/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* tring to predictative orderly movments.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ramin Edjlal\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This Class should Predict the Validity Movements Of Current Order an Enemy of Current Order\*(\_)

\* Predict Not Working\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Chess Predict Taking A Lot Of Time\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess Prediction Caused to Initial AllDraw Method in ObjectDanger state not Working.\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* 'Check' ObjectDanger Attacker by Gray Minister to Brown 'King' Not Removed by Brown Soldier\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* The State of Soldier Supporter By Soldier Brown Doesn’t Detected.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess Predict Doesn’t Act The Supporter of Soldier\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess Predict Supporter Successful For Checking 'Alice' by Person\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess 'Alice' By 'Bob' Supporter Misleading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess Predict at Tow Level Taking a lot of time.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Chess Predict Not Working\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+:Sum(10)) (\_:Sum(1))\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Threading;

using System.Threading.Tasks;

using System.IO;

namespace RefrigtzDLL

{

[Serializable]

class ChessPerdict

{

//Initiate Global Variables.

public double MaxHuristicxT = Double.MinValue;

public bool MovementsAStarGreedyHuristicFoundT = false;

public bool IgnoreSelfObjectsT = false;

public bool UsePenaltyRegardMechnisamT = true;

public bool BestMovmentsT = false;

public bool PredictHuristicT = true;

public bool OnlySelfT = false;

public bool AStarGreedyHuristicT = false;

bool ArrangmentsChanged = false;

/\*public int SodierMidle = 8;

public int SodierHigh = 16;

public int ElefantMidle = 2;

public int ElefantHigh = 4;

public int HourseMidle = 2;

public int HourseHight = 4;

public int CastleMidle = 2;

public int CastleHigh = 4;

public int MinisterMidle = 1;

public int MinisterHigh = 2;

public int KingMidle = 1;

public int KingHigh = 2;

\*/

public int SodierMidle = 0;

public int SodierHigh = 0;

public int ElefantMidle = 0;

public int ElefantHigh = 0;

public int HourseMidle = 0;

public int HourseHight = 0;

public int CastleMidle = 0;

public int CastleHigh = 0;

public int MinisterMidle = 0;

public int MinisterHigh = 0;

public int KingMidle = 0;

public int KingHigh = 0;

ChessPerdict APredict = null;

int OrderDummy = 0;

public static int SodierValue = 1;

public static int ElefantValue = 1;

public static int HourseValue = 1;

public static int CastleValue = 1;

public static int MinisterValue = 1;

public static int KingValue = 1;

int RW = 0;

int CL = 0;

int Ki = 0;

public static int LoopHuristicIndex = 0;

static List<int> RWList = new List<int>();

static List<int> ClList = new List<int>();

static List<int> KiList = new List<int>();

static public List<int[,]> TableListAction = new List<int[,]>();

public int Move = 0;

static public int MouseClick = 0;

int[] AStarGreedyIndex = new int[20];

public List<AllDraw> A = null;

public List<int[,]> TableList = new List<int[,]>();

public int AStarGreedyGreedy = 0;

public DrawSoldier[] SolderesOnTable = null;

public DrawElefant[] ElephantOnTable = null;

public DrawHourse[] HoursesOnTable = null;

public DrawCastle[] CastlesOnTable = null;

public DrawMinister[] MinisterOnTable = null;

public DrawKing[] KingOnTable = null;

int CurrentAStarGredyMax = -1;

//AllDraw. THIS;

static void Log(Exception ex)

{

try

{

Object a = new Object();

lock (a)

{

string stackTrace = ex.ToString();

File.AppendAllText(AllDraw.Root + "\\ErrorProgramRun.txt", stackTrace + ": On" + DateTime.Now.ToString()); // path of file where stack trace will be stored.

}

}

catch (Exception t) { Log(t); }

}

public void SetObjectNumbers(int[,] TabS)

{

SodierMidle = 0;

SodierHigh = 0;

ElefantMidle = 0;

ElefantHigh = 0;

HourseMidle = 0;

HourseHight = 0;

CastleMidle = 0;

CastleHigh = 0;

MinisterMidle = 0;

MinisterHigh = 0;

KingMidle = 0;

KingHigh = 0;

for (int h = 0; h < 8; h++)

for (int s = 0; s < 8; s++)

{

if (TabS[h, s] == 1)

{

SodierMidle++;

SodierHigh++;

}

else if (TabS[h, s] == 2)

{

ElefantMidle++;

ElefantHigh++;

}

else if (TabS[h, s] == 3)

{

HourseMidle++;

HourseHight++;

}

else if (TabS[h, s] == 4)

{

CastleMidle++;

CastleHigh++;

}

else if (TabS[h, s] == 5)

{

MinisterMidle++;

MinisterHigh++;

}

else if (TabS[h, s] == 6)

{

KingMidle++;

KingHigh++;

}

else

if (TabS[h, s] == -1)

{

SodierHigh++;

}

else if (TabS[h, s] == -2)

{

ElefantHigh++;

}

else if (TabS[h, s] == -3)

{

HourseHight++;

}

else if (TabS[h, s] == -4)

{

CastleHigh++;

}

else if (TabS[h, s] == -5)

{

MinisterHigh++;

}

else if (TabS[h, s] == -6)

{

KingHigh++;

}

}

}

float[] FoundLocationOfObject(ref int[,] Tabl, int Kind, bool IsGray)

{

float[] Location = { -1, -1 };

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

{

if (IsGray)

{

if (Tabl[i, j] == Kind)

{

Location[0] = i;

Location[1] = j;

Tabl[i, j] = 0;

}

}

else

{

if (Tabl[i, j] \* -1 == Kind)

{

Location[0] = i;

Location[1] = j;

Tabl[i, j] = 0;

}

}

}

return Location;

}

//Constructor.

public ChessPerdict(int CurrentAStarGredy, bool MovementsAStarGreedyHuristicTFou, bool IgnoreSelfObject, bool UsePenaltyRegardMechnisa, bool BestMovment, bool PredictHurist, bool OnlySel, bool AStarGreedyHuris, bool Arrangments//, ref AllDraw. Th

)

{

CurrentAStarGredyMax = CurrentAStarGredy;

MaxHuristicxT = Double.MinValue;

MovementsAStarGreedyHuristicFoundT = MovementsAStarGreedyHuristicTFou;

IgnoreSelfObjectsT = IgnoreSelfObject;

UsePenaltyRegardMechnisamT = UsePenaltyRegardMechnisa;

BestMovmentsT = BestMovment;

PredictHuristicT = PredictHurist;

OnlySelfT = OnlySel;

AStarGreedyHuristicT = AStarGreedyHuris;

ArrangmentsChanged = Arrangments;

//Initiate Global Variable By Local Parameters.

//THIS = Th;

//A = new List<AllDraw>();

if (TableList.Count > 0)

{

Color A = Color.Gray;

int[,] Tab = new int[8, 8];

for (int g = 0; g < 8; g++)

for (int k = 0; k < 8; k++)

Tab[g, k] = TableList[0][g, k];

int[,] Tabl = new int[8, 8];

for (int g = 0; g < 8; g++)

for (int k = 0; k < 8; k++)

Tabl[g, k] = TableList[0][g, k];

int Order = 1;

bool TB = false;

SolderesOnTable = new DrawSoldier[SodierHigh];

for (int i = 0; i < SodierHigh; i++)

{

float[] Location = null;

if (i <= SodierMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

Order = 1;

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

SolderesOnTable[i] = new DrawSoldier(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

ElephantOnTable = new DrawElefant[ElefantHigh];

for (int i = 0; i < ElefantHigh; i++)

{

float[] Location = null;

if (i <= ElefantMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

Order = 1;

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

ElephantOnTable[i] = new DrawElefant(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

HoursesOnTable = new DrawHourse[HourseHight];

for (int i = 0; i < HourseHight; i++)

{

float[] Location = null;

if (i <= HourseMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

Order = 1;

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

HoursesOnTable[i] = new DrawHourse(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

CastlesOnTable = new DrawCastle[CastleHigh];

for (int i = 0; i < CastleHigh; i++)

{

float[] Location = null;

if (i <= CastleMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

Order = 1;

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

CastlesOnTable[i] = new DrawCastle(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

MinisterOnTable = new DrawMinister[MinisterHigh];

for (int i = 0; i < MinisterHigh; i++)

{

float[] Location = null;

if (i <= MinisterMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

Order = 1;

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

MinisterOnTable[i] = new DrawMinister(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

KingOnTable = new DrawKing[KingHigh];

for (int i = 0; i < KingHigh; i++)

{

float[] Location = null;

if (i <= KingMidle)

{

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

}

KingOnTable[i] = new DrawKing(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

}

}

//Determination of Current Thinking Operations Task Finished His Worke.

public bool AllCurrentAStarGreedyThinkingFinished(AllDraw Dum, int i, int j, int Kind)

{

//Initiate Local Variables.

bool Finished = false;

//Soldeir Kind.

if (Kind == 1)

{

//Wait For Flag Become Valid.

if (Dum.SolderesOnTable[i].SoldierThinking[j].ThinkingFinished)

return true;

}

//Elephant Kind.

else if (Kind == 2)

{

//Wait For Flag Become Valid.

if (Dum.ElephantOnTable[i].ElefantThinking[j].ThinkingFinished)

return true;

}//Hourse Kind.

else if (Kind == 3)

{

//Wait For Flag Become Valid.

if (Dum.HoursesOnTable[i].HourseThinking[j].ThinkingFinished)

return true;

}//Castles Kind.

else if (Kind == 4)

{

//Wait For Flag Become Valid.

if (Dum.CastlesOnTable[i].CastleThinking[j].ThinkingFinished)

return true;

}//Minister Kind.

else if (Kind == 5)

{

//Wait For Flag Become Valid.

if (Dum.MinisterOnTable[i].MinisterThinking[j].ThinkingFinished)

return true;

}//King Kind.

else if (Kind == 6)

{

//Wait For Flag Become Valid.

if (Dum.KingOnTable[i].KingThinking[j].ThinkingFinished)

return true;

}

//Return Flag.

return Finished;

}

//Wait Method For Thinking Operation.

void Wait(AllDraw Dum, int i, int j, int Kind)

{

//Wait For All Thinking Operation Finished.

do

{

//THIS.SetBoxText("\r\nAStarGreedy Predict :" + AllDraw.SyntaxToWrite);

//THIS.RefreshBoxText();

} while (!AllCurrentAStarGreedyThinkingFinished(Dum, i, j, Kind));

}

//Initiate For Every Initiation Objects.

public void InitiateForEveryKindThingHome(AllDraw DummyHA, int ii, int jj, Color a, int[,] Table, int Order, bool TB, int IN)

{

int i = 0, j = 0;

AllDraw Dummy = new AllDraw(Order, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged);

//Gray Order.

if (Order == 1)

{

//For All Gray Soldiers.

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

{

try

{

//When Current Soldeir is Not Existing Continue Traversal Back.

if (SolderesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)SolderesOnTable[i].Row;

jj = (int)SolderesOnTable[i].Column;

//Construction of Thinking Solders Gray Object.

Dummy.SolderesOnTable[i] = new DrawSoldier( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Movments.

for (j = 0; j < AllDraw.SodierMovments; j++)

{

//Thinking Operations.

Dummy.SolderesOnTable[i].SoldierThinking[j].TableT = SolderesOnTable[i].SoldierThinking[j].TableT;

Dummy.SolderesOnTable[i].SoldierThinking[j].ThinkingBegin = true;

Dummy.SolderesOnTable[i].SoldierThinking[j].ThinkingFinished = false;

Dummy.SolderesOnTable[i].SoldierThinking[j].t = new Task(new Action(Dummy.SolderesOnTable[i].SoldierThinking[j].Thinking));

Dummy.SolderesOnTable[i].SoldierThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 1);

}

}

catch (Exception t)

{

Dummy.SolderesOnTable[i] = null; Log(t);

}

}

//For All Gray Elephant Objects.

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

{

try

{

//When Gray Elephant Not Existing Continue Traversal Back.

if (ElephantOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)ElephantOnTable[i].Row;

jj = (int)ElephantOnTable[i].Column;

//Construction of Gray Elepahnt Thinking Objectes.

Dummy.ElephantOnTable[i] = new DrawElefant( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Movments.

for (j = 0; j < AllDraw.ElefantMovments; j++)

{

//Elephant Gray Thinking Operations.

Dummy.ElephantOnTable[i].ElefantThinking[j].TableT = ElephantOnTable[i].ElefantThinking[j].TableT;

Dummy.ElephantOnTable[i].ElefantThinking[j].ThinkingBegin = true;

Dummy.ElephantOnTable[i].ElefantThinking[j].ThinkingFinished = false;

Dummy.ElephantOnTable[i].ElefantThinking[j].t = new Task(new Action(Dummy.ElephantOnTable[i].ElefantThinking[j].Thinking));

Dummy.ElephantOnTable[i].ElefantThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 2);

}

}

catch (Exception t)

{

Dummy.ElephantOnTable[i] = null; Log(t);

}

}

//For All Hourse Gray Objects.

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

{

try

{

//When Gray Hourses Not Exsisting Continue Traversal Back.

if (HoursesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)HoursesOnTable[i].Row;

jj = (int)HoursesOnTable[i].Column;

Dummy.HoursesOnTable[i] = new DrawHourse( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Movments.

for (j = 0; j < AllDraw.HourseMovments; j++)

{

//Hourse Thinking Gray Objects Operations.

Dummy.HoursesOnTable[i].HourseThinking[j].TableT = HoursesOnTable[i].HourseThinking[j].TableT;

Dummy.HoursesOnTable[i].HourseThinking[j].ThinkingBegin = true;

Dummy.HoursesOnTable[i].HourseThinking[j].ThinkingFinished = false;

Dummy.HoursesOnTable[i].HourseThinking[j].t = new Task(new Action(Dummy.HoursesOnTable[i].HourseThinking[j].Thinking));

Dummy.HoursesOnTable[i].HourseThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 3);

}

}

catch (Exception t)

{

Dummy.HoursesOnTable[i] = null; Log(t);

}

}

//For All Castles Gray Objects.

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

{

try

{

//When Gray Brideges Not Exsisting Traversal Back.

if (CastlesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)CastlesOnTable[i].Row;

jj = (int)CastlesOnTable[i].Column;

//Construction of Bridegs Gray With Local variables.

Dummy.CastlesOnTable[i] = new DrawCastle( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

for (j = 0; j < 16; j++)

{

//Gray Castles Thinking Operations.

Dummy.CastlesOnTable[i].CastleThinking[j].TableT = CastlesOnTable[i].CastleThinking[j].TableT;

Dummy.CastlesOnTable[i].CastleThinking[j].ThinkingBegin = true;

Dummy.CastlesOnTable[i].CastleThinking[j].ThinkingFinished = false;

Dummy.CastlesOnTable[i].CastleThinking[j].t = new Task(new Action(Dummy.CastlesOnTable[i].CastleThinking[j].Thinking));

Dummy.CastlesOnTable[i].CastleThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 4);

}

}

catch (Exception t)

{

Dummy.CastlesOnTable[i] = null; Log(t);

}

}

//For All Minister Objets.

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

{

try

{

//Whe Gray Minister Not Exsisting Continue Traversal back.

if (MinisterOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)MinisterOnTable[i].Row;

jj = (int)MinisterOnTable[i].Column;

//Constructionof Ministerb Gray With Local Variables.

Dummy.MinisterOnTable[i] = new DrawMinister( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Movments.

for (j = 0; j < AllDraw.MinisterMovments; j++)

{

//Thinking Gray Ministers Operations.

Dummy.MinisterOnTable[i].MinisterThinking[j].TableT = MinisterOnTable[i].MinisterThinking[j].TableT;

Dummy.MinisterOnTable[i].MinisterThinking[j].ThinkingBegin = true;

Dummy.MinisterOnTable[i].MinisterThinking[j].ThinkingFinished = false;

Dummy.MinisterOnTable[i].MinisterThinking[j].t = new Task(new Action(Dummy.MinisterOnTable[i].MinisterThinking[j].Thinking));

Dummy.MinisterOnTable[i].MinisterThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 5);

}

}

catch (Exception t)

{

Dummy.MinisterOnTable[i] = null; Log(t);

}

}

//For All Possible Gray Kings.

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

{

try

{

//When Gray King Not Exsisting Continue Traversal Back.

if (KingOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)KingOnTable[i].Row;

jj = (int)KingOnTable[i].Column;

//Construction of Gray King With Local Variables.

Dummy.KingOnTable[i] = new DrawKing( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Movements.

for (j = 0; j < AllDraw.KingMovments; j++)

{

//Thinking Gray King Operatons.

Dummy.KingOnTable[i].KingThinking[j].TableT = KingOnTable[i].KingThinking[j].TableT;

Dummy.KingOnTable[i].KingThinking[j].ThinkingBegin = true;

Dummy.KingOnTable[i].KingThinking[j].ThinkingFinished = false;

Dummy.KingOnTable[i].KingThinking[j].t = new Task(new Action(Dummy.KingOnTable[i].KingThinking[j].Thinking));

Dummy.KingOnTable[i].KingThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 6);

}

}

catch (Exception t)

{

Dummy.KingOnTable[i] = null; Log(t);

}

}

}

else//Brown Order.

{

//For All Possible Brown Solders.

for (i = SodierMidle; i < SodierHigh; i++)

{

try

{

//Whn Not Existing Braown Solder Continue Traversal Back.

if (SolderesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)SolderesOnTable[i].Row;

jj = (int)SolderesOnTable[i].Column;

//Construction Of Brown Soldeir With Local Variables.

Dummy.SolderesOnTable[i] = new DrawSoldier( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

for (j = 0; j < 4; j++)

{

//Thinking of Brown Soldiers Operations.

Dummy.SolderesOnTable[i].SoldierThinking[j].TableT = SolderesOnTable[i].SoldierThinking[j].TableT;

Dummy.SolderesOnTable[i].SoldierThinking[j].ThinkingBegin = true;

Dummy.SolderesOnTable[i].SoldierThinking[j].ThinkingFinished = false;

Dummy.SolderesOnTable[i].SoldierThinking[j].t = new Task(new Action(Dummy.SolderesOnTable[i].SoldierThinking[j].Thinking));

Dummy.SolderesOnTable[i].SoldierThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 1);

}

}

catch (Exception t)

{

Dummy.SolderesOnTable[i] = null; Log(t);

}

}

//For All Brown elepahnt Objects.

for (i = ElefantMidle; i < ElefantHigh; i++)

{

try

{

//Continue Traversal Back Of Non Existing Objects.

if (ElephantOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)ElephantOnTable[i].Row;

jj = (int)ElephantOnTable[i].Column;

//Construction of Brown Elephant Thinking Object.

Dummy.ElephantOnTable[i] = new DrawElefant( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

for (j = 0; j < AllDraw.ElefantMovments; j++)

{

//Thinking Brown Elephant Operations.

Dummy.ElephantOnTable[i].ElefantThinking[j].TableT = ElephantOnTable[i].ElefantThinking[j].TableT;

Dummy.ElephantOnTable[i].ElefantThinking[j].ThinkingBegin = true;

Dummy.ElephantOnTable[i].ElefantThinking[j].ThinkingFinished = false;

Dummy.ElephantOnTable[i].ElefantThinking[j].t = new Task(new Action(Dummy.ElephantOnTable[i].ElefantThinking[j].Thinking));

Dummy.ElephantOnTable[i].ElefantThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 2);

}

}

catch (Exception t)

{

Dummy.ElephantOnTable[i] = null; Log(t);

}

}

//For All Possible Hourse Objects.

for (i = HourseMidle; i < HourseHight; i++)

{

try

{

//For Non Existing Brown Elephant Continue Traversal Back.

if (HoursesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)HoursesOnTable[i].Row;

jj = (int)HoursesOnTable[i].Column;

//Construction of Brown Hourse With Local Variables.

Dummy.HoursesOnTable[i] = new DrawHourse( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Hourse Movments.

for (j = 0; j < AllDraw.HourseMovments; j++)

{

//Thinking of Brown Hourse Operations.

Dummy.HoursesOnTable[i].HourseThinking[j].TableT = HoursesOnTable[i].HourseThinking[j].TableT;

Dummy.HoursesOnTable[i].HourseThinking[j].ThinkingBegin = true;

Dummy.HoursesOnTable[i].HourseThinking[j].ThinkingFinished = false;

Dummy.HoursesOnTable[i].HourseThinking[j].t = new Task(new Action(Dummy.HoursesOnTable[i].HourseThinking[j].Thinking));

Dummy.HoursesOnTable[i].HourseThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 3);

}

}

catch (Exception t)

{

Dummy.HoursesOnTable[i] = null; Log(t);

}

}

//For All Bridesg Brown Objects.

for (i = CastleMidle; i < CastleHigh; i++)

{

try

{

//When Brown Castles Non Existing Continue Traversal Back.

if (CastlesOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)CastlesOnTable[i].Row;

jj = (int)CastlesOnTable[i].Column;

//Construction of Brown Castles With Local Variables.

Dummy.CastlesOnTable[i] = new DrawCastle( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Castles Movments.

for (j = 0; j < AllDraw.CastleMovments; j++)

{

//Thinking of Brown Castles Operations.

Dummy.CastlesOnTable[i].CastleThinking[j].TableT = CastlesOnTable[i].CastleThinking[j].TableT;

Dummy.CastlesOnTable[i].CastleThinking[j].ThinkingBegin = true;

Dummy.CastlesOnTable[i].CastleThinking[j].ThinkingFinished = false;

Dummy.CastlesOnTable[i].CastleThinking[j].t = new Task(new Action(Dummy.CastlesOnTable[i].CastleThinking[j].Thinking));

Dummy.CastlesOnTable[i].CastleThinking[j].t.Start();

//Wait For Thinking Finsishing.

Wait(Dummy, i, j, 4);

}

}

catch (Exception t)

{

Dummy.CastlesOnTable[i] = null; Log(t);

}

}

//For All Possible Brown Minster Objects.

for (i = MinisterMidle; i < MinisterHigh; i++)

{

try

{

//When Brown Minister Non Existing Continue Traversal Back.

if (MinisterOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)MinisterOnTable[i].Row;

jj = (int)MinisterOnTable[i].Column;

//Construction of Brown Minister Thinking Objects.

Dummy.MinisterOnTable[i] = new DrawMinister( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Minister Brown Movments.

for (j = 0; j < AllDraw.MinisterMovments; j++)

{

//Brown Minister Thinking Operations.

Dummy.MinisterOnTable[i].MinisterThinking[j].TableT = MinisterOnTable[i].MinisterThinking[j].TableT;

Dummy.MinisterOnTable[i].MinisterThinking[j].ThinkingBegin = true;

Dummy.MinisterOnTable[i].MinisterThinking[j].ThinkingFinished = false;

Dummy.MinisterOnTable[i].MinisterThinking[j].t = new Task(new Action(Dummy.MinisterOnTable[i].MinisterThinking[j].Thinking));

Dummy.MinisterOnTable[i].MinisterThinking[j].t.Start();

Wait(Dummy, i, j, 5);

}

}

catch (Exception t)

{

Dummy.MinisterOnTable[i] = null; Log(t);

}

}

//For All Brown King Objects.

for (i = KingMidle; i < KingHigh; i++)

{

try

{

//When Brown King Non Existing Continue Traversal Back.

if (KingOnTable[i] == null)

continue;

//Initiate Local Variables.

ii = (int)KingOnTable[i].Row;

jj = (int)KingOnTable[i].Column;

//Construction of Brown King Thinking Operation.

Dummy.KingOnTable[i] = new DrawKing( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ii, jj, a, Table, Order, false, i);

//For All Possible Brown King Movements.

for (j = 0; j < AllDraw.KingMovments; j++)

{

//Thinking of Brown King Thinking Operations.

Dummy.KingOnTable[i].KingThinking[j].TableT = KingOnTable[i].KingThinking[j].TableT;

Dummy.KingOnTable[i].KingThinking[j].ThinkingBegin = true;

Dummy.KingOnTable[i].KingThinking[j].ThinkingFinished = false;

Dummy.KingOnTable[i].KingThinking[j].t = new Task(new Action(Dummy.KingOnTable[i].KingThinking[j].Thinking));

Dummy.KingOnTable[i].KingThinking[j].t.Start();

//Wait For Thinking Finished.

Wait(Dummy, i, j, 6);

}

}

catch (Exception t)

{

Dummy.KingOnTable[i] = null; Log(t);

}

}

}

A.Add(Dummy);

}

//Rearrange AllDraw Object Content.

public void SetRowColumn(int index)

{

try

{

Move = 0;

//Intiate Dummy Variables.

int So1 = 0;

int So2 = SodierMidle;

int El1 = 0;

int El2 = ElefantMidle;

int Ho1 = 0;

int Ho2 = HourseMidle;

int Br1 = 0;

int Br2 = CastleMidle;

int Mi1 = 0;

int Mi2 = MinisterMidle;

int Ki1 = 0;

int Ki2 = KingMidle;

//When Conversion Occured.

if (TableList.Count > 0)

{

Color A = Color.Gray;

int[,] Tab = new int[8, 8];

for (int g = 0; g < 8; g++)

for (int k = 0; k < 8; k++)

Tab[g, k] = TableList[0][g, k];

int[,] Tabl = new int[8, 8];

for (int g = 0; g < 8; g++)

for (int k = 0; k < 8; k++)

Tabl[g, k] = TableList[0][g, k];

int Order = 1;

bool TB = false;

SolderesOnTable = new DrawSoldier[SodierHigh];

for (int i = 0; i < SodierHigh; i++)

{

float[] Location = null;

if (i <= SodierMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

Order = -1;

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

}

SolderesOnTable[i] = new DrawSoldier(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

ElephantOnTable = new DrawElefant[ElefantHigh];

for (int i = 0; i < ElefantHigh; i++)

{

float[] Location = null;

if (i <= ElefantMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

Order = -1;

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

}

ElephantOnTable[i] = new DrawElefant(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

HoursesOnTable = new DrawHourse[HourseHight];

for (int i = 0; i < HourseHight; i++)

{

float[] Location = null;

if (i <= HourseMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

HoursesOnTable[i] = new DrawHourse(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

CastlesOnTable = new DrawCastle[CastleHigh];

for (int i = 0; i < CastleHigh; i++)

{

float[] Location = null;

if (i <= CastleMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

Order = -1;

}

CastlesOnTable[i] = new DrawCastle(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

MinisterOnTable = new DrawMinister[MinisterHigh];

for (int i = 0; i < MinisterHigh; i++)

{

float[] Location = null;

if (i <= MinisterMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

Order = -1;

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

}

MinisterOnTable[i] = new DrawMinister(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

KingOnTable = new DrawKing[KingHigh];

for (int i = 0; i < KingHigh; i++)

{

float[] Location = null;

if (i <= KingMidle)

{

Order = 1;

A = Color.Gray;

Location = FoundLocationOfObject(ref Tabl, 1, true);

}

else

{

Order = -1;

A = Color.Brown;

Location = FoundLocationOfObject(ref Tabl, -1, false);

}

KingOnTable[i] = new DrawKing(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Location[0], Location[1], A, Tab, Order, TB, i);

}

AllDraw.SodierConversionOcuured = false;

}

//When Table Exist.

if (TableList.Count > 0)

{

//For Every Table Things.

for (int Column = 0; Column < 8; Column++)

for (int Row = 0; Row < 8; Row++)

{

//When Things are Soldiers.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 1)

{

//Determine int

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//When int is Gray.

if (a == Color.Gray)

{

try

{

//When Solders ate current location differs add move.

try

{

if (SolderesOnTable[So1].Row != Row || SolderesOnTable[So1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construct Soder Gray.

SolderesOnTable[So1] = new DrawSoldier( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, So1);

//Increase So1.

So1++;

if (So1 > SodierMidle)

{

SodierMidle++;

SodierHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}

//When int is Brown

else

{

try

{

//When Solders ate current location differs add move.

try

{

if (SolderesOnTable[So2].Row != Row ||

SolderesOnTable[So2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construct Soldeir Brown.

SolderesOnTable[So2] = new DrawSoldier( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, So2);

//Increase So2.

So2++;

if (So2 > SodierHigh)

SodierHigh++;

}

catch (Exception t)

{

Log(t);

}

}

}

else //For Elephant Objects.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 2)

{

//Initiate Local Variables.

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//If Gray Elepahnt

if (a == Color.Gray)

{

try

{

try

{

//Calculation of Movment Number.

if (ElephantOnTable[El1].Row != Row ||

ElephantOnTable[El1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of Draw Object.

ElephantOnTable[El1] = new DrawElefant( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, El1);

//Increament of Gray Index.

El1++;

//If New Object Increament Gray Objects.

if (El1 > ElefantMidle)

{

ElefantMidle++;

ElefantHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}

else//For Brown Elephant .Objects

{

try

{

try

{

//Calculation of Movments Numbers.

if (ElephantOnTable[El2].Row != Row ||

ElephantOnTable[El2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of Draw Brown Elephant Object.

ElephantOnTable[El2] = new DrawElefant( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, El2);

//Increament of Index.

El2++;

//When New Brown Elephant Object Increament of Index.

if (El2 > ElefantHigh)

ElefantHigh++;

}

catch (Exception t)

{

Log(t);

}

}

}

else//For Hourse Objects.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 3)

{

//Initiate Local Varibale and int.

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//If Gray Hourse.

if (a == Color.Gray)

{

try

{

try

{

//Calculation of Movments Number.

if (HoursesOnTable[Ho1].Row != Row ||

HoursesOnTable[Ho1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of Draw Brown Hourse.

HoursesOnTable[Ho1] = new DrawHourse( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, Ho1);

//Increament of Index.

Ho1++;

//when There is New Gray Hourse Increase.

if (Ho1 > HourseMidle)

{

HourseMidle++;

HourseHight++;

}

}

catch (Exception t)

{

Log(t);

}

}//For Brown Hourses.

else

{

try

{

try

{

//Calculation of Movments Number.

if (HoursesOnTable[Ho2].Row != Row |

HoursesOnTable[Ho2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of Draw Brown Hourse.

HoursesOnTable[Ho2] = new DrawHourse( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, Ho2);

//Increament of Index.

Ho2++;

//When New Brown Hourse Exist Exist Index.

if (Ho2 > HourseHight)

HourseHight++;

}

catch (Exception t)

{

Log(t);

}

}

}

else//For Castles Objects.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 4)

{

//Initiate of Local Variables.

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//For Gray int.

if (a == Color.Gray)

{

try

{

try

{

//Calculation of Movments Number.

if (CastlesOnTable[Br1].Row != Row ||

CastlesOnTable[Br1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of New Draw Gray Castles.

CastlesOnTable[Br1] = new DrawCastle( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, Br1);

//Increamnt of Index.

Br1++;

//When New Gray Briges Increamnt Max Index.

if (Br1 > CastleMidle)

{

CastleMidle++;

CastleHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}//For Brown Castles.

else

{

try

{

try

{

//Calculation of Movments Number.

if (CastlesOnTable[Br2].Row != Row ||

CastlesOnTable[Br2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction Draw of New Brown Castles.

CastlesOnTable[Br2] = new DrawCastle( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, Br2);

//Increament of Index.

Br2++;

//wehn Brown New Castles Detected Increament Max Index.

if (Br2 > CastleHigh)

CastleHigh++;

}

catch (Exception t)

{

Log(t);

}

}

}

else//For Minister Objects.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 5)

{

//Initiate Local int Varibales.

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//For Gray ints.

if (a == Color.Gray)

{

try

{

try

{

//Clculationb of Movments Number.

if (MinisterOnTable[Mi1].Row != Row ||

MinisterOnTable[Mi1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//construction of new draw Gray Minster.

MinisterOnTable[Mi1] = new DrawMinister( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, Mi1);

//Increament of Index.

Mi1++;

//Wehn New Gray Minster Detected Increament Max Indexes.

if (Mi1 > MinisterMidle)

{

MinisterMidle++;

MinisterHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}//For Brown ints.

else

{

try

{

try

{

//Calculation of Movments Number.

if (MinisterOnTable[Mi2].Row != Row ||

MinisterOnTable[Mi2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of New Draw Brown Minster.

MinisterOnTable[Mi2] = new DrawMinister( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, Mi2);

//Increament Index.

Mi2++;

//When New Brown Minister Detected Increament Max Index.

if (Mi2 > MinisterHigh)

MinisterHigh++;

}

catch (Exception t)

{

Log(t);

}

}

}

else//for King Objects.

if (System.Math.Abs(this.TableList[index][Row, Column]) == 6)

{

//Initiate Of int.

Color a;

if (this.TableList[index][Row, Column] > 0)

a = Color.Gray;

else

a = Color.Brown;

//int consideration.

if (a == Color.Gray)

{

try

{

try

{

//Calculation of Movments Number.

if (KingOnTable[Ki1].Row != Row ||

KingOnTable[Ki1].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of New Draw Gray King.

KingOnTable[Ki1] = new DrawKing( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], 1, false, Ki1);

//Increament of Index.

Ki1++;

//when New Draw Object Detected Increament Max Index.

if (Ki1 > KingMidle)

{

KingMidle++;

KingHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}//For Brown King int

else

{

try

{

try

{

//Calculation of Movment Number.

if (KingOnTable[Ki2].Row != Row ||

KingOnTable[Ki2].Column != Column)

Move++;

}

catch (Exception t) { Log(t); }

//Construction of New Draw King Brown Object.

KingOnTable[Ki2] = new DrawKing( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, this.TableList[index], -1, false, Ki2);

//Increament of Index.

Ki2++;

//When New Object Detected Increament Of Brown King Max Index.

if (Ki2 > KingHigh)

KingHigh++;

}

catch (Exception t)

{

Log(t);

}

}

}

}

//Make Empty Remaining.

for (int i = So1; i < SodierMidle; i++)

SolderesOnTable[i] = null;

for (int i = So2; i < SodierHigh; i++)

SolderesOnTable[i] = null;

for (int i = El1; i < ElefantMidle; i++)

ElephantOnTable[i] = null;

for (int i = El2; i < ElefantHigh; i++)

ElephantOnTable[i] = null;

for (int i = Ho1; i < HourseMidle; i++)

HoursesOnTable[i] = null;

for (int i = Ho2; i < HourseHight; i++)

HoursesOnTable[i] = null;

for (int i = Br1; i < CastleMidle; i++)

CastlesOnTable[i] = null;

for (int i = Br2; i < CastleHigh; i++)

CastlesOnTable[i] = null;

for (int i = Mi1; i < MinisterMidle; i++)

MinisterOnTable[i] = null;

for (int i = Mi2; i < MinisterHigh; i++)

MinisterOnTable[i] = null;

for (int i = Ki1; i < KingMidle; i++)

KingOnTable[i] = null;

for (int i = Ki2; i < KingHigh; i++)

KingOnTable[i] = null;

}

}

catch (Exception t)

{

Log(t);

}

}

//Huristic of Check Method.

public int[,] HuristicCheck(List<AllDraw> A, Color a, int ij, ref double Less, int Order)

{

//Inititae Local Varibales.

int i = 0, j = 0;

int[,] Table = new int[8, 8];

bool Act = false;

int ii = ij;

bool AAAA = false;

ChessRules AA = null;

//If List Exist.

if (A.Count > 0)

{

//Fo All Soldeirs.

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

{

//Calculate Thinking Operation of Current Soldier.

for (int k = 0; k < AllDraw.SodierMovments; k++)

for (j = 0; SolderesOnTable !=null && SolderesOnTable[i] != null && SolderesOnTable[i].SoldierThinking[k] != null && j < SolderesOnTable[i].SoldierThinking[k].TableListSolder.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (SolderesOnTable[i].SoldierThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = SolderesOnTable[i].SoldierThinking[k].TableListSolder[j];

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, TableS, Order, SolderesOnTable[i].SoldierThinking[k].Row, SolderesOnTable[i].SoldierThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

else

{

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.SolderesOnTable[i].Row, (int)APredict.SolderesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

else

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

} Act = true;

Less = SolderesOnTable[i].SoldierThinking[k].HuristicListSolder[j][0] + SolderesOnTable[i].SoldierThinking[k].HuristicListSolder[j][1] + SolderesOnTable[i].SoldierThinking[k].HuristicListSolder[j][2] + SolderesOnTable[i].SoldierThinking[k].HuristicListSolder[j][3];

continue;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.SolderesOnTable[i].Row, (int)APredict.SolderesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

ChessRules AAA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AAA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AAA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AAA.CheckGray)

{

Table = null;

continue;

}

}

}

//Initaiet Local Varibale and Syntax and Table Found.

RW = i;

CL = k;

Ki = 1;

Act = true;

AllDraw.LastRow = SolderesOnTable[i].SoldierThinking[k].Row;

AllDraw.LastColumn = SolderesOnTable[i].SoldierThinking[k].Column;

Less = SolderesOnTable[i].SoldierThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

Table = SolderesOnTable[i].SoldierThinking[k].TableListSolder[j];

Object O = new Object();

lock (O)

{

ThingsConverter.ActOfClickEqualTow = true;

}

SolderesOnTable[i].ConvertOperation(SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1], a, SolderesOnTable[i].SoldierThinking[k].TableListSolder[j], Order, false, i);

int Sign = 1;

if (a == Color.Brown)

Sign = -1;

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

Table[SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1]] = 5 \* Sign;

else if (SolderesOnTable[i].ConvertedToCastle)

Table[SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1]] = 4 \* Sign;

else if (SolderesOnTable[i].ConvertedToHourse)

Table[SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1]] = 3 \* Sign;

else if (SolderesOnTable[i].ConvertedToElefant)

Table[SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1]] = 2 \* Sign;

TableList.Clear();

TableList.Add(Table);

SetRowColumn(0);

TableList.Clear();

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

//Calculate Thinking Operation of Current Elephant.

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

{

for (int k = 0; k < AllDraw.ElefantMovments; k++)

for (j = 0; ElephantOnTable != null && ElephantOnTable[i] != null && ElephantOnTable[i].ElefantThinking[k] != null && j < ElephantOnTable[i].ElefantThinking[k].TableListElefant.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (ElephantOnTable[i].ElefantThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = ElephantOnTable[i].ElefantThinking[k].TableListElefant[j];

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 2, TableS, Order, ElephantOnTable[i].ElefantThinking[k].Row, ElephantOnTable[i].ElefantThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

//}

else

{

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.ElephantOnTable[i].Row, (int)APredict.ElephantOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

else

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

RW = i;

CL = k;

Ki = 1;

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(ii);

Table = APredict.InitiatePerdictCheck((int)APredict.ElephantOnTable[i].Row, (int)APredict.ElephantOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

//Initaiet Local Varibale and Syntax and Table Found.

AllDraw.LastRow = ElephantOnTable[i].ElefantThinking[k].Row;

AllDraw.LastColumn = ElephantOnTable[i].ElefantThinking[k].Column;

RW = i;

CL = k;

Ki = 2;

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

Table = ElephantOnTable[i].ElefantThinking[k].TableListElefant[j];

}

}

catch (Exception t)

{

Log(t);

}

}

}

//Calculate Thinking Operation of Current Hourse.

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

{

for (int k = 0; k < AllDraw.HourseMovments; k++)

for (j = 0; HoursesOnTable != null && HoursesOnTable[i] != null && HoursesOnTable[i].HourseThinking[k] != null && j < HoursesOnTable[i].HourseThinking[k].TableListHourse.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (HoursesOnTable[i].HourseThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = HoursesOnTable[i].HourseThinking[k].TableListHourse[j];

{

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 3, TableS, Order, HoursesOnTable[i].HourseThinking[k].Row, HoursesOnTable[i].HourseThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

else

{

}

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.HoursesOnTable[i].Row, (int)APredict.HoursesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.HoursesOnTable[i].Row, (int)APredict.HoursesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

else

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

} RW = i;

CL = k;

Ki = 1;

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

continue;

}

}

}

//Initaiet Local Varibale and Syntax and Table Found.

AllDraw.LastRow = HoursesOnTable[i].HourseThinking[k].Row;

AllDraw.LastColumn = HoursesOnTable[i].HourseThinking[k].Column;

RW = i;

CL = k;

Ki = 3;

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

Table = HoursesOnTable[i].HourseThinking[k].TableListHourse[j];

}

}

catch (Exception t)

{

Log(t);

}

}

}

//Calculate Thinking Operation of Current Castles.

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

{

for (int k = 0; k < AllDraw.CastleMovments; k++)

for (j = 0; CastlesOnTable != null && CastlesOnTable[i] != null && CastlesOnTable[i].CastleThinking[k] != null && j < CastlesOnTable[i].CastleThinking[k].TableListCastle.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (CastlesOnTable[i].CastleThinking[k].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (CastlesOnTable[i].CastleThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = CastlesOnTable[i].CastleThinking[k].TableListCastle[j];

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 4, TableS, Order, CastlesOnTable[i].CastleThinking[k].Row, CastlesOnTable[i].CastleThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

else

{

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.CastlesOnTable[i].Row, (int)APredict.CastlesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)CastlesOnTable[i].Row, (int)CastlesOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

else

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

} RW = i;

CL = k;

Ki = 1;

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

continue;

}

}

}

//Initaiet Local Varibale and Syntax and Table Found.

AllDraw.LastRow = CastlesOnTable[i].CastleThinking[k].Row;

AllDraw.LastColumn = CastlesOnTable[i].CastleThinking[k].Column;

RW = i;

CL = k;

Ki = 4;

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

Table = CastlesOnTable[i].CastleThinking[k].TableListCastle[j];

}

}

catch (Exception t)

{

Log(t);

}

}

}

//Calculate Thinking Operation of Current Minister.

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

{

for (int k = 0; k < AllDraw.MinisterMovments; k++)

for (j = 0; MinisterOnTable != null && MinisterOnTable[i] != null && MinisterOnTable[i].MinisterThinking[k] != null && j < MinisterOnTable[i].MinisterThinking[k].TableListMinister.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (MinisterOnTable[i].MinisterThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = MinisterOnTable[i].MinisterThinking[k].TableListMinister[j];

{

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 5, TableS, Order, MinisterOnTable[i].MinisterThinking[k].Row, MinisterOnTable[i].MinisterThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

else

{

}

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.MinisterOnTable[i].Row, (int)APredict.MinisterOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.MinisterOnTable[i].Row, (int)APredict.MinisterOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

//Initaiet Local Varibale and Syntax and Table Found.

AllDraw.LastRow = MinisterOnTable[i].MinisterThinking[k].Row;

AllDraw.LastColumn = MinisterOnTable[i].MinisterThinking[k].Column;

RW = i;

CL = k;

Ki = 5;

Act = true;

Less = MinisterOnTable[i].MinisterThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

Table = MinisterOnTable[i].MinisterThinking[k].TableListMinister[j];

}

}

catch (Exception t)

{

Log(t);

}

}

}

//Calculate Thinking Operation of Current King.

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

{

for (int k = 0; k < AllDraw.KingMovments; k++)

for (j = 0; KingOnTable != null && KingOnTable[i] != null && KingOnTable[i].KingThinking[k] != null && j < KingOnTable[i].KingThinking[k].TableListKing.Count; j++)

{

try

{

//If there is Penalty Situation Continue.

if (AllDraw.OrderPlate == Order)

if (KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsPenaltyAction() == 0)

{

Less = -200000000;

continue;

}

//For Higher Huristic Values.

if (AllDraw.OrderPlate == Order)

if (KingOnTable[i].KingThinking[k].ReturnHuristic(i, j, Order,AAAA) >= Less)

{

//Initiate Table of Current Object.

int[,] TableS = KingOnTable[i].KingThinking[k].TableListKing[j];

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 6, TableS, Order, KingOnTable[i].KingThinking[k].Row, KingOnTable[i].KingThinking[k].Column);

//Achamaz Check CheckMate of Current Table.

if (AA.ObjectDangourKingMove(Order, TableS, false) && !AllDraw.NoTableFound)

{

//If Order is Gray.

if (Order == 1)

{

if (AA.CheckGrayObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

else//If Order is Brown.

{

if (AA.CheckBrownObjectDangour && AllDraw.AStarGreadyFirstSearch)

continue;

}

}

else

{

}

}

if (Order == 1)//If Order is Gray.

{

//If CheckObjectDangour Occured and AStarGreedyGreedy Huristic Not Exist.

if (AA.CheckGrayObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.KingOnTable[i].Row, (int)APredict.KingOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

}

}

else

{

if (AA.CheckBrownObjectDangour && !AllDraw.AStarGreadyFirstSearch)

{

//Prdeict Huristic.

Color B;

if (a == Color.Gray)

B = Color.Brown;

else

B = Color.Gray;

APredict.TableList.Clear();

APredict.TableList.Add(TableS);

APredict.SetRowColumn(0);

Table = APredict.InitiatePerdictCheck((int)APredict.KingOnTable[i].Row, (int)APredict.KingOnTable[i].Column, B, TableS, Order, false);

if (Table == null)

continue;

else

{

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, AllDraw.OrderPlate, -1, -1);

AA.Check(Table, AllDraw.OrderPlate);

if (AllDraw.OrderPlate == 1 && AA.CheckGray)

{

Table = null;

continue;

}

if (AllDraw.OrderPlate == -1 && AA.CheckGray)

{

Table = null;

continue;

}

RW = i;

CL = k;

Ki = 1;

Act = true;

Less = KingOnTable[i].KingThinking[k].ReturnHuristic(i, j, Order,AAAA); ;

continue;

}

}

}

AllDraw.LastRow = KingOnTable[i].KingThinking[k].Row;

AllDraw.LastColumn = KingOnTable[i].KingThinking[k].Column;

RW = i;

CL = k;

Ki = 6;

Act = true;

Less = KingOnTable[i].KingThinking[k].ReturnHuristic(i, j, Order,AAAA);

Table = KingOnTable[i].KingThinking[k].TableListKing[j];

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

//If There is A Movments Return Table.

if (Act)

return Table;

//What Kind Of Table.

return Table;

}

//Iniatite Prediction Method.

public int[,] InitiatePerdictCheck(int ii, int jj, Color a, int[,] Table, int Order, bool TB)

{

//Initaite Local and Global Variables.

ChessRules AA = null;

bool Dummy = ThinkingChess.NotSolvedKingDanger;

int[,] TablInit = new int[8, 8];

int[,] TablInitOne = new int[8, 8];

int[,] TablInitCheck = new int[8, 8];

int Current = ChessRules.CurrentOrder;

OrderDummy = Order;

A.Clear();

TableList.Clear();

Object O = new Object();

lock (O)

{

ThinkingChess.NotSolvedKingDanger = false;

}

LoopHuristicIndex = 0;

//Clone a Copy.

for (int iii = 0; iii < 8; iii++)

for (int jjj = 0; jjj < 8; jjj++)

{

TablInitOne[iii, jjj] = Table[iii, jjj];

}

//Clone A Copy.

for (int iii = 0; iii < 8; iii++)

for (int jjj = 0; jjj < 8; jjj++)

{

TablInitCheck[iii, jjj] = TablInitOne[iii, jjj];

}

AA = new ChessRules( CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, TablInitCheck, Order, -1, -1);

//Check Consideration.

if (AA.Check(TablInitCheck, Order))

{

if (OrderDummy == 1)

{

if (AA.CheckGray)

return null;

}

else

{

if (AA.CheckGray)

return null;

}

}

//For Tow Times

for (int i = 0; i < 2; i++)

{

if (i != 0)

this.SetRowColumn(i);

if (Order == 1)

{

//THIS.SetBoxText("\r\nChess Predict Thinking AStarGreedyGreedy " + (i + 1).ToString() + " By Bob!");

//THIS.RefreshBoxText();

}

else

{

//THIS.SetBoxText("\r\nChess Predict Thinking AStarGreedyGreedy " + (i + 1).ToString() + " By Alice!");

//THIS.RefreshBoxText();

}

//Gray Order.

if (Order == 1)

a = Color.Gray;

else//Brown Order.

a = Color.Brown;

//Initiate Local Variables and Take a Randomly Soldiers.

int In = 0;

int iiii = 0;

do

{

if (Order == 1)

In = (new System.Random()).Next(0, 8);

else

In = (new System.Random()).Next(8, 16);

iiii++;

} while (SolderesOnTable[In] == null || iiii < 16);

//For Sixteen Times Take a Look At Thinking.

if (iiii < 16)

this.InitiateForEveryKindThingHome(new AllDraw(Order, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged), (int)(int)SolderesOnTable[In].Row, (int)(int)SolderesOnTable[In].Column, a, TablInit, Order, false, In);

else

this.InitiateForEveryKindThingHome(new AllDraw(Order, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged), (int)1, 2, a, TablInit, Order, false, In);

//Initiate Local Variables.

double Less = -100000;

int[,] Tab = null;

//List Not Empty.

if (A.Count > 0)

{

//Gray Order.

if (Order == 1)

{

//THIS.SetBoxText("\r\nHuristic Check Considerasion Movements AStarGreedyGreedy " + i.ToString() + " By Bob!");

//THIS.RefreshBoxText();

}

else//Brown Order.

{

//THIS.SetBoxText("\r\nHuristic Check Considerasion Movements AStarGreedyGreedy " + i.ToString() + " By Alice!");

//THIS.RefreshBoxText();

}

//Huristic Foundation of Table.

Tab = HuristicCheck(A, a, i, ref Less, Order);

}

//Table is Not Found.

if (Tab == null)

{

//Initiate Not Found Variables.

Object O3 = new Object();

lock (O3)

{

ThinkingChess.NotSolvedKingDanger = Dummy;

ChessRules.CurrentOrder = Current;

}

Order = OrderDummy;

return null;

}

//Table Foundation.

if (Tab != null)

{

//Clone a Copy.

for (int iii = 0; iii < 8; iii++)

for (int jjj = 0; jjj < 8; jjj++)

{

TablInit[iii, jjj] = Tab[iii, jjj];

}

//Initiate Local Varibales.

TableList.Add(TablInit);

ClList.Add(CL);

RWList.Add(RW);

KiList.Add(Ki);

//Order = Order \* -1;

//ChessRules.CurrentOrder = Order;

AStarGreedyGreedy++;

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

}

else//Table Not Found.

{

Object O1 = new Object();

lock (O1)

{

ThinkingChess.NotSolvedKingDanger = Dummy;

ChessRules.CurrentOrder = Current;

}

Order = OrderDummy;

return null;

}

}

//Initiat Local Variables.

Object O4 = new Object();

lock (O4)

{

ThinkingChess.NotSolvedKingDanger = Dummy;

ChessRules.CurrentOrder = Current;

}

Order = OrderDummy;

return TablInitOne;

}

//Enemy Non Used Check Predict Found.

public bool InitiatePerdictCheckEnemy(int ii, int jj, Color a, int[,] Table, int Order, bool TB)

{

//Iniatite Local and Global Variables.

int Current = ChessRules.CurrentOrder;

int OrderDummy = Order;

A.Clear();

TableList.Clear();

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

bool Dummy = ThinkingChess.NotSolvedKingDanger;

Object O = new Object();

lock (O)

{

ThinkingChess.NotSolvedKingDanger = false;

}

LoopHuristicIndex = 0;

//For One Time.

for (int i = 0; i < 1; i++)

{

//Initiate Local Variables.

int[,] TablInit = new int[8, 8];

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int In = 0;

//Found of a Randomly Soldeir.

do

{

if (Order == 1)

In = (new System.Random()).Next(0, 8);

else

In = (new System.Random()).Next(8, 16);

} while (SolderesOnTable[In] == null);

//Intiatation of Thinking.

this.InitiateForEveryKindThingHome(new AllDraw(Order, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged), (int)SolderesOnTable[In].Row, (int)SolderesOnTable[In].Column, a, Table, Order, false, In);

//Iniatite Local Variables.

double Less = 0;

int[,] Tab = null;

//When Thinking Found Take Huristic.

if (A.Count > 0)

Tab = HuristicCheck(A, a, i, ref Less, Order);

//Table Not Foundation Condition.

if (Tab == null)

{

//Initaiation of Not Founding Variables.

Object O6 = new Object();

lock (O6)

{

ThinkingChess.NotSolvedKingDanger = Dummy;

ChessRules.CurrentOrder = Current;

}

Order = OrderDummy;

return false;

}

//Table Reapetedly Consideration.

if (ThinkingChess.ExistTableInList(Tab, TableListAction, 0))

{

//Remove Whle is Repeatedly.

while (ThinkingChess.ExistTableInList(Tab, TableListAction, 0))

{

TableListAction.RemoveAt(LoopHuristicIndex);

}

//Genetic Algorithm Construction.

ChessGeneticAlgorithm R = (new ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

//Found Of Genetic Algorithm Table Method.

Tab = R.GenerateTable(TableListAction, LoopHuristicIndex, Order);

}

//Table Foundation Condition.

if (Tab != null)

{

//Clone a Copy.

for (int iii = 0; iii < 8; iii++)

for (int jjj = 0; jjj < 8; jjj++)

{

TablInit[iii, jjj] = Tab[iii, jjj];

}

//Iniatiet Local Variables.

Table = new int[8, 8];

//Clone a Copy.

for (int iii = 0; iii < 8; iii++)

for (int jjj = 0; jjj < 8; jjj++)

{

Table[iii, jjj] = TablInit[iii, jjj];

}

//Initaiation of Local and Global Variables.

TableList.Add(TablInit);

ClList.Add(CL);

RWList.Add(RW);

KiList.Add(Ki);

Order = Order \* -1;

ChessRules.CurrentOrder = Order;

AStarGreedyGreedy++;

}

}

//Iniatiation of Local and Global Variables.

Object O5 = new Object();

lock (O5)

{

ThinkingChess.NotSolvedKingDanger = Dummy;

ChessRules.CurrentOrder = Current;

}

Order = OrderDummy;

return true;

}

}

}

//End of Documentation.