Modified Quiescence Procedure in Axon Chess Engine

Vladan Vučković [1]

Abstract – **This paper presents the modified Quiescence algorithm. As the framework for this improvement basic Quiescence procedure is defined. Some original modifications and illustrative positions are presented in detail. All of these theoretical results and novelties are successfully implemented and verified in authors' chess applications Axon and Achilles.**

Keywords – **Theory of Logic Games; Computer Chess; Algorithms; Search procedures.**

I. INTRODUCTION

Computer chess is one of the very important subject for researching in the field of artificial intelligence. The large amount of theoretical and practical algorithms and applications are developed and proved their value in practice.

The modern PC chess engines are reached the grandmaster strength. There is a set of standard algorithms including *Alfa-Beta, Null Move, Transposition tables, Opening and Endgame* tablebases for constructing such a chess engines. Also, there are many of other supporting algorithms responsible to maximize the engine play strength. One of them is *Quiescence procedure*.

By definition, the evaluation function is applied to the terminal nodes in the tree [1],[2]. Using algorithms to search that contain implemented minimax procedure, the value is prolonged in reverse direction through the tree to the root node. Even in the earliest stages of computer chess researchers observed a significant negative effect that may occur in the final stage of evaluation nodes. This effect is called *horizon effect* because of fatal errors that could be generated trying to evaluate terminal dynamic positions. The *Quiescence* procedure intends to solve that problem introducing the extra tree selective searching in terminal nodes, instead of simply to evaluate all of them. By removing the horizon effect, the engines significantly increases their strength [3],[4],[5].

This paper has following structure. After Introduction, we will define *Quiescence* procedure and present basic algorithm. In the next section we will show the original improvement of the function including a few support procedures. Tuning and balance of these new procedures is very important, so we deal with it in following of the paper. We will conclude the paper with implementation of these novelties in stand-alone test application and after that directly into the author's chess engines *Axon* and *Achilles*.

[1] Vladan Vuckovic is with Faculty of Electronic Engineering, ul.Aleksandra Medvedeva 14, Niš, Serbia, E-mail: vladan.vuckovic@elfak.ni.ac.rs vladanvuckovic24@gmail.com

II. QUIESCENCE PROCEDURE

To illustrate the horizon effect problem we will first postulate the basic variant of Quiescence procedure [6]. As an example, let's try to evaluate the position given in the following diagram (Figure 1):

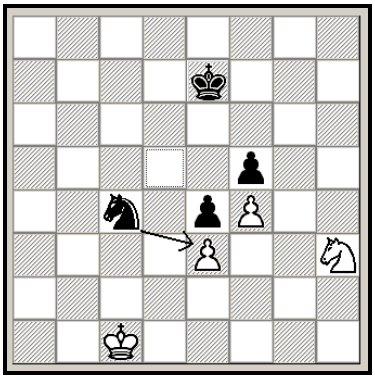

Fig. 1. The diagram shows a dynamic position that has to be evaluated.

Applying static evaluator in this position, where the material is completely identical for white and black, the evaluation would be very close to draw. However, if we take into account the dynamic parameters whether it is on the move white or black, it is clear that the white pawn on E3 cannot be saved, and that after the loss of that pawn, white will be forced to very difficult position: material evaluation is -1.00, black receives strong defended free of promoters by E line, the white pawn on F4 becomes very vulnerable. So, with the other parameters of evaluation positions created after taking a white piece could be freely approximated in interval -2.00 to -2.50 .

It is clear that attempting to evaluate a position that is not a clear static type led to significant errors, which could be crucial for the final evaluation and selection of the best moves [7]. In practical chess game, in every stage of the game, different positions generate a large number of unstable, dynamic position so that the application of the presented algorithms by definition would not lead to the realization of successful chess program. The solution for the horizon effect problem is that we must use specific procedures instead of pure evaluators to process the terminal nodes. These class of procedure are *Quiescence* searchers. This algorithm is a classic alpha-beta searcher with a slightly different logic to find the best continuation.

The procedure works as follows:

- In each generated terminal node the *Quiescence* searcher performs the evaluation [4]. If the evaluation value is greater or equal of the input variable is BETA, immediately exits the procedure. In each node, the heuristic generator determines the list of moves that can improve the existing evaluation.

These are most often moves that are taking the opponent's pieces (capture), some checks and promotions. Each of these moves is considered calling recursively *Quiescence* procedure. If the return value is greater than the variable BETA exit then the process is repeated but with improved value for ALFA. Because the moves in which take opposing figures represent in most cases the source of instability positions, their influence to these dynamic characteristics are successfully resolved. If we are in a position with no sharp bits from the group moves we mentioned, we directly do pure evaluation. In this way, in each node, the side that has the action provides the ability to maintain or improve its evaluation. Final positions are evaluated by static evaluator. In this way, this function overrides the horizon effect problem with introducing additional heuristics, so level of the game program improves significantly. Unfortunately, the introduction of *Quiescence* procedure, which in each node calls the dispute evaluative function, leads to a significant decrease in the overall speed *(PPS - position per second rate)* to the average level. It is necessary to very precisely adjust the balance among the amount of knowledge, speed and number of moves that are discussed in this procedure in order to reach the best performance [5]. In the following listing, we represent a basic form of *Quiescence* procedure:

```
procedure Quiescence (position, alpha, BETA):integer;
var …
 { Position – Current position }
 { Alpha – Upper limit }
 { BETA – Lower limit }

begin
   value:=evaluator(position)   {Static evaluation }

   if (value >= BETA) then   { If dynamic value is greater than
upper limit, exit }
   begin
     Quiescence:=BETA;
          exit;
   end;

   if (value > alpha) then alpha = value;  { New value for alpha
variable }
   GenerateCaptures();

   while (CapturesLeft) do    { Generate search branch for
every element of list }
   begin
    GenerateNextCapture;
    value := -Quiescence(-BETA, -alpha);      {   Quiescence
procedure recursive call }
    UnmakeMove;

    if (val >= BETA) then begin Quiescence:=BETA; exit; end;
{ Exit }
    if (val > alpha) then alpha = value;
   end;
   Quiescence:=alpha;
end;
```

Displayed procedure only considers the actions that take other figures (captures). Significant favorable factor in *Quiescence* procedure that considers only exchange pieces is auto-regulation of combinatorial explosion. Specifically, any recapture reduces the number of pieces on the board for one. After a series of piece recaptures, positions derives to a quasi-static position where there is no more active exchange of figures, and over which the procedure can be applied static evaluator.

III. Modified Quiescence Procedure

However, it is important to consider this procedure and extensions in which one party in chess, as in many variants comes to mate [3]. The procedure considers the checks incurred after taking some figures and reveals mates shown in the following listing:

```
procedure QuiescenceCheck (position, alpha, BETA):integer;
var …
 { Position – Current position }
 { Alpha – Upper limit }
 { BETA – Lower limit }

begin
   if (NodeIsInCheck) then  { If terminal node king is in check
… }
   begin
    Quiescence:=Quiescence(1, alpha, BETA);
    exit;
   end;

   if (value >= BETA) then   { If dynamic value is greater than
upper limit, exit }
   begin
    NULLMOVE:=BETA;
         exit;
   end;

   if (value > alpha) then alpha = value;   { New value for
alpha variable }
   GenerateCaptures();

   while (CapturesLeft) do   { Generate a branch for every
element in a list }
   begin
    GenerateNextCapture;
    value      :=      -QuiescenceCheck(-BETA,      -alpha);{
QuiescenceCheck procedure recursive check }
    UnmakeMove;
```

```
    if (val >= BETA) then begin QuiescenceCheck:=BETA; exit;
end; { Exit }
    if (val > alpha) then alpha = value;
   end;
   QuiescenceCheck:=alpha;
 end;
```

So it is the modified form of *QuiescenceCheck* recursive procedure. According to many authors, with the implementation of the checks in *Quiescence* procedure should be very careful, because a large percentage of a completely useless moves, which cannot improve evaluation leading to a significant increase in the number of processed nodes.

IV. MVV/LVA AND SEE PROCEDURES

A very important issue that must be addressed is how to sort the list of moves that need to be processed at each node. The two main techniques used are: MVV/LVA *(Most Valuable Victim/Least Valuable Attacker)* and SEE *(Static Exchange Evaluation)*. Both of these two procedure have found their place in modified *Quiescence* procedure [8]. MVV/LVA technique is used for brief screening of list moves toward the expected material gain. It will always move that takes the opponent queen that has a higher value of the moves that takes the opposing pawn. In the event that there are more pieces that can take the same opponent piece, the first deals with the suffix that means minimal material costs.

For example, if we have a move PxQ *(Pawn takes Queen)* it will be in the list moves ahead moves QxQ *(Queen takes a Queen)*. This technique does not take into account the fact that the opponent's pieces can be defended but is easily deployed and working very quickly.

As an illustration, we present an example of forming a list of moves in the position on the diagram using the second MVV/LVA techniques.

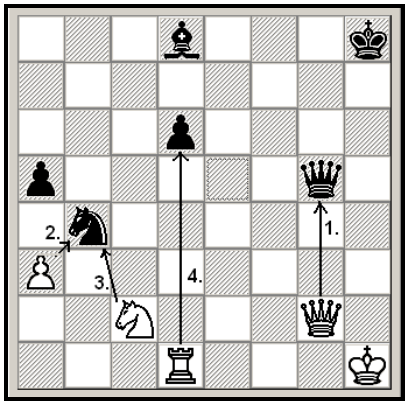

Fig. 2. The sequence of moves using the MVV/LVA procedure.

The sequence of moves is: (QxQ, PxN, NxN, RxP).

SEE technique is more complex than MVV/LVA and involves predicting the outcome of a series of trade figures. The procedure approximates the result of a series of amendments figure without having to call *Quiescence* procedures. Unlike the previously exposed techniques, if the opposing figure that is taken, defended by some other opponents figures, this fact is taken into account.

On the way to the much more realistically predicts the outcome of a series of trade figures and thus significantly improves the sequence of moves that are under consideration. Figure 2 shows the identical position to that of Figure 3, but is now used in processing SEE instead of MVV/LVA techniques:

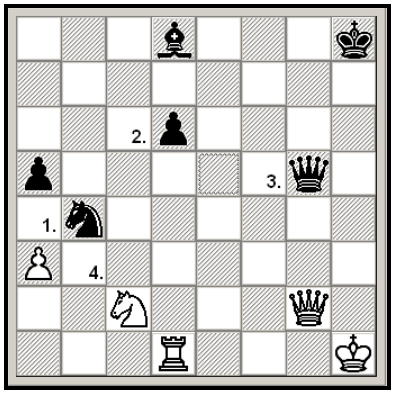

Fig. 3. The sequence of moves using the SEE technique.

The sequence of moves using the SEE techniques is slightly different: (PxN, RxP, QxQ, NxN). The sequence of moves that generates and order processing sub-tree is different in relation to the MVV/LVA technique. Bearing in mind the anticipated losses on each side, sort of moves is with much higher quality in the later stage generates a significantly higher number of cutting trees in *Quiescence* procedure.

In some older programs, SEE is used to approximate the dynamic evaluation of the terminal nodes of *Quiescence* without calling procedure, but in later versions of this it was changed in the way to eliminate too much of the tactical oversights. On the other hand, SEE procedure is considerably slower than the MVV/LVA procedure which reflects the overall NPS *(node per second)* feature of the program. However, in most modern programs including *Axon* [9], we use almost exclusively SEE.

V. APPLICATION AND TESTING OF THE QUIESCENCE PROCEDURE

After begin development of the new 32 bit version of the *Axon* chess engine, the author was faced with the problem of efficient development and testing of *Quiescence* procedure in the independent application [10].

This application is only for research purpose, it is open and intended only for the construction of some key procedures of the program [11]. Graphical environment for testing these functions is shown in Figure 4:

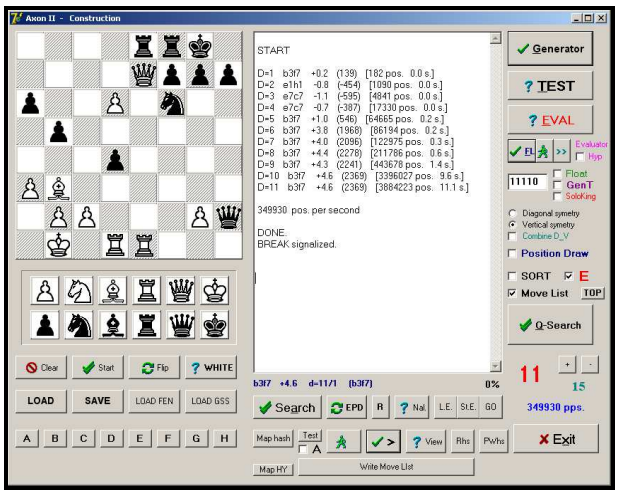

Fig. 4. Graphical environment of the program for testing of the Quiescence function.

The graphical environment contains all the necessary functions that are used for development of the *Quiescence* function. In addition to the standard options for creating and storing of the position, in relation to the visual display, various parameters and arguments and special options are embedded [12]. There is automatic test of the *Quiescence* function based on the random assignment of a set of N pieces on the chessboard, taking into account the legality of the position. Testing positions on various parameters including the symmetry is very fast, which significantly simplifies the programmer implementation and change certain aspects of the *Quiescence* function, that is necessary in a dynamic phase of development applications.

Connection to the main application is very simple and is done via UNIT common files used by the main application. After translation to machine language, the application automaticly create ENGINE.DCU file that contains all of the relevant data structures, procedures and functions for the realization of the chess machine [13].

At the stage of compilation of the basic program, *Axon II* simply lists used directive states in engine. dcu unit, so the connection is made in a very efficient manner. Control of individual processes in the unit ENGINE.DCU is done through a series of linked procedures. Every upgraded version of application then use new reference in ENGINE.DCU simply generating the new version of *Quiescence* function [14].

VI. Conclusion

This paper presents some original improvements embedded into the basic general Quiescence procedure with direct implementation in author's computer chess program *Axon.* Modified procedure with some extra procedures added (MVV/LVA,SEE) improved basic function and increase the total playing strength of the chess engine.

There are many possible tuning of these procedures. We will concentrate our algorithm to optimal solution balancing between efficiency and complexity needed for master computer chess game.

This solution has been already proved in direct implementation in *Axon's* tournament practice. Also, all other improvements will be checked in this way. The future prospect will be to increase amount of embedded chess knowledge into the *Quiescence/Evaluation* procedure using the constant rise of CPU processing power.

Acknowledgement

This paper is supported by the interdisciplinary project III44006 of the Ministry of Science and Technology of Republic of Serbia.

References

[1] Althöfer, I., Sören W. Perrey: Mathematische Methoden der Künstlichen Intelligenz: zur Quiescence-Suche in Spielbäumen. *ICCA Journal*, Vol. 14, No. 2, p. 84. ISSN 0920-234X, 1991.

[2] Beal, D. F. A Generalized Quiescence Search Algorithm. *Artificial Intelligence*, Vol. 43, No. 1, pp. 85-98. ISSN 0004-3702, 1990.

[3] Beal, D. F. Mating Sequences in the Quiescence Search. *ICCA Journal*, Vol. 7, No. 3, pp. 133-137. ISSN 0920-234X, 1984.

[4] Kaindl, H. Dynamic Control of the Quiescence Search in Computer Chess. *Cybernetics and Systems Research* (ed. R. Trappl), pp. 973-977. North-Holland, Amsterdam, 1982.

[5] Schrüfer, G. A Strategic Quiescence Search. *ICCA Journal*, Vol. 12, No. 1, pp. 3-9. ISSN 0920-234X, 1989.

[6] Slate D. J. , Atkin. L. R. Chess 4. 5 – "The Northwestern University Chess Program", *Chess Skill in Man and Machine* (ed. P. W. Frey), pp. 82-118. Springer-Verlag, New York, N. Y. 2nd ed. 1983. ISBN 0-387-90815-3. , 1977.

[7] Vučković, V. , Vidanović, Đ. "An Algorithm for the Detection of Move Repetition Without the use of Hash-Keys", *Yugoslav Journal of Operations Research (YUJOR*), Volume 17, Number 2, pp. 257- 274. Belgrade, Serbia. ISSN 0354-0243. , 2007.

[8] Vučković, V. "The Theoretical and Practical Application of the Advanced Chess Algorithms", *PhD Theses*, The Faculty of Electronic Engineering, The University of Nis, Serbia, 2006.

[9] Vučković, V. *xon/Achilles* experimental chess engines information could be find at: http://axon. elfak. ni. ac. rs , http://chess. elfak. ni. ac. rs, 2007.

[10] Vučković, V. , "The Compact Chessboard Representation", ICGA Journal, Volume31, Number 3, Tilburg, The Netherlands, ISSN 1389-6911. pp. 157- 164., 2008.

[11] Vučković, V. , "The Method of the Chess Search Algorithms Parallelization using Two-Processor Distributed System", The Scientific Journal Facta Universitatis, Series Mathematics and Informatics, Volume 22, Number 2, Niš , ISSN 0352-9665. pp. 175-188., 2007.

[12] Šolak, R. , and Šolak, Vučković, V. , "Time Management During a Chess Game", *ICGA Journal*, Volume 32, Number 4, Tilburg, The Netherlands, ISSN 1389-6911. pp. 206- 220., 2010.

[13] Vučković, V., "Advanced Chess Algorithms and Systems", monography, Zadužbina Andrejević, Biblioteka Dissertatio, ISSN 0354-7671, Belgrade, 2011

[14] Владан Вучковић, "Специјални елементи евалуационе функције", Зборник радова са 53. Конференције ЕТРАН-а, CD ROM Proceedings, Секција Вештачка интелигенција, рад VI1. 3, Vrnjačka Banja, (ISBN 978-86-80509-64-8), 2009.