**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)

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

**EnterMove**

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

// Check if the move is stupid

For each possible move

{

<Call CheckMove(Skakiera\_Thinking)> to:

* Check for mate
* Check if there is check active
* Store move’s coordinates
* Store moving piece’s value

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);

// 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;

}

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))

{

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);

}

if (Move\_Analyzed == Thinking\_Depth) // This will never be called unless ThinkingDepth is 0!!!

{

// [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;

}

// Undo the move

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

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

}

Check for mate

// DO THE BEST MOVE FOUND

// [MiniMax algorithm - skakos]

// Must comment/ uncomment according to ThinkingDepth!

// e.g. for ThinkingDepth = 4 you must uncomment the code for NodeLevel 4 but not anything before that!!!

**CheckMove**

number\_of\_moves\_analysed++;

Check correctness of move

CHECK FOR CHECK

Store the move to \*\*\*\_HY variables (Initial store)

if ((Move\_Analyzed == 0) && ... )

{

m\_StartingColumnNumber\_HY = m\_StartingColumnNumber;

m\_FinishingColumnNumber\_HY = m\_FinishingColumnNumber;

m\_StartingRank\_HY = m\_StartingRank;

m\_FinishingRank\_HY = m\_FinishingRank;

// Store the initial move coordinates (at the node 0 level)

NodeLevel\_0\_count++;

NodesAnalysis[NodeLevel\_0\_count, 21, 0] = m\_StartingColumnNumber\_HY;

NodesAnalysis[NodeLevel\_0\_count, 22, 0] = m\_FinishingColumnNumber\_HY;

NodesAnalysis[NodeLevel\_0\_count, 23, 0] = m\_StartingRank\_HY;

NodesAnalysis[NodeLevel\_0\_count, 24, 0] = m\_FinishingRank\_HY;

CHECK FOR DANGER PENALTY

...

}

**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);

If thinking depth not reached, call next level of analysis

if (Move\_Analyzed < Thinking\_Depth)

{

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);

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";