Master Thesis
Software Engineering
Thesis no: MSE-2003-02
January 2003

AgentChess – An Agent Chess Approach

- Can Agents Play Chess?

Henric Fransson

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Henric Fransson
E-mail: agentchess@hotmail.com

University advisor(s):
Stefan Johansson
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

The game of chess has many times been discussed and used for test purpose by science departments of Artificial Intelligence (AI). Although the technique of agent and as well multi-agent systems is quite old, the use of these offspring of AI within chess is limited. This report describes the project performed applying the use of agents to a chess program. To measure the performance of the logic has tests between the developed program main parts been performed. Further tests against a traditional chess program as well to position test suites have been done. The results and the impact of the different logic parts is presented and discussed. The aim of the project is to take the use of agents in chess a step forward.

[Chess, Agent, AgentChess, AI]

CONTENTS

1 INTRODUCTION

The aim of this master thesis is to try to develop and analyze a computer chess program using a multi-agent system approach. The effort made has focused on making a chess program which decisions are based on logical rules and communication between logical units, instead of the tree search approach used by traditional programs. We want to know if it is possible to use such an agent-based system as foundation for a chess program. We also want to know what the performance of such a program would be. The concept of multi-agent systems is useful in many different areas of every day life. Agents can be used to collect certain data of interest to its user from large information areas such as the Internet [3]. Multi-agent systems can also be applied within traffic control for transportation system environments such as airports. Only our imagination set the limits of where and how agents can be of use. Even though a project like this focus on agents and game playing, it might still be able to have future impact in areas we consider parts of our every day life.

The developed multi-agent chess program has been divided in different levels of logic, each one based on the previous level. This has been made for analyzing what impact the different logic of the agents has on the system's game play. The different level versions are to play against each other to visualize the difference in performance. The system is also to meet a traditional chess program represented by Crafty [4]. These tests are executed to obtain data of the difference between the two techniques. The multi-agent chess system developed has been given the name AgentChess.

1.1 Scope and Limitations

The game evaluation methods used by the chess program developed, will only take piece material value in consideration for decision-making. The material of a piece is the value of how much its worth compared to the other types of pieces existing in chess. Although a human player also considers such things as good positions of game pieces when playing chess, this is disregarded due to its complexity and requirement of deep knowledge in chess. Also considering this during development was not possible as it would have exceeded the size of the project far too much, and foremost, the knowledge needed is not in possession of the author of this report.

The chess program developed, using agent technology, is not meant to be able to beat or match today's traditional tree-searching chess programs. Although tests against such a chess program have been made, the intent of these have been to compare how far the multi-agent chess approach is able to reach and in which situations it fails to match the traditional approach.

1.2 Artificial Intelligence (AI) and Agents

The work of, and the term AI, Artificial Intelligence, was introduced about 50 years ago. The scientists in the field of AI focus on creating an entity that can be seen as something that behaves intelligent. Several definitions of AI exist as intelligence can be interpreted in many different ways. Literature define artificial intelligence as a process that *think*, *reason* or is behaving *rational* [2]. Other definitions that can be more demanding to the subjects intelligence is *human performance* or perhaps more demanding; *ideal intelligence* [2]. I consider AI to be a computer process that is behaving in what can be considered a correct manner regarding the specific problem. This computer process is capable to make decisions at equal or better capacity than a human being.

An agent can be seen as an intelligent object that interacts with the world surrounding it. It collects data it considers relevant and modifies the outside world to achieve its goals. An agent is an entity of artificial intelligence as it is to behave rationally processing its tasks. The tasks of an agent could be such as problem solving, planning and decision-making. An agent could also be intended to learn from experience of problem solving [3]. When taking input and performing output an agent could communicate with other agents as well as humans. When agents are working together as a group or against each other, striving for a common goal, the collection of agents is called a multi-agent system. The use of agents can range from many different areas as for example information collectors, market actors and control units of traffic environments [3].

This master thesis could be seen to be within the subject of artificial intelligence as its goal is to develop a chess program behaving rational. The program is also a multi-agent system. Each chess piece played by the program is a logic entity that co-operates with others to achieve a common goal. The "brain" of each entity is built up with rules that make status estimations of objects in the real world. These interpretations are parsed to make up plans and fulfill its task; play chess.

1.3 Previous Work in the Area

The use and experiment performed of multi-agent systems playing chess is very limited to the knowledge of the author of the report. The only occurrence found is the work of A. Drogoul [1]. Drogoul's developed chess program, MARCH (Multi-Agent Reactive CHess), base its move choices on simple square calculations of defense, attack and piece material. Even though the methodology of AgentChess is based on calculations of the same area, the performance of AgentChess will probably be of higher rate due to its complexity and move consideration. However, no test has been made between MARCH and AgentChess. The test data of MARCH performed by Drogoul is very limited. A hope of this project is also to present much more detailed data of the test games performed and the capabilities and limitations of AgentChess.

1.4 The Game of Chess

The reader is assumed to possess fundamental knowledge of the game chess and its rules. There exists much literature that describes the game of chess in basic, novice and expert manner. Fred Reinfeld, a famous writer of chess literature has had over 100 books produced of the subject, for example *The Complete Chess Course* [13]. The basic rules of chess can also be found online, for example at U.S. Chess Online [9].

1.5 Computer Chess Programs

What could be considered the first chess program was developed by Alan Turing in the 1950's [10]. The program was not an electronic one instead consisting of rules written on paper. The first electronic chess programs were developed during 1960 and at the year of 1974 was the first Computer Chess Championship held. The first program to win the title was Kaissa, developed by scientist from the Institute for Theoretical and Experimental Physics in Moscow [10]. The computer chess program Cray Blitz claimed the title 1983, running on a powerful mainframe computer. Crafty [4], a descent of Cray Blitz, has also been used for testing purpose during this project. The perhaps most famous chess programs for most people is Deep Blue; the first chess program to beat a human Grand Master. This was accomplished in 1997 and the Grand Master at the time was Garry Kasparov. The following match of human vs. computers will take place in Jerusalem January 2003. This time is the match between the Israeli chess program Deep Junior and Garry Kasparov; still owning the title of Grand Master.

1.5.1 Traditional Chess Program Development

The most successful technique building a chess program to compete against human opponents is simply using brute force. By calculating all possible legal move actions, and for each of these calculate the opponent's moves and so forth, a tree of move options can be constructed. When the chess program is analyzing the opponent's moves it often use a technique called *Minimax* [10]. Using Minimax the program assumes that the opponent will make what the chess program itself considers being the opponent's best move. When searching the move tree in the fashion of Minimax, with each move by the opponent assumed to give as bad result for the program as possible, a collection of leaf nodes in the end of the move tree is obtained. The chess program compares these nodes to each other and chooses the move that leads to the leaf node estimated as most profitable [2, 11]. The methodology of Minimax does not remove any moves from being calculated but makes estimations of what the action of the opponent probably will be. This helps the program from overestimating certain moves leading to great profit only if the opponent would make grave mistakes. The purpose of Minimax is to sort out these moves to focus on the ones the opponent probably would perform.

Even though the technique of Minimax is successful, it is nevertheless requiring much computer processing and memory. When developing a computer chess program, much time and effort is used to optimize the move calculations and evaluations. For each move look-a-head of the tree, the amount of nodes to compute increases dramatically compared to the previous move. Many different kinds of techniques have been invented during the years of computer chess programs to speed up the process. The *Alpha-beta* algorithm is used to avoid calculating moves which results are not necessary to consider [10]. During move search, the algorithm uses two boundary values, alpha and beta. Alpha represents the best move found for the program and beta represents the best move for the opponent; in other words, the worst move for the program itself. When a move has been evaluated and the value of the best move available for the opponent, beta, have been calculated; other moves which contain at least one move even worse for the program have not to be further investigated. The program already know that the previous move at least lead to lesser loss anyway. This technique saves the amount of nodes to search in the move tree. The effort saved can then instead be used to search the other moves even deeper [2, 11]. The technique of *Transposition tables* reuses old evaluated game positions to avoid recalculating them if they occur again through a different move combination [10].

2 PROJECT DESCRIPTION

The AgentChess chess program has been developed in the programming language C++ using Microsoft Visual Studio .NET [6]. It is built up with over 9000 lines of code, including comments. It is using C++ Standard Library and some components from MFC (Microsoft Foundation Class Library) [7]. The last library is used to handle multitasking threads and certain string management. As the chess program is using MFC, it can only be run using Windows. However, the occurrence of MFC objects are only present within one class and in its interface to a second class, so it can, with little effort, be ported to other operating systems as well.

2.1 Technical Specifications

2.1.1 WinBoard Interface

AgentChess has no graphic user interface implemented but is instead built supporting the interface WinBoard 4.2.6 (named XBoard for UNIX) [8]. The WinBoard interface is freeware and has been developed by Dr. Tim Mann Ph.D. in Computer Science at Stanford University. The interface supports graphical visualization of the chess game board and includes mouse support for the user. The user can play against any chess program supporting the interface and set up two chess programs to meet each other in one or a number of matches.

The possibility of using the WinBoard interface saved the project considerable amount of valuable implementation time and included the opportunity for easy and automatic test runs in which AgentChess was to meet other chess programs.

2.1.2 Agent Logic Module Description

The design approach used when implementing the agents of AgentChess is structured as such that each piece, in possession of the program, is represented by an agent module. The task of the piece agent module, referred to as Agent, is to evaluate its position, status and possible actions on the board. The Agent's evaluation execution is trigged by a main agent called the MasterAgent. The MasterAgent is an agent module different from the piece agent module. Its purpose is to collect the data from the other agents and, based on the results, carry out its own evaluation to reach a final move decision.

2.1.3 Design Description

The basic design structure of the AgentChess program can be divided in five main units: WinBoard, WinBoard Interpreter, ChessBoard, MasterAgent and Agent. See *Figure 2.1*.

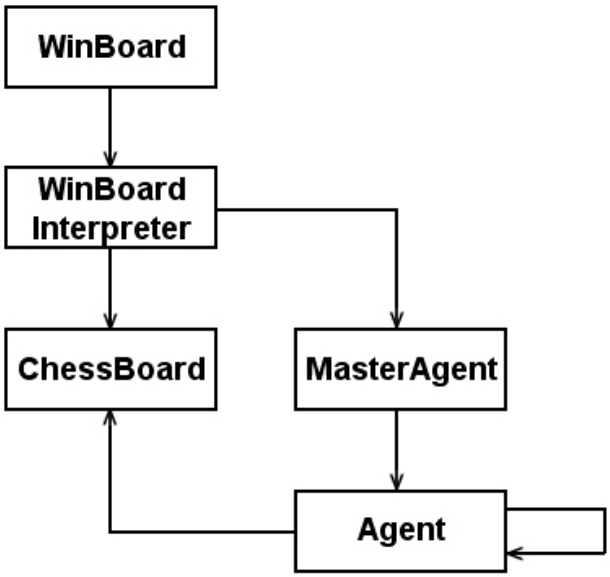

Figure 2.1: The design of the modules in the AgentChess program. The arrows display how the control of services is function. The module being pointed to is the supporting module.

The unit that starts AgentChess is the graphical interface WinBoard, which also sends commands to AgentChess's WinBoard Interpreter based on the user input and choices. This unit initializes and sends forth move commands to the ChessBoard unit that is the resource container of AgentChess's game board visualization and piece move calculations. When AgentChess it to move, the WinBoard Interpreter sends a move order to the MasterAgent unit. The MasterAgent is responsible for the Agent units, one agent unit for each piece. MasterAgent orders each Agent to evaluate their position, status and move options. The Agents are able to communicate with each other as they support and assist each other. When evaluating their positions and making calculations of move options and threats from opponent pieces, the Agents are supported by ChessBoard. After evaluation, each agent returns a result to the MasterAgent that in turn is to estimate the best move choice to make. This final move is reported back to the WinBoard Interpreter that sends forth the move to WinBoard and ChessBoard. ChessBoard and WinBoard are then performing the move. WinBoard also passes the move forth to the opponent. If the opponent then is given the opportunity to move, this move is sent by WinBoard to the WinBoard Interface. It commands ChessBoard to perform the move in support for the Agent modules. This procedure is repeated during game play.

2.1.4 Implementation Techniques - BitBoards

The chess logic of AgentChess is developed using a technique called bitboards or bitmaps [4, 5]. The technique was first used in the development of the chess program Cray Blitz [5]. Using this method can optimize the performance of the piece evaluation and move generation of the chess game. The description of the technique is not included in this report as this implementation technique is not affecting the logic of the agents or the results presented.

2.2 Agent Logic and Modules

Each chess piece of the color played by AgentChess is represented by a piece agent. The piece agent is in this report most of the time simply referred to as agent. In its most basic form, each agent concerns only about the piece it is representing. It is not aware, or bothers, about the situation of other member pieces. It studies and evaluates its own status on the board and reports a conclusion and move suggestion to the MasterAgent. The MasterAgent is the main agent bringing together all piece agent results and evaluating which agent move suggestion to prioritize and to choose as the final move.

The development of the two agent modules, the Agent and the MasterAgent, was designed to produce seven different levels of agent logic. Each level, except the first, is an enhancement of the previous one. All features of the earlier level version is included and extended with new logic. The logic functionality added to the Agent module is to increase the piece agent's capability of estimating its status and abilities, as well as cooperating and supporting other agents. The extension of the MasterAgent's logic is to make more complex and correct comparisons of the move suggestions received by the agents.

The reason of dividing the agent logic of the Agent and MasterAgent in several different versions is for testing purpose. Each AgentChess level version is to meet its predecessor and a traditional computer chess program in several chess games. The result from each agent version is compared to the other's to visualize the effect of the intelligence added to each level.

2.2.1 Vocabulary Terms

Before describing the different kinds of agent logic levels a description of the terms used within the rest of the report is presented. Some of them are used by the chess society while the author of the report has invented others. The author terms have been formed for easy explaining the logic and reasoning of the agents.

Material Value

The *material value* of a piece specifies how much its worth compared to the other types of pieces in chess. The purpose of the material value is for comparing the different pieces to each other when estimating opportunities and making compromises during game play. An example of using it could be when deciding if an attack of a certain piece is, or is not, a better choice than loosing a piece of another type.

The *profit* of a move is the material gained by the action taken. The profit can also be negative. When a profit is exclusively stated to be negative, the term *loss* is used instead.

Moves

A *move* is just a simple legal move by a chess piece on the game board. The term is being used in this report to, exclusively, state a move that does not result in capture of an opponent piece. In other cases, this will be specified in the text.

An *attack* is a move by a piece that results in a capture of an opponent piece.

A *move-attack* is a two-step attack. It consists of a move planned being followed by an attack of an opponent piece. The agent performing the move is the same one planned to carry through the attack.

Threats

A *threat* is present if an opponent piece is able to attack an agent if it was to make the move next.

The *worst threat* is a threatened piece that, estimated by MasterAgent, is the piece the opponent probably wants to attack if it was to move next.

Requests

Requests of assistance can be made to the other pieces by the piece that is estimated to be the worst threatened. Two different kinds of requests can be performed.

An *attack request* is a request to the other pieces attacking any of the opponent pieces threatening the requesting agent. The agent performing an attack request is not doing this blindly but also evaluating the possible danger of doing so.

A *defense request* is a request to the other pieces to assist by defending the requesting agent. A defense request can be supported by an agent through moving itself in a position so it can attack the threatening opponent if it would choose to attack the requesting agent. The request could also be carried through by a move that blocks a threatening opponent to attack the requesting agent. As with attack requests, the assisting agent is also evaluating the danger it might put itself into when supporting the request.

2.2.2 Agent Logic Levels

The agent logic levels specified in this chapter makes different kinds of generalized assumptions of different kinds of game board situations. They are not always true but serve as a guide that the agent can rely on when it evaluates its position and status. The basic idea is to extend the agent logic to make detailed analysis preventing it to make mistakes during certain situations. Again, the new modified assumptions are neither always correct nor complete. However, for each level the agent will hopefully, get closer to evaluate its position in what could be considered a correct and proper way. The logic contained in each agent level is visualized in *Table 2.1*. Visualization with description of how the agents communicate with each other is presented in the next subsection *2.2.3 Agent Communication*.

	Agent Level							
Logic	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Logic Foundation	X	X	X	X	X	X	X	X
Threat and Backup Consideration		X	X	X	X	X	X	X
Move-attack Consideration			X	X	X	X	X	X
Threat Move Consideration				X	X	X	X	X
Attack Request					X	X	X	X
Defense Request						X	X	
Restricted Defense Request								X
Defense Consideration							X	X

Table 2.1: The logic contained within each agent level version.

Level 1 - Logic Foundation

Agent:

Deployed with only the logic foundation the agents are only concerned of themselves and merely nothing else. They check which opponent pieces are currently threatening them and which moves and attacks they are able to perform. The moves are compared to each other based on the most valuable opponent piece it might be able to attack next. The similar method is applied to attacks; comparisons are made between the possible attacks based on the most valuable opponent piece captured. The agents are not able to make any estimation of what could be lost afterwards, due to the move or attack, when it is the opponent's turn to move. The threat evaluation is simply based on the material value of the piece the agent is representing. No defense is considered at this logic level.

MasterAgent:

The MasterAgent is prioritizing attacks during the logic foundation. As no threat is estimated at the attack position attacking is always considered profitable. The best move evaluated is only performed if no attack exists. If an agent is threatened and no attack or move seems to be able to compensate for its loss, the threatened agent's best actions are considered. Also at this time, its attack is prioritized in front of its move. If the threatened agent is not able to make any move at all, the best attack or move, of the other agents is chosen as usual.

Level 2 – Threat and Backup Consideration

Agent:

The first level of extended logic for AgentChess adds an increased threat evaluation. In case a threatened agent got defense by a member piece, this is now also taken under consideration. When a defense exists, the agent adds the possibility of the capture of its attacker to the threat estimation. When estimating the threat status, the agent assumes that the opponent will use the cheapest piece possible to attack with, as it can be lost in the next move by the defending agent. This is definitely not always true but the assumption makes the task easy and simple to handle. An example when this assumption is not true, and the opponent will not use the cheapest piece possible to attack the agent, could be if that certain piece, due to its position, constitutes a valuable defense. It might perhaps be blocking another agent from capture its queen. Using a more expensive piece to attack with might then be a better choice, even though it could be lost the following move.

As well as extending the direct threat evaluation, this level also adds threat consideration for the moves and attacks. The possible threat at the square the agent can attack to is added to the estimated profit of the action.

MasterAgent:

The MasterAgent is extended to analyze the moves and attacks better, as it now got them estimated with threat consideration by the piece agents. Moves are now prioritized in front of attacks if the attacks leads to loss and the moves seem to be profitable. The same method is used when evaluating the worst threatened agent's move options.

Level 3 – Move-attack Consideration

Agent:

The purpose of the third extension of AgentChess is to analyze the move-attacks in a more detailed manner. In the previous level the attack following the move, has been the estimation of its profit. The threat on the square the agent moves to before the attack has not been considered. The extended logic makes the agents able to estimate any threat at the square first moving to. The possible attack profit is not considered if the square to move to before is dangerous.

Another aspect taken into consideration is if the material result of a move-attack is estimated to end up equally between AgentChess and its opponent. While such a move has previously been seen as valuable as a move without any attack possible, it is now instead prioritized in front of such. This is to give AgentChess a more offensive approach.

MasterAgent:

The prioritization of move-attacks by piece agents is as well implemented to the MasterAgent when it compares which agent has the best move option to offer. The best move evaluation of the MasterAgent is performed in the same way as for each individual agent evaluating its best move.

Level 4 – Threat Move Consideration

Agent:
The piece agent module is not modified in this level extension.

MasterAgent:
From the beginning of the first logic levels, only simple evasive maneuvers have been taken by the MasterAgent when protecting the piece estimated as the worst threatened piece. When no attack has been able to guarantee compensation for the loss of the threatened piece, MasterAgent has made the simple decision of either performing the threatened piece's best attack or best move. However, there lie problems within this solution. If the threatened piece's attack or move is not guaranteeing the safety of the threatened piece, loosing it and capture an opponent piece as a small compensation might be the best solution. The situation might also be of the opposite type. Even though an attack of higher value than the threatened piece exists through a different agent, making a lower profitable attack with the threatened agent might be better if it also saves the threatened piece. In other words, the small attack profit and the saving of the threatened piece could exceed the best attack available. Situations as these are taken under consideration in this third level of main agent logic.

Level 5 – Attack Request

Agent:
When extended with the logic of this level, the agents get the possibility to request attack support from its member agents. When each agent has evaluated its move options and threat status, the agent considered to be in the greatest danger is given the opportunity to ask the other agents if they are able to capture any of its attackers. The other agents respond to this request and evaluate what possible outcome they can achieve when attacking any of the specific opponents. This is performed with the loss of the threatened agent in consideration. The result of each agent is reported back to the MasterAgent.

MasterAgent:
The logic of the MasterAgent is not changed much only for supporting the attack requests. It compares the best attack request with the attack and move possibility of the threatened agent. If it finds a request to be a better solution than the threatened agent's actions, the replacement of these with the attack request is performed.

Level 6 – Defense Request

Agent:

The level of defense request is working in a similar way as the previous level of attack request. When the attack-requesting agent was asking its member agents to attack the opponents threatening it, the defense-requesting agent is instead asking them to defend it. The agents responding to the request will try to support the threatened agent either by position themselves to attack the opponent after it has fulfilled its attack or by blocking the opponent from attacking. When evaluating the last defense variant, the supporting agents are also checking that they got defense support by another agent so they are not sacrificing themselves for nothing.

MasterAgent:

The MasterAgent's logic is not extended in this level either. Only similar support for defense requests as with attack requests is added.

Level 7 – Defense Consideration

Agent:

During all levels of agent logic, we have not been taken any consideration of what the result could be of removing a piece from its position to perform a move or an attack. However, the agent moving could also open up a path for an opponent to attack a second agent positioned behind the moving one. Not taking care of this problem has become even more critical when the defense request was added in the previous level. When a piece has been supporting a defense-requesting agent, either by position itself to attack the threatening opponent after the attack or by blocking it, this solution is destroyed if the supporting agent is chosen to move to another square next. The threatening opponent will then be free to perform its attack if it is still in its old position.

This problem is taken care of during this last level of AgentChess. Before the move evaluation begins, the worst threat is to be calculated. After this, for each move or attack to evaluate, every agent checks with the other ones that it is not causing a worse situation by performing the move. If so, the move or attack evaluated is to be disregarded. This solution prevents any defense supporting agent to move unless they are able to perform an attack that grants higher profit than can be lost by moving, or unless the old threat of the defended agent has moved or been captured.

MasterAgent:

No extra logic for MasterAgent is added in this last level either. The defense consideration is only added in the piece agent logic.

2.2.3 Agent Communication

When AgentChess is to perform a move, the MasterAgent orders all agents on the game board to evaluate their position and possible actions. The agents report their best considered move and attack together with data of their current threat status, if any. The MasterAgent is then evaluating the data collected from the agents and makes the decision of which agent's move suggestion to perform. See *Figure 2.2a*. All AgentChess of level 1 to 4 work in this manner, in a more or less advanced way.

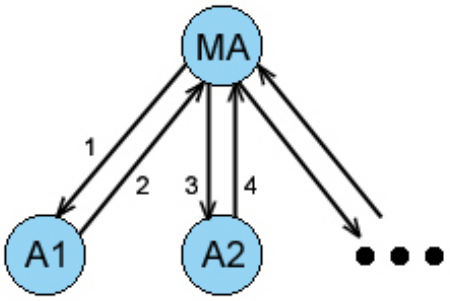

Figure 2.2a: The visualization of the communication order between MasterAgent (MA) and each agent (the dots represents the other 14 agents).

At the level 5 and 6, the features of attack request and defense request is added to respectively level. This feature is enabled after the agent communication exchange of *Figure 2.2a*. After this first communication procedure, the MasterAgent has decided which agent probably is the worst threatened. MasterAgent then gives the threatened agent the opportunity to ask for support from the other agents. The threatened agent requests help from the others and reports the best solution to MasterAgent for processing. See *Figure 2.2b*.

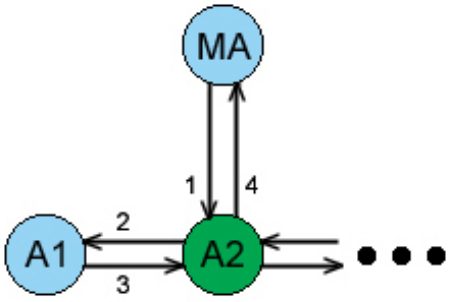

Figure 2.2b: The visualization of the communication order between the MasterAgent (MA), the worst threatened agent A2, and the other agents during request of assistance.

In AgentChess level 7, the introduction of defense consideration is made. The defense consideration demands that the worst threat first to be calculated and reported to MasterAgent. This is done as in *Figure 2.2a* as usual. However, no move calculation is made at that moment. The move calculation is made afterwards as shown in *Figure 2.2c*. The MasterAgent asks each agent for move suggestions and supports them with the estimation of the worst threat. For each move the agents evaluate, they check with the others that they are not causing them a threat worse than the first worst threat. If so, the move is disregarded. The result is reported back to the MasterAgent. *Figure 2.2c* display the move calculation by agent A1. It first gets the order to perform the calculation by the MasterAgent. During the calculations agent A1 checks the threat status of the other agents. Afterwards the agent reports its suggestions back to the MasterAgent. The same procedure is performed for all other agents as well. After this, the request logic is performed as in *Figure 2.2b.*However, the use of defense consideration is also performed here when evaluating the request moves, as in *Figure 2.2c.*

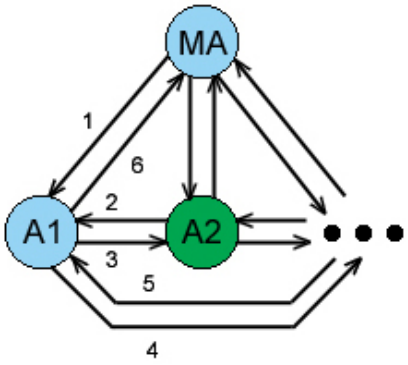

Figure 2.2c: The visualization of the communication order between the MasterAgent (MA) and the agents during move evaluation with defense consideration.

2.2.4 Agent Evaluation

As base of its estimations each agent is evaluating the material value of itself, the opponents it can attack and the opponents it is threatened by. This information is used by the agents for decision making and gives them the ability to make prioritizations and sacrifices during game play. The material value of the pieces in a chess program can vary and different weights are used by different programs. Examples of weights and ranges can be found in [14]. *Table 2.2* visualizes the material values used by the agents of AgentChess. The material value of the king is set to infinite as it is the most important piece of the game. The king of the color played by the program is never allowed to be captured and the capture of the opponent's king is the final goal. This material value of the king makes it the highest prioritized piece on the chess board.

Piece	Material Value
Pawn	100
Knight	300
Bishop	350
Rock	500
Queen	900
King	infinite

Table 2.2: The piece material values used by AgentChess for evaluation and decision making.

3 EXPERIMENTS

The main purpose of the experiments is to see if the logical modules added to the Agent and MasterAgent units give AgentChess increased game play capability. The collected data is to show the result of the agent's strategies during different periods of the game. We will compare the different versions of AgentChess to each other, to visualize the actual improvements of the new added functionality. We want to know if it is improving, worsen or maybe even not making any changes to game play. The data of the different versions of AgentChess will be analyzed to see where and how often the new functionality occurs during the game. The purpose is to visualize the effects of the difference in handling certain game situations.
Except from meeting its predecessor versions, AgentChess is also to meet a chess program of traditional implementation; represented by Crafty 19.00 [4]. The purpose also comparing AgentChess to a traditional chess program is to see if a similar effect of game play also applies to an opponent of this kind.

3.1 Test Data to Collect

The tests performed have been defined only to be run when a certain type of opponent is met. Some are only used when AgentChess meets a predecessor and others when it meets Crafty. The reason for this is that certain tests, as the move-attack test, consider a situation that is handled in the same way by all different versions of AgentChess. Because of this, these *external tests* are not considered when AgentChess plays against itself, only against Crafty. Tests of the opposite character also exist, *internal tests* that are only executed when AgentChess is facing a predecessor. The tests against the two different opponents will also produce *common* type of test data.

3.1.1 Crafty

The opponent of a traditional chess program used to test AgentChess is the chess program Crafty 19.00. Crafty has and is still being developed by Robert Hyatt, professor at the University of Alabama at Birmingham [4]. Crafty is a descent from the chess program Cray Blitz that won the title of World Computer Champion between 1983 and 1989. Crafty has been rated to play at the maximum level of 2785 at the ICC (Internet Chess Club) [9][†]. Crafty is using the technique of rotated bitboards and is searching 800,000 nodes per second using a Quad Xeon Processor [4]. However, during the tests will Crafty to be limited to a search depth of 1 level. It has not been given any move books to use for opening or end games and its learning capabilities have also been disabled. Crafty's pondering function, to think while the opponent thinks, has also been disabled. However, as it is only limited to the search depth of 1, this option should make no difference. The opening and end books disabled should not be mistaken for opening and end game evaluation. This is something still performed by Crafty. The books disabled are databases of game board positions containing information of how to handle these specific situations. The limitations taken are necessary because of Crafty's significantly ability to outperform a simple chess program like AgentChess. Only when using these restraints can we hope to get detailed data of the performance of AgentChess when meeting Crafty. It should be noted that when using these restraints the game play of Crafty is not as powerful as its normal ICC rating states. The restraint to search only one move ahead should be considered fair, as this is about the search depth for AgentChess as well. The exact description of the search depth of AgentChess is explained in section *4.1 AgentChess vs. Traditional Approach*.

[†] The ICC rating specifies at what level a chess player or program is able to play at. As an example had the current Grand Master Garry Kasparov an ICC rating of 2820 in the year of 1998 [9].

3.1.2 Common Tests

The tests described in this chapter will be performed when AgentChess both meets a predecessor as well as Crafty.

Outcome and Material Data

When AgentChess meets previous, simpler, versions of itself and Crafty, the outcome, as well as the material balance during the game, is to be collected. The material difference between two playing chess programs is gathered to get detailed information of the matches, instead of just a win or lose result. The purpose is to get an estimation of how far apart, as chess players, the two opponents are to each other during specific game periods.

Attack Requests

When testing the effects of attack requests the versions of AgentChess, supporting it, will store information of how frequently and where, during game play, the functionality is used. This data together with the material data also collected is to show how much is gained by the module.

Defense Requests

As with testing the effects of attack request, the tests of defense request are also to collect data of how frequent and where, during game play, the logic is being used. How much is gained by the extension is evaluated together with the material difference between the two playing programs

3.1.3 Internal Tests

The following test is only performed when AgentChess is meeting another version of itself.

End Game

As AgentChess contains no specific logic of handling end game situations differently, it is performing quite badly during this game phase. The result, when playing against another version of AgentChess with the same limitations, is that many games end in draw due to the “50 move rule”. This rule states that if 50 moves have been made without any piece being captured or any pawn moved, any player can call a draw. Although we get a statistical performance change of the games that ends in a win and loss; we are also interested in this clouded area of draw results. To evaluate which side has reached most far in the games that ends because of the “50 move rule” we will save data of which side has most pieces left in the end of these matches.

During development, analysis of the end game has show that many “50 move rule” draws ends up with one side only having a king left. The other side, promoting attacks of this piece over anything else, looses most of its pieces while trying to attack the opponent’s king until the rule is executed. The side having most pieces until this phase probably had the advantage until then. This is what we want to know. The end game data we will collect is the side first ending up with only its king left. We will store statistical data of these results as a complement to the other matches ending up in win and loss.

3.1.4 External Tests

The tests described in this chapter are only performed in the matches between AgentChess and Crafty.

Move-attacks

To test how well AgentChess is performing move-attacks; we will collect data of how often the planned following attack actually is carried through. This will give us an estimation of how successful the move-attack plans are.

Threats

To evaluate how well AgentChess is handling threats from the opponent, each piece's estimated threat status is to be tested and compared to the actions taken by the opponent. The piece's threat estimation is used by AgentChess to evaluate which piece is in danger and to focus on to save from being captured.
To test how sufficient AgentChess's threat evaluation is during a game we will test how often pieces, estimated not to be threatened, actually are captured. As AgentChess makes its threat calculation based on opponents threatening and member pieces defending, we will get statistical data of how often this consideration fails to be sufficient. That is, from Crafty's point of view. Due to Crafty's well-established performance, we assume that if it chooses to make a certain attack, this is probably the best choice. It should however be noted as Crafty has been restrained and limited of its features, its not as capable to make as much correct moves as it otherwise would be.

3.1.5 Test Suites

During the years of computer chess development, there has been a need for developers to test their program trough certain game situations to see how it performs and to localize possible errors [12]. Many kinds of test suites of chess positions have been developed for as well chess players and programs. Some of them are chess positions that have been analyzed by chess players calculating the best move for a player in a certain situation. Some of them are positions taken from game matches of, for example, Grand Masters where the Grand Master's move is considered being the correct one [12]. Nevertheless, even Grand Masters can make mistakes so the tests suites is not necessarily fault proof. Another uncertainty that exist is that a chessboard position not necessary has only one, but several good moves. A third problem of tests suites is that, due to the complexity of chess, some can contain errors. However, test suites are a good methodology of testing a chess program and it is used by many developers. Some of them have even developed specific tests suites for their specific software [11].

The tests suites chosen contain a mixture of chess game problems. From middle game, positional and end game. Tests suites can be used together with a timer where the chess program is to give several answers as it goes deeper in its search tree until it finds the correct move. The score given the chess program is then depending of the time it took to reach the correct conclusion. As AgentChess is not an advanced chess program and is not using search tree like a traditional chess program; the time of it reaching its conclusion is not taken under consideration. The results of the different versions of AgentChess are compared both to each other and to Crafty. Crafty will be run under several different constraints to visualize their effects.

The following tests suits are to be run [11].

BWCT – end game test suite
F. Reinfeld - 1001 Brilliant Ways to Checkmate (1955)
ECE3 – end game test suite
A. Adorjan et al. - Encyclopedia of Chess Endings vol. 3 (1986)
ECM – middle game test suite
M. Rosenboom - Encyclopaedia of Chess Middlegames
TYPP – middle game positional test suite
R. Bellin, P. Ponzetto, Macmillan - Test Your Positional Play (1985)
VA – mixture test suite
V. Albillo - Test Positions
WAC – mixture test suite
F. Reinfeld - Win at Chess (1958)

3.2 Test Results

3.2.1 Outcome and Material Test Data

The focus of this test data will be at the matches of the higher levels of AgentChess. This decision has been taken under consideration of limiting the detailed data information and size of the report. The data of these logic levels is of most interest. It was also here unexpected results was found. We make a summarized description of all match results in *Table 3.1*.

Game Results of All AgentChess Matches									
	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Crafty
Level 1	X	0,8,92	1,68,31	2,63,35	1,67,32	3,74,23	2,72,26	X	0,98,2
Level 2	-	X	3,68,29	1,62,37	1,79,20	3,61,36	0,62,38	X	0,100,0
Level 3	-	-	X	25,36,39	25,35,40	19,34,47	11,48,41	X	1,97,2
Level 4	-	-	-	X	20,27,53	18,29,53	15,35,50	X	0,97,3
Level 5	-	-	-	-	X	23,30,47	26,22,52	19,35,46	1,91,8
Level 6	-	-	-	-	-	X	21,24,55	18,25,57	3,90,7
Level 7	-	-	-	-	-	-	X	18,15,67	6,88,6
Level 8	-	-	-	-	-	-	-	X	2,90,8
Crafty	-	-	-	-	-	-	-	-	X

Table 3.1: The game result of the matches is displayed as "program 1 wins, program 2 wins, amount draw". Program 1 is the one in the left column and program 2 is the one at the top row.

AgentChess level 7 – Defense Consideration Test Data

AgentChess level 7 – defense consideration							
	Game Results			Draw Result			
	Wins	Losts	Draws	Draw Wins	Draw Losts	Draw Wins	Draw Losts
Level 1	72	2	26	4	13	15%	50%
Level 2	62	0	38	12	15	32%	39%
Level 3	48	11	41	10	17	24%	41%
Level 4	35	15	50	18	16	36%	32%
Level 5	22	26	52	13	18	25%	35%
Level 6	24	21	55	23	18	42%	33%
Crafty	6	88	6	0	0	0%	0%

Table 3.2: The game results of the matches of AgentChess level 7. The draw wins are the matches ended in draw and the opponent first ending up with only its king left.

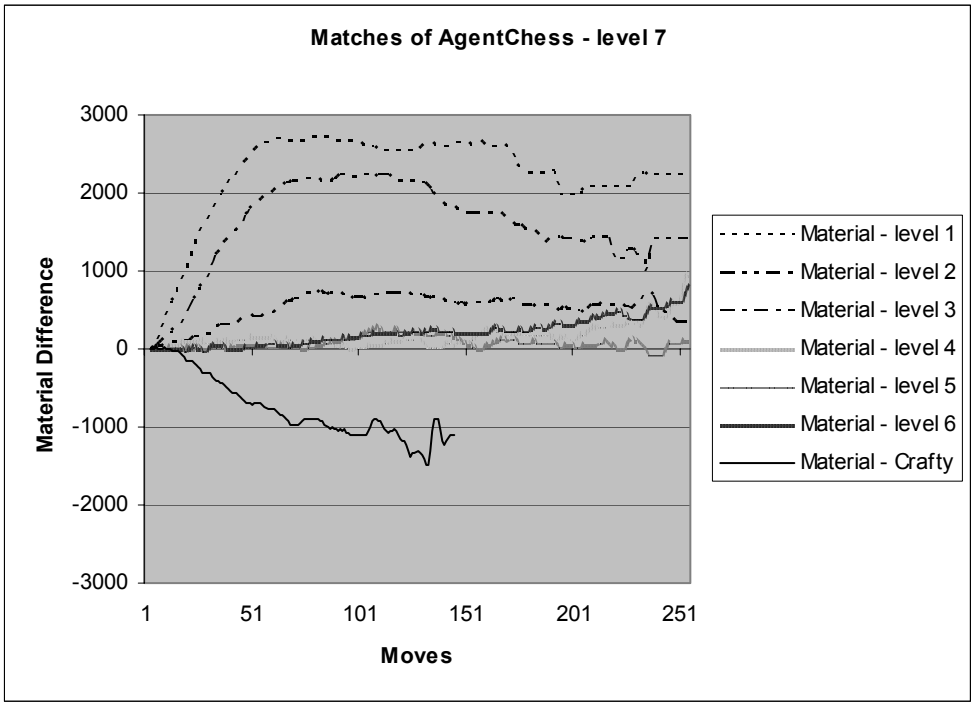

Figure 3.1: Visualization of mean value of the material difference of every game of 100 matches between AgentChess level 7 and its opponents during game play.

We will now analyze the material difference between AgentChess level 7 (defense consideration), and its opponents. *Figure 3.1* visualize all matches of level 7. The difference of game play superiority is clear between the lower levels of 1, 2 and 3. However, what can be noticed when analyzing the diagram of *Figure 3.1* in detail is that level 6 performs worse, as an opponent, compared to level 4 and 5. In fact, in the end of the matches consisting of many moves, level 5 is taking the material advantage against level 7.

As we suspected of the previous version, AgentChess level 6 (defense request), the strategy of backing up pieces without bothering when leaving these defense positions afterwards, is a bad idea. The material difference data of the matches of AgentChess level 7 indicate that it might cause more damage using this strategy without defense consideration, compared to not using it at all. The amount of won and lost games against level 5 and level 6 in *Table 3.2*, states however that there is not much difference between them. The results are unfortunately not detailed enough to give us any strong indications if it benefits using the technique or not.

Based on the results between level 5 and level 7, the decision of making a specialized version of AgentChess level 7 was taken. This version is only to apply defense requests in extreme conditions. The use of defense requests is restricted only to apply when the threatened agent's own move solutions are all estimated to result in material loss. The purpose of this new version is to find the benefits of using defense request respectively defense consideration. We want to test it against level 5 (attack request), level 6 (defense request) and to its original version, level 7. We call this new version AgentChess level 8.

AgentChess level 8 – restricted defense request							
	Game Results			Draw Results			
	Wins	Losts	Draws	Draw Wins	Draw Losts	Draw Wins	Draw Losts
Level 5	35	19	46	19	12	41%	26%
Level 6	25	18	57	26	19	46%	33%
Level 7	15	18	67	20	23	30%	37%
Crafty	2	90	8	0	1	0%	13%

Table 3.3: The game results of all matches of the restricted version of level 7; AgentChess level 8. The draw wins are the matches ended in draw and the opponent first ending up with only its king left.

The new test results of *Table 3.3* show an increase of won matches for level 8 against level 5 (attack requests). The difference is quite dramatic, stating that the use of defense request should maybe be decreased for best performance. However, level 8 uses defense consideration not only for defense requests but also for every other move taken during the game. One might suspect that not using defense request at all and combine only attack requests and defense consideration could be the best solution. However, the game data results of level 8 and level 6 (defense request) shows that level 6 is able to decrease the amount of lost matches compared to level 5. In *Table 3.4,* we see that level 6 is the superior version compared to level 5 even without defense consideration. Nevertheless, as previously stated, is the difference between level 6 and 5 only marginal when meeting level 7, see *Table 3.2*. To make a conclusion of which is the best performing version of AgentChess further test data is desirable. We will use the data of the Crafty matches in the next chapter for this purpose.

AgentChess level 6 – defense request							
	Game results			Draw Results			
	Wins	Losts	Draws	Draw Wins	Draw Losts	Draw Wins	Draw Losts
Level 1	74	3	23	6	10	26%	43%
Level 2	61	3	36	15	7	42%	19%
Level 3	34	19	47	9	12	19%	26%
Level 4	29	18	53	16	15	30%	28%
Level 5	30	23	47	19	15	40%	32%
Level 7	21	24	55	18	23	33%	42%
Crafty	3	90	7	0	2	0%	29%

Table 3.4: The game results of all AgentChess level 6 matches. The draw wins are the matches ended in draw and the opponent first ending up with only its king left.

Crafty Test Data

Crafty					
	Game Results			Draw Results	
	Wins	Losts	Draws	Draw Wins	Draw Losts
Level 1	0	98	2	0	0
Level 2	0	100	0	0	0
Level 3	1	97	2	0	0
Level 4	0	97	3	0	0
Level 5	1	91	8	0	0
Level 6	3	90	7	0	2
Level 7	6	88	6	0	0

Table 3.5: The game results of Crafty matches against AgentChess level 1 to 7.

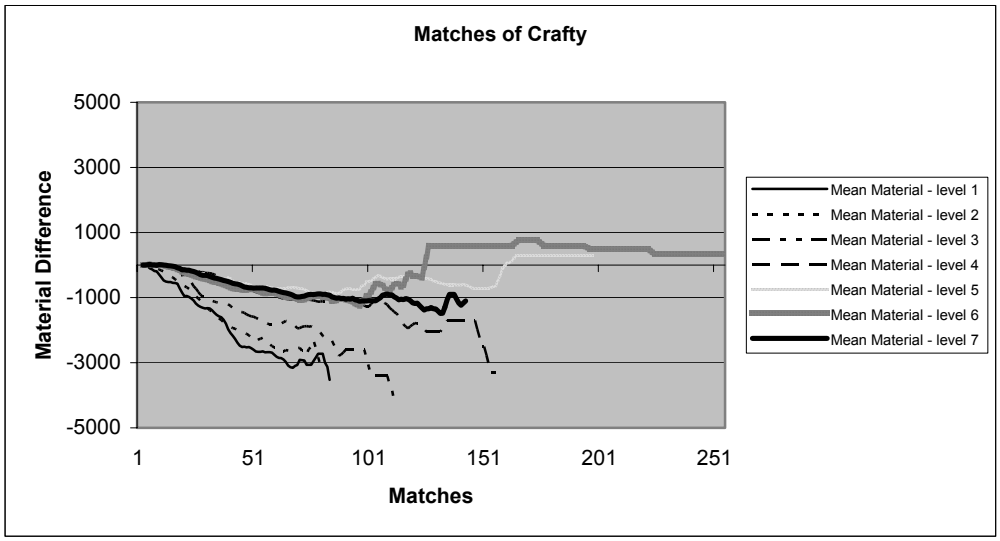

Figure 3.2: Visualization of mean value of the material difference of every game of 100 matches between Crafty and AgentChess level 1 to 7 during game play.

It is clear from the game results in *Table 3.5* that Crafty is a superior opponent to all versions of AgentChess. An interesting aspect is that although AgentChess level 7 (defense consideration) produces far better game results than version 5 and 6, the material balance between them and Crafty is about the same. The material difference between two playing chess programs is not an optimal measurement. However, this was expected result as the positional data of a chess game is another vital aspect. Nevertheless, we can see a noticeable increase of material after about 100 moves in the games of level 6. The increase of material difference occurs when about 90 percent of the games have already ended. The majority of the small amount of games left is also the ones leading to draw. The presence of the increased material is disregarded as it is caused by a small portion of the test results.

Outcome and Material Data - Discussion

We are now able to answer the contradicting results from the previous chapter regarding which AgentChess version is the most promising solution. The test results of the matches against Crafty shows us that AgentChess level 7, with full defense request and defense consideration, is the version with best performance. However, the logics of the most extended agents have issues. Using attack requests and defense requests are necessary for success but the logic balancing the features is not optimal in either version. The difference in opponent game play calls for different behavior. We showed in the previous chapter that the special version level 8, with restricted defense request, performed better against level 5 (attack request). When it performed about the same against level 6 it also proved to even play worse against its predecessor; level 7. This last result is also confirmed when level 8 was to meet Crafty. The data of these matches shows the performance of level 8 to be about the same as level 5 (attack request) and level 6 (defense request), see Table 3.1.

The draw results of the internal matches that ended because of the "50-move"-rule did not give us extra information of the games played, when we needed it. When we needed extra data as complement, comparing the highest levels of 5, 6 and 7, was the draw results giving about the same information as the actual game results anyway. See the data of *Table 3.2* and *Table 3.3*.

3.2.2 Attack Request Test Data

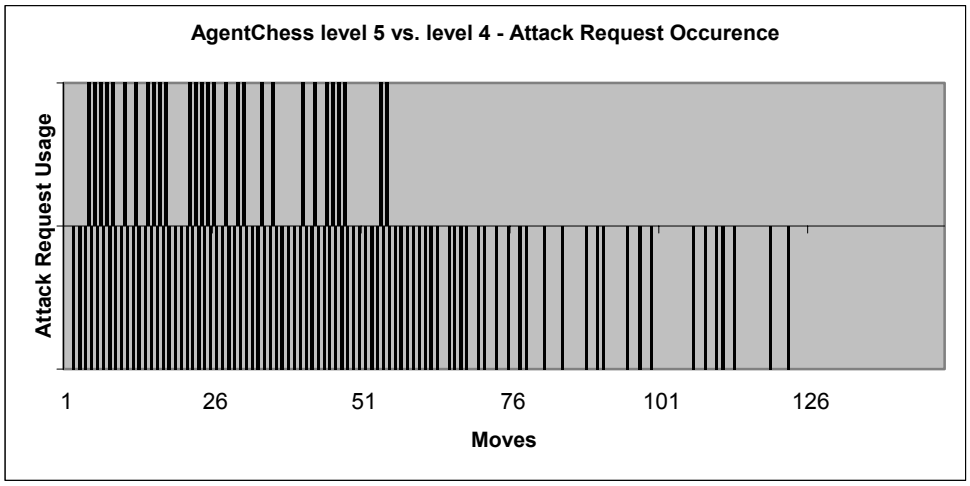

Figure 3.3: The occurrence of attack requests performed when AgentChess level 5 meets AgentChess level 4 in 100 matches. The lines at the top displays the occurrence of attack requests performed while the lines at the bottom display all possible situations attack requests could have been used.

Figure 3.3 visualizes at which move during game play that attack request is current for the matches between AgentChess level 5 (attack request) and level 4 (threat move consideration). The lines at top represent the occurrence of an attack request being performed. The lines at the bottom represents each time the worst threatened agent instead decide to move to safety by itself. What we can conclude from the data is that the feature is more useful during the beginning of the game. That is, when it exist many pieces on the board and the possibility for a member agent being able to attack a threatening opponent of the worst threatened agent, is more likely. We can see that the feature is not used much compared to all possible time the feature is available. The resulting material difference of the matches played, *Figure 3.4*, also states only limited material difference to AgentChess level 5 (attack request) advantage. However, an expected noticeable performance benefit is visualized in the game results between the two different versions, *Table 3.6*.

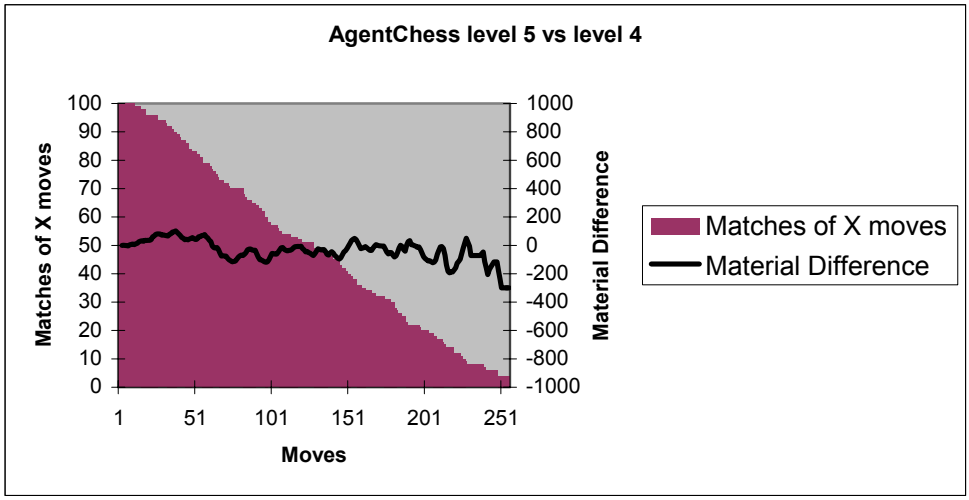

Figure 3.4: Visualization of mean value of the material difference of the 100 matches between AgentChess level 5 (attack request) and level 4 (threat move consideration).

AgentChess level 5 – attack request							
	Game Results			Draw Results			
	Wins	Losts	Draws	Draw Wins	Draw Losts	Draw Wins	Draw Losts
Level 4	27	20	53	13	18	25%	34%

Table 3.6: The game result between AgentChess level 5 and level 4.

3.2.3 Defense Request Test Data

AgentChess level 6 – defense consideration							
	Game Results			Draw Results			
	Wins	Losts	Draws	Draw Wins	Draw Losts	Draw Wins	Draw Losts
Level 1	74	3	23	6	10	26%	43%
Level 2	61	3	36	15	7	42%	19%
Level 3	34	19	47	9	12	19%	26%
Level 4	29	18	53	16	15	30%	28%
Level 5	30	23	47	19	15	40%	32%

Table 3.7: The game results of all matches of the AgentChess level 6 and its predecessors.

The introduction of defense request in AgentChess level 6 produced positive results when it faced its predecessors, see *Table 3.7*. The use of the defense request feature did not fulfill all our expectations when it was combined with defense consideration, as we previously have seen. Even though it is vital the two techniques cooperate for best result against Crafty, they perform worse against AgentChess level 5 (attack request). This problem was solved by restraining the use of defense requests. However, the effect of this modification turned out opposite against Crafty. We will now analyze the amount of defense requests used during the matches of the involved AgentChess versions and Crafty.

Defense Request Matches			
	Level 6	**Vs.**	**Level 5**
Defense Requests	123		X
	Level 7	**Vs.**	**Level 5**
Defense Requests	227		
	Level 8	**Vs.**	**Level 5**
Defense Requests	121		X
	Level 7	**Vs.**	**Crafty**
Defense Requests	163		X
	Level 8	**Vs.**	**Crafty**
Defense Requests	42		X

Table 3.8: Matches of AgentChess levels supporting defense request.

Table 3.8 visualizes the amount of defense requests used by the different AgentChess versions involved. From the first two matches we see that the amount of defense request against AgentChess level 5 is almost doubled when using defense consideration. The effect of this was not positive, as we have already stated. When AgentChess level 8 was to meet level 5, the amount of defense request used dropped back to its previous amount. The usage of defense request back to its previous frequency and with the advantage of defense consideration, the game result of the level 8 went higher than both level 6 and the original version of level 7 (see *Table 3.3* and *Table 3.7*).

When level 8 was to meet Crafty, we see that its amount of defense consideration has significantly dropped. From *Table 3.3* we also know that its game result did the same. The previous version, level 7, is keeping the amount of defense requests to the same as the modified version did against level 5. As it also performed best against Crafty compared to all other AgentChess versions. This frequency of using defense request seems to be best.

As we have concluded earlier, the balance logic of the defense request is not optimal. As now, facing different opponent require it to be modified for best performance. A different or more advanced logic for considering defense request is needed.

3.2.4 Move-attack Test Data

Move-attack Occurrence Against Crafty			
Vs.	**Planned**	**Performed**	**Performed**
Level 1	1246	148	12%
Level 2	520	29	6%
Level 3	1353	140	10%
Level 4	1451	121	8%
Level 5	1833	143	8%
Level 6	1747	183	10%
Level 7	1852	189	10%
Level 8	1914	192	10%

Table 3.9: This data visualize the occurrence of move-attacks for each AgentChess version playing against Crafty.

The data of this test lacks the possibility to be compared to any other chess program. We do not know what a good amount of performed move-attacks is. This data is not available from other chess programs either, as their logic is based on tree search. In such case is a performed move with a good opportunity of attacking not necessarily meant to carry out the possible capture. The purpose of the move could be another kind, visualized first at a deep level of the move tree searched. However, the data is presented for the purpose of possible future development or other chess programs of the same area.

The lack of difference of move-attacks between the AgentChess version is no surprise as the move-attack plans is performed in the same manner. An exception although exist, the second level of AgentChess. The introduction of better threat consideration in level 2 decreased the amount of move-attacks performed by half. At the following level, version 3, the introduction of move-attack prioritization compensated for this drop. The exact reason for this is unclear. The data collected during the tests are not able to explain the reason of the decrease event. However, based on the knowledge of the behavior of level 2 the reason for the move-attack decrease might be that as several more attacks are disregarded, because of their introduced threat estimations, more moves are instead chosen. Opposite to the attack squares, the move squares are not threat estimated yet; many move-attacking pieces might be captured before attacking. For AgentChess level 1, it might be the case that, even though many move-attacking pieces also are captured, a higher amount is performed as the square attacking to, is not threat estimated. These moves are instead disregarded by level 2.

Future development of AgentChess could be extended with the use of logic recognizing guaranteed move-attacks. This could be to prioritize moves with two possible attacks in front of move with a single attack. While the last one might be easily solved by the opponent, being able to protect yourself against two possible attacks in one move is harder. A move with a high profit attack might be best to be disregarded for a move-attack that can guarantee an attack profit, even though of a lower value.

3.2.5 Threat Test Data

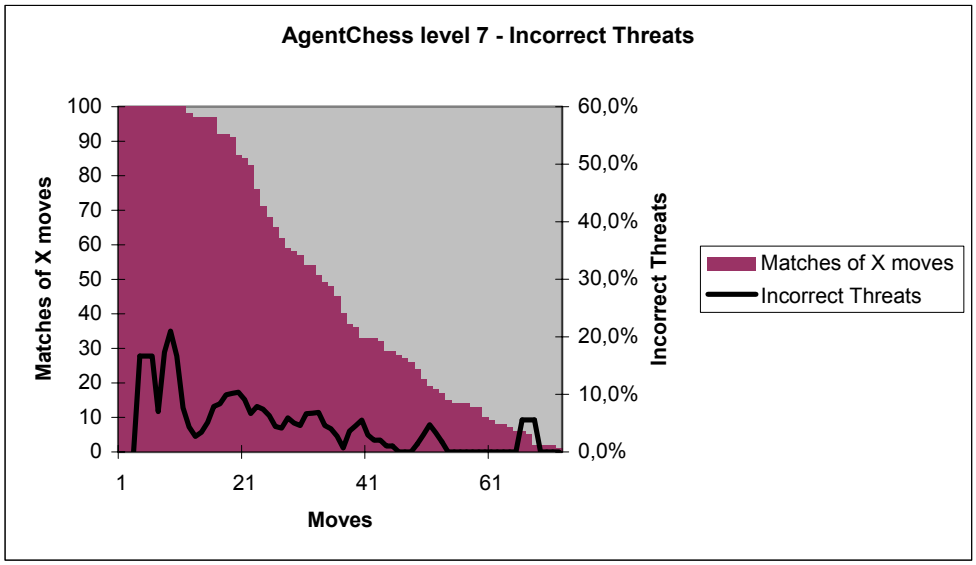

Figure 3.5: The mean value of the amount of incorrect threats during the 100 matches between AgentChess level 7 (defense consideration) and Crafty.

The test data of the amount of pieces considered not to be threatened, but even though captured, is visualized in *Figure 3.5*. The games played is performed by AgentChess level 7 (defense consideration) and Crafty. The results display a weakness of AgentChess, showing that about 20% of pieces estimated not to be in danger are nevertheless captured by Crafty during the first couple of moves. The amount of incorrect threats can be concluded to be of even higher rate as Crafty is only able to capture at most one piece at each move. As stated in the threat test description of subsection 3.1.4, we assume that the threat estimation of Crafty is correct. We based this assumption of the performance of the program and its established predecessor when playing chess.

The weakness of the threat estimation of AgentChess is no surprise. As described in AgentChess level 2 (threat consideration), the threat estimation logic of each piece is only considering at most one attacker and one defender during the calculations. This might be sufficient in some cases, but when it is not, the miscalculation can be disastrous for the program using it. The amount of incorrect threats might also be caused by game board situations such as when backup attackers are positioned behind the direct attackers of the threatened piece. Such situations of hidden threats are not possible to foresee by the ChessBoard module's logic supporting AgentChess with threat data.

A future development of AgentChess might show interesting results with performance increase if the threat estimation logic is re-written to consider a larger amount of facts.

3.2.6 Test Suit Data

Test Suite Data												
	BWCT		ECE3		ECM		TYPP		VA		WAC	
Positions	1001		1797		879		33		55		300	
Level 1	209	21%	236	13%	127	14%	3	9%	5	9%	53	18%
Level 2	39	4%	140	8%	1	0%	1	3%	6	11%	18	6%
Level 3	170	17%	364	20%	91	10%	4	12%	6	11%	46	15%
Level 4	119	12%	343	19%	81	9%	2	6%	7	13%	38	13%
Level 5	118	12%	334	19%	82	9%	4	12%	9	16%	38	13%
Level 6	121	12%	336	19%	80	9%	3	9%	7	13%	39	13%
Level 7	115	11%	341	19%	82	9%	4	12%	5	9%	36	12%
Crafty depth 1	201	20%	394	22%	72	8%	2	6%	5	9%	39	13%
Crafty depth 2	433	43%	526	29%	86	10%	4	12%	7	13%	73	24%
Crafty depth 3	658	66%	560	31%	126	14%	5	15%	9	16%	131	44%
Crafty time 5	957	96%	1051	58%	518	59%	9	27%	24	44%	287	96%

Table 3.10: The data of the test suites run for each level of AgentChess and Crafty with different restraints.

From the test suite data of *Table 3.10*, we see that in almost all cases are AgentChess able to match the move choices by Crafty searching at a tree depth of one level. The exception is test suite BWCT; the end game test suite of checkmates. The higher performance of Crafty at end games is not surprising as AgentChess is almost completely lacking logic for it. Nevertheless is AgentChess able to match Crafty, searching at depth 1, in the end game test suite ECE3 as well as the other test suits. In the middle game positional test TYPP, are several versions of AgentChess showing even better performance. Crafty is here forced to search to the depth of 2 levels to match the results of AgentChess. The same is true for test VA. However, the results of the other test suits here show Crafty to be the superior chess program. At the search depth of 3 levels, is Crafty able to outperform AgentChess. In the test suite results, as well as in the move-attack test in *Table 3.9*, are we also able to see a difference in game play between level 1 and level 2. An extra test of Crafty, able to search as much it can for a maximum of 5 seconds, was run to visualize the performance of a good traditional chess program running the test suites.
The test suite results confirm that the management of end game is a serious weakness of AgentChess. It is possible that an introduction of good end game logic could give AgentChess the ability to match Crafty if restrained to search at the level of one move ahead.

4 DISCUSSION

A discussion of the difference of chess strategy between the chess programs described in this report is performed in this chapter. A discussion of the test result data is also made.

4.1 Agent Approach vs. Traditional Approach

A justified question by the reader of the report could be "*How is this agent approach different from the traditional tree-searching approach using Minimax or similar techniques?*" The answer could be formulated in both simple and detailed manners. When simply specifying the difference, one can say that while a traditional search approach tests performing all possible moves and then make an evaluation of each result, this multi-agent system is instead just performing an evaluation of the possible result of its own and the opponent's moves. While the traditional approach is able to see all possible situations that can occur during the game play covered by the search, the agent approach have to guess and estimate the results of the possible actions available. A detailed example of what kind of uncertainty this estimation can result in will be made.

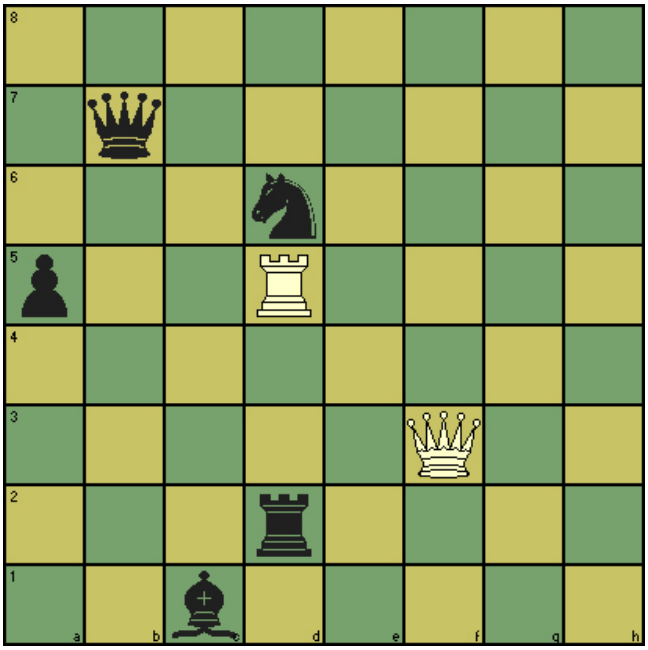

Figure 4.1: Visualization of a game position to describe the uncertainty of the AgentChess evaluation approach. Observe that as no kings exist the game board is actually illegal. They have been removed to simplify the example.

Figure 4.1 visualizes a position where AgentChess is playing as white. As its rock at D5 is threatened, the program will prioritize moving it to a save position. As the white rock has three attack possibilities, the one giving the highest profit is chosen as the escape option; the black knight at square D6. AgentChess is calculating threats by checking which pieces can move to which squares. Due to the positions of Figure 4.1, AgentChess is assuming no threats exist when capturing the knight. However, as it is moving, the black rock at D2 will get clear to move to D6 and capture the white rock. A wiser decision would then have been instead to capture the pawn to the left. Nevertheless, both these moves results in another problem. The moving of the rock results in the white queen being an undefended target of the black queen.

The example described shows some of the uncertainties that can occur using the move result estimation of a program like AgentChess. While a full move search by a traditional version have all the move result facts, the multi-agent system estimates them. The estimation is not necessarily complete or optimal. However, the threat of the queen by moving the rock is a problem that is handled in the last version of AgentChess; level 7 using defense consideration.

Another interesting aspect to discuss is the search depth performed by AgentChess. The move evaluation of AgentChess can be seen of something in the middle of 1 and 2 moves look ahead. AgentChess is calculating move possibilities of its own and its opponent simultaneously. A traditional program would test performing all its moves, follow up by testing the opponent's moves for all these cases and to finally end by making an evaluation of the results. AgentChess is not performing these two move evaluations in sequence but instead making calculations using what can be seen as estimated data of what the result would be if either side were to move next. The resulting move look ahead might be seen as 1.5 moves as the estimated data is not optimal, causing uncertainties as described in the example above. The situations missed would be seen by a traditional chess program searching at the depth of 2 moves.

Even though it exist uncertainties as described in the example above, the difference between an agent-based approach like AgentChess and a traditional program, searching only one move ahead is not so large. However, it should be stated that as the traditional chess program, represented by Crafty, is using advanced evaluation methods this advantage has to be considered when comparing them. When searching at a deeper level the theoretical difference should be larger, depending on the evaluation functions of the agent system. When developing a multi-agent chess program, it would probably be difficult to keep up with the traditional approach when estimating several moves ahead. For each move-depth to search, the complexity of the move calculations will presumably need to be widely extended. However, before a research and experiment of what a good solution for this could be is performed, first finding what the problems and solutions are at a shallow level is probably needed. That is what this report has been trying to accomplish.

4.2 AgentChess vs. Drogoul's MARCH

When comparing AgentChess to the chess strategy of Drogoul's MARCH [1], it can be stated that both methods are working in the same manner. The Drogoul program bases its decision on simple calculations of each square on the board. The square calculations are based on which piece that can move to them and the specific pieces' material value. The evaluation logic of AgentChess also uses the material of pieces and the square they can move to for deciding which side has control of which squares. However, the logic of AgentChess is more advanced; also considering other important aspects when evaluating attack request, defense request and defense consideration. This ability of the pieces cooperating, supporting and taking care of each other is the most important difference between AgentChess and MARCH. Nevertheless, MARCH was the inspiration when AgentChess was being formed and developed.

4.3 Test Results

The test data obtained gave both expected and unexpected results. While most of the extensions of AgentChess logic level performed better then its predecessor, exceptions were present. The agent level 6, first level of using defense requests, performed little better compared to its predecessor, level 5 (attack request). However, the following extended version, AgentChess level 7, which performed about the same as level 6, played worse against level 5. While level 8 of AgentChess proved to be successful against level 5 it showed opposite performance against Crafty. The different strategies gave different results against different opponents.

As AgentChess level 7 produced much better game results compared to the other versions when meeting Crafty, I concluded to consider this version to be the best. However, it is not optimal as level 8 was able to perform better in one case. The logic determining the use of defense request should probably be balanced in some other way. The solution to the problem is, however, not within this project. A possible future extension of AgentChess might answer how this problem could be solved.

The test results against Crafty produced better results than expected. Although Crafty is the superior chess player, AgentChess was at best able to win 6% of the matches and end another 6% in draw. AgentChess, which chess playing logic is fully based on material evaluation was not able to compete with Crafty's extra opening, positional and end game evaluations. The superiority of Crafty is quite clear.

5 CONCLUSIONS

The question asked in the title of this report; *"Can agents play chess?"* can be answered with a yes as well as a no. The answer yes can be motivated as follows. AgentChess, based on agent implementation technique, only looking what can be seen as 1.5 moves ahead (see section 4.1) and only using material difference to base its decision; is able actually win matches against a well-playing chess program like Crafty. This is done even though it is almost completely lacking end game logic. Although logic using better evaluation is needed to outperform this traditional chess program, AgentChess can be seen to perform relatively well. The skeptic however, might answer the question with a no. AgentChess is not able to reflect over situations that even a human beginner probably would not have much difficult to parse. Position evaluation, piece co-operation and game strategy modifications during game play is completely missed by the logic of the agents. In either case, this project shows that agents can be used to play chess at a basic level. The old saying, "*Rome was not built in a day",* fits quite well to the subject. Further development and research has to be made to show the full capabilities of agents and chess playing. The results of this project have hopefully shown and taken agent chess capabilities a step forward.

6 FUTURE OF AGENTCHESS

Further future development on AgentChess could show interesting results. There are still several important issues not taken under consideration by the logic of AgentChess during game play. A couple of examples will be described.

- An extension of the basic threat evaluation could be implemented. The logic used when calculating threats and defense is currently only considering only one threatening piece at each square; the cheapest opponent piece able to attack. Extending this logic could give good improvements of AgentChess' game play.
- Another issue to improve could be when considering move-attacks. When choosing which move-attack to prioritize the one with the highest profitable attack possibility is prioritized. This profit is based on the material of the most expensive opponent piece available to attack and its possible defense. However, this attack is not guaranteed as the opponent can decide to move the piece. Instead, prioritizing moves with two or more less profitable attacks could prove to be a wise choice. This strategy can guarantee an attack possibility, as the opponent is only able to move one piece. Even though it is sometimes possible to protect two pieces with one move, this strategy could increase profit in the other cases.
- The end evaluation is something to consider improving if extending AgentChess further. Currently any attack of the opponent king is prioritized which often can cause a disorganized witch-hunt of the king. Logic considering which move-attack of the king is most promising to lead to check mate could be introduced.
- A more challenging extension of AgentChess could be instead focus at increasing its search depth. AgentChess can be seen to be using the search depth of 1.5 levels (described in section 4.1), even though not optimal. To extend this and keep a good performance would probably not be an easy task; requiring sophisticated and advanced solutions.

7 REFERENCES

[1] A. Drogoul
When Ants Play Chess (Or Can Strategies Emerge From Tactical Behaviors) [LNAI Volume 957] (1995)

[2] S. Russel, P. Norvig
Artificial Intelligence – A Modern Approach [Prentice Hall] (1995)

[3] G. Weiss
Multiagent Systems –
A Modern Approach to Distributed Artificial Intelligence [MIT Press] (1999)

[4] R. M. Hyatt – Crafty Developer (2003-01-07)
http://www.cis.uab.edu/info/faculty/hyatt/hyatt.html
ftp://ftp.cis.uab.edu/pub/hyatt/

[5] R. M. Hyatt, H. L. Nelson
IEEE – Supercomputing '90, 354-383 (1990)
Chess and Supercomputers: details about optimizing Cray Blitz

[6] F. Grimes
Microsoft .NET for Programmers [Manning Publications Co.] (2002)

[7] R. Jones
Introduction to MFC Programming with Visual C++ [Prentice Hall PTR] (2000)

[8] Tim Mann's WinBoard / XBoard Interface (2003-01-07)
http://www.tim-mann.org/xboard.html

[9] ICC – Internet Chess Club (2003-01-07)
http://www.chessclub.com

[10] T. A. Marsland
Computer Chess Methods [University of Edmonton] (1987)

[11] .Bishop Computer Chess Center (2003-01-07)
http://www.kozachenko.com/dotbishop/

[12] K. Lang, W. Smith
A Test Suite for Chess Programs (1993)

[13] F. Reinfeld
The Complete Chess Course [Doubleday] (1990)

[14] R. Groß, K. Albrecht, W. Kantschik, W. Banzhaf
Evolving Chess Playing Programs (2002)