by the attempt of game developers to create an original game that exhibits the best of different genres (Slater, 2002).

2.1.3 Game studies

Throughout the years, video games have gained significant academic interest. Education programmes have been established that particularly focus on game design and development, and academic disciplines have emerged that focus specifically on video games. Studies into video games can be divided into three approaches: (1) the social-science approach, (2) the humanities approach, and (3) the computer-science approach.

The social-science approach refers to the study on the effect of video games on people. Some studies focus on behaviour that may be evoked by video games (e.g., gaming addiction and violent behaviour). Other studies focus on behaviour observed within a video-game environment. For instance, economic aspects of human-decision making are being studied in massively multiplayer online games (MMOGs) (Castronova, 2001). Video games are seen as a suitable research platform, where (1) a high number of participants can be included, and (2) tightly controlled experimental conditions can be achieved (Castronova, 2005).

The humanities approach refers to investigating the various roles that video games play in people's lives and activities, together with the meaning which people assign to their experiences (Consalvo and Dutton, 2006). The two main approaches within this field of research are (1) narratology, which models video games as a storytelling medium, one that arises out of interactive fiction (Murray, 1998), and (2) ludology, which models video games first and foremost as just that; games that needs to be understood in terms of its rules, interface, and in terms of the concept of play (Juul, 2001).

The computer-science approach refers to the use of video games as a platform for technological progress. As games often simulate aspects of reality (Spronck, 2005a), the simulation of these aspects drives innovation in fields such as graphics, human-computer interaction, networking, and AI. The video-game industry has discovered research into AI as a necessary ingredient to make games more entertaining and challenging, and, vice versa, AI in video games serves as a challenging application domain for researchers (Laird and Van Lent, 2001). As AI in video games must be able to simulate human-like behaviour to a large extent, video games are ideally suited to understand and develop AI systems with human-like capabilities (Laird and Van Lent, 2001; Sawyer, 2002).

In the thesis, we focus on the computer-science approach to game studies.

2.1.4 State of the industry

In the early 2000s, the video-game industry has grown to surpass the Hollywood movie industry in revenues (Fairclough *et al.*, 2001; Snider, 2002), and in recent years, video gaming has become a mass-market entertainment industry on a par with TV, movies, and music (Wolf, 2006). To illustrate the popularity of video games, Velasco (2007) notes that after the release of a top-rated game, such as HALO 3, movie theaters will suffer from smaller attendance. As a result, the movie industry currently plans the premiere of a movie in close consideration of the release dates of video games.

The video-game industry is expected to double in sales from 2005's $32.6 billion to $65.9 billion in the year 2011 (Wolf, 2006). Traditional single-player games are expected to remain high sellers, but the strongest growth is expected in online gaming, mobile gaming, serious gaming, and in-game advertising. We discuss these growing market segments below.

Online gaming: Online gaming refers to playing a game over a computer network. In the early years of online gaming, a game would be played over a modem or over local-area networks (LANs). Currently, popular games such as WORLD OF WARCRAFT connect millions of players over the internet, typically at the charge of a monthly fee. Wolf (2006) expects the online-gaming segment to grow by 95% each year until it is the dominating force in the video-game market in 2011.

Mobile gaming: Mobile gaming refers to playing a game on a wireless device, such as a mobile phone or a handheld computer. Currently, the average play time of mobile games is 11 minutes (Agarwal, 2007). The most successful mobile games therefore are simple and easy games, such as TETRIS and PAC-MAN (Furini, 2007). Driven by advancement in technology (such as location technologies), the mobile-gaming segment is expected to grow to $9.6 billion by 2011 (Nguyen *et al.*, 2007).

Serious gaming: Serious gaming is a term coined in the year 2002, that refers to the exploitation of video games and video-game software tools for a purpose other than pure entertainment (Stone, 2005). Serious games are intended to provide an engaging, self-reinforcing context in which to motivate and educate the players (Ferguson, 2007), for instance for the training of medical professionals. Sources estimated the market-segment to have grown to $2 billion in the year 2008, with a current trend of growth of around twenty per cent on a yearly basis (Alhadeff, 2007).

In-game advertising: In-game advertising refers to the use of video games as a medium in which to deliver advertising. Advertisers see in-game advertising as a prime way to target a broad audience, which is increasingly neglecting television in favour of video games (Yi, 2005). Sources estimate in-game advertising to become a market segment worth close to $3 billion by the year 2011 (Wolf, 2006). Some researchers expect that the ability to play music and media from powerful consoles and handhelds will drive overall industry growth, as consumers will begin to view gaming devices as "one-stop-shop entertainment platforms" (Wolf, 2006).

In the thesis, we focus foremost on online gaming. Currently, relatively little AI is applied in video games. Experts note, however, that advances in game AI would fundamentally change the way that games are designed, and allow the creation of entirely new types of games. Molyneux (2006) states that "advances will also allow players to have entirely unique experiences as each time you play a given scenario it will evolve differently, and will allow far richer, more realistic worlds to be created as more and more elements react more believably".[3]

2.2 Research into game AI

In this section we provide an overview of previous research on the application of AI techniques in video games. First, we discuss how game AI can contribute to the entertainment value of a game (2.2.1). Subsequently, we discuss the AI that is typically applied in present-day video games: non-adaptive game AI (2.2.2). Finally, we describe previous research on what both researchers and video-game developers often aim for: adaptive game AI (2.2.3).

2.2.1 Entertainment and game AI

The purpose of a typical video game is to provide entertainment (Tozour, 2002a; Nareyek, 2004). Naturally, the criteria of what makes a game entertaining may depend on who is playing the game. The literature suggests the concept of *immersion* as a general measure of entertainment (Manovich, 2002; Taylor, 2002).

Immersion is the state of consciousness where an immersant's awareness of physical self is diminished or lost by being surrounded in an engrossing, often artificial environment

[3] Molyneux, regarded as one of the world's most brilliant and inventive game developers, has nevertheless acquired a reputation for issuing over-enthusiastic descriptions of games under development, which are found to be somewhat less ambitious when released.

(Nechvatal, 1999). Taylor (2002) argues that evoking an immersed feeling with a video game is essential for retaining a player's interest in the game. As such, an entertaining game should at least not repel the feeling of immersion from the player (Laursen and Nielsen, 2005). Aesthetical elements of a video game, such as graphical and auditory presentation, are instrumental in establishing an immersive game environment. Once established, the game environment needs to uphold some form of consistency for the player to remain immersed within it (Laursen and Nielsen, 2005). A lack of consistency in a game results in player immersion breakdowns (Taylor, 2002).

To this end, the task for game AI is to control game characters in such a way, that behaviour exhibited by the characters is *consistent* within the game environment. In a realistic game environment, realistic character behaviour is expected. As a result, game AI that is focussed solely on exhibiting the most challenging behaviour, is not regarded as realistic. For instance, in a first-person shooter (FPS) game it would not be realistic if characters controlled by game AI would aim with an accuracy of one hundred per cent. Game AI for shooter games, in practice, is designed to make intentional mistakes (Lidén, 2004), such as warning the player of an opponent character's whereabouts by intentionally missing the first shot.

For game characters to be consistent with the game environment that they are situated in, their behaviour is often established on the basis of tricks and cheats. While it is true that tricks and cheats may be required to uphold the consistency of the game environment, they often are implemented only to compensate for the lack of sophistication in game AI (Buro and Furtak, 2004). Despite the use of tricks and cheats, game AI in most complex games still is not consistent with the game environment, and exhibits what has been called 'artificial stupidity' (Schaeffer, 2001; Lidén, 2004) rather than artificial intelligence. To increase game consistency, and thus the entertainment value of a video game, we agree with Buro and Furtak (2004) that researchers should foremost strive to create an optimally playing game AI. In complex video games, such as real-time strategy (RTS) games, near-optimal game AI is seen as the only way to obtain consistency of the game environment (Laursen and Nielsen, 2005). Once near-optimal game AI is established, difficulty-scaling techniques can be applied to downgrade the playing-strength of game AI, to ensure that a suitable challenge is created for the player (Spronck, 2005a).

2.2.2 Non-adaptive game AI

Many researchers argue that the goal of game AI is to make human players believe that opponent characters are actually controlled by other humans (Laird and Van Lent, 2001; Sawyer, 2002; Livingstone and McGlinchey, 2004). To achieve this goal, game developers typically opt for the use of proven AI techniques. The three most commonly used AI techniques in video games are (1) decision trees, (2) finite state machines, and (3) scripting. We give a concise description of these AI techniques below.

Decision trees: A decision tree is a model of a decision-making process. In video-game practice, a decision tree is typically implemented as a series of if-then statements, that, when applied to a game observation, results in the classification of that observation. Commonly, the resulting classification is the decision to be made. Decision trees are fast, easily implemented, and simple to understand (Millington, 2006).

Finite state machines: A finite state machine (commonly abbreviated as FSM) is a model of behaviour that is composed of (1) a finite number of behavioural states, (2) transitions between those states, and (3) conditions for state transition. For example, a simple game AI may consist of the two states 'fight' and 'flee', and a parametric condition that defines when to transit between the states. Finite state machines in present-day video games can become fairly large, which makes them difficult to maintain (Millington, 2006).

Scripting: A script in a video-game environment is a sequence of expected behaviours for a given situation. Scripts are often separated from the game engine and are written in relatively well understandable scripting languages, such as Lua. This enables non-programmers to implement the behaviour of game AI. However, as a consequence of game complexity, scripts tend to be quite long and complex (Brockington and Darrah, 2002). Nareyek (2002) notes that manually developed complex scripts are likely to contain design flaws and programming mistakes.

The use of so-called non-adaptive AI techniques, such as the ones mentioned above, may suffice to establish game AI for simple games. However, non-adaptive AI techniques, as the name implies, do not allow the game AI to adapt to changing circumstances. Therefore, non-adaptive game AI is grossly inadequate to deal with the challenges presented in complex video games, where game AI that is consistent with the game environment is required. With complex video games in mind, such as real-time strategy (RTS) games, Buro (2004) states four reasons as to why the current AI performance in complex games is lagging behind the developments in related areas such as classic board games. These four reasons are summarised below.

1. AI research for video games receives relatively little attention by game developers.
2. Typical video games feature complex environments, imperfect information, and a need for rapid decision making. Present-day AI techniques are not suitable for complex video games.
3. For commercial success, not all video games require good AI.
4. It is difficult to set up an infrastructure for conducting AI experiments in complex video games.[4]

In the near future, the complexity of video games will only increase (Spronck, 2005a). Many researchers argue that the more complex a video game is, the greater the need for game AI that has the ability to adapt to changing circumstances (Fairclough *et al.*, 2001; Laird and Van Lent, 2001; Fyfe, 2004). It is therefore safe to say that the necessity for game AI with such an ability will increase accordingly.

[4] Buro (2004) notes that partly because video games typically are closed-source, there is a lack of AI competition in the field of video games, whereas in the field of classic games, the element of competition is one of the driving forces of AI research. In the year 2006, Buro, Aha, Sturtevant, and Corruble established an RTS game-AI competition that now takes place on a yearly basis, prior to the AIIDE conference.

2.2.3 Adaptive game AI

As present-day video games present a complex and realistic environment, one would expect characters controlled by game AI in such an environment to behave realistically ('human-like'). One feature of human-like behaviour of game AI, namely the ability to adapt to changing circumstances, has been explored with some success in previous research (Demasi and Cruz, 2002; Graepel *et al.*, 2004; Spronck *et al.*, 2004b). This ability is called 'adaptive game AI.'

When implemented successfully, adaptive game AI is able to fix errors in programmed game AI, and to seek counter-tactics to human gameplay. Research performed by Spronck (2005a) indicated that machine learning techniques may be used to establish adaptive AI in complex video games. There are two different approaches in which machine learning may be applied to establish adaptive game AI, namely as (1) offline learning, and as (2) online learning. We discuss the different approaches below.

Offline learning: Offline learning of game AI is learning that takes place while the game is not being played by a human (Charles and McGlinchey, 2004; Funge, 2004). Learning takes place on the basis of game observations of the game AI (1) in competition against a human, (2) in competition against another game AI, or (3) in competition against itself (i.e., self-play). Offline learning can typically be applied in the 'quality assurance' phase of game development, for instance, to tune game parameters automatically, and to create new AI behaviour automatically (Spronck, 2005a).

Online learning: Online learning of game AI is learning that takes place while the game is being played by a human (Charles and McGlinchey, 2004; Funge, 2004). Learning takes place on the basis of the behaviour exhibited by the human player. Online learning can typically be applied to adapt game AI in accordance with the human player's experience level, playing style, strategy, or tactics. Currently, only a few games have been released that incorporated a capability of online learning (e.g., THE FALL OF MAX PAYNE and NASCAR RACING 2003 SEASON). This is attributed to the concern of game publishers that game AI may learn inferior behaviour (Woodcock, 2002; Charles and Livingstone, 2004). Spronck (2005a) expects, however, that in the near future online-learning will be a standard element of games.

There are two different approaches in which a machine learning technique for game AI can be controlled, namely (1) human-controlled learning, and (2) computer-controlled learning.[5] We discuss the different approaches below.

Human-controlled learning: Human-controlled learning implements adaptations to the game AI by processing immediate feedback on specific decisions that the game AI makes. Typically, the feedback indicates whether a decision is desired or undesired

[5] We note that these are concepts different from 'supervised learning' and 'unsupervised learning'. Learning is supervised when both the input signals and outputs signals can be perceived. Learning is unsupervised when the correct output signals are unknown. In video games, human-controlled learning is typically implemented in a supervised fashion, and computer-controlled learning is typically implemented in either a supervised or an unsupervised fashion.

by the player. By implication, the human player controls what is being learned. The games Black & White and Creatures are two well-known examples of games that incorporate human-controlled learning.

Computer-controlled learning: Computer-controlled learning implements adaptations to the game AI without requiring human intervention. Advanced computer-controlled learning has not yet been incorporated in an actual video game. However, relatively simple implementations of computer-controlled learning can be surprisingly effective (Funge, 2004). For instance, in the game The Fall of Max Payne, the difficulty of the game AI was adapted automatically, dependent on behaviour exhibited by the human player.

In the thesis, we focus on online, computer-controlled learning. As human beings are able to deduce successful behaviour effectively from observations (Ross, 1989), our aim is to create adaptive game AI with a similar capability. Current approaches to adaptive game AI, however, insufficiently integrate machine learning techniques for reaching this aim in video-game environments. Generally, these techniques require numerous learning trials, make numerous mistakes before obtaining successful behaviour, or are computationally expensive (Spronck, 2005a). Therefore, we will investigate a novel approach to adaptive game AI.

2.3 Chapter summary

This chapter provided background information on the research in this thesis. It introduced the reader to the topic of video games, and gave an overview of previous research on the application of AI techniques in video games. It discussed the need for *adaptive* game AI, and distinguished two different ways in which AI techniques can be applied to establish adaptive game AI, namely by (1) offline learning, and by (2) online learning. Additionally, it distinguised two different ways in which adaptive game AI can be controlled, namely by (1) human-controlled learning, and by (2) computer-controlled learning. The focus of this thesis is on online, computer-controlled learning that is capable of adapting *rapidly* and *reliably* game AI.

3 Incremental adaptive game AI

In recent years, video-game developers have emphasised the importance of high-quality game AI. When implementing game AI, even state-of-the-art games resort mainly to long-time proven rule-based techniques, such as expert systems and finite state machines (Spronck, 2005a). These techniques are prone to introducing errors in the game AI (Brockington and Darrah, 2002), and lack the capability of adaptive behaviour (Manslow, 2002). Game AI with the capability of adaptive behaviour is, if present, predominantly implemented via a so-called incremental approach.

Chapter 3 focusses on incremental adaptive game AI. In the chapter we first define incremental adaptive game AI (Section 3.1). Subsequently, we experiment with a mechanism that implements incremental adaptive game AI (Section 3.2). Then, we experiment with an extension of the mechanism, in which we incorporate an increased focus on exploiting recent game observations (Section 3.3). Next, we give an analysis of the practical applicability of incremental adaptive game AI (Section 3.4). Finally, we present the chapter conclusions (Section 3.5).

This chapter is based on the following four publications.

1) Bakkes, S. C. J., Spronck, P. H. M., and Postma, E. O. (2004). TEAM: The Team-oriented Evolutionary Adaptability Mechanism. In Rauterberg, M., editor, *Entertainment Computing - ICEC 2004*, volume 3166 of *Lecture Notes in Computer Science*, pages 273–282. Springer-Verlag, Heidelberg, Germany.

2) Bakkes, S. C. J., Spronck, P. H. M., and Postma, E. O. (2005a). Best-response learning of team behaviour in Quake III. In Aha, D. W., Muñoz-Avila, H., and Van Lent, M., editors, *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 13–18. Navy Center for Applied Research in Artificial Intelligence, Washington, DC., USA.

3) Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2005b). Learning to play as a team. In Wanders, A., editor, *Proceedings of the Learning Solutions 2005 symposium*, pages 16–17. SNN Adaptive Intelligence, Nijmegen, The Netherlands.

4) Bakkes, S. C. J. and Spronck, P. H. M. (2005). Symbiotic learning in commercial computer games. In Mehdi, Q. H., Gough, N. E., and Natkin, S., editors, *Proceedings of the 7th International Conference on Computer Games (CGAMES 2005)*, pages 116–120. University of Wolverhampton, Wolverhampton, UK.

3.1 What is incremental adaptive game AI?

We define incremental adaptive game AI as an approach to game AI where the adaptive capability is provided by incremental techniques. The approach, illustrated in Figure 3.1, implements a direct feedback loop for control of characters that are operating in a game environment. The behaviour of a game character is determined by the game AI. Each game character feeds the game AI with data on its current situation, and with the observed results of its actions. The game AI adapts by processing the observed results, and generates actions in response to the character's current situation. An adaptation mechanism is incorporated to determine what the best way is to adapt the game AI. For instance, reinforcement learning may be applied to assign rewards and penalties to certain behaviour determined by the game AI.

Typically, incremental adaptive game AI will be implemented for performing adaptation of the game AI in an *online* and *computer-controlled* fashion. Improved behaviour is established by continuously making (small) adaptations to the game AI. Incremental adaptive game AI in such an implementation can be characterised by its scheme for incremental adaptation. This scheme can be summarised as follows.

Incremental adaptation: To adapt to circumstances in the current game, the adaptation process is based only on observations of the *current* game. Naturally, a bias in the adaptation process may be included based on domain knowledge of the experimenter. In addition, to ensure that AI is effective from the onset of a game, it is common practice to initialise adaptive game AI with long-time proven behaviour.

Online adaptation via an incremental approach may be used to improve significantly the quality of game AI by endowing it with the capability of adapting its behaviour while the game is in progress. The approach has been applied successfully to simple video games (Demasi and Cruz, 2002; Johnson, 2004) and to complex video games (Spronck *et al.*, 2006). However, to our knowledge advanced online adaptation has never been used in a commercially released video game.

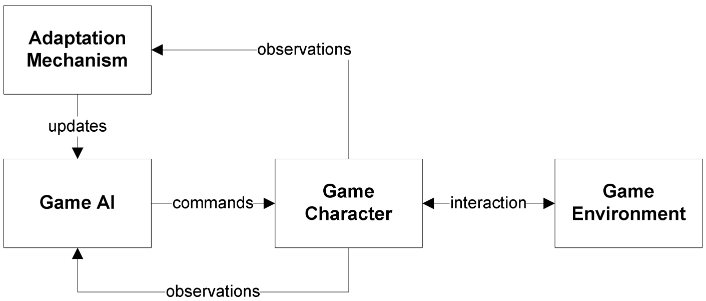


Figure 3.1: Incremental adaptive game AI (see text for details).

3.2 Incremental adaptive game AI with TEAM

In this section we report on an experiment to investigate how incremental adaptive game AI can be used effectively in an actual video game. We present an approach to incremental adaptive game AI that we named TEAM. TEAM stands for Tactics Evolutionary Adaptability Mechanism. It is a mechanism to establish adaptive AI for team behaviour.

The remainder of the section is organised as follows. We first discuss team AI (3.2.1). Second, we present the TEAM mechanism (3.2.2). Third, we discuss an experiment that tests TEAM in an actual video game (3.2.3). Finally, we end the section by a discussion of the experimental results (3.2.4).

3.2.1 Team AI

We define team AI as the behaviour of a team of computer-controlled opponents that competes with other teams within a video-game environment. We note that in the domain of video games, a computer-controlled opponent is often referred to as a 'non-player character' (NPC).

Team AI consists of three components: (1) the AI of individual NPCs, (2) a means of communication, and (3) the organisation of the team. We concisely discuss each of these three components below.

AI of individual NPCs: The AI of individual NPCs is, as the name implies, the AI of an NPC which controls the NPC's behaviour. Typically, the AI of an NPC controls relatively low-level AI behaviour, such as pathfinding and aiming. The AI of individual NPCs is game-specific.

Communication: Coordinated behaviour in a team requires communication. In typical implementations of team AI, NPCs pass along messages containing information or commands. The information can be used to compute tactics and distribute commands amongst the team members. An implementation of the communication component can often be generalised to other games.

Organisation: A form of internal organisation is required to establish team cohesion. Two distinctive approaches to organising a team are: (1) a decentralised approach, and (2) a centralised approach. An implementation of the organisation component can often be generalised to other games.

The decentralised approach is an extension of the AI of individual NPCs. In the decentralised approach, NPCs operate in a non-hierarchical communication structure. Team behaviour emerges through the combined interaction of all NPCs. Figure 3.2 (left) is an illustration of a decentralised team of NPCs.

The centralised approach, schematically displayed in Figure 3.2 (right), is strictly hierarchical and is specifically designed to create and maintain well-organised team cohesion. In this approach, the process of decision-making is centralised. Typically, a centralised decision-making mechanism (1) observes NPCs, (2) makes a decision, and

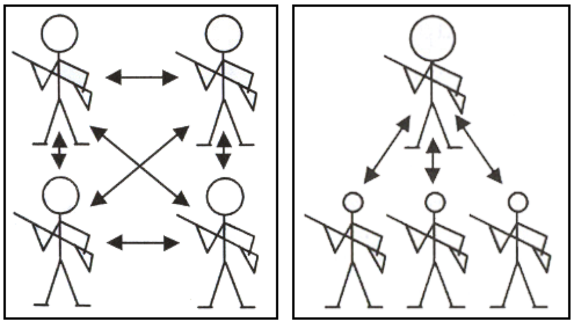


Figure 3.2: Decentralised organisation (left) and centralised organisation (right) (Van der Sterren, 2002).

(3) processes the decision into NPC commands. We note that it resembles the OODA loop for modelling military decision making (Boyd, 1987). The implementation of the centralised mechanism varies in style from authoritarian (which focusses on team performance by forcing NPCs to perform commands), to coaching (which advises, rather than forces NPCs).

3.2.2 The TEAM adaptation mechanism

To establish *adaptive* team AI, an adaptation mechanism is needed as an additional component. An adaptation mechanism for team AI in video games does not yet exist. We decided to design and implement an adaptation mechanism for team AI. Game AI of the popular video game Quake III Capture The Flag (CTF) (Van Waveren and Rothkrantz, 2001), is used for illustrative purposes. Quake III CTF is a first-person shooter action game. An extensive description of Quake III CTF is provided in Appendix A.1. The game environment is illustrated in Figure 3.3. The proposed adaptation mechanism is discussed next.

The Tactics Evolutionary Adaptability Mechanism (TEAM) is an evolutionary inspired adaptation mechanism. We first discuss (A) the concept of TEAM. Subsequently, we discuss four notable features of TEAM. These four features are: (B) a centralised NPC control mechanism evolution, (C) a symbiotic evolution, (D) a delayed FSM-based evaluation, and (E) an evolution with history fall-back. We assume homogeneous NPCs.

A. Concept of TEAM

TEAM is designed to be a generic adaptation mechanism for team-oriented games in which the game state can be represented as a finite state machine (FSM). An instance of TEAM is created for each state of the FSM. Each instance is an evolutionary algorithm which learns state-specific behaviour for *a team as a whole.* For our experiment in Quake III CTF, the evolutionary algorithm was designed to learn optimal team behaviour for each state of the FSM. Particular team behaviour is expressed by a limited set of parameter values.

Figure 3.3: Screenshot of the Quake III CTF game environment. In the screenshot, two NPCs are engaged in combat.

Cooperatively, all instances of TEAM learn successful team-oriented behaviour for *all states* of a game. We consider the game Quake III CTF to have four states (see Appendix A.1). The concept of TEAM is illustrated in Figure 3.4.

B. Centralised NPC control mechanism evolution

TEAM evolves the NPC control mechanism in a centralised fashion. The choice for centralised control is motivated by the desire to evolve directly the behaviour for a team as a whole. The performance of a team's behaviour is assessed by performing a high-level evaluation of the whole team.

C. Symbiotic evolution

The evolutionary approach of TEAM is inspired by symbiotic animal communities (Raven and Johnson, 2001). In such communities, individuals postpone short-term individual goals in favour of long-term community goals.

The focus of TEAM's evolutionary mechanism lies on learning team-oriented behaviour by the cooperation of multiple evolutionary algorithms. For each state of the FSM that controls the team-oriented behaviour, a separate evolutionary algorithm is used. Each of these evolutionary algorithms learns relatively uncomplicated team-oriented behaviour for the specific state only. Still, the behaviour is learned in consideration of the long-term effects on the other evolutionary algorithms. As a result, relatively complex team-oriented behaviour emerges in a computationally fast fashion.

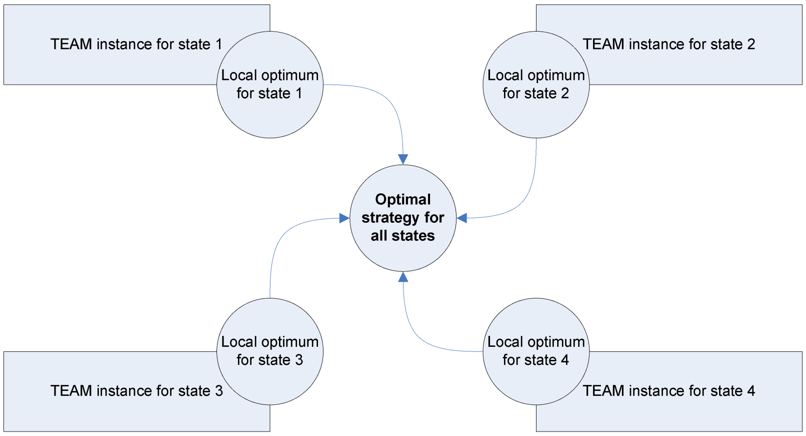

TEAM instance for state 1
Local optimum for state 1
TEAM instance for state 2
Local optimum for state 2
Optimal strategy for all states
Local optimum for state 3
TEAM instance for state 3
Local optimum for state 4
TEAM instance for state 4

Figure 3.4: Conceptually, TEAM learns adaptive behaviour for a team as a whole (rather than learning adaptive behaviour for each individual). Instances of TEAM cooperatively learn team-oriented behaviour, which is defined as the combination of the local optima for the states (in this example there are four states).

D. Delayed FSM-based evaluation

For evaluating the behaviour that is learned by the evolutionary algorithms, we defined a delayed FSM-based evaluation function. The function postpones the determination of the fitness value of a genome until a predetermined number of state transitions (the so-called "depth") have been processed after employing the particular genome. The delayed FSM-based evaluation function consists of two components: (1) a scaled fitness function Fs_i, and (2) a delayed fitness function Fd.

The scaled fitness function is defined as:

$$Fs_i = \begin{cases} 1.0 - \min\left(\frac{\sqrt{t_i} - \sqrt{t_i/3}}{10}, 1.0\right) & \{\textit{desirable transition}\} \\ \min\left(\frac{\sqrt{t_i} - \sqrt{t_i/3}}{10}, 1.0\right) & \{\textit{undesirable transition}\} \end{cases} \tag{3.1}$$

where t_i denotes the time elapsed between state transitions at depth i, and where the type of transition is calculated by using annotations on the FSM which describes the states of a game. Preliminary experiments and knowledge of the game led to this function. The square roots in the function are used to the effect of emphasising relatively short state transitions, and clearing relatively long state transitions. In Figure 3.5, the used annotations on the FSM of Quake III CTF is given.

The delayed fitness function for a state transition is defined as:

$$Fd = \sum_{i=0}^{n} \frac{1}{i+1} (Fs_i) \tag{3.2}$$

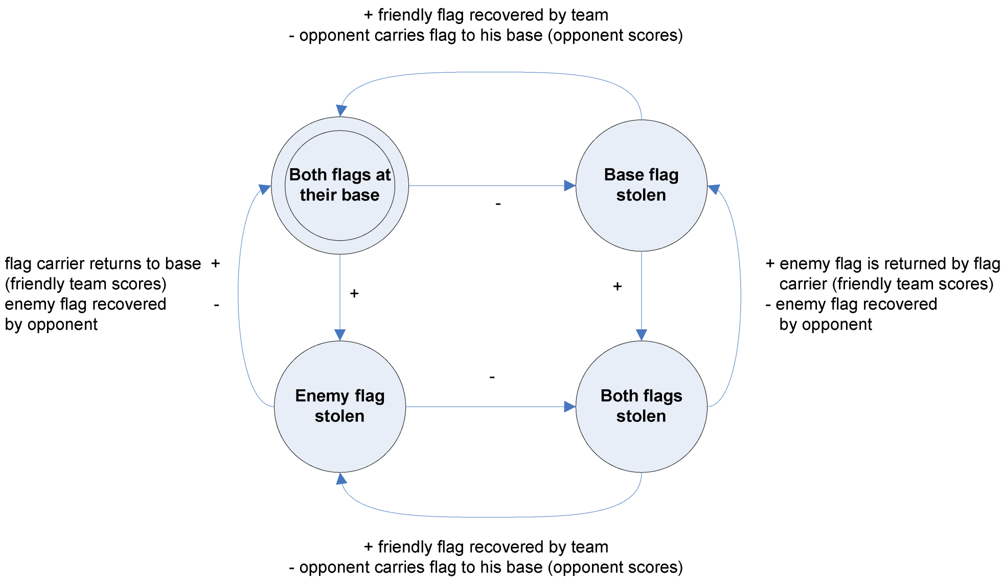

+ friendly flag recovered by team
- opponent carries flag to his base (opponent scores)
Both flags at their base
Base flag stolen
Enemy flag stolen
Both flags stolen
flag carrier returns to base (friendly team scores) +
enemy flag recovered by opponent -
+ enemy flag is returned by flag carrier (friendly team scores)
- enemy flag recovered by opponent
+ friendly flag recovered by team
- opponent carries flag to his base (opponent scores)

Figure 3.5: Annotated finite state machine of QUAKE III CTF. Desirable state transitions are denoted by "$+$", and undesirable state transitions are denoted by "$-$".

where i is the depth, n is a positive integer, and Fs_i is the scaled fitness value of a genome at depth i. The delayed reward is used to consider the long-term effects of genomes, because seemingly desirable behaviour is only desirable if the team can either retain or improve on the behaviour. In our experiment a delayed reward of $n = 2$ is used.

E. Evolution with history fall-back

The game environment of team-oriented games is typically accompanied by a large amount of randomness. The randomness poses a challenge for most adaptation mechanisms since one cannot be sure that a successful course of the evolution is the direct result of (1) a high quality of genetic information in the population, or of (2) lucky circumstances. Consequently, we designed the evolutionary mechanism of TEAM so that it is capable of reverting to an earlier population. This so-called history fall-back is implemented by continuously recalculating fitness values, in order to filter out unsuccessful genomes in due time.

3.2.3 TEAM vs. Quake III CTF team AI

To assess the performance of the TEAM mechanism, we incorporated it in the QUAKE III CTF game. We performed an experiment in which an adaptive team (controlled by TEAM) is pitted against a non-adaptive team (controlled by the original team AI of QUAKE III CTF). In the experiment, the TEAM mechanism adapts the tactical behaviour of a team to the opponent. A tactic consists of a small number of parameters which represent the offensive and defensive division of roles of NPCs that operate in the game. The inherent randomness in the QUAKE III CTF environment requires the adaptation mechanism to be able to adapt successfully to significant behavioural changes of the opponent.

In the remainder of this subsection, we subsequently give (A) the experimental setup, (B) the performance assessment, (C) the obtained results, and (D) an analysis of the results.

A. Experimental setup

In our experiment, an experimental run consists of two teams playing QUAKE III CTF until the game is interrupted by the experimenter. One team is controlled by TEAM, the other team is controlled by the QUAKE III CTF team AI, which offers non-static and fine-tuned, but non-adaptive behaviour. On average, the game is interrupted after three hours of gameplay. Here we remark that typical QUAKE III CTF games require considerably less time to finish. We perform fifteen experimental runs with the TEAM mechanism.

Both competing teams consist of four NPCs with identical individual AI, identical means of communication, and an identical organisation of the team. The teams only differ in the control mechanism employed (adaptive or non-adaptive). We note that the QUAKE III CTF team AI switches intelligently its behaviour; the moment of switching depends on whether the AI is winning or losing the game. Thus, adaptive team AI in competition against this opponent has to be able to deal with significant behavioural changes.

B. Performance assessment

To quantify the performance of TEAM in competition with the opponent AI, two properties of an experimental run are used: (1) the absolute turning point, and (2) the relative turning point.

We define the absolute turning point as the time at which the adaptive team obtains a win-loss ratio of a least 15 wins against 5 losses in a sliding window of 20. When the ratio is reached, the probability of the adaptive team outperforming the non-adaptive team is greater than 98% (cf. randomisation test, Cohen, 1995).

We define the relative turning point, which quantifies the noticeable effect of successful adaptive behaviour, as the last time at which the adaptive team has a zero lead with respect to the non-adaptive team, with the requirement that from this moment on the adaptive team does not lose its lead for the rest of the experimental run. The lead of the adaptive team is defined as the score of the adaptive team minus the score of the non-adaptive team.

C. Results

In Table 3.1 an overview of the experimental results is given. The mean absolute turning point acquired is 108, and the mean relative turning point is 71. TEAM requires several dozens of trials to evolve excellent behaviour. This is a good result, considering that evolutionary algorithms typically require several thousands of trials to evolve effective behaviour.

To illustrate the course of a typical experimental run, we plotted the absolute and relative performance in Figure 3.6. As shown in the top graph of Figure 3.6 (absolute performance),

	Absolute turning point	Relative turning point
Mean	108	71
StDev	62	45
Median	99	54
Minimum	38	20
Maximum	263	158

Table 3.1: Summary of experimental results obtained by TEAM. They are average results over fifteen experimental runs with the TEAM mechanism.

initially, that is after the sliding window of 20, the adaptive team obtains approximately 10 wins against 10 losses; this is considered to be neutral performance. At the absolute turning point (i.e., 99 in the figure) a drastic increase of the performance of the adaptive team is observed. In the bottom graph of Figure 3.6 (relative performance), we observe that, initially, the adaptive team obtains a lead of approximately zero. At the relative turning point (i.e., 54 in the figure) the lead of the adaptive team increases substantially.

D. Analysis of the results

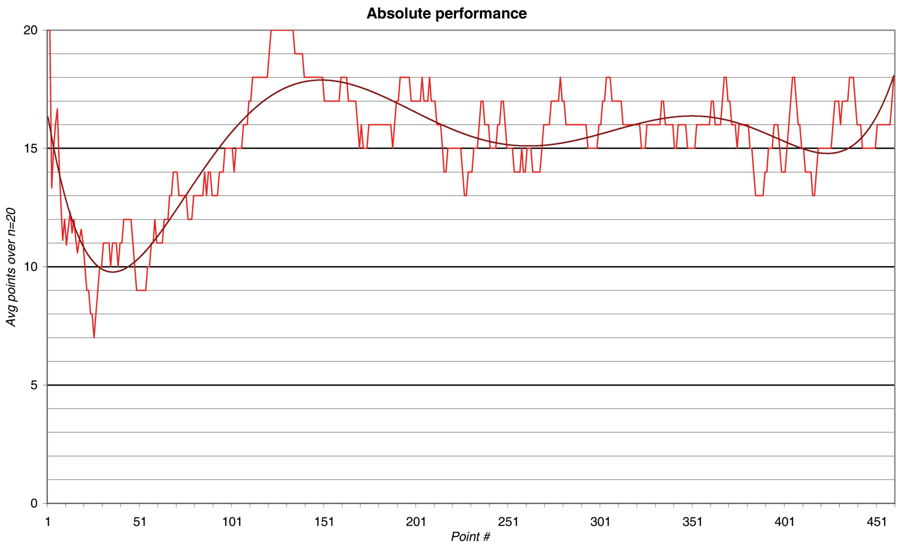
The obtained experimental results reveal that TEAM is able to counter successfully non-static opponent behaviour, as it defeated the non-static Quake III CTF team AI. In Subsection 3.2.4 we will discuss that this is the result of TEAM discovering and applying unforeseen dominant tactics.

Table 3.1 shows that the mean relative turning point is much below the mean absolute turning point. From this observation we may conclude that before we can reliably determine whether the absolute turning point is reached, the opponent is already subject to the dominant effect of TEAM. Figure 3.6 reveals that TEAM learned to outperform the opponent without any substantial degradation in the lead of the adaptive team.

Considering that in all runs we were able to obtain relatively low turning points, implying that TEAM learned to outperform the opponent (in this case the Quake III CTF team AI), we may draw the conclusion that TEAM is capable of adapting successfully to significant changes in the opponent behaviour.

3.2.4 Discussion of TEAM

Results of the experiment performed in Quake III CTF indicate that TEAM is a successful adaptation mechanism for team-oriented behaviour. In this subsection we will first provide (A) a qualitative evaluation of TEAM. Subsequently, we discuss (B) the behaviour learned by TEAM. Then, we draw (C) the experimental conclusions. Finally, suggestions are given for (D) broadening the application domain of TEAM.



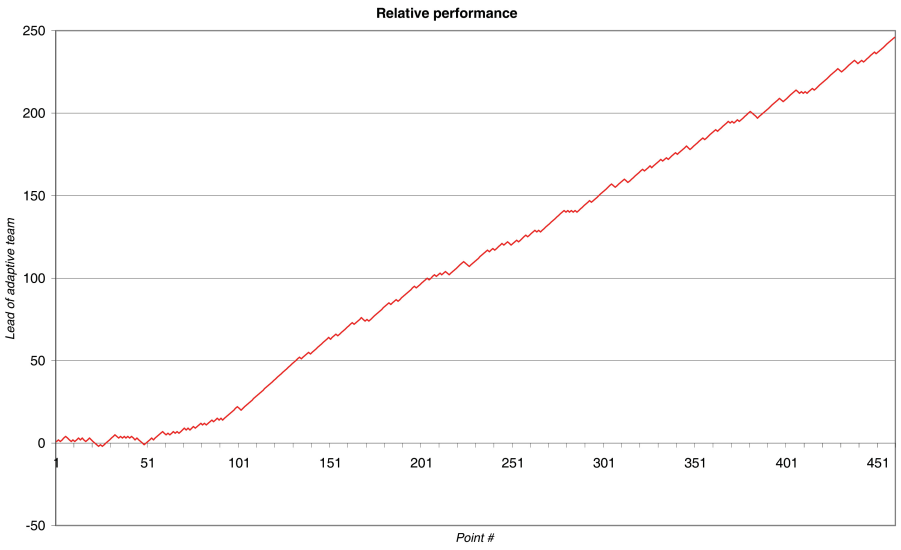


Figure 3.6: Illustration of typical experimental results obtained by TEAM. The top graph shows the points scored by the adaptive team over a sliding window of 20 as a function of the amount of scored points. The bottom graph shows the lead of the adaptive team over the non-adaptive team as a function of the amount of scored points. The bottom graph reveals that the adaptive team outperforms the non-adaptive team without any substantial degradation in the lead of the adaptive team.

A. Qualitative evaluation of TEAM

TEAM is an online adaptation mechanism. For online adaptation to be applied in practice, we denote four requirements for qualitatively acceptable performance. It must be (1) computationally fast, (2) robust with respect to randomness inherent to the environment, (3) efficient with respect to the number of adaptation trials, and (4) effective with respect to the intermediate AI generated during the adaptation phase (Spronck *et al.*, 2004b).

TEAM is computationally fast in a sense that it only needs to return a single genome to the application and update a small population after each state transition. In addition, TEAM is robust in a sense it is able to cope with a large amount of randomness inherent in the environment. As reasoned in Subsection 3.2.3, TEAM is efficient with respect to the limited number of trials required for an opponent to be subject to the effects of dominant adaptive behaviour. The effectiveness of TEAM is expressed by the fact that it outperforms non-adaptive AI without any substantial degradation the lead of the adaptive team. We may therefore conclude that TEAM is computationally fast, robust, efficient, and effective, and may be applied in practice in the examined first-person shooter game.

B. Learned behaviour

Analysing the behaviour of TEAM, we observed that the learned behaviour does not converge to merely one set of dominant tactics. TEAM is continuously adapting to the environment in order to remain dominant throughout the game. In our experiment, the adaptive team has learned relatively risky, yet successful tactics. These tactics can be best described as ‘rush’ tactics, which are often applied in real-time strategy (RTS) games. Rush tactics are tactics aimed at rapidly obtaining field supremacy. If rush tactics are successful, the opponent can seldom recover from the momentum of the attacking team.

The original QUAKE III CTF team AI uses only moderate tactics in all states. Therefore, it is not able to counter significant field supremacy. This exemplifies the inadequacy of non-adaptive AI. Despite the fact that the original QUAKE III CTF team AI is fine-tuned to be suitable for typical situations, it cannot adapt to superior player tactics, whereas TEAM can.

C. Experimental conclusions

From the results of the experiment performed in QUAKE III CTF, we may draw the conclusion that TEAM is capable of adapting successfully to changes in the opponent behaviour. TEAM adapted the team behaviour in such a way that dominant tactics were discovered. These dominant tactics outperformed the tactics of the original QUAKE III CTF team AI, which exhibits non-adaptive behaviour. Therefore, we may conclude that by creating a team-oriented mechanism capable of automated and intelligent adaptation to the environment, we established a solid basis for adaptive team AI for the examined first-person shooter game.

D. Broadening the application domain of TEAM

Of the four requirements for qualitatively acceptable performance set by Spronck (2005a), the requirement of efficiency is of main relevance for enhancing an adaptation mechanism for video games. Here, efficiency is defined as the learning time of the mechanism. In adaptive game AI, efficiency depends on the number of learning trials needed to adopt effective behaviour. Occasionally, we observed that the TEAM mechanism required over two hours of real-time play to outperform the opponent. In the case of QUAKE III CTF, matches take on average half an hour. Due to the large variation in the time that is needed to learn the appropriate tactics, the application of TEAM is therefore limited.

To broaden the application domain of TEAM, its efficiency needs to be enhanced. This will be discussed in the next section.

3.3 Incremental adaptive game AI with TEAM2

The TEAM2 mechanism is an extension of the previously explored TEAM mechanism for team-oriented adaptive behaviour. It focusses on the exploitation of recent game observations. Our experiment with TEAM revealed that the mechanism is applicable to first-person shooter video games, such as QUAKE III CTF. However, the applicability of TEAM is limited due to the large variation in the time that is needed to learn the appropriate tactics.

This section describes our attempts to improve the efficiency of the TEAM mechanism by using implicit opponent models (see Van den Herik *et al.*, 2005). We propose an extension of TEAM called TEAM2. The TEAM2 mechanism employs a data set of a limited history of results of tactical team behaviour. Observations gathered in the data set constitute an implicit opponent model, on which a best-response strategy (cf. Carmel and Markovitch, 1997) is formulated. We will show that *best-response learning* of team-oriented behaviour can be applied in an actual video game, and investigate to what extent it is suitable for online learning.

In the remainder of this section, we first discuss the TEAM2 mechanism (3.3.1). Subsequently, we report on an experiment that tests the mechanism in play against the original QUAKE III CTF team AI (3.3.2). Finally, we provide a discussion of the obtained results (3.3.3).

3.3.1 The TEAM2 adaptation mechanism

We established an enhanced design for online learning of team-oriented behaviour, named TEAM2.[1] The four features of the enhanced design are as follows: (1) a symbiotic learning concept, (2) a state-transition-based fitness function, (3) learning a best-response team strategy, and (4) a scaled roulette-wheel selection. Both the symbiotic learning concept as well as the state-transition-based fitness function are the same as applied in TEAM. Below we discuss our approach to (A) learning a best-response team strategy, and the applied (B) scaled roulette-wheel selection. The popular QUAKE III CTF game, is again used for illustrative purposes.

[1] Since TEAM2 is not inspired by evolutionary algorithms, we let the reader imagine that the letter 'E' is an abbreviation for 'Exploitative' (instead of 'Evolutionary').

Team configuration	History	Fitness
(0,0,4)	[0.1,0.6,...,0.5]	0.546
(0,1,3)	[0.3,0.1,...,0.2]	0.189
⋮	⋮	⋮
(4,0,0)	[0.8,0.6,...,0.9]	0.853

Table 3.2: Example of an implicit opponent model for a specific state of the Quake III CTF game.

A. Learning a best-response team strategy

Adaptation to the opponent takes place via an implicit opponent model, which is built and updated when the team game is in progress. Per state of the game (i.e., four in Quake III CTF), data is sampled that merely concerns a specific state and represents all possible team configurations for that particular state. The implicit opponent model consists of historic data of results per team configuration per state. An example of the structure of an implicit opponent model is given in Table 3.2. In the example, the team configuration represents the role division of a team with four members, each of which has either an offensive, a defensive, or a roaming role. The history can be anything ranging from a store of fitness values to a complex data structure.

On this basis, a best-response strategy is formulated when the game transits from one state to another. For reasons of efficiency and relevance, only relatively recent historic data are used for the learning process.

B. Scaled roulette-wheel selection

The best-response learning mechanism selects the preferred team configuration by implementing a roulette-wheel method (Nolfi and Floreano, 2000), where (1) each slot of the roulette wheel corresponds to a team configuration in the state-specific solution space, and (2) the size of the slot is proportional to the obtained fitness value of the team configuration. The selection mechanism quadratically scales the fitness values to select the higher-ranking team configurations more often, considering that behaviour exhibited by the team AI must be non-degrading. In acknowledgement of the inherent randomness of a game environment, the scaled roulette-wheel selection mechanism averts selecting inferior top-ranking team configurations.

3.3.2 TEAM2 vs. QUAKE III CTF team AI

To assess the performance of the TEAM2 mechanism, we incorporated it in the Quake III CTF game. We repeated the original experiment with the TEAM mechanism, by pitting an adaptive team (now controlled by TEAM2) against a non-adaptive team (controlled by the Quake III CTF team AI). In the experiment, the TEAM2 mechanism adapts the tactical behaviour of a team to the opponent. The results obtained by TEAM2 will be compared to those obtained by the TEAM mechanism. The experimental setup is identical

	TEAM	TEAM2
Experimental runs	15	20
Outliers	4	6
Outliers (%)	27%	30%
Mean	71.33	102.20
StDev	44.78	125.29
Median	54	38
Minimum	20	2
Maximum	158	358

Table 3.3: Summary of experimental results. With TEAM2 the median performance has improved substantially, yet, outliers have a negative effect on the mean performance.

to that of the experiment with the TEAM mechanism (see Subsection 3.2.3). On average, the game is interrupted after two hours of gameplay; the original TEAM mechanism typically requires two hours to learn successful behaviour, whereas the TEAM2 mechanism should perform more efficiently. We perform twenty experimental runs with the TEAM2 mechanism.

The performance assessment is comparable to that of the experiment with the TEAM mechanism. To investigate the efficiency of the TEAM2 mechanism, we defined two performance indicators: (1) the median relative turning point, and (2) the mean relative turning point. The choice for the two indicators is motivated by the observation that the amount of variance influences the learning performance of the mechanism. To investigate the variance of the experimental results, we defined an outlier as an experimental run which needed more than 91 time steps to acquire the turning point (the equivalent of two hours of gameplay).

In the remainder of this subsection, we subsequently give (A) the obtained results, and (B) an analysis of the results.

A. Results

In Table 3.3 an overview of the experimental results of the TEAM2 experiment is given. It should be noted that in two tests, the run was prematurely interrupted without a turning point being reached. We incorporated these two tests as having a turning as high as the highest outlier, which is 358. The experimental results indicate that, should the runs not be prematurely interrupted, their turning points would have been no more than half of this value.

The median turning point acquired is 38, which is substantially lower than the median turning point of the TEAM mechanism, which is 54. The mean turning point acquired by TEAM2, however, is substantially higher than the mean turning point acquired by the TEAM mechanism (102 and 71, respectively). The percentage of outliers in the total number of tests is equal by approximation. However, the range of the outliers, expressed by the measured minimum and maximum turning points, has increased considerably for TEAM2.

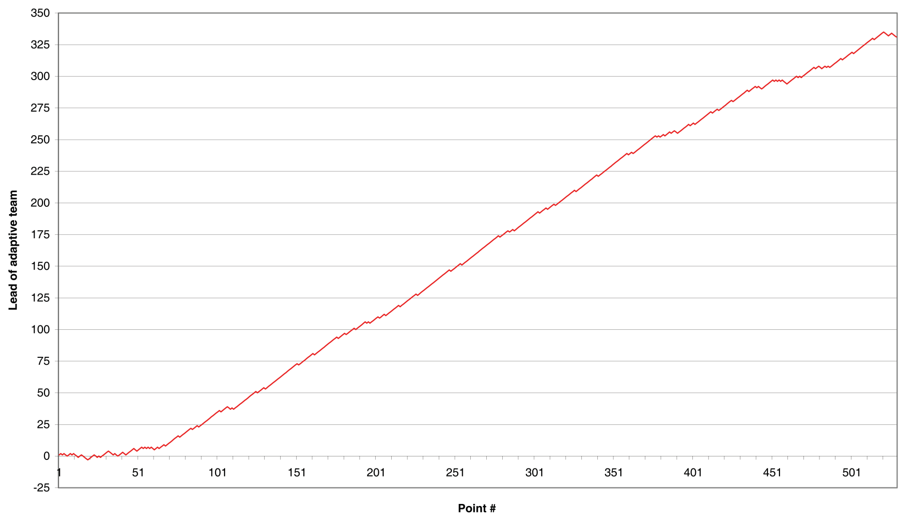


Figure 3.7: Illustration of typical experimental results obtained by the TEAM2 mechanism. The graph shows the lead of the adaptive team over the non-adaptive team as a function of the number of scored points.

To illustrate the course of an experimental run, we plotted the performance for a typical run in Figure 3.7. The performance is expressed in terms of the lead of the adaptive team, which is defined as the score of the adaptive team minus the score of the non-adaptive team. The graph shows that, initially, the adaptive team attains a lead of approximately zero. At the turning point (i.e., 38 in Figure 3.7, as compared to 54 in Figure 3.6), the adaptive team takes the lead over the non-adaptive team. Additionally, the graph reveals that the adaptive team outperforms the non-adaptive team without any substantial degradation in the lead of the adaptive team.

B. Analysis of the results

The experimental results indicate that TEAM2 is able to adapt successfully the behaviour of team AI in a highly non-deterministic environment, as it challenged and defeated the fine-tuned Quake III CTF team AI.

The results listed in Table 3.3 show that the TEAM2 mechanism outperforms the TEAM mechanism in terms of the median turning point. However, the mean turning point is larger for TEAM2 than for TEAM, which is explained by the increased range of the outliers. The median turning point indicates that the TEAM2 best-response learning mechanism is more efficient than the TEAM online evolutionary learning mechanism, as the adaptation to successful behaviour progresses more swiftly than before; expressed in time only forty-eight minutes are required (as compared to sixty-nine minutes).

On the basis of the obtained increase in efficiency, we may draw the conclusion that the TEAM2 mechanism broadens the application domain of the TEAM mechanism for the purpose of rapid learning in the examined first-person shooter game. A discussion of the experimental results is given next.

3.3.3 Discussion of TEAM2

Our experimental results show that the TEAM2 mechanism succeeded in enhancing the learning performance of the TEAM mechanism with regard to its median performance, but not to its mean performance. In this subsection, we first give (A) a comparison of the learned behaviour of both mechanisms. Subsequently, we discuss (B) the exploitation versus exploration dilemma for online learning in a video-game environment. We end the subsection by drawing (C) experimental conclusions.

A. Comparison of the behaviour learned by TEAM and TEAM2

In the original TEAM experiment we observed that the adaptive team would learn so-called 'rush' tactics. Rush tactics aim at quickly obtaining offensive field supremacy. We noted that the original QUAKE III CTF team AI, as it was designed by the developers of the game, uses only moderate tactics in all states, and therefore, it is not able to counter significant field supremacy.

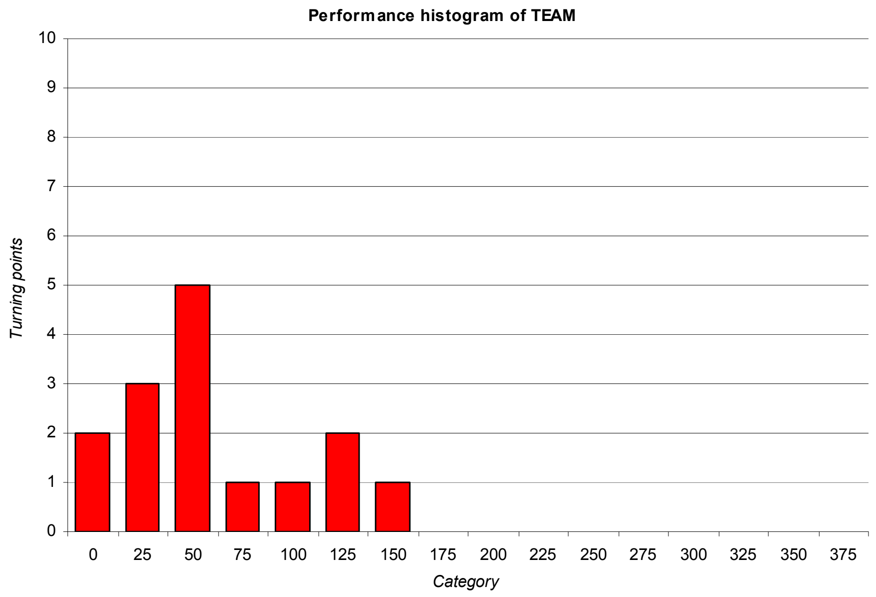
The TEAM2 mechanism is inclined to learn rush tactics as well. Notably, the experiment showed that if the adaptive team uses tactics that are slightly more offensive than the non-adaptive team, it is already able to outperform the opponent. Besides the fact that the original QUAKE III CTF team AI cannot adapt to superior player tactics (whereas an adaptation mechanism can), it is not sufficiently fine-tuned; it implements a seemingly obvious and easily detectable local optimum.

B. Exploitation versus exploration

The experimental results revealed that the exploitative TEAM2 mechanism obtained a substantial difference between the median performance (which was relatively low) and mean performance (which was relatively high), whereas the original, less exploitative, TEAM mechanism obtained a moderate difference between the median and mean performance.

An analysis of the phenomenon revealed that it is due to a well-known dilemma in machine learning: the exploitation versus exploration dilemma (Carmel and Markovitch, 1997). This dilemma entails that a learning mechanism requires the exploration of derived results to yield successful behaviour in the future, whereas at the same time the mechanism needs to exploit directly the derived results to yield successful behaviour in the present. Acknowledging the need for an enhanced efficiency, the emphasis of the TEAM2 mechanism lies on exploiting the data represented in a small amount of samples. Performance histograms of results obtained by TEAM and TEAM2 are given in Figure 3.8.

In the highly non-deterministic QUAKE III CTF environment, a long series of fitness values may occur that, due to chance, is not representative for the quality of the tactic employed. Indeed, this problem results from the emphasis on exploiting the small samples ta-



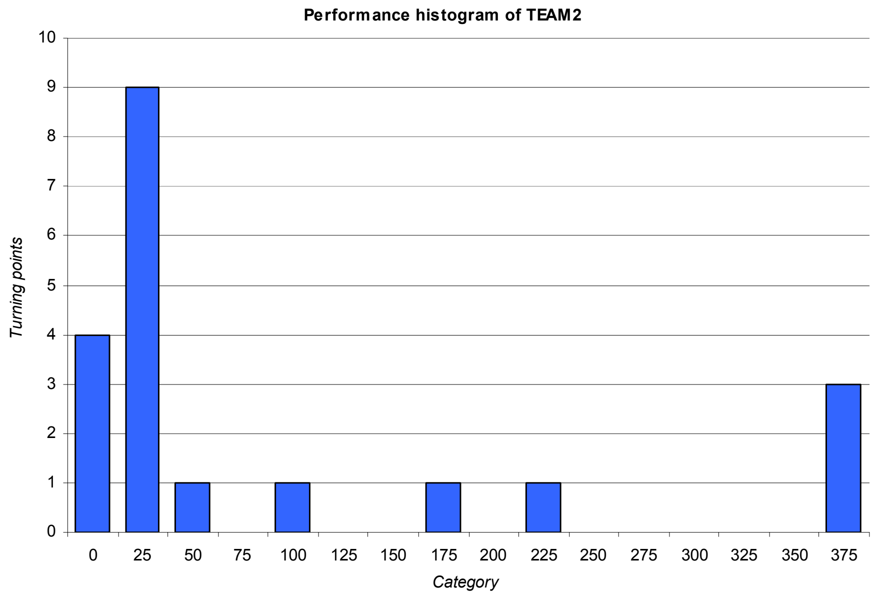


Figure 3.8: Histograms of the results of both the TEAM and TEAM2 experiment. The graphs show the number of turning points as a function of the value of the turning point, grouped by a category value of 25.

ken from the distribution of all states. To increase the number of samples, an exploration mechanism can be added. The TEAM online evolutionary learning mechanism employed such an exploration mechanism by propagating fitness values, which led to loss of efficiency. We tested several exploration mechanisms in TEAM2, which we found also led to loss of efficiency. However, since it is impossible to rule out chance runs completely, an online learning mechanism must be balanced between an exploitative and explorative emphasis.

C. Experimental conclusions

From the results of the experiment performed in Quake III CTF, we may draw the conclusion that the TEAM2 best-response learning mechanism succeeded in enhancing the median learning performance, but not the mean learning performance. This reveals that, in the current experimental setup, the exploitation and exploration are not sufficiently well balanced to allow efficient and effective online learning in the Quake III CTF game. As the TEAM2 mechanism is easily able to defeat a non-adaptive opponent, we may therefore conclude that the mechanism is suitable for online learning in an actual game if a balance between exploitation and exploration is found for that specific game. Moreover, the TEAM2 mechanism can be used during game-development practice to validate and produce game AI automatically.

3.4 Practical applicability

In this section we give an analysis of the practical applicability of incremental adaptive game AI. Following an incremental approach to adaptive game AI, in the two previously discussed experiments we established the TEAM and TEAM2 mechanisms for *online* adaptation of game AI. We compared the performance of the TEAM2 mechanism with that of the original TEAM mechanism operating in the actual video game Quake III CTF. The obtained results showed that the TEAM2 mechanism could more efficiently learn effective team behaviour. Still, we observed that the application as an online learning mechanism is limited due to occasionally very long learning times that result from an improper balance between exploitation and exploration. From these results we concluded that the TEAM2 mechanism is suitable for online learning in an actual game if a balance between exploitation and exploration is found for that specific game.

However, establishing an effective balance between exploitation and exploration is hampered by the fact that in an incremental approach to adaptive AI, this would require either (1) a high quality of the domain knowledge used (which generally is unavailable to the AI), or (2) a large number of trials to learn effective behaviour online (which is highly undesirable in an actual video game). Naturally, going through a large number of adaptation trials does not coincide with our goal of adapting rapidly (and by extension reliably) game AI. As a result, one can only increase the applicability of game AI that is established by following an incremental approach, by improving the quality of the domain knowledge that is exploited. Spronck (2005a) argues that machine learning techniques may be applied in an offline fashion, to improve the quality of domain knowledge after a game has been played. However, research into offline improvements of domain knowledge, though important, is strictly domain specific, i.e., results achieved are only applicable in one particular video game. For instance, research by Ponsen *et al.* (2007) showed that offline improvements in the quality of the domain knowledge may indeed lead to more effective play in the WARGUS game. Still, offline improvements are typically achieved on the basis of self-play, or on the basis of play against other game AIs. Thus, offline improvements will generally not reflect the knowledge that is required in actual game-playing conditions, i.e., in online play against human opponents. As a result, numerous learning trials will still be required for online adaptation of the game AI.

Considering the previous discussion, it is clear that incremental adaptive game AI can be applied successfully to some extent, but not to the extent that it can be used to adapt game AI rapidly and reliably in an online fashion. For rapid and reliable online adaptation of game AI, it therefore is necessary that an alternative is established for the incremental adaptive game AI approach. Our proposal for an alternative approach will be the focus of the chapters that follow.

3.5 Chapter conclusions

This chapter discussed an approach that is commonly applied to create adaptive game AI, i.e., incremental adaptive game AI. Following the approach, we established the TEAM and TEAM2 mechanism for online adaptation of game AI. We tested TEAM and TEAM2 in the actual first-person shooter game QUAKE III CTF, and from our experimental results we concluded that the mechanisms are capable of adapting successfully to changes in the opponent behaviour. However, application of TEAM and TEAM2 as an online learning mechanism was hampered by occasionally very long learning times due to an improper balance between exploitation and exploration. We discussed that this issue characteristically follows from the incremental adaptive game AI approach, which requires either (1) a high quality of the domain knowledge used (which generally is unavailable to the AI), or (2) a large number of trials to learn effective behaviour online (which is highly undesirable in an actual video game). Thus, from the results of this chapter we may conclude that the characteristics of incremental adaptive game AI prohibit our goal of establishing game AI capable of adapting rapidly and reliably to game circumstances. Therefore, we propose to investigate an alternative approach to adaptive AI that comes without these characteristics.

4 Case-based adaptive game AI

In this chapter we propose an alternative, novel approach to adaptive game AI that comes without the hampering characteristics of incremental adaptive game AI. The approach is coined 'case-based adaptive game AI'.

Chapter 4 is organised as follows. We first outline case-based adaptive game AI (Section 4.1). Subsequently, we perform two experiments to obtain an early indication of the effectiveness of case-based adaptive game AI. In the first experiment, we test a limited form of case-based adaptive game AI in an actual video game (Section 4.2). In the second experiment, we test case-based adaptive game AI in a limited, simulated video-game environment (Section 4.3). Next, we describe how to establish case-based adaptive game AI in actual video games (Section 4.4). Finally, we provide a summary of the chapter (Section 4.5).

4.1 What is case-based adaptive game AI?

This section outlines our approach to adaptive game AI. First, we describe case-based reasoning, by which our approach is inspired (4.1.1). Subsequently, we define the approach, which we coined 'case-based adaptive game AI' (4.1.2). Then, we discuss the contributions and limitations of the approach (4.1.3).

This chapter is based on the following two publications.

1) Bakkes, S. C. J. and Spronck, P. H. M. (2006). Gathering and utilising domain knowledge in commercial computer games. In Schobbens, P.-Y., Vanhoof, W., and Schwanen, G., editors, *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2006)*, pages 35–42. University of Namur, Namur, Belgium.

2) Van der Blom, L. L., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Map-adaptive artificial intelligence for video games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 53–60. EUROSIS-ETI, Ghent University, Ghent, Belgium.

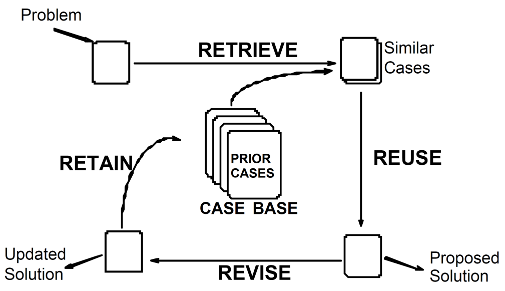


Figure 4.1: The case-based reasoning cycle. Adapted from Aamodt and Plaza (1994).

4.1.1 Case-based reasoning

For establishing adaptive game AI, we draw inspiration from case-based reasoning (Kolodner, 1993; Aamodt and Plaza, 1994; Leake, 1996; Lopez de Mantaras *et al.*, 2005). Case-based reasoning (CBR) is a *methodology* (Watson, 1999) for interpretation and problem solving based on the explicit storage and reuse of experiences (or their generalisations). An observation on which problem solving is based in CBR, namely that similar problems have similar solutions (e.g., Leake and Wilson, 1999), has been shown to hold in expectation for simple scenarios (Faltings, 1997), and is empirically validated in many real-world domains (Lopez de Mantaras *et al.*, 2005).

Conceptually case-based reasoning is commonly described by the CBR-cycle (illustrated in Figure 4.1). This cycle comprises four stages: Retrieve, Reuse, Revise, and Retain. Ontañón *et al.* (2009) describe the stages as follows. In the Retrieve stage, the system selects a subset of cases from the case base that are relevant to the current problem. The Reuse stage adapts the solution of the cases selected in the retrieve stage to the current problem. In the Revise stage, the obtained solution is verified (either by testing it in the real world or by examination by an expert), which provides feedback about the correctness of the predicted solution. Finally, in the Retain stage, the system decides whether or not to store the newly solved case into the case base.

For adaptive game AI, we desire to generate character behaviour and player models automatically, readily fitted to circumstances in actual, online play, on the basis of previous gameplay experiences. Case-based reasoning provides an strong starting point for realising this desire in the form of a proof of concept.

4.1.2 Approach

We define case-based adaptive game AI as a CBR-inspired approach to game AI where domain knowledge is gathered automatically by the game AI, and is immediately (i.e., without trials and without resource-intensive learning) exploited to create effective behaviour. The approach collects character and game-environment observations, and extracts from those a 'case base'. This approach to adaptive game AI is expected to be particularly successful in games that have access to the internet to store and retrieve samples of gameplay experiences.

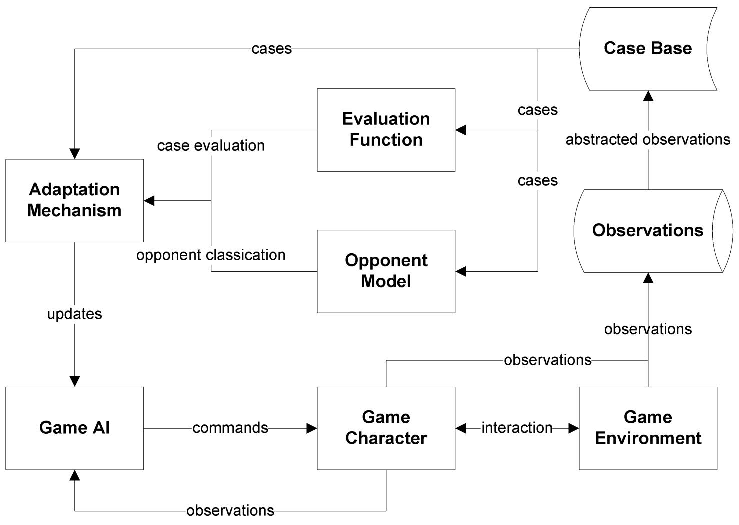


Figure 4.2: Case-based adaptive game AI (see text for details).

For instance, in the popular Massive Multiplayer Online Games (MMOGs), observations from many games played against many different opponents will be available to the game AI.

The approach, illustrated in Figure 4.2, selects a subset of cases from the case base that are relevant for the current game circumstances (cf. retrieval). The selected cases are adapted to fit the current game circumstances (cf. reuse), and are verified in online gameplay (cf. revise). Newly generated gameplay experiences are stored in the case base (cf. retain). As such, the approach extends the feedback loop of incremental adaptive game AI by (1) explicitly processing observations from the game AI, and (2) allowing the use of game-environment attributes which are not directly observed by the game character (e.g., observations of team members).[1]

Like incremental adaptive game AI, case-based adaptive game AI will typically be implemented for performing adaptation of the game AI in an *online* and *computer-controlled* fashion. Improved behaviour is established by continuously making (small) adaptations to the game AI. Case-based adaptive game AI in such an implementation can be characterised by its scheme for case-based adaptation. This scheme can be summarised as follows.

Case-based adaptation: To adapt to circumstances in the current game, the adaptation process is based on domain knowledge drawn from observations of a *multitude* of games. The domain knowledge is gathered in a case base, and is typically used to ex-

[1] In many games, human team members share game observations with each other in order to play effectively. Analogously, the designer of game AI may choose to incorporate game observations from several sources into the game AI in order to enhance its effectiveness. In some cases, game AI incorporates game observations that could not possibly be observed by the game AI itself (e.g., observations of the opponent players). The latter is typically considered as a form of 'cheating' behaviour.

tract models of game behaviour, but can also be exploited directly to adapt the AI to game circumstances.

In our proposal of case-based adaptive game AI three main components are required for game adaptation, viz. (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. A case base is used to extract an evaluation function and opponent models. Subsequently, the evaluation function and opponent models are incorporated in an adaptation mechanism that directly exploits the gathered cases. We will introduce the three main components in Section 4.4, and discuss each component in more detail in the chapters that follow.

The approach to case-based adaptive game AI is inspired by the human capability to reason reliably on a preferred course of action with only a few *online* observations on the problem domain. Following from the complexity of modern video games we propose that for effective and rapid use, game observations should be (1) represented in such a way that the stored cases can be reused for previously unconsidered situations, and (2) be compactly stored in terms of the amount of gathered observational data. As far as we know, case-based adaptive game AI has not yet been applied in an actual, commercially released video game.

The actual video game in which we intend to implement case-based adaptive game AI, is the RTS game Spring. Spring is a complex game for which establishing effective adaptive game AI is a challenging task. The game, as well as the topic of complexity, are described further in Appendix A.2. To obtain an early indication of the effectiveness of case-based adaptive game AI, we first will implement a limited form of case-based adaptive game AI in Spring, and subsequently, we will implement case-based adaptive game AI in a limited, simulated video-game environment.

4.1.3 Contributions and limitations of the approach

Game developers may consider using an alternative approach to game AI when they are convinced of its qualitative effectiveness. The main contribution of our research therefore is demonstrating that case-based adaptive game AI can be applied in an actual, commercially released video game. In the remainder of the thesis we will demonstrate in a complex RTS game that the approach can be applied to both generate effective game strategies, and to scale automatically the difficulty level to the player.

To this end, the approach is adjusting a pre-existing game AI, rather than 'being the AI' (as most previous approaches to game AI). It is based on a case base drawn from observations of a multitude of games. Particularly in the popular massive multiplayer online games, the case base is expected to grow rapidly. With a sufficiently large and diverse case base, the adaptation process no longer needs to go through a large number of adaptation trials, but instead can adapt instantly to game circumstances. Furthermore, the game AI will become robust in dealing with non-determinism, since the case base can be used as a model to predict the results of game strategies. An advantage of the approach tying in with a pre-existing game AI, is that it enables game developers to control and predict with relative accuracy the behaviour that is exhibited by the game AI. In addition, the case base can be utilised for providing feedback on distinct strengths and weaknesses of a player, and can provide inexpensive insight into the balance of a game. Thus, it can help in testing and debugging the game.

Indeed, the approach has certain limitations. A first limitation is that it is not fully knowledge free, but requires some domain knowledge to steer the adaptation process. For instance, the features to compare the similarity of game observations need to be established. We assume that such domain knowledge is available to the developers of the game. A second limitation is that for effective adaptation, the case base needs to contain cases relevant to the current circumstances. This limitation is partially addressed by generalising over stored cases. A third limitation is that the ability to adapt to changing circumstances is restricted by the behavioural expressiveness provided by the game AI that the approach ties in with.

4.2 Case-based adaptive game AI for map adaptation

In this section, we report on an experiment to test a limited form of case-based adaptive game AI in an actual video game. A limited form of case-based adaptive game AI is to include offline-generated domain knowledge in the adaptation process of game AI. This form of case-based adaptive game AI does not implement the cycle to abstract actual game observations into a case base, to be used in the adaptation process (see Figure 4.2); the adaptation mechanism operates without reasoning on the generated domain knowledge.

Still, the described form of adaptation may be used in many instances of adaptive game AI. One such instance of adaptive game AI, is game AI with the ability to automatically establish effective behaviour dependent on features of the game environment (i.e., the so-called map). This is called 'map-adaptive game AI'. In our implementation of map-adaptive game AI, a particular map is first analysed for specific features. Subsequently, an automatically constructed decision tree is applied to adapt the game AI according to features of the map. To automatically construct the decision tree, information is incorporated from a data set of effective game strategies.

In this section we will first present our approach to establish map-adaptive game AI (4.2.1). Next, we report on the experiment that tests the approach (4.2.2). The section is concluded by a discussion of the obtained results (4.2.3).[2]

4.2.1 Approach to map adaptation

Our approach to map adaptation consists of three components: (A) definition of the features of a map, (B) determination of compound actions of map-adaptive strategies, and (C) automatic construction of a decision tree of map-adaptive strategies.

We establish map-adaptive game AI in an RTS game environment, i.e., a simulated war game. Here, a player needs to gather resources for the construction of units and buildings. The goal of the game is to defeat an enemy army in a real-time battle. We use RTS games for their highly challenging nature, which stems from three factors: (1) their high complexity, (2) the large amount of inherent uncertainty, and (3) the need for rapid decision making

[2] The work reported on in this section is performed by Van der Blom *et al.* (2007), under supervision of the author.

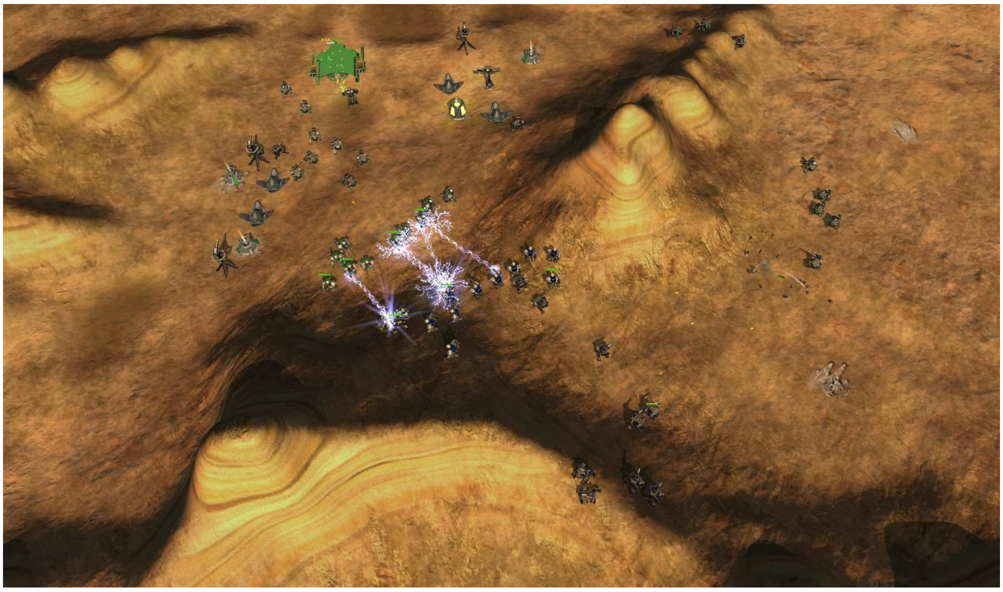


Figure 4.3: Screenshot of the Spring game environment. In the screenshot, the base on the top left is being attacked via a narrow cliff passage.

(Buro and Furtak, 2004). In the present research we use Spring, illustrated in Figure 4.3, which is a typical and open-source RTS game. A Spring game is won by the player who first destroys the opponent's 'Commander' unit. An extensive description of Spring is provided in Appendix A.2.

A. Features of a map

To automatically establish map-adaptive game AI, we start by defining a basic set of features that will play an essential role in the strategy of a player. For our experiment, we decided to use five features of a map. These five features are (1) number of metal resources (RN), (2) resource density (RD), (3) number of relatively narrow roads (e.g., due to obstacles such as mountains and rivers) (NR), (4) number of cliffs (CL), and (5) distance between base locations (DE). Indeed, we admit that by manually defining and selecting the features of a map we may restrict the effectiveness of map adaptation. The investigation of further improvements with respect to the definition and selection of features is considered a topic for future research.

Feature values will be used to automatically construct a decision tree of map-adaptive strategies. If we would allow all possible feature values, an explosion in the number of nodes in the decision tree would occur.[3] We therefore divide the range of feature values in bands (Evans, 2002), in our case 'None', 'Few', and 'Many'. We note that game maps may vary in size. Therefore, feature values are scaled proportionally to the size of the map.

B. Map-adaptive game AI actions

When the game map has been analysed, game AI is established on the basis of compound actions of map-adaptive strategies. For our experiment, we decided to use seven compound actions. These seven compound actions are (1) construction of metal extractors at nearby metal resources, (2) construction of metal extractors at far away metal resources, (3) placement of offensive units at relatively narrow roads, (4) placement of offensive units at own base, (5) placement of artillery on cliffs, (6) protection of operational metal extractors, and (7) protection of artillery on cliffs. The defined compound actions can be used to establish offensive as well as defensive stances of game AI.

C. Decision tree of map-adaptive strategies

In our experiment, we consider a decision tree to be a tree where (1) each internal node represents a feature to be analysed, (2) each branch corresponds to a band of feature values, and (3) each leaf node assigns a classification. In this setup, the decision tree has a discrete outcome, namely a classification based on the input features values. Such a decision tree may be referred to as a 'classification tree'.

In the Spring game, each feature of the map can be expressed by discrete values (e.g., the number of metal resources). Analogous to work by Fu and Houlette (2004), we employ the standard ID3 learning algorithm (Quinlan, 1986; Mitchell, 1997) to construct a decision tree for the Spring game. The ID3 algorithm has a preference for constructing shallow trees, with the features with a high information gain located near the root of the tree.

4.2.2 Experiment with map adaptation

This subsection reports on an experiment that tests our approach to establish map-adaptive game AI. We first describe (A) the process of constructing the decision tree, that will be used in the experiment. Subsequently, we discuss (B) the experimental setup. Next, we present and discuss (C) the obtained results.

A. Constructing the decision tree

As mentioned in Subsection 4.2.1, we use the ID3 learning algorithm to construct the decision tree from experimentally determined training data. The training data consist of input data with values for features of the map, and the corresponding target output data in the form of actions of the game AI.

[3] Such an explosion in the number of nodes would occur for the adopted ID3 algorithm (see Subsection 4.2.1-C) but not necessarily for other TDIDT algorithms, such as those that perform a type of pruning (cf. Breslow and Aha, 1997).

Example	Feature					Action(s)
	RN	RD	NR	CL	DE	
X1	Few	High	No	None	Far	1
X2	Few	High	No	Few	Far	1
X3	Few	High	No	Many	Far	1,5
X4	Few	High	Yes	None	Far	1,3
X5	Few	High	Yes	Few	Far	1,3
X6	Few	High	Yes	Many	Far	1,3,5
X7	Few	Low	No	None	Far	1—2
X8	Few	Low	No	Few	Far	1—2
X9	Few	Low	No	Many	Far	1—2,5
X10	Few	Low	Yes	None	Far	1—2,3
X11	Few	Low	Yes	Few	Far	1—2,3,5
X12	Few	Low	Yes	Many	Far	1—2,3,5
X13	Many	High	No	None	Far	1+2
X14	Many	High	No	Few	Far	1+2
X15	Many	High	No	Many	Far	1+2,5
X16	Many	High	Yes	None	Far	1+2,3
X17	Many	High	Yes	Few	Far	1+2,3
X18	Many	High	Yes	Many	Far	1+2,3,5
X19	Many	Low	No	None	Far	1+2
X20	Many	Low	No	Few	Far	1+2
X21	Many	Low	No	Many	Far	1+2,5
X22	Many	Low	Yes	None	Far	1+2,3
X23	Many	Low	Yes	Few	Far	1+2,3,5
X24	Many	Low	Yes	Many	Far	1+2,3,5
X25	Few	High	No	None	Close	1,4,6
X26	Few	High	No	Few	Close	1,4,6
X27	Few	High	No	Many	Close	1,5,6,7
X28	Few	High	Yes	None	Close	1,3,4,6
X29	Few	High	Yes	Few	Close	1,3,4,6
X30	Few	High	Yes	Many	Close	1,3,5,6,7
X31	Few	Low	No	None	Close	1—2,4,6
X32	Few	Low	No	Few	Close	1—2,4,6
X33	Few	Low	No	Many	Close	1—2,5,6,7
X34	Few	Low	Yes	None	Close	1—2,3,4,6
X35	Few	Low	Yes	Few	Close	1—2,3,5,6,7
X36	Few	Low	Yes	Many	Close	1—2,3,5,6,7
X37	Many	High	No	None	Close	1+2,4,6
X38	Many	High	No	Few	Close	1+2,4,6
X39	Many	High	No	Many	Close	1+2,5,6,7
X40	Many	High	Yes	None	Close	1+2,3,4,6
X41	Many	High	Yes	Few	Close	1+2,3,4,6
X42	Many	High	Yes	Many	Close	1+2,3,5,6,7
X43	Many	Low	No	None	Close	1+2,4,6
X44	Many	Low	No	Few	Close	1+2,4,6
X45	Many	Low	No	Many	Close	1+2,5,6,7
X46	Many	Low	Yes	None	Close	1+2,3,4,6
X47	Many	Low	Yes	Few	Close	1+2,3,5,6,7
X48	Many	Low	Yes	Many	Close	1+2,3,5,6,7

Table 4.1: Training data for map-adaptive game AI.

In Table 4.1, the training data used for map-adaptive game AI is given. Training data concerns the features of the map of the Spring game, and constitutes the entire instance space. In the first column, the labels of the training examples are listed, X1 to X48. Then the value of each feature is listed in column two through column six. Column seven lists the actions to be taken by the game AI. The legend of the table is explained earlier and repeated here. “RN” means number of metal resources. “RD” means resource density. “NR” means

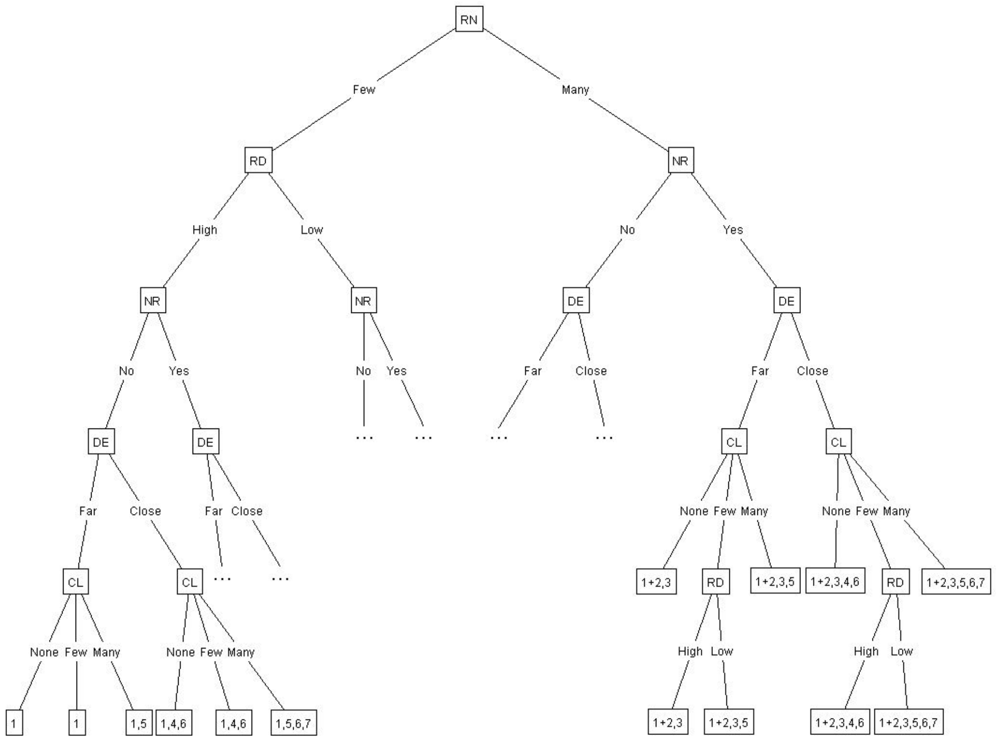


Figure 4.4: The automatically constructed decision tree.

presence of relatively narrow roads (e.g., due to obstacles such as mountains and rivers). "CL" means number of cliffs. "DE" means distance between base locations. "–" means first executing the action before the dash sign, then the action after the dash sign. "+" means executing the actions on both sides of the plus sign simultaneously.

In Figure 4.4, an exemplar portion of the constructed decision tree is displayed; it is constructed on the basis of the training data. The legend of the figure is equal to that of Table 4.1: "RN" means number of metal resources, etc. Below, we give two observations on the constructed decision tree.

First, the feature 'Number of metal resources' is placed at the root of the tree. This implies that the number of metal resources is the most important feature when analysing a map. This observation fits our expectation, since metal resources are vital to expanding the base and constructing the first units.

Second, if the number of metal resources is low, the constructed decision tree considers the 'Resource density' as the next-most important feature of the map. In contrast, if the number of metal resources is high, the resource density is the least important feature. This observation fits our expectation, since the use of the feature 'Resource density' is correlated to the number of metal resources.

B. Experimental setup

To test our approach to map adaptation experimentally, a map-adaptive game AI is pitted in the Spring game against the same game AI without map-adaptive capability. In the experiment, we measure the performance of the map-adaptive game AI, which is expressed by its ability to win when pitted against the game AI without map-adaptive capability. We used a game AI which is open source, which the author of the game AI labelled 'Alexander AI' ('AAI') (Seizinger, 2006). We enhanced the game AI with the capability to adapt its behaviour dependent on the features of the map. For map adaptation, the map-adaptive game AI utilises the constructed decision tree.

Next to pitting the map-adaptive game AI against the same game AI without map-adaptive capability, we also pit the map-adaptive game AI against a human-controlled opponent. The human-controlled opponent will generally play at an expert level. Thereby, as the constructed decision tree contains some nodes that focus on opponent behaviour on a certain map, its purpose is to incite different adaptations from the map-adaptive game AI.

We performed one experiment that consists of five experimental trials, i.e., one trial for each of the five maps used. Each experimental trial to test the map-adaptive game AI was repeated five times. An overview of the used maps is given in Figure 4.5. A description of each map is given below.

Map 1 - Speed Ball: The Speed Ball map has many metal resources, and the players are always located relatively close to each other. This allows us to determine whether the map-adaptive game AI attempts to protect its own units. In the depicted black area of the map nothing can be constructed. When two AI players compete against each other, they may easily be triggered to attack each other early in the game. It is therefore expected that different AI behaviour will be observed when a relatively idle human player is involved as the second player.

Gameplay on this map is focussed on responding effectively to the following features of the map: 'Number of Resources', 'Resource Density', and 'Distance between base locations'. Action '1+2,4,6' is expected to be executed, which corresponds to example X37 from Table 4.1.

Map 2 - Speed Ball Ring 8-way: The Speed Ball Ring 8-way map has many similarities with the Speed Ball map. However, instead of one big circle, this map is divided into eight equally sized circles, and one larger circle in the middle. All the circles are interconnected. The pathways that connect the circles are not very wide, which implies that they are considered as 'Narrow Roads'. There is a relatively large number of metal resources available on this map. Players are positioned randomly at the beginning of the game, thus the distance between the players may vary.

Gameplay on this map is focussed on all of the features other than 'Number of cliffs'. It is expected that the actions '1+2,3' or '1+2,3,4,6' will be executed, corresponding to the examples X16 and X40 from Table 4.1, respectively.

Map 3 - Mountain Range: The Mountain Range map does not have many metal resources. Additionally, the mountains are obstacles that obstruct the roads, making navigation

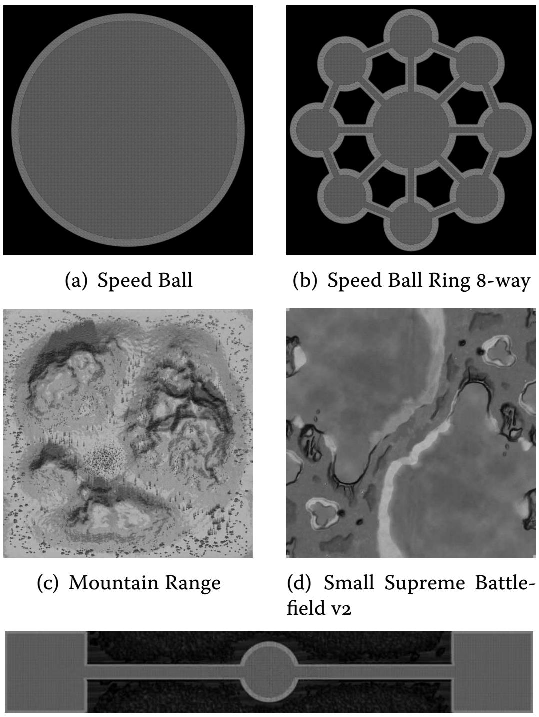

(a) Speed Ball (b) Speed Ball Ring 8-way (c) Mountain Range (d) Small Supreme Battlefield v2 (e) No Metal Speed Metal

Figure 4.5: The five maps that have been used in the experiment.

relatively difficult. Each location for constructing a base on this map has its own advantages and disadvantages. The use of metal storages is recommended on this map, because of diminishing metal resources.

The map is relatively large. Therefore, the distance between the base of two players will typically remain large in a one-versus-one battle, as is the case here. Gameplay on this map is focussed on the features 'Number of Metal Resources', 'Resource Density', and 'Narrow Roads'. We expect that the game AI will execute action '1,3', corresponding to example X4 from Table 4.1.

Map 4 - Small Supreme Battlefield v2: The Small Supreme Battlefield v2 map contains a long bottleneck in the centre of the map. On each side of the map there is a large area of water. On the other two sides of the map there are small areas of land. The map has relatively few metal resources, some of which are available in the water areas.

We expect that the game AI will execute action '1—2,3', corresponding to example X10, or '1—2,3,4,6', corresponding to example X34 from Table 4.1.

Map 5 - No Metal Speed Metal: The No Metal Speed Metal map is based on the Speed Metal map. However, as the name implies, this particular map does not contain any metal resources. The lack of metal resources makes it difficult to quickly produce units and expand the base. A challenge for the trained map-adaptive game AI is that it is not provided with training examples where no metal resources are present on the map.

We expect that the game AI will choose action '1—2,3', corresponding to example X10, or '1—2,3,5', corresponding to example X11 and X12 from Table 4.1. Actions '1—2,3,4,6', corresponding to example X34, and '1—2,3,5,6,7', corresponding to examples X35 and X36, are alternative possibilities.

In addition to pitting the map-adaptive game AI against a computer opponent, the AI was also tested in play against a human opponent (i.e., the experimenter). Our expectation was that different behaviour will be exhibited by the map-adaptive game AI when pitting it against a human opponent that is following a superior game strategy, as compared to when the map-adaptive game AI is pitted against a computer opponent.

C. Results

In Table 4.2, an overview of the experimental results is given. The results reveal that in all of the trials, the map is classified as expected. When in competition against a computer-controlled opponent, the map-adaptive game strategies that emerge from the given classifications lead to superior behaviour in four of the five trials (two times a 4:1 win:loss, and two times a 3:2 win:loss), and to inferior behaviour in one of the five trials (a 0:5 win:loss). When in competition against a human-controlled opponent, the map-adaptive game AI was outperformed in all of the five trials. However, we observed that the map-adaptive game AI responded to strong play of the human player by adapting its own strategy. So, we have to improve this adaptive behaviour.

Below we discuss in detail the behaviour observed of the map-adaptive game AI for each of the five maps.

Map 1 - Speed Ball: We observed that when the map-adaptive game AI played against the computer-controlled player, the players did not focus on gathering resources and building an army. Players became offensive early in the game and continued battling while constructing buildings and units (usually infantry and artillery), which assisted them in defeating their opponent. This occurred in each of the five times that they played on this map.

When a human player was involved, the map-adaptive game AI behaved differently. One time the human player played offensively, and seemed to be defeating the game AI. However, the game AI was able to counter this behaviour by also playing offensively. Yet, the result was a win for the human player. A second time the human player stayed in the background and remained relatively idle for a long period of time, only taking actions in order to defend himself. The result was that the game AI focussed to a larger extent on gathering resources before actually attacking its opponent. Also here, the human player won. This outcome (the human player wins) was repeated three more times.

	Computer	Human
Map 1 - Speed Ball		
Classification	X37 (100%)	X37 (100%)
Win:Loss	3:2	0:5
Map 2 - Speed Ball Ring 8-way		
Classifications	X16 (80%)	X16 (60%)
	X40 (20%)	X40 (40%)
Win:Loss	4:1	0:5
Map 3 - Mountain Range		
Classification	X4 (100%)	X4 (100%)
Win:Loss	4:1	0:5
Map 4 - Small Supreme Battlefield v2		
Classification(s)	X10 (100%)	X10 (60%)
		X34 (40%)
Win:Loss	3:2	0:5
Map 5 - No Metal Speed Metal		
Classification(s)	X10 (100%)	X10 (80%)
		X34 (20%)
Win:Loss	0:5	0:5

Table 4.2: Classifications of the map-adaptive game AI in competition against the computer-controlled player, and against the human player.

Map 2 - Speed Ball Ring 8-way: When the map-adaptive game AI played against another computer-controlled player, the game AI often performed action '1+2,3', as expected. There was only one occurrence of action '1+2,3,4,6' being performed, when the game AI was in the middle circle and decided to place offensive units at the entrance of the pathways connecting to the other circles, while the other player was on one of the outer circles.

The map-adaptive game AI fared well against the human player with the same actions as expected, but again the human player had to remain inactive for the most part in order to provoke the game AI to exhibit behaviour that was expected of it. Otherwise, more offensive subroutines of the game AI would take over and battles occurred early in the game. There were two occasions where the game AI was located in the middle of the map, resulting in a different action, namely '1+2,3,4,6'. The human player won all five encounters.

Because starting positions of both players were random, classifications by the decision tree were also different at each time that a new game was started. This is caused, in particular, by the distance between both players. It explains the large difference in classifications.

Map 3 - Mountain Range: We observed that on this map, the map-adaptive game AI was strongly focussed on gathering nearby metal resources early in the game. Additionally, the game AI was blocking the passageways between the mountains, and, if applicable, between the edges of the map and the mountains. The game AI protected the expanded position, and launched an attack on the opponent when it constructed a relatively large army. The described strategy was consistently observed against both the computer-controlled opponent, as well as against the human opponent. However, the outcome was completely different (4:1 for the map-adaptive game AI against the computer-controlled player, and only losses against the human player).

We observed that early in the game, relatively few attacks would take place. The phenomenon results from the relatively large distance between the base of each player. As lower-level subroutines will not be called for handling an attack, the map-adaptive game AI can focus on its first priority: establishing and expanding a strong base. Thus, we observed that in all cases action '1,3' was executed, as expected.

Map 4 - Small Supreme Battlefield v2: When in competition against the computer-controlled opponent, the map-adaptive game AI had a preference for executing action '1—2,3'. In most of the cases the game AI blocked the map's bottleneck with offensive units. In other cases the focus of the game was not so much on the bottleneck, but more on the water area, on which the battle continued by sea units and submarine units. The computer-controlled player always built its base on land located in the corner of the map, which implied that the distance between the bases remained fairly large. All in all, the map-adaptive game AI won by 3:2.

Identical behaviour by the map-adaptive game AI was observed in the competition against a human player. However, the outcome was entirely different, since the human player won by 5:0. On one occasion the human player constructed the base nearby the base of the map-adaptive game AI. This led the AI to increase the protection of its base and metal extractors, by using offensive units.

Map 5 - No Metal Speed Metal: In competition against the computer-controlled game AI, both players focussed on constructing offensive units to oppose their opponent. In addition, the map-adaptive game AI utilised units for protecting the entrance of its own area. Apparently, it was not the correct strategy since the map-adaptive game AI lost all five games. Similar behaviour was observed when competing against the human player. In one case, the human player constructed the base in the centre of the map, near the map-adaptive game AI. This led to a different classification, and thus different behaviour from the map-adaptive game AI. Yet, the outcome of the experimental trial was the same. The human player won by 5:0.

Though the map-adaptive game AI was not trained for circumstances where no metal resources are present, it was able to utilise the learned decision tree by traversing the node for 'few metal resources'. However, it did not exhibit behaviour suitable for defeating the opponent players. This indicates possible directions of improvements in the provided domain knowledge.

4.2.3 Discussion of the results

This subsection provides a discussion of the experimental results. We first discuss (A) how well the results generalise to other games. Subsequently, we draw (B) experimental conclusions of the present research.

A. Generalisation to other games

In our approach to map adaptation, we implemented map-adaptive game AI as a high-level planner of strategic actions. We allowed low-level actions, such as handling an imminent threat of the opponent, to interfere with the established high-level plan.

In a typical RTS game, early phases of the game are focussed on planning the construction and expansion of the base. Later phases of the game are typically focussed on engaging in offensive or defensive actions. However, if an opponent would decide to attack relatively early, a player would be forced to abandon the established plan and focus on combat. Therefore, our implementation of map-adaptive game AI as a high-level planner of strategic actions, may be considered suitable for RTS games.

The approach to map adaptation can be generalisised to other games that require a form of high-level planning dependent on features of the map. Of course, game developers should consider that to apply our approach in practice, in each game environment a specific balance should be established between pursuing a high-level map-adaptive plan, and allowing the game AI to respond to low-level actions.

B. Experimental conclusions

The results of the experiment with map adaptation show that against a computer-controlled opponent, the map-adaptive game AI generally constructed effective game strategies. However, the map-adaptive game AI was outperformed by a human opponent. Still, we observed that the map-adaptive game AI responded to the strong play by the human player by adapting its own strategy. From these results, we may conclude that the proposed approach to map adaptation can be used to automatically construct effective strategies dependent on the map of a game.

Obviously, the behaviour expressed by the map-adaptive game AI is effected heavily by the quality of the domain knowledge used. In our implementation of map-adaptive game AI, domain knowledge was established offline, based on the experimenter's experience with the game. Ideally, domain knowledge should be established fully automatically, based on observations of the game AI. This will be the focus of the next section.

4.3 Case-based adaptive game AI in a simulated game

In this section, we report on two experiments to test case-based adaptive game AI in a limited, simulated video-game environment. We investigate to what extent a game character can gather domain knowledge from a few observations, and immediately (i.e., without trials and without resource-intensive learning) exploit the domain knowledge to create effective behaviour. We compare three approaches to exploit domain knowledge, namely (1) 1-nearest neighbour (1-NN) classification, (2) weighted 1-nearest neighbour (weighted 1-NN) classification, and (3) k-nearest neighbour (k-NN) classification.

The remainder of the section is organised as follows. We first discuss the simulated video game that we used for our experiments (4.3.1). Subsequently, we discuss how we gather domain knowledge in a case base (4.3.2). Next, three approaches for exploiting the domain knowledge are described (4.3.3). Then, we report on the two experiments to test the performance of the three approaches (4.3.4). The section is concluded by a discussion of the obtained results (4.3.5).

4.3.1 Simulated video game

For our investigation we have created a simulator environment representing a simple video game. The game is an obstacle course, in which a non-player game character (NPC) has to travel from the bottom of a grid to the top of the grid. As NPCs can only obtain a good game result if they are able to discover and exploit properties of the game, it poses an interesting environment for our investigation.

A game grid is 12 cells wide, and 100 cells high. Each cell of the grid can contain an object. The following six object types can be located on the grid.

1. **NPC.** One NPC is located in the grid. It is the only object that can change its position. Initially the NPC is placed in a random empty location on the bottom row of the grid (row 1). To manoeuvre to its goal (i.e., row 100 of the grid) it has three actions available: (1) to move to the cell directly to its upper left, (2) to move to the cell directly above it, and (3) to move to the cell directly to its upper right. As input, the NPC can observe all cells of the five rows above it. It has two properties: health and fitness. Initially it is provided with a health value of 100. The NPC 'dies' (i.e., is removed from the grid) when its health reaches zero. The fitness value of the NPC is determined when it either dies or reaches the top row of the grid. It is calculated as $S/(H-1)$, where S is the number of steps taken by the NPC, and H is the height of the grid. On the screen, the NPC object is visually represented by a light-blue colour.

2. **Wall.** All the leftmost and rightmost cells of the grid contain wall objects. When an NPC tries to enter a cell in which a wall object is located, it dies. On the screen, the wall object is visually represented by a black colour.

3. **Goal.** All the cells on the top row of the grid (except for those containing a wall object) contain goal objects. When an NPC enters a cell in which a goal object is located, its fitness is determined and the NPC is removed from the grid. On the screen, a goal object is visually represented by a grey colour.

4. **Tree.** An arbitrary number of trees can be located in the grid. Trees are treated as walls, i.e., when an NPC tries to enter a cell in which a tree is located, it dies. On the screen, a tree object is visually represented by a green colour.

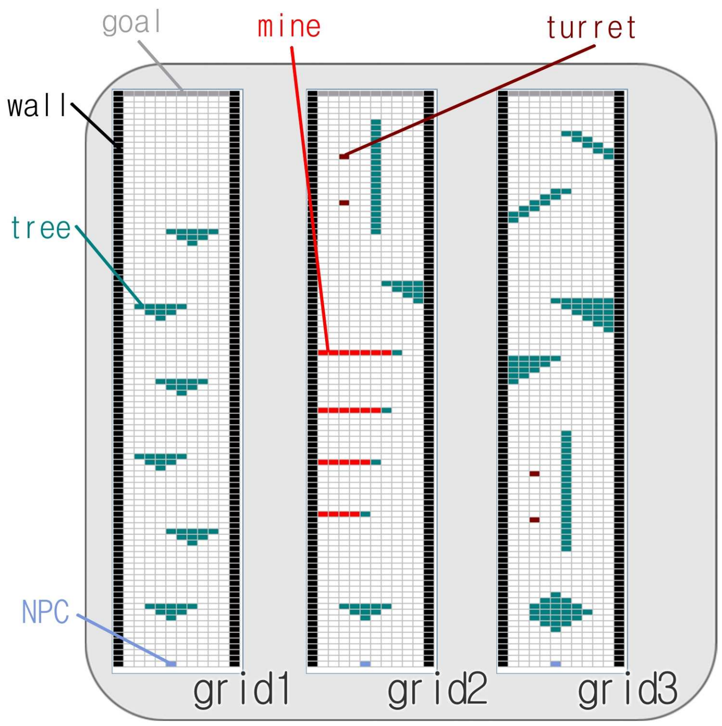


Figure 4.6: Three grids designed for our experiments. Note that mines are not visible to the NPC.

5. **Turret.** An arbitrary number of turrets can be located in the grid. When an NPC is within reach of a turret (defined as within a square of seven cells on each side and the turret at its centre), its health is reduced by 10. Furthermore, when the NPC enters a cell where a turret is located, it dies. On the screen, a turret object is visually represented by a brown colour.

6. **Mine.** An arbitrary number of mines can be located in the grid. NPCs *cannot* detect mines. NPCs are allowed to move into a cell containing a mine. However, when that happens the mine 'explodes', reducing the NPC's health by 33. On the screen, a mine object is visually represented by a red colour.

Only an NPC can co-exist with another object in a cell. Of all the other objects, each cell can contain at most one. The three grids designed for our experiments are displayed in Figure 4.6.

4.3.2 Gathering domain knowledge

To gather, and later exploit, domain knowledge of the simulated video game, we draw inspiration from the case-based reasoning methodology (see Subsection 4.1.1). In video games,

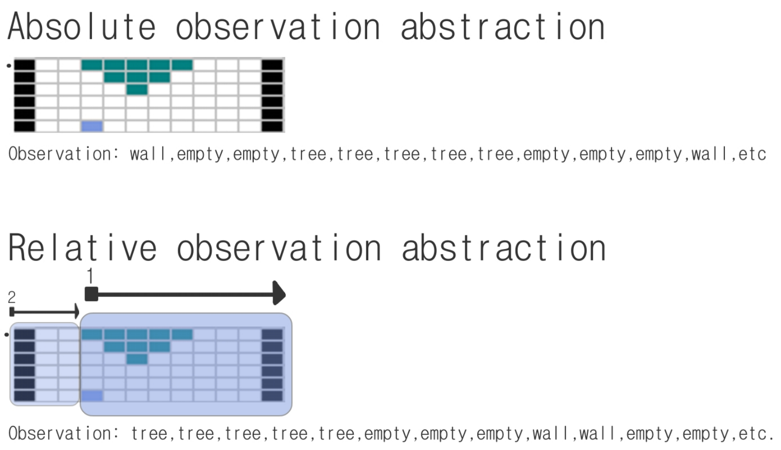

Absolute observation abstraction
Observation: wall,empty,empty,tree,tree,tree,tree,tree,empty,empty,empty,wall,etc
Relative observation abstraction
1
2
Observation: tree,tree,tree,tree,tree,empty,empty,empty,wall,wall,empty,empty,etc.

Figure 4.7: Example of an absolute and relative observation abstraction. In both abstractions, first the top-row cells are observed from left to right. Subsequently, the rows below are observed analogously. The absolute abstraction commences at the top-left cell of the observed part of the grid. The relative abstraction commences at the top cell located above the current position of the NPC. In our experiment we used the relative observation abstraction.

the environments in which NPCs are situated typically excel in visual richness and state and action-space complexity. This poses a challenge for a case-based reasoning approach. We therefore propose that environment information which is gathered by the NPC should, for effective and rapid use, be (1) represented in such a way that stored cases can be reused for previously unconsidered situations, and (2) compactly stored in terms of the amount of gathered observational data.

In our simulation, we wish to store the observations of an NPC, with the action it took, and an assessment of the success of that action. Regarding the representation we decided to store observations in the form of labelled state-action pairs. Each case consists of an abstraction of the world visible to the NPC, together with a description of the NPC's action. The environment state is abstracted relative to the NPC position (rather than from an absolute viewpoint). This abstraction is illustrated in Figure 4.7. In addition, each state-action pair is labelled (and relabelled during a simulation trial) with the average observed fitness to which the stored action has led when it were applied. The labelling process is algorithmically described in Figure 4.8.

Obtaining a compact storage of cases is an important topic in case-based reasoning research. The main goal is to obtain a compact case base (i.e., with a reduced number of cases) but without losing problem-solving accuracy (Lopez de Mantaras *et al.*, 2005; Craw *et al.*, 2007; Cummins and Bridge, 2009). A common strategy to achieve this goal, is not incorporating cases that do not contribute to performance in a positive sense; thereby essentially limiting the size of the case base (Smyth and Keane, 1995). Our implementation of this strategy is to store only those state-action pairs that did not directly lead to a death in the next

```
1.  To determine the fitness value, observe the NPC until it reaches a goal
    cell or dies.
2.  Retrieve NPC observations for every step performed.
3.  Check for each retrieved observation whether it is already stored.
4a. If so, update the fitness value of the retrieved observation by averaging
    over the fitness values.
4b. If not, store the particular observation with the action performed and the
    fitness obtained by the NPC.
```

Figure 4.8: Labelling algorithm applied when gathering domain knowledge.

observable state. Our motivation for this strategy is that accordingly, a labelled case-base arises of which all state-action pairs lead either to a local optimum, or to a global optimum.

4.3.3 Exploiting domain knowledge

We propose three approaches for exploiting domain knowledge gathered in the simulated video game. The approaches are based on the *reuse* process-step of the case-based reasoning methodology, in which solutions from previously stored cases are mapped to a new case.

Our case base contains the observations of NPCs traversing grids. To exploit the case base for a new case, similarity values are calculated between the vector of observations of the new case, and the vectors of observations of all the cases in the case base. Of the most similar cases, the particular case with the highest fitness value is retrieved, and the action which was performed by the NPC for that case is selected for the new case.

The three proposed approaches vary in their similarity-calculation algorithm. They are: (A) 1-NN classification, (B) weighted 1-NN classification, and (C) k-NN classification. The 1-NN approach was chosen as a way of baseline comparison. The weighted 1-NN and the k-NN classification approaches were chosen under the presumption of improved performance with regard to feature weighting, and an increase in generalisation power, respectively. The three approaches for exploiting domain knowledge are described as follows.

A. 1-NN classification

For 1-NN classification, similarity is calculated by a matching in which every identical observed feature receives the same weight. The features of all three classification approaches concern the cells of the five rows above the NPC (one feature per cell), which is abstracted using a relative observation abstraction (as illustrated in Figure 4.7). In our implementation of 1-NN classification, the algorithm scales the similarity value to one hundred per cent when all observed features are identical.

When similarity values are calculated for each case in the case base, the matching algorithm subsequently retrieves the case(s) with the highest similarity value within a *similarity window*. The similarity window defines the threshold similarity value of cases to be denoted as 'similar'. Should multiple cases be retrieved, the first case with the highest fitness value is selected for execution. Should no cases be retrieved, the process is iterated by adjusting

```
function getSimilarity(c,observation,F): real;
begin
 similarity := 0;
 for (f ∈ F) do
  if (observation.f = c.f)
   then similarity := similarity + observation.f.Weight;
 result := similarity / |F|;
end;

function getMostSimilarCase(C): c;
begin
case_selected := null;
fitness_best := 0;
similarity_window := 100;
similarity_windowstepsize := 1;
repeat
 begin
  for (c ∈ C) do
   begin
    similarity_value := getSimilarity(c,currentObservation,F);
    if (similarity >= similarity_window)
     then
      if (c.fitness > fitness_best)
       then
        begin
         case_selected := c;
         fitness_best := c.fitness;
        end
   end;
  if not(case_selected)
   then
    begin
     similarity_window := similarity_window − similarity_windowstepsize;
     fitness_best := 0;
    end;
 end;
until (case_selected);
result := case_selected;
end;
```

Figure 4.9: Pseudo-code for the 1-NN classification process.

the similarity window so that it allows for less similar cases to be retrieved. This process is illustrated in pseudo-code in Figure 4.9.

We note that algorithms that operate with similarity windows are inefficient when applied only for retrieving the case with highest fitness. However, their structure allows for a different selection of cases, that depends on the similarity, and not the fitness of cases.

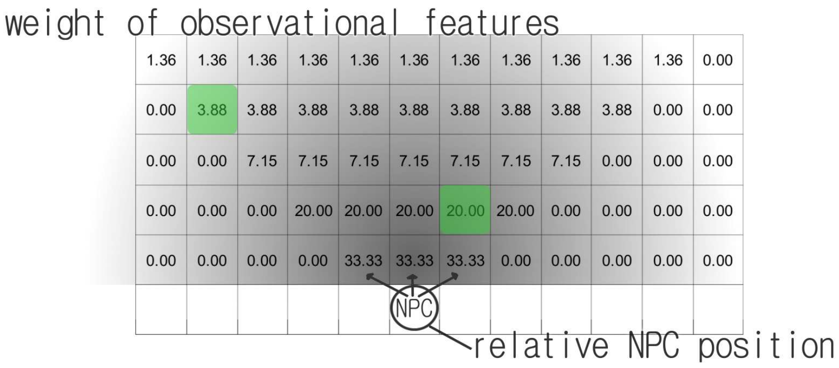


Figure 4.10: Weighted similarity matching is established by weighting each observational feature with respect to the relative NPC position. The portrayed situation displays that part of the grid observed by the NPC. The highlighted object on the top-left receives an example weight of 3.88, and the highlighted object in the bottom-middle receives a weight of 20.00. Allowed actions of the NPC are denoted by arrows.

B. Weighted 1-NN classification

For weighted 1-NN classification, similarity is calculated by a matching in which features closer to the NPC are weighed heavier than features further from the NPC. In our implementation, environmental observations are evaluated with consideration of the relevance to the NPC. For instance, an observational feature directly above an NPC is assigned a larger weight than a feature that is located two rows above the NPC. The weights assigned to observational features are shown in Figure 4.10. The weights are determined by the experimenter, on the basis of domain knowledge of the game environment. For the situation portrayed in the figure, the observational feature on the left (the darker cell) will receive a relatively low weight as it is relatively far from the NPC.

The similarity matching and case retrieval algorithm follow the same procedure as that of the 1-NN classification approach.

C. k-NN classification

For k-NN classification (Fix and Hodges, 1951), similarity is calculated as a generalisation over stored cases. A straightforward k-NN algorithm is used to classify a new case based on the majority of the k-NN category. This process consists of two steps.

1. Given a new case, the k-NN algorithm retrieves the k number of state-action pairs of which the states are most similar to the new case (i.e., with highest similarity as calculated by the 1-NN classification approach). If state-action pairs exist with an identical state-description, but with a different observed action, only the state-action pair with the highest fitness is retrieved.

2. Action classification is established by a majority vote among the actions of the retrieved state-action pairs.

4.3.4 Experiments with exploiting domain knowledge

This subsection reports on two experiments that test our implementation of case-based adaptive game AI in the simulated video game. We first discuss (A) the experimental setup. Subsequently, we present and discuss (B) the obtained results.

A. Experimental setup

Our first experiment consists of a comparative study of the three approaches proposed for exploiting domain knowledge. Our goal is to detect whether the approaches are able to evoke successful NPC behaviour without feeding prior knowledge to the NPCs. The experimental procedure is as follows. Given a particular grid, we sample data of NPCs attempting to reach the goal objects by choosing only random moves. We perform separate trials with game observations stored of 2, 10, and 50 subsequent randomly behaving NPCs. Subsequently, we let an NPC use each of the three approaches to exploit domain knowledge, and we determine the fitness achieved. We consider that a different random behaviour leads to different observations (and thus different domain knowledge gathered). Therefore, for each grid and each approach, the results are averaged over 30 repetitions of the experimental trial. Regarding the k-NN classification algorithm, the value of k was chosen by assessing five different values for each trial, and selecting the best performing value. The tested k values were 3, 5, 10, 25, and 50.

In our second experiment, we test how each of the three approaches would generalise to different environments (i.e., test whether knowledge learned in one environment can be transferred to another environment). We created a case base for grid3 by running 50 randomly-behaving NPCs. Subsequently, we applied the three approaches to control an NPC that used the established case base on the other two grids (grid1 and grid2). We chose grid3 for gathering knowledge, as it is the most difficult grid for an NPC to traverse. For the second experiment, we repeated the process 100 times for each of the grids, and for each of the approaches.

B. Results

In Table 4.3 an overview of the experimental results is given. The first column of the table lists on which grid the experimental trial is conducted. The second column lists the number of games of which observational data is exploited for the experimental trial. The third to fifth column lists the performance obtained by 1-NN classification, weighted 1-NN classification, and k-NN classification, respectively. The sixth column lists the best performing setting of k that was used by the k-NN classification approach.

Experimental results of the first experiment reveal that, as may be expected, all proposed knowledge exploitation approaches endow an NPC with more effective behaviour whenever an increased amount of domain knowledge is available to the exploitation process. In addition, when training and testing on the same grid, both the 1-NN classification

	# Obs.	1-NN	Weighted 1-NN	k-NN	
Grid1	2	0.23 (0.27)	0.26 (0.31)	0.17 (0.20)	*k=3*
(30 repetitions)	10	0.55 (0.40)	0.60 (0.39)	0.16 (0.17)	*k=5*
	50	0.66 (0.33)	0.70 (0.34)	0.35 (0.37)	*k=3*
Grid2	2	0.25 (0.21)	0.24 (0.21)	0.18 (0.21)	*k=10*
(30 repetitions)	10	0.50 (0.28)	0.48 (0.29)	0.27 (0.24)	*k=3*
	50	0.55 (0.24)	0.55 (0.22)	0.35 (0.26)	*k=3*
Grid3	2	0.18 (0.17)	0.18 (0.17)	0.12 (0.13)	*k=3*
(30 repetitions)	10	0.38 (0.21)	0.41 (0.26)	0.20 (0.21)	*k=10*
	50	0.60 (0.19)	0.61 (0.24)	0.37 (0.26)	*k=25*
Knowledge transfer (Grid 3 to 1, 100 rep.)	50	0.33 (0.29)	0.40 (0.34)	0.33 (0.34)	*k=10*
Knowledge transfer (Grid 3 to 2, 100 rep.)	50	0.43 (0.25)	0.46 (0.27)	0.38 (0.26)	*k=25*

Table 4.3: Comparison of the performance of the three approaches. The performance is expressed in terms of the mean fitness and the standard deviation, denoted as 'mean (standard deviation)'.

and weighted 1-NN classification approach outperform the k-NN classification approach significantly with ten or more observations (cf. t-test, Cohen, 1995). The performance of the 1-NN classification and weighted 1-NN classification approach do not differ substantially in this setting.

Experimental results of the second experiment, reveal that when training and testing on different grids for the purpose of generalising over domain knowledge, the performances of the 1-NN classification and weighted 1-NN classification approach differ. When transferring domain knowledge from grid3 to grid1 and grid2, the weighted 1-NN classification approach outperforms the 1-NN classification approach with statistical reliabilities of 94% and 79%, respectively (cf. t-test, Cohen, 1995).

Note that the k-NN classification approach is the only approach where the performance does not degrade substantially when transferring domain knowledge. This is an important issue with regard to generalisation. However, despite the obtained non-degrading performance of the k-NN classification approach, in absolute terms it performs worst of all three approaches. Namely, for knowledge transfer from grid3, to grid1 and to grid2, it is outperformed by the 1-NN classification approach with statistical reliabilities of 51% and 92%, respectively (cf. t-test, Cohen, 1995).

4.3.5 Discussion of the results

This subsection provides a discussion of the experimental results. We first discuss (A) which behaviour was learned by the NPCs. Subsequently, we draw (B) experimental conclusions of the present research.

A. Generalisation of the learned behaviour

In our experiments, we tested our implementation of case-based adaptive game AI in a simulated video game. The observed ability to exploit effectively domain knowledge (that was gathered on grid3) on other grids (grid1 and grid2), indicated the capability to generalise over stored cases. For instance, in both grid3 and grid2 the 'trick' that needed to be learned was to avoid the numerous trees at the start, and subsequently remain on the right side of the grid. Thus, even though the grids are different, the NPC behaviour to be expected should be similar in particular circumstances. In the experimental results we noticed that the k-NN classification approach is the only approach where the obtained performance does not degrade when transferring gathered domain knowledge to another domain, compared to when gathered domain knowledge is exploited in the same domain. The capability that the k-NN classification approach showed for generalising over stored cases, is invaluable for adaptive game AI in actual video games, since it would allow a means of learning relatively reliably from only a few trials.

To this extent, in our view it is recommendable to apply previously established effective behaviour in known circumstances. If in unknown circumstances such predictably effective behaviour is not available, game AI should generalise over previous observations that are stored in the case-base.

B. Experimental conclusions

From the results of the experiments that tested the proposed approaches to exploit domain knowledge, we may draw the conclusion that the approach to weighted 1-NN classification performs best in the simulated video game. Our findings on the approaches' capabilities to transfer domain knowledge showed that the approach to weighted 1-NN classification performed best, but applying the k-NN classification approach resulted in the most non-degrading performance. We indicated the importance of generalising over stored cases.

In the next section, we discuss how to extend our findings to actual video games. In our discussion, we will focus particularly on how the practical applicability of case-based adaptive game AI may be demonstrated.

4.4 Towards case-based adaptation in video games

In this section we aim at understanding how to implement case-based adaptive game AI in an actual video game. So far, we performed two experiments to obtain an early indication of the effectiveness of case-based adaptive game AI.

In the first experiment (Section 4.2), we implemented a limited form of case-based adaptive game AI in the Spring game. The experiment aimed at establishing map-adaptive game strategies. The experimental results revealed that map-adaptive game strategies may be established automatically by exploiting domain knowledge effectively. However, the domain knowledge used in our experiments was generated manually by the experimenter. This indicates that effective adaptive game AI can be established if domain knowledge of high quality is available. In our preferred situation, domain knowledge is generated fully automatically, based on observations of the game AI.

In the second experiment (Section 4.3), we implemented case-based adaptive game AI in a limited, simulated video-game environment. The experiment performed in the simulated video game concerned the automatic gathering and exploitation of domain knowledge. The experimental results revealed that domain knowledge may be gathered automatically, and may be exploited immediately (i.e., without trials and without resource-intensive learning) to create effective behaviour. Though the investigated environment concerned a simulated video game, and not an actual, complex video game, it indicated that effective adaptive game AI may be established automatically, following the proposed approach to case-based adaptive game AI.

Of course, the question remains which components in particular are required to implement case-based adaptive game AI in an actual, complex video game, such as Spring. As briefly discussed in Section 4.1, in our view three main components are required for case-based adaptive game AI. These three components are introduced below. In the three chapters that follow, each component is discussed in detail.

First, in the discussed simulated video game it is an easy task to evaluate the quality of behaviour expressed by the game AI. In the complex Spring game, it is a difficult task to establish such evaluations. In Chapter 5, we discuss how a function to evaluate the behaviour expressed by game AI can be established. Such a function is called an *evaluation function*.

Second, adaptation to game circumstances requires a mechanism that exploits game observations effectively and rapidly. The mechanism needs to be capable of exploiting game observations that have been gathered from a multitude of games, together with observations of the current game. This is a challenging task. In Chapter 6, we discuss how a mechanism to adapt AI to circumstances of the game can be established. Such a mechanism is called an *adaptation mechanism*.

Third, the challenge of adapting AI in video games effectively, stems for a large part from the fact that effective adaptation needs to depend on the strategy of the opponent player. In a typical video game, the inherent randomness and imperfect information of the game environment renders establishing models of the opponent player a difficult task. In Chapter 7, we discuss how to establish and exploit models of the opponent player. This is called *opponent modelling*.

4.5 Chapter summary

In this chapter we proposed an alternative, novel approach to create adaptive game AI, i.e., case-based adaptive game AI. We performed two experiments to obtain an early indication of the effectiveness of case-based adaptive game AI. In the first experiment, effective map-adaptive game strategies were established by a limited form of case-based adaptive game AI. In the second experiment, effective behaviour was established by case-based adaptive game AI in a limited, simulated video-game environment. The results of these two experiments indicated that effective AI in an actual video game may indeed be established by following the approach to case-based adaptive game AI.

We discussed that for case-based adaptive game AI to be successful in an actual video game, such as the complex Spring game, three main components are required. These three components are (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. Each component is investigated in detail in the chapters that follow.

5 The evaluation function

In this chapter we concentrate on the evaluation function, one of the three main components of case-based adaptive game AI. To be precise, we investigate a static heuristic evaluation function (cf. Pearl, 1984). In the present research, we use the evaluation function as a predictor of the final outcome of a game. The evaluation function drives the behaviour that is exhibited by case-based adaptive game AI. Obviously, the accuracy of the function plays a major role in the effectiveness of the exhibited behaviour. We will show that in complex video games, it is possible to establish an evaluation function that rates accurately the state of the game.

Chapter 5 is organised as follows. We first describe the concept of evaluations functions (Section 5.1). Subsequently, we outline our evaluation function for an actual, complex RTS game (Section 5.2). Next, we report on the validation of the evaluation function (Section 5.3). Finally, we present the chapter conclusions (Section 5.4).

This chapter is based on the following four publications.

1) Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007a). Automatically evaluating the status of an RTS game. In Van Someren, M., Katrenko, S., and Adriaans, P., editors, *Proceedings of the Annual Belgian-Dutch Machine Learning Conference (Benelearn 2007)*, pages 143–144. University of Amsterdam, Amsterdam, The Netherlands.

2) Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007b). Predicting success in an imperfect-information game. In Van den Herik, H. J., Uiterwijk, J. W. H. M., Winands, M. H. M., and Schadd, M. P. D., editors, *Proceedings of the Computer Games Workshop 2007 (CGW 2007)*, MICC Technical Report Series 07-06, pages 219–230. Maastricht University, Maastricht, The Netherlands.

3) Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2007c). Phase-dependent evaluation in RTS games. In Dastani, M. M. and de Jong, E., editors, *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 3–10. Utrecht University, Utrecht, The Netherlands.

4) Bakkes, S. C. J. and Spronck, P. H. M. (2008). Automatically generating a score function for strategy games. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 647–658. Charles River Media, Inc., Hingham, Massachusetts, USA.

5.1 Evaluation functions

We consider an evaluation function to be a function that rates the state of the game. To evaluate the state of the game adequately, one has to model the features that are important to the domain (Gomboc *et al.*, 2005; Billings, 2006; Fürnkranz, 2007). Often, the rating of a game state is expressed by game features that are determined by the experimenter (Fürnkranz, 2007). Evaluation functions are commonly applied in the AI of classic games. In addition, evaluation functions are increasingly applied in the AI of video games. The general goal of an evaluation function applied in both classic as well as video games, is to drive the decision making of the game AI.

In this section we describe how evaluation functions are applied in games. First, to illustrate the concepts that underlie evaluation functions, we outline how evaluation functions are applied in classic games (5.1.1). Second, we outline how evaluation functions are applied in video games (5.1.2). Third, we discuss the general procedure for establishing an evaluation function (5.1.3). Finally, we describe how game knowledge that is gathered can be incorporated into the evaluation function (5.1.4).

5.1.1 Evaluation functions in classic games

As stated earlier in this section, evaluation functions are commonly applied in the AI of classic games, such as in computer programs that play the game of chess (cf., e.g., Van Diepen and Van den Herik, 1987). The general characteristics of evaluation functions applied in classic games, can be described in terms of (1) the game environment in which they are applied, and (2) the goal of evaluation. Both are discussed below.

Game environment: Classic games present an abstract game environment in which game actions typically are performed in a turn-based fashion. Evaluation functions in classic games usually operate in a discrete, perfect-information environment, with little to no inherent randomness.

Goal of evaluation: In classic games, the goal of the evaluation function typically is to arrive at the *optimally reachable* game state. Here, we remark that it is possible to define other goals for evaluation functions. However, such a discussion is beyond the scope of our framework. For detailed reading on this topic we refer to Donkers *et al.* (2003).

In classic games, it is common that search techniques are applied to find possible game actions (i.e., moves to play). Search techniques, in turn, typically incorporate an evaluation function to determine the rating of a subsequent state of the game. In general, search techniques are designed to arrive at a quasi-optimal - instead of an optimal - solution with a significant cost reduction (Pearl, 1984). An evaluation function, therefore, is often designed to be relatively fast, while a certain amount of inaccuracy is permitted.

In many classic games, expert players are able to model the game features that are important to the domain.[1] For instance, in the game of chess an important feature of evaluation is the so-called 'material balance'. The material balance is a measure of which pieces are located on the board for each side, and is typically implemented as a summation of piece values. A common (and rough) assignment of piece values is: Queen equals nine Pawns, Rook equals five Pawns, Bishop and Knight equal three Pawns (Euwe *et al.*, 1982; Bramer, 1983; Beal, 1986). The piece value of a Pawn can be chosen randomly, though often programmers have chosen for a value of 100 or 1000 (Van Diepen and Van den Herik, 1987). As the King cannot be captured or traded during the course of the game, the piece value of the King is commonly undefined, as already proposed by Turing (1953). Alternatively, the King can be assigned an arbitrary high value, such as 200 Pawns (Shannon, 1950), to ensure that a checkmate outweighs all other factors (Levy and Newborn, 1991).

A chess evaluation function that only takes the material balance into account can be denoted as:

$$E = \sum_{p} V_p (P_w - P_b) \tag{5.1}$$

where Vp is the value of a piece of type p, and where P_w and P_b are the number of white or black pieces of that particular type on the board, respectively.

Next to the material balance, additional game features are important in the game of chess (Van Diepen and Van den Herik, 1987). Still, researchers estimate that even a significant advantage in so-called non-material factors, such as position, mobility, and safety is worth less than 1.5 Pawns (Frey, 1984; Atkin and Slate, 1988). Admittedly, a chess evaluation function that only takes the material balance into account is already capable of playing a decent game of chess (Spracklen and Spracklen, 1978). Yet, in an attempt to play better, more sophisticated evaluation functions are desired. In chess, many techniques have been tried to improve the quality of the evaluation function. Here we mention some important publications that give an adequate impression of the development of techniques in the area of evaluation functions in the game of chess: Beal (1984), Schaeffer (1984), Hartmann (1987a,b), Anantharaman (1992), Beal and Smith (1994), and Baxter *et al.* (1998). The techniques are also applicable in the domain of chess variants (Droste and Fürnkranz, 2008).

All in all, we may state that in the domain of classic games establishing an evaluation function that accurately determines the rating of a game state remains a challenging task. The same applies to the domain of video games.

5.1.2 Evaluation functions in video games

Until recently, AI of video games rarely took into account the incorporation of an evaluation function. Nowadays, however, evaluation functions are applied with an increasing pace, to accommodate game developers into their previously discussed desire to establish consistent

[1] Still, this is a difficult and time-consuming process in several games; the most notorious example is the game of Go (Bouzy and Cazenave, 2001). It is one of the reasons why Go programs using the classic tree-search method with a dedicated evaluation function have difficulties with achieving a strong level, despite intensive research and additional use of knowledge-based methods (Chaslot *et al.*, 2008d).

game AI (see Section 2.2). To this end, game developers require the ability to adequately assess the current game circumstances. Such an assessment can be modelled in game AI using an evaluation function.

The general characteristics of evaluation functions applied in video games, can be described in terms of the game environment in which they are applied, and the goal of evaluation, as is done in the classic games. Both are discussed below.

Game environment: Video games present a realistic and complex environment in which game actions typically are performed in a real-time fashion. The AI of video games is expected to exhibit behaviour that is consistent with the presented environment. Often, game AI is expected to exhibit 'human-like' behaviour (Laird and Van Lent, 2001). In these game environments, the determination of an adequate evaluation function can be difficult. This difficulty arises in particular from the fact that evaluation functions for video games usually operate in a continuous, imperfect-information environment with a large amount of inherent randomness. In addition, the evaluation function needs to be capable of modelling the behaviour of a multiplicity of game characters, which may be controlled by more than one friendly player, and by more than one opponent player.

Goal of evaluation: Evaluation functions in video games are typically applied for two goals. A first goal for applying an evaluation function in a video game, is to steer the game AI towards arriving at the most *entertaining* game state (instead of the optimally reachable game state). A second goal for applying an evaluation function in a video game, is to provide feedback on the effectiveness of the playing style of a player of the game. Such feedback can be incorporated to adjust automatically the gameplay of the video game, or can be an integral part of the game itself, such as in serious games and game-driven intelligent tutoring systems.

The general procedure to establish an evaluation function for games is to start by defining features that are important for rating the state of the game. Subsequently, the defined features are combined in a way that their relative importance is quantified. This general procedure is discussed in more detail in the next subsection. In video-game practice, the defined features most frequently are combined linearly, so that the task is to adjust the weights of a weighted sum of features (Fürnkranz, 2007). To this end, game developers typically opt to exploit solely their own expertise with the game environment. However, academics working in video-game environments, often opt for a more knowledge-free approach, and incorporate machine learning techniques to automatically tune the evaluation function.

An example of an automatically tuned evaluation function can be seen in research applied to the game of Tetris. Tetris is a popular video game that was created in 1985 by Alexey Pajitnov, and has been made available for nearly every operating system and hardware platform in existence (Carr, 2005). Though the game is seemingly simple, it has been shown that optimally playing Tetris is NP-complete, and is highly inapproximable (Demaine *et al.*, 2002). Still, by incorporating genetic algorithms to tune a game-specific evaluation function, researchers were able to create a robot that was able to play Tetris better than a human, under normal Tetris-playing conditions (Fahey, 2003; Flom and Robinson, 2004).[2] For more information on evaluation functions for Tetris, we refer the reader to the recent articles by Thiery and Scherrer (2009a,b) that adequately review the state of the art, and also add two more features to an existing evaluation function that played well so far. The features are: (1) hole depth, and (2) the number of rows with holes. Incorporating the seemingly straightforward features resulted in an improvement in playing strength up to becoming the strongest program in the world.

5.1.3 Establishing an evaluation function

The general procedure for establishing an evaluation function consists of two steps. These two steps are: (A) feature definition and selection, and (B) feature weighting. The first step, feature definition and selection, may include methods to define and select features automatically. The second step, feature weighting, may include methods to establish appropriate weights automatically.

A. Feature definition and selection

Determining which features to make available to an evaluation function is regarded as the most difficult problem in artificial intelligence in games (Gomboc *et al.*, 2005). The problem was already mentioned in the early days of game-playing programs (Samuel, 1959). In the domain of classic games, the task of defining and selecting features is generally performed by the researcher, sometimes with the help of the literature or an expert grandmaster (e.g., IGM Boris Alterman was approached as expert for the chess program Junior). Also in the domain of video games, where the game environment is highly complex, the task of defining and selecting features is generally performed by the researcher. For instance, in the game of Tetris, it was shown, as stated above, that the precise selection of two adequate features was sufficient for establishing a top-rated evaluation function (Thiery and Scherrer, 2009a,b).

We note that numerous methods are available for defining and selecting features automatically (cf. Guyon and Elisseeff, 2002). The automated construction of features for game playing has been discussed by Utgoff (2001). Using the game tic-tac-toe as a testbed, Gomboc *et al.* (2005) demonstrated (1) the incremental learning of layers of features, (2) the ability to train particular features in isolation, and (3) tools for the specification and refactoring of features. Steps towards selecting features automatically, without using human expertise, have been studied extensively in games such as chess (Hartmann, 1989; Beal and Smith, 1994; Anantharaman, 1997), checkers (Chellapilla and Fogel, 2001), and backgammon (Pollack and Blair, 1998). A generic pattern-based approach to define and select features automatically was established by Kaneko *et al.* (2003), and an approach to produce conjunctions of basic features was established by Agrawal *et al.* (1995), both of which have been applied successfully to the game of Othello (Buro, 2003).

[2] In this particular research, a robot was constructed to play the game under real-time conditions on an actual Tetris arcade machine. The robot's only input signals came from a webcam monitoring the screen of the Tetris machine. A relay board was used to have the robot "press keys" on the machine.

Despite some progress being made over the past decades, researchers argue that the problem of feature definition and selection remains still largely unsolved (Fürnkranz, 2007; Droste and Fürnkranz, 2008).

B. Feature weighting

Given a decision as to which features to include in the evaluation function, one must then decide upon their relative importance (see Samuel, 1959; Gomboc *et al.*, 2005). As was the case with feature definition and selection, both in classic games and video games, it is common that the task of feature weighting is performed by the researcher. This is a relatively straightforward task, yet one that requires (1) deep knowledge of the game environment, and (2) much time for tuning the evaluation function (Van der Meulen, 1989; Miwa *et al.*, 2006). To avoid these potential obstacles, it is possible to establish appropriate feature weights automatically. However, one also needs to consider the significant complexity of video games, which has increased continuously over the years. Therefore, one may assume that the task to establish appropriate feature weights for evaluation functions in video games, is one that will grow too difficult for game developers to perform. We therefore expect that feature weighting of evaluation functions in video games will increasingly be performed automatically.

Yet, establishing appropriate feature weights automatically is difficult too, since there typically are no direct target values that could be used as training signals (Fürnkranz, 2007). Fürnkranz (2007) states that for tuning the evaluation function, in general, algorithms use preference learning (Fürnkranz and Hüllermeier, 2005) (where pairs of moves or positions are labelled according to the preference of an expert player) or reinforcement learning (Sutton and Barto, 1998) (where moves or positions are trained based on information about the eventual outcome of the game). Researchers typically opt for a linear combination of features, as this keeps the evaluation time overhead low, and allows a fast approximation of optimal weights, even in large systems (Buro, 2002).

TD-Gammon (Tesauro, 1992) is typically referred to as the premier example of how learning feature weights in an evaluation function can be used to yield a computer program capable of playing at expert strength. TD-Gammon is a neural network that trains itself to be an evaluation function for the game of backgammon, by playing against itself and learning from the outcome (Tesauro, 1992). To this end it incorporates temporal-difference (TD) learning (Sutton, 1988). This learning technique has, for instance, been applied successfully for learning piece values in the domain of chess (Beal and Smith, 1997), and in the chess game-playing program KnightCap (Baxter *et al.*, 1998). Here we repeat the successful example of learning automatically feature weights for an evaluation function in the game of Tetris. In this game, the feature weights in top-rated evaluation functions have been learned automatically (Thiery and Scherrer, 2009a,b). Still, in video games the application of automatically weighted evaluation functions is rare. For additional ideas in the domain of learning feature weights we refer, for instance, to CBR-related research by Leake *et al.* (1995), Muñoz-Avila and Hüllen (1996), Wettschereck *et al.* (1997), and Zhang and Yang (2001), and to research by Böhm *et al.* (2005), Szita and Lörincz (2006), and Chaslot *et al.* (2008c).

5.1.4 Exploiting game knowledge

The use of a data set (and by extension, a case base) of game knowledge can be instrumental to establishing effective game AI. A premier example of effective use of a data set of game knowledge, is how the game of checkers was solved. In order to solve checkers within a more-or-less reasonable timeframe, a data set was used that consisted of 3.9×10^{13} game positions (Schaeffer *et al.*, 2007).

Successful use of a data set of game knowledge in the domain of classic games has been illustrated in, for instance, the game of shogi, where game records were used to tune an evaluation function automatically (Miwa *et al.*, 2006), and in the game of chess, where evaluation functions were tuned based upon a global ordering of a multiplicity of positions (Gomboc *et al.*, 2005).

In the last decade, also in the domain of video games, approaches have been established for creating game AI based upon game knowledge gathered in data sets. For instance, researchers established a successful method to imitate playing styles of players in the popular fighting game Street Fighter Zero 3 (Thunputtarakul and Kotrajaras, 2007). In related work, researchers applied data sets for solving some of the decision-making problems presented in a clone of the game Civilization II (Sánchez-Pelegrín *et al.*, 2005). An observation of particular interest is that video games capable of supporting hundreds or thousands of players simultaneously (i.e., massively multiplayer online games (MMOGs)) allow a data set of game knowledge to expand rapidly (Spronck, 2005b). The popularity of these video games has increased drastically over the years (Harding-Rolls, 2009). For illustration, World of Warcraft, the most popular MMOG, counts over 11.5 million monthly subscribers (Blizzard, 2008), and has generated over $2.2 billion dollars since it was released to the market (Harding-Rolls, 2009). Considering the growing availability of game observations, we expect that in the near future increasingly more game AIs, and by implication evaluation functions, will be established on the basis of automatically gathered game knowledge.

5.2 An evaluation function for SPRING

In this section we outline our evaluation function for an actual, complex RTS game. To provide context for the reader, we first give a concise description of the typical RTS game Spring (5.2.1), in which we will later validate the evaluation function. Subsequently, we describe how we will exploit a case base of game knowledge (5.2.2). Next, we propose the general form of our evaluation function (5.2.3). The function consists of three components, which are described in their own subsection. The three components are (1) a parameter to indicate the phase of the game (5.2.4), (2) a term to measure the material strength (5.2.5), and (3) a term to measure the commander safety (5.2.6).

5.2.1 The game environment

To provide context for the reader, in this subsection we give a concise description of the game Spring. We organise our description as follows. First, we outline the play of the game. Second, we describe the topic of environment visibility. Third, we discuss how a so-called tech tree is incorporated in the game. An extensive description of SPRING is provided in Appendix A.2.

Play of the game: SPRING is a typical and open-source RTS game. It presents a strategic, simulated war-game environment. In the game, generally two players are pitted against each other. This type of play is investigated in the thesis. Alternatively, it is also common that two teams of players are pitted against each other.

Each player of the game needs to gather in-game resources for the construction of units and buildings. The game involves play with up to 200 different unit types. At the start of the game, each player is provided with one unit, the Commander unit.

Some unit types are directly involved in combat and are mobile (e.g., tank units). Other unit types are not directly involved in combat and have a fixed location (e.g., metal extractor buildings). The goal of the game is to defeat an opponent army in a real-time battle, by using effectively the constructed units and buildings. A SPRING game is won by the player who first destroys the opponent's Commander unit.

Environment visibility: SPRING implements a so-called 'Line of Sight' visibility mechanism to each unit. This implies that game AI only has access to observational data of those parts of the environment that are visible to its own units (illustrated in Figure 5.1). When the game AI's information is restricted to what its own units can observe, we call this an 'imperfect-information environment'. When we allow the game AI to access all information, regardless whether it is visible to its own units or not, we call this a 'perfect-information environment'.

Tech tree: SPRING employs a so-called tech tree (Adams and Rollings, 2006) that represents the possible paths of in-game 'research actions' that a player of the game can take. The name 'tech tree' is common terminology in the domain of video games, and is derived from 'technology tree'. A tech tree is a directed acyclic graph. For all players of the game, the employed tech tree is identical. The manner in which each player traverses the tree is independent from the in-game actions of other players. Each player starts in the root of the tech tree. By traversing the tech tree, the particular player will be provided with advanced levels of technology. For instance, by performing certain in-game research actions, a player may be provided with more advanced aircraft units. Traversing the tech tree is generally advantageous, yet there is a cost for doing so in time and in in-game resources.

A concise representation of the tech tree of SPRING is provided in Figure 5.2. In the SPRING game, three levels of technology are predefined: Level 1, Level 2, and Level 3. At the start of the game, each player can only construct Level-1 units and Level-1 buildings. Later in the game, after a player has performed the required in-game research, advanced units and buildings of Level 2 and Level 3 become available to the particular player. For instance, a player may choose to focus on constructing the most advanced Level-3 k-bot units. However, by doing so, a player invests a significant amount of time and in-game resources, that cannot be spent on constructing other potentially helpful units, such as Level-2 tanks and Level-2 aircrafts.

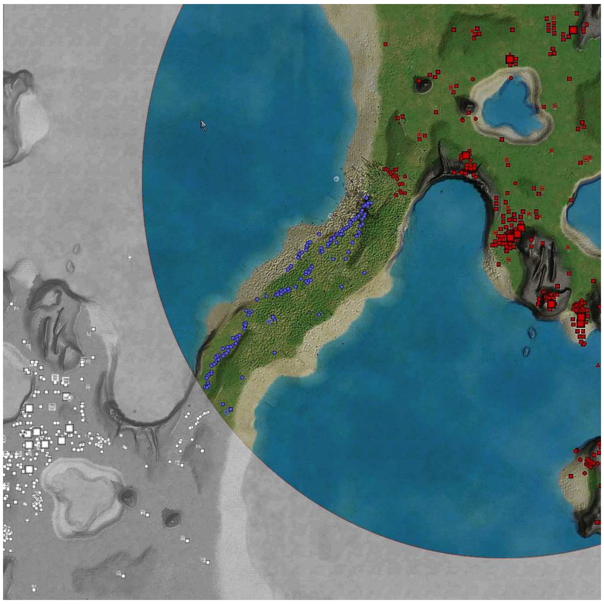


Figure 5.1: A game observation in the Spring game. Units controlled by the game AI are currently residing in the highlighted region. In an imperfect-information environment, only information in the highlighted region is available. In a perfect-information environment, information for both the highlighted and the grey region is available.

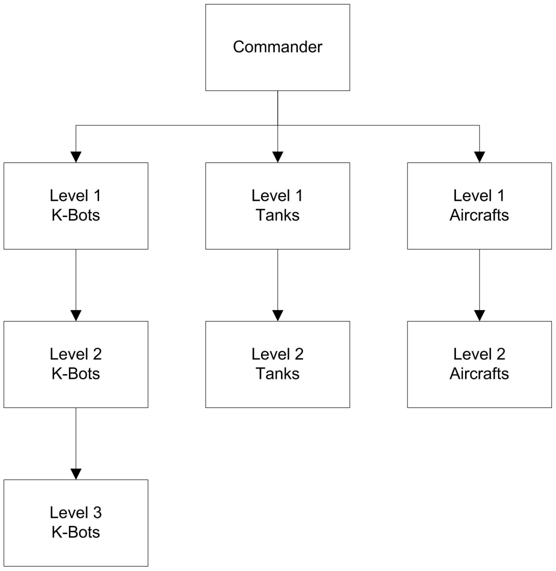


Figure 5.2: Concise representation of the tech tree of Spring.

5.2.2 Towards a case base of game knowledge

For the purpose of establishing our evaluation for the Spring game, we will create a case base of game knowledge. We define game knowledge as a set of observable features of the environment at a certain point in time. Our procedure of gathering feature data in the case will be described in Subsection 5.3.1. Subsequently, our procedure to establishing an evaluation function follows the two steps described in Subsection 5.1.3, viz. (A) feature definition and selection, and (B) feature weighting. The implementation of the two steps is described below.

A. Feature definition and selection

In Subsection 5.1.3 we discussed that determining which features to make available to an evaluation function is regarded as an important but difficult task. We observed that the task is usually achieved by a researcher that exploits expert knowledge of the game. For instance, in the video-game program Tetris, highly accurate evaluation functions have been established solely on the basis of manually defined features (Thiery and Scherrer, 2009a,b). We note that any set of game features may be incomplete, and may thereby limit the accuracy of the evaluation function. Yet, as a first step to establishing an evaluation function for the Spring game, we accompanied our predecessors in exploiting our expert knowledge of the game, and determine by hand which features to make available to the evaluation function.

Our expert knowledge of Spring dictates that in the game the evaluation function should foremost be capable of estimating (1) the phase of the game, (2) the material strength, and (3) the commander safety. In order to provide accurate estimates, the evaluation function needs to take into account the environment visibility. These considerations lead us to define four features, on the basis of which the three estimates can be calculated. The four features are as follows.

1. **Phase of the game**. The feature represents the phase of the game (e.g., the opening or endgame). It is selected to acknowledge that certain behaviour is only important at distinct moments in the game. The value of the feature can be inferred from the presence of particular units that are operating in the game environment. We consider five distinct game phases. This is discussed in more detail in Subsection 5.2.4.
2. **Material strength**. The feature represents the strength of the player's army, in comparison to the strength of the opponent's army. It is selected to acknowledge the strategic nature of the game. The value of the feature can be inferred from the number of units observed of each of the 200 available unit types. This is discussed in more detail in Subsection 5.2.5.
3. **Commander safety**. The feature represents the safety of the player's Commander unit, in comparison to the safety of the opponent's Commander unit. It is selected to acknowledge that a Spring game is won by the player who first destroys the Commander unit of the opponent. The value of the feature can be inferred from the number of enemy units that are located in the vicinity of the Commander units. This is discussed in more detail in Subsection 5.2.6.

4. **Environment visibility**. The feature represents the percentage of the game environment that is visible to the friendly player. It is selected to acknowledge that, despite partial observability of the environment, the evaluation function needs to rate accurately the game state. The value of the feature can be observed directly.

For convenience of the reader, in Appendix B we will provide an overview of all features that are used in the three main components of case-based adaptive game AI.

B. Feature weighting

Once that the game features are defined, we can gather feature data in the case base. We will exploit data that is gathered in the case base for the purpose of weighting the evaluation function. As we will describe in the next subsections, the defined features are incorporated into the evaluation function as two terms (i.e., (1) material strength, and (2) commander safety), and as a parameter to represent the phase of the game. We will optimise the weights for each term of the evaluation function. Also, we will learn weights for each of the 200 available unit types (incorporated into the material strength term). We refer to the entire process as 'tuning the evaluation function'. The implemented procedure to gathering feature data and tuning the evaluation function will be described in detail in Subsection 5.3.1.

With regard to tuning the evaluation function, one may assume that the accuracy of the evaluation function is highest when perfect information is used to tune it. Therefore, we will tune our evaluation function based on observations gathered in a perfect-information environment. In Subsection 5.3.1 we will create a case base that consists of three different data sets: the first containing *training* data gathered in a *perfect-information* environment, the second containing *test* data gathered in a *perfect-information* environment, and the third containing *test* data gathered in an *imperfect-information* environment. The latter test set is used to determine how well an evaluation function that is trained in a perfect-information environment, can be applied to an imperfect-information environment.

5.2.3 Our evaluation function

Derived from our expert knowledge of the SPRING game (see Subsection 5.2.2), we decided for an evaluation function that includes a parameter to represent the phase of the game, a term to estimate the material strength, and a term to estimate the commander safety. We will use the evaluation function as a predictor of the final outcome of a game.

To allow the evaluation function to deal with imperfect information inherent to the SPRING environment, we assume that is it possible to map reliably the imperfect feature data, to a prediction of the perfect feature data. Our straightforward implementation of this mapping is to scale linearly the number of observed opponent units to the non-observed region of the environment. If the opponent units are homogenously distributed over the environment, the evaluation function applied to an imperfect-information environment will produce results close to those of the evaluation function applied to a perfect-information environment.

The considerations mentioned above lead us to denote our evaluation function as follows.

$$v(p) = w_p v_1 + (1 - w_p) v_2 \tag{5.2}$$

where $w_p \in [0, 1]$ is a free parameter to determine the weight of each term v_n of the evaluation function, where $n \in \{1, 2\}$, and where $p \in \{1, 2, 3, 4, 5\}$ is a parameter that represents the current *phase of the game* (see Subsection 5.2.4). The evaluation function incorporates two evaluation terms, the term v_1 that represents the material strength (see Subsection 5.2.5), and the term v_2 that represents the commander safety (see Subsection 5.2.6).

5.2.4 Phase of the game

The parameter p (see Equation 5.2) represents the current phase of the game. The parameter is incorporated to acknowledge that certain behaviour may only be important at distinct moments in the game. For instance, at the start of the game it may be important to construct a strong army, where near the end of the game it may be more important to defend the Commander unit. The parameter value is derived from data on the phase of the game, which is gathered in the case base (see Subsection 5.2.2-A).

Analogously to how one may distinguish game phases in classic games (e.g., opening, middlegame, and endgame in the game of chess), one may distinguish game phases in RTS games. Modern RTS games typically progress through several distinct phases. The phase of an RTS game can be derived straightforwardly from the observed traversal in the tech tree of the game. This use of phases is similar to the background knowledge that Ponsen and Spronck (2004) introduced in their work, and was used later in Aha *et al.* (2005).

We recall that in the Spring game three levels of technology are predefined (see Subsection 5.2.1). To distinguish game phases in the Spring game, we map the three predefined levels of technology to game phases. In addition, to distinguish between when tech levels are "new", and when they are "mature", we also regard the transition from one level of technology to another as a game phase. The maturity of a tech level is indicated by the presence of units with a relatively long construction time. For the Spring game, this leads us to define five game phases. The five phases are described below.

Phase 1: Level-1 buildings observed.
Phase 2: Level-1 units observed that have a construction time $\geqslant$ 2500 in-game time units.
Phase 3: Level-2 buildings observed.
Phase 4: Level-2 units observed that have a construction time $\geqslant$ 15000 in-game time units.
Phase 5: Level-3 buildings observed.

5.2.5 Material strength

The evaluative term v_1 (see Equation 5.2) represents the material strength. The material strength is expressed by the number of units in the army of the two players, as well as by the weight (i.e., relative importance) of each unit in the particular army. The term is incorporated to acknowledge the strategic nature of the game. It utilises data on material strength and environment visibility, which is gathered in the case base (see Subsection 5.2.2-A). Term v_1 is denoted by

$$v_1 = \sum_{u} w_u (c_{u_1} - \frac{o_{u_2}}{R}) \tag{5.3}$$

where w_u is the experimentally determined weight of the unit u, c_{u_1} is the number of own units of type u that the game AI has, o_{u_2} is the *observed* number of opponent's units of type u, and $R \in [0, 1]$ is the fraction of the environment that is visible to the game AI.

5.2.6 Commander safety

The evaluative term v_2 (see Equation 5.2) represents the safety of the current tactical position. The safety is expressed by the number of enemy units that are located in the vicinity of the Commander unit. The term is incorporated to acknowledge that a Spring game is won by destroying the opponent's Commander unit. It utilises data on commander safety and environment visibility, which is gathered in the case base (see Subsection 5.2.2-A). Term v_2 is denoted by

$$v_2 = \sum_{r \in d} w_r (\frac{o_{r_2}}{R_{r_2}} - \frac{o_{r_1}}{R_{r_1}}) \tag{5.4}$$

where w_r is the experimentally determined weight of the radius r, o_{r_2} is the *observed* number of own units of the game AI within a radius r of the opponent's Commander unit, $R_{r_2} \in [0, 1]$ is the fraction of the environment that is visible to the opponent within the radius r, o_{r_1} is the *observed* number of units of the opponent within a radius r of the game AI's Commander unit, $R_{r_1} \in [0, 1]$ is the fraction of the environment that is visible to the game AI within the radius r, and d is the set of experimentally determined game-unit radii $\{500, 1000, 2000\}$.[3]

5.3 Validating the evaluation function

This section reports on two experiments that validate the evaluation function in the Spring game. We are foremost interested in the performance of the evaluation function. In the experiments, we therefore measure the performance of the evaluation function in actual play. The performance is expressed by the accuracy of the function in predicting the final outcome of a game. A high accuracy indicates that the established evaluation function may be suitable for incorporation into case-based adaptive game AI.

In the remainder of the section, we first describe the experimental setup (5.3.1). Second, we discuss how the performance of the evaluation function is assessed by means of two measures (5.3.2). Third, we give the measured performance (5.3.3). Fourth, the section is concluded by a discussion of the results (5.3.4).

[3] A 'game-unit' is an internal distance unit that is employed in the Spring game. For instance, the size of a typical game map in the Spring game is 8000 x 8000 game units.

Friendly player	Opponent player	Training set	Test set (perf. inf.)	Test set (impf. inf.)
AAI (cb)	AAI (original)	500	200	200
AAI (cb)	TSI	100	200	200
AAI (cb)	CSAI	100	200	200
AAI (cb)	RAI	-	200	200

Table 5.1: The number of SPRING games gathered in the case base.

5.3.1 Experimental setup

The general experimental setup is as follows. First, in the case base we gather feature data of a multitude of games. Subsequently, we utilise this feature data for tuning our evaluation function. Next, with the established evaluation function, we announce two experiments that will test the function. In the first experiment, we test the function while operating in a perfect-information SPRING environment. In the second experiment, we test the function while operating in an imperfect-information SPRING environment.

Below, the three steps, viz. (A) gathering feature data, (B) tuning the evaluation function, and (C) testing the evaluation function, are discussed in more detail.

A. Gathering feature data

For establishing the evaluation function, in the case base we gather feature data of games where two game AIs are pitted against each other. As multiple SPRING game AIs are available, we first have to select a game AI that is suitable for our experiments. We used one open-source game AI, which the author of the game AI labelled 'Alexander AI' ('AAI') (Seizinger, 2006). We enhanced this game AI with two abilities, viz. (1) the ability to gather feature data in a case base, and (2) the ability to disregard limited environment visibility, so that perfect information on the environment is available. We note that the decision-making process of the enhanced AAI is identical to that of the original AAI; exhibited behaviour in actual play is not affected by the enhancements. We refer to the enhanced AAI, that gathers feature data in a case base, as 'AAI (cb)'. As opponent game AIs, we used the original AAI, as well as three other AIs, namely 'TSI' (Baran and Urbanczyk, 2006), 'CSAI' (Perkins, 2006), and 'RAI' (Reth, 2007). A description of these game AIs is provided in Appendix A.2. Table 5.1 lists the number of games from which we built the case base. The case base consists of three sets. The first set is a training set containing data gathered in a perfect-information environment. The remaining two sets are test sets containing data gathered in a perfect and imperfect-information environment, respectively.

The data gathering process was as follows. During each game, feature data was gathered every 127 in-game cycles, which corresponds to the decision-making frequency of AAI. With 30 in-game cycles per second this resulted in feature data being gathered every 4.233 seconds. The games were played on the default map of the SPRING game, the map SmallDivide (it will be illustrated in Figure 6.2). The map is symmetrical, and has no water areas. All observed games were played under identical conditions.

B. Tuning the evaluation function

Feature data that is gathered in the case base, is utilised for tuning the evaluation function. The process consists of two steps. First, we learn the unit-type weight w_u (see Equation 5.3) for each of the available 200 unit types of the Spring game. Second, we optimise the term weights w_p for each phase of the game (see Equation 5.2). Both steps are discussed below.

To learn the unit-type weights w_u of the evaluation function, we used a MatLab implementation of TD-learning (Sutton, 1988). Though the choice for a particular learning algorithm is immaterial as we are establishing a proof of concept, our choice for TD-learning is motivated by findings in the literature. That is, our application of TD-learning in the Spring game is analogous to its application by Beal and Smith (1997) for determining piece values in chess. Unit-type weights were learned from feature data gathered with perfect information, that is contained in the training set in the case base (see Table 5.1). The learning rate α was set to 0.1, and the recency parameter λ was set to 0.95. Both parameter values were chosen in accordance with the research by Beal and Smith (1997). The unit-type weights were initialised to 1.0 before learning started. For our setup, weights of the radii defined in the set d (see Subsection 5.2.6) were chosen by the experimenter as 0.05 for a radius of 2000, and 0.20 and 0.75 for a radius of 1000 and 500, respectively.

In addition to learning the unit-type weights, we optimised the term weights w_p for each phase of the game. Term weights were determined with the gradient descent optimisation algorithm (Snyman, 2005). Though the choice for a particular optimisation algorithm is immaterial as we are establishing a proof of concept, we chose for gradient descent optimisation out of pragmatic considerations. A step value of 0.01 was used initially to allow the algorithm to explore the state space using random jumps. Gradually, the step value was decreased to 0.001, to encourage the algorithm to perform a local optimisation at the end. Term weights for each phase were initialised to a neutral value of 0.5 before optimisation started.

For convenience of the reader, in Appendix C we will provide an extensive description of the tuned evaluation function.

C. Testing the evaluation function

We determined the performance of the evaluation function in two experiments. The first experiment tests the evaluation function in an perfect-information environment. To this end, game observations that are gathered in the set 'test set (perf. inf.)' are used (see Table 5.1). The second experiment tests the evaluation function in an imperfect-information environment. To this end, game observations that are gathered in the set 'test set (impf. inf.)' are used. For each game observation that is gathered, the evaluation function attempts to rate the state of the game. The evaluation value is an implicit prediction of the final outcome of the game. A positive evaluation value indicates that the friendly player is winning the game. A negative evaluation value indicates that the opponent player is winning the game. A neutral evaluation function of zero indicates that the game resides in a draw position. How we measure the performance of the evaluation function is discussed next.

5.3.2 Two measures for performance assessment

We defined two measures to assess the performance of the evaluation function operating in the Spring game: (1) the final prediction accuracy, which reflects the function's accuracy at the end of the game, and (2) the turning points, which reflect the function's accuracy throughout the play of the game. Both are discussed below.

Final prediction accuracy: We defined the measure 'final prediction accuracy' as the percentage of games of which the outcome is predicted correctly just before the end of the game. A high final prediction accuracy indicates that the evaluation function has the ability to evaluate correctly the state of the game.

Turning points: We defined the measure 'weak turning point' as the game time at which at least 50% of the outcomes of the test games is predicted correctly. We defined the measure 'normal turning point' and 'strong turning point' as the game time at which at least 60% and 70% of the outcomes of the test games is predicted correctly, respectively. A low value of a turning point indicates that the evaluation function can correctly evaluate a game's state early in the game.

We note that not all games last for an equal amount of time. To determine the turning points, we scaled all games to one hundred per cent of the game time of the longest game. This is implemented by averaging over the outcome predictions made for each game observation. As a result, the lowest (i.e., best) turning point that can be obtained is 1, and the highest (i.e., worst) turning point that can be obtained is 100.

5.3.3 Measured performance

This subsection discusses the results of the two performed experiments. The first experiment tests the evaluation function in a perfect-information environment. The second experiment tests the evaluation function in an imperfect-information environment. The performance of the evaluation function is measured in terms of (1) final prediction accuracy, and (2) obtained turning points.

The evaluation function used is qualitatively discussed in terms of (A) the unit-type weights that, when tuning the evaluation function, were learned for the 200 available unit types (see Subsection 5.3.1). Also, we present the results of (B) optimising the term weights for each of the five phases of the game. Then, the measured performance is reported. That is, we give (C) the final prediction accuracy results, and subsequently, we give (D) the turning point results.

A. Learned unit-type weights

When gathering feature data, we found that of the 200 available unit types of the Spring game, 89 unit types were used. The remaining unit types were not used due to the AIs' preferences for different unit types, and due to the characteristics of the map on which the data was gathered. For instance, on a map without water areas it is not possible to build ship units. The TD-learning algorithm learned the weights w_u (see Equation 5.3) for these

Unit type	Description	Weight
108	Advanced Metal Extractor (Building)	5.91
155	Thunder Bomber (Combat unit)	5.57
110	Metal Storage (Building)	4.28
68	Freedom Fighter (Combat unit)	4.23
149	Medium Assault Tank (Combat unit)	4.18
…	…	
106	Minelayer / Minesweeper with Anti-Mine Rocket (Combat unit)	-1.10
11	Advanced Solar Collector (Building)	-1.28
114	Light Amphibious Tank (Combat unit)	-1.52
56	Energy Storage (Building)	-1.70
124	Defender Anti-Air Tower (Building)	-2.82

Table 5.2: Learned unit-type weights.

89 unit types. A summary of ten learned weights is given in Table 5.2. The table lists the five unit types with the largest learned weight, and the five unit types with the smallest learned weight. A full list is given in Appendix C. Below we give a qualitative discussion of the learned weights.

The table reveals that the highest weight has been assigned to the Advanced Metal Extractor. This result confirms our expert knowledge with the game. When the AI destroys an Advanced Metal Extractor, then the opponent's ability to gather resources reduced, and on top of that it is also likely that the AI has already penetrated the opponent's defences, since the Advanced Metal Extractor is typically well protected and resides close to the Commander unit. This implies that, when the AI destroys an Advanced Metal Extractor, it is a good indicator that the game AI in question is likely to win the game.

In addition, the table reveals that some unit types obtained weights less than zero. This result confirms our expert knowledge with the game. In our experimental setup, several unit types are of little use to a player, and are actually a waste of in-game resources. For instance, the map used in our experiments contained no water areas. As a result, the Light Amphibious Tank unit will indeed be of limited use.

B. Optimising term weights results

Table 5.3 displays the prediction accuracy before and after optimising the term weights w_p for each phase of the game (see Equation 5.2). The first column lists the phase of the game in which the optimisation took place. The second column lists the total number of game observations done in the corresponding phase of the game. The third and fourth column lists how many game observations the outcome has been predicted accurately before optimising and after optimising, respectively. The fifth column lists the increase (in percentage) in prediction accuracy after optimising.

The table reveals that after optimising the term weights, the evaluation function's ability to predict the outcome of the game increases for all phases of the game. The most substantial

Phase	Game observations	Correct (before)	Correct (after)	Incr. (%)
1	96,966	30,649	38,451	25.5%
2	74,570	43,150	44,048	2.1%
3	97,892	43,441	64,495	48.5%
4	46,535	35,464	36,139	1.9%
5	14,784	12,365	12,401	0.3%

Table 5.3: Optimising term weights results.

		Final prediction accuracy	
Friendly player	Opponent player	Perfect-inf. env.	Imperfect-inf. env.
AAI (cb)	AAI (original)	99%	92%
AAI (cb)	TSI	96%	88%
AAI (cb)	CSAI	98%	93%
AAI (cb)	RAI	95%	87%
Average		97%	90%

Table 5.4: Final prediction accuracy results.

increase in prediction performance is observed in phase three (48.5%), and in phase one (25.5%). These two phases are the phases in which the game mostly resides.

C. Final prediction accuracy results

Table 5.4 lists the final prediction accuracy of the evaluation function. For the evaluation function operating in a perfect-information environment, the final prediction accuracy is 97% on average. For the evaluation function operating in an imperfect-information environment, the final prediction accuracy is 90% on average.

We note that in the Spring game it is not evident that the evaluation function achieves a high prediction accuracy just before the game is finished. Namely, the game is finished when a player's Commander unit is destroyed. This can occur suddenly, for instance, by an unlucky move of the Commander unit, or by a powerful opponent unit that is able to destroy the Commander with a single shot. Still, in these circumstances our evaluation function achieves a high prediction accuracy, which can be attributed to the term to evaluate the commander safety. From these results, we may conclude that the established evaluation function provides an effective basis for evaluating a game's state.

D. Turning point results

In Table 5.5 the obtained turning points are given for the evaluation function applied in a perfect-information environment. In Table 5.6 the obtained turning points are given for the evaluation function applied in an imperfect-information environment. Note that a low turning point indicates that the evaluation function can correctly evaluate a game's state

Friendly player	Opponent player	Turning points Weak t.p.	Normal t.p.	Strong t.p
AAI (cb)	AAI (original)	39	50	64
AAI (cb)	TSI	67	74	87
AAI (cb)	CSAI	85	88	92
AAI (cb)	RAI	1	72	89
Average		48	71	83

Table 5.5: Turning point results obtained for the evaluation function applied in a perfect-information environment.

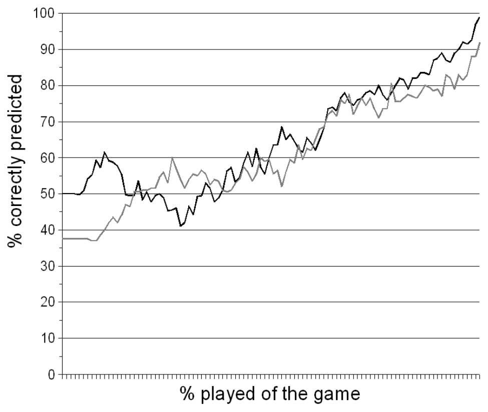
Friendly player	Opponent player	Turning points Weak t.p.	Normal t.p.	Strong t.p
AAI (cb)	AAI (original)	18	59	64
AAI (cb)	TSI	57	87	98
AAI (cb)	CSAI	1	1	86
AAI (cb)	RAI	48	85	94
Average		31	58	86

Table 5.6: Turning point results obtained for the evaluation function applied in an imperfect-information environment.

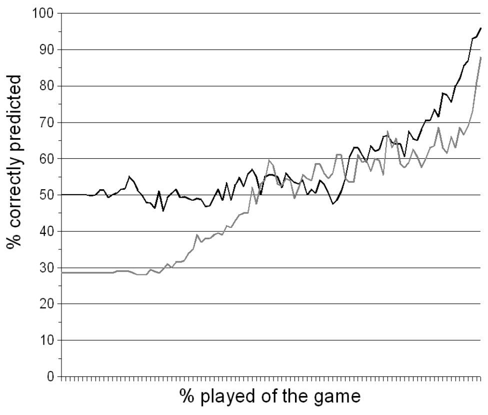
early in the game. We observe that the weak turning point is on average 48 in a perfect-information environment, and is on average 31 in an imperfect-information environment. This result reveals that the evaluation function makes fairly accurate predictions before half of the game is played.

An interesting observation is that on average, the weak turning point in an imperfect-information environment is smaller (i.e., better) than the weak turning point in a perfect-information environment. The phenomenon can be explained straightforwardly. Relatively early in the game, little information is available to the evaluation function operating in an imperfect-information environment. Still, using this little information, the function attempts to estimate the actual, perfect information. Based on this estimate, the final outcome of the game is being predicted. Due to lucky circumstances (e.g., correctly overestimating the strength of the opponent), this prediction may in fact be more accurate than the prediction that is based on actual game observations. An instance of the phenomenon is illustrated in the lower-left graph of Figure 5.3, whose legend will be described below.

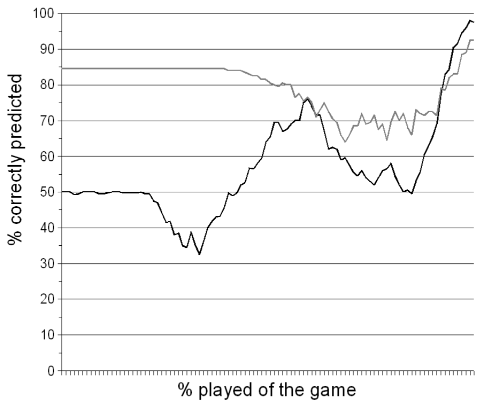
The normal turning point is on average 71 and 58 in a perfect-information and imperfect-information environment, respectively. The strong turning point is on average 83 and 86 in a perfect-information and imperfect-information environment, respectively. These results reveal that when more environment information becomes available in an imperfect-information environment, the performance of the evaluation function approaches that of the function operating in a perfect-information environment. This is also shown by Figure 5.3,



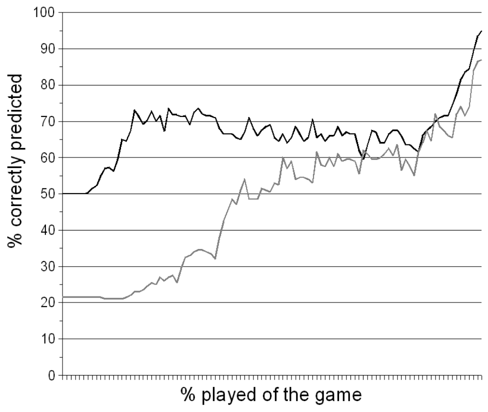
(a) AAI (cb) - AAI (original)



(b) AAI (cb) - TSI



(c) AAI (cb) - CSAI



(d) AAI (cb) - RAI

Figure 5.3: Outcomes predicted correctly as a function over time. The black line represents the prediction performance of the evaluation function operating in a perfect-information environment. The grey line represents the prediction performance of the evaluation function operating in an imperfect-information environment.

which displays the percentage of outcomes predicted correctly as a function over time. The figure gives the prediction accuracy of the evaluation function operating in a perfect-information environment (the black line), as well as the prediction accuracy of the evaluation function operating in an imperfect-information environment (the grey line).

5.3.4 Discussion of the results

Experiments that tested our evaluation function showed that the function was able to predict accurately the outcome of the Spring game. Naturally, there are possibilities for further improvements. For instance, the accuracy of the evaluation function may be restricted by the selection of game features that were incorporated into the function. In addition, the accuracy of the evaluation function will depend on the effectiveness of tuning the evaluation function. Particularly, the accuracy will depend on the training data that is provided to the tuning process. For example, our results showed some signs of overfitting, which might have resulted from the fact that most of the feature data was gathered from games that are played against the original AAI opponent. To avoid overfitting, we recommend gathering data from training games played against a relatively large number of different types of opponents on different types of maps.

Alternatively, one may choose to tune a distinct evaluation function for each known opponent. If the game AI is able to determine which opponent it is pitted against, it can use the appropriate evaluation function to evaluate the state of the game. However, the alternative requires the game AI to classify accurately the opponent player. This is a difficult task.

5.4 Chapter conclusions

In this chapter we discussed the function that rates the state of a game, viz. the evaluation function. We outlined our evaluation function for the complex Spring game. The evaluation function takes into account the phase of the game, and incorporates two terms to evaluate the game state, viz. (1) a term to measure the material strength, and (2) a term to measure the commander safety. Experiments performed in the Spring game revealed that, just before the game's end, the evaluation function was able to predict correctly the outcome of the game with an accuracy that approached one hundred per cent. Considering that a Spring game may be won suddenly, and thus the outcome of the game is difficult to predict, this is a satisfactory result. In addition, the evaluation function made fairly accurate predictions before half of the game was played, operating in both a perfect and imperfect-information environment. From these results, we may conclude that our evaluation function predicts accurately the outcome of a Spring game.

6 The adaptation mechanism

In this chapter we concentrate on the adaptation mechanism, one of the three main components of case-based adaptive game AI. The adaptation mechanism allows game AI to adapt online to game circumstances. In our investigation of an adaptation mechanism for video game AI, we focus particularly on establishing a mechanism that addresses the concerns of game developers with regard to adaptive game AI, i.e., we investigate a mechanism capable of adapting rapidly and reliably to game circumstances.

Chapter 6 is organised as follows. We first discuss adaptation mechanisms for video game AI (Section 6.1). Subsequently, we describe our mechanism for online adaptation of video game AI (Section 6.2). Next, we report on the validation of the adaptation mechanism (Section 6.3). Finally, we present the chapter conclusions (Section 6.4).

6.1 Adaptation mechanisms

For the purpose of the present investigation, we reiterate that an adaptation mechanism is a mechanism that allows the game AI to adapt online to game circumstances. In video games, each game character feeds the game AI with data on its current situation, and with the observed results of its actions. The game AI adapts by processing the observed results, and generates actions in response to the character's current situation. An adaptation mechanism is incorporated into game AI to determine what the best way is to adapt the AI. For instance,

This chapter is based on the following three publications.

1) Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008a). Rapid adaptation of video game AI. In Botti, V., Barella, A., and Carrascosa, C., editors, *Proceedings of the 9th International Conference on Intelligent Games and Simulation (GAMEON'2008)*, pages 69–76. EUROSIS-ETI, Ghent University, Ghent, Belgium.

2) Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008b). Rapid adaptation of video game AI (Extended version of 2008a). In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 79–86. IEEE Press, Piscataway, New York, USA.

3) Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009b). Rapid and reliable adaptation of video game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104.

reinforcement learning may be applied to assign rewards and penalties to certain behaviour determined by the game AI.

In the motivation of our research, we discussed the general goal of adapting rapidly and reliably to game circumstances (see Chapter 1). In the remainder of this section, we describe how the general goal affects the design considerations of our adaptation mechanism. We first describe the aim of exploiting observations gathered in a case base for the purpose of enabling rapid adaptation of game AI (6.1.1). Subsequently, we discuss how a case-based approach to game adaptation may be used to increase the reliability of game AI (6.1.2).

6.1.1 Rapid adaptation

In recent years researchers have increasingly adopted case-based reasoning (CBR) and case-based planning (CBP) approaches in their work in order to deal with the complexities of video games. Often, these case-based approaches are focussed on decreasing the time required to learn effective behaviour in online play. For instance, Sharma *et al.* (2007) developed an approach for achieving transfer learning in the MADRTS game, by using a hybrid case-based reasoning and reinforcement learning algorithm. Auslander *et al.* (2008) used case-based reasoning to allow reinforcement learning to respond as rapidly as possible to changing circumstances in the UNREAL TOURNAMENT domination game.

Often, the focus of these approaches lies on learning effective behaviour. Yet, it is not uncommon that a game has finished before any effective behaviour could have been established, or that the game characters in a game do not live sufficiently long to benefit from learning. As a result, it is difficult for the players of a video game to detect and understand that the game AI is learning. This renders the benefits of online learning in video games subjective and unclear (Rabin, 2008).

Therefore, our adaptation mechanism is not so much focussed on learning, but on exploiting game observations stored in the case base for the purpose of *instant* application in game circumstances. We build upon (1) the ability to gather and identify relevant game observations, and (2) the ability to apply effectively these observations in similar game circumstances. Corresponding case-based approaches have been applied to various game genres (see Aha *et al.* (2005) for an overview). We observe that often the focus lies on relatively simple tasks in relatively simple environments, e.g., predicting the next action of a player in SPACE INVADERS (Fagan and Cunningham, 2003). However, in recent years, case-based research has expanded into the domain of RTS games, albeit predominantly to relatively simple instances of the genre. For instance, Aha *et al.* (2005) developed a retrieval mechanism for tactical plans in the WARGUS game, that built upon domain knowledge generated by Ponsen and Spronck (2004). Ontañón *et al.* (2007) and Mehta *et al.* (2009) established a framework for case-based planning on the basis of annotated knowledge drawn from expert demonstrations in the WARGUS game. Louis and Miles (2005) applied case-injected genetic algorithms to learn resource allocation tasks in RTS games. Baumgarten *et al.* (2009) established a mechanism for simulating human gameplay in strategy games using a variety of AI techniques, including, among others, case-based reasoning.

6.1.2 Reliable adaptation

For video games it is difficult to create adaptive game AI that is reliable, i.e., game AI in which adaptive behaviour is established in a controlled and predictable manner. One way of increasing the reliability of an adaptation mechanism is to incorporate it in a framework for case-based adaptation.[1] In such a framework, adaptations are performed on the basis of game observations drawn from a multitude of games. The effect of the game adaptations, therefore, can be inferred directly from previous observations that are gathered in the case base. We incorporate our adaptation mechanism in a framework for case-based adaptation, i.e., the adaptation mechanism adapts to game circumstances by exploiting observations gathered in the case base.

An important requirement, to this end, is an evaluation function that can rate adequately the state of a game. From results of experiments discussed in Chapter 5, we have already concluded that our evaluation function is able to predict accurately the outcome of a Spring game. Therefore, the established evaluation function is incorporated in the implementation of our adaptation mechanism.

In our research, we apply the adaptation mechanism foremost for the purpose of (A) adapting for effective behaviour, while aspiring that adaptations are performed reliably. An additional application of the adaptation mechanism is (B) difficulty scaling. Both applications are discussed next.

A. Adapting for effective behaviour

As indicated in Chapter 4, in a case-based framework the game AI may be expected to become robust in dealing with non-determinism, since the case base can be used as a model to predict the effect of game adaptations. For adapting game AI for effective behaviour in a reliable fashion, we propose that the adaptation mechanism is adjusting a pre-existing game AI (cf. Aha *et al.* (2005), Sugandh *et al.* (2008), and Ontañón *et al.* (2009)), on the basis of a case base drawn from observations of a multitude of games. We note that this is a contrast with most previous approaches to game AI, that do not tie in with a pre-existing game. That is, in those approaches the adaptation mechanism itself is 'being' the game AI. An advantage of the adaptation mechanism tying in with a pre-existing game AI, is that it enables game developers to control and predict with relative accuracy the behaviour that is exhibited by the game AI. In addition, the case base can be utilised for providing feedback on distinct strengths and weaknesses of a player, and can provide inexpensive insight into the balance of a game. Thus, it can help in testing and debugging the game.

A plain example of adapting for effective behaviour is as follows. Assume that a particular game developer prefers predictable game adaptations over highly effective game adaptations, and is considering to execute one of two game strategies. On the one hand, the case base indicates that in a similar game state executing strategy 'X' has previously led to the targeted outcome in 70 per cent of the cases. On the other hand, executing strategy 'Y' has previously led to an outcome slightly below the target value, but has effectively led to that

[1] We note that 'case based adaptation' differs from the frequently studied CBR task of 'case adaptation'. In the thesis, with case-based adaptation we refer to case-based learning for adapting game AI.

outcome in 90 per cent of the cases. Here, the previous observations that are gathered in the case-base enable the adaptation mechanism to make an informed decision on the preferred strategy. Considering the preference of the game developer for predictable game adaptations, the adaptation mechanism will select strategy 'Y'. We note that the preference for selecting game strategies is not inherent to our mechanism, but instead is free to be determined by the game developer.

B. Difficulty scaling

An adaptation mechanism may potentially be applied to adapt automatically the challenge that a game poses to the skills of a human player. This is called difficulty scaling (Spronck *et al.*, 2004a), or alternatively, challenge balancing (Olesen *et al.*, 2008). When applied to game AI, difficulty scaling aims usually at achieving an "even game", i.e., a game wherein the playing strength of the computer and the human player match.

In most games, the only implemented means of difficulty scaling is typically provided by a *difficulty setting*, i.e., a discrete parameter that determines how difficult the game will be. The purpose of a difficulty setting is to allow both novice and experienced players to enjoy the appropriate challenge that the game offers. Usually the parameter affects plain in-game properties of the game opponents, such as their physical strength. Only in exceptional cases the parameter influences the strategy of the opponents. Consequently, even on a "hard" difficulty setting, opponents may exhibit inferior behaviour, despite their, for instance, high physical strength. Because the challenge provided by a game is typically multifaceted, it is tough for the player to estimate reliably the particular difficulty level that is appropriate for himself. Furthermore, generally only a limited set of discrete difficulty settings is available (e.g., easy, normal, and hard). This entails that the available difficulty settings are not fine-tuned to be appropriate for each player.

In recent years, researchers have developed advanced techniques for difficulty scaling of game AI. Demasi and Cruz (2002) used coevolutionary algorithms to learn game characters that best fit the challenge level of a human player. Hunicke and Chapman (2004) explored difficulty scaling by controlling the game environment (i.e., controlling the number of weapons and power-ups available to a player). Spronck *et al.* (2004a) investigated three methods to adapt the difficulty of a game by adjusting automatically weights assigned to possible game strategies. In related work, Yannakakis and Hallam (2007) provided a qualitative and quantitative method for measuring player entertainment in real time.

Our adaptation mechanism offers a straightforward means to difficulty scaling. Generally, we prefer to use the adaptation mechanism that exploits the case base of game observations for the purpose of obtaining more effective play. Analogously, the adaptation mechanism can be applied for the purpose of obtaining a predefined target fitness value. For instance, the adaptation mechanism could exploit information of the fitness progress of previously observed, similar games, to obtain a draw position instead of a winning position.

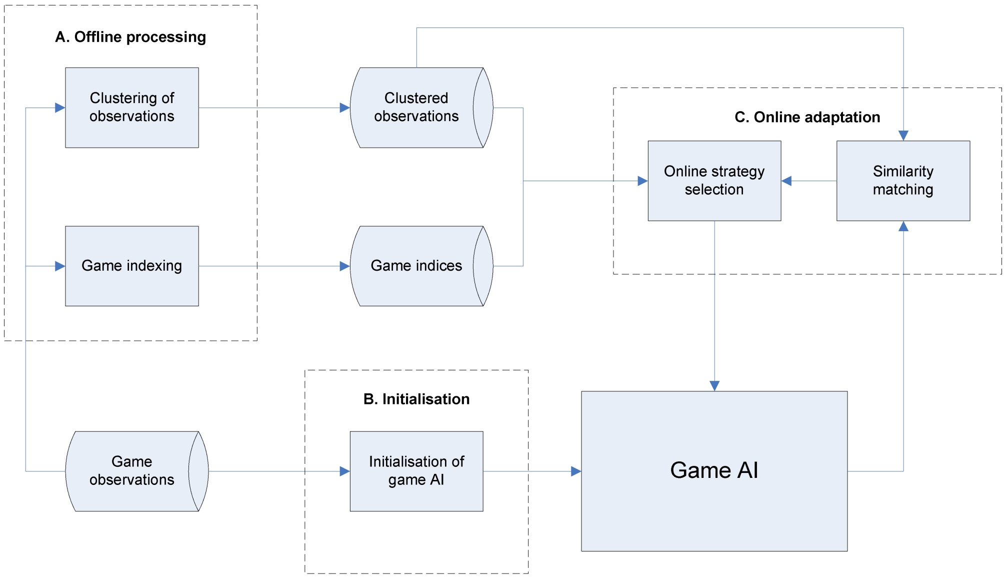


Figure 6.1: General adaptation procedure. The procedure consists of three steps, (A) offline processing, (B) initialisation, and (C) online adaptation. The three steps are composed of five components. The components are game indexing, clustering of observations, initialisation of game AI, similarity matching, and online strategy selection (see text for details).

6.2 An adaptation mechanism for SPRING

In this section we describe our adaptation mechanism for Spring. In case-based adaptive game AI, domain knowledge collected in a case base is exploited by the adaptation mechanism for adapting online the game AI to game circumstances.

The remainder of this section is organised as follows. First, we outline the general procedure of the adaptation mechanism (6.2.1). Subsequently, the mechanism is discussed in detail in Subsection 6.2.2 to 6.2.6, the topics being game indexing, clustering of observations, initialisation of game AI, similarity matching, and online strategy selection. The topics refer to five components of the adaptation mechanism.

6.2.1 General adaptation procedure

We define our adaptation mechanism as follows. It consists of three steps: (A) offline processing, (B) initialisation, and (C) online adaptation. The general adaptation procedure, illustrated in Figure 6.1, is described below.

In step A, game observations that are gathered in the case base are processed offline. The purpose of step A is to generalise over the gathered observations. The offline processing step incorporates components (1) to index gathered games (described in Subsection 6.2.2), and (2) to cluster observations (described in Subsection 6.2.3). We define a game observation as an observation on the state of the game. Of each game observation, a vector of feature data is gathered in the case base. This process is described in more detail in Subsection 6.3.1-A.

In step B, the initialisation of the game AI is performed. The purpose of step B is to ensure that game AI is effective from the onset of a game. To this end, the step incorporates one component which initialises the game AI with a previously observed, effective game strategy (described in Subsection 6.2.4). For the present experiments, we define a game strategy as the configuration of parameters of the game AI that determine strategic behaviour. The term 'opponent strategy' is used analogous to game strategy, to reflect that it concerns a game strategy that is employed by the opponent player. In the game AI that we will experiment with, we found 27 parameters that determine the game strategy of the game AI. The parameters affect the game AI's behaviour on a high, strategic level, and not on a low, tactical level. For example, the parameter 'aircraft_rate' determines on a high level how often aircraft units should be constructed. How exactly the constructed aircraft units should be employed is decided by lower-level game AI.[2] All 27 parameters are described in Appendix D.

In step C, the game AI is adapted online. The purpose of step C is to adapt the game AI in such a way that it exhibits behaviour that is effective in actual game circumstances. The online adaptation step incorporates components (1) to perform similarity matching (described in Subsection 6.2.5), and (2) to perform online strategy selection (described in Subsection 6.2.6). The component to perform similarity matching is supportive for the component to perform online strategy selection. In addition, the component to perform online strategy selection exploits the game indices that were established offline in step A, as well as the clustering of observations that was established offline in step A.

6.2.2 Game indexing

The component 'game indexing' is employed in step A. As calculating the game indices is computationally relatively expensive, as all stored game observations need to be processed, it is performed offline (in step A). The calculated game indices are exploited for online strategy selection (in step C).

We define a game's index as a vector of fitness values, containing one entry for each observed game state. The fitness values represent the desirability of the observed game states. To calculate the fitness value of an observed game state, we use the previously established evaluation function (see Chapter 5).

6.2.3 Clustering of observations

The component 'clustering of observations' is employed in step A. As clustering of observations is computationally relatively expensive, as all stored game observations need to be processed, it is performed offline (in step A). The established clustering of observations is exploited for online strategy selection (in step C).

[2] We surmise that adaptation of the game AI on a high, strategic level will affect most significantly the outcome of the game. Still, also on a low, tactical level the game AI may be adapted automatically, based on previous game observations. For instance, Szczepanski and Aamodt (2009) have already established a case-based reasoning approach to control the actions of individual units in the real-time strategy game WARCRAFT 3.

As an initial means to cluster similar observations, we apply the standard k-means clustering algorithm (Hartigan and Wong, 1979). The metric that expresses an observation's position in the cluster space is determined by the composed sum of observational features, that also is applied for similarity matching (see Subsection 6.2.5). Even though the k-means clustering algorithm is effective for our current setup, alternatives such as tree-indexing structures (e.g., *k*d-trees (Bentley, 1975) or cover trees (Beygelzimer *et al.*, 2006)) may be considered when working with increasingly large collections of cases. That is, we currently exploit hundreds of thousands observations. The number of observations may increase to millions, for instance, in massively multiplayer online games (MMOGs).

6.2.4 Initialisation of game AI

The component 'initialisation of game AI' is employed in step B. It concerns the selection of a game strategy that is adopted by the game AI at the start of the game. To select intelligently the strategy that is initially followed by the game AI, we need to determine which strategy the opponent player is likely to employ. To this end, we model opponent players based on actual game observations.

In the current experiments, we construct opponent models on the basis of observations of the parameter values of the opponent strategies, which indicate the strategic preferences of particular opponents. This might be considered "cheating", as we use information that, in general, is not available to a player. However, we adhere to the mentality and habits as employed in classic game matches. There, the accurate preparation on an opponent is part of the game (or model) and is never related to cheating. In our opinion the same holds for video games. Still, in experiments discussed in Chapter 8, we will assume that parameters that underlie opponent behaviour cannot be observed directly, and thus opponent models will have to be established via alternative techniques, such as statistical learning.

The considerations given above lead us to define the procedure to initialise the game AI as follows. First, determine the parameter values of the game strategy that is adopted by the opponent player. Second, determine in which parameter bands (Evans, 2002) the opponent strategy can be abstracted. We define three bands for each parameter, 'low', 'medium' and 'high'. Third, initialise the game AI with a strategy that was observed as effective against the most similar opponent. We consider a strategy effective when in previous play it achieved a predefined goal (thus, the game AI will never be initialised with a predictably ineffective strategy). Moreover, we consider opponents strictly similar when the abstracted values of the parameter bands are identical.

6.2.5 Similarity matching

The component 'similarity matching' is employed in step C. It is supportive for the component to perform online strategy selection. For selecting an effective game strategy, the similarity matching component compares directly the strategic similarity of game observations.

As a first step to match observations for similarity, the selection of the features and the weights assigned to each feature are determined by the researchers, to reflect their expertise with the game environment. It is a heuristically composed function, that works well when seen in the current stage of development of adaptive game AI. We admit that by only utilising our expert knowledge we may restrict the effectiveness of our implementation. However, the investigation of further improvements with regard to the selection of features is considered a topic for future research. To compare a given observation with another observation, we use six observational features to provide measures for strategic similarity, namely (1) phase of the game, (2) material strength, (3) commander safety, (4) positions captured, (5) economical strength, and (6) unit count. The first three features are also applied for establishing our evaluation function (see Chapter 5). Features four to six are incorporated to provide additional measures for strategic similarity. For convenience of the reader, in Appendix B we will provide an overview of all features that are used in the three main components of case-based adaptive game AI.

Our function to calculate the strategic similarity is defined by a composed sum. The terms concern the absolute difference in features values. By default, the features are assigned a weight of one. The first term is composed by the feature 'phase of the game' and 'unit count'. The feature 'unit count' is assigned a weight of 0.5, to reflect its lesser importance. The feature 'phase of the game' is incremented by a value of one, to enforce a positive feature value in the case that there is no difference in the phase of the game. As a result, this leads us to denote the function to calculate strategic similarity as follows.

$$\begin{aligned}
\text{similarity}(\text{obs1},\text{obs2}) \quad &= \quad (\ (1+\text{diff\_phase\_of\_the\_game}(\text{obs1},\text{obs2}))\\
&* \quad (0.5*\text{diff\_unit\_count}(\text{obs1},\text{obs2}))\ )\\
&+ \quad \text{diff\_material\_strength}(\text{obs1},\text{obs2})\\
&+ \quad \text{diff\_commander\_safety}(\text{obs1},\text{obs2})\\
&+ \quad \text{diff\_positions\_captured}(\text{obs1},\text{obs2})\\
&+ \quad \text{diff\_economical\_strength}(\text{obs1},\text{obs2})
\end{aligned} \tag{6.1}$$

In Subsection 6.2.3 we noted that game observations are clustered. As a result, calculating the similarity between clustered observations is computationally relatively inexpensive. This is important, as similarity matching is performed online (in step C).

6.2.6 Online strategy selection

The component 'online strategy selection' is employed in step C. It concerns selecting online which strategy to employ in actual play. Online strategy selection is performed at every phase transition of the game. The selection procedure consists of three steps.

First, we preselect the N games in the case base that are most similar to the current game. To this end, we use the computed game indices to preselect the games with the smallest accumulated fitness difference with the current game, up until the current game state.

Second, from the preselected N games, we select the M games that satisfy a particular goal criterion. The goal criterion can be any metric to represent preferred behaviour. In our experiments, the goal criterion is a desired fitness value. For instance, a desired fitness value of one hundred represents a significant victory, and a fitness value of zero represents a draw situation for the players, which may be considered balanced gameplay.

Third, of the selected M games, we perform the strategy of the game observation that is most similar to the current game state in terms of strategic features. The similarity of observations is determined via the similarity function that is outlined in Subsection 6.2.5.

We note that we consider that performing strategies associated with similar observations may not necessarily yield the same outcome when applied to the current state. It may happen that observations that are strategically similar may result from distinct circumstances earlier in the game. Therefore, to estimate the effect of performing the retrieved game strategy, we measure the difference in fitness values between the current and the selected observation, and straightforwardly adjust the expected fitness value. For instance, we consider (1) that after playing the game for a certain amount of time, the fitness value of the current game is -5, (2) that the fitness value of a similar game at that same time was +5, and (3) that the game strategy followed in a similar game resulted ultimately in a fitness value of +10 when the game had finished. In such a situation, we estimate that applying the game strategy of the particular similar game will result ultimately in a fitness value of $-5 + (10 - 5) = 0$.

6.3 Validating the adaptation mechanism

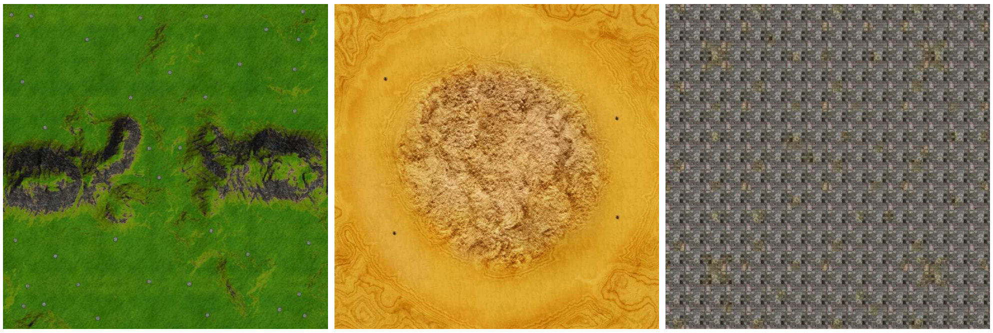
This section reports on two experiments that validate our case-based based adaptation mechanism in the Spring game. We are foremost interested in the performance of the adaptation mechanism. In the experiments, we therefore measure the performance of the adaptation mechanism in actual play. The performance is expressed by the ability of the adaptation mechanism to endow an adaptive game AI with effective behaviour in the Spring game. For an extensive description of Spring, we refer the reader to Appendix A.2.

The remainder of this section is organised as follows. We first describe the experimental setup (6.3.1). Subsequently, we discuss how the performance of the adaptation mechanism is assessed (6.3.2). Next, we give the experimental results of the first experiment (6.3.3) and of the second experiment (6.3.4). Finally, the section is concluded by a discussion of the results (6.3.5).

6.3.1 Experimental setup

The general experimental setup can be described in two parts. (A) In the case base we gather feature data of a multitude of games. (B) We exploit data gathered in the case base in two experiments that test the adaptation mechanism in the Spring game. In experiment one, we test the ability of the mechanism to establish effective behaviour by performing game adaptation. In experiment two, we test the ability of the mechanism to provide a straightforward form of difficulty scaling (i.e., in our experiment, uphold a draw position).

The two steps, viz. (A) gathering feature data, and (B) testing the adaptation mechanism, are described in more detail below. As the step 'gathering feature data' was already given in Subsection 5.3.1, in the description that follows we will focus on the differences of the step for the current experiments.



(a) SmallDivide (b) TheRing (c) MetalHeckv2

Figure 6.2: The three maps that were used in our experiments.

A. Gathering feature data

We collect data of the defined observational features (see Subsection 6.2.5) from Spring games in which two game AIs are pitted against each other. As friendly player, we use the 'AAI (cb)' game AI. As opponent player, we use the 'AAI (original)' game AI.

Feature data is collected on three different RTS maps (see Figure 6.2). The three maps are (a) SmallDivide, (b) TheRing, and (c) MetalHeckv2. All maps are virtually symmetrical and have no water areas. The map SmallDivide, illustrated in Figure 6.2(a), is the default map of the Spring game, and has one choke point in the centre of the map. The map TheRing, illustrated in Figure 6.2(b), is a map with an impassable mountain in the centre of the map. The map MetalHeckv2, illustrated in Figure 6.2(c), is a map without significant obstacles, that in addition is abundant with in-game metal resources. For each of the three maps, we gather data from 325 observed games that are played on the particular map.

For gathering feature data, we simulate competition between different players. This is performed for each game by pseudo-randomising the 27 strategic parameters of each player of the game (see Subsection 6.2.1). The pseudo-randomisation results in the players following a randomly generated strategic variation of an effective strategy.

We submit two considerations. First, the amount of offline storage should be relatively low for a case-based approach to be regarded practical. That is, low with respect to the computational power of the system on which the game runs. Second, as the selection of the game features plays an important role in the effectiveness of the adaptation mechanism, we should allow for *a posteriori* re-definition and selection of features. In acknowledgement of these two considerations, we decided to store in the case base *raw game observations*, from which feature data can be derived directly. The raw game observations are stored in a lightweight fashion, by only abstracting the position and unit type of each unit for each game observation. This abstraction, of approximately 3 KB per observation, allows for deriving a plethora of additional observational features. Accordingly, a case base was built from

Map	Games in case base	Obs. in case base	Data size
SmallDivide	325	213,005	650 MB
TheRing	325	128,481	341 MB
MetalHeckv2	325	107,081	201 MB
Total	975	448,567	1192 MB

Table 6.1: Contents of the case base.

448,567 observations of 975 games (three times 325 games), resulting in 1192 MB of uncompressed observational data.[3]

An overview of the contents of the case base is given in Table 6.1. All games from which observations are gathered are played under identical conditions. We observe that the amount of gathered observations depends on the structure of the map. For instance, due to the choke point in the centre of the SmallDivide map, games on this map generally take a relatively long time to finish.

B. Testing the adaptation mechanism

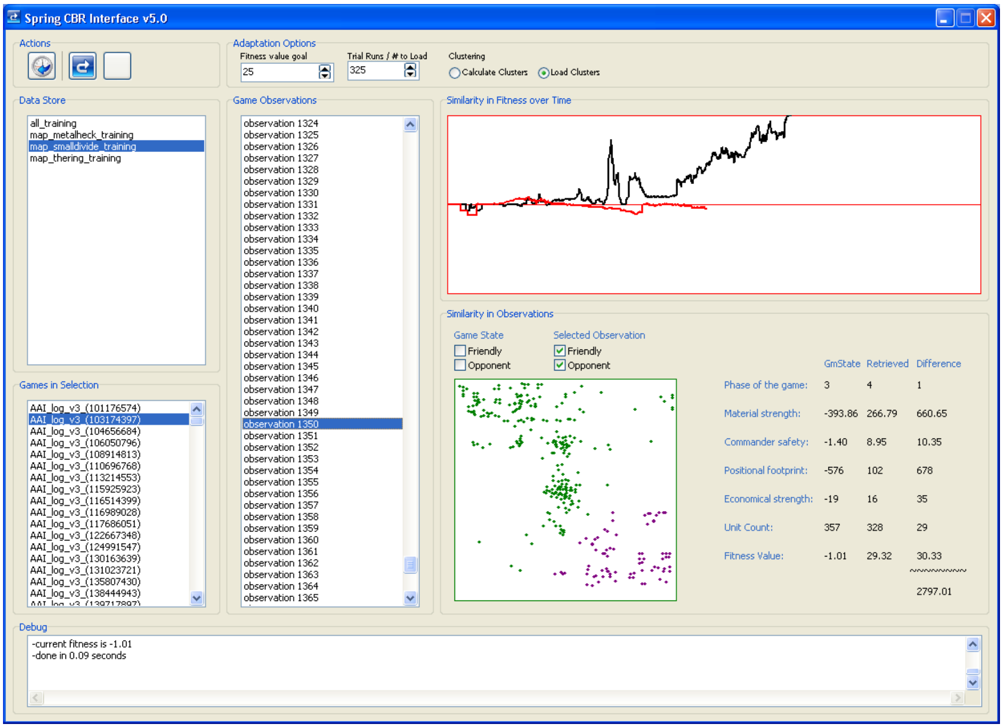
To assess the performance of the adaptation mechanism, we determine to what extent the mechanism is capable of adapting effectively to game circumstances in the Spring game. To this end, the adaptation mechanism exploits game observations that are gathered in the case base. We perform two different experiments with the adaptation mechanism.

In the first experiment, we test to what extent the mechanism is capable of adapting effectively to behaviour of the opponent player. The experiment is performed in play where the adaptive 'AAI (cb)' game AI is pitted against two types of opponent, (1) against the original AAI opponent, and (2) against a random opponent. For play against the latter type of opponent, the adaptive game AI is pitted against the original AAI opponent that is initialised with a randomly generated strategy. That is, in each trial of the experiment, a variation of an effective strategy is generated randomly. On each of the three RTS maps (i.e., SmallDivide, TheRing, and MetalHeckv2) we perform 150 adaptation trials for play against each two types of opponent. All adaptation trials are performed under identical conditions.

In the second experiment, we test to what extent the mechanism is capable of upholding a draw position. The experiment is performed in play where the adaptive 'AAI (cb)' game AI is pitted against the same two types of opponent as the first experiment. The second experiment is performed on the default map of the Spring game, the map SmallDivide. On the map, we perform 150 adaptation trials for play against each two types of opponent. All adaptation trials are performed under identical conditions.

Detailed experimental settings that apply for both experiments are as follows. For offline clustering of observations, k is set to ten per cent of the total number of observations.

[3] Approaches are available to keep reducing the size of the case base, such as offline data compression and subsequent online data decompression (Abou-Samra *et al.*, 2002), and automatic condensation of the case base (Angiulli and Folino, 2007). However, incorporating these approaches lies outside the scope of the present research.


Spring CBR Interface v5.0

Actions

Adaptation Options
Fitness value goal: 25
Trial Runs / # to Load: 325
Clustering: ( ) Calculate Clusters [✓ Load Clusters]

Data Store
all_training
map_metalheck_training
map_smalldivide_training
map_thering_training

Games in Selection
AAI_log_v3_(101176574)
AAI_log_v3_(103174397)
AAI_log_v3_(104656684)
AAI_log_v3_(106050796)
AAI_log_v3_(108914813)
AAI_log_v3_(110696768)
AAI_log_v3_(113214553)
AAI_log_v3_(115925923)
AAI_log_v3_(116514399)
AAI_log_v3_(116989028)
AAI_log_v3_(117686051)
AAI_log_v3_(122667348)
AAI_log_v3_(124991547)
AAI_log_v3_(130163639)
AAI_log_v3_(131023721)
AAI_log_v3_(135807430)
AAI_log_v3_(138444943)

Game Observations
observation 1324
observation 1325
observation 1326
observation 1327
observation 1328
observation 1329
observation 1330
observation 1331
observation 1332
observation 1333
observation 1334
observation 1335
observation 1336
observation 1337
observation 1338
observation 1339
observation 1340
observation 1341
observation 1342
observation 1343
observation 1344
observation 1345
observation 1346
observation 1347
observation 1348
observation 1349
observation 1350
observation 1351
observation 1352
observation 1353
observation 1354
observation 1355
observation 1356
observation 1357
observation 1358
observation 1359
observation 1360
observation 1361
observation 1362
observation 1363
observation 1364
observation 1365

Similarity in Fitness over Time

Similarity in Observations
Game State: ☐ Friendly ☐ Opponent
Selected Observation: ☑ Friendly ☑ Opponent

	GmState	Retrieved	Difference
Phase of the game:	3	4	1
Material strength:	-393.86	266.79	660.65
Commander safety:	-1.40	8.95	10.35
Positional footprint:	-576	102	678
Economical strength:	-19	16	35
Unit Count:	357	328	29
Fitness Value:	-1.01	29.32	30.33
			~~
			2797.01

Debug
```
-current fitness is -1.01
-done in 0.09 seconds
```

Figure 6.3: Screenshot of the Spring CBR interface. In the screenshot, the current game state is compared with states of previously observed games that are gathered in the case base. The top-right panel displays the similarity in fitness value throughout the course of the compared games. The bottom-right panel displays the strategic similarity of the selected game states.

Before the game starts, the initial strategy is determined. Online (i.e., while the game is in progress) strategy selection is performed at every phase transition. The parameter N for online strategy selection is set to 50, and the parameter M is set to 5 (see Subsection 6.2.6).[4]

The Spring CBR Interface, which we developed to interface between the Spring game and the case base, is illustrated in Figure 6.3. The interface allows the adaptation mechanism to exploit the case base, and enables the mechanism to affect directly the behaviour of 'AAI (cb)' game AI that is operating in the Spring game.

[4] Offline processing of the case-base takes approximately 2 minutes, excluding clustering of observations. One-time only clustering of observations takes approximately 36 minutes. Online strategy selection takes approximately 0.1 seconds. Experiments are performed on a PC built around an Intel Core 2 Quad CPU @ 2.40 GHz, with 2 GB of RAM.

6.3.2 Performance assessment

To establish a baseline for comparing the experimental results, both experiments are performed in a setting where the adaptation mechanism is disabled. In this setting, the game AI does not intelligently determine the initial strategy, but instead randomly selects the initial strategy, and performs no online adaptation to game circumstances.

For the first experiment, we set the adaptation mechanism to win the game (i.e., obtain a positive fitness value). In this experiment, the effectiveness of the mechanism is expressed by the number of games that are won by the friendly player when it uses the adaptation mechanism.

For the second experiment, we set the adaptation mechanism to uphold a draw position. In this experiment, the effectiveness of the mechanism is expressed by the amount of time that a draw can be upheld by the player that uses the adaptation mechanism. We consider a game state strictly a draw when its fitness value is zero. Naturally, we need to incorporate some flexibility in our working definition of a draw, as in RTS games players perform actions in an asynchronous manner (e.g., player 1 may slightly delay performing the same, effective actions as player 2, and as a result may seem weaker for a short period of time). In addition, we make the assumption that also human players incorporate a certain flexibility in determining when a game is tied. Using our expert knowledge with the game environment, this leads us to define an ongoing game as tied, when the friendly player has never obtained a fitness value less than -10, and has never obtained a fitness value greater than +10.

6.3.3 Exp1: Results of game adaptation

In this subsection, we give the results of the first experiment. A general discussion of the obtained results is provided in Subsection 6.3.5.

Table 6.2 gives an overview of the baseline results of the first experiment performed in the Spring game, obtained with disabled adaptation mechanism. The first column of each table lists against which type of opponent the game AI was pitted. The second column lists how often the trial was repeated. The third and fourth column list how often the goal of winning the game was achieved in absolute terms, and in terms of percentage, respectively.

The baseline results reveal that on the map SmallDivide, the game AI with disabled adaptation mechanism is able to win 39% of the games played against the original AAI opponent. This indicates that on the map SmallDivide the original AAI opponent is initialised with effective behaviour. In contrast, on the map TheRing, the game AI with disabled adaptation mechanism is able to win 60% of the games played against the original AAI opponent. This indicates that on the map TheRing the orginal opponent is easier to defeat, compared to play on the map SmallDivide. On the map MetalHeckv2, the effectiveness of the game AI with disabled adaptation mechanism is 47%. This indicates that, by approximation, on this map both players are equally effective.

In addition, the baseline results reveal that in play against randomly generated opponents on the map SmallDivide and TheRing, the effectiveness of the game AI approximates 50%. As both players follow a randomly selected strategy, this result may be expected. However, on the map MetalHeckv2, the effectiveness of the game AI is substantially less than 50%. It is difficult to pin down a precise explanation for why on this map the effectiveness is not

SmallDivide

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	59	39%
Random	150	71	47%

TheRing

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	90	60%
Random	150	76	51%

MetalHeckv2

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	70	47%
Random	150	54	36%

Table 6.2: Performances by AAI(cb); baseline effectiveness with disabled adaptation mechanism (experiment 1).

roughly 50%. We surmise that the effectiveness is hampered by certain low-level AI effects that, in randomised play on the map, are not influenced by adapting the high-level AI parameters. This means, while the starting positions might seem equal from the perspective of a human player, the low-level AI might be biased in being more effective at starting, for instance, at the top of the map rather than at the bottom of the map.

Table 6.3 gives an overview of the results of the first experiment performed in the Spring game, obtained with enabled adaptation mechanism. The legend of Table 6.3 is equal to that of Table 6.2. Figure 6.4 displays the obtained fitness value as a function over time of two typical experimental runs on the map SmallDivide. Figure 6.5 displays the obtained median fitness value over all game trials against the original AAI opponent on the map SmallDivide, as a function over the relative game time.

The results reveal that when pitted against the original AAI game AI, the adaptation mechanism improves significantly on the established baseline effectiveness on the map SmallDivide (77%, compared to the baseline 39%) (cf. chi-square test, Cohen, 1995). In addition, the adaptation mechanism improves substantially on the established baseline effectiveness on the map TheRing (81%, compared to the baseline 60%). Subsequently, the adaptation mechanism improves significantly on the established baseline effectiveness on the map MetalHeckv2 (83%, compared to the baseline 47%) (cf. chi-square test, Cohen, 1995). These results indicate that the adaptation mechanism is generically effective in play against the original AAI game AI.

In addition, the results reveal that in play against randomly generated opponents, the adaptation mechanism obtains an effectiveness of 64% on the map SmallDivide. Thereby, it improves on the established baseline effectiveness of 47%. This improvement in effectiveness is consistent with our findings on the map TheRing, where the adaptation mechanism

SmallDivide

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	115	77%
Random	150	96	64%

TheRing

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	122	81%
Random	150	93	62%

MetalHeckv2

Opponent	Trials	Goal achv.	Goal achv. (%)
Original AAI	150	124	83%
Random	150	60	40%

Table 6.3: Performances by AAI(cb); effectiveness with enabled adaptation mechanism (experiment 1).

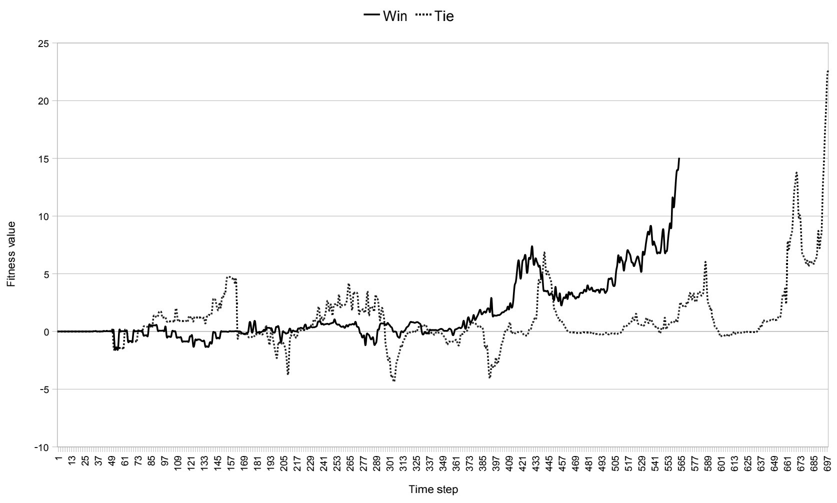


Figure 6.4: Obtained fitness values as a function over time, when pitted against the original AAI game AI on the map SmallDivide. The figure displays a typical experimental result of (1) the adaptation mechanism set to win the game, and (2) the adaptation mechanism set to uphold a draw (indicated at the top of the figure). A time step equals 4.233 seconds (cf. Subsection 5.3.1).

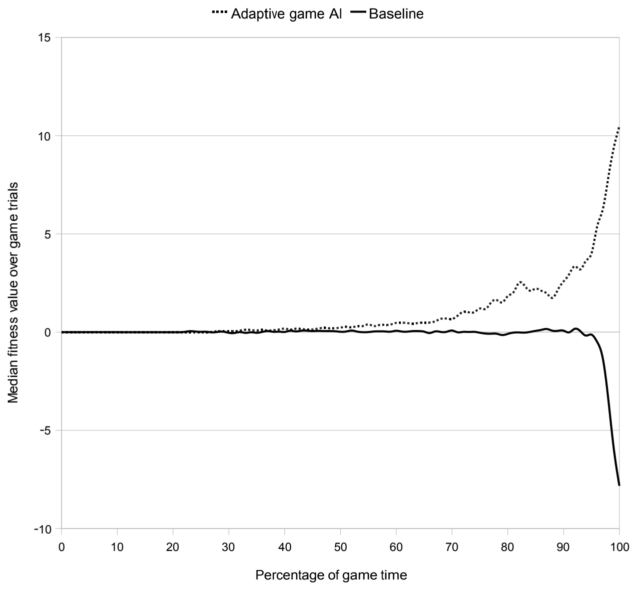


Figure 6.5: Median fitness value over all game trials against the original AAI opponent on the map SmallDivide, as a function over the relative game time.

obtains an effectiveness of 62% (compared to the baseline 51%). Against randomly generated opponents on the map MetalHeckv2, the adaptation mechanism obtains an effectiveness of 40% (compared to the baseline 36%). These results indicate that even in randomised play, the adaptation mechanism is able to increase the effectiveness of game AI.

6.3.4 Exp2: Results of difficulty scaling

In this subsection, we give the results of the second experiment. A general discussion of the obtained results is provided in Subsection 6.3.5.

Table 6.4 gives an overview of the baseline results of the second experiment, obtained with disabled adaptation mechanism. The first column of each table lists against which opponent the game AI was pitted. The second column lists how often the trial was repeated. The third column lists the average time to uphold a draw position, and, between brackets, the accompanying standard deviation of the obtained result.

The baseline results reveal that without applying difficulty-scaling techniques, the game AI will on average uphold a draw position for approximately 27 minutes when it is pitted against the original AAI opponent. When the game AI is pitted against randomly generated opponents, the game AI will uphold a draw position for approximately 28 minutes.

SmallDivide		
Opponent	Trials	Time to uphold draw
Original AAI	150	26.98 min (5.98 min)
Random	150	27.56 min (7.78 min)

Table 6.4: Performances by AAI(cb); baseline effectiveness upholding a draw (experiment 2).

SmallDivide		
Opponent	Trials	Time to uphold draw
Original AAI	150	37.37 min (16.72 min)
Random	150	36.23 min (17.64 min)

Table 6.5: Performances by AAI(cb); upholding a draw with the adaptation mechanism (experiment 2).

Table 6.5 gives an overview of the results of the second experiment, obtained with enabled adaptation mechanism. The legend of Table 6.5 is equal to that of the table with the baseline results.

The results reveal that when pitted against the original AAI opponent, the adaptation mechanism improves significantly on the time in which a draw is upheld (37 minutes, compared to the baseline 27 minutes) (cf. t-test, Cohen, 1995). The typical result given in Figure 6.4, reveals that a draw can be upheld for a sustained period of time. At certain point in time, inevitably, the game AI will no longer be able to compensate play of the opponent, and the game will either be won or lost by the game AI. Comparable performance is obtained when the adaptation mechanism is pitted against opponents with randomly generated strategies. The results reveal that when pitted against opponents with randomly generated strategies, the adaptation mechanism improves significantly on the time in which a draw is upheld (36 minutes, compared to the baseline 28 minutes) (cf. t-test, Cohen, 1995).

6.3.5 Discussion of the results

In the first experiment that tested our implementation of the adaptation mechanism, we observed that game AI that incorporated the mechanism was well able to achieve a victory when pitted against the original AAI game AI. We noticed that the adaptation mechanism was able to find in the case base strategies that could effectively defeat the original AAI game AI. As the original AAI game AI is not able to adapt its behaviour, the adaptation mechanism could exploit its discovery indefinitely. Note that in some cases, the adaptive game AI did not win the game, despite it exhibiting strong behaviour. Such outliers cannot be avoided due to the inherent randomness that is typical to video games.[5]

In addition, we observed that even in play against randomly generated opponents, the adaptation mechanism is generally able to find effective strategies in the case base, and was thereby capable of improving on the baseline performance. As play against randomly generated opponents may be considered a simulated way to test the game AI against previously unobserved opponents, this is a satisfactory result. Naturally, the question remains how the performance in randomised play can be further enhanced. We discuss two possible ways to enhance the performance in play with randomly generated opponents, (1) gather more games in the case base, and (2) incorporate opponent modelling.

First, note that for each map our case base currently consists of observations that were collected over 325 games. For randomised play, determined by 27 pseudo-randomised behavioural parameters (see Subsection 6.2.1), it would be beneficial to collect more games in the case base in order to increase the probability of it containing effective game strategies. As case-based adaptive game AI can be expected to be applied in the playtesting phase of game development, and predictably in multi-player games, the case base in practical applications is expected to grow rapidly to contain a multitude of effective strategies.

Second, we observed that the final outcome of a Spring game is largely determined by the strategy that is adopted in the beginning of the game. This exemplifies the importance of initialising the game AI with effective behaviour. In order to do so, a player needs to determine accurately the opponent against whom it will be pitted. We assume that in video-game practice, (human) game opponents do not exhibit behaviour as random as in our experimental setup, but will exhibit behaviour that can be abstracted into a limited set of opponent models. Therefore, on the condition that accurate models of the opponent player can be established, we will follow the expert opinion that game AI should not so much be focussed on directly exploiting current game observations, but should rather focus on effectively applying models of the opponent in actual game circumstances (Rabin, 2008).

In the second experiment that tested our implementation of the adaptation mechanism, we observed that the adaptation mechanism is able to uphold a draw for a sustained period of time. This ability may be regarded as a straightforward form of difficulty scaling. If a metric can be established that represents the preferred level of challenge for the human player, then in theory our adaptation mechanism is capable of scaling the difficulty level to the human player. Such a capability provides an interesting topic for future research.

6.4 Chapter conclusions

In this chapter we reported on our adaptation mechanism for video game AI. The mechanism aims at allowing game AI to adapt rapidly and reliably to game circumstances. To this

[5] We note that in the Spring game, the most powerful unit is able to destroy a Commander unit with a single shot. Should the Commander be destroyed in such a way, the question would arise if this was due to bad luck, or due to an effective strategy by the opponent. For game AI to be accepted as effective players, one could argue, recalling the previously mentioned need for consistent AI behaviour, that game AI should not force a situation that may be regarded as the result of lucky circumstances.

end, it is incorporated in a framework for case-based adaptation (cf. Section 6.1). The adaptation mechanism exploits game observations that are gathered in the case base to (A) generalise offline over observations, (B) initialise the game AI with a predictably effective game strategy, and (C) adapt online the game AI to game circumstances. The adaptation mechanism was tested on three different maps in the Spring game environment. Our experimental results show that the adaptation mechanism can successfully obtain effective performance. In addition, the adaptation mechanism is capable of upholding a draw for a sustained period of time. From these results, we may conclude that the mechanism for case-based adaptation of game AI provides a strong basis for adapting rapidly and reliably the player's behaviour in an actual, complex video game: Spring.

Moreover, we noted that to play a video game adequately, it is important to be able to model the opponent player, and utilise effectively the models in actual game circumstances. In the next chapter we will therefore investigate opponent modelling for video games.

7 Opponent modelling

In this chapter we concentrate on opponent modelling, one of the three main components of case-based adaptive game AI. Opponent modelling is a technique that enables game AI to establish and exploit models of the opponent player. The goal of opponent modelling is to improve the decision-making capabilities of game AI, by allowing it to adapt to the playing strategy of the opponent player. In modern, complex video games, establishing and exploiting effectively opponent models is a difficult task.

Chapter 7 is organised as follows. We first describe the concept of opponent modelling (Section 7.1). Subsequently, we report on two experiments that concern establishing models of the opponent player in a complex RTS game (Section 7.2). Then, we report on an experiment that concerns exploiting models of the opponent player in a complex RTS game (Section 7.3). Finally, we present the chapter conclusion (Section 7.4).

7.1 Opponent modelling

Opponent modelling is an important research area in game playing. It concerns establishing models of the opponent player, and exploiting the models in actual play. Opponent modelling has also received attention outside the domain of games, for instance in military simulations (see, e.g., Wray and Laird, 2003) and intrusion detection (see, e.g., Yampolskiy, 2007). In general, an opponent model is an abstracted description of a player or of a player's behaviour in a game. The goal of opponent modelling is to improve the capabilities of the artificial player by allowing it to adapt to its opponent and exploit his weaknesses (Carmel and Markovitch, 1993; Iida *et al.*, 1993; Donkers *et al.*, 2001; Donkers, 2003). Even if a game-theoretical optimal solution to a game is known, a computer program that has the capability

This chapter is based on the following two publications.

1) Schadd, F. C., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Opponent modeling in real-time strategy games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 61–68. EUROSIS-ETI, Ghent University, Ghent, Belgium.

2) Mehlkop, B. (2009). Adaptive game AI using opponent modelling. Master's thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

to model its opponent's behaviour may obtain a higher reward. A recent example that illustrates the importance of opponent modelling, derived from Fürnkranz (2007), is as follows.

Consider, the game of roshambo (also known as rock-paper-scissors). Assume both players play their optimal strategies (i.e., randomly select one of their three moves). In this case either player can expect to win one third of the games (with one third of the games drawn). However, against an opponent that always plays rock, a player that is able to adapt his strategy to always playing paper can maximize his reward, while a player that sticks with the 'optimal' random strategy will still win only one third of the games.

The remainder of this section is organised as follows. First, we discuss concisely opponent modelling in classic games (7.1.1). Subsequently, we discuss concisely opponent modelling in video games (7.1.2).

7.1.1 Opponent modelling in classic games

In classic games, opponent modelling has as its main goal to improve the performance of the own (artificial) player (Van den Herik *et al.*, 2005). The objective is to exploit the opponent's weaknesses. Better game results are positively correlated with a higher playing strength. Computer programs that play classic games generally incorporate search techniques to find possible game actions by the opponent, of which a model can be constructed. As a result, the role of opponent modelling in classic games is to *guide the search process* towards improved game results.

In the remainder of this subsection, we provide (A) a brief history of opponent modelling in classic games, and describe (B) the state of the industry with regard to incorporating opponent-modelling techniques.

A. History

In the domain of classic games, opponent modelling is a research topic that was envisaged already a long time ago. Van den Herik *et al.* (2005) observe that, for instance, in the 1970s chess programs incorporated a contempt factor, meaning that against a stronger opponent a draw was accepted even if the player was +0.5 ahead, and a draw was declined against a weaker opponent even when the player had a minus score.

The first attempt to opponent modelling in classic games was taken by Slagle and Dixon (1970), who incorporated rudimentary knowledge of the opponent in the search process. For instance, such knowledge can concern assumptions on the fallibility of an opponent (Reibman and Ballard, 1983); game AI can consider the chance that the opponent performs a non-rational game action. In related work, Jansen (1992, 1993) investigated using knowledge about the opponent in game-tree search.

Research specifically focussed on the topic of opponent-modelling search started in 1993. In that year, two research groups, one in Haifa, Israel and one in the Maastricht, the Netherlands, simultaneously developed a search method that took knowledge of the opponent player into account. They both called it: opponent-model search. In Israel, Carmel and Markovitch (1993, 1998) investigated in depth the learning of models of opponent strategies. In the Netherlands, Iida *et al.* (1993) investigated potential applications of opponent-model search. An extensive description of the history of opponent modelling is given by Donkers (2003).

In the year 1994, Uiterwijk and Van den Herik (1994) developed a search technique to "speculate" on the fallibility of the opponent player. In the 2000s, Donkers *et al.* (2001) and Donkers (2003) defined probabilistic opponent models, that attempted to avoid the pitfalls of opponent modelling by incorporating the player's uncertainty about the opponent's strategy.

B. State of the industry

The realisation of most ideas concerning opponent modelling is still in its infancy. There are three successful instances of actual implementation, viz. (1) roshambo (Egnor, 2000), (2) iterated prisoner's dilemma (Kendall, 2005), and (3) poker (Billings, 2006). Still, there is a wealth of techniques that are waiting for implementation in actual games (Van den Herik *et al.*, 2005).

7.1.2 Opponent modelling in video games

Opponent modelling is of increasing importance in modern video games (Fürnkranz, 2007). In video games, opponent modelling has as its main goal raising the entertainment factor (instead of raising the playing strength) (Van den Herik *et al.*, 2005). In the remainder of this subsection, we subsequently describe (A) the roles of opponent modelling in video games, (B) the challenges of opponent modelling in video-game environments, (C) approaches applicable for opponent modelling in video games, and (D) the state of the industry with regard to incorporating opponent-modelling techniques.

A. Roles of opponent modelling

In order the increase the entertainment factor of a video game, game AI that incorporates opponent modelling may fulfil two roles: (1) as a companion, and (2) as an opponent. Each role entails distinct requirements for the game AI. A description of the two roles is given next. The description is derived from a review article by Van den Herik *et al.* (2005), to which we refer the reader for more information on the topic.

Companion role: In the companion role, the game AI must behave according to the expectations of the human player. For instance, when the human player prefers an inconspicuous approach to dealing with opponent characters (e.g., by attempting to maintain undetected), he will not be pleased when the computer-controlled companions immediately attack every opponent character that is near. If the companions fail to predict with a high degree of success what the human player desires, they will likely annoy the human player, which is detrimental for the entertainment value of the game.

Opponent role: In the opponent role, the game AI must be able to match the playing skills of the human player, and respond adequately to the player's playing style. This is a difficult task. Research has shown that when the opponent characters play too weakly a game against the human player, the human player loses interest in the game (Scott, 2002). In addition, research has shown that when the opponent characters play too strong a game against the human player, the human player gets stuck in the game and will quit playing too (Livingstone and Charles, 2004; Van Lankveld *et al.*, 2009).

B. Challenges

Houlette (2004) discussed the challenges of opponent modelling in video-game environments, and suggested some possible implementations of opponent modelling. A challenge for opponent modelling in video games is that models of the opponent player have to be established (1) in game environments that generally are relatively realistic and relatively complex, (2) with typically little time for observation, and (3) often with only partial observability of the environment.

Once opponent models are established, classification of the opponent player has to be performed in real time. Other computations, such as rendering the game graphics, have to be performed simultaneously. Researchers estimate that generally only 20% of all computing resources are available to the game AI (Millington, 2006). Of these 20%, a large portion will be spent on rudimentary AI behaviour, such as manoeuvring game characters within the game environment. This implies that only computationally inexpensive approaches to opponent modelling are suitable for incorporation in the game AI.

C. Applicable approaches

In video games, a common approach to establishing models of the opponent player is by modelling the *actions* of the player (Donkers and Spronck, 2006), for instance, by using *n*-grams (Laramée, 2002). An alternative approach is to model the *preferences* of opponent players, instead of the actions resulting from those preferences. This preference-based approach is viable (Donkers and Spronck, 2006) and identifies the model of an opponent by analysing the opponent's choices in predefined game states.

In the preference-based approach, opponent modelling can be seen as a classification problem, where an opponent is classified as one of a number of available models based on data that is collected during the game. Behaviour of the game AI is established based on the classification of the opponent. Modelling of preferences may be viewed as similar to approaches that regard known opponent models (1) as stereotypes, and (2) as an abstraction of observations (Denzinger and Hamdan, 2004). As a means to generalise over observed game actions, in the remainder of the chapter, we follow the preference-based approach.

D. State of the industry

In recent years there have been several successful implementations of opponent modelling. For instance, Rohs (2007) was able to model accurately the preferences of opponent players in the game Civilization IV. Yannakakis and Hallam (2009) investigated the modelling of opponent players, for the purpose of optimising player satisfaction, and Sailer *et al.* (2008) incorporated opponent modelling to enhance simulation-based planning in RTS games. In addition, researchers designed techniques to predict the position of opponent players in first-person shooters (Darken and Anderegg, 2008), and in the game World of Warcraft (Valkenberg, 2007). Laviers *et al.* (2009a,b) investigated improving AI performance through opponent modelling in the Rush 2008 football simulator. In the related domain of interactive storytelling, Thue *et al.* (2008) investigated how models of the opponent player can be applied to create stories that can be adapted to fit individual players.

7.2 Establishing opponent models in SPRING

In a typical RTS game such as Spring, an important factor that influences the choice of strategy, is the strategy that is employed by the opponent player. For instance, if one knows what types of units the opponent has, then typically one would choose to build units that are (considered as) strong against those of the opponent. To make predictions about the opponent's strategy, an AI player can establish an opponent model. In this section we will discuss to what extent models of the opponent's strategy can be established in the Spring game.[1]

In the remainder of this section, we first discuss the so-called hierarchical approach to establishing opponent models in the Spring game (7.2.1). Subsequently, we describe our implementation of the approach (7.2.2). Next, we report on two experiments that test the implementation (7.2.3). Finally, we provide a discussion of the obtained results (7.2.4).[2]

7.2.1 Approach to establishing opponent models

A straightforward preference-based approach to opponent modelling in Spring is the following (cf. Schadd *et al.*, 2007). First, establish a predefined number of possible opponent models. Subsequently, in actual play, track the confidence value of the game observations resulting from the behaviour that is modelled by each opponent model. Finally, the observed behaviour is classified as resulting from the particular opponent model that obtained the highest confidence value.

An enhancement to the straightforward approach is to apply an hierarchical ordering on the possible opponent models (Houlette, 2004). This so-called *hierarchical approach* allows the division of a relatively complicated classification problem into several relatively straightforward classification problems. In addition, the hierarchical approach makes it possible to use different classification methods at each level of the hierarchy. For establishing opponent modelling in Spring, we follow the hierarchical approach. Indeed, we admit that by predefining the number of possible opponent models, we may restrict the accuracy of the classification. The investigation of further improvements is reported on in Chapter 8.

The opponent models that we will establish, will model the strategy that is employed by the opponent player. For the present experiments, we define a strategy as *the general playing*

[1] In the current section, the topic of 'establishing' opponent models would be relatively aimless without the incorporation of a means to validate the accuracy of the models. Therefore, in our investigation we focus also on the topic of validating the models by means of 'classifying' opponent behaviour in actual play. We note that strictly speaking, classification of models falls under the topic 'exploiting' the established opponent models.

[2] The work reported on in this section is performed by Schadd *et al.* (2007), under supervision of the author.

style combined with the opponent's choice of units built. The most characteristic element of an opponent's strategy is the general playing style. We therefore place the general playing style at the top of the hierarchy. Each general playing style has its own submodels that further determine behavioural characteristics of the style.

7.2.2 Implementation of the approach

In this subsection we describe our implementation of the hierarchical approach to establishing opponent models. The approach is implemented in the Spring game. We first describe (A) the opponent behaviour that is to be modelled. The opponent behaviour can be abstracted in a hierarchy of two levels, viz. (1) the top level, and (2) the bottom level. Subsequently, we describe (B) the top-level classification of opponent behaviour, and (C) the bottom-level classification of opponent behaviour.

A. The opponent behaviour

For modelling a strategy in the Spring game we decided to distinguish at the top level between an aggressive, and a defensive playing style.

For an aggressive playing style we distinguish at the bottom level between an opponent that is predominantly using one of the four predefined unit types: (1) k-bots, (2) tanks, (3) aircrafts, and (4) ships. Each unit type has specific strengths and weaknesses (see Appendix A.2), and as a result is used to execute a particular strategy. For instance, k-bots are relatively fragile but can cross mountains, and are therefore useful for a strategy against an opponent which attempts to block cliffs between mountains. Tanks can only manoeuvre on non-elevated terrain but are relatively sturdy, and are therefore useful for a strategy against an opponent who constructs strong defenses.

For a defensive playing style we distinguish at the bottom level between an opponent that pursues one of the following three building preferences: (1) nuke, (2) tech, and (3) bunker. These three building preferences are commonly observed in actual Spring games.

The hierarchy of the opponent models is displayed in Figure 7.1. The hierarchy consists of two levels (i.e., a top level, and a bottom level), and encompasses the following strategies.

- *Aggressive→K-bots.* The opponent will attack early, and will typically use k-bots.
- *Aggressive→Tanks.* The opponent will attack early, and will typically use tanks.
- *Aggressive→Aircrafts.* The opponent will attack early, and will typically use aircrafts.
- *Aggressive→Ships.* The opponent will attack early, and will typically use ships.
- *Defensive→Nuke.* The opponent will not attack early, and will attempt to construct a nuclear super weapon.
- *Defensive→Tech.* The opponent will not attack early, and will attempt to reach a high technology level in order to have rapid access to advanced units.
- *Defensive→Bunker.* The opponent will not attack early, and will construct a wall of static defenses around his base so that it has time to construct an army.

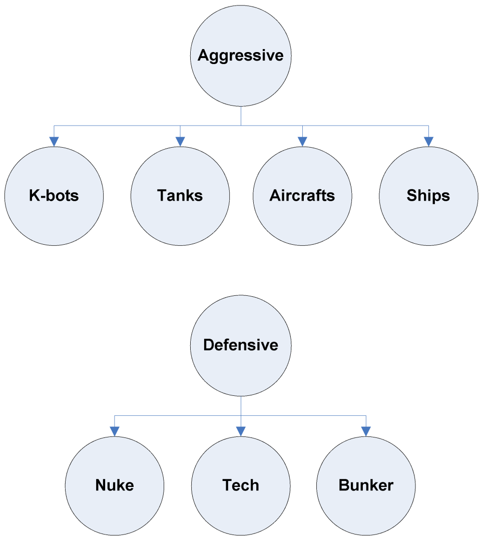


Figure 7.1: Hierarchy of the opponent models in SPRING.

B. The top-level classifier

We implement the top-level classifier by using fuzzy models (Zarozinski, 2002). Fuzzy models create models of several classes based on a single numerical feature. The choice of the numerical feature is crucial, as it should be able to distinguish between the defined classes. Fuzzy models may be considered a computationally-inexpensive form of modelling.

In our hierarchy of opponent models, the top-level classifier has to distinguish between the classes 'aggressive' and 'defensive'. An aggressive opponent, on the one hand, typically will employ the available game time for preparing many small attacks, and as a result will spend a large part of the game time on executing attacks. A defensive opponent, on the other hand, typically will employ the available game time for preparing one large attack, and as a result will spend only a small part of game time on executing attacks. Thus, an appropriate numerical feature to distinguish between the two classes would be the relative amount of game time that the opponent spends on executing attacks.

For the present experiments, we define an attack as the observed loss of a player's own units. When a loss of units is detected, the time span around this moment can be regarded as the time that an attack took place. Because information about a player's own units is always available, this definition is suitable for use in an imperfect-information environment. The time span is defined as a moving window of N seconds, and is described further in Subsection 7.2.3-A.

C. The bottom-level classifier

We implement the bottom-level classifier for distinguishing between the submodels that further determine behavioural characteristics of the hierarchy's top level. The dynamic nature of RTS games implies that a player's strategy may change during the game. Particularly for classifying the bottom-level strategy, the classifier needs to emphasise recent in-game observations more than earlier in-game observations.[3] To achieve the emphasis in classification, the principle of discounted rewards in repeated matrix-games is applied (cf., e.g., Gibbons, 1992).

We classify the bottom-level strategy on the basis of observations made during in-game events. In classical games, a game event would generally be one move of both players. Since RTS games do not operate in strictly defined moves, the term of an event must be adapted for our implementation. When playing against an aggressive opponent, we regard the occurrence of an attack as a suitable event, since the player can then observe the army of the opponent player. When playing against a defensive opponent, we regard the occurrence of scouting units that reach the opponent's base as a suitable event, since the player can then observe the base of the opponent player.

When a game event is detected, the confidence values of each possible strategy will be updated according to the observed game information. If δ is the discount factor, $\psi_{s,t}$ the belief that the opponent uses strategy s at event t, ranging between 0 and 1, π the total reward added at each event, and i the most recent event, then the confidence c_s that the opponent uses strategy s is computed as follows (adapted from Gibbons, 1992).

$$c_s = \sum_{t=i}^{0} \psi_{s,t} * \pi * \delta^{i-t} \tag{7.1}$$

The value of parameter $\psi_{s,t}$ is acquired by inspecting all visible units and buildings during event t. Each unit or building has a value representing a tendency to a certain strategy. These so-called unit-tendency values were determined by the experimenter, using his own knowledge of the game (Schadd, 2007). We give two examples of unit-tendency values: (1) a common defensive tower has a relatively small tendency towards being used in the defensive bunkering strategy, and (2) a nuclear super-weapon building has a relatively high tendency towards being used in the defensive nuke strategy.

7.2.3 Experiments with the implementation

This subsection reports on two experiments. In the first experiment we test the top-level classifier. In the second experiment we test the bottom-level classifier. In the remainder of the subsection, we first discuss (A) the experimental setup. Subsequently, we give (B) the results of the experiment that tests the top-level classifier. Next, we give (C) the results of the experiment that tests the bottom-level classifier. Finally, we provide (D) a summary of the experimental results.

[3] To some extent, emphasising recent in-game observations more than earlier in-game observations may also be the desirable for classifying the top-level strategy. Based on our experience with the SPRING game, however, we here assume that a player generally does not switch the top-level strategy during play of one game.

A. Experimental setup

The general experimental setup is as follows. We pit two game AIs against each other, and during play of the game we measure the confidence values obtained by the top-level and bottom-level classifiers. As friendly player we use the AAI game AI (Seizinger, 2006). As opponent player, that will follow the defined strategies (described in Subsection 7.2.2-A), we use the NTAI game AI (Nowell, 2007). Opponent behaviour is classified as resulting from the opponent strategy that models the behaviour with the highest confidence. The performance of a classifier is expressed by the correctness of the classification.

We test the top-level classifier's ability to distinguish between an aggressive and a defensive strategy. In addition, we test the bottom-level classifier's ability to distinguish between the aggressive submodels k-bots and tanks, and between the defensive submodels bunker, tech, and nuke. Tests with classifying the aggressive submodels aircrafts and ships were omitted by the experimenter. However, considering the relative similarity of the two submodels with the aggressive submodels k-bots and tanks, we surmise that a comparable performance may be expected. Detailed settings for each of the two experiments are described below.

Testing the top-level classifier: The top-level classifier requires the AI to detect if an attack took place (see Subsection 7.2.2). Detection of an attack is implemented as follows. The AI will register all visible units every N seconds, where N is the size of the time window. The detection algorithm will scan each game state for lost units. If the amount of lost units is above a certain threshold value, then each game state inside the analysed time window is labelled as a moment in which the opponent player was attacking the friendly player.

Two parameters determine the accuracy of the attack detection algorithm. The first parameter is the size of the time window. The second parameter is the unit-lost threshold, which is a percentage value. In comparative tests it was determined that effective parameter settings are a time window of 30 seconds, and a unit-lost threshold of 10% (Schadd *et al.*, 2007). We pitted the AAI player fifty times against an aggressive opponent, and fifty times against a defensive opponent.

Testing the bottom-level classifier: For testing the bottom-level classifier, the NTAI game AI has been configured such that the employed game strategies reflect each of the specified aggressive submodels. However, since NTAI is not able to play adequately the defensive strategies, a human opponent was chosen to play against the AAI game AI, by following the defensive strategies. For the discounted reward algorithm, the parameters were set by the experimenter to $\delta = 20\%$ and $\pi = 0.8$. For each submodel, we pitted the AAI player ten times against an opponent following the strategy that is determined by the particular submodels.

B. Performance of the top-level classifier

Below we give the measured confidence values of the top-level classifier. We note that game developers should interpret obtained confidence values in their own context. That is, a low

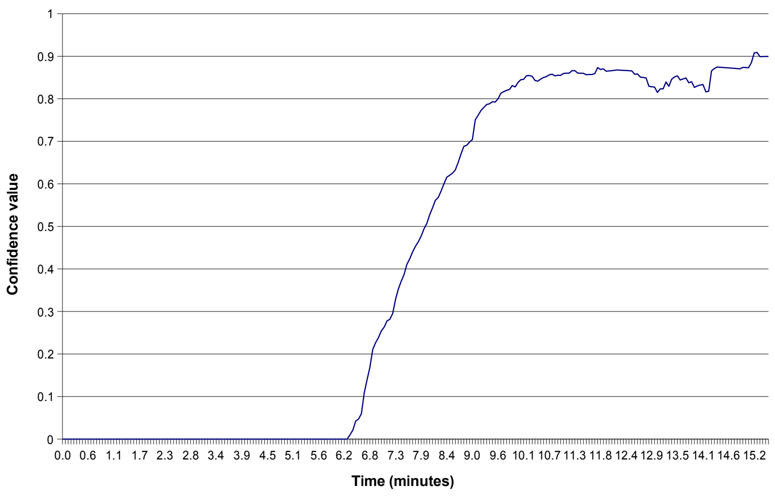


Figure 7.2: Average confidence value over time against an aggressive opponent.

confidence value early in the game, based on limited observations, does not exclude a high confidence value later in the game, when game observations may be more representative for the actual play of the opponent. The average confidence value over time against an aggressive opponent is displayed in Figure 7.2. The average confidence value over time against a defensive opponent is displayed in Figure 7.3. For ease of interpreting the two figures, we omitted listing the confidence value over time of the "incorrect" classification, as it amounts a value of one minus the confidence value of the correct classification.

In Figure 7.2 we observe that the average confidence value is low in the beginning of the game. This is due to the fact that the opponent is hardly able to attack early in the game, since it needs to construct a base first. Therefore, one can safely disregard the confidence values at the beginning of the game. After approximately seven minutes of game time, the average confidence value increases until it stabilizes at approximately 0.85.

In Figure 7.3 we observe that in the beginning of the game, the confidence values are nearly 1.0. The value is high because the enemy does not attack in the beginning of the game. The top-level classifier will therefore respond with the maximum defensive confidence value early in the game. One observes that after about ten minutes of game time, the average confidence value stabilises between 0.96 and 0.97.

C. Performance of the bottom-level classifier

Below we give the measured confidence values of the bottom-level classifier. We subsequently discuss the confidence values obtained when classifying the aggressive submodels k-bots and tanks, and when classifying the defensive submodels bunker, tech, and nuke.

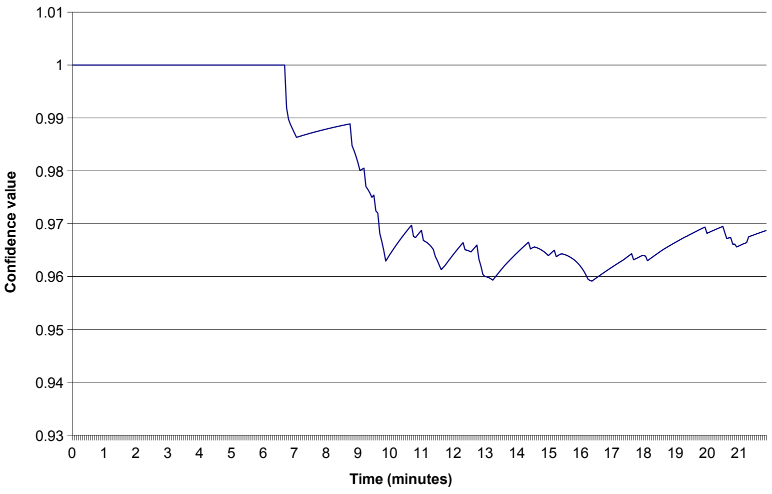


Figure 7.3: Average confidence value over time against a defensive opponent.

Aggressive Opponent - K-bots and Tanks

Figure 7.4 displays the average confidence over time of an opponent using the aggressive k-bot strategy. It also displays the average confidence over time of an opponent using the aggressive tank strategy. We observe that both confidence values eventually approximate a value over 0.90. We note that the average confidence of the aggressive tank strategy increases more slowly, and more late in the games than the average confidence of the aggressive k-bot strategy. This can be explained by the fact that constructing tanks requires more resources than constructing k-bots. Therefore, more game time will be needed for the opponent player to attack with tank units.

Defensive Opponent - Bunker

Figure 7.5 displays the average confidence values obtained against an opponent using the defensive bunker strategy. It also displays the confidence values of the defensive tech and the defensive nuke strategy. In the discussion of the performance, we focus on the confidence values of the defensive bunker strategy. We observe that the bunker confidence values increases rapidly after approximately five minutes of game time. Over time the obtained average confidence value is 0.83. The instabilities that occur after 35 minutes of game time can be explained by the fact that at this moment the game AI has discovered buildings that may also be used by an opponent player that is following the tech strategy.

Defensive Opponent - Tech

Figure 7.6 displays the average confidence values obtained against an opponent using the defensive tech strategy. We observe that for the largest part of the game, the confidence values of the bunker strategy are higher than the confidence values of the tech strategy. This

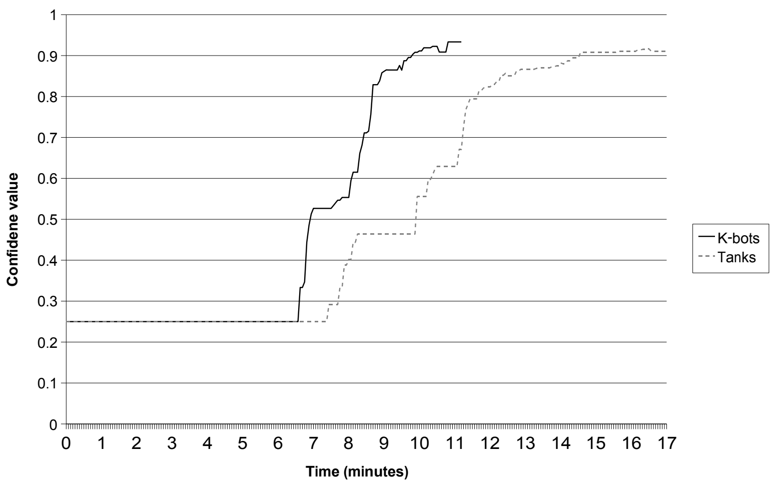


Figure 7.4: Average confidence value over time for two aggressive strategies.

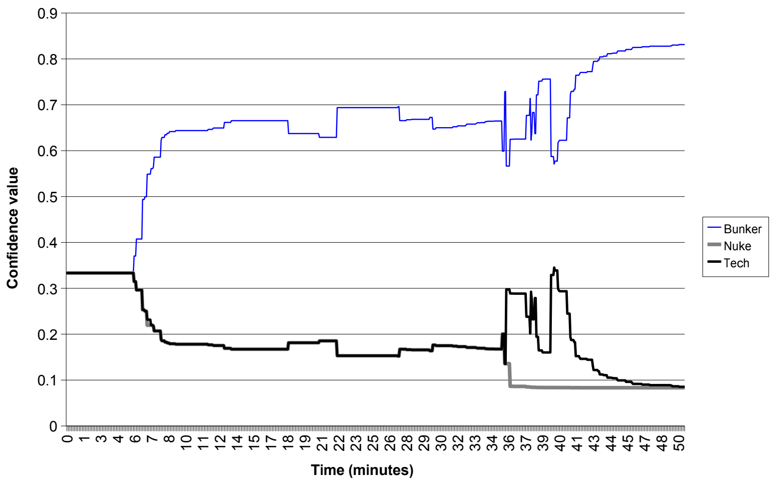


Figure 7.5: Average confidence value over time for an opponent using the defensive bunker strategy.

can be explained by the fact that the units and buildings that define the tech strategy can only be constructed relatively late in the game. This implies that early in the game only units and buildings that belong to a different strategy can be observed. When later in the game the

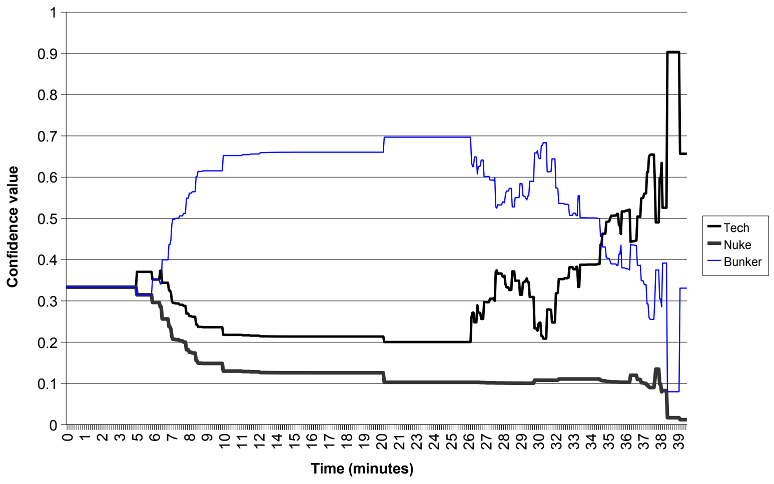


Figure 7.6: Average confidence value over time for an opponent using the defensive tech strategy.

AI is able to observe buildings that are characteristic for the tech strategy, the confidence value of the tech strategy increases.

Defensive Opponent - Nuke

Figure 7.7 displays the average confidence values obtained against an opponent using the defensive nuke strategy. Analogously to the results obtained against an opponent using the defensive tech strategy, we observe that the confidence values of the bunker strategy are higher than the confidence values of the nuke strategy. After approximately 25 minutes the confidence value of nuke strategy steadily increases. After approximately 41 minutes, the discounted-reward algorithm temporarily decreased the confidence value because the AI did not observe any more buildings associated with the nuke strategy. Eventually the bottom-level classifier obtains a confidence value of 0.75.

D. Summary of the experimental results

Experimental results obtained with the top-level classifier show that the top-level classifier can accurately distinguish between an aggressive and a defensive opponent. Experimental results obtained with the bottom-level classifier show that the bottom-level classifier can accurately distinguish between the established submodels relatively late in the game. Relatively early in the game, the bottom-level classifier was not always able to distinguish accurately between the established submodels. This is discussed next.

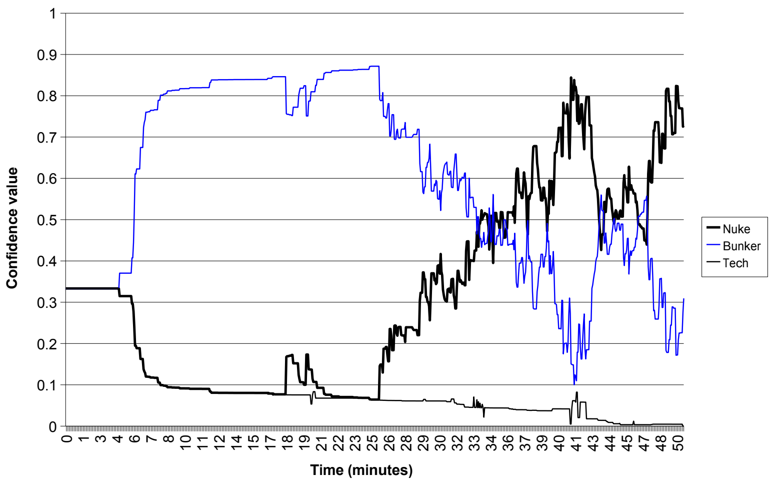


Figure 7.7: Average confidence value over time for an opponent using the defensive nuke strategy.

7.2.4 Discussion of the results

This subsection provides a discussion of the experimental results. We first discuss (A) the obtained experimental results in general terms. Subsequently, we draw (B) experimental conclusions of the present research.

A. General discussion

Ideally, an accurate classification of the opponent's strategy is available relatively early in the game, at a time when the player is still able to counter the opponent's strategy. In the experiments that tested our implementation of opponent modelling, we observed that the bottom-level classifier was not always able to distinguish accurately between the established submodels early in the game. This phenomenon can be explained by the fact that, typically, the AI cannot directly observe units and buildings that are characteristic for a particular bottom-level strategy. To observe the units and buildings associated with the particular bottom-level strategies, the AI relies on scouting.

A straightforward approach to achieve improved results, therefore, is to adapt the AI's scouting behaviour dependent on the need for information on the opponent's activities. For instance, in competition against an aggressive opponent, scouting is relatively unimportant. In competition against a defensive opponent, however, scouting is vital. Analogously, one may choose to adapt the parameters of the delayed reward algorithm, to emphasise the information that is obtained during a scout event.

B. Experimental conclusions

Our experimental results show that an opponent's general playing style can accurately be classified by the top level of the hierarchy. In addition, experimental results obtained with the bottom level of the hierarchy show that early in the game it is difficult to obtain accurate opponent classifications. Later in the game, however, the bottom level of the hierarchy will accurately classify between specific strategies of the opponent. From these results, we may conclude that the approach to establishing opponent models in RTS games can be used successfully to classify the strategy of the opponent player online, while the game is still in progress.

7.3 Exploiting opponent models in SPRING

In the previous section we discussed an hierarchical approach to establishing opponent models in the Spring game. It was shown that in this complex game relatively accurate models of opponent behaviour can be established. In this section, we will focus on exploiting the established models in online play. Our assumption is that the established models may be exploited successfully to increase the effectiveness of game AI. In the remainder of the section, we investigate the assumption.

The outline of this section is as follows. First, we describe an approach to exploiting the established opponent models (7.3.1). Subsequently, we report on an experiment that tests the approach in the Spring game (7.3.2). Finally, we provide a discussion of the results (7.3.3).[4]

7.3.1 Approach to exploiting opponent models

In this subsection, we discuss the approach adopted for exploiting opponent models in the Spring game (cf. Mehlkop, 2009). We first describe (A) the game strategies that are considered by the approach. Subsequently, we discuss (B) exploiting opponent models in play of the Spring game.

A. Game strategies

For the present experiment with exploiting opponent models, we employ a set of game strategies for the approach to respond to effectively in online play. For the present experiment, we handcraft three game strategies; two aggressive strategies (aircraft and k-bot), and one defensive strategy (nuke). To ensure that opponent classification is uncomplicated, the three game strategies are simplified versions of actual strategies. Hence, the focus lies on the effect of using the opponent classification. The strategies are described below.

Aircraft: The aircraft game strategy is an aggressive strategy. The strategy constructs exclusively aircraft units to attack the opponent player. To ensure a complete focus on attacking the opponent player, it does not construct defence buildings. The game AI rapidly sends constructed aircraft units across the map, with the aim to discover the

[4] The work reported on in this section is performed by Mehlkop (2009), under supervision of the author.

location of the Commander unit. Once the Commander unit is located, the game AI will order increasingly large groups of aircraft units to attack the opponent.

K-Bot: The k-bot game strategy is a moderately aggressive strategy. The strategy constructs exclusively k-bot units to attack the opponent player. In addition, around its own base, the game AI constructs relatively light defence buildings, such as light laser turrets. The game AI will order small groups of units to attack the opponent. Gradually, the game AI will send increasingly large groups of k-bot units to attack the opponent.

Nuke: The nuke game strategy is a defensive strategy. The strategy does not construct any units that directly attack the opponent player. Instead, the strategy is aimed at constructing multiple nuclear missile silos. Once the silos launch their containing missiles at the opponent's base, the effect on the base will be so devastating that the opponent will typically lose the game instantly. The game AI constructs relatively strong defence buildings, such as heavy laser turrets, to ensure that the missile silos are not destroyed during their construction.

B. Exploiting opponent models

We incorporate means for exploiting opponent models in the game AI of the Spring game. Our implementation of exploiting opponent models consists of two parts. First, we classify the opponent player, based on actual observations in online play. Second, we use the classification of the opponent player, for the purpose of improving the effectiveness of the game AI. The two parts of our implementation are discussed below.

Opponent classification: The procedure to classify the strategy of the opponent player is as follows. During online play, the game state is polled once every two seconds. Subsequently, observations of each game state are used to classify the strategy of the opponent player. To this end, we incorporate the opponent classifiers that were established previously by Schadd *et al.* (2007) (see Section 7.2). Adopted for the present experiment, a separate classifier for each game strategy tracks the probability of the game observations resulting from the particular strategy. If the classification probability of one of the classifiers reaches a predefined threshold value, the opponent is classified as following the strategy that fits the game observations with the highest probability. If the classification probability of none of the classifiers reaches the predefined threshold value, no opponent classification is performed, as we do not consider the classification as potentially reliable.

Using the classification: We use the opponent classification for adapting the game strategy in an informed manner. Naturally, knowledge on which strategy the opponent player employs, is only of use if there are means to exploit this knowledge. In our experiment, we assume that for each opponent classification, a counter strategy is available that can be employed directly. We surmise that (1) the aircraft strategy will win the game when pitted against the nuke strategy, (2) the nuke strategy will win the game when pitted against the k-bot strategy, and (3) the k-bot strategy will win the game when

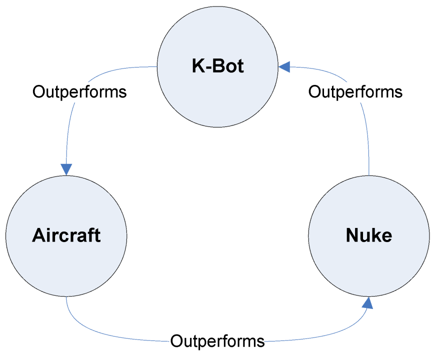


Figure 7.8: Relationship between the predefined game strategies.

pitted against the aircraft strategy. The surmised relationship between the three game strategies is illustrated in Figure 7.8.

When the domain knowledge on suitable counter strategies is confirmed in our experiment, it can be used together with the opponent classification for intelligently adapting the game strategy in online play. Specifically, upon opponent classification the adaptation mechanism intelligently switches the game strategy, on the basis of the available domain knowledge.

7.3.2 An experiment with exploiting opponent models

In this subsection, we report on an experiment with exploiting opponent models in the Spring game. We first describe (A) the experimental setup. Subsequently, we discuss (B) the obtained experimental results.

A. Experimental setup

In our experiment, two identical game AIs are pitted against each other on the map 'Small-Divide', the default map of the Spring game. One game AI exploits the available opponent models, the other game AI does not exploit the available opponent models. The goal of the experiment is to determine to what extent exploiting opponent models allows a game AI to increase its performance. The performance is expressed by the number of games that are won by the AI that exploits the models. All three game strategies were implemented in the game AI. Subsequently, we implemented (1) the previously described technique for opponent classification (see Section 7.2), and (2) the previously described mechanism for using the opponent classification (see Subsection 7.3.1). Upon classification of the opponent player, the adaptation mechanism exploits knowledge on which counter strategy is effective against which opponent. As expected, it thereby is able to switch effectively the game strategy.

Player 1 / Player 2	Aircraft	K-Bot	Nuke	Informed AI
Aircraft	-	0 / 10	10 / 0	0 / 10
K-Bot	10 / 0	-	0 / 10	0 / 10
Nuke	0 / 10	10 / 0	-	0 / 10
Informed AI	10 / 0	10 / 0	10 / 0	-

Table 7.1: Results of exploiting opponent models in SPRING.

The general experimental procedure is as follows. First, we determine the effectiveness of each of the three game strategies being pitted against another strategy. Second, each of the three strategies is being pitted against an adaptive 'informed AI', which exploits domain knowledge on the effectiveness of the game strategies. Each experimental trial is repeated ten times. The informed AI follows strictly one of the three predefined game strategies. The only difference in behaviour is that in order to provide the opponent classification technique with information, the game AI constructs several scout units that attempt to observe the base of the opponent. We ascertained that the construction of these scouting units has no effect on the playing strength of AI, as scout units are highly vulnerable, and can be produced with negligible investment in time and resources.

To focus solely on the effect of adapting the game strategy, in our experiment each player is provided with unlimited resources for constructing units and buildings. The informed AI is initialised with the nuke strategy, for this strategy may be regarded as exhibiting a moderate playing strength, and can be adapted straightforwardly in online play. Classification of the opponent player is attempted once every two seconds. The threshold value for classification of the opponent is 0.7.

B. Results

Table 7.1 gives an overview of the obtained experimental results. Each cell of the table lists the number of games that are won / lost when pitting player 1 (row) against player 2 (column), respectively.

The results confirm our assumption that, when the opponent models are not exploited, the aircraft strategy outperforms the nuke strategy. In addition, we observe that the nuke strategy outperforms the k-bot strategy, and that the k-bot strategy outperforms the aircraft strategy. This knowledge is exploited by the informed AI.

In addition, the results reveal that when the adaptive 'informed AI' intelligently exploits the opponent classification, the effectiveness of switching strategy is one hundred per cent. That is, by performing an informed switch in strategy, the informed AI is able to win all games against all three opponents.

7.3.3 Discussion of the results

This subsection provides a discussion of the experimental results. We discuss (A) the obtained experimental results in general terms. Subsequently, we draw (B) experimental conclusions of the present research.

A. General discussion

In our experiment, we observed that in the complex Spring game, established models of the opponent player can be exploited successfully, to adapt the employed game strategy in such a way that it outperforms the opponent player. To focus on the effect of adapting the game strategy on the basis of the opponent classification, we simplified the conditions in which the Spring game is being played. Most importantly, we did not require the game AI to perform complicated resource management tasks. Moreover, the game strategies that were under investigation in the present research were simplified versions of actual strategies. For instance, it is unlikely that a human player (or a competitive game AI) would follow the nuke strategy without constructing any units for occasionally attacking the opponent player, since without occasionally attacking the opponent a player would render his base relatively vulnerable. So, we note that our approach has not been validated under strict game-playing conditions. Still, we posit that the obtained (save perfect) results may apply generically to games where the intelligent adaptation of game strategy plays an important role in the outcome of the game. This covers the majority of strategic games.

We note that the opponent models that were exploited in the present research were handcrafted. This may limit the effectiveness when exploiting the opponent models in actual play, as the models were not updated periodically to include recent game observations. In Chapter 8 we will experiment with opponent models that are generated and updated automatically, based on observations of a multitude of games.

B. Experimental conclusions

Our experimental results with exploiting opponent models in the Spring game, show that intelligently exploiting the established opponent models leads to more effective behaviour in online play. The obtained results reveal the importance of gathering domain knowledge on exactly which strategy is effective in play against a particular opponent. In our experiment, the informed game AI exploits successfully this knowledge once behaviour of the opponent is classified. Afterwards, it is able to outperform consistently the opponent player. From these results, we may conclude that the described approach to exploiting opponent models may be applied successfully in this RTS game to adapt online the strategy that is followed by the game AI.

7.4 Chapter conclusion

In this chapter we discussed the technique that enables game AI to establish and exploit models of the opponent player, viz. opponent modelling. Experiments with establishing opponent models in the complex Spring game revealed that for the game relatively accurate models of the opponent player can be established. Furthermore, an experiment with exploiting opponent models showed that in the Spring game, exploiting the established opponent models in an informed manner leads to more effective behaviour in online play. From these results, we may conclude that opponent modelling may successfully be incorporated in game AI that operates in a complex video game, such as Spring.

Now that we have investigated each of the three main components of case-based adaptive game AI, viz. (1) an evaluation function (see Chapter 5), (2) an adaptation mechanism (see Chapter 6), and (3) opponent modelling (the present chapter), in the next chapter we will investigate how the three main components can be integrated into a complete implementation of case-based adaptive game AI.

8 Integrating the three components

Chapter 8 reports on the experiments that integrate the three main components of case-based adaptive game AI: (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. We first outline our proposal for integrating the three components into a complete implementation of case-based adaptive game AI (Section 8.1). As a result of the outlined proposal, we discuss how to incorporate opponent modelling in the implementation of case-based adaptive game AI (Section 8.2). Next, we report on two experiments that test case-based adaptive game AI in an actual video game (Section 8.3). Then, we give an analysis of the practical applicability of case-based adaptive game AI (Section 8.4). Finally, we present the chapter conclusions (Section 8.5).

8.1 How to integrate the three components

In this section, we first reflect concisely on the three main components that were investigated in the previous chapters, viz.(1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. We focus specifically on which steps need to be taken to integrate each components into case-based adaptive game AI. The reflection leads to our proposal for a complete implementation of case-based adaptive game AI.

Evaluation function: In Chapter 5 we investigated an evaluation function for rating the game state in the complex SPRING game. We showed that the evaluation function is able to accurately predict the outcome of a SPRING game. As such, the evaluation function may be incorporated directly into case-based adaptive game AI.

Adaptation mechanism: In Chapter 6 we investigated an adaptation mechanism that allows video game AI to adapt to game circumstances. The mechanism performs adaptations on the basis of game observations that are gathered in a case base, and

This chapter is based on the following publication.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009a). Opponent modelling for case-based adaptive game AI. *Entertainment Computing*, 1(1):27–37.

in addition receives input of an evaluation function. Thereby, we essentially established a basic form of case-based adaptive game AI; one that incorporated two main components (only the component 'opponent modelling' was still missing, see below). An experiment that tested the basic form (with two components only) of case-based adaptive game AI in the SPRING game, revealed that the basic case-based adaptive game AI could play a strong game. That is, from game observations that were gathered in the case base, the basic case-based adaptive game AI was able to deduce strategies that could effectively defeat its opponent AI.

Then we noted that the final outcome of a SPRING game is largely determined by the strategy that is adopted in the beginning of the game. This exemplifies the importance of initialising the game AI with effective behaviour. In order to do so, the basic case-based adaptive game AI needs to be extended with the ability to establish and exploit models of the opponent player; this may be done systematically and in the best way by means of incorporating opponent modelling.

Opponent modelling: In Chapter 7 we investigated opponent modelling in the SPRING game. We showed that in the complex SPRING game, relatively accurate models of the opponent player can be established. In addition, we showed that the models can be exploited to evoke effective behaviour in online play. The results revealed that opponent modelling may successfully be incorporated in the AI of complex video games, such as SPRING. By extending the basic case-based adaptive game AI with the ability of opponent modelling, a complete implementation of case-based adaptive game AI is established. Naturally, the question remains how exactly opponent modelling should be incorporated into the case-based adaptive game AI. Our general proposal to this end is discussed below.

Our general proposal for incorporating opponent modelling into the case-based adaptive game AI is to make the component an integral part of *case-based* adaptation. That is, let the component exploit the case base that is built from a multitude of observed games, for the purpose of automatically establishing models of the opponent player. Subsequently, exploit the established models for the purpose of allowing the adaptation mechanism to adapt more intelligently the game AI to game circumstances. The adaptation process, we propose, is performed in two steps. First, classify in online play the opponent player. Second, exploit the classification together with previous observations that are gathered in the case base, to reason on the preferred game strategy. This latter proposal is similar to work by Van der Heijden *et al.* (2008), who successfully implemented opponent modelling in a relatively simple mode of the ORTS game. Discussed next is how we implement the general proposal for incorporating opponent modelling into the case-based adaptive game AI.

8.2 Incorporating opponent modelling

In this section we discuss how we incorporate opponent modelling into the case-based adaptive game AI. In our approach to case-based adaptation of game AI, opponent models are established automatically, on the basis of a case base of game observations. Our goal of exploiting the opponent models is to improve the effectiveness of the adaptive game AI. We first describe how we establish models of the opponent player (8.2.1). Subsequently, we discuss how we exploit models of the opponent player (8.2.2).

8.2.1 Establishing opponent models

In our approach to case-based adaptation of game AI, opponent models are established automatically. It happens on the basis of game observations gathered in the case base. The models of the opponent players are established as follows. We start defining and selecting features of an opponent's high-level strategic behaviour. As a first step to establishing opponent models, the features are defined and selected by the researchers, to reflect their expertise with the game environment. We admit that by manually defining and selecting the game features we may restrict the accuracy of the models. The investigation of further improvements with respect to the definition and selection of features is considered a topic for future research.

At the start of a Spring game, the strategy that an opponent will adopt is unknown, as (1) players of the game remain anonymous, and (2) starting conditions are identical for each player of the game (i.e., each player is initially provided with only one unit, the Commander unit). As a result, opponent models need to be established solely on the basis of later, in-game observations on the strategic behaviour of an opponent player.

Strategic behaviour, e.g., the opponent's preference of unit type, the focus of an opponent's technological development, the strength of his economy, and the aggressiveness of the opponent, can generally be inferred from observing the values of selected features during actual play. The features that are selected for the present research are given below. We note that the features are different from those used in earlier chapters, as the task of modelling an opponent's strategy is distinct from, e.g., comparing the similarity of game observations. Also, we note that no features that concern ship units are incorporated, as our experiments are performed on a map without water areas. For convenience of the reader, in Appendix B we will provide an overview of all features that are used in the three main components of case-based adaptive game AI.

1. Number of observed k-bot units.
2. Number of observed tank units.
3. Number of observed aircraft units.
4. Number of technologically advanced buildings (i.e., level 2 or higher).
5. Number of metal extractors.
6. Number of solar panels.
7. Number of wind turbines.
8. Time of first attack on one of the metal extractors.
9. Time of first attack on one of the solar panels.
10. Time of first attack on one of the wind turbines.

The first three features express the global strategic preference of an opponent, which is important in determining placement of units and buildings. For instance, in the map SmallDivide, the mountain ridges can be crossed by k-bot units, but not by tanks. If the game AI can deduce that the opponent is not constructing k-bot units, it can safely distribute resources for a purpose other than defending the mountain passes.

The fourth feature expresses the technological development of a player. If the game AI can deduce that the opponent is constructing advanced units, it can respond by constructing such units too.

The fifth, sixth, and seventh feature express the strength of an opponent's economy, and by implication, the strength of the opponent's army. If the game AI can deduce that the opponent's economy is relatively weak, it can safely shift the balance from constructing defensive units to offensive units.

The eighth, ninth, and tenth feature express the aggressiveness of the opponent player. If the game AI can deduce that the opponent follows an aggressive playing style, which implies that it launches many relatively small attacks, it can attempt to build defences to withstand the small attacks, and simultaneously construct a relatively strong army. The time of the first attack is expressed in terms of time steps that have passed since the game started. A time step is defined as 4.233 seconds (cf. Chapter 5 and 6).

In establishing the opponent models, we prefer to exploit the models in a relatively early state of playing the game. The reason for this preference is straightforward. In SPRING, game actions that are performed early in the game will have a relatively strong impact on the final outcome of the game. As a result, when knowledge on the opponent player is available in a relatively early state of the playing the game, then the game AI can already steer its own behaviour towards arriving at the preferred game state. To this end, we gather feature data of observed opponent behaviour in a specific time step that is relatively early in the game, but that is not too early for observing strategic choices of the opponent player. In the later described experiments, opponent models will be established after 150 time steps of play, which amounts to approximately ten minutes of real-time play. Based on the feature data gathered when observing numerous games, opponent models are generated by clustering the feature data via the standard k-means clustering algorithm (Hartigan and Wong, 1979). To determine automatically the difference in opponent behaviour expressed by the feature data, the Euclidean distance measure is incorporated in the applied clustering algorithm.

8.2.2 Exploiting opponent models

We exploit opponent models as follows. We extend the offline processing phase of the basic case-based adaptive game AI (see Section 6.2), by labelling each game in the case base with the classification of the opponent player against whom the game AI was pitted. This process is performed by a classification algorithm, which classifies the game AI's opponent on the basis of observed feature data of the particular game. As we establish opponent models by a clustering of feature data (see Subsection 8.2.1), the classification of an opponent is straightforward; namely by calculating the nearest neighbouring cluster of opponents for a given game observation. The number of available classes is determined automatically by the clustering algorithm.

The classification of the opponent player is exploited for (A) initialisation of game AI, and (B) online strategy selection in actual play. This is discussed below.

A. Initialisation of game AI (with OM)

In our experiments, an adaptive game AI will be pitted against various opponent players (cf. Section 6.3). The procedure to intelligently select the strategy initially followed by the adaptive game AI consists of two steps. The steps are as follows.

First, based on previous observations, we determine which opponent the game AI is likely to be pitted against in a new game. As an initial step to determining the opponent player, in our experiments we surmise that the game AI will be pitted against the opponent which over the course of numerous games has been observed the most. We note that this is a rough assumption which may be enhanced by incorporating additional domain knowledge, such as the likelihood of opponents adapting their own behaviour after play of a successful (or an unsuccessful) game.

Second, we initialise the game AI with the strategy that in previous games has proven most effective against this particular opponent. We consider a game strategy effective when in previous play it achieved a set goal criterion (thus, the game AI will never be initialised with a predictably ineffective strategy). The goal criterion can be any metric to represent preferred behaviour (cf. Subsection 6.2.6). In our experiments, the goal criterion is a desired fitness value. For instance, a desired fitness value of one hundred represents a significant victory.

B. Online strategy selection (with OM)

Here we select online which strategy to employ when a transition in the phase of the game takes place (cf. Subsection 6.2.6). In the case that no opponent models are available, the procedure is as described in Subsection 6.2.6.

In the case that opponent models are available, there is an additional decision point for game adaptation besides at the occurrence of phase transitions, namely at the moment at which the opponent player can be classified accurately. This moment is the exact time step at which the opponent models were established previously (see Subsection 8.2.1). We recall that the game AI is initialised with a game strategy on the basis of a prediction on who the opponent player will be. The game strategy is adapted if the initial prediction of the opponent player is different from the actually observed opponent.

Furthermore, to compare with an increased reliability the similarities between the current game, and games that are gathered in the case base, the classification of the opponent player is incorporated in the process of selection of the game strategy (see Subsection 6.2.4). The classification of the opponent player is incorporated as a means to narrow down the preselection process. This is performed by preselecting the N games with the smallest accumulated fitness difference with the current game, that as an additional requirement have also been played against the actually observed opponent.

We assume that the proposed application of opponent modelling allows for exploiting game strategies more effectively. The experiments that are described next investigate our assumption.

8.3 Experiments with case-based adaptive game AI

This section reports on two experiments that test our complete implementation of case-based adaptive game AI. We first describe the experimental setup (8.3.1). Second, we discuss how the performance of the case-based adaptive game AI is assessed (8.3.2). Third, we report on the generated opponent models (8.3.3). Fourth, we give the experimental results from the two experiments (8.3.4). Finally, the section is concluded by a discussion of the results (8.3.5).

8.3.1 Experimental setup

The experimental setup is largely identical to the setup of the first experiment that was reported on in Chapter 6. For brevity, we here describe only the global procedure and discuss the differences between the current and the previous setup. For additional details we refer the reader to Subsection 6.3.1.

For readability of the present chapter, we split the first experiment that was reported on in Chapter 6 into two experiments. In the first experiment, the case-based adaptive game AI is pitted against the original AAI opponent. In the second experiment, the case-based adaptive game AI is pitted against (sets of) randomly generated opponents. Before the game starts, offline processing of the case-base, as well as selecting the initial strategy, is performed according to the procedure described in Section 6.2 (when no opponent models are available), or as described in Section 8.2 (when opponent models are available). Online (i.e., while the game is in progress), strategy selection is performed at every phase transition.

Opponent models are established after 150 time steps of play (see Subsection 8.2.1), which amounts to approximately ten minutes of real-time play. The parameter k for the k-means clustering of opponents is set to ten per cent of the total number of games. Empty clusters are removed automatically, in case the particular value of k was set too large.

8.3.2 Performance assessment

As the experimental setup is largely identical to the setup of the first experiment that was reported on in Chapter 6, we decided to do the performance assessment largely identical too. We recall that in the first experiment of Chapter 6, we set the case-based adaptation mechanism to win the game (i.e., obtain a positive fitness value). In the experiment, the effectiveness of the mechanism was expressed by the number of games that were won by the friendly player when it used the case-based adaptation mechanism. This measure for performance assessment is also used in the present two experiments. For clarity, we note that the present experiments have no relation to the second experiment that was reported on in Chapter 6, as that experiment concerned difficulty scaling.

In Chapter 6, we performed the first experiment in what, for readability, we here distinguish as two modes. For the experiments reported on in the present chapter, we add a third mode of adaptation. In the first mode, the case-based adaptation mechanism is disabled. We refer to the first mode as 'disabled' adaptation. In the second mode, the case-based adaptation mechanism is enabled. We refer to the second mode as 'basic' adaptation, as the mode

	SmallDivide	TheRing	MetalHeckv2
# Opponent models	9	8	9
Typical playing style	Defensive	Defensive	Aggressive
Building preference	Advanced buildings	Advanced buildings	No advanced buildings
Unit preference	Tank units	K-bot units	Tank units

Table 8.1: Overview of the characteristics of the generated opponent models.

essentially implements a basic form of case-based adaptive game AI (i.e., without opponent modelling) (see Section 8.1).

The third adaptation mode implements a complete form of case-based adaptive game AI, by incorporating opponent modelling as mentioned in Section 8.2. We refer to the third mode as 'OM' adaptation. In the present experiments, we compare the results obtained with the 'disabled' and 'basic' adaptation mode, with the results obtained with the 'OM' adaptation mode.

8.3.3 Generated opponent models

Opponent models were generated based on observations gathered from play on three distinct maps. Each map was observed over 325 games (cf. Chapter 5 and 6). On the map SmallDivide, nine opponent models were generated automatically. On the map TheRing, eight opponent models were generated automatically. On the map MetalHeckv2, nine opponent models were generated automatically. We list them in Table 8.1 together with their characteristics.

The generated models reveal that opponents observed on the map SmallDivide typically employ a defensive playing style, have a preference for constructing advanced buildings, and have a preference for constructing tank units. Opponents observed on the map TheRing are typically similar to those observed on the map SmallDivide, with the difference that they have a preference for constructing k-bot units, instead of tank units. Opponents observed on the map MetalHeckv2 typically employ an aggressive playing style, do not have a preference for constructing advanced buildings, and have a preference for constructing tank units.

8.3.4 Results of game adaptation

In this subsection, we give the results of the two experiments. A general discussion of the obtained results is provided in Subsection 8.3.5.

Table 8.2 gives an overview of the results of the first experiment performed in the Spring game. In the experiment, the case-based adaptive game AI was pitted against the original AAI game AI on the three different maps. The first column of the table lists the adaptation mode of the case-based adaptive game AI. The second column lists how often the trial was repeated. The third and fourth column list how often the goal was achieved in absolute terms, and in terms of percentage, respectively.

SmallDivide

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	59	39%
Basic	150	115	77%
OM	150	135	90%

TheRing

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	90	60%
Basic	150	122	81%
OM	150	127	85%

MetalHeckv2

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	70	47%
Basic	150	124	83%
OM	150	130	87%

Table 8.2: Effectiveness of case-based adaptive game AI against the original opponent.

The results reveal that when pitted against the original AAI game AI on the map SmallDivide, the effectiveness of case-based adaptive game AI increases substantially when opponent modelling techniques are incorporated (90%, compared to 77% without opponent modelling). Also on the maps TheRing and MetalHeckv2 the effectiveness increases when opponent modelling techniques are incorporated (85%, compared to 81%, and 87% compared to 83%, respectively). These results indicate that incorporating opponent modelling techniques indeed increases the effectiveness of case-based adaptive game AI. Figure 8.1 displays the obtained median fitness value over all game trials against the original AAI opponent on the map SmallDivide, as a function over the relative game time.

Table 8.3 and Table 8.4 give an overview of the results of the second experiment performed in the Spring game. In the experiment, the case-based adaptive game AI was pitted against random opponents, i.e., against the original AAI initialised with randomly generated strategies. The legend of Table 8.3 is equal to that of the first experiment. The legend of Table 8.4 is as follows. The first column of the table lists the label of the randomly generated opponent. The second column lists how often the trial was repeated. The third, fourth and fifth column list how often the goal was achieved in absolute terms in the mode where case-based adaptive game AI was disabled, where it operated in basic mode, and where it incorporated opponent modelling techniques, respectively. In the two bottom rows of the table, the average effectiveness over all trials is listed.

The results given in Table 8.3 reveal that when pitted against the original AAI game AI on the map SmallDivide, initialised with randomly generated strategies, the effectiveness of case-based adaptive game AI increases significantly when opponent modelling techniques

SmallDivide

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	71	47%
Basic	150	96	64%
OM	150	136	91%

TheRing

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	76	51%
Basic	150	93	62%
OM	150	93	62%

MetalHeckv2

Adaptation mode	Trials	Goal achv.	Goal achv. (%)
Disabled	150	54	36%
Basic	150	60	40%
OM	150	79	53%

Table 8.3: Effectiveness of case-based adaptive game AI against random opponents.

SmallDivide

Opponent	Trials	Adaptation mode		
		Disabled	Basic	OM
1	10	4	9	9
2	10	6	8	9
3	10	5	5	9
4	10	4	3	5
5	10	7	9	9
6	10	7	6	9
7	10	6	7	9
8	10	7	7	9
9	10	3	6	10
10	10	5	7	9
11	10	6	8	5
12	10	6	8	7
13	10	5	7	9
14	10	5	8	7
15	10	6	6	6
Goal achv. avg.		81	104	121
Goal achv. avg. (%)		55%	69%	81%

Table 8.4: Effectiveness of case-based adaptive game AI against *sets of* random opponents.

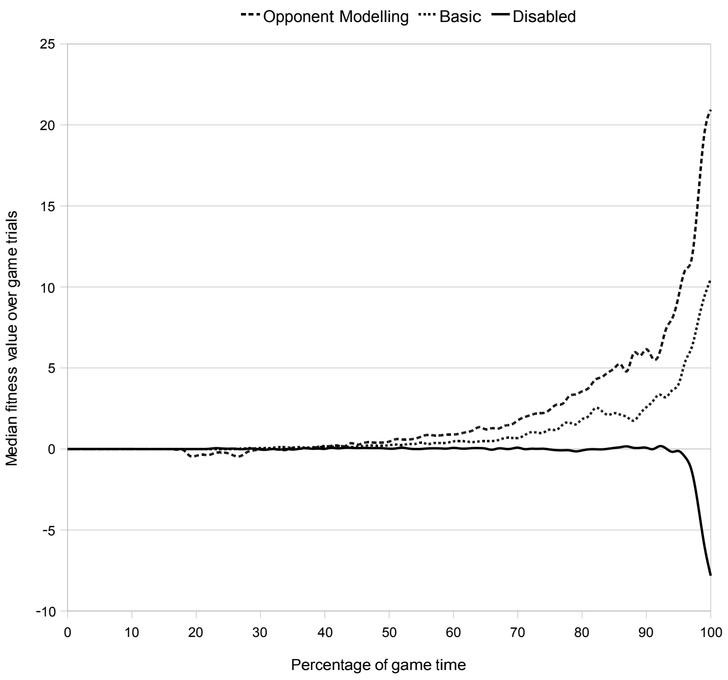


Figure 8.1: Median fitness value over all game trials against the original AAI opponent on the map SmallDivide, as a function over the relative game time.

are incorporated (91%, compared to 64% without opponent modelling) (cf. chi-square test, Cohen, 1995). On the maps TheRing and MetalHeckv2, the effectiveness of case-based adaptive game AI remains stable when opponent modelling techniques are incorporated (62%, compared to 62%), or increases (53%, compared to 40%), respectively. These results confirm our previous indication that applying opponent modelling techniques generally increases the effectiveness of case-based adaptive game AI.

The results given in Table 8.4 reveal an analogous, positive trend. That is, when case-based adaptive game AI is pitted against *sets of* randomly generated opponents on the map SmallDivide, the effectiveness of case-based adaptive game AI increases when opponent modelling techniques are incorporated (81%, compared to 69% without opponent modelling).

8.3.5 Discussion of the results

This subsection provides a discussion of the experimental results. We observed that by incorporating opponent modelling techniques, the case-based adaptive game AI was generally able to increase its effectiveness. However, in some circumstances, the increase in effectiveness was relatively modest, and in one situation the effectiveness remained stable. An analysis of this phenomenon shows that our proposed use of opponent modelling works best in circumstances where gameplay is highly strategic (e.g., the map SmallDivide), compared to circumstances where strategic gameplay is a matter of less importance (e.g., the map TheRing). We therefore conjecture that to increase the effectiveness in these circumstances, (1) the opponent models should incorporate additional features that model in more detail facets of the opponent behaviour. In addition, improved results may be established (2) by incorporating knowledge on how heavily the features of the models should be weighted, and (3) by investigating the principal features for certain tasks, for instance by applying principal component analysis (Pearson, 1901).

Naturally, the case-based adaptive game AI may still be confronted with an opponent that it has not observed previously. In this situation, the inherent generalisation that is provided by the clustering of opponent models may already have led to the game AI being initialised with a strategy that is also effective against the previously unobserved opponent. Should the strategy still be ineffective, then it can be adapted during online play. To continue increasing the robustness of case-based adaptive game AI, it may be beneficial to track *dynamically* whether the projected effectiveness of applying a certain strategy is met, and when it is necessary adapt the strategy directly. In any case, the next time that offline processing is performed, the new game will be included in the case base. As a result, the previously unobserved opponent will be covered in the opponent models, and accordingly game AI that is predictable more effective will be generated automatically.

8.4 Practical applicability

In this section we give an analysis of the practical applicability of case-based adaptive game AI. We first discuss the topic of scalability (8.4.1). Subsequently, we describe ways of dealing with imperfect information in video-game environments (8.4.2). Next, we outline the generalisation of our approach to different games (8.4.3). Finally, we discuss the possible acceptance of the approach by game developers (8.4.4).

8.4.1 Scalability

In experiments discussed in the thesis we demonstrated the effectiveness of case-based adaptive game AI operating in an actual video game. However, there is one aspect of the approach that is not evaluated, namely how much the performance of the approach varies when the size of the case base changes. Basically, we have presented a single point in the learning curve of the system, while the shape of the curve remains to be investigated. Here, two topics are of importance.

First, the effectiveness of behaviour established via a case-based approach largely depends on the *quality* of the cases that are gathered in the case base. For our setup, where game strategies are established from pseudo-randomised self-play, our insight is that for each map several hundred games need to be observed before effective behaviour can be established. In practice, case-based adaptive game AI may be expected to be incorporated in multi-player games. In these games, the case base will grow rapidly, based on observations of many different players competing against other distinct players. Analogously, the case base may be expected to grow rapidly to incorporate qualitatively effective cases.

Second, the *computational efficiency* of case-based adaptive game AI may be hampered by a large size of the case base. In our current setup we were able to establish computationally efficient online adaptation, by performing offline a processing of game observations. To ensure that this procedure remains feasible with increased data sizes, one may opt to enhance the generalisation performed when clustering observations, for instance, by increasing the role of opponent modelling in the clustering process. In addition, one may focus on improving the efficiency of the clustering process, for instance, by applying tree-indexing structures (e.g., *k*d-trees (Bentley, 1975) or cover trees (Beygelzimer *et al.*, 2006)). Indeed, one may opt to reduce specifically the size of the case base, for instance, by offline data compression and subsequent online data decompression (Abou-Samra *et al.*, 2002), and by automatic condensation of the case base (Angiulli and Folino, 2007).

8.4.2 Dealing with imperfect information

Game AI in modern video games usually operates in an imperfect-information environment. This characteristic of the environment provides a serious challenge for game AI. However, in video game practice, the fact that a game presents an imperfect-information environment is not an issue. The reason for this phenomenon is quite conspicuous; game developers choose where necessary, or for the sake of convenience, to sidestep the challenge by providing game AI with perfect information of the environment (Millington, 2006). As information is exploited that strictly speaking should not be available to the game AI, from a purist perspective, this is regarded as a form of cheating behaviour. As stated before (see Subsection 6.2.4) we oppose this opinion. Still, as a scientific endeavour, it certainly is interesting to establish effective, strictly non-cheating AI for video games that present an imperfect-information environment.

However, as stated in Chapter 1, one of the goals for providing entertaining game AI is to obtain effectiveness without cheating *obviously*. Game AI which executes actions that are in principle unavailable, will not be regarded as entertaining. By contrast, game AI that exploits perfect information discreetly in order to provide challenging gameplay, will be generally regarded as entertaining.

In our research we investigated to some extent how effective game AI can be established in a video game with an imperfect-information environment. The evaluation function, discussed in Chapter 5, incorporated a straightforward mechanism to map the imperfect observational data to a prediction of the perfect observational data. Obtained results showed that in an imperfect-information Spring environment, the accuracy of the evaluation function approaches that of the function in a perfect-information environment. With regard to opponent modelling, discussed in Chapter 7, we showed that relatively accurate models of the opponent player can be established in the imperfect-information Spring environment. We leave it for future research to expand our implementation to incorporate the three components into a case-based adaptive game AI for Spring, with the additional requirement that the game AI has to operate on the basis of strictly imperfect information.

If we were to propose an approach to demonstrate case-based adaptive game AI in a video game where only imperfect information is available, our proposal would be to enhance the role of opponent modelling. Consider, for instance, an imperfect-information RTS game. Here, the game strategy of the opponent player will be unclear in the beginning of the game. However, at the end of the game, when the battlefield is revealed, most players will be well able to deduce the game strategy that was followed by the opponent. On the basis of this deduction, the game AI can *a posteriori* map imperfect observational data to models of the opponent's behaviour. The next time that a game is played, these mappings can be used in the game to classify relatively early the opponent player, and adapt the game strategy accordingly.

As the proposal resembles how most human players adapt to opponent behaviour over the course of playing a video game many times (i.e., optimise the own game strategy, while considering the partially observed opponent strategy), we surmise that this proposal may be suitable for establishing effective behaviour in imperfect-information video games.

8.4.3 Generalisation to different games

We demonstrated case-based adaptive game AI in an actual, complex video game: Spring. Indeed, solely on the basis of our experiments we may not draw generic conclusions that apply to all games. However, we would like to reason that case-based adaptive game AI is not bound to specific game genres where it regards offline processing of game observations. For all genres, game observations can be gathered in a case base. This case base can be utilised for providing feedback on distinct strengths and weaknesses of a player, and can provide inexpensive insight into the balance of a game. Thus, it can help in testing and debugging the game.

For online adaptation, case-based adaptive game AI has been shown applicable to an actual RTS game with high complexity. By extrapolation, case-base adaptive game AI is also applicable to different games, and to different game genres, that use game AI with a complexity less than that of RTS games. This is a significant part of video games that are on the market today. Indeed, in implementation of the approach one needs to consider the characteristics of the game under investigation. For instance, in action games such as first-person shooters (FPS), gameplay is more reactive, and is oriented on tactics (instead of strategy). In these games, case-based adaptive game AI should poll the game state frequently, and analogously to modelling the effectiveness of game strategies, it should extensively keep track of the effectiveness of game tactics. In hindsight, case-based adaptive game AI may be suitable for implementation in the Quake III CTF game environment, to adapt the team-oriented behaviour in online play (see Chapter 3).

Here, we make special mention of so-called serious games. These games focus on exploiting game environments for a purpose other than pure entertainment, for instance, for training of medical professionals. Case-based adaptive game AI may be incorporated in serious games to monitor the player's behaviour in the game environment. Dependent on observations on the progress of the player, case-based adaptive game AI can be used to adapt the game environment to ensure that the training goals are achieved.

8.4.4 Acceptance by game developers

Game developers often emphasise that advanced game AI is needed to create game characters that are able to operate consistently in modern video-game environments (i.e., react believably, in a human-like manner) (Rabin, 2004b; Millington, 2006; Church and Pfeifer, 2008). As game environments are becoming more complex and more realistic, an ineffectiveness of the game AI will become directly apparent to the player. For instance, a recent trend in many games is that of deformable or dynamically changing terrain (Rabin, 2008). As a result, game developers strictly require the ability of game characters to adapt adequately to changing circumstances.[1]

To operate consistently in modern video games, the decision-making processes of game AI need to become more advanced. We observe in recent games that, in fact, they are. For instance, the video-game series SKATE (2007-2009) from leading publisher Electronic Arts (EA) incorporates complex locomotion that is able to exploit rapidly physical properties of the presented game environment (Wesley, 2008). Games such as OBLIVION (2006) and the game series THE SIMS (2000-2009) incorporate reputation systems to model the behaviour of game characters, and exploit these models to respond believably to in-game events (Evans, 2007; Sellers, 2008). The video game HALO 3 (2007) uses an advanced scripting technique called 'objective trees' (Isla, 2008). The technique follows a declarative authoring paradigm to ensure that when a player encounters game characters, the characters do not only behave tactically intelligent, but also consider previous experiences with the game environment, and thereby attempt to establish a balanced challenge for the player.

An advanced form of game AI is found in the critically acclaimed video game SPORE (2008), where behaviour of both game characters as well as the game environment is tailored to fit the in-game behaviour of the human player (Grundstrom, 2008). In fact, the game's approach to AI reminds one of case-based adaptive game AI. In a recent interview, Wright (2009), the designer of the game, was quoted stating:

> *"SPORE as a program is not creative at all, but it does a very good job of distilling the creativity of millions of individuals and presenting them back to you. I think you can get a lot more traction using that approach, and I think reversing that we're also starting to look at how we can analyze human metrics inside of a game or any kind of computer experience, and then change that experience to customize it to that person. Using the intelligence of other people is kind of the base data set for that. So I think we're going to see a lot more progress in what we think of as AI from that approach. For the future, there are still people out there fundamentally*

[1] A related observation is that game developers often spend a considerable amount of time in fine-tuning a game's challenge. For instance, at the time of writing, the game STARCRAFT II (2010, expected) is being fine-tuned for already nearly two years. AI techniques, such as automatically generated evaluation functions to express the strengths and weaknesses of units, may assist game developers in this important and time-consuming task.

trying to recreate human intelligence... but they're still on this very slow, linear slope, whereas the other approach is really taking off exponentially."

Considering the necessity of advanced game AI, as well as the observed trend to incorporate advanced game AI into actual, commercially released games, we are optimistic that approaches such as the one investigated in our research have potential to be accepted by game developers. In designing our approach to case-based adaptive game AI, we addressed specifically the aspect of rapid adaptation, while keeping in mind that for reliable adaptation the AI system needs to be controllable by, and predictable to the game developers. These characteristics may appeal to game developers, as it meets the industry belief that behaviour of an artificially intelligent system should lie firmly in the hands of the developers (Isla, 2008).

8.5 Chapter conclusions

In this chapter we discussed two experiments that brought together the three components of case-based adaptive game AI: an evaluation function, an adaptation mechanism, and opponent modelling. We observed that without opponent modelling, case-based adaptive game AI already provides a strong basis for adapting rapidly and reliably the player's behaviour in an actual video game: Spring (Chapter 6). The present chapter focussed particularly on enhancing the approach by incorporating opponent modelling techniques. We performed two experiments that tested the enhanced approach in the complex Spring game, and observed an increased effectiveness of the player's behaviour when the ideas on opponent modelling were incorporated. From the results we may conclude that opponent modelling further improves the strength of case-based adaptive game AI, and thus makes its implementation in an actual video game even more worthwhile.

In the analysis of the practical applicability of case-based adaptive game AI, we discussed four topics, namely (1) scalability, (2) dealing with imperfect information, (3) generalisation to different games, and (4) acceptance by game developers.

With regard to scalability, we noted that, in practice, the case base may be expected to grow rapidly to incorporate qualitatively effective cases. We stated that care should be taken to ensure that the approach remains computationally efficient with increased data sizes.

With regard to dealing with imperfect information, we discussed that, in practice, game developers will opt generally to provide perfect information to the game AI. Still, an enhancement was proposed that may enable the case-based adaptation mechanism to operate effectively in a strictly imperfect-information environment.

With regard to generalisation to different games, we discussed that our approach to adaptive game AI may be applicable to games with a complexity equal or less than that of RTS games. This is a significant part of video games that are on the market today.

With regard to acceptance by game developers, we discussed that we are optimistic on case-based approaches to game AI being implemented in practice. Our optimism is based on the described observation that game developers themselves have shown a clear necessity for implementing advanced game AI. We noted that recently game developers started to incorporate advanced game AI into actual, commercially released video games.

9 Conclusions

Chapter 9 provides a conclusive answer to the research questions and the problem statement posed in Chapter 1. In the chapter we first restate and answer the five research questions (Section 9.1). Subsequently, we translate the answers to the research questions to an answer to the problem statement (Section 9.2). Finally, we provide recommendations for future research (Section 9.3).

9.1 Answers to the research questions

In Section 1.3, we formulated five research questions. This section provides an answer to each of these questions, based on the conclusions from the previous chapters.

> **Research question 1:** *To what extent is incremental adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?*

The answer to the first research question is derived from Chapter 3. In the chapter, we discussed that incremental adaptive game AI is commonly applied for creating adaptive game AI. Following the approach, we established the TEAM and TEAM2 mechanism for online adaptation of game AI. We tested TEAM and TEAM2 in the video game Quake III CTF, and from our experimental results we may conclude that the mechanisms are capable of adapting successfully to changes in the opponent behaviour.

However, application of TEAM and TEAM2 as an online learning mechanism is hampered by occasionally very long learning times due to an improper balance between exploitation and exploration. This issue characteristically follows from the incremental adaptive game AI approach, which requires either (1) a high quality of the domain knowledge used (which generally is unavailable to the AI), or (2) a large number of trials to learn effective behaviour online (which is highly undesirable in an actual video game). From results of the investigation we may therefore conclude that the characteristics of incremental adaptive game AI prohibit our goal of establishing game AI capable of adapting rapidly and reliably to game circumstances.

Therefore, we examined an alternative for the incremental approach, which we coined *case-based adaptive game AI*. For case-based adaptive game AI to be successful in an actual, complex video game, three main components are required. The three components are (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. The components are the subjects of research question 2, 3, and 4.

Research question 2: *To what extent can a suitable evaluation function for a complex video game be established?*

The answer to the second research question is derived from Chapter 5. In the chapter, we reported on our evaluation function for the complex SPRING game. We incorporated machine learning techniques to automatically tune the evaluation function on the basis of a case base of game observations. Experiments that tested the evaluation function showed that just before the game's end the function is able to predict correctly the outcome of the game with an accuracy that approached one hundred per cent. Considering that a SPRING game may be won suddenly, and thus the outcome of the game is difficult to predict, this is a satisfactory result. In addition, the evaluation function made fairly accurate predictions before half of the game was played. From these results, we may conclude that a suitable evaluation function for the complex SPRING game can be established by exploiting a case base of game observations.

Research question 3: *To what extent can a mechanism be employed to provide online adaptation of game AI?*

The answer to the third research question is derived from Chapter 6. In the chapter, we discussed an adaptation mechanism for video games. The mechanism aims at allowing game AI to adapt rapidly and reliably to game circumstances. To this end, it is incorporated in a framework for case-based adaptation. The mechanism exploits game observations that are gathered in the case base to (A) generalise offline over observations, (B) initialise the game AI with a predictably effective game strategy, and (C) adapt online the game AI to game circumstances. The case-based adaptation mechanism was tested on three different maps in the SPRING game. Experiments that tested the adaptation mechanism in online play showed that the mechanism can successfully obtain effective performance. In addition, the adaptation mechanism is capable of upholding a draw for a sustained period of time. From these results, we may conclude that the mechanism for case-based adaptation of game AI provides a strong basis for adapting rapidly and reliably behaviour online, in an actual video game.

Research question 4: *To what extent can models of the opponent player be established and exploited in a complex video game?*

The answer to the fourth research question is derived from Chapter 7. In the chapter, we discussed the technique that enables game AI to establish and exploit models of the opponent player (i.e., opponent modelling). Experiments with establishing opponent models in the complex SPRING game revealed that for the game relatively accurate models of the opponent player can be established. Furthermore, an experiment with exploiting opponent models showed that in the Spring game, exploiting the established opponent models in an informed manner leads to more effective behaviour in online play. From these results, we may conclude that opponent modelling may successfully be incorporated in game AI that operates in actual video games, such as the complex Spring game.

Research question 5: *To what extent is case-based adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?*

The answer to the fifth research question is derived from Chapter 8. In the chapter, we reported on the experiments that integrate the three main components of case-based adaptive game AI, viz. (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. The experiments tested case-based adaptive game AI in the complex Spring game. Without opponent modelling, case-based adaptive game AI already provided a strong basis for adapting rapidly and reliably the player's behaviour in the game. In our case-based approach to game adaptation, opponent models were generated automatically, on the basis of player observations that were gathered in the case base. When enhancing the approach by incorporating opponent modelling, in the experiments, we observed an increased effectiveness of the player's behaviour. From these results, we may conclude that opponent modelling further improves the strength of case-based adaptive game AI, and thus makes its implementation in an actual video game even more worthwhile. From the analysis of the practical applicability of case-based adaptive game AI we may reason that the approach is a strong candidate for incorporation in game AI of the future, i.e., in actual, commercially released video games.

This leads to our answer to the fifth research question. By exploiting a case base of game observations (1) to tune automatically an evaluation function, (2) to determine which game adaptations are effective, and (3) to generate automatically opponent models, case-based adaptive game AI for a complex video game such as Spring is able to adapt rapidly and reliably to game circumstances. We demonstrated case-based adaptive game AI in an actual video game.

9.2 Answer to the problem statement

In this section, we provide an answer to the problem statement posed in Chapter 1. Our answer is based on the answers to the five research questions discussed in the previous section.

Problem statement: *To what extent can adaptive game AI be created with the ability to adapt rapidly and reliably to game circumstances?*

Experiments with the common approach to adaptive game AI, i.e., incremental adaptive game AI, illustrated that the approach can be applied to adapt game AI to some extent. However, application of the incremental approach is limited, for characteristics of the approach prohibit the goal of establishing game AI capable of adapting *rapidly* and *reliably* to game circumstances. We therefore proposed to investigate an alternative approach to adaptive AI that comes without these characteristics. We coined the alternative approach *case-based adaptive game AI.*

With respect to adapting rapidly, we noted that case-based adaptive game AI is capable of exploiting game observations gathered in a case base for the purpose of instant application in game circumstances. The approach builds on the ability to gather and identify relevant game observations, and the ability to apply effectively these observations in similar game circumstances. Our experiments demonstrated that case-based adaptive game AI may be incorporated successfully in an actual video game, for the purpose of adapting rapidly to game circumstances.

With respect to adapting reliably, we noted that considered game adaptations are evaluated intensively within a case-based framework. The decision-making process of case-based adaptive game AI may, in practice, be expected to be based on domain knowledge of high quality, gathered from a multitude of observed games. Gathered domain knowledge is exploited for, among others, (1) rating the game state, (2) tracking the effectiveness of units in the game, (3) adapting automatically the game AI, (4) scaling automatically the difficulty level, and (5) modelling behaviour of the opponent player. As desired, case-based adaptive game AI thereby provides considerable means for game developers to tailor reliably gameplay to the human player.

Given that *case-based adaptive game AI* is capable of adapting to game circumstances rapidly and reliably, and considering that we demonstrated its effectiveness in an actual, complex video game, we may conclude that the approach is a strong candidate to be incorporated in game AI of the future, i.e., in actual, commercially released video games.

9.3 Future research

The research presented in the thesis indicates several important and promising areas of future research. In this section, we mention three of the most interesting areas.

Learning automatically relevant game features: In our research, we investigated case-based adaptive game AI as a proof of concept. However, for none of the three main components of case-based adaptive game AI we attempted to establish the best solution for our problem domain. Indeed, for each component such an investigation would be a thesis in itself. Potentially the largest improvement in the performance of case-based adaptive game AI, we expect, will result from an improved selection in the set and weights of relevant game features (cf., e.g., Thiery and Scherrer, 2009a,b). On the one hand, the improved selection may be established by domain experts who exploit their knowledge with the game. On the other hand, the improved selection may be established by methods to learn automatically relevant game features. As video-game environments are growing increasingly complex, we foresee that game developers will increasingly utilise automatic methods to obtain inexpensive insight into relevant features of the problem domain.

Multifaceted difficulty scaling: In our research we gave brief attention to the topic of scaling the difficulty level to the human player. Our approach to difficulty scaling was straightforward. It was based on the assumption that players may regard game AI that is capable of upholding a certain fitness value as challenging (e.g., upholding a value that represents a draw position). The assumption should be validated if this form of difficulty scaling were to be implemented in commercially released games. Moreover, the challenge provided by a game is typically multifaceted. Our approach to difficulty scaling adapts only the strategical challenge that is provided by the game AI. Therefore, additional facets of challenge remain to be investigated. A case base of game observations may prove instrumental to this investigation.

Adaptation of the game environment: Research in the domain of AI for video games is focussed generally on learning behaviour for game characters. Experts recently pointed out, however, that game characters often do not live long enough to benefit from learning, and argue that it is difficult for human players to detect and understand that game characters are in fact learning (Rabin, 2008). This certainly does not imply that research efforts of many individuals have been in vain, as the design of a game can be adapted to render the benefits of advanced AI more apparent. For instance, developers of the game SPORE found that revealing directly what the AI is "thinking" makes it much more apparent to the player when the AI is doing something smart - which leads to higher satisfaction with the game AI (Lewis, 2009). Still, it is important to take note of the given expert opinion.

Finally, we may conjecture that the game environment has the most significant effect on a player's satisfaction, and to a lesser extent the game characters present in that environment. On this basis, we would like to propose that developed AI techniques should be applied for the purpose of adapting the game environment itself, and not so much for the purpose of adapting the behaviour of game characters in the environment. An interesting and novel direction for future research (as well as for future game design) may be tailoring actual games (i.e., the appearance of the environment, its psychical properties, its rules, etc.) to a set of automatically generated models of the player.

References

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1).

Abou-Samra, S., Comair, C., Champagne, R., Fam, S. T., Ghali, P., Lee, S., Pan, J., and Li, X. (2002). Data compression/decompression based on pattern and symbol run length encoding for use in a portable handheld video game system. USA Patent #6,416,410.

Abt, C. C. (1970). *Serious Games*. The Viking Press, New York City, New York, USA, 1st edition.

Adams, E. and Rollings, A. (2006). *Fundamentals of Game Design*. Game Design and Development. Prentice Hall, Upper Saddle River, New Jersey, USA.

Agarwal, K. (2007). Preloading game demos is a key merchandizing lever to drive purchases. Nielsen Mobile, New York City, New York, USA. [Online]. Available: http://www.telephia.com/html/GDC07_press_release_template.html.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1995). Fast discovery of association rules. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, California, USA.

Aha, D. W., Molineaux, M., and Ponsen, M. J. V. (2005). Learning to win: Case-based plan selection in a real-time strategy game. In Muñoz-Avila, H. and Ricci, F., editors, *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, pages 5–20. DePaul University, Chicago, Illinois, USA.

Alhadeff, E. (2007). Serious games: A sizeable market - Update. Future-Making Serious Games. [Online]. Available: http://elianealhadeff.blogspot.com/2007/06/serious-games-sizeable-market-update.html.

Allen, J. D. (1989). A note on the computer solution of connect-four. In Levy, D. N. L. and Beal, D. F., editors, *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*, pages 134–135. Ellis Horwood, Chichester, UK.

Allen, M. J., Suliman, H., Wen, Z., Gough, N. E., and Mehdi, Q. H. (2001). Directions for future game development. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of the 2nd International Conference of Intelligent Games and Simulation (GAMEON'2001)*, pages 22–32. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Allis, L. V. (1988). A knowledge-based approach of connect four: The game is over, white to move wins. Master's thesis, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands. Report no. IR-163.

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Anantharaman, T. S. (1992). Extension heuristics. *International Computer Chess Association (ICCA) Journal*, 14(2):47–65.

Anantharaman, T. S. (1997). Evaluation tuning for computer chess: Linear discriminant methods. *International Computer Chess Association (ICCA) Journal*, 20(4):224–242.

Angiulli, F. and Folino, G. (2007). Distributed nearest neighbor-based condensation of very large data sets. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1593–1606.

Atkin, L. and Slate, D. (1988). Chess 4.5 - The northwestern university chess program. In Levy, D. N. L., editor, *Computer chess compendium*, pages 80–103. Springer-Verlag, Heidelberg, Germany.

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 Althoff, K.-D., Bergmann, R., Minor, M., and Hanft, A., editors, *Proceedings of the 9th European Conference on Case-Based Reasoning (ECCBR 2008)*, pages 59–73. University of Trier, Trier, Germany.

Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007a). Automatically evaluating the status of an RTS game. In Van Someren, M., Katrenko, S., and Adriaans, P., editors, *Proceedings of the Annual Belgian-Dutch Machine Learning Conference (Benelearn 2007)*, pages 143–144. University of Amsterdam, Amsterdam, The Netherlands.

Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007b). Predicting success in an imperfect-information game. In Van den Herik, H. J., Uiterwijk, J. W. H. M., Winands, M. H. M., and Schadd, M. P. D., editors, *Proceedings of the Computer Games Workshop 2007 (CGW 2007)*, MICC Technical Report Series 07-06, pages 219–230. Maastricht University, Maastricht, The Netherlands.

Bakkes, S. C. J. and Spronck, P. H. M. (2005). Symbiotic learning in commercial computer games. In Mehdi, Q. H., Gough, N. E., and Natkin, S., editors, *Proceedings of the 7th International Conference on Computer Games (CGAMES 2005)*, pages 116–120. University of Wolverhampton, Wolverhampton, UK.

Bakkes, S. C. J. and Spronck, P. H. M. (2006). Gathering and utilising domain knowledge in commercial computer games. In Schobbens, P.-Y., Vanhoof, W., and Schwanen, G., editors, *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2006)*, pages 35–42. University of Namur, Namur, Belgium.

Bakkes, S. C. J. and Spronck, P. H. M. (2008). Automatically generating a score function for strategy games. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 647–658. Charles River Media, Inc., Hingham, Massachusetts, USA.

Bakkes, S. C. J., Spronck, P. H. M., and Postma, E. O. (2004). TEAM: The Team-oriented Evolutionary Adaptability Mechanism. In Rauterberg, M., editor, *Entertainment Computing - ICEC 2004*, volume 3166 of *Lecture Notes in Computer Science*, pages 273–282. Springer-Verlag, Heidelberg, Germany.

Bakkes, S. C. J., Spronck, P. H. M., and Postma, E. O. (2005a). Best-response learning of team behaviour in Quake III. In Aha, D. W., Muñoz-Avila, H., and Van Lent, M., editors, *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 13–18. Navy Center for Applied Research in Artificial Intelligence, Washington, DC., USA.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2005b). Learning to play as a team. In Wanders, A., editor, *Proceedings of the Learning Solutions 2005 symposium*, pages 16–17. SNN Adaptive Intelligence, Nijmegen, The Netherlands.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2007c). Phase-dependent evaluation in RTS games. In Dastani, M. M. and de Jong, E., editors, *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 3–10. Utrecht University, Utrecht, The Netherlands.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008a). Rapid adaptation of video game AI. In Botti, V., Barella, A., and Carrascosa, C., editors, *Proceedings of the 9th International Conference on Intelligent Games and Simulation (GAMEON'2008)*, pages 69–76. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008b). Rapid adaptation of video game AI (Extended version of 2008a). In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 79–86. IEEE Press, Piscataway, New York, USA.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008c). Rapidly adapting game AI. In Nijholt, A., Pantic, M., Poel, M., , and Hondorp, H., editors, *Proceedings of the 20th Belgian-Dutch Artificial Intelligence Conference (BNAIC 2008)*, pages 9–16. University of Twente, Enschede, The Netherlands.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009a). Opponent modelling for case-based adaptive game AI. *Entertainment Computing*, 1(1):27–37.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009b). Rapid and reliable adaptation of video game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104.

Baran, M. and Urbanczyk, M. (2006). AI:TSI. Creator of the game AI 'TSI', [Online]. Available: http://spring.clan-sy.com/wiki/AI:TSI.

Baratz, A. (2001). The stage of the game. Ars Technica. [Online]. Available: http://arstechnica.com/reviews/01q3/gaminghistory/ghistory-1.html.

Baumgarten, R., Colton, S., and Morris, M. (2009). Combining AI methods for learning bots in a real-time strategy game. *International Journal on Computer Game Technologies*, 2009. Article ID 129075. Special issue on Artificial Intelligence for Computer Games.

Baxter, J., Tridgell, A., and Weaver, L. (1998). Experiments in parameter learning using temporal differences. *International Computer Chess Association (ICCA) Journal*, 21(2):84–99.

Beal, D. F. (1984). Mixing heuristic and perfect evaluations: Nested minimax. *International Computer Chess Association (ICCA) Journal*, 7(1):10–15.

Beal, D. F., editor (1986). *Advances in Computer Chess 4*. Elsevier Science Publishers, Amsterdam, The Netherlands.

Beal, D. F. and Smith, M. C. (1994). Random evaluations in chess. *International Computer Chess Association (ICCA) Journal*, 17(1):91–94.

Beal, D. F. and Smith, M. C. (1997). Learning piece values using temportal differences. *International Computer Chess Association (ICCA) Journal*, 20(3):147–151.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In Cohen, W. and Moore, A., editors, *Proceedings of the 23rd international conference on Machine Learning (ICML '06)*, pages 97–104. ACM, New York City, New York, USA.

Billings, D. (2006). *Algorithms and Assessment in Computer Poker*. PhD thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

Blizzard (2008). Press release - World of Warcraft subscriber base reaches 11.5 million worldwide. [Online]. Available: http://eu.blizzard.com/en/press/081223.html.

Böhm, N., Kóokai, G., and Mandl, S. (2005). An evolutionary approach to Tetris. In Doerner, K. F., Gendreau, M., Greistorfer, P., Gutjahr, W. J., Hartl, R. F., , and Reimann, M., editors, *Proceedings of the 6th Metaheuristics International Conference (MIC2005)*, pages 137–148. University of Vienna, Vienna, Austria.

Bouzy, B. and Cazenave, T. (2001). Computer Go: An AI-oriented survey. *Artificial Intelligence*, 132(1):39–103.

Boyd, J. R. (1987). A discourse on winning and losing. Unpublished set of briefing slides available at Air University Library, Maxwell AFB AL, Report no: mu43947.

Bramer, M. A. (1983). *Computer Game-Playing: Theory and Practice*. Ellis Horwood, Chichester, UK.

Brand, S. (1974). *II Cybernetic Frontiers*. Random House, New York City, New York, USA.

Breslow, L. A. and Aha, D. W. (1997). Simplifying decision trees: A survey. *Knowledge Engineering Review*, 12(1):1–40.

Brockington, M. and Darrah, M. (2002). How *not* to implement a basic scripting language. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 548–554. Charles River Media, Inc., Hingham, Massachusetts, USA.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence. Special Issue on Games, Computers and Artificial Intelligence.*, 134(1–2):85–99.

Buro, M. (2003). The evolution of strong Othello programs. In Nakatsu, R. and Hoshino, J., editors, *Entertainment Computing*, volume 112 of *IFIP Advances in Information and Communication Technology*, pages 81–88. Springer-Verlag, Heidelberg, Germany.

Buro, M. (2004). Call for AI research in RTS games. In Fu, D., Henke, S., and Orkin, J., editors, *Proceedings of the AAAI-04 Workshop on Challenges in Game Artificial Intelligence*, pages 481–484. AAAI Press, Menlo Park, California, USA.

Buro, M. and Furtak, T. (2003). RTS games as test-bed for real-time AI research. In Chen, K., Chen, S.-H., Cheng, H.-D., Chiu, D. K. Y., Das, S., Duro, R., Jiang, Z., Kasabov, N., Kerre, E., Leong, H. V., Li, Q., Lu, M., Romay, M. G., Ventura, D., Wang, P. P., and Wu, J., editors, *Proceedings of the 7th Joint Conference on Information Sciences (JCIS 2003)*, pages 481–484. Duke University, Durham, North Carolina, USA.

Buro, M. and Furtak, T. M. (2004). RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, pages 34–41. Simulation Interoperability Standards Organization (SISO), Arlington, Virginia, USA.

Carmel, D. and Markovitch, S. (1993). Learning models of opponent's strategy in game playing. In *Proceedings of AAAI Fall Symposium on Games: Planning and Learning*, pages 140–147, Raleigh, NC.

Carmel, D. and Markovitch, S. (1997). Exploration and adaptation in multiagent systems: A model-based approach. In Pollack, M. E., editor, *Proceedings of the 15th International Joint Conference for Artificial Intelligence (IJCAI 97)*, pages 606–611. Morgan Kaufmann Publishers, San Francisco, California, USA.

Carmel, D. and Markovitch, S. (1998). Pruning algorithms for multi-model adversary search. *Artificial Intelligence*, 99(2):325–355.

Carr, D. (2005). Applying reinforcement learning to Tetris. Department of Computer Science, Rhodes University, Grahamstown, South Africa. Technical Report.

Castronova, E. (2001). Virtual worlds: A first-hand account of market and society on the cyberian frontier. In Sinn, H.-W. and Stimmelmayr, M., editors, *CESifo Working Paper Series No. 618*. Indiana University Bloomington, Bloomington, Indiana, USA. [Online]. Available: http://ssrn.com/abstract=294828.

Castronova, E. (2005). On the research value of large games: Natural experiments in norrath and camelot. In Sinn, H.-W. and Stimmelmayr, M., editors, *CESifo Working Paper Series No. 1621*. Indiana University Bloomington, Bloomington, Indiana, USA. [Online]. Available: http://ssrn.com/abstract=875571.

Chan, B., Denzinger, J., Gates, D., Loose, K., and Buchanan, J. (2004). Evolutionary behavior testing of commercial computer games. In Greenwood, G. W., editor, *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 125–132. IEEE Press, Piscataway, New York, USA.

Charles, D. and Livingstone, D. (2004). AI: The missing link in game interface design. In Rauterberg, M., editor, *Entertainment Computing - ICEC 2004*, volume 3166 of *Lecture Notes in Computer Science*, pages 351–354. Springer-Verlag, Heidelberg, Germany.

Charles, D. and McGlinchey, S. (2004). The past, present and future of artificial neural networks in digital games. In Mehdi, Q. H., Gough, N. E., Natkin, S., and Al-Dabass, D., editors, *Computer Games: Artificial Intelligence, Design and Education*, pages 163–169. University of Wolverhampton, Wolverhampton, UK.

Chaslot, G. M. JB., Bakkes, S. C. J., Szita, I., and Spronck, P. H. M. (2008a). Monte-carlo tree search: A new framework for game AI. In Mateas, M. and Darken, C., editors, *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008)*, pages 216–217. AAAI Press, Menlo Park, California, USA.

Chaslot, G. M. JB., Bakkes, S. C. J., Szita, I., and Spronck, P. H. M. (2008b). Monte-carlo tree search: A new framework for game AI. In Nijholt, A., Pantic, M., Poel, M., and Hondorp, H., editors, *Proceedings of the 20th Belgian-Dutch Artificial Intelligence Conference (BNAIC 2008)*, pages 389–390. University of Twente, Enschede, The Netherlands.

Chaslot, G. M. JB., Winands, M. H. M., Szita, I., and Van den Herik, H. J. (2008c). Cross-entropy for monte-carlo tree search. *International Computer Games Association (ICGA) Journal*, 31(3):145–156.

Chaslot, G. M. JB., Winands, M. H. M., Uiterwijk, J. W. H. M., Van den Herik, H. J., and Bouzy, B. (2008d). Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4(3):343–357.

Chellapilla, K. and Fogel, D. B. (2001). Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, 5(4):422–428.

Church, D. and Pfeifer, B. (2008). High fidelity, believable human characters. Invited lecture at the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008).

Cohen, P. R. (1995). *Emperical Methods for Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, USA.

Consalvo, M. and Dutton, N. (2006). Game analysis: Developing a methodological toolkit for the qualitative study of games. *The International Journal of Computer Game Research*, 6(1). [Online] Available: http://gamestudies.org/0601/articles/consalvo_dutton.

Craw, S., Massie, S., and Wiratunga, N. (2007). Informed case base maintenance: a complexity profiling approach. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1618–1621. AAAI Press, Menlo Park, California, USA.

Crawford, C. (1984). *The Art of Computer Game Design.* Mcgraw-Hill Osborne Media, New York City, New York, USA.

Cummins, L. and Bridge, D. (2009). Maintenance by a committee of experts: The MACE approach to case-base maintenance. In Mcginty, L. and Wilson, D. C., editors, *Proceedings of the 8th International Conference on Case-Based Reasoning (ICCBR 2009)*, pages 120–134. Springer-Verlag, Heidelberg, Germany.

Darken, C. J. and Anderegg, B. G. (2008). Particle filters and simulacra for more realistic opponent tracking. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 419–428. Charles River Media, Inc., Hingham, Massachusetts, USA.

Demaine, E. D., Hohenberger, S., and Liben-Nowell, D. (2002). Tetris is hard, even to approximate. Massachusetts Institute of Technology, Cambridge, Massachusetts, USA. Technical Report MIT-LCS-TR-865.

Demasi, P. and Cruz, A. J. de. O. (2002). Online coevolution for action games. *International Journal of Intelligent Games and Simulation*, 2(3):80–88.

Denzinger, J. and Hamdan, J. (2004). Improving modeling of other agents using tentative stereotypes and compactification of observations. In Liu, J. and Cercone, N., editors, *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004)*, pages 106–112. IEEE Press, Piscataway, New York, USA.

Donkers, H. H. L. M. (2003). *Nosce Hostem - Searching with Opponent Models.* PhD thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Donkers, H. H. L. M. and Spronck, P. H. M. (2006). Preference-based player modeling. In Rabin, S., editor, *AI Game Programming Wisdom 3*, pages 647–659. Charles River Media, Inc., Hingham, Massachusetts, USA.

Donkers, H. H. L. M., Uiterwijk, J. W. H. M., and Van den Herik, H. J. (2001). Probabilistic opponent-model search. *Information Sciences*, 135(3–4):123–149.

Donkers, H. H. L. M., Uiterwijk, J. W. H. M., and Van den Herik, H. J. (2003). Admissibility in opponent-model search. *Information Sciences*, 154(3–4):119–140.

Douglas, A. S. (1954). *Some Computations in Theoretical Physics.* PhD thesis, Faculty of Mathematics, University of Cambridge, Cambridge, UK.

Droste, S. and Fürnkranz, J. (2008). Learning the piece values for three chess variants. *International Computer Games Association (ICGA) Journal*, 31(4):209–232.

Egnor, D. (2000). Iocaine power. *International Computer Games Association (ICGA) Journal*, 23(1):33–35.

Euwe, M., Blaine, M., and Rumble, J. F. S. (1982). *The Logical Approach to Chess.* Dover Publications, Mineola, New York, USA.

Evans, R. (2002). Varieties of Learning. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 571–575. Charles River Media, Inc., Hingham, Massachusetts, USA.

Evans, R. (2007). Exotic AI techniques for Sims 3. Invited lecture at the 3th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2007).

Fagan, M. and Cunningham, P. (2003). Case-based plan recognition in computer games. In Ashley, K. D. and Bridge, D., editors, *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR 2003)*, pages 161–170. Springer-Verlag, Heidelberg, Germany.

Fahey, C. (2003). Computer plays Tetris. [Online]. Available: http://colinfahey.com/tetris/tetris_en.html.

Fairclough, C., Fagan, M., MacNamee, B., and Cunningham, P. (2001). Research directions for AI in computer games. In O'Donoghue, D., editor, *Proceedings of the 12th Irish Conference on Artificial Intelligence & Cognitive Science (AICS 2001)*, pages 333–344. National University of Maynooth, Maynooth, Ireland.

Faltings, B. (1997). Probabilistic indexing for case-based prediction. In Leake, D. B. and Plaza, E., editors, *Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR 1997)*, pages 611–622. Springer-Verlag, Heidelberg, Germany.

Fenley, R. (2002). Baby steps to our future. HAL-PC Magazine. [Online]. Available: http://www.hal-pc.org/journal/02novdec/column/babystep/babystep.html.

Ferguson, D. (2007). Blockdot to participate in serious games panel at SXSW festival. Press release Blockdot.com, January 29, 2007. [Online]. Available: http://www.blockdot.com/news/pressDetail.aspx?NewsId=76.

Fix, E. and Hodges, J.L., Jr. (1951). Discriminatory analysis: Nonparametric discrimination: Consistency properties. Technical Report Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolf Field, Texas, USA.

Flom, L. and Robinson, C. (2004). Using a genetic algorithm to weight an evaluation function for Tetris. Colorado State University, Fort Collins, Colorado, USA. Technical Report.

Frey, P. W. (1984). *Chess Skill in Man and Machine.* Springer-Verlag, Heidelberg, Germany.

Fu, D. and Houlette, R. (2004). Constructing a Decision Tree Based on Past Experience. In Rabin, S., editor, *AI Game Programming Wisdom 2*, pages 567–577. Charles River Media, Inc., Hingham, Massachusetts, USA.

Funge, J. D. (2004). *Artificial Intelligence for Computer Games.* A K Peters, Ltd., Wellesley, Massachusetts, USA.

Furini, M. (2007). Mobile games: What to expect in the near future. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 93–95. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Fürnkranz, J. (2007). Recent advances in machine learning and game playing. *ÖGAI-Journal*, 26(2):19–28.

Fürnkranz, J. and Hüllermeier, E. (2005). Preference learning. *Künstliche Intelligenz*, 19(1):60–61.

Fyfe, C. (2004). Independent component analysis against camouflage. In Mehdi, Q. H., Gough, N. E., Natkin, S., and Al-Dabass, D., editors, *Computer Games: Artificial Intelligence, Design and Education*, pages 259–262. University of Wolverhampton, Wolverhampton, UK.

Gibbons, R. (1992). *A Primer in Game Theory*. Pearson Education Ltd., Harlow, Essex, UK.

Goldsmith, Jr., T. T., Grove, C., and Mann, E. R. (1948). Cathode-ray tube amusement device. USA Patent #2,455,992.

Gomboc, D., Buro, M., and Marsland, T. A. (2005). Tuning evaluation functions by maximizing concordance. *Theoretical Computer Science*, 349(2):202–229.

Graepel, T., Herbrich, R., and Gold, J. (2004). Learning to fight. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, pages 193–200. University of Wolverhampton, Wolverhampton, UK.

Grundstrom, E. (2008). The AI of Spore. Invited lecture at the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008).

Guyon, I. and Elisseeff, A., editors (2002). *JMLR Special Issue on Variable and Feature Selection*, volume 3(Mar). Journal of Machine Learning Research. [Online] Available: http://jmlr.csail.mit.edu/papers/special/feature03.html.

Halck, O. M. and Dahl, F. A. (1999). On classifications of games and evaluation of players - with some sweeping generalizations about the literature. In Fürnkranz, J. and Kubat, M., editors, *Proceedings of the ICML-99 Workshop on Machine Learning in Game Playing*. J. Stefan Institute, Bled, Slovenia.

Harding-Rolls, P. (2009). Subscription MMOGs: Life beyond World of Warcraft. In *Screen Digest*. London, UK. [Online]. Available: http://www.screendigest.com/press/releases/pdf/PR-LifeBeyondWorldOfWarcraft-240309.pdf.

Hartigan, J. A. and Wong, M. A. (1979). A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108.

Hartmann, D. (1987a). How to extract relevant knowledge from grandmaster games. Part 1: Grandmasters have insights - the problem is what to incorporate into practical problems. *International Computer Chess Association (ICCA) Journal*, 10(1):14–36.

Hartmann, D. (1987b). How to extract relevant knowledge from grandmaster games. Part 2: The notion of mobility, and the work of De Groot and Slater. *International Computer Chess Association (ICCA) Journal*, 10(2):78–90.

Hartmann, D. (1989). Notions of evaluation functions tested against grandmaster games. In Beal, D. F., editor, *Advances in Computer Chess 5*, pages 91–141. Elsevier Science Publishers, Amsterdam, The Netherlands.

Houlette, R. (2004). Player modeling for adaptive games. In *AI Game Programming Wisdom 2*, pages 557–566. Charles River Media, Inc., Hingham, Massachusetts, USA.

Hsu, F.-H. (2002). *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion.* Princeton University Press.

Hunicke, R. and Chapman, V. (2004). AI for dynamic difficulty adjustment in games. In *AAAI Workshop on Challenges in Game Artificial Intelligence*, pages 91–96. AAAI Press, Menlo Park, California, USA.

Iida, H., Uiterwijk, J. W. H. M., Van den Herik, H. J., and Herschberg, I. S. (1993). Potential applications of opponent-model search. Part 1: the domain of applicability. *International Computer Chess Association (ICCA) Journal*, 16(4):201–208.

Isla, D. (2008). Halo 3 objective trees: A declarative approach to multiagent coordination. Invited lecture at the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008).

Jansen, P. J. (1992). *Using knowledge about the opponent in game-tree search.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Jansen, P. J. (1993). KQKR: Speculative thwarting a human opponent. *International Computer Chess Association (ICCA) Journal*, 16(1):3–18.

Johnson, S. (2004). Adaptive AI: A practical example. In Rabin, S., editor, *AI Game Programming Wisdom 2*, pages 639–647. Charles River Media, Inc., Hingham, Massachusetts, USA.

Juul, J. (2001). Games telling stories? A brief note on games and narratives, in game studies. *The International Journal of Computer Game Research*, 1(1). [Online] Available: http://gamestudies.org/0101/juul-gts/.

Kaneko, T., Yamaguchi, K., and Kawai, S. (2003). Automated identification of patterns in evaluation functions. In Van den Herik, H. J., Iida, H., and Heinz, E. A., editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 279–298. Springer-Verlag, Heidelberg, Germany.

Kendall, G. (2005). Iterated prisoner's dilemma competition. [Online]. Available: http://www.prisoners-dilemma.com/.

Koller, D. and Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215.

Kolodner, J. (1993). *Case-based reasoning.* Morgan Kaufmann Publishers, San Francisco, California, USA.

Laird, J. E. (2000). Bridging the gap between developers & researchers. *Game Developers Magazine*, 7(8):34.

Laird, J. E. and Van Lent, M. (2001). Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15–25.

Laramée, F. D. (2002). Using *n*-gram statistical models to predict player behavior. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 596–601. Charles River Media, Inc., Hingham, Massachusetts, USA.

Laursen, R. and Nielsen, D. (2005). Investigating small scale combat situations in real-time-strategy computer games. Master's thesis, Department of Computer Science, University of Aarhus, Aarhus, Denmark.

Laviers, K., Sukthankar, G., Klenk, M., Aha, D. W., and Molineaux, M. (2009a). Opponent modeling and spatial similarity to retrieve and reuse superior plays. In Lamontagne, L. and Calero, P. G., editors, *Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning (ICCBR 2009)*, pages 97–106. AAAI Press, Menlo Park, California, USA.

Laviers, K., Sukthankar, G., Molineaux, M., and Aha, D. W. (2009b). Improving offensive performance through opponent modeling. In *Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2009)*.

Leake, D. B. (1996). *Case-based reasoning: Experiences, lessons and future directions*. MIT Press, Cambridge, Massachusetts, USA.

Leake, D. B., Kinley, A., and Wilson, D. (1995). Learning to improve case adaptation by introspective reasoning and CBR. In Veloso, M. M. and Aamodt, A., editors, *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR 1995)*, pages 229–240. Springer-Verlag, Heidelberg, Germany.

Leake, D. B. and Wilson, D. (1999). When experience is wrong: examining CBR for changing tasks and environments. In Althoff, K.-D., Bergmann, R., and Branting, K., editors, *Proceedings of the 3rd International Conference on Case-Based Reasoning (ICCBR 1999)*, pages 218–232. Springer-Verlag, Heidelberg, Germany.

Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. (2009). The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):73–89.

Levy, D. N. L. and Newborn, M. (1991). *How Computers Play Chess*. Computer Science Press, Inc., New York City, New York, USA.

Lewis, M. (2009). AI postmortem: Spore - Inside the GDC 2009. [Online]. Available: http://www.gamedev.net/columns/events/gdc2009/article.asp?id=1742.

Lidén, L. (2004). Artificial stupidity: The art of making intentional mistakes. In Rabin, S., editor, *AI Game Programming Wisdom 2*, pages 41–48. Charles River Media, Inc., Hingham, Massachusetts, USA.

Livingstone, D. and Charles, D. (2004). Intelligent interfaces for digital games. In Fu, D., Henke, S., , and Orkin, J., editors, *Proceedings of the AAAI-04 Workshop on Challenges in Game Artifcial Intelligence*, pages 6–10. AAAI Press, Menlo Park, California, USA.

Livingstone, D. and McGlinchey, S. J. (2004). What believability testing can tell us. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, pages 273–277. University of Wolverhampton, Wolverhampton, UK.

Lopez de Mantaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., Keane, M., Aamodt, A., and Watson, I. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Engineering Review*, 20(3):215–240.

Louis, S. J. and Miles, C. (2005). Playing to learn: Case-injected genetic algorithms for learning to play computer games. *IEEE Transactions on Evolutionary Computation*, 9(6):669–681.

Manovich, L. (2002). *The Language of New Media*. MIT Press, Cambridge, Massachusetts, USA.

Manslow, J. (2002). Learning and adaptation. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 557–566. Charles River Media, Inc., Hingham, Massachusetts, USA.

Mehlkop, B. (2009). Adaptive game AI using opponent modelling. Master's thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Mehta, M., Ontañón, S., and Ram, A. (2009). Authoring behaviors for games using learning from demonstration. In Lamontagne, L. and Calero, P. G., editors, *Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning (ICCBR 2009)*, pages 107–116. AAAI Press, Menlo Park, California, USA.

Millington, I. (2006). *Artificial Intelligence for Games*. Morgan Kaufmann Publishers, San Francisco, California, USA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Gill Series in Computer Science.

Miwa, M., Yokoyama, D., and Chikayama, T. (2006). Automatic construction of static evaluation functions for computer game players. In *Lecture Notes in Computer Science*, volume 4265, pages 332–336. Springer-Verlag, Heidelberg, Germany.

Molyneux, P. D. (2006). The future of game AI - Lecture. Imperial College London, London, UK. October 4, 2006.

Muñoz-Avila, H. and Hüllen, J. (1996). Feature weighting by explaining case-based planning episodes. In Smith, I. F. C. and Faltings, B., editors, *Proceedings of 3rd European Workshop on Case-Based Reasoning (EWCBR-96)*, pages 280–294. Springer-Verlag, Heidelberg, Germany.

Murray, J. H. (1998). *Hamlet on the Holodeck: The Future of Narrative in Cyberspace.* MIT Press, Cambridge, Massachusetts, USA.

Nareyek, A. (2002). Intelligent agents for computer games. In Marsland, T. A. and Frank, I., editors, *Computers and Games, Second International Conference, CG 2002*, volume 2063 of *Lecture Notes in Computer Science*, pages 414–422. Springer-Verlag, Heidelberg, Germany.

Nareyek, A. (2004). Artificial intelligence in computer games: State of the art and future directions. *ACM Queue*, 1(10):58–65.

Nechvatal, J. (1999). *Immersive Ideals / Critical Distances. A study of the Affinity Between Artistic Ideologies Based in Virtual Reality and Previous Immersive Idioms.* PhD thesis, Centre for Advanced Inquiry in the Interactive Arts (CAiiA), University of Wales College, Newport, UK.

Nguyen, T. H., Ekholm, J., and Ingelbrecht, N. (2007). Dataquest insight: More growth ahead for mobile gaming. Press Release Gartner, Inc., Stamford, Connecticut, USA. [Online]. Available: http://www.gartner.com/DisplayDocument?ref=g_search&id=504622.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics.* MIT Press, Cambridge, Massachusetts, USA.

Nowell, T. (2007). AI:NTAI. Creator of the game AI 'NTAI', [Online]. Available: http://spring.clan-sy.com/wiki/AI:NTAI.

Olesen, J. K., Yannakakis, G. N., and Hallam, J. (2008). Real-time challenge balance in an RTS game using rtNEAT. In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 87–94. IEEE Press, Piscataway, New York, USA.

Ontañón, S., Mishra, K., Sugandh, N., and Ram, A. (2007). Case-based planning and execution for real-time strategy games. In Weber, R. O. and Richter, M. M., editors, *Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR 2007)*, pages 164–178. Springer-Verlag, Heidelberg, Germany.

Ontañón, S., Mishra, K., Sugandh, N., and Ram, A. (2009). On-line case-based planning. *Computational Intelligence Journal.* (To appear).

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Co., Reading, Massachusetts, USA.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572.

Perkins, H. (2006). AI:CSAI. Creator of the game AI 'CSAI', [Online]. Available: http://spring.clan-sy.com/wiki/AI:CSAI.

Pollack, J. B. and Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32(3):225–240.

Ponsen, M. J. V. and Spronck, P. H. M. (2004). Improving adaptive game AI with evolutionary learning. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, pages 389–396. University of Wolverhampton, Wolverhampton, UK.

Ponsen, M. J. V., Spronck, P. H. M., Muñoz-Avila, H., and Aha, D. W. (2007). Knowledge acquisition for adaptive game AI. *Science of Computer Programming*, 67(1):59–75.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Rabin, S. (2004a). Filtered randomness for AI decisions and game logic. In Rabin, S., editor, *AI Game Programming Wisdom 2*, pages 71–82. Charles River Media, Inc., Hingham, Massachusetts, USA.

Rabin, S. (2004b). Promising game AI techniques. In Rabin, S., editor, *AI Game Programming Wisdom 2*, pages 15–27. Charles River Media, Inc., Hingham, Massachusetts, USA.

Rabin, S. (2008). Preface. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages ix–xi. Charles River Media, Inc., Hingham, Massachusetts, USA.

Raven, P. H. and Johnson, G. B. (2001). *Biology*. McGraw-Hill Science/Engineering/Math, New York City, New York, USA.

Reibman, A. L. and Ballard, B. W. (1983). Nonminimax search strategies for use against fallible opponents. In Tenenbaum, J. M., editor, *Proceedings of the 3rd National Conference on Artificial Intelligence (AAAI-83)*, pages 338–342. Morgan Kaufmann Publishers, San Francisco, California, USA.

Reth (2007). AI:RAI. Creator of the game AI 'RAI', [Online]. Available: http://spring.clan-sy.com/wiki/AI:RAI.

Richtel, M. (2008). Bid for game maker seen as effort to buy innovation. In *The New York Times*. Arthur Ochs Sulzberger, Jr., New York City, New York, USA. February 26.

Rogers, R. (2005). The business of software - Cost of developing a software game. [Online]. Available: http://discuss.joelonsoftware.com/default.asp?biz.5.180453.13.

Rohs, M. (2007). Preference-based player modelling for Civilization IV. Bachelor's thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Ross, B. H. (1989). Some psychological results on case-based reasoning. In Hammond, K. J., editor, *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 144–147. Morgan Kaufmann Publishers, San Francisco, California, USA.

Sailer, F., Lanctot, M., and Buro, M. (2008). Simulation-based planning in RTS games. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 405–418. Charles River Media, Inc., Hingham, Massachusetts, USA.

Samuel, A. (1959). Some studies in machine learning using the game of Checkers. *IBM Journal*, 3(3):210–229.

Sánchez-Pelegrín, R., Gómez-Martín, M. A., and Díaz-Agudo, B. (2005). A CBR module for a strategy videogame. In Aha, D. W. and Wilson, D., editors, *Proceedings of the 1st Workshop on Computer Gaming and Simulation Environments, 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, pages 217–226. DePaul University, Chicago, Illinois, USA.

Sawyer, B. (2002). Serious games: Improving public policy through game-based learning and simulation. Technical report, Foresight & Governance Project, Woodrow Wilson International Center for Scholars, Washington, DC., USA. [Online]. Available: http://www.wilsoncenter.org/topics/docs/ACF3F.pdf.

Schadd, F. C. (2007). Hierarchical opponent models for real-time strategy games. Bachelor's thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Schadd, F. C., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Opponent modeling in real-time strategy games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 61–68. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Schaeffer, J. (1984). The relative importance of knowledge. *International Computer Chess Association (ICCA) Journal*, 7(3):138–145.

Schaeffer, J. (2001). A gamut of games. *AI Magazine*, 22(3):29–46.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is solved. *Science*, 317(5844):1518–1522.

Schaeffer, J., Lake, R., Lu, P., and Bryant, M. (1996). Chinook: The man-machine world Checkers champion. *AI Magazine*, 17(1):21–29.

Scott, B. (2002). The illusion of intelligence. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 16–20. Charles River Media, Inc., Hingham, Massachusetts, USA.

Seizinger, A. (2006). AI:AAI. Creator of the game AI 'AAI', [Online]. Available: http://spring.clan-sy.com/wiki/AI:AAI.

Sellers, M. (2008). Otello: A next-generation reputation system for humans and NPCs. In Mateas, M. and Darken, C., editors, *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008)*, pages 149–154. AAAI Press, Menlo Park, California, USA.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine, 7th series*, 41(314):256–275.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., and Ram, A. (2007). Transfer learning in real-time strategy games using hybrid CBR/RL. In Veloso, M. M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1041–1046. AAAI Press, Menlo Park, California, USA.

Slagle, J. R. and Dixon, J. K. (1970). Experiments with the M & N tree-searching program. *Communications of the ACM*, 13(3):147–154.

Slater, S. I. (2002). Enhancing the immersive experience. In Mehdi, Q. H., Gough, N. E., and Cavazza, M., editors, *Proceedings of the 3rd International Conference on Intelligent Games and Simulation (GAMEON'2002)*, pages 5–9. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Smyth, B. and Keane, M. T. (1995). Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 377–382. Morgan Kaufmann Publishers, San Francisco, California, USA.

Snider, M. (2002). Where movies end, games begin. In *USA Today*. Gannett Company, McLean, Virginia, USA. May 23.

Snyman, J. A. (2005). *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer-Verlag, Heidelberg, Germany.

Spracklen, D. and Spracklen, K. (1978). *Sargon: A Computer Chess Program*. Hayden Book Company.

Spronck, P. H. M. (2005a). *Adaptive Game AI*. PhD thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Spronck, P. H. M. (2005b). A model for reliable adaptive game intelligence. In Aha, D. W., Muñoz-Avila, H., and Van Lent, M., editors, *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 95–100. Navy Center for Applied Research in Artificial Intelligence, Washington, DC., USA.

Spronck, P. H. M., Ponsen, M. J. V., Sprinkhuizen-Kuyper, I. G., and Postma, E. O. (2006). Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248.

Spronck, P. H. M., Sprinkhuizen-Kuyper, I. G., and Postma, E. O. (2004a). Difficulty scaling of game AI. In Rhalibi, A. E. and Van Welden, D., editors, *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAMEON'2004)*, pages 33–37. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Spronck, P. H. M., Sprinkhuizen-Kuyper, I. G., and Postma, E. O. (2004b). Online adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):45–53.

Stone, R. J. (2005). Serious gaming. *Defence Management Journal*, 31. [Online] Available: http://www.defencemanagement.com/article.asp?id=200&content_name=Education_and_Training&article=5156.

Sugandh, N., Ontañón, S., and Ram, A. (2008). On-line case-based plan adaptation for real-time strategy games. In Fox, D. and Gomes, C. P., editors, *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 702–707. AAAI Press, Menlo Park, California, USA.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

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

Szczepanski, T. and Aamodt, A. (2009). Case-based reasoning for improved micromanagement in real-time strategy games. In Lamontagne, L. and Calero, P. G., editors, *Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning (ICCBR 2009)*, pages 139–148. AAAI Press, Menlo Park, California, USA.

Szita, I. and Lörincz, A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941.

Taylor, L. N. (2002). Video games: Perspective, point-of-view, and immersion. Master's thesis, Graduate Art School, University of Florida, Gainesville, Florida, USA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257—277.

Thiery, C. and Scherrer, B. (2009a). Building controllers for Tetris. *International Computer Games Association (ICGA) Journal*, 32(1):3–11.

Thiery, C. and Scherrer, B. (2009b). Tetris: Comparing the performances. *International Computer Games Association (ICGA) Journal*, 32(1):23–34.

Thue, D. J., Bulitko, V., and Spetch, M. (2008). Player modeling for interactive storytelling: A practical approach. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 633–646. Charles River Media, Inc., Hingham, Massachusetts, USA.

Thunputtarakul, W. and Kotrajaras, V. (2007). Data analysis for ghost AI creation in commercial fighting games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 37–41. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Tomlinson, S. (2003). Working at Thinking About Playing or A year in the life of a Games AI Programmer. In Mehdi, Q. H., Gough, N. E., and Natkin, S., editors, *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAMEON'2003)*, pages 5–12. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Tozour, P. (2002a). The evolution of game AI. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 3–15. Charles River Media, Inc., Hingham, Massachusetts, USA.

Tozour, P. (2002b). The perils of AI scripting. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 541–547. Charles River Media, Inc., Hingham, Massachusetts, USA.

Turing, A. M. (1953). Digital computers applied to games. In Bowden, B. V., editor, *Faster than thought*, pages 286–310. Pitman Publishing, London , UK.

Uiterwijk, J. W. H. M. and Van den Herik, H. J. (1994). Speculative play in computer chess. In Van den Herik, H. J., Herschberg, I. S., and Uiterwijk, J. W. H. M., editors, *Advances in Computer Chess 7*, pages 79–90. Maastricht University, Maastricht, The Netherlands.

Utgoff, P. E. (2001). Feature construction for game playing. In Fürnkranz, J. and Kubat, M., editors, *Machines that Learn to Play Games*, volume 8 of *Advances in Computation: Theory and Practice*, pages 131–152. Nova Scientific Publishers, Hauppauge, New York, USA.

Valkenberg, A. J. J. (2007). Opponent modelling in World of Warcraft. Bachelor's thesis, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands.

Van den Herik, H. J., Donkers, H. H. L. M., and Spronck, P. H. M. (2005). Opponent modelling and commercial games. In Kendall, G. and Lucas, S., editors, *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, pages 15–25. IEEE Press, Piscataway, New York, USA.

Van der Blom, L. L., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Map-adaptive artificial intelligence for video games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 53–60. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Van der Heijden, M. J. M., Bakkes, S. C. J., and Spronck, P. H. M. (2008). Dynamic formations in real-time strategy games. In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 47–54. IEEE Press, Piscataway, New York, USA.

Van der Meulen, M. (1989). Weight assessment in evaluation functions. In Beal, D. F., editor, *Advances in Computer Chess 5*, pages 81–90. Elsevier Science Publishers, Amsterdam, The Netherlands.

Van der Sterren, W. (2002). Squad tactics - Team AI and emergent maneuvers. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 233–246. Charles River Media, Inc., Hingham, Massachusetts, USA.

Van Diepen, P. and Van den Herik, H. J. (1987). *Schaken voor Computers.* Academic Service, Schoonhoven, The Netherlands.

Van Lankveld, G., Spronck, P. H. M., and Van den Herik, H. J. (2009). Incongruity-based adaptive game balancing. In Van den Herik, H. J. and Spronck, P. H. M., editors, *Proceedings of the 12th Advances in Computer Games conference (ACG12).* Tilburg centre for Creative Computing (TiCC), Tilburg University, Tilburg, The Netherlands. (To appear).

Van Waveren, J. M. P. and Rothkrantz, L. J. M. (2001). Artificial player for Quake III Arena. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of the 2nd International Conference on Intelligent Games and Simulation (GAMEON'2001)*, pages 48–55. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Velasco, C. (2007). John Woo's stranglehold music team interview. Music 4 Games. [Online]. Available: http://www.music4games.net/Features_Display.aspx?id=161.

Watson, I. (1999). Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems*, 12(5–6):303–308.

Weigand, H., Wieringa, R. J., Meyer, J.-J. Ch., and Starmans, R. J. C. M. (2005). Lecture material of the SIKS basic course: Research methods and methodology for IKS. SIKS, Utrecht University, Utrecht, The Netherlands.

Wesley, M. (2008). Navigating detailed worlds with a complex, physically driven locomotion: NPC skateboarder AI in EA's skate. In Mateas, M. and Darken, C., editors, *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008)*, pages 155–159. AAAI Press, Menlo Park, California, USA.

Wettschereck, D., Aha, D. W., and Mohri, T. (1997). A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314.

Wolf, M. (2006). Video game hardware, software and services. ABI Research, Oyster Bay, New York, USA. [Online] Available: http://www.abiresearch.com/research/1003212-Video+Game+Hardware,+Software+and+Services.

Woodcock, S. (1999). Game AI: The state of the industry. *Game Developers Magazine*, 6(8). [Online] Available: http://www.gamasutra.com/view/feature/3371/game_ai_the_state_of_the_industry.php.

Woodcock, S. (2002). AI roundtable moderator's report. [Online]. Available: http://www.gameai.com/cgdc02notes.html.

Wray, R. E. and Laird, J. E. (2003). Variability in human behaviour modeling for military simulations. In *Proceedings of the 2003 Behaviour Representation in Modeling and Simulation (BRIMS) Conference.* [Online] Available: http://www.sisostds.org/.

Wright, I. and Marshall, J. (2000). Egocentric AI processing for computer entertainment: A real-time process manager for games. In Mehdi, Q. H., Gough, N. E., and Al-Dabass, D., editors, *Proceedings of the 1st International Conference on Intelligent Games and Simulation (GAMEON'2000)*, pages 34–41. EUROSIS-ETI, Ghent University, Ghent, Belgium.

Wright, W. (2009). Interview: Will Wright on stupid fun club, Spore, iPhone and more. IndustryGamers on Wednesday, June 17, 2009. [Online]. Available: http://www.gamedaily.com/articles/news/interview-will-wright-on-stupid-fun-club-spore-iphone-and-more/?biz=1.

Yampolskiy, R. V. (2007). Human computer interaction based intrusion detection. In *ITNG '07: Proceedings of the International Conference on Information Technology*, pages 837–842. IEEE Press, Piscataway, New York, USA.

Yannakakis, G. N. and Hallam, J. (2007). Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–971.

Yannakakis, G. N. and Hallam, J. (2009). Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133.

Yi, M. (2005). Advertisers pay for video games - Product placement tradition no longer free ride for business. San Francisco Chronicle, Hearst Communications, San Francisco, California, USA. July 25, 2005.

Zarozinski, M. (2002). An open-fuzzy logic library. In Rabin, S., editor, *AI Game Programming Wisdom*, pages 90–101. Charles River Media, Inc., Hingham, Massachusetts, USA.

Zhang, Z. and Yang, Q. (2001). Feature weight maintenance in case bases using introspective learning. *Journal of Intelligent Information Systems*, 16(2):95–116.

A Game environments

In this appendix, we provide a description of the video-game environments that were utilised in the present research. First, we describe QUAKE III CTF (Section A.1). Subsequently, we describe SPRING (Section A.2).

A.1 Quake III CTF

The QUAKE III CTF game environment presents an action game in which the game AI operates on a tactical level. In the experiments reported on in the thesis, the game AI controls four game characters.

The game environment provides multi-player, first-person shooter gameplay. Players of the game operate in relatively minimalist maps, in which they attempt to defeat opponent players by utilising weapons (e.g., a rocket launcher), and so-called power-ups (e.g., a health package). Both the weapons and power-ups are distributed over the map. The most played game mode of QUAKE III is Capture the Flag (CTF). Throughout this thesis we abbreviated ‘the QUAKE III Capture the Flag game mode’ to ’QUAKE III CTF’ to improve readability.

Play of the game: QUAKE III CTF is a team game that is played on symmetrical maps. In the game, two teams are pitted against each other. Each team of players starts at their ’base’, which contains a flag. A team scores a point when it is able to recover the flag from the opponent’s base, while retaining its own flag. The team to first reach a predefined score, or to obtain the most points within a predefined time, wins the game.

States of the game: Play of a CTF game can be abstracted in a finite state machine, where the state in which the game resides determines the tactical behaviour of the game characters. QUAKE III CTF is generally abstracted in a finite state machine consisting of four states, viz. (1) both flags at their base, (2) enemy flag stolen, (3) both flags stolen, and (4) base flag stolen. The relationship between the four states is illustrated in Figure A.1. To play QUAKE II CTF effectively, for each state of the game the game AI needs to be able to perform distinct tasks, such as cooperating with team mates to retrieve the base flag, or cooperating with team mates to attack the opponent players.

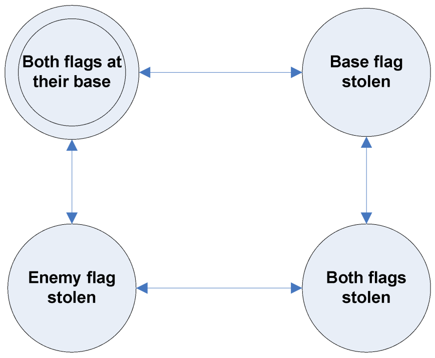


Figure A.1: Finite state machine of QUAKE III CTF. The game starts in the state 'both flags at their base'.

Weapons and power-ups: In QUAKE III CTF, a player is provided with eight distinct weapons that are distributed across the map. In order of increasing effectiveness, the eight weapons are (1) the machine gun, (2) the shotgun, (3) the plasma gun, (4) the lightening gun, (5) the grenade launcher, (6) the rocket launcher, (7) the rail gun, and (8) the BFG.

In addition to weapons, a player can gather power-ups that are distributed across the map. When gathered, the power-ups provide a player with (1) additional health, or (2) additional psychical strength (i.e., so-called armor).

In 2000, parts of the game source code were released by the developers of the game. The released source code enabled enthousiasts to design their own modifications of the game, and provided AI researchers with the means to adapt the behaviour of the QUAKE III CTF game characters. In 2005, the complete source code of the game was released under the GNU General Public License.

A.2 SPRING

The SPRING game environment presents a real-time strategy game in which the game AI operates on a strategic level. In the experiments reported on in the thesis, the game AI controls over one hundred game characters.

Below we provide a description of the game SPRING. We organise our description as follows. First, we outline the play of the game. Second, we describe the unit types that can be employed by the player. Third, we go into the topic of environment visibility. Fourth, we discuss how a so-called tech tree is incorporated in the game. Fifth, we discuss the complexity of the game. Sixth, we give an overview of third-party game AIs that are available for the SPRING game.

Play of the game: SPRING is a typical and open-source RTS game. It presents a strategic, simulated war-game environment. In the game, generally two players are pitted against each other. This type of play is investigated in the thesis. Alternatively, it is also common that two teams of players are pitted against each other.

At the start of the game, each player is provided with one unit, the Commander unit. Each player needs to gather in-game resources for the construction of units and buildings. The in-game resources that can be gathered are (1) metal, which can be gathered directly, and (2) energy, which needs to be generated via wind turbines or solar panels. The game supports over 200 different unit types. For instance, some unit types are directly involved in combat and are mobile (e.g., tank units), where other unit types are not directly involved in combat and have a fixed location (e.g., metal extractor buildings). The goal of the game is to defeat an opponent army in a real-time battle, by using effectively the constructed units and buildings. A SPRING game is won by the player who first destroys the opponent's Commander unit.

Unit types: As mentioned, SPRING supports over 200 different unit types. The unit types available to the player can be roughly divided into two groups, viz. (1) combat units, and (2) building units.

Combat units are units that are directly involved in combat with the opponent player, and are mobile. Units in this group are either (1) k-bot units, (2) tank units, (3) aircraft units, (4) ship units, or (5) amphibious units. Generally, the player can choose to construct many relatively inexpensive combat units (that are relatively powerless), or save in-game resources to construct few relatively expensive combat units (that are relatively powerful). In the SPRING game, the most powerful combat unit is able to destroy a Commander unit with a single shot.

Building units (or in short, buildings) are not directly involved in combat, and have a fixed location. Units in this group are either (1) buildings that gather in-game resources (e.g., a metal extractor), (2) buildings that construct combat units (e.g., a k-bot factory), (3) buildings that defend the base of the player (e.g., a light-laser turret), or (4) buildings that provide additional information to the player (e.g., a radar tower), or hide information from the opponent player (e.g., a radar jammer).

Environment visibility: SPRING implements a so-called 'Line of Sight' visibility mechanism to each unit. This implies that game AI only has access to observational data of those parts of the environment that are visible to its own units (illustrated in Figure A.2). When the game AI's information is restricted to what its own units can observe, we call this an 'imperfect-information environment'. When we allow the game AI to access all information, regardless whether it is visible to its own units or not, we call this a 'perfect-information environment'.

Tech tree: SPRING employs a so-called tech tree (Adams and Rollings, 2006) that represents the possible paths of in-game 'research actions' that a player of the game can take. The name 'tech tree' is common terminology in the domain of video games, and is derived from 'technology tree'. A tech tree is a directed acyclic graph. For all players of the game, the employed tech tree is identical. The manner in which each player traverses the tree is independent from the in-game actions of other players. Each player starts

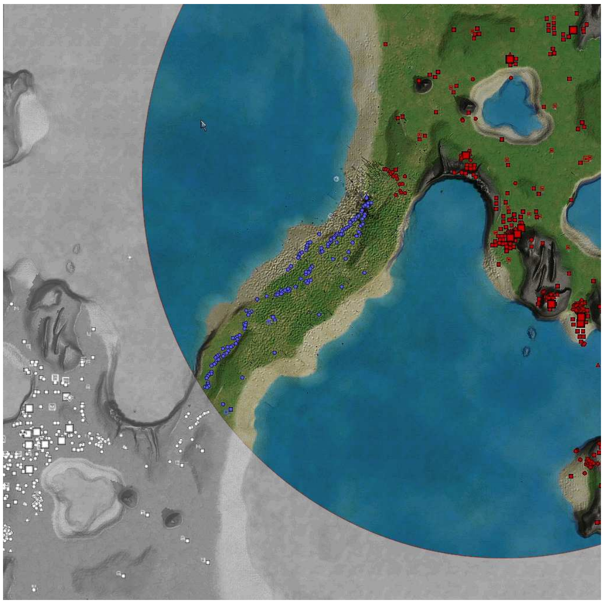


Figure A.2: Game observation in the Spring game. Units controlled by the game AI are currently residing in the highlighted region. In an imperfect-information environment, only information in the highlighted region is available. In a perfect-information environment, information for both the highlighted and the grey region is available.

in the root of the tech tree. By traversing the tech tree, the particular player will be provided with advanced levels of technology. For instance, by performing certain in-game research actions, a player may be provided with more advanced aircraft units. Traversing the tech tree is generally advantageous, yet there is a cost for doing so in time and in in-game resources.

A concise representation of the tech tree of Spring is provided in Figure A.3. In the Spring game, three levels of technology are available: Level-1, Level-2, and Level-3. At the start of the game, each player can only construct Level-1 units and Level-1 buildings. Later in the game, after a player has performed the required in-game research, advanced units and buildings of Level-2 and Level-3 become available to the particular player. For instance, a player may choose to focus on constructing the most advanced Level-3 k-bot units. However, by doing so, a player invests a significant amount of time and in-game resources, that cannot be spent on constructing other potentially helpful units, such as Level-2 tanks and Level-2 aircrafts.

Complexity: In the domain of games, the complexity of a game is generally expressed in terms of (1) the state space, and (2) the action space. In addition to having a large state space (e.g., we experiment with typical 137x137 maps, while maps up to 548x548 are

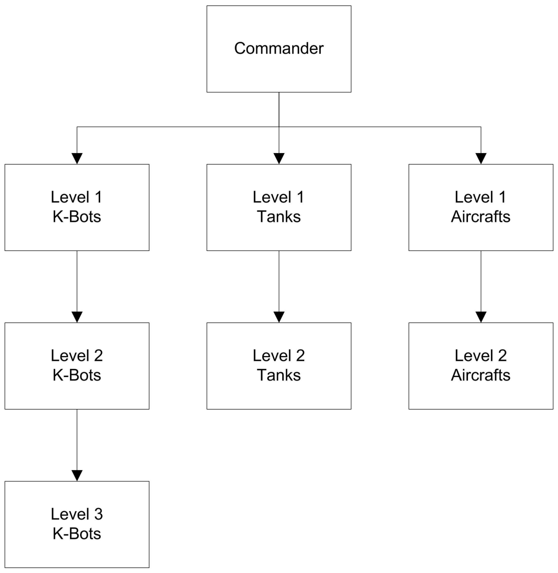


Figure A.3: Concise representation of the tech tree of Spring.

not uncommon, that can involve *hundreds* of units and buildings), Spring's action space is comparatively large.

We roughly estimate the action space in Spring to be $\mathrm{O}(2^W(A{*}\mathrm{P}){+}2^{\mathrm{T}}(\mathrm{D}{+}\mathrm{S}){+}\mathrm{B}(\mathrm{C}))$ (cf. Aha *et al.*, 2005), where W is the current number of workers, A is the number of assignments workers can perform (e.g., construct a building, repair a unit), P is the average number of workplaces, T is the number of troops (combat units plus workers), D is the average number of directions that a unit can move, S is the choice of troop's stance (e.g., stand, patrol, attack), B is the number of buildings, and C is the average choice of units to construct at a building. For a simple early game scenario, this estimate yields a decision complexity of $1.3\text{x}10^7$, which is substantially higher than the average number of possible moves in similar RTS games (e.g., for Wargus, this is approximately $1.5\text{x}10^3$ (Aha *et al.*, 2005)), and many board games (e.g., for chess, this is approximately 30).

In addition, Spring features (1) a partially observable environment that contains adversaries who modify the game state asynchronously (and whose decision models are unknown), and (2) a need for players to make their decisions in real-time (i.e., under severe time constraints) and execute multiple orders simultaneously (cf. Ponsen *et al.*, 2007).

Game AIs: In our experiments with case-based adaptive game AI we used a total of five game AI's, viz. (1) AAI, (2) TSI, (3) CSAI, (4) RAI, and (5) NTAI.

AAI is a configuration file based game AI that was developed by Alexander Seizinger. It may be considered the default game AI of Spring, as it is shipped with the game.

The game AI features powerful group handling, effective resource management, and the ability to learn and adjust its behavior offline on the basis of game observations. It can detect zones of conflict in online play. The configuration files allow AI designers to tweak rates of base expansion and unit production.

TSI is a configuration file based game AI that was developed by Mateusz Baran and Michal Urbańczyk. The game AI uses an extensive pathfinding system that efficiently finds and exploits so-called choke points in the map. Its configuration files allow AI designers to tweak rates of unit production.

CSAI is a generalized game AI that was developed by Hugh Perkins. It is developed as a proof of concept for a C# based game AI. The game AI implements an aggressive and rapid "rush" strategy.

RAI is a generalized game AI that was developed by "Reth". The game AI features effective handling of individual units and of groups of units. The strategy employed by the game AI is focussed on rapid construction of the base, and erecting secondary bases of operation.

NTAI is a configuration file based game AI that was developed by Tom Nowell. The game AI features highly effective resource management, and detection of zones of conflict. The strategy that is followed by the game AI is highly customizable. Its configuration files allow AI designers to tweak rates of unit production. The game AI is bundled with a visual toolkit, that allows AI designers to easily generate configuration files.

B
Features of game observations in SPRING

In this appendix, we provide an overview of all features that are used in the three main components of case-based adaptive game AI. That is, the overview concerns features of game observations in the Spring game. Table B.1 successively lists for each row of the table the name of the feature, a description of the feature, and marks whether the feature was used in our evaluation function (Chapter 5), in our adaptation mechanism (Chapter 6), or in our incorporation of opponent modelling (Chapter 8).

In our research we investigated case-based adaptive game AI as a proof of concept. We focussed primarily on demonstrating the effectiveness of case-based adaptive game AI in an actual video game. However, we did not attempt to establish the best solution for our problem domain. We admit that by manually defining and selecting the game features, we may restrict the effectiveness of case-based adaptive game AI. The investigation of further improvements with respect to the definition and selection of features is considered a topic for future research. We hope that the presented overview of incorporated features provides valuable insight into possible limitations, and enables fellow researchers to establish in the future increasingly effective implementations of case-based adaptive game AI.

Feature	Description	EVA	AM	OM
Visibility	Percentage of the game environment visible to the friendly player	×		
Phase	The phase of the game	×	×	
Material strength	The strength of the player armies	×	×	
Commander safety	The safety of the Commander units	×	×	
Positions captured	The number of map positions captured by the players		×	
Economical strength	The in-game resources generated by the players		×	
Unit count	Number of units constructed by the players		×	
K-Bots	Number of k-bot units constructed by the opponent			×
Tanks	Number of tank units constructed by the opponent			×
Aircrafts	Number of aircraft units constructed by the opponent			×
Advanced buildings	Number of advanced buildings constructed by the opponent			×
Metal extractors	Number of metal extractors constructed by the opponent			×
Solar panels	Number of solar panels constructed by the opponent			×
Wind turbines	Number of wind turbines constructed by the opponent			×
Metal extractors (first attack)	Time of first attack on a friendly-player metal extractor			×
Solar panels (first attack)	Time of first attack on a friendly-player solar panel			×
Wind turbines (first attack)	Time of first attack on a friendly-player wind turbine			×

Table B.1: Features of game observations in SPRING. The table marks whether a feature was used in our evaluation function (EVA), in our adaptation mechanism (AM), or in our incorporation of opponent modelling (OM).

C Evaluation function for SPRING

In this appendix, we give an extensive description of the evaluation function that was established in Chapter 5. The evaluation function for Spring is denoted as follows.

$$v(p) = w_p \sum_{u} w_u (c_{u_1} - \frac{o_{u_2}}{R}) + (1 - w_p) \sum_{r \in d} w_r (\frac{o_{r_2}}{R_{r_2}} - \frac{o_{r_1}}{R_{r_1}}) \tag{C.1}$$

where $w_p \in [0, 1]$ is a free parameter to determine the weight of each term of the evaluation function, $p \in \{1, 2, 3, 4, 5\}$ is a parameter that represents the current *phase of the game*, w_u is the experimentally determined weight of the unit u, c_{u_1} is the number of own units of type u that the game AI has, o_{u_2} is the *observed* number of opponent's units of type u, $R \in [0, 1]$ is the fraction of the environment that is visible to the game AI, w_r is the experimentally determined weight of the radius r, o_{r_2} is the *observed* number of own units of the game AI within a radius r of the opponent's Commander unit, $R_{r_2} \in [0, 1]$ is the fraction of the environment that is visible to the opponent within the radius r, o_{r_1} is the *observed* number of units of the opponent within a radius r of the game AI's Commander unit, $R_{r_1} \in [0, 1]$ is the fraction of the environment that is visible to the game AI within the radius r, and d is the set of experimentally determined game-unit radii $\{500, 1000, 2000\}$.

The unit-type weights w_u are as follows. We note that the unit-type weights have been learned automatically, on the basis of a case base of game observations (see Subsection 5.3.1).

- unit-type[1]=1.2288643
- unit-type[2]=1.0000000
- unit-type[3]=1.0000000
- unit-type[4]=1.3302611
- unit-type[5]=1.0197739
- unit-type[6]=1.0000000
- unit-type[7]=0.87109564
- unit-type[8]=1.805922
- unit-type[9]=1.0000000
- unit-type[10]=2.9940753
- unit-type[11]=-1.2578526
- unit-type[12]=0.42465532
- unit-type[13]=-0.15140171
- unit-type[14]=1.2774299
- unit-type[15]=1.0000000
- unit-type[16]=1.2829862
- unit-type[17]=1.0391217
- unit-type[18]=1.0000000
- unit-type[19]=1.0819025
- unit-type[20]=1.3350561
- unit-type[21]=1.4267497
- unit-type[22]=1.0000000
- unit-type[23]=1.0000000
- unit-type[24]=1.0000000
- unit-type[25]=1.0000000
- unit-type[26]=1.0000000
- unit-type[27]=1.0000000
- unit-type[28]=0.82647168
- unit-type[29]=1.0000000
- unit-type[30]=1.0000000
- unit-type[31]=1.0000000
- unit-type[32]=0.98415256
- unit-type[33]=1.0000000
- unit-type[34]=1.2487913
- unit-type[35]=2.4603451
- unit-type[36]=1.5177068

- unit-type[37]=1.0000000
- unit-type[38]=1.1020264
- unit-type[39]=0.24720546
- unit-type[40]=2.6632323
- unit-type[41]=1.0000000
- unit-type[42]=1.0000000
- unit-type[43]=1.0000000
- unit-type[44]=1.7157615
- unit-type[45]=1.0000000
- unit-type[46]=1.0000000
- unit-type[47]=1.0000000
- unit-type[48]=2.3602369
- unit-type[49]=0.9713257
- unit-type[50]=1.0094703
- unit-type[51]=1.0000000
- unit-type[52]=1.0000000
- unit-type[53]=0.093345074
- unit-type[54]=1.0000000
- unit-type[55]=1.0000000
- unit-type[56]=-1.6993186
- unit-type[57]=1.0000000
- unit-type[58]=0.83349991
- unit-type[59]=1.0000000
- unit-type[60]=1.0000000
- unit-type[61]=0.93975233
- unit-type[62]=1.4669915
- unit-type[63]=1.0000000
- unit-type[64]=1.0000000
- unit-type[65]=1.0000000
- unit-type[66]=1.0000000
- unit-type[67]=1.6751795
- unit-type[68]=4.2321691
- unit-type[69]=1.0162432
- unit-type[70]=2.6487194
- unit-type[71]=1.3491328
- unit-type[72]=1.0000000
- unit-type[73]=1.0000000
- unit-type[74]=1.0000000
- unit-type[75]=1.0000000
- unit-type[76]=1.0000000
- unit-type[77]=2.344249
- unit-type[78]=0.97728515
- unit-type[79]=1.4413914
- unit-type[80]=1.0000000
- unit-type[81]=1.0506602
- unit-type[82]=0.79954942
- unit-type[83]=0.93506761
- unit-type[84]=0.59036946
- unit-type[85]=0.82761479
- unit-type[86]=0.22075553
- unit-type[87]=2.8562736
- unit-type[88]=1.3747437
- unit-type[89]=0.43585614
- unit-type[90]=2.4064984
- unit-type[91]=0.97177108
- unit-type[92]=1.0980737
- unit-type[93]=0.48464166
- unit-type[94]=-0.92639815
- unit-type[95]=2.3913276
- unit-type[96]=1.0000000
- unit-type[97]=1.3108631
- unit-type[98]=1.0000000
- unit-type[99]=0.01186554
- unit-type[100]=1.0000000
- unit-type[101]=0.9369954
- unit-type[102]=1.0000000
- unit-type[103]=1.0000000
- unit-type[104]=1.0000000
- unit-type[105]=1.0000000
- unit-type[106]=-1.1009617
- unit-type[107]=2.6566827
- unit-type[108]=5.9130355
- unit-type[109]=1.0000000
- unit-type[110]=4.2839375
- unit-type[111]=1.0000000
- unit-type[112]=1.3173113
- unit-type[113]=1.2070099
- unit-type[114]=-1.5242612
- unit-type[115]=1.0000000
- unit-type[116]=1.2070843
- unit-type[117]=1.0000000
- unit-type[118]=2.4214381
- unit-type[119]=2.3476243
- unit-type[120]=1.0000000
- unit-type[121]=1.040402
- unit-type[122]=1.0000000
- unit-type[123]=1.0000000
- unit-type[124]=-2.8233522
- unit-type[125]=3.7920158
- unit-type[126]=1.0000000
- unit-type[127]=1.0000000
- unit-type[128]=-0.30900427
- unit-type[129]=1.0000000
- unit-type[130]=1.0000000
- unit-type[131]=1.2790655
- unit-type[132]=1.0000000
- unit-type[133]=1.0000000
- unit-type[134]=1.0000000
- unit-type[135]=1.0000000
- unit-type[136]=1.0000000
- unit-type[137]=0.70938711
- unit-type[138]=1.0000000
- unit-type[139]=1.0000000
- unit-type[140]=1.0000000
- unit-type[141]=1.0000000
- unit-type[142]=1.4531767
- unit-type[143]=0.059104327
- unit-type[144]=1.0000000
- unit-type[145]=0.98732357
- unit-type[146]=1.3830774
- unit-type[147]=1.0000000
- unit-type[148]=0.83156308
- unit-type[149]=4.182812
- unit-type[150]=1.0000000
- unit-type[151]=1.0000000
- unit-type[152]=1.0000000
- unit-type[153]=1.0000000
- unit-type[154]=1.0530515
- unit-type[155]=5.5687555
- unit-type[156]=1.0000000
- unit-type[157]=1.0000000
- unit-type[158]=1.0000000
- unit-type[159]=1.4530096
- unit-type[160]=1.0000000
- unit-type[161]=1.0000000
- unit-type[162]=1.0000000
- unit-type[163]=1.0000000
- unit-type[164]=1.0000000
- unit-type[165]=1.0000000
- unit-type[166]=1.0000000
- unit-type[167]=1.0000000
- unit-type[168]=2.9512551
- unit-type[169]=3.9523296
- unit-type[170]=1.0000000
- unit-type[171]=3.3754926
- unit-type[172]=1.0000000
- unit-type[173]=1.6271645
- unit-type[174]=0.94109938

- unit-type[175]=1.0000000
- unit-type[176]=1.0000000
- unit-type[177]=1.0000000
- unit-type[178]=1.0000000
- unit-type[179]=1.0000000
- unit-type[180]=1.0000000
- unit-type[181]=1.0000000
- unit-type[182]=1.0000000
- unit-type[183]=2.5297929
- unit-type[184]=1.0000000
- unit-type[185]=1.0000000
- unit-type[186]=1.0000000
- unit-type[187]=1.0000000
- unit-type[188]=1.0000000
- unit-type[189]=1.0000000
- unit-type[190]=1.0000000
- unit-type[191]=1.0000000
- unit-type[192]=1.0000000
- unit-type[193]=1.0000000
- unit-type[194]=1.0000000
- unit-type[195]=1.0000000
- unit-type[196]=1.0000000
- unit-type[197]=1.0000000
- unit-type[198]=1.0000000
- unit-type[199]=1.0000000
- unit-type[200]=1.0000000

The radius weights w_r are as follows. We note that the radius weights have been determined by the researcher.

- radius[500]=0.70
- radius[1000]=0.25
- radius[2000]=0.05

The term weights w_p are as follows. We note that the term weights have been learned automatically, on the basis of a case base of game observations (see Subsection 5.3.1).

- phase[1]=-0.370
- phase[2]=0.017
- phase[3]=-0.001
- phase[4]=0.079
- phase[5]=0.041

D Parameters of strategic behaviour in SPRING

In this appendix, we describe the 27 parameters of strategic behaviour that were used in our experiments with case-based adaptive game AI in Spring. The parameters affect the behaviour of the 'AAI (cb)' game AI on a high, strategic level, and not on a low, tactical level. For example, the parameter 'aircraft_rate' determines on a high level how often aircraft units should be constructed. How exactly the constructed aircraft units should be employed is decided by lower-level game AI.

Aircraft_rate: Determines how many air units AAI will build (a value of 7 means that every 7th unit will be an air unit; a value of 1 means that constructing air units is disabled).

Air_defence: Determines how often air-defence units will be built.

Fast_units_rate: Determines the amount of units that will be selected taking their maximum speed into account (e.g., $4 \rightarrow 25\%$).

High_range_units_rate: Determines the amount of units that will be selected taking weapons range into account (e.g., $4 \rightarrow 25\%$).

Max_air_group_size: Maximum air group size.

Max_anti_air_group_size: Maximum size of anti-air groups (ground, hover, or sea).

Max_assistants: Maximum number of builders assisting construction of other units/buildings.

Max_base_size: Maximum base size in sectors.

Max_builders: Maximum builders used at the same time

Max_builders_per_type: How many builders of a certain type may be built.

Max_defences: Maximum number of defences AAI will build in a sector.

Max_factories_per_type: How many factories of a particular type may be built.

Max_group_size: Maximum group size; AAI will create additional groups if all groups of a certain type are full.

Max_metal_cost: Maximum metal cost, units that cost more metal will not be built.

Max_metal_makers: Maximum number of metal makers, set to 0 if you want to disable usage of metal makers.

Max_mex_distance: Tells AAI how many sectors away from its main base it is allowed to build metal extractors.

Max_mex_defence_distance: Maximum distance to base where AAI defends metal extractors with cheap defence-buildings.

Max_scouts: Maximum number of scouts that are used at the same time.

Max_stat_arty: Maximum number of stationary artillery (e.g., big-bertha artillery).

Max_storage: Maximum number of storage buildings.

Min_air_support_efficiency: Minimum efficiency of an enemy unit to call for air support.

Min_assistance_buildspeed: Minimum workertime / buildspeed of a unit to be taken into account when.

Min_factories_for_defences: AAI will not start to build stationary defences before it has built at least that number of factories.

Min_factories_for_storage: AAI will not start to build stationary defences before it has built at least that number of storage buildings.

Min_factories_for_radar_jammer: AAI will not start to build stationary defences before it has built at least that number of radars and jammers.

Min_sector_threat: The higher the value the earlier AAI will stop to build further defences (if it has not already reached the maximum number of defences per sector).

Unit_speed_subgroups: AAI sorts units of the same category (e.g. ground assault units) into different groups according to their max speed (so that slow and fast units are in different groups to prevent the slower ones from arriving in combat much later). This parameter indicates how many different groups will be made.

Summary

Over the last decades, modern video games have become increasingly realistic in their visual and auditory presentation. The games in question generally rely on Artificial Intelligence (AI). However, AI in games has not yet reached a high degree of realism. Now and in the future, game AI may be enhanced by enabling it to adapt intelligently exhibited behaviour to game circumstances. Such enhanced game AI is called 'adaptive game AI'.

Our research is motivated by the fact that, in practice, adaptive game AI in video games is seldom implemented because currently it requires numerous trials to learn effective behaviour in online gameplay (i.e., game adaptation is not rapid). In addition, game developers are concerned that applying adaptive game AI may result in uncontrollable and unpredictable behaviour (i.e., game adaptation is not reliable).

From the above motivation for the research, we derive the following problem statement: *To what extent can adaptive game AI be created with the ability to adapt rapidly and reliably to game circumstances?* To address the problem statement, we first investigate the currently typical approach to adaptive game AI: incremental adaptive game AI. Subsequently, we investigate an alternative, novel approach to adaptive game AI: case-based adaptive game AI.

After providing some background in Chapter 2, we start our research in Chapter 3 by studying RQ1: *To what extent is incremental adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?* To answer the question, we implement the approach in the game QUAKE III CTF. From experiments that test the approach we may conclude that the approach is capable of adapting successfully to changes in the opponent behaviour. However, application of the approach as an *online* learning mechanism is hampered by occasionally very long learning times due to an improper balance between exploitation and exploration. We discuss why this issue characteristically follows from the incremental adaptive game AI approach, which requires either (1) a high quality of the domain knowledge used (which generally is unavailable to the AI), or (2) a large number of trials to learn effective behaviour online (which is highly undesirable in an actual video game). From the results of the chapter we may conclude that the characteristics of incremental adaptive game AI prohibit our goal of establishing game AI capable of adapting rapidly and reliably to game circumstances. Therefore, we examine an alternative for the incremental approach, which we coin *case-based adaptive game AI.*

In Chapter 4 we define case-based adaptive game AI as an approach to game AI where domain knowledge is gathered automatically by the game AI, and is immediately (i.e., without trials and without resource-intensive learning) exploited to create effective behaviour. The approach collects character and game-environment observations, and extracts from those a 'case base'. In the chapter we report on two experiments to obtain an early indication of the effectiveness of case-based adaptive game AI. The results of these two experiments indicate that effective AI in an actual video game may indeed be established by following the approach to case-based adaptive game AI. For case-based adaptive game AI to be successful in an actual, complex video game, three main components are required. The three components are (1) an evaluation function, (2) an adaptation mechanism, and (3) opponent modelling. The three main components are investigated in Chapter 5, 6, and 7, respectively.

In Chapter 5, we study RQ2: *To what extent can a suitable evaluation function for a complex video game be established?* To answer the question, we establish an evaluation function for the Spring game. Spring is an actual, complex real-time strategy (RTS) game. We incorporate machine learning techniques to automatically tune the evaluation function on the basis of a case base of game observations. Experiments that test the evaluation function show that just before the game's end the function is able to predict correctly the outcome of the game with an accuracy that approaches one hundred per cent. Considering that a Spring game may be won suddenly, and thus the outcome of the game is difficult to predict, this is a satisfactory result. In addition, the evaluation function makes fairly accurate predictions before half of the game is played. From these results, we may conclude that a suitable evaluation function for Spring can be established by exploiting a case base of game observations.

In Chapter 6, we study RQ3: *To what extent can a mechanism be employed to provide online adaptation of game AI?* To answer the question, we establish an adaptation mechanism for video games. The mechanism aims at allowing game AI to adapt rapidly and reliably to game circumstances. To this end, it is incorporated in a framework for case-based adaptation. The mechanism exploits game observations that are gathered in a case base to (A) generalise offline over observations, (B) initialise the game AI with a predictably effective game strategy, and (C) adapt online the game AI to game circumstances. The case-based adaptation mechanism is tested on three different maps in the Spring game. Experiments that test the adaptation mechanism in online play show that the mechanism can successfully obtain effective performance. In addition, the adaptation mechanism is capable of upholding a draw for a sustained period of time. From these results, we may conclude that the mechanism for case-based adaptation of game AI provides a strong basis for adapting rapidly and reliably behaviour online, in an actual video game.

In Chapter 7, we study RQ4: *To what extent can models of the opponent player be established and exploited in a complex video game?* To answer the question, we implement techniques to establish and exploit models of the opponent player in the game AI of Spring. Experiments with establishing opponent models in Spring reveal that for the game relatively accurate models of the opponent player can be established. Furthermore, an experiment with exploiting opponent models shows that in Spring, exploiting the established opponent models in an informed manner leads to more effective behaviour in online play. From these results, we may conclude that opponent modelling may successfully be incorporated in game AI that operates in actual video games, such as the complex Spring game.

After the investigation of the three main components of case-based adaptive game AI, in Chapter 8 we study RQ5: *To what extent is case-based adaptive game AI able to adapt rapidly and reliably to game circumstances in an actual video game?* To answer the question, we perform experiments that integrate the three main components of case-based adaptive game AI. The experiments test case-based adaptive game AI in Spring. Without opponent modelling, case-based adaptive game AI already provides a strong basis for adapting rapidly and reliably the player's behaviour in the game. In our case-based approach to adaptive game AI, opponent models are generated automatically, on the basis of player observations that are gathered in the case base. When enhancing the approach by incorporating opponent modelling, in the experiments, we observe an increased effectiveness of the player's behaviour. From these results, we may conclude that opponent modelling further improves the strength of case-based adaptive game AI, and thus makes its implementation in an actual video game even more worthwhile. In addition, we provide an analysis of the practical applicability of case-based adaptive game AI. We discuss four topics, namely (1) scalability, (2) dealing with imperfect information, (3) generalisation to different games, and (4) acceptance by game developers.

Chapter 9 concludes the thesis by answering the five research questions and the problem statement. Given that *case-based adaptive game AI* is capable of adapting to game circumstances rapidly and reliably, and considering that we demonstrated its effectiveness in an actual, complex video game, we may conclude that the approach is a strong candidate to be incorporated in game AI of the future, i.e., in actual, commercially released video games. In addition to the above conclusion, Chapter 9 presents recommendations and ideas for future research.

Samenvatting

Moderne videospelen zijn door de jaren heen steeds realistischer geworden, met name wat hun visuele en auditieve presentatie betreft. Daarentegen heeft de kunstmatige intelligentie die is opgenomen in de spelen nog geen hoge graad van realisme bereikt. Deze zogenoemde *game AI* zal in dit onderzoek en in de toekomst worden verbeterd door de game AI in staat te stellen zich intelligent aan te passen aan de omstandigheden in het spel. Game AI met deze mogelijkheid noemen we *adaptive game AI*. De motivatie van ons onderzoek is gelegen in twee problemen die ervoor zorgen dat in de praktijk adaptive game AI zelden wordt opgenomen in spelen. Het eerste probleem is dat de huidige adaptive game AI een aanzienlijk aantal leermomenten nodig heeft om effectief gedrag te leren tijdens het spelen van een spel (dat wil zeggen, de game AI past zich niet snel genoeg aan de omstandigheden in het spel aan). Het tweede probleem is dat de ontwikkelaars van spelen bezorgd zijn dat het opnemen van adaptive game AI ertoe leidt dat er oncontroleerbaar en onvoorspelbaar gedrag zal worden vertoond (dat wil zeggen, de game AI is niet betrouwbaar).

Vanuit de genoemde motivatie formuleren we onze probleemstelling: *In welke mate kan adaptive game AI worden opgesteld die de mogelijkheid heeft zich snel en betrouwbaar aan te passen aan de omstandigheden in een videospel?* Om de probleemstelling aan te pakken, onderzoeken we eerst een voor de hand liggende, typische benadering om adaptive game AI te verkrijgen: incrementele adaptive game AI. Vervolgens onderzoeken we een alternatieve, nieuwe benadering voor adaptive game AI: case-based adaptive game AI.

In Hoofdstuk 2 beginnen we met een overzicht van gerelateerde achtergrondinformatie. Vervolgens starten we ons onderzoek in Hoofdstuk 3 met het bestuderen van de eerste onderzoeksvraag (RQ, afkorting van het Engelse “research question”), RQ1: *In welke mate is incrementele adaptive game AI in staat zich snel en betrouwbaar aan te passen aan de spelomstandigheden in een bestaand videospel?* Teneinde de vraag te beantwoorden, implementeren we incrementele adaptive game AI in het spel Quake III CTF. Uit experimenten met de incrementele benadering mogen we concluderen dat de benadering in staat is zich succesvol aan te passen aan wisselend gedrag van tegenstanders in het spel. We constateren echter ook dat toepassing van de benadering als een *online* leermechanisme niet tot volle ontplooiing kan komen door incidenteel zeer lange leerperioden. Deze zijn het gevolg van een onjuiste balans tussen exploratie en exploitatie. We beschrijven waarom dit fenomeen op logische wijze volgt uit karakteristieken van de incrementele benadering, welke vereist (1) domeinkennis van een hoge kwaliteit (dit is doorgaans niet beschikbaar voor de game AI), of (2) een aanzienlijk aantal leermomenten om tijdens het spel effectief gedrag te leren (dit is hoogst ongewenst in een videospel). Uit de resultaten van het hoofdstuk mogen we concluderen dat de karakteristieken van de incrementele benadering het bereiken van ons doel belemmeren. Het doel is hier het ontwikkelen van game AI die in staat is zich snel en betrouwbaar aan te passen aan de spelomstandigheden. Derhalve stellen we een alternatief op voor de incrementele benadering, die we *case-based adaptive game AI* noemen.

In Hoofdstuk 4 definiëren we case-based adaptive game AI als een benadering voor game AI waarin domeinkennis automatisch wordt vergaard door de AI, en direct (dat wil zeggen, zonder de benodigde leermomenten) wordt geëxploiteerd om effectief gedrag op te stellen.

De benadering verzamelt spelkarakter- en spelomgevingsobservaties, en leidt daaruit een *case base* af. In het hoofdstuk doen we verslag van twee experimenten die we uitvoeren om een vroege indicatie van de effectiviteit van case-based adaptive game AI te verkrijgen. De resultaten van de twee experimenten geven aan dat een effectieve AI in een modern videospel inderdaad opgesteld kan worden door de *case-based* benadering tot adaptive game AI te volgen. Om succesvol te zijn in een modern, complex videospel, vereist case-based adaptive game AI drie componenten. Deze drie componenten zijn (1) een evaluatiefunctie, (2) een adaptatiemechanisme, en (3) modellering van de tegenstander. Deze drie componenten worden achtereenvolgens onderzocht in Hoofdstuk 5, 6, en 7.

In Hoofdstuk 5 onderzoeken we RQ2: *In welke mate kan een geschikte evaluatiefunctie voor een complex videospel worden opgesteld?* Teneinde de vraag te beantwoorden, ontwerpen we een evaluatiefunctie voor het spel Spring. Spring is een modern, complex real-time strategy (RTS) spel. We gebruiken *machine learning* technieken om de functie automatisch af te stellen op basis van een case base van spelobservaties. Experimenten met de evaluatiefunctie laten zien dat de functie vlak voor het einde van het spel de uitslag correct kan voorspellen met een nauwkeurigheid die de honderd procent benadert. Aangezien Spring plotseling gewonnen kan worden, en de uitkomst van het spel derhalve lastig is te voorspellen, is dit resultaat tevredenstellend. Bovendien doet de evaluatiefunctie tamelijk nauwkeurige voorspellingen voordat de helft van het spel is gespeeld. Uit deze resultaten mogen we concluderen dat een geschikte evaluatiefunctie voor Spring kan worden opgesteld door het exploiteren van een case base van spelobservaties.

In Hoofdstuk 6 onderzoeken we RQ3: *In welke mate kan een mechanisme worden aangewend dat voorziet in online adaptatie van game AI?* Teneinde de vraag te beantwoorden, stellen we een adaptatiemechanisme op voor videospelen. Het mechanisme is erop gericht game AI in staat te stellen zich snel en betrouwbaar aan te passen aan de omstandigheden in het spel. Om dit te bereiken is het opgenomen in een raamwerk voor case-based adaptatie. Het mechanisme exploiteert spelobservaties die zijn verzameld in een case base voor (A) offline generalisatie over spelobservaties, (B) initialisatie van game AI met een naar verwachting effectieve spelstrategie, en (C) online adaptatie van de game AI naar de omstandigheden in het spel. Het case-based adaptatiemechanisme is getest op drie verschillende spelomgevingen in Spring. Experimenten die het adaptatiemechanisme testen tijdens spelen van het spel, laten zien dat het mechanisme effectief gedrag oplegt aan de game AI. Het adaptatiemechanisme is bovendien in staat een gelijkspel in stand te houden voor een onafgebroken tijdsperiode. Uit deze resultaten mogen we concluderen dat het mechanisme voor case-based adaptatie van game AI een solide basis biedt voor snelle en betrouwbare online adaptatie van gedrag in een echt videospel.

In Hoofdstuk 7 onderzoeken we RQ4: *In welke mate kunnen modellen van de tegenstander worden opgesteld en geëxploiteerd in een complex videospel?* Teneinde de vraag te beantwoorden, implementeren we technieken tot opstellen en exploitatie van modellen van de tegenstander in de game AI van Spring. Experimenten met het opstellen van tegenstandermodellen in Spring laten zien dat in het spel relatief nauwkeurige modellen van de tegenstander kunnen worden opgesteld. Een experiment met het exploiteren van tegenstandermodellen laat daarnaast zien dat in Spring het der zake kundig exploiteren van tegenstandermodellen tot meer effectief gedrag leidt tijdens het spelen van het spel. Uit deze resultaten mogen we concluderen dat modellering van de tegenstander succesvol kan worden opgenomen in game AI welke opereert in echte videospelen, zoals het complexe Spring.

Na het onderzoeken van de drie componenten van case-based adaptive game AI, onderzoeken we in Hoofdstuk 8 RQ5: *In welke mate is case-based adaptive game AI in staat zich snel en betrouwbaar aan te passen aan de spelomstandigheden in een bestaand videospel?* Teneinde de vraag te beantwoorden, voeren we experimenten uit met een adaptive game AI waarin de drie componenten van case-based adaptive game AI zijn geïntegreerd. De experimenten testen case-based adaptive game AI in Spring. Zonder modellering van de tegenstander is case-based adaptive game AI reeds in staat een solide basis te bieden voor snelle en betrouwbare aanpassing van game AI. In onze case-based benadering tot adaptive game AI worden tegenstandermodellen automatisch gegenereerd, op basis van spelobservaties die zijn verzameld in de case base. Bij exploitatie van deze modellen nemen we een toename van de effectiviteit van case-based adaptive game AI waar. Uit deze resultaten mogen we concluderen dat het modelleren van de tegenstander de kracht van case-based adaptive game AI verder verbetert. Derhalve maakt het de implementatie van case-based adaptive game AI in een bestaand spel nog meer lonend. In aanvulling op deze conclusie geven we een analyse van de praktische toepasbaarheid van case-based adaptive game AI, waarin we vier onderwerpen bespreken: (1) schaalbaarheid, (2) omgaan met imperfecte informatie, (3) generalisatie naar andere spelen, en (4) acceptatie door spelontwikkelaars.

In Hoofdstuk 9 sluiten we het proefschrift af door de vijf onderzoeksvragen en de probleemstelling te beantwoorden. Gezien het feit dat case-based adaptive game AI in staat blijkt zich snel en betrouwbaar aan te passen aan de spelomstandigheden, en gezien het feit dat we de effectiviteit van de benadering hebben gedemonstreerd in een echt, complex videospel, mogen we concluderen dat de benadering een sterke kandidaat is voor inlijving in game AI van de toekomst, dat wil zeggen, in moderne, commercieel uitgebrachte videospelen. In aanvulling op deze conclusie, presenteren we in Hoofdstuk 9 aanbevelingen en ideeën voor toekomstig onderzoek.

Curriculum vitae

Sander Bakkes was born in Swalmen, the Netherlands, on January 27, 1980. He studied Information Technology (IT) at the Eindhoven Fontys University for Applied Sciences, where he received his B.Sc. degree in 2001 with a specialisation in Telematics. During these studies, he successfully completed internship research at Alcoa, Atos Origin, and Océ R&D. Subsequently, he studied Knowledge Engineering at Maastricht University, where he received his M.Sc. degree in 2003 with a specialisation in Artificial Intelligence (AI). During these studies, he investigated AI for team-oriented video games, under the daily supervision of Dr. ir. P.H.M. Spronck.

After his graduation, he took up the position of IT engineer at Unilogic Networks. In addition, he ventured to a small language school near Taipei (Taiwan), to be active as a teacher of the English language, on a voluntary basis.

Three days after his return from Taiwan, he joined Maastricht University as a Ph.D. candidate, supervised by Prof. dr. H.J. van den Herik, in cooperation (daily supervision) with Dr. ir. P.H.M. Spronck. Later, he joined his supervisors in their move to Tilburg University, to the newly established Tilburg centre for Creative Computing (TiCC). As a Ph.D. candidate, Sander investigated intensively AI for video games. He was funded by the Netherlands Organisation for Scientific Research (NWO), in the framework of the ROLEC project (grant number 612.066.406). In addition, he received funding from the Dutch Ministry of Economic Affairs, in the framework of the Interactive Collaborative Information Systems (ICIS) project (grant number BSIK03024).

His research was published in international refereed journals, in the book series 'AI Game Programming Wisdom', and in the proceedings of numerous (international) conferences and workshops. His work 'Rapid Adaptation of Video Game AI' received the award for best paper at the 9th International Conference on Intelligent Games and Simulation (GAMEON'2008). As of 2009, he is a reviewer of the IEEE Transactions on Computational Intelligence and AI in Games.

Besides performing research, he was involved in lecturing (mainly in the courses Games and AI, Informatics, and Philosophy of Science), guiding skills classes, and supervising and cooperating with three Bachelor students, as well as two Master students. In addition, he took a modest role in organising the DIR 2008 workshop, the ALAMAS 2007 symposium, and the MICC-IKAT Promovendidag 2006. Outside the academia, Sander freelanced as a photographer, and as a DJ.

Currently, Sander is working as a post-doctoral researcher at the Digital Life Centre of the Amsterdam University of Applied Sciences (HvA). The general focus of his research is applying AI techniques for the purpose on improving the quality of life of senior citizens. The research is performed in close collaboration with Vivium Zorggroep, and the University of Amsterdam.

Publications

The scientific work performed during the author's Ph.D. research resulted in the following publications.

Journal articles

1. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009a). Opponent modelling for case-based adaptive game AI. *Entertainment Computing*, 1(1):27–37.
2. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009b). Rapid and reliable adaptation of video game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104.

Book chapters

3. Bakkes, S. C. J. and Spronck, P. H. M. (2008). Automatically generating a score function for strategy games. In Rabin, S., editor, *AI Game Programming Wisdom 4*, pages 647–658. Charles River Media, Inc., Hingham, Massachusetts, USA.

Conference and workshop proceedings

4. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008a). Rapid adaptation of video game AI. In Botti, V., Barella, A., and Carrascosa, C., editors, *Proceedings of the 9th International Conference on Intelligent Games and Simulation (GAMEON'2008)*, pages 69–76. EUROSIS-ETI, Ghent University, Ghent, Belgium.
5. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008b). Rapid adaptation of video game AI (Extended version of 2008a). In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 79–86. IEEE Press, Piscataway, New York, USA.
6. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2008c). Rapidly adapting game AI. In Nijholt, A., Pantic, M., Poel, M., , and Hondorp, H., editors, *Proceedings of the 20th Belgian-Dutch Artificial Intelligence Conference (BNAIC 2008)*, pages 9–16. University of Twente, Enschede, The Netherlands.
7. Chaslot, G. M. JB., Bakkes, S. C. J., Szita, I., and Spronck, P. H. M. (2008a). Monte-carlo tree search: A new framework for game AI. In Mateas, M. and Darken, C., editors, *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008)*, pages 216–217. AAAI Press, Menlo Park, California, USA.
8. Chaslot, G. M. JB., Bakkes, S. C. J., Szita, I., and Spronck, P. H. M. (2008b). Monte-carlo tree search: A new framework for game AI. In Nijholt, A., Pantic, M., Poel, M., and Hondorp, H., editors, *Proceedings of the 20th Belgian-Dutch Artificial Intelligence Conference (BNAIC 2008)*, pages 389–390. University of Twente, Enschede, The Netherlands.

9. Van der Heijden, M. J. M., Bakkes, S. C. J., and Spronck, P. H. M. (2008). Dynamic formations in real-time strategy games. In Hingston, P. and Barone, L., editors, *Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08)*, pages 47–54. IEEE Press, Piscataway, New York, USA.
10. Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007a). Automatically evaluating the status of an RTS game. In Van Someren, M., Katrenko, S., and Adriaans, P., editors, *Proceedings of the Annual Belgian-Dutch Machine Learning Conference (Benelearn 2007)*, pages 143–144. University of Amsterdam, Amsterdam, The Netherlands.
11. Bakkes, S. C. J., Kerbusch, P., Spronck, P. H. M., and Van den Herik, H. J. (2007b). Predicting success in an imperfect-information game. In Van den Herik, H. J., Uiterwijk, J. W. H. M., Winands, M. H. M., and Schadd, M. P. D., editors, *Proceedings of the Computer Games Workshop 2007 (CGW 2007)*, MICC Technical Report Series 07-06, pages 219–230. Maastricht University, Maastricht, The Netherlands.
12. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2007c). Phase-dependent evaluation in RTS games. In Dastani, M. M. and de Jong, E., editors, *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 3–10. Utrecht University, Utrecht, The Netherlands.
13. Schadd, F. C., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Opponent modeling in real-time strategy games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 61–68. EUROSIS-ETI, Ghent University, Ghent, Belgium.
14. Van der Blom, L. L., Bakkes, S. C. J., and Spronck, P. H. M. (2007). Map-adaptive artificial intelligence for video games. In Roccetti, M., editor, *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON'2007)*, pages 53–60. EUROSIS-ETI, Ghent University, Ghent, Belgium.
15. Bakkes, S. C. J. and Spronck, P. H. M. (2006). Gathering and utilising domain knowledge in commercial computer games. In Schobbens, P.-Y., Vanhoof, W., and Schwanen, G., editors, *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2006)*, pages 35–42. University of Namur, Namur, Belgium.
16. Bakkes, S. C. J., Spronck, P. H. M., and Postma, E. O. (2005a). Best-response learning of team behaviour in Quake III. In Aha, D. W., Muñoz-Avila, H., and Van Lent, M., editors, *Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 13–18. Navy Center for Applied Research in Artificial Intelligence, Washington, DC., USA.
17. Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2005b). Learning to play as a team. In Wanders, A., editor, *Proceedings of the Learning Solutions 2005 symposium*, pages 16–17. SNN Adaptive Intelligence, Nijmegen, The Netherlands.
18. Bakkes, S. C. J. and Spronck, P. H. M. (2005). Symbiotic learning in commercial computer games. In Mehdi, Q. H., Gough, N. E., and Natkin, S., editors, *Proceedings of the 7th International Conference on Computer Games (CGAMES 2005)*, pages 116–120. University of Wolverhampton, Wolverhampton, UK.

SIKS dissertation series

1998

1 Johan van den Akker (CWI) *DEGAS - An Active, Temporal Database of Autonomous Objects*

2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

4 Dennis Breuker (UM) *Memory versus Search in Games*

5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*

2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*

3 Don Beal (UM) *The Nature of Minimax Search*

4 Jacques Penders (UM) *The Practical Art of Moving Physical Objects*

5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*

7 David Spelt (UT) *Verification Support for Object Database Design*

8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

2000

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2 Koen Holtman (TU/e) *Prototyping of CMS Storage Management*

3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*

4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*

6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*

7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*

8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*

9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*

10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*

11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*

2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*

3 Maarten van Someren (UvA) *Learning as Problem Solving*

4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*

6 Martijn van Welie (VU) *Task-Based User Interface Design*

7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*

8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*

Abbreviations. SIKS – Dutch Research School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; KUB – Katholieke Universiteit Brabant, Tilburg; KUN – Katholieke Universiteit Nijmegen; OU – Open Universiteit Nederland; RUG – Rijksuniversiteit Groningen; RUL – Rijksuniversiteit Leiden; RUN – Radboud Universiteit Nijmegen; TUD – Technische Universiteit Delft; TU/e – Technische Universiteit Eindhoven; UL – Universiteit Leiden; UM – Universiteit Maastricht; UT – Universiteit Twente; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; UvT – Universiteit van Tilburg; VU – Vrije Universiteit, Amsterdam.

9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*

11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*

2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*

3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*

4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*

5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments Inhabited by Privacy-Concerned Agents*

6 Laurens Mommers (UL) *Applied Legal Epistemology; Building a Knowledge-based Ontology of the Legal Domain*

7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TU/e) *A Reference Architecture for Adaptive Hypermedia Applications*

14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

4 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law – A Modelling Approach*

6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*

7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

8 Yong-Ping Ran (UM) *Repair-Based Scheduling*

9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*

10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*

13 Jeroen Donkers (UM) *Nosce Hostem – Searching with Opponent Models*

14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*

16 Menzo Windhouwer (CWI) *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouse*

17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*

3 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*

4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*

5 Viara Popova (EUR) *Knowledge Discovery and Monotonicity*

6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*

7 Elise Boltjes (UM) *Voorbeeld$_{IG}$ Onderwijs; Voorbeeldgestuurd Onderwijs, een Opstap naar Abstract Denken, vooral voor Meisjes*

8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale Politiële Gegevensuitwisseling en Digitale Expertise*

9 Martin Caminada (VU) *For the Sake of the Argument; Explorations into Argument-based Reasoning*

10 Suzanne Kabel (UvA) *Knowledge-rich Indexing of Learning-objects*

11 Michel Klein (VU) *Change Management for Distributed Ontologies*

12 The Duy Bui (UT) *Creating Emotions and Facial Expressions for Embodied Agents*

13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*

14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*

15 Arno Knobbe (UU) *Multi-Relational Data Mining*

16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*

17 Mark Winands (UM) *Informed Search in Complex Games*

18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*

19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*

20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*

2 Erik van der Werf (UM) *AI techniques for the game of Go*

3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*

4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*

5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*

6 Pieter Spronck (UM) *Adaptive Game AI*

7 Flavius Frasincar (TU/e) *Hypermedia Presentation Generation for Semantic Web Information Systems*

8 Richard Vdovjak (TU/e) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*

9 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*

10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*

11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*

12 Csaba Boer (EUR) *Distributed Simulation in Industry*

13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*

14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*

15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*

16 Joris Graaumans (UU) *Usability of XML Query Languages*

17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*

18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*

19 Michel van Dartel (UM) *Situated Representation*

20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*

21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TU/e) *Foundations of B2B Electronic Contracting*

2 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*

3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*

4 Marta Sabou (VU) *Building Web Service Ontologies*

5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*

6 Ziv Baida (VU) *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*

7 Marko Smiljanic (UT) *XML schema matching – balancing efficiency and effectiveness by means of clustering*

8 Eelco Herder (UT) *Forward, Back and Home Again - Analyzing User Behavior on the Web*

9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*

10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*

11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*

12 Bert Bongers (VU) *Interactivation - Towards an e-cology of people, our technological environment, and the arts*

13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*

14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*

15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*

16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*

17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*

18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*

19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*

20 Marina Velikova (UvT) *Monotone models for prediction in data mining*

21 Bas van Gils (RUN) *Aptness on the Web*

22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*

23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*

24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*

25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*

26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*

27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*

28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures*

2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*

3 Peter Mika (VU) *Social Networks and the Semantic Web*

4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*

5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*

6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*

7 Natasa Jovanovic' (UT) *To Whom It May Concern - Addressee Identification in Face-to-Face Meetings*

8 Mark Hoogendoorn (VU) *Modeling of Change in Multi-Agent Organizations*

9 David Mobach (VU) *Agent-Based Mediated Service Negotiation*

10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*

11 Natalia Stash (TU/e) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*

12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*

13 Rutger Rienks (UT) *Meetings in Smart Environments; Implications of Progressing Technology*

14 Niek Bergboer (UM) *Context-Based Image Analysis*

15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*

16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*

17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*

18 Bart Orriens (UvT) *On the development and management of adaptive business collaborations*

19 David Levy (UM) *Intimate relationships with artificial partners*

20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*

21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*

22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*

23 Peter Barna (TU/e) *Specification of Application Logic in Web Information Systems*

24 Georgina Ramírez Camps (CWI) *Structural Features in XML Retrieval*

25 Joost Schalken (VU) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbán (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*

2 Alexei Sharpanskykh (VU) *On Computer-Aided Methods for Modeling and Analysis of Organizations*

3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*

4 Ander de Keijzer (UT) *Management of Uncertain Data - towards unattended integration*

5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*

6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*

7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*

8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*

9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*

10 Wauter Bosma (UT) *Discourse oriented Summarization*

11 Vera Kartseva (VU) *Designing Controls for Network Organizations: a Value-Based Approach*

12 Jozsef Farkas (RUN) *A Semiotically oriented Cognitive Model of Knowlegde Representation*

13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*

14 Arthur van Bunningen (UT) *Context-Aware Querying; Better Answers with Less Effort*

15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*

16 Henriette van Vugt (VU) *Embodied Agents from a User's Perspective*

17 Martin Op't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*

18 Guido de Croon (UM) *Adaptive Active Vision*

19 Henning Rode (UT) *From document to entity retrieval: improving precision and performance of focused text search*

20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met een overheid op de administratieve lasten van bedrijven*

21 Krisztian Balog (UvA) *People search in the enterprise*

22 Henk Koning (UU) *Communication of IT-architecture*

23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*

24 Zharko Aleksovski (VU) *Using background knowledge in ontology matching*

25 Geert Jonker (UU) *Efficient and Equitable exchange in air traffic management plan repair using spender-signed currency*

26 Marijn Huijbregts (UT) *Segmentation, diarization and speech transcription: surprise data unraveled*

27 Hubert Vogten (OU) *Design and implementation strategies for IMS learning design*

28 Ildiko Flesh (RUN) *On the use of independence relations in Bayesian networks*

29 Dennis Reidsma (UT) *Annotations and subjective machines- Of annotators, embodied agents, users, and other humans*

30 Wouter van Atteveldt (VU) *Semantic network analysis: techniques for extracting, representing and querying media content*

31 Loes Braun (UM) *Pro-active medical information retrieval*

32 Trung B. Hui (UT) *Toward affective dialogue management using partially observable markov decision processes*

33 Frank Terpstra (UvA) *Scientific workflow design; theoretical and practical issues*

34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*

35 Benjamin Torben-Nielsen (UvT) *Dendritic morphology: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*

2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*

3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*

4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*

5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*

6 Muhammad Subianto (UU) *Understanding Classification*

7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*

8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*

9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*

10 Jan Wielemaker (UVA) *Logic programming for knowledge-intensive interactive applications*

11 Alexander Boer (UVA) *Legal Theory, Sources of Law & the Semantic Web*

12 Peter Massuthe (TU/e, Humboldt-Universtät zu Berlin) *Operating Guidelines for Services*

13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*

14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*

15 Rinke Hoekstra (UVA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*

16 Fritz Reul (UvT) *New Architectures in Computer Chess*

17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*

18 Fabian Groffen (CWI) *Armada, An Evolving Database System*

19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*

20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*

21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*

22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*

23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*

24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*

25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*

26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*

27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*

28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*

29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*

30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*

31 Sofiya Katrenko (UVA) *A Closer Look at Learning Relations from Text*

32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*

33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*

34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*

35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*

36 Marco Kalz (OU) *Placement Support for Learners in Learning Networks*

37 Hendrik Drachsler (OU) *Navigation Support for Learners in Informal Learning Networks*

38 Riina Vuorikari (OU) *Tags and Self-Organisation: A Metadata Ecology for Learning Resources in a Multilingual Context*

39 Christian Stahl (TU/e, Humboldt-Universtät zu Berlin) *Service Substitution – A Behavioral Approach Based on Petri Nets*

40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*

41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*

42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*

43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*

44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*

45 Jilles Vreeken (UU) *Making Pattern Mining Useful*

46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

TiCC Ph.D. series