**Notes for debugging**

m\_StartingColumnNumber/ RankNumber = From 1 to 8 (must deduce 1 to use in arrays)

m\_FinishingColumnNumber/ RankNumber = From 1 to 8 (must deduce 1 to use in arrays)

**Skakiera[(Column),(Rank)]**

Thinking depth must be an even number because the nodes are recorded only in the Analyze\_Computer functions!

**V0.970/1**

Fixed the Nodes Analysis: Created a different NodesAnalysis array for each level. Fixed bugs so that the analysis stores correctly the values and the MiniMax algorithm performs as it should.

Fixed the “Stupid move” filter for the computer moves.

Fixed the “Possibility to eat back” functionality.

**CHANGES**

Variables declaration

CheckForBlackCheck

CheckForBlackMate

CheckForWhiteCheck

CheckForWhiteMate

CheckMove

ComputerMove

CountScore

Enter\_move

Analyze\_Move\_1\_HumanMove (new)

Analyze\_Move\_3\_HumanMove (new)

Analyze\_Move\_5\_HumanMove (new – to be removed)

Analyze\_Move\_2\_ComputerMove (new)

Analyze\_Move\_4\_ComputerMove (new)

Analyze\_Move\_6\_ComputerMove (new – to be removed)

FindAttackers

FindDefenders

**If human plays first => EnterMove**

**(else ComputerMove is called directly from** Main\_Console()**)**

For the move entered by the human opponent…

* Check legality of the move
* Check for mate
* Check if there is check active
* Store move’s coordinates
* Store the value of the piece human moves
* Store the coordinates of where that piece moved [Human\_last\_move\_target\_rank/column]

**ComputerMove [Move\_Analyzed = 0]**

#region InitializeNodes //Initialize all nodes

#region StoreInitialPosition //Store initial position

#region OpeningBookCheck //OPENING BOOK CHECK

#region DangerousSquares //CHECK FOR DANGEROUS SQUARES

// Initialize variables (POTENTIALLY NOT NEEDED!)

For each possible move

{

// Check if the move is stupid

#region CheckStupidMove

If Move < 5 and you move the knight to the edges => Stupid move etc…

+ Store moving piece’s value

#endregion CheckStupidMove

If not stupid & Destination square not dangerous => ...

if ((ThisIsStupidMove.CompareTo("N") == 0) && (Skakiera\_Dangerous\_Squares[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)] == 0))...

Call CheckMove to check validity of the move

CheckMove(Skakiera\_Thinking, m\_StartingRank, m\_StartingColumnNumber, m\_FinishingRank, m\_FinishingColumnNumber, MovingPiece);

<Call CheckMove(Skakiera\_Thinking)> to:

* Check legality of the move
* Check for mate (This is not done! Must add it!)
* Check if there is check active
* ~~Store move’s coordinates~~ (REMOVED! This is done in ComputerMove!)
* ~~Store moving piece’s value~~ (This is not done! This is done in ComputerMove!)

// If move analyzed in the first: Store move to \*\*\*\_HY variables (so as to keep the initial move somewhere)

if (((m\_OrthotitaKinisis == true) && (m\_NomimotitaKinisis == true)) && (Move\_Analyzed == 0)) => ...

if ((m\_OrthotitaKinisis == true) && (m\_NomimotitaKinisis == true))

// Do the move

ProsorinoKommati = Skakiera\_Thinking[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)];

Skakiera\_Thinking[(m\_StartingColumnNumber - 1), (m\_StartingRank - 1)] = "";

Skakiera\_Thinking[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)] = MovingPiece;

Check the score after the computer move

~~if (Move\_Analyzed == 0) then...~~

NodeLevel\_0\_count++;

Temp\_Score\_Move\_0 = CountScore(Skakiera\_Thinking, humanDangerParameter);

~~if (Move\_Analyzed == 2) then...~~

~~NodeLevel\_2\_count++;~~

~~Temp\_Score\_Move\_2 = CountScore(Skakiera\_Thinking, humanDangerParameter);~~

// Store the best move score at this level

if ((m\_PlayerColor.CompareTo("Black") == 0) && (Temp\_Score\_Move\_0 > bestScoreLevel0))

{

bestScoreLevel0 = Temp\_Score\_Move\_0;

}

Document...

* ValueOfKommati
* ValueOfTargetPiece
* Check possibility to eat back

// v0.970: Check if you can eat back the piece of the human which moved!

if ((m\_FinishingColumnNumber == Human\_last\_move\_target\_column)

&& (m\_FinishingRank == Human\_last\_move\_target\_row)

&& (ValueOfMovingPiece <= ValueOfHumanMovingPiece))

{

Best\_Move\_StartingColumnNumber = m\_StartingColumnNumber;

Best\_Move\_StartingRank = m\_StartingRank;

Best\_Move\_FinishingColumnNumber = m\_FinishingColumnNumber;

Best\_Move\_FinishingRank = m\_FinishingRank;

possibility\_to\_eat\_back = true;

}

* Check possibility to eat

If thinking depth not reached, call next level of analysis

// v0.970: If you can eat back the piece of the human, then go for it and don't analyze!

if ((Move\_Analyzed < Thinking\_Depth) && (possibility\_to\_eat\_back == false) && (possibility\_to\_eat == false))

{

Move\_Analyzed = Move\_Analyzed + 1;

for (i = 0; i <= 7; i++)

{

for (j = 0; j <= 7; j++)

{

Skakiera\_Move\_After[(i), (j)] = Skakiera\_Thinking[(i), (j)];

}

}

Who\_Is\_Analyzed = "Human";

//First\_Call\_Human\_Thought = true;

// Check human move (to find the best possible answer of the human

// to the move currently analyzed by the HY Thought process)

if (Move\_Analyzed == 1)

Analyze\_Move\_1\_HumanMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 3)

Analyze\_Move\_3\_HumanMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 5)

Analyze\_Move\_5\_HumanMove(Skakiera\_Move\_After);

}

// Undo the move

Skakiera\_Thinking[(m\_StartingColumnNumber0 - 1), (m\_StartingRank0 - 1)] = MovingPiece0;

Skakiera\_Thinking[(m\_FinishingColumnNumber0 - 1), (m\_FinishingRank0 - 1)] = ProsorinoKommati0;

}

...

}

**Analyze\_Move\_1\_HumanMove [Move\_Analyzed = 1]**

Check all possible moves

{

if ((m\_OrthotitaKinisis == true) && (m\_NomimotitaKinisis == true)) then...

// Do the move

ProsorinoKommati = Skakiera\_Human\_Thinking\_2[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)];

Skakiera\_Human\_Thinking\_2[(m\_StartingColumnNumber - 1), (m\_StartingRank - 1)] = "";

Skakiera\_Human\_Thinking\_2[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)] = MovingPiece;

// Measure score AFTER the move

~~if (Move\_Analyzed == 1)~~

NodeLevel\_1\_count++;

Temp\_Score\_Move\_1\_human = CountScore(Skakiera\_Human\_Thinking\_2, humanDangerParameter);

~~if (Move\_Analyzed == 3)~~

~~NodeLevel\_3\_count++;~~

~~Temp\_Score\_Move\_3\_human = CountScore(Skakiera\_Human\_Thinking\_2, humanDangerParameter);~~

// Store the best move at this level

if ((m\_PlayerColor.CompareTo("Black") == 0) && (Temp\_Score\_Move\_1\_human < bestScoreLevel1))

{

bestScoreLevel1 = Temp\_Score\_Move\_1\_human;

}

If thinking depth not reached, call next level of analysis

if (Move\_Analyzed < Thinking\_Depth) && (Temp\_Score\_Move\_1\_human better than bestScoreLevel1)

{

Move\_Analyzed = Move\_Analyzed + 1;

Who\_Is\_Analyzed = "HY";

if (Move\_Analyzed == 2)

Analyze\_Move\_2\_ComputerMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 4)

Analyze\_Move\_4\_ComputerMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 6)

Analyze\_Move\_6\_ComputerMove(Skakiera\_Move\_After);

}

// Undo the move

Skakiera\_Human\_Thinking\_2[(m\_StartingColumnNumber1 - 1), (m\_StartingRank1 - 1)] = MovingPiece1;

Skakiera\_Human\_Thinking\_2[(m\_FinishingColumnNumber1 - 1), (m\_FinishingRank1 - 1)] = ProsorinoKommati1;

}

Move\_Analyzed = Move\_Analyzed - 1;

Who\_Is\_Analyzed = "HY";

**Analyze\_Move\_2\_ComputerMove [Move\_Analyzed = 2]**

Check all possible moves

{

if ((m\_OrthotitaKinisis == true) && (m\_NomimotitaKinisis == true))

{

// huo\_sw1.WriteLine(string.Concat("Human move 1: Found a legal move!"));

// Do the move

ProsorinoKommati = Skakiera\_Thinking\_HY\_2[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)];

Skakiera\_Thinking\_HY\_2[(m\_StartingColumnNumber - 1), (m\_StartingRank - 1)] = "";

Skakiera\_Thinking\_HY\_2[(m\_FinishingColumnNumber - 1), (m\_FinishingRank - 1)] = MovingPiece;

// Check the score after the computer move.

~~if (Move\_Analyzed == 0)~~

~~NodeLevel\_0\_count++;~~

~~Temp\_Score\_Move\_0 = CountScore(Skakiera\_Thinking\_HY\_2, humanDangerParameter);~~

~~if (Move\_Analyzed == 2)~~

NodeLevel\_2\_count++;

Temp\_Score\_Move\_2 = CountScore(Skakiera\_Thinking\_HY\_2, humanDangerParameter);

// Store the best score at this level

if ((m\_PlayerColor.CompareTo("Black") == 0) && (Temp\_Score\_Move\_2 > bestScoreLevel2))

{

bestScoreLevel2 = Temp\_Score\_Move\_2;

}

If thinking depth not reached, call next level of analysis

if (Move\_Analyzed < Thinking\_Depth)

{

Move\_Analyzed = Move\_Analyzed + 1;

Who\_Is\_Analyzed = "Human";

First\_Call\_Human\_Thought = true; < WHAT IS THIS???

// Check human move

if (Move\_Analyzed == 1)

Analyze\_Move\_1\_HumanMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 3)

Analyze\_Move\_3\_HumanMove(Skakiera\_Move\_After);

else if (Move\_Analyzed == 5)

Analyze\_Move\_5\_HumanMove(Skakiera\_Move\_After);

}

if (Move\_Analyzed == Thinking\_Depth)

{

// [MiniMax algorithm - skakos]

// Record the node in the Nodes Analysis array (to use with MiniMax algorithm) skakos

NodesAnalysis0[NodeLevel\_0\_count, 0] = Temp\_Score\_Move\_0;

NodesAnalysis1[NodeLevel\_1\_count, 0] = Temp\_Score\_Move\_1\_human;

NodesAnalysis2[NodeLevel\_2\_count, 0] = Temp\_Score\_Move\_2;

NodesAnalysis3[NodeLevel\_3\_count, 0] = Temp\_Score\_Move\_3\_human;

// Store the parents (number of the node of the upper level)

NodesAnalysis0[NodeLevel\_0\_count, 1] = 0;

NodesAnalysis1[NodeLevel\_1\_count, 1] = NodeLevel\_0\_count;

NodesAnalysis2[NodeLevel\_2\_count, 1] = NodeLevel\_1\_count;

NodesAnalysis3[NodeLevel\_3\_count, 1] = NodeLevel\_2\_count;

}

=> Because the analysis ends only in Analyze\_Move\_2\_ComputerMove functions, the ThinkigDepth must be an even number!

// Undo the move

Skakiera\_Thinking\_HY\_2[(m\_StartingColumnNumber2 - 1), (m\_StartingRank2 - 1)] = MovingPiece2;

Skakiera\_Thinking\_HY\_2[(m\_FinishingColumnNumber2 - 1), (m\_FinishingRank2 - 1)] = ProsorinoKommati2;

}

Move\_Analyzed = Move\_Analyzed - 1;

Who\_Is\_Analyzed = "Human";

**ComputerMove [Continued... - The End of Analysis]**

Check for mate

// DO THE BEST MOVE FOUND: Use MiniMax algorithm

if (possibility\_to\_eat\_back == false)

{

// [MiniMax algorithm - skakos]

// Find node 1 move with the best score via the MiniMax algorithm.

int counter0, counter1, counter2;

// ------------------------------------------------------

// NodesAnalysis

// ------------------------------------------------------

// Nodes structure...

// [ccc, xxx, 0]: Score of node No. ccc at level xxx

// [ccc, xxx, 1]: Parent of node No. ccc at level xxx-1

// ------------------------------------------------------

int parentNodeAnalyzed = -999;

//v0.980: Remove

//parentNodeAnalyzed = -999;

for (counter2 = 1; counter2 <= NodeLevel\_2\_count; counter2++)

{

if (Int32.Parse(NodesAnalysis2[counter2, 1].ToString()) != parentNodeAnalyzed)

{

//parentNodeAnalyzedchanged = true;

parentNodeAnalyzed = Int32.Parse(NodesAnalysis2[counter2, 1].ToString());

NodesAnalysis1[Int32.Parse(NodesAnalysis2[counter2, 1].ToString()), 0] = NodesAnalysis2[counter2, 0];

}

if (NodesAnalysis2[counter2, 0] >= NodesAnalysis1[Int32.Parse(NodesAnalysis2[counter2, 1].ToString()), 0])

NodesAnalysis1[Int32.Parse(NodesAnalysis2[counter2, 1].ToString()), 0] = NodesAnalysis2[counter2, 0];

}

// Now the Node1 level is filled with the score data

parentNodeAnalyzed = -999;

for (counter1 = 1; counter1 <= NodeLevel\_1\_count; counter1++)

{

if (Int32.Parse(NodesAnalysis1[counter1, 1].ToString()) != parentNodeAnalyzed)

{

//parentNodeAnalyzedchanged = true;

parentNodeAnalyzed = Int32.Parse(NodesAnalysis1[counter1, 1].ToString());

NodesAnalysis0[Int32.Parse(NodesAnalysis1[counter1, 1].ToString()), 0] = NodesAnalysis1[counter1, 0];

}

if (NodesAnalysis1[counter1, 0] <= NodesAnalysis0[Int32.Parse(NodesAnalysis1[counter1, 1].ToString()), 0])

NodesAnalysis0[Int32.Parse(NodesAnalysis1[counter1, 1].ToString()), 0] = NodesAnalysis1[counter1, 0];

}

// Choose the biggest score at the Node0 level

// Check example at http://en.wikipedia.org/wiki/Minimax#Example\_2

// Initialize the score with the first score and move found

double temp\_score = NodesAnalysis0[1, 0];

Best\_Move\_StartingColumnNumber = Int32.Parse(NodesAnalysis0[1, 2].ToString());

Best\_Move\_StartingRank = Int32.Parse(NodesAnalysis0[1, 4].ToString());

Best\_Move\_FinishingColumnNumber = Int32.Parse(NodesAnalysis0[1, 3].ToString());

Best\_Move\_FinishingRank = Int32.Parse(NodesAnalysis0[1, 5].ToString());

for (counter0 = 1; counter0 <= NodeLevel\_0\_count; counter0++)

{

if (NodesAnalysis0[counter0, 0] > temp\_score)

{

temp\_score = NodesAnalysis0[counter0, 0];

Best\_Move\_StartingColumnNumber = Int32.Parse(NodesAnalysis0[counter0, 2].ToString());

Best\_Move\_StartingRank = Int32.Parse(NodesAnalysis0[counter0, 4].ToString());

Best\_Move\_FinishingColumnNumber = Int32.Parse(NodesAnalysis0[counter0, 3].ToString());

Best\_Move\_FinishingRank = Int32.Parse(NodesAnalysis0[counter0, 5].ToString());

}

}

}

If no move found => Resign