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

\* Initiate and Decision making class.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\* Call Of Constructor From Constructor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* The Storing AllDraw Object in Self Constructor Caused Stack Overflow\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Link List Of Storing String Caused A Stack Over Flow\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Wait For Finished Current AStarGreedy Caused To Long Time\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Need To Heuristic (Arvin) Function(s) to Manage Cell in Form1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* First Scanning Movements of Things Anomaly\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* In Current Version of Heuristic Table Doesn’t Reached(Zero)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* In Current Version InitiateForEveryThisngsHome Dosn't Work\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* In This Version Thinking Taking A LotofTime(AStarGreedyt Array Tree)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Work In AStarGreedys. But Scanning Dosen’t Works\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Adding Clone Caused To Stack Overflow\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Clone Caused To StackOverFlow\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Row And Column Become Zero in Virtualization\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Initiate Error\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Seems To Be Logical Drawing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* int Suddenly Changing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* AllDraw Object Sub Objects List When Return from local Scope Become Zero.\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Huristic Dosn't Work\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* int Order Of Visualization Changed Suddenly\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* int Changes with no movement\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Table Not Gate (Inversion of Table List) Doesn’t help to do Normally\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Literally Errors Correction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* From Arrangements of Things Reaches Suddenly Things Location OccuRS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* The Arrangements is Logical\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* The int changes and the arrangements changes are not clearly obvious\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* int Changes Solved. no movements\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)(-+)

\* Things movements Anomally\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Chess Rules Anomally\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Function Not Work\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Work But the Table is Empty\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Table is Not Empty But the Movement is Not Logical\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Clear Dirty Part.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Need to Restricted Approval. Taking a lot of time Thinking Computation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* No movements In Virtualization\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Chess Rules Abnormal thinking movements. No movement greater than 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Problem For Drawing of Thinking Things\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Constant Result\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* One movements Right .Heuristic Remaining Constant Results\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Constant Heuristic Result\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Need To Add A Heuristic Useful Another\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Function Does’ Work Allis suddenly Become Zero that Previously Working\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* No Movement Greater than one order in Computer 'Alice'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Tow movements in Computer 'Alice' Of two Different Order int\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Not Work Greater than 3 Length Count of A\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'They Don't Really Take care about us'. Misleading in Heuristic King Supported\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Non Order Movments\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Misleading at Stage three. no illegal movement greater than three\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Thinking Order Misleading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Hit Mechanism Malfunctional\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Tow movements At One 'Alice' Order time\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Computer By Computer 'Alice' by 'Bob' Caused to Loop Heuristic.\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Learning Automata of Quantum also leads to re loop heuristic\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Heuristic Learning Automata 'Alice' By 'Bob' Leads to Re loop\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Heuristic Things Loop 'Alice' By 'Bob'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Self 'Check' Detection Failure By 'Alice'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'Penalty' Value Of All Become zero althouth the one should be non Penalty\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)

\* Clone Dosn't Copy All Content of AllDraw Dummy\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* CheckRemovable By Self King Solved.Penalty Action Misleading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(-\*)QC-OK.

\* 'Check' Detection Failure\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Mechanisam Of Order in Predict Failed.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'Alice' King Virtualization or Table Content of King Misleading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* With The All Things Huristic Signing Mechnisam Some Movments become null Table.\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* AStarGreedy First Search Not Working. Misleading MalFunction Virtualization.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* AStarGreedy First Table is Null at Bob Order.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* AStarGreedy First SetVirtualization and Table Misleading By Alice.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* No Reason Logically For MalFunction Refrigtz.Timer AStarGreedy First Dynamic Refrigtz.Timer.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* AStarGreedyt Thinking Taking a lot of time.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*([+]

\* AStarGreedy First Not Work.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[+]

\* AStarGreedy First Not Work.Refrigtz.Timer Stop At Greater than 2,3,4,5,6,7 Movments.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[+]

\* No Reason For MalFunction of AStarGreedytNotFoundHuristicAStarGreedyt.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[+]

\* Problem Solved.No Reason to NullExeption of AStarGreedytHuristic Algorithm.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[-\*]QC-OK.

\* Function Evaluation Disabled .At Initiate AStarGreedytGenetic Found Sysntax.\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[\*]

\* Index Was Out Of Range Exeption Was Not Handled.Colud Not Be Handle.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* No Logical Mechanism To Reconstructe Current AllDraw Objects.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* AStarGreedy First Sysntax is legal and The table is constant table.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Table Content Empty. No Syntax Exist.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Game Begin From First When the Soldiers Move Ordinary Complete in AStarGreedy First\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* New Instatnt Of Program Cuase to Begin Fron First.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* No Logically Reason For New Game Of Program. New Instatnt Not Detected.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Internal New Instatnt Of FormeRefregitz is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* AStarGreedy First CC Changes to CC Normal Game.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Game CC UnContoroled.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* MalFunction of Syntax and Movments.By Alice and Bob.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Threading Solved! The OutOfRangeIndex Not Work.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[-+]

\* Vituallization error!No Best Matches between Truth of table content and irtualization\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[+]

\* Dynamic Programming for Stroring ADraw THISDummy Adraw Value MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Order is Constant in Dynamic Programming.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Table MalFunction at Dynamic Programming.At Step 3.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Some Movments are MalFuncational at Dynamic Programming.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Huristic Overlay Tow Part of ADraw and StoreADraw Sections at Different levels Tab Cal.0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Not to be needing again calculation. MalFunction is depend of tow part.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* BackWard Loos of Things AllDraw Mechnisam.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Some Dynamic Programming MalFunction Movments.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Syntax and Forward and Backward Movments Syntax is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Database and Virtualization Forward and Backward MalFunction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Reproduction of Thinfs Missleading.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC-OK.

\* Reproduction of Some Things are MalFunction Movments.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* AStarGreedy Count of Dynamic Programming Misleadig.AStarGreedy Operation Count Mal Function.\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Huristic By Alice is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* ObjectDanger Identification By Alice is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Check Identification By Alice is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Check Recognized But CheckMate Not Recognized!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\_+)

\* Penalty Regard Mechanism Misleading.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Inhereted LearningAtamata Caused to Shared Parent Allocated Variable.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* 'Check' By 'Alice' Not Removed Unreasonably.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* AStarGreedyt Huristic Found MalFunction at Check Alice.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Sortments of ADRAW and Construction is MalFunction at AStarGreedy Dynamic Programming.\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Huristic AStarGreedy First were Worked Out Unreasonably such Situation(Golden Sword Magic).\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Converted 'King' of 'Alice' to 'Elephant' UnReasonably.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'Long Game' ; But MalFunction of Game.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'Always' in Current game is 'Bob'.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Current Table of ADRAW is Correct Table But the Game is MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Move of Current Table AStarGreedy First Huristic found ;found an ovelay in 'Bob' and 'Alice'\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* Current Table in High Level Become Null and prevent of 'LongGame' Strategy.\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+)

\* 'LongGame' Become short Undetectably Unreasonably;Clear Store Non Detectably.\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* All Draw AStarGreedy First section some movments have not been accurred considerably.\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* 'Long Game' Breaks Suddendly without Monitor Caused.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Overlay Some Movments of 'Long Game' Breaked.Caused Probability to break.\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* SomeTimes All Situation of Current Games Become Cleared and No Table Founded.\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Gray Soldeir is Only Movmnets and Converts in Huristic and No Move are detectable.\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* DEEPLY Recursive Tree of Second Version Become in Some Null At Hurristic Finsished.\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* AStarGreedy Huristic Content is Zero. No Calculation of AStarGreedy Huristic Calculation.\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* MalFunction of Dep Huristic Person and MalFunction Movments of CC AStarGreedy Huristic\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Mal Function of Reconstruction of AStarGreedy Objects In Initiate AStarGreedy First.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Hurisic Operantional Have Mal Function Behaviour.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Table Zero of AStarGreedy First Huristic Mal Function.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* AStarGreedy First Initiate Method Result Object Content Mal Function.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Table Nopt Found Of AStarGreedy First Huristic.Mal Function of Initiate and Huristic.\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Table Foundation Successfule. Traversaling of All Tree Not Successfule.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<+>

\* Table Some Movments Intiaiazation Mal Function.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* BackWard Max Check CheckMate Mechanism For Best Huristic is Unknown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{+}

\* Minister After Calculation AStarGreedyHuristic At AStarGreedyHuristic becomes Null.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK

\* All Objects Possible Movments Not calculating During AStarGreedytSerach Method.\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC\_OK

\* Mechanisam olf AStarGreedytHuristic and Hurisistic is QC-Ok. But Table foundation Illegal.0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC-OK

\* Full Game Indexing Parameters Misleading UnLogically.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_OK

\* Index out of Range Unlogically at Full Game Soldier Order Brown.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC\_OK

\* Execution make zero table but trace make valid table.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}Qc-OK.

\* Virtualization need to more hardware capabilities gone to malfunction virtualization.\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-BAD.

\* MalFunction on AllDraw Hadeling of Draw Midle Target Motion Graphics.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-BAD.

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(+:Sum(63))

\* 1394/12/19\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*:Sum(4))

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(-:sum(2)) (\_:Sum(0)):2:(+:Sum(3)) (-:Sum(1)) (\*:Sum(2)) 3: (+:Sum(4)) (\*:Sum(1)) 4:(+:Sum(6)) 5:(+:Sum(2)) (-:Sum(1)) 6:(+:Sum(6)) (\*:Sum(2)) 7.(+:Sum(2)) (\*:Sum(1)) 8.(+:Sum(1)) 9.(+:Sum(4)) (\*:Sum(1)) (-:Sum(1)) 10.(+:Sum(4)) (\*:Sum(2)) 11.(+:Sum(4)) 12.(+:Sum(2)) (\*:Sum(2)) 13.(+:Sum(4)) 14.(+:Sum(2)) (\*:Sum(1)) 15.(+:Sum(6)) 16.(+:Sum(2)) 17.(QC-OK.:Sum(13))

\*/

using System;

using System.Collections.Generic;

using System.Text;

using System.Drawing;

using System.Threading;

using System.Threading.Tasks;

using System.IO;

using System.Diagnostics;

namespace RefrigtzDLL

{

[Serializable]

public class AllDraw

{

StackFrame callStack = new StackFrame(1, true);

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

public int OrderP = 0;

public static int DepthIterative = 0;

int PerceptionCount = 0;

public String OutPutAction = "";

public static String OutPut = "";

public static String ActionString = "";

public static bool ActionStringReady = false;

//static variable to be Initiate

List<int[]> ValuableSelfSupported = new List<int[]>();

public static bool RegardOccurred = false;

public static int SuppportCountStaticGray = 0;

public static int SuppportCountStaticBrown = 0;

int CurrentAStarGredyMax = 0;

public static int TaskBegin = 0;

public static int TaskEnd = 0;

public static String Root = System.IO.Path.GetDirectoryName(Environment.GetCommandLineArgs()[0]);

public static int OrderPlate = 1;

public static bool Blitz = false;

public static int ConvertedKind = -2;

public static bool ConvertWait = true;

public static bool Stockfish = false;

public static bool Person = true;

public static bool THISSecradioButtonGrayOrderChecked = false;

public static bool THISSecradioButtonBrownOrderChecked = false;

public static String THIScomboBoxMaxLevelText = "";

public static AllDraw THISDummy = null;

public static bool StateCP = false;

public static int LastRow = -1;

public static int LastColumn = -1;

public static int NextRow = -1;

public static int NextColumn = -1;

public static int MovmentsNumber = 0;

public static int MaxAStarGreedyHuristicProgress = 0;

public static bool EndOfGame = false;

//Initiate Variables.

const int ThresholdBlitz = 10000;

const int ThresholdFullGame = 20000;

public bool SetRowColumnFinished = false;

public static int MinThinkingTreeDepth = Int32.MaxValue;

static int MaxDuringLevelThinkingCreation = 0;

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;

int[] Index = { -1, -1, -1, -1, -1, -1 }, jindex = { -1, -1, -1, -1, -1, -1 }, Kind = { -1, -1, -1, -1, -1, -1 };

public bool ArrangmentsChanged = false;

public static double AStarGreedytMaxCount = 0;

public static bool FoundATable = false;

public static double Less = Double.MinValue;

public bool CastlesKing = false;

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

public static int increasedProgress = 0;

public static double CurrentHuristic = Double.MinValue;

public static double SignAttack = 1;

public static double SignObjectDangour = 1;

public static double SignReducedAttacked = -1;

public static double SignSupport = 1;

public static double SignKiller = 1;

public static double SignMovments = 1;

public static double SignDistance = -1;

public static double SignKingSafe = -1;

public static double SignKingDangour = 1;

public static bool DrawTable = true;

public static int[,] TableVeryfy = new int[8, 8];

public static int MaxAStarGreedy = 1;

public static int[,] TableVeryfyConst = new int[8, 8];

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

public static bool NoTableFound = false;

public static bool DynamicAStarGreedytPrograming = false;

public static List<AllDraw> StoreADraw = new List<AllDraw>();

public static List<int> StoreADrawAStarGreedy = new List<int>();

public static bool UseDoubleTime = false;

public static int AStarGreedyiLevelMax;

public static bool AStarGreadyFirstSearch = true;

public static String ImageRoot = AllDraw.Root + "\\Images";

public static String ImagesSubRoot = AllDraw.ImageRoot + "\\Fit\\Small\\";

public static bool RedrawTable = true;

public static String SyntaxToWrite = "";

public static bool SodierConversionOcuured = false;

public static int SodierMovments = 1;

public static int ElefantMovments = 1;

public static int HourseMovments = 1;

public static int CastleMovments = 1;

public static int MinisterMovments = 1;

public static int KingMovments = 1;

/\*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 RW = 0;

int CL = 0;

int Ki = 0;

int RW1 = 0;

int CL1 = 0;

int Ki1 = 0;

double MaxLess1 = 0;

int RW2 = 0;

int CL2 = 0;

int Ki2 = 0;

double MaxLess2 = 0;

int RW3 = 0;

int CL3 = 0;

int Ki3 = 0;

double MaxLess3 = 0;

int RW4 = 0;

int CL4 = 0;

int Ki4 = 0;

double MaxLess4 = 0;

int RW5 = 0;

int CL5 = 0;

int Ki5 = 0;

double MaxLess5 = 0;

int RW6 = 0;

int CL6 = 0;

int Ki6 = 0;

double MaxLess6 = 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<int[,]> TableList = new List<int[,]>();

public int AStarGreedy = 0;

public DrawSoldier[] SolderesOnTable = new DrawSoldier[16];

public DrawElefant[] ElephantOnTable = new DrawElefant[4];

public DrawHourse[] HoursesOnTable = new DrawHourse[4];

public DrawCastle[] CastlesOnTable = new DrawCastle[4];

public DrawMinister[] MinisterOnTable = new DrawMinister[2];

public DrawKing[] KingOnTable = new DrawKing[2];

List<double[]> MaxHuristicAStarGreedytBackWard = new List<double[]>();

const int MaxSoldeirFounded = 2;

const int MaxElephntFounded = 6;

const int MaxHourseFounded = 10;

const int MaxCastlesFounded = 14;

const int MaxMinisterFounded = 18;

const int MaxKingFounded = 22;

[NonSerialized()] public Task ob;

//Making String datastructure to root variable

public AllDraw AStarGreedyString = null;

//Error Handling

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)

{

Object a = new Object();

lock (a)

{

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)

{

Object a = new Object();

lock (a)

{

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 AllDraw(int Order, bool MovementsAStarGreedyHuristicTFou, bool IgnoreSelfObject, bool UsePenaltyRegardMechnisa, bool BestMovment, bool PredictHurist, bool OnlySel, bool AStarGreedyHuris, bool Arrangments)

{

Object a = new Object();

lock (a)

{

MaxHuristicxT = Double.MinValue;

MovementsAStarGreedyHuristicFoundT = MovementsAStarGreedyHuristicTFou;

IgnoreSelfObjectsT = IgnoreSelfObject;

UsePenaltyRegardMechnisamT = UsePenaltyRegardMechnisa;

BestMovmentsT = BestMovment;

PredictHuristicT = PredictHurist;

OnlySelfT = OnlySel;

AStarGreedyHuristicT = AStarGreedyHuris;

ArrangmentsChanged = Arrangments;

FoundATable = false;

CastlesKing = false;

increasedProgress = 0;

CurrentHuristic = Double.MinValue;

DrawTable = false;

TableVeryfy = new int[8, 8];

TableVeryfyConst = new int[8, 8];

TableCurrent.Clear();

NoTableFound = false;

DynamicAStarGreedytPrograming = false;

UseDoubleTime = false;

AStarGreadyFirstSearch = true;

ImageRoot = AllDraw.Root + "\\Images";

ImagesSubRoot = AllDraw.ImageRoot + "\\Fit\\Small\\";

RedrawTable = true;

SodierConversionOcuured = false;

SodierMovments = 1;

ElefantMovments = 1;

HourseMovments = 1;

CastleMovments = 1;

MinisterMovments = 1;

KingMovments = 1;

/\*SodierMidle = 8;

SodierHigh = 16;

ElefantMidle = 2;

ElefantHigh = 4;

HourseMidle = 2;

HourseHight = 4;

CastleMidle = 2;

CastleHigh = 4;

MinisterMidle = 1;

MinisterHigh = 2;

KingMidle = 1;

KingHigh = 2;\*/

APredict = null;

RW = 0;

CL = 0;

Ki = 0;

RW1 = 0;

CL1 = 0;

Ki1 = 0;

MaxLess1 = 0;

RW2 = 0;

CL2 = 0;

Ki2 = 0;

MaxLess2 = 0;

RW3 = 0;

CL3 = 0;

Ki3 = 0;

MaxLess3 = 0;

RW4 = 0;

CL4 = 0;

Ki4 = 0;

MaxLess4 = 0;

RW5 = 0;

CL5 = 0;

Ki5 = 0;

MaxLess5 = 0;

RW6 = 0;

CL6 = 0;

Ki6 = 0;

MaxLess6 = 0;

LoopHuristicIndex = 0;

Move = 0;

MouseClick = 0;

AStarGreedyIndex = new int[20];

AStarGreedy = 0;

SolderesOnTable = null;

ElephantOnTable = null;

HoursesOnTable = null;

CastlesOnTable = null;

MinisterOnTable = null;

KingOnTable = null;

MaxHuristicAStarGreedytBackWard.Clear();

//Initiayte Locally Variables.

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

APredict = new ChessPerdict(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged//, ref th

);

OrderP = Order;

}

}

//Clone Copy Method

public void Clone(AllDraw AA)

{

Object O = new Object();

lock (O)

{

if (AA == null)

{

AA = new AllDraw(OrderP, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged);

AA.TableList.Add(TableList[0]);

}

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

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

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

AA.Tabl[i, j] = Tabl[i, j];

AA.OrderP = OrderP;

AA.PerceptionCount = PerceptionCount;

AA.OutPutAction = OutPutAction;

//static variable to be Initiate

AA.ValuableSelfSupported = new List<int[]>();

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

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

AA.ValuableSelfSupported.Add(ValuableSelfSupported[i]);

AA.CurrentAStarGredyMax = CurrentAStarGredyMax;

for (int i = 0; i < 6; i++)

AA.Index[i] = Index[i];

for (int i = 0; i < 6; i++)

AA.jindex[i] = jindex[i];

for (int i = 0; i < 6; i++)

AA.Kind[i] = Kind[i];

if (AStarGreedyString != null)

AStarGreedyString.Clone(AA.AStarGreedyString);

if (TableList.Count == 1)

SetObjectNumbers(TableList[0]);

MaxHuristicxT = Double.MinValue;

AA.MovementsAStarGreedyHuristicFoundT = MovementsAStarGreedyHuristicFoundT;

AA.IgnoreSelfObjectsT = IgnoreSelfObjectsT;

AA.UsePenaltyRegardMechnisamT = UsePenaltyRegardMechnisamT;

AA.BestMovmentsT = BestMovmentsT;

AA.PredictHuristicT = PredictHuristicT;

AA.OnlySelfT = OnlySelfT;

AA.AStarGreedyHuristicT = AStarGreedyHuristicT;

AA.ArrangmentsChanged = ArrangmentsChanged;

AA.CastlesKing = CastlesKing;

AA.SodierMidle = SodierMidle;

AA.SodierHigh = SodierHigh;

AA.ElefantMidle = ElefantMidle;

AA.ElefantHigh = ElefantHigh;

AA.HourseMidle = HourseMidle;

AA.HourseHight = HourseHight;

AA.CastleMidle = CastleMidle;

AA.CastleHigh = CastleHigh;

AA.MinisterMidle = MinisterMidle;

AA.MinisterHigh = MinisterHigh;

AA.KingMidle = KingMidle;

AA.KingHigh = KingHigh;

//Initiate a new class object and clone a copy.

AA.SolderesOnTable = new DrawSoldier[SodierHigh];

AA.ArrangmentsChanged = ArrangmentsChanged;

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

{

try

{

SolderesOnTable[i].Clone(ref AA.SolderesOnTable[i]);

}

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

}

AA.ElephantOnTable = new DrawElefant[ElefantHigh];

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

{

try

{

ElephantOnTable[i].Clone(ref AA.ElephantOnTable[i]);

}

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

}

AA.HoursesOnTable = new DrawHourse[HourseHight];

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

{

try

{

HoursesOnTable[i].Clone(ref AA.HoursesOnTable[i]);

}

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

}

AA.CastlesOnTable = new DrawCastle[CastleHigh];

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

{

try

{

CastlesOnTable[i].Clone(ref AA.CastlesOnTable[i]);

}

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

}

AA.MinisterOnTable = new DrawMinister[MinisterHigh];

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

{

try

{

MinisterOnTable[i].Clone(ref AA.MinisterOnTable[i]);

}

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

}

AA.KingOnTable = new DrawKing[KingHigh];

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

{

try

{

KingOnTable[i].Clone(ref AA.KingOnTable[i]);

}

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

}

AA.AStarGreedy = AStarGreedy;

if (AA.TableList.Count > 0)

AA.TableList.Clear();

for (int i = 0; i < TableList.Count; i++)

AA.TableList.Add(TableList[i]);

if (AA.TableList.Count > 0)

AA.SetObjectNumbers(AA.TableList[0]);

AA.AStarGreedy = AStarGreedy;

}

}

//aBlanck Constructor

public AllDraw(int Order, bool MovementsAStarGreedyHuristicTFou, bool IgnoreSelfObject, bool UsePenaltyRegardMechnisa, bool BestMovment, bool PredictHurist, bool OnlySel, bool AStarGreedyHuris, bool Arrangments, AllDraw THi)

{

Object a = new Object();

lock (a)

{

OrderP = Order;

MaxHuristicxT = Double.MinValue;

MovementsAStarGreedyHuristicFoundT = MovementsAStarGreedyHuristicTFou;

IgnoreSelfObjectsT = IgnoreSelfObject;

UsePenaltyRegardMechnisamT = UsePenaltyRegardMechnisa;

BestMovmentsT = BestMovment;

PredictHuristicT = PredictHurist;

OnlySelfT = OnlySel;

AStarGreedyHuristicT = AStarGreedyHuris;

if (!Arrangments)

ArrangmentsChanged = Arrangments;

else

ArrangmentsChanged = Arrangments;

AStarGreedytMaxCount = 0;

FoundATable = false;

CastlesKing = false;

increasedProgress = 0;

CurrentHuristic = Double.MinValue;

DrawTable = false;

TableVeryfy = new int[8, 8];

TableVeryfyConst = new int[8, 8];

TableCurrent.Clear();

NoTableFound = false;

DynamicAStarGreedytPrograming = false;

UseDoubleTime = false;

AStarGreadyFirstSearch = true;

ImageRoot = AllDraw.Root + "\\Images";

ImagesSubRoot = AllDraw.ImageRoot + "\\Fit\\Small\\";

RedrawTable = true;

SodierConversionOcuured = false;

SodierMovments = 1;

ElefantMovments = 1;

HourseMovments = 1;

CastleMovments = 1;

MinisterMovments = 1;

KingMovments = 1;

SodierMidle = 8;

SodierHigh = 16;

ElefantMidle = 2;

ElefantHigh = 4;

HourseMidle = 2;

HourseHight = 4;

CastleMidle = 2;

CastleHigh = 4;

MinisterMidle = 1;

MinisterHigh = 2;

KingMidle = 1;

KingHigh = 2;

APredict = null;

RW = 0;

CL = 0;

Ki = 0;

RW1 = 0;

CL1 = 0;

Ki1 = 0;

MaxLess1 = 0;

RW2 = 0;

CL2 = 0;

Ki2 = 0;

MaxLess2 = 0;

RW3 = 0;

CL3 = 0;

Ki3 = 0;

MaxLess3 = 0;

RW4 = 0;

CL4 = 0;

Ki4 = 0;

MaxLess4 = 0;

RW5 = 0;

CL5 = 0;

Ki5 = 0;

MaxLess5 = 0;

RW6 = 0;

CL6 = 0;

Ki6 = 0;

MaxLess6 = 0;

LoopHuristicIndex = 0;

Move = 0;

MouseClick = 0;

AStarGreedyIndex = new int[20];

AStarGreedy = 0;

/\*SolderesOnTable = null;

ElephantOnTable = null;

HoursesOnTable = null;

CastlesOnTable = null;

MinisterOnTable = null;

KingOnTable = null;

\*/

MaxDuringLevelThinkingCreation = System.Convert.ToInt32(AllDraw.THIScomboBoxMaxLevelText);

}

}

//Check For Thinking Of Current Item Movments Finished.

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

{

Object a = new Object();

lock (a)

{

//For All kind of Current Thinking depend of current type consider finshing state thinking.

bool Finished = false;

{

//For Soldier

if (Kind == 1)

{

if (Dum.SolderesOnTable[i].SoldierThinking[0].ThinkingFinished)

return true;

}

//For Elephant

else if (Kind == 2)

{

if (Dum.ElephantOnTable[i].ElefantThinking[0].ThinkingFinished)

return true;

}

//For Hourse.

else if (Kind == 3)

{

if (Dum.HoursesOnTable[i].HourseThinking[0].ThinkingFinished)

return true;

}

//For Castles.

else if (Kind == 4)

{

if (Dum.CastlesOnTable[i].CastleThinking[0].ThinkingFinished)

return true;

}

//For Minsters.

else if (Kind == 5)

{

if (Dum.MinisterOnTable[i].MinisterThinking[0].ThinkingFinished)

return true;

}

//For Kings.

else if (Kind == 6)

{

if (Dum.KingOnTable[i].KingThinking[0].ThinkingFinished)

return true;

}

}

return Finished;

}

}

//Rearrange AllDraw Object Content.

public void SetRowColumn(int index)

{

Object a1 = new Object();

lock (a1)

{

SetObjectNumbers(TableList[0]);

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;

try

{

SetRowColumnFinished = false;

Move = 0;

//Intiate Dummy Variables.

//When Conversion Occured.

SolderesOnTable = new DrawSoldier[SodierHigh];

ElephantOnTable = new DrawElefant[ElefantHigh];

HoursesOnTable = new DrawHourse[HourseHight];

CastlesOnTable = new DrawCastle[CastleHigh];

MinisterOnTable = new DrawMinister[MinisterHigh];

KingOnTable = new DrawKing[KingHigh];

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++)

{

if (TableList[index][Row, Column] == 0)

continue;

//When Things are Soldiers.

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

{

//Determine int

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//When int is Gray.

if (a == Color.Gray)

{

try

{

if (SolderesOnTable[So1] != null)

SolderesOnTable[So1].Dispose();

//Construct Soder Gray.

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

//Increase So1.

So1++;

if (So1 > SodierMidle)

{

SodierMidle++;

SodierHigh++;

}

}

catch (Exception t)

{

Log(t);

}

}

//When int is Brown

else

{

try

{

if (SolderesOnTable[So2] != null)

SolderesOnTable[So2].Dispose();

//Construct Soldeir Brown.

SolderesOnTable[So2] = new DrawSoldier(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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(TableList[index][Row, Column]) == 2)

{

//Initiate Local Variables.

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//If Gray Elepahnt

if (a == Color.Gray)

{

try

{

if (ElephantOnTable[El1] != null)

ElephantOnTable[El1].Dispose();

//Construction of Draw Object.

ElephantOnTable[El1] = new DrawElefant(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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

{

if (ElephantOnTable[El2] != null)

ElephantOnTable[El2].Dispose();

//Construction of Draw Brown Elephant Object.

ElephantOnTable[El2] = new DrawElefant(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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(TableList[index][Row, Column]) == 3)

{

//Initiate Local Varibale and int.

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//If Gray Hourse.

if (a == Color.Gray)

{

try

{

if (HoursesOnTable[Ho1] != null)

HoursesOnTable[Ho1].Dispose();

//Construction of Draw Brown Hourse.

HoursesOnTable[Ho1] = new DrawHourse(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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

{

if (HoursesOnTable[Ho2] != null)

HoursesOnTable[Ho2].Dispose();

//Construction of Draw Brown Hourse.

HoursesOnTable[Ho2] = new DrawHourse(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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(TableList[index][Row, Column]) == 4)

{

//Initiate of Local Variables.

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//For Gray int.

if (a == Color.Gray)

{

try

{

if (CastlesOnTable[Br1] != null)

CastlesOnTable[Br1].Dispose();

//Construction of New Draw Gray Castles.

CastlesOnTable[Br1] = new DrawCastle(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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

{

if (CastlesOnTable[Br2] != null)

CastlesOnTable[Br2].Dispose();

//Construction Draw of New Brown Castles.

CastlesOnTable[Br2] = new DrawCastle(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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(TableList[index][Row, Column]) == 5)

{

//Initiate Local int Varibales.

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//For Gray ints.

if (a == Color.Gray)

{

try

{

if (MinisterOnTable[Mi1] != null)

MinisterOnTable[Mi1].Dispose();

//construction of new draw Gray Minster.

MinisterOnTable[Mi1] = new DrawMinister(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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

{

if (MinisterOnTable[Mi2] != null)

MinisterOnTable[Mi2].Dispose();

//Construction of New Draw Brown Minster.

MinisterOnTable[Mi2] = new DrawMinister(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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(TableList[index][Row, Column]) == 6)

{

//Initiate Of int.

Color a;

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

a = Color.Gray;

else

a = Color.Brown;

//int consideration.

if (a == Color.Gray)

{

try

{

if (KingOnTable[Ki1] != null)

KingOnTable[Ki1].Dispose();

//Construction of New Draw Gray King.

KingOnTable[Ki1] = new DrawKing(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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

{

if (KingOnTable[Ki2] != null)

KingOnTable[Ki2].Dispose();

//Construction of New Draw King Brown Object.

KingOnTable[Ki2] = new DrawKing(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, Row, Column, a, 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.

}

}

catch (Exception t)

{

Log(t);

}

SetObjectNumbers(TableList[0]);

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;

SetRowColumnFinished = true;

}

}

void SetRowColumnFinishedWait()

{

Object a = new Object();

lock (a)

{

do

{

Thread.Sleep(1);

} while (!SetRowColumnFinished);

}

}

//Max Index List Of Huristic AStarGreedy First Method.

public void BeginIndexFoundingMaxLessofMaxList(int ListIndex, List<double> Founded, ref double Less)

{

Object a = new Object();

lock (a)

{

//When There is Maximum Huristsic AStar Gredy Back Ward in Blitz Games.

if (MaxHuristicAStarGreedytBackWard.Count > 0)

{

//When List Index is less than Founded.

if (ListIndex < MaxHuristicAStarGreedytBackWard.Count)

return;

//Initiate Variable.

bool Added = false;

//Recursive Method.

BeginIndexFoundingMaxLessofMaxList(ListIndex++, Founded, ref Less);

//When Greater Less of First index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][1])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][1];

Added = true;

Founded.Add(2);

}

//When Greater Less of Second index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][5])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][5];

if (Added)

Founded.RemoveAt(Founded.Count - 1);

Added = true;

Founded.Add(6);

}

//When Greater Less of Third index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][9])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][9];

if (Added)

Founded.RemoveAt(Founded.Count - 1);

Added = true;

Founded.Add(10);

}

//When Greater Less of Foutrh index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][13])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][13];

if (Added)

Founded.RemoveAt(Founded.Count - 1);

Added = true;

Founded.Add(14);

}

//When Greater Less of Fifth index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][18])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][18];

if (Added)

Founded.RemoveAt(Founded.Count - 1);

Added = true;

Founded.Add(19);

}

//When Greater Less of Sith index Object Found.

if (Less < MaxHuristicAStarGreedytBackWard[ListIndex][22])

{

Less = MaxHuristicAStarGreedytBackWard[ListIndex][22];

if (Added)

Founded.RemoveAt(Founded.Count - 1);

Added = true;

Founded.Add(23);

}

}

}

}

//Method for Check of Existence of Checkmate less than for checked mate.

bool IsToCheckMateHasLessDeeperThanForCheckMate(AllDraw A, int Order, ref int ToCheckMate, ref int ForCheckMate, int AStarGreedy)

{

Object a = new Object();

lock (a)

{

//Initiate variables.

bool AA = false;

int CDummy = Order;

//For Gray One.

if (Order == 1)

{

//For Solderis.

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

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

{

try

{

//When there is Brown checked mate.

if (A.SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy == -1)

{

//Set.

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.KingOnTable[i].KingThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.KingOnTable[i].KingThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

}

else

{

//ChessRules.CurrentOrder = -1;

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

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

{

try

{

//When there is Brown checked mate.

if (A.SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

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

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

{

try

{

//When there is Brown checked mate.

if (A.KingOnTable[i].KingThinking[0].CheckMateAStarGreedy == -1)

{

ForCheckMate = AStarGreedy;

if (ToCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

else

{

//When there is Gray Checked mate.

if (A.KingOnTable[i].KingThinking[0].CheckMateAStarGreedy == 1)

{

ToCheckMate = AStarGreedy;

if (ForCheckMate >= 0)

if (ToCheckMate < ForCheckMate && ToCheckMate >= 0)

AA = true;

}

}

}

catch (Exception t)

{

Log(t);

}

Order \*= -1;

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

AA = AA || IsToCheckMateHasLessDeeperThanForCheckMate(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order, ref ToCheckMate, ref ForCheckMate, AStarGreedy++);

Order = CDummy;

}

}

ChessRules.CurrentOrder = CDummy;

return AA;

}

}

//When Penalty Regard Branches expanded to sub branches.

void IsPenaltyRegardCheckMateAtBranch(int Order, ref int Do, AllDraw Base)

{

Object a = new Object();

lock (a)

{

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

//For Gray Order.

if (Order == 1)

{

ChessRules AA = null;

//ChessRules.CurrentOrder = 1;

//For Soldeirs.

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

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

{

try

{

//Create Rules Objects For Soldiers.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, SolderesOnTable[i].SoldierThinking[0].TableListSolder[j][SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][1]]

, SolderesOnTable[i].SoldierThinking[0].TableListSolder[j]

, Order

, SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][0]

, SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][1]);

//When CheckMate Occured for Current Sodiers

if (AA.CheckMate(SolderesOnTable[i].SoldierThinking[0].TableListSolder[j], Order))

//When Self CheckMate

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

{

//Return Ignore

Do = -1;

//Set Superposition.

SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//When Enemy CheckMate

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

{

//Set Regard and Set Movements.

Do = 1;

//Regard Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

//Set Superpostion.

SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy = 1;

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//For Subbranchs.

for (int ii = 0; SolderesOnTable != null && SolderesOnTable[i] != null && SolderesOnTable[i].SoldierThinking[0] != null && ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

SolderesOnTable[ii].SoldierThinking[0].AStarGreedy[i].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//For Elephant.

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

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

{

try

{

//Create Elephant Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ElephantOnTable[i].ElefantThinking[0].TableListElefant[j][ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][0], ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][1]]

, ElephantOnTable[i].ElefantThinking[0].TableListElefant[j]

, Order

, ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][0]

, ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][1]);

//When CheckMate Occured for Current Elephant.

if (AA.CheckMate(ElephantOnTable[i].ElefantThinking[0].TableListElefant[j], Order))

//For Self Order CheckMate.

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

{

//Set Penalty Ignore.

Do = -1;

//Set Superposition.

ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//For Enemy Order CheckMate.

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

{

//Set Regard Continue.

Do = 1;

//Regard Subolders.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

//Set Superposition.

ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy = 1;

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//For Subbranchs.

for (int ii = 0; ElephantOnTable != null && ElephantOnTable[i] != null && ElephantOnTable[i].ElefantThinking[0] != null && ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

ElephantOnTable[ii].ElefantThinking[0].AStarGreedy[i].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//For Hourse.

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

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

{

try

{

//Set Hourse Rules Objects.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, HoursesOnTable[i].HourseThinking[0].TableListHourse[j][HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][0], HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][1]]

, HoursesOnTable[i].HourseThinking[0].TableListHourse[j]

, Order

, HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][0]

, HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][1]);

//When CheckMate Occured.

if (AA.CheckMate(HoursesOnTable[i].HourseThinking[0].TableListSolder[j], Order))

//For Self CheckMate.

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

{

//Set Ignore.

Do = -1;

//Set Superposition.

HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//For Enemy CheckMate.

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

{

//Set Regard.

Do = 1;

//Superposition.

HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy = 1;

//Set Regard For Sub Branches.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Sub branchs For Hourse.

for (int ii = 0; HoursesOnTable != null && HoursesOnTable[i] != null && HoursesOnTable[i].HourseThinking[0] != null && ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//For Gray Briges.

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

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

{

try

{

//Castles Gray Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, CastlesOnTable[i].CastleThinking[0].TableListCastle[j][CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][0], CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][1]]

, CastlesOnTable[i].CastleThinking[0].TableListCastle[j]

, Order

, CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][0]

, CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][1]);

//When Current Gray Castles CheckMate.

if (AA.CheckMate(CastlesOnTable[i].CastleThinking[0].TableListCastle[j], Order))

//For Self CheckMate

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

{

//Set Penalty Ignore.

Do = -1;

//Set Superposition.

CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy = -1;

//Penalty Sub branchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//For Enemy CheckMate.

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

{

//Set Regard.

Do = 1;

//Superpoistion.

CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy = 1;

//Set Regard Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//For Castles Gray Subbranchs.

try

{

for (int ii = 0; CastlesOnTable != null && CastlesOnTable[i] != null && CastlesOnTable[i].CastleThinking[0] != null && ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//For Ministers Gray.

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

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

{

try

{

//Minister Gray Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, MinisterOnTable[i].MinisterThinking[0].TableListMinister[j][MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][0], MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][1]]

, MinisterOnTable[i].MinisterThinking[0].TableListMinister[j]

, Order

, MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][0]

, MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][1]);

//When M ate Occured in Minister Gray.

if (AA.CheckMate(MinisterOnTable[i].MinisterThinking[0].TableListMinister[j], Order))

//Self CheckMate.

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

{

//Penalty Ignore.

Do = -1;

//Superpostion.

MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//For Enemy CheckMate.

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

{

//Regard Setting.

Do = 1;

//Superpoistion.

MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy = 1;

//Set Subbranchs Regard.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//For Gray Ministers Subbranchs.

try

{

for (int ii = 0; MinisterOnTable != null && MinisterOnTable[i] != null && MinisterOnTable[i].MinisterThinking[0] != null && ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//For Gray King.

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

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

{

try

{

//Gray King Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, KingOnTable[i].KingThinking[0].TableListKing[j][KingOnTable[i].KingThinking[0].RowColumnKing[j][0], KingOnTable[i].KingThinking[0].RowColumnKing[j][1]]

, KingOnTable[i].KingThinking[0].TableListKing[j]

, Order

, KingOnTable[i].KingThinking[0].RowColumnKing[j][0]

, KingOnTable[i].KingThinking[0].RowColumnKing[j][1]);

//When CheckMate Occured in King Gray.

if (AA.CheckMate(KingOnTable[i].KingThinking[0].TableListKing[j], Order))

//Self CheckMate.

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

{

//Penalty Ignore.

Do = -1;

//Superposition.

KingOnTable[i].KingThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//Self CheckMate.

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

{

//Regard Setting.

Do = 1;

//Superpoistion.

KingOnTable[i].KingThinking[0].CheckMateAStarGreedy = 1;

//Regard Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//For King Gray Subbranchs.

try

{

for (int ii = 0; KingOnTable != null && KingOnTable[i] != null && KingOnTable[i].KingThinking[0] != null && ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

}

//For Brown Order.

else

{

ChessRules AA = null;

//ChessRules.CurrentOrder = -1;

//For Solders Brown.

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

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

{

try

{

//Solders Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, SolderesOnTable[i].SoldierThinking[0].TableListSolder[j][SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][0], SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][1]]

, SolderesOnTable[i].SoldierThinking[0].TableListSolder[j]

, Order

, SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][0]

, SolderesOnTable[i].SoldierThinking[0].RowColumnSoldier[j][1]);

//When Solders Brown CheckMate Occured.

if (AA.CheckMate(SolderesOnTable[i].SoldierThinking[0].TableListSolder[j], Order))

//Self CheckMate.

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

{

//Ignore Penalty.

Do = -1;

//Supperpoistion.

SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs Soders Brown.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//Self CheckMate.

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

{

//Set Regard.

Do = 1;

//Superpoition.

SolderesOnTable[i].SoldierThinking[0].CheckMateAStarGreedy = 1;

//Penalty Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Solders Brown Subbranchs Calling.

try

{

for (int ii = 0; SolderesOnTable != null && SolderesOnTable[i] != null && SolderesOnTable[i].SoldierThinking[0] != null && ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//Elephant Brown

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

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

{

try

{

//Elephant Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ElephantOnTable[i].ElefantThinking[0].TableListElefant[j][ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][0], ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][1]]

, ElephantOnTable[i].ElefantThinking[0].TableListElefant[j]

, Order

, ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][0]

, ElephantOnTable[i].ElefantThinking[0].RowColumnElefant[j][1]);

//CheckMate Occured in Elephenat Brown.

if (AA.CheckMate(ElephantOnTable[i].ElefantThinking[0].TableListElefant[j], Order))

//Self CheckMate.

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

{

//Ignore Penalty.

Do = -1;

//Superpoistion.

ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//CheckMate Enemy.

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

{

//Set Regrading.

Do = 1;

//Superposition.

ElephantOnTable[i].ElefantThinking[0].CheckMateAStarGreedy = 1;

//Regrad Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Subbranchs Elephenat Brown Calling.

try

{

for (int ii = 0; ElephantOnTable != null && ElephantOnTable[i] != null && ElephantOnTable[i].ElefantThinking[0] != null && ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//Hourse Brown

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

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

{

try

{

//Hourse Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, HoursesOnTable[i].HourseThinking[0].TableListHourse[j][HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][0], HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][1]]

, HoursesOnTable[i].HourseThinking[0].TableListHourse[j]

, Order

, HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][0]

, HoursesOnTable[i].HourseThinking[0].RowColumnHourse[j][1]);

//When Hourse Broin CheckMate Ocuucred.

if (AA.CheckMate(HoursesOnTable[i].HourseThinking[0].TableListSolder[j], Order))

//Self CheckMate.

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

{

//Ignore Penalty.

Do = -1;

//Superposition.

HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranchs.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//CheckMate Enemy.

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

{

//Set Regrad.

Do = 1;

//Superposition.

HoursesOnTable[i].HourseThinking[0].CheckMateAStarGreedy = 1;

//Regrad Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Hourse Brown Calling Subbranchs.

try

{

for (int ii = 0; HoursesOnTable != null && HoursesOnTable[i] != null && HoursesOnTable[i].HourseThinking[0] != null && ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//Castles Brown

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

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

{

try

{

//Castles Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, CastlesOnTable[i].CastleThinking[0].TableListCastle[j][CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][0], CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][1]]

, CastlesOnTable[i].CastleThinking[0].TableListCastle[j]

, Order

, CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][0]

, CastlesOnTable[i].CastleThinking[0].RowColumnCastle[j][1]);

//When Brown Castles CheckMate Occured.

if (AA.CheckMate(CastlesOnTable[i].CastleThinking[0].TableListCastle[j], Order))

//Self CheckMate.

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

{

//Ignore CheckMate.

Do = -1;

//Superpoistion.

CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy = -1;

//Subbranchs Penalty.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//CheckMate Enemy.

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

{

//Set Regard.

Do = 1;

//Superpoistion.

CastlesOnTable[i].CastleThinking[0].CheckMateAStarGreedy = 1;

//Regard Subbranchs.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Brown Castles Calling Subbranches.

try

{

for (int ii = 0; CastlesOnTable != null && CastlesOnTable[i] != null && CastlesOnTable[i].CastleThinking[0] != null && ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//Minister Brown

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

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

{

try

{

//Minister Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, MinisterOnTable[i].MinisterThinking[0].TableListMinister[j][MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][0], MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][1]]

, MinisterOnTable[i].MinisterThinking[0].TableListMinister[j]

, Order

, MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][0]

, MinisterOnTable[i].MinisterThinking[0].RowColumnMinister[j][1]);

//When Minister Borwn CheckMate Occcured.

if (AA.CheckMate(MinisterOnTable[i].MinisterThinking[0].TableListMinister[j], Order))

//Self CheckMate.

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

{

//Set Ignore.

Do = -1;

//Superpoistion.

MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy = -1;

//Penalty Subbranches.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//CheckMate Enemy.

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

{

//Set Regard.

Do = 1;

//Superposition.

MinisterOnTable[i].MinisterThinking[0].CheckMateAStarGreedy = 1;

//Regard SubBranches.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//Minister Brown SubBranches Calling.

try

{

for (int ii = 0; MinisterOnTable != null && MinisterOnTable[i] != null && MinisterOnTable[i].MinisterThinking[0] != null && ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

//King Brown

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

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

{

try

{

//King Brown Rules.

AA = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, KingOnTable[i].KingThinking[0].TableListKing[j][KingOnTable[i].KingThinking[0].RowColumnKing[j][0], KingOnTable[i].KingThinking[0].RowColumnKing[j][1]]

, KingOnTable[i].KingThinking[0].TableListKing[j]

, Order

, KingOnTable[i].KingThinking[0].RowColumnKing[j][0]

, KingOnTable[i].KingThinking[0].RowColumnKing[j][1]);

//When King Brown Rules CheckMate Occcured.

if (AA.CheckMate(KingOnTable[i].KingThinking[0].TableListKing[j], Order))

//Self CheckMate.

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

{

//Set Ignore.

Do = -1;

//Superposition.

KingOnTable[i].KingThinking[0].CheckMateAStarGreedy = -1;

//Penalty SubBranches.

MakePenaltyAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

else

{

//CheckMate Enemy.

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

{

//Set Regard.

Do = 1;

//Superposition.

KingOnTable[i].KingThinking[0].CheckMateAStarGreedy = 1;

//Regard Subbranches.

MakeRegardAllCheckMateBranches(Base, AllDraw.OrderPlate);

}

}

}

catch (Exception t)

{

Log(t);

}

if (Do != -1)

{

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

//King Brown Subbranches Calling.

try

{

for (int ii = 0; KingOnTable != null && KingOnTable[i] != null && KingOnTable[i].KingThinking[0] != null && ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, Base);

}

catch (Exception t)

{

Log(t);

}

Order = COrder;

ChessRules.CurrentOrder = CDummy;

}

}

}

ChessRules.CurrentOrder = CDummy;

}

}

public void MakePenaltyAllCheckMateBranches(AllDraw A, int Order)

{

Object a = new Object();

lock (a)

{

int COrder = Order;

int CDummy = ChessRules.CurrentOrder;

if (Order == 1)

{

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

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

{

try

{

A.SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[i].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order);

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

}

else

{

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

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

{

try

{

A.SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[i].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmPenalty();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

MakePenaltyAllCheckMateBranches(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

}

}

}

public AllDraw RemovePenalltyFromFirstBranches(int Order)

{

Object a = new Object();

lock (a)

{

if (Order == 1)

{

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j].Initiate();

for (int k = 0; k < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; k++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].Initiate();

for (int k = 0; k < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; k++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].Initiate();

for (int k = 0; k < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; k++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].Initiate();

for (int k = 0; k < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; k++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].Initiate();

for (int k = 0; k < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; k++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].Initiate();

for (int k = 0; k < KingOnTable[i].KingThinking[0].AStarGreedy.Count; k++)

KingOnTable[i].KingThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j].Initiate();

for (int k = 0; k < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; k++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].Initiate();

for (int k = 0; k < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; k++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].Initiate();

for (int k = 0; k < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; k++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].Initiate();

for (int k = 0; k < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; k++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

UsePenaltyRegardMechnisamT = false;

try

{

MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].Initiate();

for (int k = 0; k < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; k++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

for (int j = 0; KingOnTable != null && KingOnTable[i] != null

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

{

UsePenaltyRegardMechnisamT = false;

try

{

KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].Initiate();

for (int k = 0; k < KingOnTable[i].KingThinking[0].AStarGreedy.Count; k++)

KingOnTable[i].KingThinking[0].AStarGreedy[k].RemovePenalltyFromFirstBranches(Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

}

return this;

}

}

public AllDraw FoundOfCurrentTableNode(int[,] Tab, int Order, ref AllDraw THIS, ref bool Found)

{

Object a = new Object();

lock (a)

{

//if (Found)

//return THIS;

if (TableList.Count > 0 && ThinkingChess.TableEqual(TableList[0], Tab))

{

THIS = this;

Found = true;

return THIS;

}

else

if (Order == 1)

{

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

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

{

try

{

if (ThinkingChess.TableEqual(SolderesOnTable[i].SoldierThinking[0].TableListSolder[j], Tab))

{

if (SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count > j && SolderesOnTable[i].SoldierThinking[0].AStarGreedy != null)

{

THIS = SolderesOnTable[i].SoldierThinking[0].AStarGreedy[i];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(ElephantOnTable[i].ElefantThinking[0].TableListElefant[j], Tab))

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count > j && ElephantOnTable[i].ElefantThinking[0].AStarGreedy != null)

{

THIS = ElephantOnTable[i].ElefantThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(HoursesOnTable[i].HourseThinking[0].TableListHourse[j], Tab))

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count > j && HoursesOnTable[i].HourseThinking[0].AStarGreedy != null)

{

THIS = HoursesOnTable[i].HourseThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(CastlesOnTable[i].CastleThinking[0].TableListCastle[j], Tab))

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count > j && CastlesOnTable[i].CastleThinking[0].AStarGreedy != null)

{

THIS = CastlesOnTable[i].CastleThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(MinisterOnTable[i].MinisterThinking[0].TableListMinister[j], Tab))

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count > j && MinisterOnTable[i].MinisterThinking[0].AStarGreedy != null)

{

THIS = MinisterOnTable[i].MinisterThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(KingOnTable[i].KingThinking[0].TableListKing[j], Tab))

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count > j && KingOnTable[i].KingThinking[0].AStarGreedy != null)

{

THIS = KingOnTable[i].KingThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

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

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

{

try

{

if (ThinkingChess.TableEqual(SolderesOnTable[i].SoldierThinking[0].TableListSolder[j], Tab))

{

if (SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count > j && SolderesOnTable[i].SoldierThinking[0].AStarGreedy != null)

{

THIS = SolderesOnTable[i].SoldierThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(ElephantOnTable[i].ElefantThinking[0].TableListElefant[j], Tab))

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count > j && ElephantOnTable[i].ElefantThinking[0].AStarGreedy != null)

{

THIS = ElephantOnTable[i].ElefantThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(HoursesOnTable[i].HourseThinking[0].TableListHourse[j], Tab))

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count > j && HoursesOnTable[i].HourseThinking[0].AStarGreedy != null)

{

THIS = HoursesOnTable[i].HourseThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(CastlesOnTable[i].CastleThinking[0].TableListCastle[j], Tab))

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count > j && CastlesOnTable[i].CastleThinking[0].AStarGreedy != null)

{

THIS = CastlesOnTable[i].CastleThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(MinisterOnTable[i].MinisterThinking[0].TableListMinister[j], Tab))

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count > j && MinisterOnTable[i].MinisterThinking[0].AStarGreedy != null)

{

THIS = MinisterOnTable[i].MinisterThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ThinkingChess.TableEqual(KingOnTable[i].KingThinking[0].TableListKing[j], Tab))

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count > j && KingOnTable[i].KingThinking[0].AStarGreedy != null)

{

THIS = KingOnTable[i].KingThinking[0].AStarGreedy[j];

Found = true;

return THIS;

}

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].FoundOfCurrentTableNode(Tab, Order \* -1, ref THIS, ref Found);

}

catch (Exception t)

{

Log(t);

}

}

}

return THIS;

}

}

public AllDraw FoundOfLeafDepenOfKind(int Kind, ref AllDraw Leaf, ref bool Found, int Order, ref int OrderLeaf)

{

Object a = new Object();

lock (a)

{

//if (ThinkingChess.FoundFirstMating > MaxAStarGreedy)

//return Leaf

if (Found)

return Leaf;

if (Order == 1)

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].IsThereMateOfEnemy || SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0 && Kind == 1)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count - 1; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0 && Kind == 2)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count - 1; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0 && Kind == 3)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count - 1; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0 && Kind == 4)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count - 1; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0 && Kind == 5)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count - 1; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0 && Kind == 6)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count - 1; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0 && Kind == 1)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count - 1; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0 && Kind == 2)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count - 1; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0 && Kind == 3)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count - 1; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0 && Kind == 4)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count - 1; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0 && Kind == 5)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count - 1; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0 && Kind == 6)

{

Found = true;

Leaf = this;

return Leaf;

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count - 1; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].FoundOfLeafDepenOfKind(Kind, ref Leaf, ref Found, Order \* -1, ref OrderLeaf);

}

catch (Exception t)

{

Log(t);

}

}

}

return Leaf;

}

}

public bool IsFoundOfLeafDepenOfKindhaveVictory(int Kind, ref bool Found, int Order)

{

Object a = new Object();

lock (a)

{

//if (ThinkingChess.FoundFirstMating > MaxAStarGreedy)

//return Leaf

if (Found)

return true;

if (Order == 1)

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].IsThereMateOfEnemy// && SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0

&& Kind == 1)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count ; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].IsThereMateOfEnemy //&& ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0

&& Kind == 2)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count ; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].IsThereMateOfEnemy //&& HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0

&& Kind == 3)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count ; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].IsThereMateOfEnemy //&& CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0

&& Kind == 4)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count ; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].IsThereMateOfEnemy //&& MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0

&& Kind == 5)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count ; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].IsThereMateOfEnemy //&& KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0

&& Kind == 6)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count ; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].IsThereMateOfEnemy //&& SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0

&& Kind == 1)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count ; ii++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].IsThereMateOfEnemy //&& ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0

&& Kind == 2)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count ; ii++)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].IsThereMateOfEnemy //&& HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0

&& Kind == 3)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count ; ii++)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].IsThereMateOfEnemy //&& CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0

&& Kind == 4)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count ; ii++)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].IsThereMateOfEnemy //&& MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0

&& Kind == 5)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count ; ii++)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].IsThereMateOfEnemy //&& KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0

&& Kind == 6)

{

Found = true;

return true;

}

else

for (int ii = 0; ii < KingOnTable[i].KingThinking[0].AStarGreedy.Count ; ii++)

KingOnTable[i].KingThinking[0].AStarGreedy[ii].IsFoundOfLeafDepenOfKindhaveVictory(Kind, ref Found, Order \* -1);

}

catch (Exception t)

{

Log(t);

}

}

}

return Found;

}

}

public void FoundOfLeafDepenOfKindFullGame(int[,] table, int Order, int iAStarGreedy, int ii, int jj, int ik, int jjj, bool FOUND, int LeafAStarGreedy)

{

Object a1 = new Object();

lock (a1)

{

//if()

bool FullGameFound = false;

//if (ThinkingChess.FoundFirstMating > MaxAStarGreedy)

// return;

Object O = new Object();

lock (O)

{

table = CloneATable(table);

OutPut = "\r\nLeaf Tree Creation is " + LeafAStarGreedy.ToString() + "at AStarGreedy " + iAStarGreedy.ToString();

if (Order == 1)

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(SolderesOnTable[i].SoldierThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(ElephantOnTable[i].ElefantThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

HoursesOnTable[i].HourseThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(HoursesOnTable[i].HourseThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

CastlesOnTable[i].CastleThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(CastlesOnTable[i].CastleThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(MinisterOnTable[i].MinisterThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

KingOnTable[i].KingThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(KingOnTable[i].KingThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

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

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

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; ii < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(SolderesOnTable[i].SoldierThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(ElephantOnTable[i].ElefantThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

HoursesOnTable[i].HourseThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(HoursesOnTable[i].HourseThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

CastlesOnTable[i].CastleThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(CastlesOnTable[i].CastleThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(MinisterOnTable[i].MinisterThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

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

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

{

try

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count == 0)

{

FullGameFound = true;

FullGameThinkingTree(Order, iAStarGreedy, ii, jj, ik, jjj, false, LeafAStarGreedy);

}

else

for (int iii = 0; iii < KingOnTable[i].KingThinking[0].AStarGreedy.Count; iii++)

{

ThinkingChess.NumbersOfAllNode++;

KingOnTable[i].KingThinking[0].AStarGreedy[iii].FoundOfLeafDepenOfKindFullGame(KingOnTable[i].KingThinking[0].TableT, Order \* -1, iAStarGreedy, ii, jj, ik, jjj, FOUND, LeafAStarGreedy++);

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

if (!FullGameFound)

{

Object O1 = new Object();

lock (O1)

{

iAStarGreedy++;

Color a = Color.Gray;

if (Order == -1)

a = Color.Brown;

InitiateAStarGreedytObject(MaxAStarGreedy, ii, jj, a, table, Order, false, false, LeafAStarGreedy);

//Initiate(ii, jj, a, table, Order, false, false,LeafAStarGreedy);

}

}

return;

}

}

public void MakeRegardAllCheckMateBranches(AllDraw A, int Order)

{

Object a = new Object();

lock (a)

{

int COrder = Order;

int CDummy = ChessRules.CurrentOrder;

if (Order == 1)

{

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

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

{

try

{

A.SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[i].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

}

else

{

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

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

{

try

{

A.SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[i].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.SolderesOnTable != null && SolderesOnTable[i] != null && A.SolderesOnTable[i].SoldierThinking[0] != null && ii < A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.SolderesOnTable[i].SoldierThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.ElephantOnTable != null && ElephantOnTable[i] != null && A.ElephantOnTable[i].ElefantThinking[0] != null && ii < A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.ElephantOnTable[i].ElefantThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.HoursesOnTable != null && HoursesOnTable[i] != null && A.HoursesOnTable[i].HourseThinking[0] != null && ii < A.HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.HoursesOnTable[i].HourseThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.CastlesOnTable != null && CastlesOnTable[i] != null && A.CastlesOnTable[i].CastleThinking[0] != null && ii < A.CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.CastlesOnTable[i].CastleThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.MinisterOnTable != null && MinisterOnTable[i] != null && A.MinisterOnTable[i].MinisterThinking[0] != null && ii < A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.MinisterOnTable[i].MinisterThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

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

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

{

try

{

A.KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmRegard();

}

catch (Exception t)

{

Log(t);

}

Order \*= -1; ChessRules.CurrentOrder \*= -1;

try

{

for (int ii = 0; A.KingOnTable != null && KingOnTable[i] != null && A.KingOnTable[i].KingThinking[0] != null && ii < A.KingOnTable[i].KingThinking[0].AStarGreedy.Count; ii++)

MakeRegardAllCheckMateBranches(A.KingOnTable[i].KingThinking[0].AStarGreedy[ii], Order);

}

catch (Exception t)

{

Log(t);

}

Order = COrder; ChessRules.CurrentOrder = CDummy;

}

}

}

}

int[,] HuristicAStarGreadySearchPenalties(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

ChessRules AB = null;

int ToCheckMate = -1, ForCheckMate = -1, j, i;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

//For Every Soldeir

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

{

//For Every Soldier Movments AStarGreedy.

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

//When There is an Movment in such situation.

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 && //UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < SolderesOnTable[i].SoldierThinking[k].AStarGreedy.Count - 1; ij++)

SolderesOnTable[i].SoldierThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, SolderesOnTable[i].SoldierThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsPenaltyAction() != 0 && SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[i].SoldierThinking[k].NumberOfPenalties;

TableHuristic = 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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

}

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (SolderesOnTable[i].SoldierThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (SolderesOnTable[i].SoldierThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

// }

else

{

}

}

//Sodleirs Initiate.

RW1 = i;

CL1 = k;

Ki1 = j;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

//Set Max of Soldier.

MaxLess1 = (SolderesOnTable[RW1].SoldierThinking[CL1].NumberOfPenalties

);

//When Soldeirs is Greater than Others these Set Max.

if (MaxLess1 > MaxLess2)

MaxLess2 = -1;

if (MaxLess1 > MaxLess3)

MaxLess3 = -1;

if (MaxLess1 > MaxLess4)

MaxLess4 = -1;

if (MaxLess1 > MaxLess5)

MaxLess5 = -1;

if (MaxLess1 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[i].SoldierThinking[k].NumberOfPenalties;

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

Object O1 = new Object();

lock (O1)

{

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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

}

}

}

else

{

//Set Table and Huristic Value and Syntax.

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxSoldeirFounded)

continue;

Act = true;

AllDraw.LastRow = SolderesOnTable[RW1].SoldierThinking[CL1].Row;

AllDraw.LastColumn = SolderesOnTable[RW1].SoldierThinking[CL1].Column;

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[RW1].SoldierThinking[CL1].ReturnHuristic(RW1, Ki1, Order, false);

TableHuristic = SolderesOnTable[RW1].SoldierThinking[CL1].TableListSolder[Ki1];

Object O1 = new Object();

lock (O1)

{

ThingsConverter.ActOfClickEqualTow = true;

}

SolderesOnTable[RW1].ConvertOperation(SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1], a, SolderesOnTable[RW1].SoldierThinking[CL1].TableListSolder[Ki1], Order, false, i);

int Sign = 1;

if (a == Color.Brown)

Sign = -1;

//If there is Soldier Convert.

if (SolderesOnTable[RW1].Convert)

{

if (SolderesOnTable[RW1].ConvertedToMinister)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 5 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToCastle)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 4 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToHourse)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 3 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToElefant)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 2 \* Sign;

}

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

//Do For Remaining Objects same as Soldeir Documentation.

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

{

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count - 1; ij++)

ElephantOnTable[i].ElefantThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, ElephantOnTable[i].ElefantThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsPenaltyAction() != 0 && ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].NumberOfPenalties; ;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (ElephantOnTable[i].ElefantThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (ElephantOnTable[i].ElefantThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

AB = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 2, TableS, Order, -1, -1);

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW2 = i;

CL2 = k;

Ki2 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess2 = (ElephantOnTable[RW2].ElefantThinking[CL2].NumberOfPenalties);

if (MaxLess2 > MaxLess1)

MaxLess1 = -1;

if (MaxLess2 > MaxLess3)

MaxLess3 = -1;

if (MaxLess2 > MaxLess4)

MaxLess4 = -1;

if (MaxLess2 > MaxLess5)

MaxLess5 = -1;

if (MaxLess2 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Elephant By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Elephant By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].NumberOfPenalties;

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

}

}

else

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxElephntFounded)

continue;

AllDraw.LastRow = ElephantOnTable[RW2].ElefantThinking[CL2].Row;

AllDraw.LastColumn = ElephantOnTable[RW2].ElefantThinking[CL2].Column;

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[RW2].ElefantThinking[CL2].ReturnHuristic(RW2, Ki2, Order, false);

TableHuristic = ElephantOnTable[RW2].ElefantThinking[CL2].TableListElefant[Ki2];

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < HoursesOnTable[i].HourseThinking[k].AStarGreedy.Count - 1; ij++)

HoursesOnTable[i].HourseThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, HoursesOnTable[i].HourseThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsPenaltyAction() != 0 && HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (HoursesOnTable[i].HourseThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (HoursesOnTable[i].HourseThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

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

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

{

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW3 = i;

CL3 = k;

Ki3 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess3 = (HoursesOnTable[RW3].HourseThinking[CL3].NumberOfPenalties);

if (MaxLess3 > MaxLess1)

MaxLess1 = -1;

if (MaxLess3 > MaxLess2)

MaxLess2 = -1;

if (MaxLess3 > MaxLess4)

MaxLess4 = -1;

if (MaxLess3 > MaxLess5)

MaxLess5 = -1;

if (MaxLess3 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Hourse By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Hourse By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].NumberOfPenalties;

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

}

}

else

//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxHourseFounded)

continue;

AllDraw.LastRow = HoursesOnTable[RW3].HourseThinking[CL3].Row;

AllDraw.LastColumn = HoursesOnTable[RW3].HourseThinking[CL3].Column;

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[RW3].HourseThinking[CL3].ReturnHuristic(RW3, Ki3, Order, false);

TableHuristic = HoursesOnTable[RW3].HourseThinking[CL3].TableListHourse[Ki3];

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

}

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

{

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < CastlesOnTable[i].CastleThinking[k].AStarGreedy.Count - 1; ij++)

CastlesOnTable[i].CastleThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, CastlesOnTable[i].CastleThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsPenaltyAction() != 0 && CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (CastlesOnTable[i].CastleThinking[0].NumberOfPenalties < Less) continue;

//When There is greater Huristic Movments.

if (CastlesOnTable[i].CastleThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW4 = i;

CL4 = k;

Ki4 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess4 = (CastlesOnTable[RW4].CastleThinking[CL4].NumberOfPenalties);

if (MaxLess4 > MaxLess1)

MaxLess1 = -1;

if (MaxLess4 > MaxLess2)

MaxLess2 = -1;

if (MaxLess4 > MaxLess3)

MaxLess3 = -1;

if (MaxLess4 > MaxLess5)

MaxLess5 = -1;

if (MaxLess4 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Castles By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Castles By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxCastlesFounded)

continue;

AllDraw.LastRow = CastlesOnTable[RW4].CastleThinking[CL4].Row;

AllDraw.LastColumn = CastlesOnTable[RW4].CastleThinking[CL4].Column;

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[RW4].CastleThinking[CL4].ReturnHuristic(RW4, Ki4, Order, false);

TableHuristic = CastlesOnTable[RW4].CastleThinking[CL4].TableListCastle[Ki4];

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

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

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < MinisterOnTable[i].MinisterThinking[k].AStarGreedy.Count - 1; ij++)

MinisterOnTable[i].MinisterThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, MinisterOnTable[i].MinisterThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsPenaltyAction() != 0 && MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[i].MinisterThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

if (Order != AllDraw.OrderPlate)

if (MinisterOnTable[i].MinisterThinking[0].NumberOfPenalties < Less)

continue;

if (MinisterOnTable[i].MinisterThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

{

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW5 = i;

CL5 = k;

Ki5 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess5 = (MinisterOnTable[RW5].MinisterThinking[CL5].NumberOfPenalties);

if (MaxLess5 > MaxLess1)

MaxLess1 = -1;

if (MaxLess5 > MaxLess2)

MaxLess2 = -1;

if (MaxLess5 > MaxLess3)

MaxLess3 = -1;

if (MaxLess5 > MaxLess4)

MaxLess4 = -1;

if (MaxLess5 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Minister By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Minister By Alice!";

//THIS.RefreshBoxText();

} //Set Table and Huristic Value and Syntax.

}

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[i].MinisterThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxMinisterFounded)

continue;

AllDraw.LastRow = MinisterOnTable[RW5].MinisterThinking[CL5].Row;

AllDraw.LastColumn = MinisterOnTable[RW5].MinisterThinking[CL5].Column;

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[RW5].MinisterThinking[CL5].ReturnHuristic(RW5, Ki5, Order, false);

TableHuristic = MinisterOnTable[RW5].MinisterThinking[CL5].TableListMinister[Ki5];

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < KingOnTable[i].KingThinking[k].AStarGreedy.Count - 1; ij++)

KingOnTable[i].KingThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, KingOnTable[i].KingThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsPenaltyAction() != 0 && KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[i].KingThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (KingOnTable[i].KingThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (KingOnTable[i].KingThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW6 = i;

CL6 = k;

Ki6 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

MaxLess6 = (KingOnTable[RW6].KingThinking[CL6].NumberOfPenalties);

if (MaxLess6 > MaxLess1)

MaxLess1 = -1;

if (MaxLess6 > MaxLess2)

MaxLess2 = -1;

if (MaxLess6 > MaxLess3)

MaxLess3 = -1;

if (MaxLess6 > MaxLess4)

MaxLess4 = -1;

if (MaxLess6 > MaxLess5)

MaxLess5 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic King By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic King By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[i].KingThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxKingFounded)

continue;

AllDraw.LastRow = KingOnTable[RW6].KingThinking[CL6].Row;

AllDraw.LastColumn = KingOnTable[RW6].KingThinking[CL6].Column;

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[RW6].KingThinking[CL6].ReturnHuristic(RW6, Ki6, Order, false);

TableHuristic = KingOnTable[RW6].KingThinking[CL6].TableListKing[Ki6];

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{ Log(t); }

}

}

catch (Exception t)

{

Log(t);

}

}

// else

{

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchSoldier(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

//For Every Soldier Movments AStarGreedy.

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

//When There is an Movment in such situation.

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (SolderesOnTable[i].SoldierThinking[k].AStarGreedy.Count > j)

SolderesOnTable[i].SoldierThinking[k].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(1, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (SolderesOnTable[i].SoldierThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsPenaltyAction() != 0 && SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

Object O1 = new Object();

lock (O1)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

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

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

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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

}

Object O = new Object();

lock (O)

{

RegardOccurred = true;

}

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (SolderesOnTable[i].SoldierThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

continue;

//When There is greater Huristic Movments.

if (SolderesOnTable[i].SoldierThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

// }

else

{

}

}

//Sodleirs Initiate.

RW1 = i;

CL1 = k;

Ki1 = j;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

//Set Max of Soldier.

MaxLess1 = (SolderesOnTable[RW1].SoldierThinking[CL1].ReturnHuristic(i, j, Order, AA)

);

//When Soldeirs is Greater than Others these Set Max.

if (MaxLess1 > MaxLess2)

MaxLess2 = -1;

if (MaxLess1 > MaxLess3)

MaxLess3 = -1;

if (MaxLess1 > MaxLess4)

MaxLess4 = -1;

if (MaxLess1 > MaxLess5)

MaxLess5 = -1;

if (MaxLess1 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O = new Object();

lock (O)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

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

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

Object O1 = new Object();

lock (O1)

{

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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

}

}

}

else

{

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchSoldierGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (SodierMidle != 0)

{

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

TableHuristic = HuristicAStarGreadySearchSoldier(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchSoldierBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (SodierMidle != SodierHigh)

{

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

TableHuristic = HuristicAStarGreadySearchSoldier(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchElephantGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (0 != ElefantMidle)

{

//Do For Remaining Objects same as Soldeir Documentation.

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

TableHuristic = HuristicAStarGreadySearchElephant(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchElephantBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (ElefantHigh != ElefantMidle)

{

//Do For Remaining Objects same as Soldeir Documentation.

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

TableHuristic = HuristicAStarGreadySearchElephant(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchElephant(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (ElephantOnTable[i].ElefantThinking[0].AStarGreedy.Count > j)

ElephantOnTable[i].ElefantThinking[0].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(2, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (ElephantOnTable[i].ElefantThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsPenaltyAction() != 0 && ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

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

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

}

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (ElephantOnTable[i].ElefantThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

continue;

//When There is greater Huristic Movments.

if (ElephantOnTable[i].ElefantThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

}

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

AB = new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 2, TableS, Order, -1, -1);

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW2 = i;

CL2 = k;

Ki2 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess2 = (ElephantOnTable[RW2].ElefantThinking[CL2].ReturnHuristic(RW2, Ki2, Order, false));

if (MaxLess2 > MaxLess1)

MaxLess1 = -1;

if (MaxLess2 > MaxLess3)

MaxLess3 = -1;

if (MaxLess2 > MaxLess4)

MaxLess4 = -1;

if (MaxLess2 > MaxLess5)

MaxLess5 = -1;

if (MaxLess2 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Elephant By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Elephant By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

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

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

}

}

else

{

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchHourseGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (0 != HourseMidle)

{

//For Every Soldeir

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

TableHuristic = HuristicAStarGreadySearchHourse(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchHourseBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

if (HourseHight != HourseMidle)

{

//For Every Soldeir

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

TableHuristic = HuristicAStarGreadySearchHourse(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchHourse(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object a1 = new Object();

lock (a1)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (HoursesOnTable[i].HourseThinking[0].AStarGreedy.Count > j)

HoursesOnTable[i].HourseThinking[0].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(3, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (HoursesOnTable[i].HourseThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsPenaltyAction() != 0 && HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

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

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

RegardOccurred = true;

}

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (HoursesOnTable[i].HourseThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

continue;

//When There is greater Huristic Movments.

if (HoursesOnTable[i].HourseThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

}

//retrive table of current huristic.

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

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

{

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW3 = i;

CL3 = k;

Ki3 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess3 = (HoursesOnTable[RW3].HourseThinking[CL3].ReturnHuristic(RW3, Ki3, Order, false));

if (MaxLess3 > MaxLess1)

MaxLess1 = -1;

if (MaxLess3 > MaxLess2)

MaxLess2 = -1;

if (MaxLess3 > MaxLess4)

MaxLess4 = -1;

if (MaxLess3 > MaxLess5)

MaxLess5 = -1;

if (MaxLess3 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Hourse By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Hourse By Alice!";

//THIS.RefreshBoxText();

}

}//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

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

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

}

}

else

//Set Table and Huristic Value and Syntax.

{

}

}

catch (Exception t)

{

Log(t);

}

}

// else

{

}

}

}

catch (Exception t)

{

Log(t);

}

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchCastleGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (0 != HourseMidle)

{

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

TableHuristic = HuristicAStarGreadySearchCastle(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchCastleBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (CastleMidle != CastleHigh)

{

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

TableHuristic = HuristicAStarGreadySearchCastle(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchCastle(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O1 = new Object();

lock (O1)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (CastlesOnTable[i].CastleThinking[0].AStarGreedy.Count > j)

CastlesOnTable[i].CastleThinking[0].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(4, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (CastlesOnTable[i].CastleThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsPenaltyAction() != 0 && CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

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

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

RegardOccurred = true;

}

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (CastlesOnTable[i].CastleThinking[0].ReturnHuristic(i, j, Order, AA) > Less) continue;

//When There is greater Huristic Movments.

if (CastlesOnTable[i].CastleThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

}

//retrive table of current huristic.

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW4 = i;

CL4 = k;

Ki4 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess4 = (CastlesOnTable[RW4].CastleThinking[CL4].ReturnHuristic(RW4, Ki4, Order, false));

if (MaxLess4 > MaxLess1)

MaxLess1 = -1;

if (MaxLess4 > MaxLess2)

MaxLess2 = -1;

if (MaxLess4 > MaxLess3)

MaxLess3 = -1;

if (MaxLess4 > MaxLess5)

MaxLess5 = -1;

if (MaxLess4 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object OO1 = new Object();

lock (OO1)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Castles By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Castles By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

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

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

}

}

else//Set Table and Huristic Value and Syntax.

{

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchMinsisterGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (0 != MinisterMidle)

{

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

TableHuristic = HuristicAStarGreadySearchMinsister(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchMinsisterBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (MinisterHigh != MinisterMidle)

{

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

TableHuristic = HuristicAStarGreadySearchMinsister(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchMinsister(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O3 = new Object();

lock (O3)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

try

{

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

{

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (MinisterOnTable[i].MinisterThinking[0].AStarGreedy.Count > j)

MinisterOnTable[i].MinisterThinking[0].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(5, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (MinisterOnTable[i].MinisterThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsPenaltyAction() != 0 && MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

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

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

RegardOccurred = true;

}

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

if (Order != AllDraw.OrderPlate)

if (MinisterOnTable[i].MinisterThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

continue;

if (MinisterOnTable[i].MinisterThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

Object O = new Object();

lock (O)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

}

//retrive table of current huristic.

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

{

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW5 = i;

CL5 = k;

Ki5 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess5 = (MinisterOnTable[RW5].MinisterThinking[CL5].ReturnHuristic(RW5, Ki5, Order, false));

if (MaxLess5 > MaxLess1)

MaxLess1 = -1;

if (MaxLess5 > MaxLess2)

MaxLess2 = -1;

if (MaxLess5 > MaxLess3)

MaxLess3 = -1;

if (MaxLess5 > MaxLess4)

MaxLess4 = -1;

if (MaxLess5 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Minister By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Minister By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

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

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

}

}

else//Set Table and Huristic Value and Syntax.

{

}

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchKingGray(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (0 != MinisterMidle)

{

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

TableHuristic = HuristicAStarGreadySearchKing(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchKingBrown(ref int[,] TableHuristic, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

if (KingHigh != KingMidle)

{

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

TableHuristic = HuristicAStarGreadySearchKing(ref TableHuristic, i, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

CodeClass.SaveByCode(1, callStack.GetFileLineNumber(), callStack.GetFileName());

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchKing(ref int[,] TableHuristic, int i, int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

ChessRules AB = null;

int j;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

try

{

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

{

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

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

continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

if (KingOnTable[i].KingThinking[0].AStarGreedy.Count > j)

KingOnTable[i].KingThinking[0].AStarGreedy[j].IsFoundOfLeafDepenOfKindhaveVictory(6, ref AA, Order \* -1);

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if (KingOnTable[i].KingThinking[0].ReturnHuristic(i, j, Order, AA) > Less && (KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsPenaltyAction() != 0 && KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA

)

{

Object OO = new Object();

lock (OO)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

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

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

RegardOccurred = true;

}

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (KingOnTable[i].KingThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

continue;

//When There is greater Huristic Movments.

if (KingOnTable[i].KingThinking[0].ReturnHuristic(i, j, Order, AA) > Less)

{

Object OO = new Object();

lock (OO)

{

ActionString = ThinkingChess.ActionsString; AllDraw.ActionStringReady = true;

}

//retrive table of current huristic.

//retrive table of current huristic.

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

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW6 = i;

CL6 = k;

Ki6 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

MaxLess6 = (KingOnTable[RW6].KingThinking[CL6].ReturnHuristic(RW6, Ki6, Order, false));

if (MaxLess6 > MaxLess1)

MaxLess1 = -1;

if (MaxLess6 > MaxLess2)

MaxLess2 = -1;

if (MaxLess6 > MaxLess3)

MaxLess3 = -1;

if (MaxLess6 > MaxLess4)

MaxLess4 = -1;

if (MaxLess6 > MaxLess5)

MaxLess5 = -1;

if (AStarGreedyi == 1)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic King By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic King By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

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

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

}

}

else//Set Table and Huristic Value and Syntax.

{

}

}

catch (Exception t)

{

Log(t);

}

}

// else

{

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchGray(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

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

TableHuristic = HuristicAStarGreadySearchSoldierGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchElephantGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchHourseGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchCastleGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchMinsisterGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchKingGray(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

return TableHuristic;

}

}

int[,] HuristicAStarGreadySearchBrown(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

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

TableHuristic = HuristicAStarGreadySearchSoldierBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchElephantBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchHourseBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchCastleBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchMinsisterBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

TableHuristic = HuristicAStarGreadySearchKingBrown(ref TableHuristic, AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

return TableHuristic;

}

}

int[,] BrownHuristicAStarGreaedySearchPenalites(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic, ref bool Act)

{

Object O = new Object();

lock (O)

{

ChessRules AB = null;

int ToCheckMate = -1, ForCheckMate = -1, j, i;

List<double> Founded = new List<double>();

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

bool AA = false;

int Do = 0;

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

//For Every Soldeir

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

{

//For Every Soldier Movments AStarGreedy.

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

//When There is an Movment in such situation.

try

{

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

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < SolderesOnTable[i].SoldierThinking[k].AStarGreedy.Count - 1; ij++)

SolderesOnTable[i].SoldierThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, SolderesOnTable[i].SoldierThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsPenaltyAction() != 0 && SolderesOnTable[i].SoldierThinking[k].PenaltyRegardListSolder[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[i].SoldierThinking[k].NumberOfPenalties;

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

Object OO = new Object();

lock (OO)

{

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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (SolderesOnTable[i].SoldierThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (SolderesOnTable[i].SoldierThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//if (CheckG || CheckB)

//{

//retrive table of current huristic.

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

// }

else

{

}

}

RW1 = i;

CL1 = k;

Ki1 = j;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess1 = SolderesOnTable[RW1].SoldierThinking[CL1].NumberOfPenalties;

if (MaxLess1 > MaxLess2)

MaxLess2 = -1;

if (MaxLess1 > MaxLess3)

MaxLess3 = -1;

if (MaxLess1 > MaxLess4)

MaxLess4 = -1;

if (MaxLess1 > MaxLess5)

MaxLess5 = -1;

if (MaxLess1 > MaxLess6)

MaxLess6 = -1;

//Set Table and Huristic Value and Syntax.

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

Act = true;

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

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

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[i].SoldierThinking[k].NumberOfPenalties;

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

Object O1 = new Object();

lock (O1)

{

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 there is Soldier Convert.

if (SolderesOnTable[i].Convert)

{

if (SolderesOnTable[i].ConvertedToMinister)

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

else if (SolderesOnTable[i].ConvertedToCastle)

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

else if (SolderesOnTable[i].ConvertedToHourse)

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

else if (SolderesOnTable[i].ConvertedToElefant)

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

}

}

else

{ //Set Table and Huristic Value and Syntax.

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki1 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxSoldeirFounded)

continue;

Act = true;

AllDraw.LastRow = SolderesOnTable[RW1].SoldierThinking[CL1].Row;

AllDraw.LastColumn = SolderesOnTable[RW1].SoldierThinking[CL1].Column;

AllDraw.NextRow = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][0];

AllDraw.NextColumn = SolderesOnTable[i].SoldierThinking[k].RowColumnSoldier[j][1];

Less = SolderesOnTable[RW1].SoldierThinking[CL1].ReturnHuristic(RW1, Ki1, Order, false);

TableHuristic = SolderesOnTable[RW1].SoldierThinking[CL1].TableListSolder[Ki1];

Object O1 = new Object();

lock (O1)

{

ThingsConverter.ActOfClickEqualTow = true;

}

SolderesOnTable[RW1].ConvertOperation(SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1], a, SolderesOnTable[RW1].SoldierThinking[CL1].TableListSolder[Ki1], Order, false, i);

int Sign = 1;

if (a == Color.Brown)

Sign = -1;

//If there is Soldier Convert.

if (SolderesOnTable[RW1].Convert)

{

if (SolderesOnTable[RW1].ConvertedToMinister)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 5 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToCastle)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 4 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToHourse)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 3 \* Sign;

else if (SolderesOnTable[RW1].ConvertedToElefant)

TableHuristic[SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][0], SolderesOnTable[RW1].SoldierThinking[CL1].RowColumnSoldier[Ki1][1]] = 2 \* Sign;

}

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{/\*

Order \*= -1;

ChessRules.CurrentOrder \*= -1;

for (int p = 0; p < SolderesOnTable[i].SoldierThinking[0].AStarGreedy.Count; p++)

SolderesOnTable[i].SoldierThinking[0].AStarGreedy[p].HuristicAStarGreedySearch(AStarGreedyi, A, a, ref Less, Order, false);

\*/

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

//Do For Remaining Objects same as Soldeir Documentation.

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

{

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

try

{

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

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < ElephantOnTable[i].ElefantThinking[k].AStarGreedy.Count - 1; ij++)

ElephantOnTable[i].ElefantThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, ElephantOnTable[i].ElefantThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsPenaltyAction() != 0 && ElephantOnTable[i].ElefantThinking[k].PenaltyRegardListElefant[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

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

continue;

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (ElephantOnTable[i].ElefantThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (ElephantOnTable[i].ElefantThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//if (CheckG || CheckB)

//{

//retrive table of current huristic.

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW2 = i;

CL2 = k;

Ki2 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess2 = (ElephantOnTable[RW2].ElefantThinking[CL2].NumberOfPenalties);

MaxLess1 = -1;

if (MaxLess2 > MaxLess3)

MaxLess3 = -1;

if (MaxLess2 > MaxLess4)

MaxLess4 = -1;

if (MaxLess2 > MaxLess5)

MaxLess5 = -1;

if (MaxLess2 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Elephant By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Elephant By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[i].ElefantThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki2 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxElephntFounded)

continue;

AllDraw.LastRow = ElephantOnTable[RW2].ElefantThinking[CL2].Row;

AllDraw.LastColumn = ElephantOnTable[RW2].ElefantThinking[CL2].Column;

AllDraw.NextRow = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][0];

AllDraw.NextColumn = ElephantOnTable[i].ElefantThinking[k].RowColumnElefant[j][1];

Act = true;

Less = ElephantOnTable[RW2].ElefantThinking[CL2].ReturnHuristic(RW2, Ki2, Order, false);

TableHuristic = ElephantOnTable[RW2].ElefantThinking[CL2].TableListElefant[Ki2];

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < HoursesOnTable[i].HourseThinking[k].AStarGreedy.Count - 1; ij++)

HoursesOnTable[i].HourseThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, HoursesOnTable[i].HourseThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsPenaltyAction() != 0 && HoursesOnTable[i].HourseThinking[k].PenaltyRegardListHourse[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (HoursesOnTable[i].HourseThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (HoursesOnTable[i].HourseThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//retrive table of current huristic.

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

{

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = i;

CL3 = k;

Ki3 = j;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess3 = HoursesOnTable[RW3].HourseThinking[CL3].NumberOfPenalties;

if (MaxLess3 > MaxLess1)

MaxLess1 = -1;

if (MaxLess3 > MaxLess2)

MaxLess2 = -1;

if (MaxLess3 > MaxLess4)

MaxLess4 = -1;

if (MaxLess3 > MaxLess5)

MaxLess5 = -1;

if (MaxLess3 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Hourse By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Hourse By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[i].HourseThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki3 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxHourseFounded)

continue;

AllDraw.LastRow = HoursesOnTable[RW3].HourseThinking[CL3].Row;

AllDraw.LastColumn = HoursesOnTable[RW3].HourseThinking[CL3].Column;

AllDraw.NextRow = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][0];

AllDraw.NextColumn = HoursesOnTable[i].HourseThinking[k].RowColumnHourse[j][1];

Act = true;

Less = HoursesOnTable[RW3].HourseThinking[CL3].ReturnHuristic(RW3, Ki3, Order, false);

TableHuristic = HoursesOnTable[RW3].HourseThinking[CL3].TableListHourse[Ki3];

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{ Log(t); }

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < CastlesOnTable[i].CastleThinking[k].AStarGreedy.Count - 1; ij++)

CastlesOnTable[i].CastleThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, CastlesOnTable[i].CastleThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsPenaltyAction() != 0 && CastlesOnTable[i].CastleThinking[k].PenaltyRegardListCastle[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

if (Order != AllDraw.OrderPlate)

if (CastlesOnTable[i].CastleThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (CastlesOnTable[i].CastleThinking[0].NumberOfPenalties < Less)

{

//retrive table of current huristic.

//retrive table of current huristic.

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW4 = i;

CL4 = k;

Ki4 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess4 = (CastlesOnTable[RW4].CastleThinking[CL4].NumberOfPenalties);

if (MaxLess4 > MaxLess1)

MaxLess1 = -1;

if (MaxLess4 > MaxLess2)

MaxLess2 = -1;

if (MaxLess4 > MaxLess3)

MaxLess3 = -1;

if (MaxLess4 > MaxLess5)

MaxLess5 = -1;

if (MaxLess4 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Castles By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Castles By Alice!";

//THIS.RefreshBoxText();

}

}

//Set Table and Huristic Value and Syntax.

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

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

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[i].CastleThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki4 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxCastlesFounded)

continue;

AllDraw.LastRow = CastlesOnTable[RW4].CastleThinking[CL4].Row;

AllDraw.LastColumn = CastlesOnTable[RW4].CastleThinking[CL4].Column;

AllDraw.NextRow = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][0];

AllDraw.NextColumn = CastlesOnTable[i].CastleThinking[k].RowColumnCastle[j][1];

Act = true;

Less = CastlesOnTable[RW4].CastleThinking[CL4].ReturnHuristic(RW4, Ki4, Order, false);

TableHuristic = CastlesOnTable[RW4].CastleThinking[CL4].TableListCastle[Ki4];

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

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

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

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

{

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

try

{

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

{

try

{

//For Penalty Reagrad Mechanisam of Current Check CheckMate Current Movments.

//if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < MinisterOnTable[i].MinisterThinking[k].AStarGreedy.Count - 1; ij++)

MinisterOnTable[i].MinisterThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, MinisterOnTable[i].MinisterThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsPenaltyAction() != 0 && MinisterOnTable[i].MinisterThinking[k].PenaltyRegardListMinister[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[i].MinisterThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (MinisterOnTable[i].MinisterThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (MinisterOnTable[i].MinisterThinking[0].NumberOfPenalties < Less)

//retrive table of current huristic.

{

//retrive table of current huristic.

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

{

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

}

RW5 = i;

CL5 = k;

Ki5 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxLess5 = (MinisterOnTable[RW5].MinisterThinking[CL5].NumberOfPenalties);

if (MaxLess5 > MaxLess1)

MaxLess1 = -1;

if (MaxLess5 > MaxLess2)

MaxLess2 = -1;

if (MaxLess5 > MaxLess3)

MaxLess3 = -1;

if (MaxLess5 > MaxLess4)

MaxLess4 = -1;

if (MaxLess5 > MaxLess6)

MaxLess6 = -1;

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Minister By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Minister By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[i].MinisterThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

RW5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki5 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxMinisterFounded)

continue;

AllDraw.LastRow = MinisterOnTable[RW5].MinisterThinking[CL5].Row;

AllDraw.LastColumn = MinisterOnTable[RW5].MinisterThinking[CL5].Column;

AllDraw.NextRow = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][0];

AllDraw.NextColumn = MinisterOnTable[i].MinisterThinking[k].RowColumnMinister[j][1];

Act = true;

Less = MinisterOnTable[RW5].MinisterThinking[CL5].ReturnHuristic(RW5, Ki5, Order, false);

TableHuristic = MinisterOnTable[RW5].MinisterThinking[CL5].TableListMinister[Ki5];

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

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

{

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

try

{

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

{

try

{

////if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT)

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

// continue;

int CDummy = ChessRules.CurrentOrder;

int COrder = Order;

try

{

ChessRules.CurrentOrder \*= -1;

Order \*= -1;

Do = 0;

if (UsePenaltyRegardMechnisamT)

{

for (int ij = 0; ij < KingOnTable[i].KingThinking[k].AStarGreedy.Count - 1; ij++)

KingOnTable[i].KingThinking[k].AStarGreedy[ij].IsPenaltyRegardCheckMateAtBranch(Order, ref Do, KingOnTable[i].KingThinking[k].AStarGreedy[ij]);

Order = COrder;

ChessRules.CurrentOrder = CDummy;

ToCheckMate = -1; ForCheckMate = -1; AA = IsToCheckMateHasLessDeeperThanForCheckMate(this, Order, ref ToCheckMate, ref ForCheckMate, 0); if (Do == -1)

continue;

}

}

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

Order = COrder;

ChessRules.CurrentOrder = CDummy;

if (AllDraw.OrderPlate == Order && AStarGreedyi == 1 //&& UsePenaltyRegardMechnisamT

)

if ((KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsPenaltyAction() != 0 && KingOnTable[i].KingThinking[k].PenaltyRegardListKing[j].IsRewardAction() == 1 && AStarGreedyi == 1) || Do == 1 || AA)

{

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[i].KingThinking[k].NumberOfPenalties;

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

RegardOccurred = true;

//if (Do == 1 || AA)

//return TableHuristic;

continue;

}

//When There is No Movments in Such Order Enemy continue.

if (Order != AllDraw.OrderPlate)

if (KingOnTable[i].KingThinking[0].NumberOfPenalties < Less)

continue;

//When There is greater Huristic Movments.

if (KingOnTable[i].KingThinking[0].NumberOfPenalties < Less)

//retrive table of current huristic.

{

//retrive table of current huristic.

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

//checked for Legal Movments ArgumentOutOfRangeException curnt game.

if (DynamicAStarGreedytPrograming && !CurrentTableHuristic && AStarGreedyi == 1)

{

try

{

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

catch (Exception t)

{

Log(t);

if (!IsEnemyThingsinStable(TableS, AllDraw.TableListAction[AllDraw.TableListAction.Count - 1], AllDraw.OrderPlate))

continue;

}

}

//When there is not Penalty regard mechanism.

//if (!UsePenaltyRegardMechnisamT)

{

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

//If there is kish or kshachamaz Order.

if (AB.Check(TableS, Order))

{

//When Order is Gray.

if (Order == 1)

{

//Continue When is kish CheckObjectDangour and AStarGreadyFirstSearch .

if (AB.CheckGray)

continue;

}

else

{

//Continue when CheckBrown and AStarGreadyFirstSearch.

if (AB.CheckBrown)

continue;

}

}

else

{

}

}

RW6 = i;

CL6 = k;

Ki6 = j;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

MaxLess6 = (KingOnTable[RW6].KingThinking[CL6].NumberOfPenalties);

if (MaxLess6 > MaxLess1)

MaxLess1 = -1;

if (MaxLess6 > MaxLess2)

MaxLess2 = -1;

if (MaxLess6 > MaxLess3)

MaxLess3 = -1;

if (MaxLess6 > MaxLess4)

MaxLess4 = -1;

if (MaxLess6 > MaxLess5)

MaxLess5 = -1;

if (AStarGreedyi == 1)

{

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic King By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic King By Alice!";

//THIS.RefreshBoxText();

}

}

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

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

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[i].KingThinking[k].NumberOfPenalties;

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

}

}

else//Set Table and Huristic Value and Syntax.

{

try

{

if (AStarGreedyi == 1)

{

//TakeRoot.Pointer = this;

//Found of Max Non Probable Movments.

Founded.Clear();

double LessB = Double.MinValue; ;

BeginIndexFoundingMaxLessofMaxList(0, Founded, ref LessB);

if (Founded[0] != 1)

continue;

RW6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0]];

CL6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 1];

Ki6 = (int)MaxHuristicAStarGreedytBackWard[0][(int)Founded[0] + 2];

if (Founded[0] != MaxKingFounded)

continue;

AllDraw.LastRow = KingOnTable[RW6].KingThinking[CL6].Row;

AllDraw.LastColumn = KingOnTable[RW6].KingThinking[CL6].Column;

AllDraw.NextRow = KingOnTable[i].KingThinking[k].RowColumnKing[j][0];

AllDraw.NextColumn = KingOnTable[i].KingThinking[k].RowColumnKing[j][1];

Act = true;

Less = KingOnTable[RW6].KingThinking[CL6].ReturnHuristic(RW6, Ki6, Order, false);

TableHuristic = KingOnTable[RW6].KingThinking[CL6].TableListKing[Ki6];

Object OO = new Object();

lock (OO)

{

if (Order == 1)

{

OutPut = "\r\nChess Huristic Sodier By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Huristic Sodier By Alice!";

//THIS.RefreshBoxText();

}

}

}

}

catch (Exception t)

{ Log(t); }

}

//else

{

}

}

catch (Exception t)

{

Log(t);

}

}

}

catch (Exception t)

{

Log(t);

}

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

catch (Exception t)

{

Log(t);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

return TableHuristic;

}

}

//AStarGreedy First Huristic Method.

public int[,] HuristicAStarGreedySearch(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic)

{

Object O = new Object();

lock (O)

{

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

AStarGreedyi++;

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

//Initiate For Dynamic Backward Current AStarGreedyi Non Minus Founded Max Movments Detection Global Variables.

List<double> Founded = new List<double>();

//Initiateing Indicating Huristic Multiple Same Value Best Found of Movments.

MaxLess1 = -1;

MaxLess2 = -1;

MaxLess3 = -1;

MaxLess4 = -1;

MaxLess5 = -1;

MaxLess6 = -1;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

double[] BacWard = new double[25];

if (AStarGreedyi > MaxAStarGreedy)

return TableHuristic;

bool Act = false;

if (Order == 1)

{

TableHuristic = HuristicAStarGreadySearchGray(AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

{

TableHuristic = HuristicAStarGreadySearchBrown(AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Store In Local Variable and Dynamic Purpose Proccessing.

//Every Non Minuse Non Idept in List Has Gretest Max Order.

//Is Desired of Idept Oner Best Movments.

BacWard[0] = AStarGreedyi;

BacWard[1] = MaxLess1;

BacWard[2] = RW1;

BacWard[3] = RW1;

BacWard[4] = Ki1;

BacWard[5] = MaxLess2;

BacWard[6] = RW2;

BacWard[7] = RW2;

BacWard[8] = Ki2;

BacWard[9] = MaxLess3;

BacWard[10] = RW3;

BacWard[11] = RW3;

BacWard[12] = Ki3;

BacWard[13] = MaxLess4;

BacWard[14] = RW4;

BacWard[15] = RW4;

BacWard[16] = Ki4;

BacWard[17] = MaxLess5;

BacWard[18] = RW5;

BacWard[19] = RW5;

BacWard[20] = Ki5;

BacWard[21] = MaxLess6;

BacWard[22] = RW6;

BacWard[23] = RW6;

BacWard[24] = Ki6;

//We Have Information of Maximum of Huristic in Each Level and Table.

MaxHuristicAStarGreedytBackWard.Add(BacWard);

MaxHuristicAStarGreedytBackWardTable.Add(TableHuristic);

Founded.Clear();

//If Found retrun table.

if (Act)

return TableHuristic;

//Return what found table.

return TableHuristic;

}

}

public int[,] HuristicAStarGreedySearchPenalties(int AStarGreedyi, Color a, int Order, bool CurrentTableHuristic)

{

Object O = new Object();

lock (O)

{

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

AStarGreedyi++;

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

//Initiate For Dynamic Backward Current AStarGreedyi Non Minus Founded Max Movments Detection Global Variables.

List<double> Founded = new List<double>();

//Initiateing Indicating Huristic Multiple Same Value Best Found of Movments.

MaxLess1 = -1;

MaxLess2 = -1;

MaxLess3 = -1;

MaxLess4 = -1;

MaxLess5 = -1;

MaxLess6 = -1;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

double[] BacWard = new double[25];

if (AStarGreedyi > MaxAStarGreedy)

return TableHuristic;

bool Act = false;

if (Order == 1)

{

TableHuristic = HuristicAStarGreadySearchPenalties(AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

else

{

TableHuristic = BrownHuristicAStarGreaedySearchPenalites(AStarGreedyi, a, Order, CurrentTableHuristic, ref Act);

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Store In Local Variable and Dynamic Purpose Proccessing.

//Every Non Minuse Non Idept in List Has Gretest Max Order.

//Is Desired of Idept Oner Best Movments.

BacWard[0] = AStarGreedyi;

BacWard[1] = MaxLess1;

BacWard[2] = RW1;

BacWard[3] = RW1;

BacWard[4] = Ki1;

BacWard[5] = MaxLess2;

BacWard[6] = RW2;

BacWard[7] = RW2;

BacWard[8] = Ki2;

BacWard[9] = MaxLess3;

BacWard[10] = RW3;

BacWard[11] = RW3;

BacWard[12] = Ki3;

BacWard[13] = MaxLess4;

BacWard[14] = RW4;

BacWard[15] = RW4;

BacWard[16] = Ki4;

BacWard[17] = MaxLess5;

BacWard[18] = RW5;

BacWard[19] = RW5;

BacWard[20] = Ki5;

BacWard[21] = MaxLess6;

BacWard[22] = RW6;

BacWard[23] = RW6;

BacWard[24] = Ki6;

//We Have Information of Maximum of Huristic in Each Level and Table.

MaxHuristicAStarGreedytBackWard.Add(BacWard);

MaxHuristicAStarGreedytBackWardTable.Add(TableHuristic);

Founded.Clear();

//If Found retrun table.

if (Act)

return TableHuristic;

//Return what found table.

return TableHuristic;

}

}

//Genethic Algorithm Game Method.

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

{

Object O = new Object();

lock (O)

{

//Initiate Local and Global Variables.

int Current = ChessRules.CurrentOrder;

int DummyOrder = Order;

TableList.Add(Table);

Object OO = new Object();

lock (OO)

{

ThinkingChess.NotSolvedKingDanger = false;

}

LoopHuristicIndex = 0;

//For One time.

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

{

//If Order is Gray.

Object O2 = new Object();

lock (O2)

{

if (Order == 1)

{

OutPut = "\r\nChess Genetic By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nChess Genetic By Alice!";

//THIS.RefreshBoxText();

}

}

//Initiate Local Variables.

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

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int In = 0;

//Found Of Random Movments.

do

{

if (Order == 1)

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

else

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

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

//If Order is Gray.

Object OOO = new Object();

lock (OOO)

{

if (Order == 1)

{

OutPut = "\r\nGenetic Algorithm Begin AStarGreedy " + i.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nGenetic Algirithm Begin AStarGreedy " + i.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

}

//Found Of Genetic Algorithm Movments By GeneticAlgorithm Call Objectsand Method.

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

//Found Table.

int[,] Tab = R.GenerateTable(TableListAction, 0, Order);

//If Order is Gray.

Object OOO1 = new Object();

lock (OOO1)

{

if (Order == 1)

{

OutPut = "\r\nGenetic Algorithm Finsished AStarGreedy " + i.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else//If Order is Brown.

{

OutPut = "\r\nGenetic Algirithm Finished AStarGreedy " + i.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

}

//If Table Found.

if (Tab != null)

{

//Construct a Clone Copy of Table.

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

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

{

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

}

//Initiate a Table.

Table = new int[8, 8];

//Construct a Clone Copy of Table.

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

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

{

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

}

//Initiate Local and Global Varibales.

TableList.Add(TablInit);

ClList.Add(CL);

RWList.Add(RW);

KiList.Add(Ki);

// Order = Order \* -1;

// ChessRules.CurrentOrder = Order;

AStarGreedy++;

//return;

}

}

//Determination of CheckMate Consideration.

(new ChessRules(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, 1, Table, Order, -1, -1)).CheckMate(Table, Order);

//Reconstruction of Order Global Varibales.

Order = DummyOrder;

ChessRules.CurrentOrder = Current;

}

}

//AStarGreedy First Initiat Thinking Main Method.

public AllDraw InitiateAStarGreedytOneNode(int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, int iIndex, int KindIndex, int LeafAStarGreedy

)

{

Object O = new Object();

lock (O)

{

SetObjectNumbers(Tab);

//List<Task> tHA = new List<Task>();

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

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

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

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

//ParallelOptions parallelOptions = new ParallelOptions();

//parallelOptions.MaxDegreeOfParallelism = PlatformHelper.ProcessorCount;

ThinkingChess.BeginThread = 0;

ThinkingChess.EndThread = 0;

//Initiate of global Variables Byte Local Variables.

int DummyOrder = new int();

DummyOrder = Order;

int DummyCurrentOrder = new int();

DummyCurrentOrder = ChessRules.CurrentOrder;

List<Task> tH = new List<Task>();

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

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int j = 0;

if (iAStarGreedy >= MaxAStarGreedy)

return null;

iAStarGreedy++;

{

//Initiate Of Local Variables.

{

{

//If Order is Gray.

if (Order == 1)

{

//For Gray Soldeirs Objects.

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

if (KindIndex == 1)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If Solders Not Exist Continue and Traversal Back.

//If There is no Thinking Movments on Current Object

if (!AllDraw.Blitz)

{

//Thinking of Gray Solder Operation.

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[iIndex].SoldierThinking[0].t = new Task(new Action(SolderesOnTable[iIndex].SoldierThinking[0].Thinking));

SolderesOnTable[iIndex].SoldierThinking[0].t.Start();

if (SolderesOnTable[iIndex].SoldierThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(SolderesOnTable[iIndex].SoldierThinking[0].t); } }

}

else if (AllDraw.Blitz)

//If There is A Soldeir Movments.

{

//Thinking of Gray Soldeir Operations.

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[iIndex].SoldierThinking[0].t = new Task(new Action(SolderesOnTable[iIndex].SoldierThinking[0].Thinking));

SolderesOnTable[iIndex].SoldierThinking[0].t.Start();

if (SolderesOnTable[iIndex].SoldierThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(SolderesOnTable[iIndex].SoldierThinking[0].t); } }

}

}

catch (Exception t)

{

//SolderesOnTable[iIndex] = null;

Log(t);

}

}

//Progressing.

//For All Gray Elephant Objects.

if (KindIndex == 2)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Elephant Gray Objects.

if (!AllDraw.Blitz)

{

//Operational Thinking Gray Elephant.

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[iIndex].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[iIndex].ElefantThinking[0].Thinking));

ElephantOnTable[iIndex].ElefantThinking[0].t.Start();

if (ElephantOnTable[iIndex].ElefantThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(ElephantOnTable[iIndex].ElefantThinking[0].t); } }

}//If There is Movment Thinking Gary Elphant Object List.

else if (AllDraw.Blitz)

{

//For Every Gray Elephant Thinking Movments.

//Gray Elephant Object Thinking Operations.

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[iIndex].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[iIndex].ElefantThinking[0].Thinking));

ElephantOnTable[iIndex].ElefantThinking[0].t.Start();

if (ElephantOnTable[iIndex].ElefantThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(ElephantOnTable[iIndex].ElefantThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

//Progressing.

//For All Gray Hourse Objects.

if (KindIndex == 3)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{

//Thinking of Gray Hourse Oprational.

HoursesOnTable[iIndex].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[iIndex].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[iIndex].HourseThinking[0].t = new Task(new Action(HoursesOnTable[iIndex].HourseThinking[0].Thinking));

HoursesOnTable[iIndex].HourseThinking[0].t.Start();

if (HoursesOnTable[iIndex].HourseThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(HoursesOnTable[iIndex].HourseThinking[0].t); } }

}

else if (AllDraw.Blitz)//If Table List Exist int The Thinking.

{

//Thinking Operation of Gray Hourse.

HoursesOnTable[iIndex].HourseThinking[0].TableT = HoursesOnTable[iIndex].HourseThinking[0].TableListHourse[j];

HoursesOnTable[iIndex].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[iIndex].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[iIndex].HourseThinking[0].t = new Task(new Action(HoursesOnTable[iIndex].HourseThinking[0].Thinking));

HoursesOnTable[iIndex].HourseThinking[0].t.Start();

if (HoursesOnTable[iIndex].HourseThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(HoursesOnTable[iIndex].HourseThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

//Progressing.

if (KindIndex == 4)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{

//When There is Possible Thinking Castle of Gray Table

//Thinking of Gray Castles Operational.

CastlesOnTable[iIndex].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[iIndex].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[iIndex].CastleThinking[0].t = new Task(new Action(CastlesOnTable[iIndex].CastleThinking[0].Thinking));

CastlesOnTable[iIndex].CastleThinking[0].t.Start();

if (CastlesOnTable[iIndex].CastleThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(CastlesOnTable[iIndex].CastleThinking[0].t); } }

}

else if (AllDraw.Blitz)

{

//When There is Possible Thinking Castle of Gray Table

//Thinking of Gray Castles Objective Movments.

CastlesOnTable[iIndex].CastleThinking[0].TableT = CastlesOnTable[iIndex].CastleThinking[0].TableListCastle[j];

CastlesOnTable[iIndex].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[iIndex].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[iIndex].CastleThinking[0].t = new Task(new Action(CastlesOnTable[iIndex].CastleThinking[0].Thinking));

CastlesOnTable[iIndex].CastleThinking[0].t.Start();

if (CastlesOnTable[iIndex].CastleThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(CastlesOnTable[iIndex].CastleThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

if (KindIndex == 5)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When There is Table Gray Minister Count of Thinking.

//Thinking of Gray Minister Operational.

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[iIndex].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[iIndex].MinisterThinking[0].Thinking));

MinisterOnTable[iIndex].MinisterThinking[0].t.Start();

if (MinisterOnTable[iIndex].MinisterThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(MinisterOnTable[iIndex].MinisterThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Table Gray Minister Count of Thinking.

{

//Thinking.

MinisterOnTable[iIndex].Table = MinisterOnTable[iIndex].MinisterThinking[0].TableListMinister[j];

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[iIndex].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[iIndex].MinisterThinking[0].Thinking));

MinisterOnTable[iIndex].MinisterThinking[0].t.Start();

if (MinisterOnTable[iIndex].MinisterThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(MinisterOnTable[iIndex].MinisterThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

if (KindIndex == 6)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When Thinking Gray King Count of Existing Operations.

//Thinking Of Gray King Operatins.

KingOnTable[iIndex].KingThinking[0].ThinkingBegin = true;

KingOnTable[iIndex].KingThinking[0].ThinkingFinished = false;

KingOnTable[iIndex].KingThinking[0].t = new Task(new Action(KingOnTable[iIndex].KingThinking[0].Thinking));

KingOnTable[iIndex].KingThinking[0].t.Start();

if (KingOnTable[iIndex].KingThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(KingOnTable[iIndex].KingThinking[0].t); } }

}

else if (AllDraw.Blitz)//When Thinking Gray King Count of Existing Operations.

{

//Gray King Thinking Operations.

KingOnTable[iIndex].KingThinking[0].ThinkingBegin = true;

KingOnTable[iIndex].KingThinking[0].ThinkingFinished = false;

KingOnTable[iIndex].KingThinking[0].t = new Task(new Action(KingOnTable[iIndex].KingThinking[0].Thinking));

KingOnTable[iIndex].KingThinking[0].t.Start();

if (KingOnTable[iIndex].KingThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(KingOnTable[iIndex].KingThinking[0].t); } }

}

}

catch (Exception t)

{

// KingOnTable[iIndex] = null;

Log(t);

}

}

}

else//Brown Order Considarations.

{

if (KindIndex == -1)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Wheen Brown King Object There is Not Continue Traversal Back.

//Thinking Operations of Brown Current Objects.

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[iIndex].SoldierThinking[0].t = new Task(new Action(SolderesOnTable[iIndex].SoldierThinking[0].Thinking));

SolderesOnTable[iIndex].SoldierThinking[0].t.Start();

if (SolderesOnTable[iIndex].SoldierThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(SolderesOnTable[iIndex].SoldierThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Thinking of Thinking Brown CurrentTable Objective Operations.

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[iIndex].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[iIndex].SoldierThinking[0].t = new Task(new Action(SolderesOnTable[iIndex].SoldierThinking[0].Thinking));

SolderesOnTable[iIndex].SoldierThinking[0].t.Start();

if (SolderesOnTable[iIndex].SoldierThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(SolderesOnTable[iIndex].SoldierThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

if (KindIndex == -2)

{

try

{

if (!AllDraw.Blitz)

{//When There is Current Brown Existing Objective Thinking Movments.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Thinking Operations of Brown Current Objects.

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[iIndex].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[iIndex].ElefantThinking[0].Thinking));

ElephantOnTable[iIndex].ElefantThinking[0].t.Start();

if (ElephantOnTable[iIndex].ElefantThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(ElephantOnTable[iIndex].ElefantThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Inititae Local Varibale By Global Gray Elephant Objects Varibales.

//Thinking of Thinking Brown CurrentTable Objective Operations.

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[iIndex].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[iIndex].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[iIndex].ElefantThinking[0].Thinking));

ElephantOnTable[iIndex].ElefantThinking[0].t.Start();

if (ElephantOnTable[iIndex].ElefantThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(ElephantOnTable[iIndex].ElefantThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

if (KindIndex == -3)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When There is Current Brown Existing Objective Thinking Movments.

//Thinking Operations of Brown Current Objects.

//HoursesOnTable[iIndex].HourseThinking[0].TableT = HoursesOnTable[iIndex].HourseThinking[0].TableT;

HoursesOnTable[iIndex].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[iIndex].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[iIndex].HourseThinking[0].t = new Task(new Action(HoursesOnTable[iIndex].HourseThinking[0].Thinking));

HoursesOnTable[iIndex].HourseThinking[0].t.Start();

if (HoursesOnTable[iIndex].HourseThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(HoursesOnTable[iIndex].HourseThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Thinking of Thinking Brown CurrentTable Objective Operations. SolderesOnTable[iIndex].SoldierThinking[0].Table = SolderesOnTable[iIndex].SoldierThinking[0].TableListSolder[j];

HoursesOnTable[iIndex].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[iIndex].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[iIndex].HourseThinking[0].t = new Task(new Action(HoursesOnTable[iIndex].HourseThinking[0].Thinking));

HoursesOnTable[iIndex].HourseThinking[0].t.Start();

if (HoursesOnTable[iIndex].HourseThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(HoursesOnTable[iIndex].HourseThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

//Progressing.

if (KindIndex == -4)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When There is Current Brown Existing Objective Thinking Movments.

//Thinking Operations of Brown Current Objects.

CastlesOnTable[iIndex].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[iIndex].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[iIndex].CastleThinking[0].t = new Task(new Action(CastlesOnTable[iIndex].CastleThinking[0].Thinking));

CastlesOnTable[iIndex].CastleThinking[0].t.Start();

if (CastlesOnTable[iIndex].CastleThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(CastlesOnTable[iIndex].CastleThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Thinking of Thinking Brown CurrentTable Objective Operations.

CastlesOnTable[iIndex].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[iIndex].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[iIndex].CastleThinking[0].t = new Task(new Action(CastlesOnTable[iIndex].CastleThinking[0].Thinking));

CastlesOnTable[iIndex].CastleThinking[0].t.Start();

if (CastlesOnTable[iIndex].CastleThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(CastlesOnTable[iIndex].CastleThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

if (KindIndex == -5)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When There is Current Brown Existing Objective Thinking Movments.

//Thinking Operations of Brown Current Objects.

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[iIndex].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[iIndex].MinisterThinking[0].Thinking));

MinisterOnTable[iIndex].MinisterThinking[0].t.Start();

if (MinisterOnTable[iIndex].MinisterThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(MinisterOnTable[iIndex].MinisterThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Thinking of Thinking Brown CurrentTable Objective Operations. SolderesOnTable[iIndex].SoldierThinking[0].Table = SolderesOnTable[iIndex].SoldierThinking[0].TableListSolder[j];

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[iIndex].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[iIndex].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[iIndex].MinisterThinking[0].Thinking));

MinisterOnTable[iIndex].MinisterThinking[0].t.Start();

if (MinisterOnTable[iIndex].MinisterThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(MinisterOnTable[iIndex].MinisterThinking[0].t); } }

}

}

catch (Exception t)

{

Log(t);

}

}

//Progressing.

if (KindIndex == -6)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (!AllDraw.Blitz)

{//When There is Current Brown Existing Objective Thinking Movments.

//Thinking Operations of Brown Current Objects.

KingOnTable[iIndex].KingThinking[0].ThinkingBegin = true;

KingOnTable[iIndex].KingThinking[0].ThinkingFinished = false;

KingOnTable[iIndex].KingThinking[0].t = new Task(new Action(KingOnTable[iIndex].KingThinking[0].Thinking));

KingOnTable[iIndex].KingThinking[0].t.Start();

if (KingOnTable[iIndex].KingThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(KingOnTable[iIndex].KingThinking[0].t); } }

}

else if (AllDraw.Blitz)//When There is Current Brown Existing Objective Thinking Movments.

{

//Thinking of Thinking Brown CurrentTable Objective Operations.

KingOnTable[iIndex].KingThinking[0].TableT = KingOnTable[iIndex].KingThinking[0].TableListKing[j];

KingOnTable[iIndex].KingThinking[0].ThinkingBegin = true;

KingOnTable[iIndex].KingThinking[0].ThinkingFinished = false;

KingOnTable[iIndex].KingThinking[0].t = new Task(new Action(KingOnTable[iIndex].KingThinking[0].Thinking));

KingOnTable[iIndex].KingThinking[0].t.Start();

if (KingOnTable[iIndex].KingThinking[0].t != null) { Object tttt = new Object(); lock (tttt) { tH.Add(KingOnTable[iIndex].KingThinking[0].t); } }

}

}

catch (Exception t)

{

KingOnTable[iIndex] = null;

Log(t);

}

}

try

{

//IncreaseprogressBarRefregitzValue(THIS.progressBarVerify, increasedProgress);

//THIS.progressBarVerify.Invalidate();

//SetprogressBarUpdate(THIS.progressBarVerify);

}

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

}

}

//Thread arrayT = new Thread(() => do\_check(tH));

//rayT.Start();

//ile (WaitSome) { Thread.Sleep(1000); }

{

try

{

/\*foreach (Task ij in tH)

{

ij.Start();

//Thread.Sleep(10);

}

\*/

Parallel.ForEach(tH, items => Task.WaitAll(items));

}

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

}

}

//while ((ThinkingChess.BeginThread) != (ThinkingChess.EndThread))

//{

}

bool FOUND = false;

if (KindIndex == 1 || KindIndex == -1)

{

SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy[SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Clear();

SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy[SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy[SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy[SolderesOnTable[iIndex].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

else

if (KindIndex == 2 || KindIndex == -2)

{

ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy[ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Clear();

ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy[ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy[ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy[ElephantOnTable[iIndex].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

else

if (KindIndex == 3 || KindIndex == -3)

{

HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy[HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy.Count - 1].TableList.Clear();

HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy[HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy[HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy[HoursesOnTable[iIndex].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

else

if (KindIndex == 4 || KindIndex == -4)

{

CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy[CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy.Count - 1].TableList.Clear();

CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy[CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy[CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy[CastlesOnTable[iIndex].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

else

if (KindIndex == 5 || KindIndex == -5)

{

MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Clear();

MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

else

if (KindIndex == 6 || KindIndex == -6)

{

KingOnTable[iIndex].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[iIndex].KingThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Clear();

KingOnTable[iIndex].KingThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(Tab));

KingOnTable[iIndex].KingThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

KingOnTable[iIndex].KingThinking[0].AStarGreedy[MinisterOnTable[iIndex].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, Tab, Order \* -1, false, FOUND, LeafAStarGreedy);

}

// }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

return this;//.CopyRemeiningItems(Dummy, Order,false);

//return

//>;

}

}

int MaxGrayMidle()

{

Object O = new Object();

lock (O)

{

int[] Tab = new int[6];

Tab[0] = SodierMidle;

Tab[1] = ElefantMidle;

Tab[2] = HourseMidle;

Tab[3] = CastleMidle;

Tab[4] = MinisterMidle;

Tab[5] = KingMidle;

int Max = 0;

for (int i = 0; i < 6; i++)

{

if (Tab[i] > Max)

Max = Tab[i];

}

return Max;

}

}

int MaxBrownHigh()

{

Object O = new Object();

lock (O)

{

int[] Tab = new int[6];

Tab[0] = SodierHigh;

Tab[1] = ElefantHigh;

Tab[2] = HourseHight;

Tab[3] = CastleHigh;

Tab[4] = MinisterHigh;

Tab[5] = KingHigh;

int Max = 0;

for (int i = 0; i < 6; i++)

{

if (Tab[i] > Max)

Max = Tab[i];

}

return Max;

}

}

int MinBrownMidle()

{

Object O = new Object();

lock (O)

{

int[] Tab = new int[6];

Tab[0] = SodierHigh;

Tab[1] = ElefantHigh;

Tab[2] = HourseHight;

Tab[3] = CastleHigh;

Tab[4] = MinisterHigh;

Tab[5] = KingHigh;

int Min = MaxBrownHigh();

for (int i = 0; i < 6; i++)

{

if (Tab[i] < Min)

Min = Tab[i];

}

return Min;

}

}

AllDraw InitiateAStarGreedytObjectGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy //, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

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

{

Parallel.Invoke(() =>

{

Object ooo = new Object();

lock (ooo)

{

if (SodierMidle > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If Solders Not Exist Continue and Traversal Back.

if (SolderesOnTable != null && SolderesOnTable[i] != null)

{

//Initiate of Local Variables By Global Objective Gray Current Solder.

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

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

//Construction of Thinking Gray Soldier By Local Variables.

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

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

//If There is no Thinking Movments on Current Object

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

{

//For All Movable Gray Solders.

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

//Parallel.For(0, AllDraw.SodierMovments, j =>

{

//Thinking of Gray Solder Operation.

Object OOO = new Object();

lock (OOO)

{

SolderesOnTable[i].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[i].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[i].SoldierThinking[0].Kind = 1;

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

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

}

}//);

}

}

}

catch (Exception t)

{

// SolderesOnTable[i] = null;

Log(t);

}

}

}

}

},

() =>

{

Object ooo = new Object();

lock (ooo)

{

if (ElefantMidle > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Elephant Gray Objects.

if (ElephantOnTable != null && ElephantOnTable[i] != null)

{

//Inititae Local Varibale By Global Gray Elephant Objects Varibales.

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

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

//Construction of Thinking Objects By Local Varibales.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

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

//If There is Not Thinking Objetive List Elephant Gray.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

{

//For All Possible Movments.

//Parallel.For(0, AllDraw.ElefantMovments, j =>

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

{

//Operational Thinking Gray Elephant.

Object OOO = new Object();

lock (OOO)

{

ElephantOnTable[i].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[i].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[i].ElefantThinking[0].Kind = 2;

ElephantOnTable[i].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[i].ElefantThinking[0].Thinking));

ElephantOnTable[i].ElefantThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},

() =>

{

Object ooo = new Object();

lock (ooo)

{

if (HourseMidle > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Gray Hourse Objects.

if (HoursesOnTable != null && HoursesOnTable[i] != null)

{

//Initiate of Local Variables By Global Gray Hourse Objectives.

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

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

//Construction of Gray Hourse Thinking Objects..

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

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

//When There is Not HourseList Count.

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

{

//For All Possible Movments.

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

//Parallel.For(0, AllDraw.HourseMovments, j =>

{

//Thinking of Gray Hourse Oprational.

Object OOO = new Object();

lock (OOO)

{

HoursesOnTable[i].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[i].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[i].HourseThinking[0].Kind = 3;

HoursesOnTable[i].HourseThinking[0].t = new Task(new Action(HoursesOnTable[i].HourseThinking[0].Thinking));

HoursesOnTable[i].HourseThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},

() =>

{

Object ooo = new Object();

lock (ooo)

{

if (CastleMidle > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//When Current Castles Gray Not Exist Continue Traversal Back.

if (CastlesOnTable != null && CastlesOnTable[i] != null)

{

//Initaiate of Local Varibales By Global Varoiables.

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

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

//Construction of Thinking Variables By Local Variables.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

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

//When Count of Table Castles of Thinking Not Exist Do Operational.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

{

//For All Possible Movments.

//Parallel.For(0, AllDraw.CastleMovments, j =>

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

{

Object OOO = new Object();

lock (OOO)

{

//Thinking of Gray Castles Operational.

CastlesOnTable[i].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[i].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[i].CastleThinking[0].Kind = 4;

CastlesOnTable[i].CastleThinking[0].t = new Task(new Action(CastlesOnTable[i].CastleThinking[0].Thinking));

CastlesOnTable[i].CastleThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},

() =>

{

Object ooo = new Object();

lock (ooo)

{

if (MinisterMidle > i)

{

try

{

Object O = new Object();

lock (O)

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//For Each Non Exist Gray Minister Objectives.

if (MinisterOnTable != null && MinisterOnTable[i] != null)

{

//Inititate Local Variables By Global Varibales.

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

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

//Construction of Thinking Objects Gray Minister.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

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

//If There is Not Minister Of Gray In The Thinking Table List.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

{

//For All Possible Movments.

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

//Parallel.For(0, AllDraw.MinisterMovments, j =>

{

//Thinking of Gray Minister Operational.

Object OOO = new Object();

lock (OOO)

{

MinisterOnTable[i].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[i].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[i].MinisterThinking[0].Kind = 5;

MinisterOnTable[i].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[i].MinisterThinking[0].Thinking));

MinisterOnTable[i].MinisterThinking[0].t.Start();

}

}//);

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

},

() =>

{

Object ooo = new Object();

lock (ooo)

{

if (KingMidle > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If There is Not Current Object Continue Traversal Back.

if (KingOnTable != null && KingOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Gray King Thinking Objects.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

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

//When There is Not Thinking Table Gray King Movments.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

{

//For All Possible Gray King Movments.

//Parallel.For(0, AllDraw.KingMovments, j =>

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

{

//Thinking Of Gray King Operatins.

Object OOO = new Object();

lock (OOO)

{

KingOnTable[i].KingThinking[0].ThinkingBegin = true;

KingOnTable[i].KingThinking[0].ThinkingFinished = false;

KingOnTable[i].KingThinking[0].Kind = 6;

KingOnTable[i].KingThinking[0].t = new Task(new Action(KingOnTable[i].KingThinking[0].Thinking));

KingOnTable[i].KingThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

});

}//);

}

return this;

}

AllDraw InitiateAStarGreedytObjectBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//Parallel.For(MinBrownMidle(), MaxBrownHigh(), i =>

for (int i = MinBrownMidle(); i < MaxBrownHigh(); i++)

{

Parallel.Invoke(() =>

{

Object ooo = new Object();

lock (ooo)

{

if (SodierMidle <= i && SodierHigh > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If Solders Not Exist Continue and Traversal Back.

if (SolderesOnTable != null && SolderesOnTable[i] != null)

{

//Initiate of Local Variables By Global Objective Gray Current Solder.

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

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

//Construction of Thinking Gray Soldier By Local Variables.

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

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

//If There is no Thinking Movments on Current Object

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

{

//For All Movable Gray Solders.

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

//Parallel.For(0, AllDraw.SodierMovments, j =>

{

//Thinking of Gray Solder Operation.

Object OOO = new Object();

lock (OOO)

{

SolderesOnTable[i].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[i].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[i].SoldierThinking[0].Kind = 1;

SolderesOnTable[i].SoldierThinking[0].t = new Task(new Action(SolderesOnTable[i].SoldierThinking[0].Thinking));

SolderesOnTable[i].SoldierThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

// SolderesOnTable[i] = null;

Log(t);

}

}

}

}

},() =>

{

Object oooo = new Object();

lock (oooo)

{

if (ElefantMidle <= i && ElefantHigh > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Elephant Gray Objects.

if (ElephantOnTable != null && ElephantOnTable[i] != null)

{

//Inititae Local Varibale By Global Gray Elephant Objects Varibales.

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

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

//Construction of Thinking Objects By Local Varibales.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

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

//If There is Not Thinking Objetive List Elephant Gray.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

{

//For All Possible Movments.

//Parallel.For(0, AllDraw.ElefantMovments, j =>

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

{

//Operational Thinking Gray Elephant.

Object OOO = new Object();

lock (OOO)

{

ElephantOnTable[i].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[i].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[i].ElefantThinking[0].Kind = 2;

ElephantOnTable[i].ElefantThinking[0].t = new Task(new Action(ElephantOnTable[i].ElefantThinking[0].Thinking));

ElephantOnTable[i].ElefantThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},() =>

{

Object oooo = new Object();

lock (oooo)

{

if (HourseMidle <= i && HourseHight > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Gray Hourse Objects.

if (HoursesOnTable != null && HoursesOnTable[i] != null)

{

//Initiate of Local Variables By Global Gray Hourse Objectives.

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

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

//Construction of Gray Hourse Thinking Objects..

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

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

//When There is Not HourseList Count.

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

{

//For All Possible Movments.

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

//Parallel.For(0, AllDraw.HourseMovments, j =>

{

//Thinking of Gray Hourse Oprational.

Object OOO = new Object();

lock (OOO)

{

HoursesOnTable[i].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[i].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[i].HourseThinking[0].Kind = 3;

HoursesOnTable[i].HourseThinking[0].t = new Task(new Action(HoursesOnTable[i].HourseThinking[0].Thinking));

HoursesOnTable[i].HourseThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},() =>

{

Object oooo = new Object();

lock (oooo)

{

if (CastleMidle <= i && CastleHigh < i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//When Current Castles Gray Not Exist Continue Traversal Back.

if (CastlesOnTable != null && CastlesOnTable[i] != null)

{

//Initaiate of Local Varibales By Global Varoiables.

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

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

//Construction of Thinking Variables By Local Variables.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

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

//When Count of Table Castles of Thinking Not Exist Do Operational.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

{

//For All Possible Movments.

//Parallel.For(0, AllDraw.CastleMovments, j =>

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

{

Object OOO = new Object();

lock (OOO)

{

//Thinking of Gray Castles Operational.

CastlesOnTable[i].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[i].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[i].CastleThinking[0].Kind = 4;

CastlesOnTable[i].CastleThinking[0].t = new Task(new Action(HoursesOnTable[i].HourseThinking[0].Thinking));

CastlesOnTable[i].CastleThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

},() =>

{

Object oooo = new Object();

lock (oooo)

{

try

{

if (MinisterMidle <= i && MinisterHigh > i)

{

Object O = new Object();

lock (O)

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//For Each Non Exist Gray Minister Objectives.

if (MinisterOnTable != null && MinisterOnTable[i] != null)

{

//Inititate Local Variables By Global Varibales.

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

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

//Construction of Thinking Objects Gray Minister.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

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

//If There is Not Minister Of Gray In The Thinking Table List.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

{

//For All Possible Movments.

//Parallel.For(0, AllDraw.MinisterMovments, j =>

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

{

//Thinking of Gray Minister Operational.

Object OOO = new Object();

lock (OOO)

{

MinisterOnTable[i].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[i].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[i].MinisterThinking[0].Kind = 5;

MinisterOnTable[i].MinisterThinking[0].t = new Task(new Action(MinisterOnTable[i].MinisterThinking[0].Thinking));

MinisterOnTable[i].MinisterThinking[0].t.Start();

}

}//);

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

},

() =>

{

Object oooo = new Object();

lock (oooo)

{

if (KingMidle <= i && KingHigh > i)

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If There is Not Current Object Continue Traversal Back.

if (KingOnTable != null && KingOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Gray King Thinking Objects.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

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

//When There is Not Thinking Table Gray King Movments.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

{

//For All Possible Gray King Movments.

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

{

//Thinking Of Gray King Operatins.

Object OOO = new Object();

lock (OOO)

{

KingOnTable[i].KingThinking[0].ThinkingBegin = true;

KingOnTable[i].KingThinking[0].ThinkingFinished = false;

KingOnTable[i].KingThinking[0].Kind = 6;

KingOnTable[i].KingThinking[0].t = new Task(new Action(KingOnTable[i].KingThinking[0].Thinking));

KingOnTable[i].KingThinking[0].t.Start();

}

}//);

}

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

});

}//);

}

return this;

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

AllDraw InitiateAStarGreedytSodlerGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

//List<Task> tH = new List<Task>();

Object oo = new Object();

lock (oo)

{

//For Gray Soldeirs Objects.

Parallel.For(0, SodierMidle, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If Solders Not Exist Continue and Traversal Back.

if (SolderesOnTable != null && SolderesOnTable[i] != null)

{

//Initiate of Local Variables By Global Objective Gray Current Solder.

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

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

//Construction of Thinking Gray Soldier By Local Variables.

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

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

//If There is no Thinking Movments on Current Object

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

{

//For All Movable Gray Solders.

Parallel.For(0, AllDraw.SodierMovments, j =>

{

//Thinking of Gray Solder Operation.

Object OOO = new Object();

lock (OOO)

{

SolderesOnTable[i].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[i].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[i].SoldierThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

// SolderesOnTable[i] = null;

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedytElephantGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(0, ElefantMidle, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Elephant Gray Objects.

if (ElephantOnTable != null && ElephantOnTable[i] != null)

{

//Inititae Local Varibale By Global Gray Elephant Objects Varibales.

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

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

//Construction of Thinking Objects By Local Varibales.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

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

//If There is Not Thinking Objetive List Elephant Gray.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

{

//For All Possible Movments.

Parallel.For(0, AllDraw.ElefantMovments, j =>

{

//Operational Thinking Gray Elephant.

Object OOO = new Object();

lock (OOO)

{

ElephantOnTable[i].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[i].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[i].ElefantThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythHourseGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

//For All Gray Hourse Objects.

Parallel.For(0, HourseMidle, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Ignore of Non Exist Current Gray Hourse Objects.

if (HoursesOnTable != null && HoursesOnTable[i] != null)

{

//Initiate of Local Variables By Global Gray Hourse Objectives.

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

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

//Construction of Gray Hourse Thinking Objects..

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

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

//When There is Not HourseList Count.

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

{

//For All Possible Movments.

Parallel.For(0, AllDraw.HourseMovments, j =>

{

//Thinking of Gray Hourse Oprational.

Object OOO = new Object();

lock (OOO)

{

HoursesOnTable[i].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[i].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[i].HourseThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythCastleGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

//For All Possible Gray Castles Objects.

Parallel.For(0, CastleMidle, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//When Current Castles Gray Not Exist Continue Traversal Back.

if (CastlesOnTable != null && CastlesOnTable[i] != null)

{

//Initaiate of Local Varibales By Global Varoiables.

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

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

//Construction of Thinking Variables By Local Variables.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

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

//When Count of Table Castles of Thinking Not Exist Do Operational.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

{

//For All Possible Movments.

Parallel.For(0, AllDraw.CastleMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking of Gray Castles Operational.

CastlesOnTable[i].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[i].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[i].CastleThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythMinisterGray(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

//For All Possible Gray Minister Movments.

Parallel.For(0, MinisterMidle, i =>

{

try

{

Object O = new Object();

lock (O)

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//For Each Non Exist Gray Minister Objectives.

if (MinisterOnTable != null && MinisterOnTable[i] != null)

{

//Inititate Local Variables By Global Varibales.

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

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

//Construction of Thinking Objects Gray Minister.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

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

//If There is Not Minister Of Gray In The Thinking Table List.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

{

//For All Possible Movments.

Parallel.For(0, AllDraw.MinisterMovments, j =>

{

//Thinking of Gray Minister Operational.

Object OOO = new Object();

lock (OOO)

{

MinisterOnTable[i].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[i].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[i].MinisterThinking[0].Thinking();

}

});

}

}

}

}

catch (Exception t)

{

Log(t);

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythKingGray(int iii, int jjjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

//For All Possible Gray King Objects.

Parallel.For(0, KingMidle, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//If There is Not Current Object Continue Traversal Back.

if (KingOnTable != null && KingOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Gray King Thinking Objects.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

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

//When There is Not Thinking Table Gray King Movments.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

{

//For All Possible Gray King Movments.

Parallel.For(0, AllDraw.KingMovments, j =>

{

//Thinking Of Gray King Operatins.

Object OOO = new Object();

lock (OOO)

{

KingOnTable[i].KingThinking[0].ThinkingBegin = true;

KingOnTable[i].KingThinking[0].ThinkingFinished = false;

KingOnTable[i].KingThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

// KingOnTable[i] = null;

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythSoldierBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

//For Each Objects of Brown Sodiers.

Parallel.For(SodierMidle, SodierHigh, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

//Wheen Brown King Object There is Not Continue Traversal Back.

if (SolderesOnTable != null && SolderesOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

{

//When There is Current Brown Object Table List Thinking Objective Movments.

if (SolderesOnTable[i].SoldierThinking[0].TableListSolder.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.SodierMovments, j =>

{

//Thinking Operations of Brown Current Objects.

Object OOO = new Object();

lock (OOO)

{

SolderesOnTable[i].SoldierThinking[0].ThinkingBegin = true;

SolderesOnTable[i].SoldierThinking[0].ThinkingFinished = false;

SolderesOnTable[i].SoldierThinking[0].Thinking();

}

});

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythElephantBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(ElefantMidle, ElefantHigh, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (ElephantOnTable != null && ElephantOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

{

//When There is Current Brown Object Table List Thinking Objective Movments.

if (ElephantOnTable[i].ElefantThinking[0].TableListElefant.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.ElefantMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking Operations of Brown Current Objects.

ElephantOnTable[i].ElefantThinking[0].ThinkingBegin = true;

ElephantOnTable[i].ElefantThinking[0].ThinkingFinished = false;

ElephantOnTable[i].ElefantThinking[0].Thinking();

}

});

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythHourseBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(HourseMidle, HourseHight, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (HoursesOnTable != null && HoursesOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

{

//When There is Current Brown Object Table List Thinking Objective Movments.

if (HoursesOnTable[i].HourseThinking[0].TableListHourse.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.HourseMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking Operations of Brown Current Objects.

//HoursesOnTable[i].HourseThinking[0].TableT = HoursesOnTable[i].HourseThinking[0].TableT;

HoursesOnTable[i].HourseThinking[0].ThinkingBegin = true;

HoursesOnTable[i].HourseThinking[0].ThinkingFinished = false;

HoursesOnTable[i].HourseThinking[0].Thinking();

}

});

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythCastleBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(CastleMidle, CastleHigh, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (CastlesOnTable != null && CastlesOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

{

//When There is Current Brown Object Table List Thinking Objective Movments.

if (CastlesOnTable[i].CastleThinking[0].TableListCastle.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.CastleMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking Operations of Brown Current Objects.

CastlesOnTable[i].CastleThinking[0].ThinkingBegin = true;

CastlesOnTable[i].CastleThinking[0].ThinkingFinished = false;

CastlesOnTable[i].CastleThinking[0].Thinking();

}

});

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythMinisterBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(MinisterMidle, MinisterHigh, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (MinisterOnTable != null && MinisterOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

//When There is Current Brown Object Table List Thinking Objective Movments.

if (MinisterOnTable[i].MinisterThinking[0].TableListMinister.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.MinisterMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking Operations of Brown Current Objects.

MinisterOnTable[i].MinisterThinking[0].ThinkingBegin = true;

MinisterOnTable[i].MinisterThinking[0].ThinkingFinished = false;

MinisterOnTable[i].MinisterThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

AllDraw InitiateAStarGreedythKingBrown(int iii, int jjj, int[,] Table, int DummyOrder, int DummyCurrentOrder, int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

//List<Task> tH = new List<Task>();

Parallel.For(KingMidle, KingHigh, i =>

{

Object O = new Object();

lock (O)

{

try

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (KingOnTable != null && KingOnTable[i] != null)

{

//Initiate Local varibale By Global Objective Varibales.

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

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

//Construction of Thinking Brown Current Objects.

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

//When There is Current Brown Object Table List Thinking Objective Movments.

if (KingOnTable[i].KingThinking[0].TableListKing.Count == 0)

{

//For Each Brown Possible Movments.

Parallel.For(0, AllDraw.KingMovments, j =>

{

Object OOO = new Object();

lock (OOO)

{

//Thinking Operations of Brown Current Objects.

KingOnTable[i].KingThinking[0].ThinkingBegin = true;

KingOnTable[i].KingThinking[0].ThinkingFinished = false;

KingOnTable[i].KingThinking[0].Thinking();

}

});

}

}

}

catch (Exception t)

{

//KingOnTable[i] = null;

Log(t);

}

}

});

}

//Parallel.ForEach(tH, items => Task.WaitAny(items));

return this;

}

//AStarGreedy First Initiat Thinking Main Method.

public AllDraw InitiateAStarGreedyt(int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

Object oo = new Object();

lock (oo)

{

OrderP = Order;

SetObjectNumbers(Tab);

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

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

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

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

ThinkingChess.BeginThread = 0;

ThinkingChess.EndThread = 0;

//Initiate of global Variables Byte Local Variables.

int DummyOrder = new int();

DummyOrder = Order;

int DummyCurrentOrder = new int();

DummyCurrentOrder = ChessRules.CurrentOrder;

List<Task> ThB = new List<Task>();

int i = 0, ik = 0;

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

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int j = 0;

//if (AllDraw.Blitz)

{

//if (iAStarGreedy <= 0)

//return null;

if (ThinkingChess.FoundFirstMating >= MaxAStarGreedy)

return null;

}

CurrentAStarGredyMax = AStarGreedyiLevelMax - iAStarGreedy;

iAStarGreedy--;

bool Do = false;

if (iAStarGreedy >= 0 && iAStarGreedy < MaxDuringLevelThinkingCreation)

{

MaxDuringLevelThinkingCreation = iAStarGreedy;

Object O = new Object();

DepthIterative++;

lock (O)

{

OutPut = "\r\nMinimum Level During Thinking Tree Creation is " + MaxDuringLevelThinkingCreation.ToString() + "at Iterative " + DepthIterative.ToString();

}

}

try

{

if (!FOUND)

{

//Initiate Of Local Variables.

Object o = new Object();

lock (o)

{

//If Order is Gray.

if (Order == 1)

{

int i1 = i, j1 = j;

int[,] Tabl = CloneATable(Table);

int DummyOrder1 = DummyOrder, DummyCurrentOrder1 = DummyCurrentOrder, iAStarGreedy1 = iAStarGreedy, ii1 = ii, jj1 = jj, Ord1 = Order;

bool TB1 = TB;

Color aa = a;

Parallel.Invoke(() =>

{

//For All Gray Soldier Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedytSodlerGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

,

() =>

{

//For All Gray Elephant Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedytElephantGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

//For All Gray Hourse Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythHourseGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

,

() =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythCastleGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythMinisterGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythKingGray(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

);

}

else//Brown Order Considarations.

{

int i1 = i, j1 = j;

int[,] Tabl = CloneATable(Table);

int DummyOrder1 = DummyOrder, DummyCurrentOrder1 = DummyCurrentOrder, iAStarGreedy1 = iAStarGreedy, ii1 = ii, jj1 = jj, Ord1 = Order;

bool TB1 = TB;

Color aa = a;

//If Order is Gray.

Parallel.Invoke(() =>

{

//For All Gray Soldier Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythSoldierBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

,

() =>

{

//For All Gray Elephant Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythElephantBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

//For All Gray Hourse Objects.

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythHourseBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

,

() =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythCastleBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythMinisterBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

, () =>

{

Object O = new Object();

lock (O)

{

this.InitiateAStarGreedythKingBrown(i1, j1, Tabl, DummyOrder1, DummyCurrentOrder1, iAStarGreedy1, ii1, jj1, aa, Tab, Ord1, TB1, FOUND, LeafAStarGreedy);

}

}

);

}

}

}

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (FOUND)

{

Object O = new Object();

lock (O)

{

Tabl = CloneATable(Table);

FoundOfLeafDepenOfKindFullGame(Tabl, Order, iAStarGreedy, ii, jj, ik, j, FOUND, LeafAStarGreedy);

}

}

else

{

Object O = new Object();

lock (O)

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

int Ord = Order \* -1, iAStarGreedy1 = iAStarGreedy, ii1 = ii, jj1 = jj, ik1 = ik, j1 = j;

//System.Threading.Thread.Sleep(100);

Parallel.Invoke(() =>

{

Do = this.FullGameThinkingTree(Ord, iAStarGreedy1, ii1, jj1, ik1, j1, false, LeafAStarGreedy);

});

}

if (!Do)

if (iAStarGreedy < MinThinkingTreeDepth)

MinThinkingTreeDepth = iAStarGreedy;

}

}

catch (Exception t)

{

Log(t);

Object O = new Object();

lock (O)

{

OutPut = "\r\nPossibly Aggregation need.";

}

}

return this;

}

}

public AllDraw InitiateAStarGreedytObject(int iAStarGreedy, int ii, int jj, Color a, int[,] Tab, int Order, bool TB, bool FOUND, int LeafAStarGreedy//, ref Refrigtz.Timer timer, ref Refrigtz.Timer Timerint, ref double Less

)

{

bool Do = false;

Object oo = new Object();

lock (oo)

{

OrderP = Order;

SetObjectNumbers(Tab);

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

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

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

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

ThinkingChess.BeginThread = 0;

ThinkingChess.EndThread = 0;

//Initiate of global Variables Byte Local Variables.

int DummyOrder = new int();

DummyOrder = Order;

int DummyCurrentOrder = new int();

DummyCurrentOrder = ChessRules.CurrentOrder;

List<Task> ThB = new List<Task>();

int i = 0, ik = 0;

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

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int j = 0;

//if (iAStarGreedy <= 0)

//return null;

if (ThinkingChess.FoundFirstMating >= MaxAStarGreedy)

{

OutPut = "\r\nCheckedMate Boundry Conditon in Leafs found at " + ThinkingChess.FoundFirstMating.ToString();

return null;

}

CurrentAStarGredyMax = AStarGreedyiLevelMax - iAStarGreedy;

iAStarGreedy--;

if (iAStarGreedy >= 0 && iAStarGreedy < MaxDuringLevelThinkingCreation)

{

MaxDuringLevelThinkingCreation = iAStarGreedy;

Object O = new Object();

DepthIterative++;

lock (O)

{

OutPut = "\r\nMinimum Level During Thinking Tree Creation is " + MaxDuringLevelThinkingCreation.ToString() + "at Iterative " + DepthIterative.ToString();

}

//THIS.RefreshBoxText();

}

if (!FOUND)

{

Object o = new Object();

lock (o)

{

if (Order == 1)

InitiateAStarGreedytObjectGray(i, j, Table, DummyOrder, DummyCurrentOrder, iAStarGreedy, ii, jj, a, Tab, Order, TB, FOUND, LeafAStarGreedy);

else

InitiateAStarGreedytObjectBrown(i, j, Table, DummyOrder, DummyCurrentOrder, iAStarGreedy, ii, jj, a, Tab, Order, TB, FOUND, LeafAStarGreedy);

}

}

if (FOUND)

{

Object O = new Object();

lock (O)

{

Tabl = CloneATable(Table);

FoundOfLeafDepenOfKindFullGame(Tabl, Order, iAStarGreedy, ii, jj, ik, j, FOUND, LeafAStarGreedy);

}

}

else

{

Object O = new Object();

lock (O)

{

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

int Ord = Order \* -1, iAStarGreedy1 = iAStarGreedy, ii1 = ii, jj1 = jj, ik1 = ik, j1 = j;

//int Ord = Order, iAStarGreedy1 = iAStarGreedy, ii1 = ii, jj1 = jj, ik1 = ik, j1 = j;

//System.Threading.Thread.Sleep(100);

//Parallel.Invoke(() =>

{

Do |= this.FullGameThinkingTree(Ord, iAStarGreedy1, ii1, jj1, ik1, j1, false, LeafAStarGreedy);

}//);

}

}

if (!Do)

if (iAStarGreedy < MinThinkingTreeDepth)

MinThinkingTreeDepth = iAStarGreedy;

return this;

}

}

void BlitzGameThinkingTreeSolderGray(ref double PreviousLessS, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Soldeir

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

try

{

if (SolderesOnTable == null || SolderesOnTable[ik] == null)

continue;

for (j = 0; j < SolderesOnTable[ik].SoldierThinking[0].HuristicListSolder.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessS || SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() == 0)

{

}

else

{

PreviousLessS = SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false);

Index[0] = ik;

jIndex[0] = j;

}

}

else

{

if (SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessS || SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() == 0)

{

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = null;

//SolderesOnTable[ik] = null;

//continue;

}

else

{

PreviousLessS = SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false);

Index[0] = ik;

jIndex[0] = j;

}

}

}

}

//Elephant

}

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

}

void BlitzGameThinkingTreeElephantGray(ref double PreviousLessE, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Elephant

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

try

{

if (ElephantOnTable == null || ElephantOnTable[ik] == null)

continue;

for (j = 0; j < ElephantOnTable[ik].ElefantThinking[0].HuristicListElefant.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessE || ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() == 0)

{

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = null;

//ElephantOnTable[ik] = null;

//continue;

}

else

{

PreviousLessE = ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false);

Index[1] = ik;

jIndex[1] = j;

}

}

else

{

if (ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessE || ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() == 0)

{

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = null;

//ElephantOnTable[ik] = null;

//continue;

}

else

{

PreviousLessE = ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false);

Index[1] = ik;

jIndex[1] = j;

}

}

}

}

}

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

}

void BlitzGameThinkingTreeHourseGray(ref double PreviousLessH, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Hourse.

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

try

{

if (HoursesOnTable == null || HoursesOnTable[ik] == null)

continue;

for (j = 0; j < HoursesOnTable[ik].HourseThinking[0].HuristicListHourse.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessH || HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() == 0)

{

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy = null;

//continue;

}

else

{

PreviousLessH = HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false);

Index[2] = ik;

jIndex[2] = j;

}

}

else

{

if (HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessH || HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() == 0)

{

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy = null;

//continue;

}

else

{

PreviousLessH = HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false);

Index[2] = ik;

jIndex[2] = j;

}

}

}

}

}

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

}

void BlitzGameThinkingTreeCastleGray(ref double PreviousLessB, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

//Castle.

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

try

{

if (CastlesOnTable == null || CastlesOnTable[ik] == null)

continue;

for (j = 0; j < CastlesOnTable[ik].CastleThinking[0].HuristicListCastle.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessB || CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() == 0)

{

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy = null;

//CastlesOnTable[ik] = null;

//continue;

}

else

{

PreviousLessB = CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false);

Index[3] = ik;

jIndex[3] = j;

}

}

else

{

if (CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessB || CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() == 0)

{

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy = null;

//CastlesOnTable[ik] = null;

//continue;

}

else

{

PreviousLessB = CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false);

Index[3] = ik;

jIndex[3] = j;

}

}

}

}

}

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

}

void BlitzGameThinkingTreeMinisterGray(ref double PreviousLessM, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Minister.

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

try

{

if (MinisterOnTable == null || MinisterOnTable[ik] == null)

continue;

for (j = 0; j < MinisterOnTable[ik].MinisterThinking[0].HuristicListMinister.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessM || MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = null;

// MinisterOnTable[ik] = null;

// continue;

}

else

{

Index[4] = ik;

jIndex[4] = j;

PreviousLessM = MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

else

{

if (MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessM || MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = null;

// MinisterOnTable[ik] = null;

// continue;

}

else

{

Index[4] = ik;

jIndex[4] = j;

PreviousLessM = MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

}

}

}

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

}

void BlitzGameThinkingTreeKingGray(ref double PreviousLessK, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //King.

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

try

{

if (KingOnTable == null || KingOnTable[ik] == null)

continue;

for (j = 0; j < KingOnTable[ik].KingThinking[0].HuristicListKing.Count; j++)

{

Object O = new Object();

lock (O)

{

if (AllDraw.OrderPlate == Order)

{

if (KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessK || KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() == 0)

{

//KingOnTable[ik].KingThinking[0].AStarGreedy = null;

//KingOnTable[ik] = null;

//continue;

}

else

{

Index[5] = ik;

jIndex[5] = j;

PreviousLessK = KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

else

{

if (KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessK || KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() == 0)

{

//KingOnTable[ik].KingThinking[0].AStarGreedy = null;

//KingOnTable[ik] = null;

//continue;

}

else

{

Index[5] = ik;

jIndex[5] = j;

PreviousLessK = KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

}

}

}

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

}

void BlitzGameTreeCreationThinkingTreeSolder(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[0] != -1)

{

if (SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count == 0)

SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Clear();

SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Add(SolderesOnTable[Index[0]].SoldierThinking[0].TableListSolder[jIndex[0]]);

SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, SolderesOnTable[Index[0]].SoldierThinking[0].TableListSolder[jIndex[0]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy[SolderesOnTable[Index[0]].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, SolderesOnTable[Index[0]].SoldierThinking[0].RowColumnSoldier[jIndex[0]][0], SolderesOnTable[Index[0]].SoldierThinking[0].RowColumnSoldier[jIndex[0]][1], a, SolderesOnTable[Index[0]].SoldierThinking[0].TableListSolder[jIndex[0]], Order, false, FOUND, LeafAStarGreedy));

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

array.Wait();

//array.Name = "S" + i.ToString();

//array.Start();

}

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

void BlitzGameTreeCreationThinkingTreeElephant(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[1] != -1)

{

if (ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count == 0)

ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Clear();

ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Add(ElephantOnTable[Index[1]].ElefantThinking[0].TableListElefant[jIndex[1]]);

ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, ElephantOnTable[Index[1]].ElefantThinking[0].TableListElefant[jIndex[1]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy[ElephantOnTable[Index[1]].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ElephantOnTable[Index[1]].ElefantThinking[0].RowColumnElefant[jIndex[1]][0], ElephantOnTable[Index[1]].ElefantThinking[0].RowColumnElefant[jIndex[1]][1], a, ElephantOnTable[Index[1]].ElefantThinking[0].TableListElefant[jIndex[1]], Order, false, FOUND, LeafAStarGreedy));

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

//array.Name = "E" + i.ToString();

//array.Start();

array.Wait();

}

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

void BlitzGameTreeCreationThinkingTreeHourse(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[2] != -1)

{

if (HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count == 0)

HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].TableList.Clear(); ;

HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].TableList.Add(HoursesOnTable[Index[2]].HourseThinking[0].TableListHourse[jIndex[2]]);

HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, HoursesOnTable[Index[2]].HourseThinking[0].TableListHourse[jIndex[2]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy[HoursesOnTable[Index[2]].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, HoursesOnTable[Index[2]].HourseThinking[0].RowColumnHourse[jIndex[2]][0], HoursesOnTable[Index[2]].HourseThinking[0].RowColumnHourse[jIndex[2]][1], a, HoursesOnTable[Index[2]].HourseThinking[0].TableListHourse[jIndex[2]], Order, false, FOUND, LeafAStarGreedy));

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

//array.Name = "H" + i.ToString();

//array.Start();

array.Wait();

}

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

void BlitzGameTreeCreationThinkingTreeCastle(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[3] != -1)

{

if (CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count == 0)

CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].TableList.Clear();

CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].TableList.Add(CastlesOnTable[Index[3]].CastleThinking[0].TableListCastle[jIndex[3]]);

CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CastlesOnTable[Index[3]].CastleThinking[0].TableListCastle[jIndex[3]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy[CastlesOnTable[Index[3]].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, CastlesOnTable[Index[3]].CastleThinking[0].RowColumnCastle[jIndex[3]][0], CastlesOnTable[Index[3]].CastleThinking[0].RowColumnCastle[jIndex[3]][1], a, CastlesOnTable[Index[3]].CastleThinking[0].TableListCastle[jIndex[3]], Order, false, FOUND, LeafAStarGreedy));

array.Wait();

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

//array.Name = "B" + i.ToString();

//array.Start();

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

}

void BlitzGameTreeCreationThinkingTreeMinister(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[4] != -1)

{

if (MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count == 0)

MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Clear();

MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Add(MinisterOnTable[Index[4]].MinisterThinking[0].TableListMinister[jIndex[4]]);

MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, MinisterOnTable[Index[4]].MinisterThinking[0].TableListMinister[jIndex[4]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy[MinisterOnTable[Index[4]].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, MinisterOnTable[Index[4]].MinisterThinking[0].RowColumnMinister[jIndex[4]][0], MinisterOnTable[Index[4]].MinisterThinking[0].RowColumnMinister[jIndex[4]][1], a, MinisterOnTable[Index[4]].MinisterThinking[0].TableListMinister[jIndex[4]], Order, false, FOUND, LeafAStarGreedy));

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

//array.Name = "M" + i.ToString();

//array.Start();

array.Wait();

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

}

void BlitzGameTreeCreationThinkingTreeKing(Color a, int[] Index, int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

List<Task> tHA = new List<Task>();

Object O1 = new Object();

lock (O1)

{

if (Index[5] != -1)

{

if (KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count == 0)

KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].TableList.Clear();

KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].TableList.Add(KingOnTable[Index[5]].KingThinking[0].TableListKing[jIndex[5]]);

KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

//KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, KingOnTable[Index[5]].KingThinking[0].TableListKing[jIndex[5]], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

Task array = Task.Factory.StartNew(() => KingOnTable[Index[5]].KingThinking[0].AStarGreedy[KingOnTable[Index[5]].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, KingOnTable[Index[5]].KingThinking[0].RowColumnKing[jIndex[5]][0], KingOnTable[Index[5]].KingThinking[0].RowColumnKing[jIndex[5]][1], a, KingOnTable[Index[5]].KingThinking[0].TableListKing[jIndex[5]], Order, false, FOUND, LeafAStarGreedy));

//Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

//array.Name = "K" + i.ToString();

//array.Start();

array.Wait();

}

//Parallel.ForEach(tHA, items => Task.WaitAny(items));

}

}

void BlitzGameThinkingTreeSolderBrown(ref double PreviousLessS, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{

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

try

{

if (SolderesOnTable == null || SolderesOnTable[ik] == null)

continue;

//Soldier.

for (j = 0; j < SolderesOnTable[ik].SoldierThinking[0].HuristicListSolder.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessS || SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() == 0)

{

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = null;

//SolderesOnTable[ik] = null;

//continue;

}

else

{

Index[0] = ik;

jIndex[0] = j;

PreviousLessS = SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

else

{

if (SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessS || SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() == 0)

{

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = null;

//SolderesOnTable[ik] = null;

//continue;

}

else

{

Index[0] = ik;

jIndex[0] = j;

PreviousLessS = SolderesOnTable[ik].SoldierThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

}

}

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

}

void BlitzGameThinkingTreeElephantBrown(ref double PreviousLessE, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Elephant

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

try

{

if (ElephantOnTable == null || ElephantOnTable[ik] == null)

continue;

for (j = 0; j < ElephantOnTable[ik].ElefantThinking[0].HuristicListElefant.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessE || ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() == 0)

{

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = null;

//ElephantOnTable[ik] = null;

//continue;

}

else

{

Index[1] = ik;

jIndex[1] = j;

PreviousLessE = ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

else

{

if (ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessE || ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() == 0)

{

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = null;

//ElephantOnTable[ik] = null;

//continue;

}

else

{

Index[1] = ik;

jIndex[1] = j;

PreviousLessE = ElephantOnTable[ik].ElefantThinking[0].ReturnHuristic(-1, j, Order, false);

}

}

}

}

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

}

void BlitzGameThinkingTreeHourseBrown(ref double PreviousLessH, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Hourse.

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

try

{

if (HoursesOnTable == null || HoursesOnTable[ik] == null)

continue;

for (j = 0; j < HoursesOnTable[ik].HourseThinking[0].HuristicListHourse.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessH || HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() == 0)

{

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy = null;

//HoursesOnTable[ik] = null;

//continue;

}

else

{

Index[2] = ik;

jIndex[2] = j;

PreviousLessH = HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

else

{

if (HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessH || HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() == 0)

{

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy = null;

//HoursesOnTable[ik] = null;

//continue;

}

else

{

Index[2] = ik;

jIndex[2] = j;

PreviousLessH = HoursesOnTable[ik].HourseThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

}

}

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

}

int FullGameMakimgBlitz(ref int[] Index, ref int[] jIndex, int Order, int LeafAStarGreedy)

{

int Kind = -1;

double PS = Double.MinValue, PE = Double.MinValue, PH = Double.MinValue, PB = Double.MinValue, PM = Double.MinValue, PK = Double.MinValue;

if (Order == AllDraw.OrderPlate)

{

PS = Double.MaxValue;

PE = Double.MaxValue;

PH = Double.MaxValue;

PB = Double.MaxValue;

PM = Double.MaxValue;

PK = Double.MaxValue;

}

int[] index = new int[6];

int[] jindex = new int[6];

if (Order == -1)

{

Object O = new Object();

lock (O)

{

BlitzGameThinkingTreeSolderGray(ref PS, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeElephantGray(ref PE, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeHourseGray(ref PH, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeCastleGray(ref PB, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeMinisterGray(ref PM, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeKingGray(ref PK, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

}

}

else

{

Object O = new Object();

lock (O)

{

BlitzGameThinkingTreeSolderBrown(ref PS, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeElephantBrown(ref PE, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeHourseBrown(ref PH, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeCastleBrown(ref PB, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeMinisterBrown(ref PM, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

BlitzGameThinkingTreeKingBrown(ref PK, ref index, ref jindex, Order \* -1, 0, 0, 0, false, LeafAStarGreedy);

}

}

int JI = -1;

Object O1 = new Object();

lock (O1)

{

JI = MaxOfThreeHuristic(PS, PE, PH, PB, PM, PK);

}

if (JI != -1)

{

Kind = JI;

for (int i = 0; i < 6; i++)

{

Object O = new Object();

lock (O)

{

Index[i] = index[i];

jIndex[i] = jindex[i];

}

}

}

return System.Math.Abs(Kind);

}

void BlitzGameThinkingTreeCastleBrown(ref double PreviousLessB, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Castles.

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

try

{

if (CastlesOnTable == null || CastlesOnTable[ik] == null)

continue;

for (j = 0; j < CastlesOnTable[ik].CastleThinking[0].HuristicListCastle.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessB || CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() == 0)

{

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy = null;

//CastlesOnTable[ik] = null;

//continue;

}

else

{

Index[3] = ik;

jIndex[3] = j;

PreviousLessB = CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

else

{

if (CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessB || CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() == 0)

{

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy = null;

//CastlesOnTable[ik] = null;

//continue;

}

else

{

Index[3] = ik;

jIndex[3] = j;

PreviousLessB = CastlesOnTable[ik].CastleThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

}

}

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

}

void BlitzGameThinkingTreeMinisterBrown(ref double PreviousLessM, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Minister.

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

try

{

if (MinisterOnTable == null || MinisterOnTable[ik] == null)

continue;

for (j = 0; j < MinisterOnTable[ik].MinisterThinking[0].HuristicListMinister.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessM || MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = null;

//MinisterOnTable[ik] = null;

//continue;

}

else

{

Index[4] = ik;

jIndex[4] = j;

PreviousLessM = MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

else

{

if (MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessM || MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() == 0)

{

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = null;

//MinisterOnTable[ik] = null;

//continue;

}

else

{

Index[4] = ik;

jIndex[4] = j;

PreviousLessM = MinisterOnTable[ik].MinisterThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

}

}

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

}

void BlitzGameThinkingTreeKingBrown(ref double PreviousLessK, ref int[] Index, ref int[] jIndex, int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //King.

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

try

{

if (KingOnTable == null || KingOnTable[ik] == null)

continue;

for (j = 0; j < KingOnTable[ik].KingThinking[0].HuristicListKing.Count; j++)

{

if (AllDraw.OrderPlate == Order)

{

if (KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false) < PreviousLessK || KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() == 0)

{

//KingOnTable[ik].KingThinking[0].AStarGreedy = null;

//KingOnTable[ik] = null;

//continue;

}

else

{

Index[5] = ik;

jIndex[5] = j;

PreviousLessK = KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

else

{

if (KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false) > PreviousLessK || KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() == 0)

{

//KingOnTable[ik].KingThinking[0].AStarGreedy = null;

//KingOnTable[ik] = null;

//continue;

}

else

{

Index[5] = ik;

jIndex[5] = j;

PreviousLessK = KingOnTable[ik].KingThinking[0].ReturnHuristic(-1, j, Order, false); ;

}

}

}

}

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

}

void BlitzGameThinkingTree(int Order, int iAStarGreedy, int ik, int j, bool FOUND, int LeafAStarGreedy)

{ //Initiatye Variables.

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

Color a;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int[] Index = new int[6];

int[] jIndex = new int[6];

double PreviousLessS = Double.MinValue, PreviousLessE = Double.MinValue, PreviousLessH = Double.MinValue, PreviousLessB = Double.MinValue, PreviousLessM = Double.MinValue, PreviousLessK = Double.MinValue;

//For Gray Order calculating foreach Objects Maximum total Huristic Count Incl;usively.

if (Order == 1)

{

Object O1 = new Object();

lock (O1)

{

Index[0] = -1;

BlitzGameThinkingTreeSolderGray(ref PreviousLessS, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[1] = -1;

BlitzGameThinkingTreeElephantGray(ref PreviousLessE, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[2] = -1;

BlitzGameThinkingTreeHourseGray(ref PreviousLessH, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[3] = -1;

BlitzGameThinkingTreeCastleGray(ref PreviousLessB, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[4] = -1;

BlitzGameThinkingTreeMinisterGray(ref PreviousLessM, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[5] = -1;

BlitzGameThinkingTreeKingGray(ref PreviousLessK, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

}

int JI = -1;

Object O2 = new Object();

lock (O2)

{

JI = MaxOfThreeHuristic(PreviousLessS, PreviousLessE, PreviousLessH, PreviousLessB, PreviousLessM, PreviousLessK);

}

Object O3 = new Object();

lock (O3)

{

if (JI != -1)

{

if (JI == 0)

BlitzGameTreeCreationThinkingTreeSolder(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 1)

BlitzGameTreeCreationThinkingTreeElephant(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 2)

BlitzGameTreeCreationThinkingTreeHourse(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 3)

BlitzGameTreeCreationThinkingTreeCastle(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 4)

BlitzGameTreeCreationThinkingTreeMinister(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 5)

BlitzGameTreeCreationThinkingTreeKing(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

}

}

}

//For Brown Order Blitz Game Calculate Maximum Huristic Inclusive AStarGreedy First Game Search.

else

{

Object O1 = new Object();

lock (O1)

{

Index[0] = -1;

BlitzGameThinkingTreeSolderBrown(ref PreviousLessS, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[1] = -1;

BlitzGameThinkingTreeElephantBrown(ref PreviousLessE, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[2] = -1;

BlitzGameThinkingTreeHourseBrown(ref PreviousLessH, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[3] = -1;

BlitzGameThinkingTreeCastleBrown(ref PreviousLessB, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[4] = -1;

BlitzGameThinkingTreeMinisterBrown(ref PreviousLessM, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

Index[5] = -1;

BlitzGameThinkingTreeKingBrown(ref PreviousLessK, ref Index, ref jIndex, Order, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

}

int JI = -1;

Object O2 = new Object();

lock (O2)

{

JI = MaxOfThreeHuristic(PreviousLessS, PreviousLessE, PreviousLessH, PreviousLessB, PreviousLessM, PreviousLessK);

}

Object O3 = new Object();

lock (O3)

{

if (JI != -1)

{

BlitzGameTreeCreationThinkingTreeSolder(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

//if (JI == 1)

BlitzGameTreeCreationThinkingTreeElephant(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 2)

BlitzGameTreeCreationThinkingTreeHourse(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 3)

BlitzGameTreeCreationThinkingTreeCastle(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 4)

BlitzGameTreeCreationThinkingTreeMinister(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

if (JI == 5)

BlitzGameTreeCreationThinkingTreeKing(a, Index, jIndex, Order \* -1, iAStarGreedy, ik, j, FOUND, LeafAStarGreedy);

}

}

}

}

String Alphabet(int RowRealesed)

{

String A = "";

if (RowRealesed == 0)

A = "a";

else

if (RowRealesed == 1)

A = "b";

else

if (RowRealesed == 2)

A = "c";

else

if (RowRealesed == 3)

A = "d";

else

if (RowRealesed == 4)

A = "e";

else

if (RowRealesed == 5)

A = "f";

else

if (RowRealesed == 6)

A = "g";

else

if (RowRealesed == 7)

A = "h";

return A;

}

String Number(int ColumnRealeased)

{

String A = "";

if (ColumnRealeased == 7)

A = "0";

else

if (ColumnRealeased == 6)

A = "1";

else

if (ColumnRealeased == 5)

A = "2";

else

if (ColumnRealeased == 4)

A = "3";

else

if (ColumnRealeased == 3)

A = "4";

else

if (ColumnRealeased == 2)

A = "5";

else

if (ColumnRealeased == 1)

A = "6";

else

if (ColumnRealeased == 0)

A = "7";

return A;

}

bool FullGameThinkingTreeSoldier(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (SolderesOnTable[ik].SoldierThinking[0].ThinkingBegin && (!SolderesOnTable[ik].SoldierThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

//int j = new int();

//if (SolderesOnTable[ik].SoldierThinking[0].TableListSolder.Count == 0)

// return Do;

Parallel.For(0, SolderesOnTable[ik].SoldierThinking[0].TableListSolder.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() != 0 //&& SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsRewardAction() != 1

)

{

//if (JI == 0)

// if (Index[0] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 1)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[0] != -1)

{

if (ik != Index[0])

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[0])

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Clear();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(SolderesOnTable[ik].SoldierThinking[0].Row) + Number(SolderesOnTable[ik].SoldierThinking[0].Column) + Alphabet(SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][0]) + Number(SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Soldier AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Soldier AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

Do = true;

int iii = SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][0];

int jjj = SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][1];

Color aa = a;

int[,] Tab = CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]);

int Ord = Order;

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]), Order, false, FOUND, LeafAStarGreedy));

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "S" + i.ToString();

Do = true;

}

}

}

}

}

else

{

if (//SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsPenaltyAction() != 0 &&

SolderesOnTable[ik].SoldierThinking[0].PenaltyRegardListSolder[j].IsRewardAction() != 1

)

{

//if (JI == 0)

// if (Index[0] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 1)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[0] != -1)

{

if (ik != Index[0])

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[0])

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy == null)

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy = new List<AllDraw>();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Clear();

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]));

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

}

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(SolderesOnTable[ik].SoldierThinking[0].Row) + Number(SolderesOnTable[ik].SoldierThinking[0].Column) + Alphabet(SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][0]) + Number(SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Soldier AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Soldier AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

Do = true;

int iii = SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][0];

int jjj = SolderesOnTable[ik].SoldierThinking[0].RowColumnSoldier[j][1];

Color aa = a;

int[,] Tab = CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]);

int Ord = Order;

SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(SolderesOnTable[ik].SoldierThinking[0].TableListSolder[j]), Order, false, FOUND, LeafAStarGreedy));

//SolderesOnTable[ik].SoldierThinking[0].AStarGreedy[SolderesOnTable[ik].SoldierThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "S" + i.ToString();

Do = true;

}

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

//array.Start();

Task.WaitAll(array);

}

\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

//Elephant

}

bool FullGameThinkingTreeSoldierGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

try

{

Parallel.For(0, SodierMidle, ik =>

{

if (SolderesOnTable != null && SolderesOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeSoldier(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeElephant(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (ElephantOnTable[ik].ElefantThinking[0].ThinkingBegin && (!ElephantOnTable[ik].ElefantThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

//if (ElephantOnTable[ik].ElefantThinking[0].TableListElefant.Count == 0)

// return Do;

Parallel.For(0, ElephantOnTable[ik].ElefantThinking[0].TableListElefant.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() != 0 //&& ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsRewardAction() != 1

)

{

//if (Index[1] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 2)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[1] != -1)

{

if (ik != Index[1])

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[1])

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Clear();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(ElephantOnTable[ik].ElefantThinking[0].Row) + Number(ElephantOnTable[ik].ElefantThinking[0].Column) + Alphabet(ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][0]) + Number(ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Elephant AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Elephant AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][0];

int jjj = ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][1];

Color aa = a;

int[,] Tab = CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]);

int Ord = Order;

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]), Order, false, FOUND, LeafAStarGreedy));

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "E" + i.ToString();

Do = true;

}

}

}

}

}

}

else

{

if (//ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsPenaltyAction() != 0 &&

ElephantOnTable[ik].ElefantThinking[0].PenaltyRegardListElefant[j].IsRewardAction() != 1

)

{

//if (Index[1] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 2)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[1] != -1)

{

if (ik != Index[1])

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[1])

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy == null)

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy = new List<AllDraw>();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Clear();

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]));

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

//ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(ElephantOnTable[ik].ElefantThinking[0].Row) + Number(ElephantOnTable[ik].ElefantThinking[0].Column) + Alphabet(ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][0]) + Number(ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Elephant AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Elephant AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][0];

int jjj = ElephantOnTable[ik].ElefantThinking[0].RowColumnElefant[j][1];

Color aa = a;

int[,] Tab = CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]);

int Ord = Order;

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

Task array = Task.Factory.StartNew(() => ElephantOnTable[ik].ElefantThinking[0].AStarGreedy[ElephantOnTable[ik].ElefantThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(ElephantOnTable[ik].ElefantThinking[0].TableListElefant[j]), Order, false, FOUND, LeafAStarGreedy));

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "E" + i.ToString();

Do = true;

}

}

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

//array.Start();

Task.WaitAll(array);

}

\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

}

bool FullGameThinkingTreeElephantGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Elephant

try

{

Parallel.For(0, ElefantMidle, ik =>

{

if (ElephantOnTable != null && ElephantOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeElephant(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeHourse(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (HoursesOnTable[ik].HourseThinking[0].ThinkingBegin && (!HoursesOnTable[ik].HourseThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

//if (HoursesOnTable[ik].HourseThinking[0].TableListHourse.Count == 0)

// return Do;

Parallel.For(0, HoursesOnTable[ik].HourseThinking[0].TableListHourse.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() != 0 //&& HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsRewardAction() != 1

)

{

//if (Index[2] != -1)

{

if (AllDraw.Blitz)

{

/\* if (Kind != -1)

{

if (Kind != 3)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[2] != -1)

{

if (ik != Index[2])

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[2])

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

Object O3 = new Object();

lock (O3)

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].TableList.Clear(); ;

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, HoursesOnTable[ik].HourseThinking[0].TableListHourse[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(HoursesOnTable[ik].HourseThinking[0].Row) + Number(HoursesOnTable[ik].HourseThinking[0].Column) + Alphabet(HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][0]) + Number(HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Hourse AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Hourse AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][0];

int jjj = HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][1];

Color aa = a;

int[,] Tab = CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]);

int Ord = Order;

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

Task array = Task.Factory.StartNew(() => HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]), Order, false, FOUND, LeafAStarGreedy));

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "H" + i.ToString();

Do = true;

}

}

}

}

}

else

{

if (//HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsPenaltyAction() != 0 //&&

HoursesOnTable[ik].HourseThinking[0].PenaltyRegardListHourse[j].IsRewardAction() != 1

)

{

//if (Index[2] != -1)

{

if (AllDraw.Blitz)

{

/\* if (Kind != -1)

{

if (Kind != 3)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[2] != -1)

{

if (ik != Index[2])

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[2])

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

}

Object O3 = new Object();

lock (O3)

{

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy == null)

HoursesOnTable[ik].HourseThinking[0].AStarGreedy = new List<AllDraw>();

HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].TableList.Clear(); ;

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]));

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, HoursesOnTable[ik].HourseThinking[0].TableListHourse[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(HoursesOnTable[ik].HourseThinking[0].Row) + Number(HoursesOnTable[ik].HourseThinking[0].Column) + Alphabet(HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][0]) + Number(HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Hourse AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Hourse AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][0];

int jjj = HoursesOnTable[ik].HourseThinking[0].RowColumnHourse[j][1];

Color aa = a;

int[,] Tab = CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]);

int Ord = Order;

HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(HoursesOnTable[ik].HourseThinking[0].TableListHourse[j]), Order, false, FOUND, LeafAStarGreedy));

//HoursesOnTable[ik].HourseThinking[0].AStarGreedy[HoursesOnTable[ik].HourseThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "H" + i.ToString();

Do = true;

}

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

//array.Start();

Task.WaitAll(array);

}

\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

}

bool FullGameThinkingTreeHourseGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Hourse.

try

{

Parallel.For(0, HourseMidle, ik =>

{

if (HoursesOnTable != null && HoursesOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeHourse(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeCastle(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (CastlesOnTable[ik].CastleThinking[0].ThinkingBegin && (!CastlesOnTable[ik].CastleThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

//if (CastlesOnTable[ik].CastleThinking[0].TableListCastle.Count == 0)

// return Do;

Parallel.For(0, CastlesOnTable[ik].CastleThinking[0].TableListCastle.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() != 0 //&& CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsRewardAction() != 1

)

{

//if (Index[3] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 4)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[3] != -1)

{

if (ik != Index[3])

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[0])

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].TableList.Clear();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CastlesOnTable[ik].CastleThinking[0].TableListCastle[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(CastlesOnTable[ik].CastleThinking[0].Row) + Number(CastlesOnTable[ik].CastleThinking[0].Column) + Alphabet(CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][0]) + Number(CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Castle AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Castle AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][0];

int jjj = CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][1];

Color aa = a;

int[,] Tab = CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]);

int Ord = Order;

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]), Order, false, FOUND, LeafAStarGreedy));

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "B" + i.ToString();

Do = true;

}

}

}

}

}

else

{

if (//CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsPenaltyAction() != 0 &&

CastlesOnTable[ik].CastleThinking[0].PenaltyRegardListCastle[j].IsRewardAction() != 1

)

{

//if (Index[3] != -1)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 4)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[3] != -1)

{

if (ik != Index[3])

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[0])

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy == null)

CastlesOnTable[ik].CastleThinking[0].AStarGreedy = new List<AllDraw>();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].TableList.Clear();

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]));

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CastlesOnTable[ik].CastleThinking[0].TableListCastle[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(CastlesOnTable[ik].CastleThinking[0].Row) + Number(CastlesOnTable[ik].CastleThinking[0].Column) + Alphabet(CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][0]) + Number(CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Castle AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Castle AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][0];

int jjj = CastlesOnTable[ik].CastleThinking[0].RowColumnCastle[j][1];

Color aa = a;

int[,] Tab = CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]);

int Ord = Order;

CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(CastlesOnTable[ik].CastleThinking[0].TableListCastle[j]), Order, false, FOUND, LeafAStarGreedy));

//CastlesOnTable[ik].CastleThinking[0].AStarGreedy[CastlesOnTable[ik].CastleThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}

\*/

//array.Name = "B" + i.ToString();

Do = true;

}

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

//array.Start();

Task.WaitAll(array);

}\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

}

bool FullGameThinkingTreeCastleGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Castle.

try

{

Parallel.For(0, CastleMidle, ik =>

{

if (CastlesOnTable != null && CastlesOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeCastle(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeMinister(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (MinisterOnTable[ik].MinisterThinking[0].ThinkingBegin && (!MinisterOnTable[ik].MinisterThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

// if (MinisterOnTable[ik].MinisterThinking[0].TableListMinister.Count == 0)

// return Do;

Parallel.For(0, MinisterOnTable[ik].MinisterThinking[0].TableListMinister.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() != 0 //&& MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsRewardAction() != 1

)

{

//if (Index[4] != -1)

{

if (AllDraw.Blitz)

{

/\* if (Kind != -1)

{

if (Kind != 5)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[4] != -1)

{

if (ik != Index[4])

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[4])

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Clear();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(MinisterOnTable[ik].MinisterThinking[0].Row) + Number(MinisterOnTable[ik].MinisterThinking[0].Column) + Alphabet(MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][0]) + Number(MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Minister AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Minister AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

Do = true;

int iii = MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][0];

int jjj = MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][1];

Color aa = a;

int[,] Tab = CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]);

int Ord = Order;

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]), Order, false, FOUND, LeafAStarGreedy));

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "M" + i.ToString();

Do = true;

}

}

}

}

}

else

{

if (//MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsPenaltyAction() != 0 &&

MinisterOnTable[ik].MinisterThinking[0].PenaltyRegardListMinister[j].IsRewardAction() != 1

)

{

//if (Index[4] != -1)

{

if (AllDraw.Blitz)

{

/\* if (Kind != -1)

{

if (Kind != 5)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[4] != -1)

{

if (ik != Index[4])

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[4])

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy == null)

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy = new List<AllDraw>();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Clear();

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]));

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(MinisterOnTable[ik].MinisterThinking[0].Row) + Number(MinisterOnTable[ik].MinisterThinking[0].Column) + Alphabet(MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][0]) + Number(MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][1]);

if (Order == 1)

OutPut = "\r\nPerception Minister AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception Minister AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][0];

int jjj = MinisterOnTable[ik].MinisterThinking[0].RowColumnMinister[j][1];

Color aa = a;

int[,] Tab = CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]);

int Ord = Order;

MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(MinisterOnTable[ik].MinisterThinking[0].TableListMinister[j]), Order, false, FOUND, LeafAStarGreedy));

///MinisterOnTable[ik].MinisterThinking[0].AStarGreedy[MinisterOnTable[ik].MinisterThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "M" + i.ToString();

Do = true;

}

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

Task.WaitAll(array);

//array.Start();

}

\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

}

bool FullGameThinkingTreeMinisterGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Minister.

try

{

Parallel.For(0, MinisterMidle, ik =>

{

if (MinisterOnTable != null && MinisterOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeMinister(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeKing(int ik, Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

TaskBegin++;

while (KingOnTable[ik].KingThinking[0].ThinkingBegin && (!KingOnTable[ik].KingThinking[0].ThinkingFinished))

{

System.Threading.Thread.Sleep(1);

}// S += 100; if (AllDraw.Blitz) { if (S > ThresholdBlitz)break; } else { if (S > ThresholdFullGame)break; } }

//List<Task> tHA = new List<Task>();

if (KingOnTable[ik].KingThinking[0].TableListKing.Count == 0)

return Do;

Parallel.For(0, KingOnTable[ik].KingThinking[0].TableListKing.Count, j =>

{

Object ooo = new Object();

lock (ooo)

{

try

{

if (AllDraw.OrderPlate == Order)

{

if (KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() != 0 //&& KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsRewardAction() != 1

)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 6)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (Index[5] != -1)

{

if (ik != Index[5])

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[5])

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

Object O3 = new Object();

lock (O3)

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].TableList.Clear();

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, KingOnTable[ik].KingThinking[0].TableListKing[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (KingOnTable[ik].KingThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(KingOnTable[ik].KingThinking[0].Row) + Number(KingOnTable[ik].KingThinking[0].Column) + Alphabet(KingOnTable[ik].KingThinking[0].RowColumnKing[j][0]) + Number(KingOnTable[ik].KingThinking[0].RowColumnKing[j][1]);

if (Order == 1)

OutPut = "\r\nPerception King AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception King AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = KingOnTable[ik].KingThinking[0].RowColumnKing[j][0];

int jjj = KingOnTable[ik].KingThinking[0].RowColumnKing[j][1];

Color aa = a;

int[,] Tab = CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]);

int Ord = Order;

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]), Order, false, FOUND, LeafAStarGreedy));

//KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "K" + i.ToString();

Do = true;

}

}

}

}

else

{

if (//KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsPenaltyAction() != 0 &&

KingOnTable[ik].KingThinking[0].PenaltyRegardListKing[j].IsRewardAction() != 1

)

{

if (AllDraw.Blitz)

{

/\*if (Kind != -1)

{

if (Kind != 6)

continue;

else

if (ik != Index)

continue;

else

if (j != jindex)

continue;

}

else

continue;

\*/

if (ik != Index[5])

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

else

if (j != jindex[5])

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

}

else

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

return;

}

Object O3 = new Object();

lock (O3)

{

if (KingOnTable[ik].KingThinking[0].AStarGreedy == null)

KingOnTable[ik].KingThinking[0].AStarGreedy = new List<AllDraw>();

KingOnTable[ik].KingThinking[0].AStarGreedy.Add(new AllDraw(OrderPlate, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].TableList.Clear();

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].TableList.Add(CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]));

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].SetRowColumn(0);

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].SetRowColumnFinishedWait();

}

//KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, KingOnTable[ik].KingThinking[0].TableListKing[j], Order, false);

//ParameterizedThreadStart start = new ParameterizedThreadStart(KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt);

if (KingOnTable[ik].KingThinking[0].AStarGreedy.Count > 0)

{

Object O = new Object();

lock (O)

{

OutPutAction = " " + Alphabet(KingOnTable[ik].KingThinking[0].Row) + Number(KingOnTable[ik].KingThinking[0].Column) + Alphabet(KingOnTable[ik].KingThinking[0].RowColumnKing[j][0]) + Number(KingOnTable[ik].KingThinking[0].RowColumnKing[j][1]);

if (Order == 1)

OutPut = "\r\nPerception King AstarGreedy By Bob at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

else

OutPut = "\r\nPerception King AstarGreedy By Alice at Level " + iAStarGreedy.ToString() + " By " + PerceptionCount.ToString() + "th Perception String " + OutPutAction;

PerceptionCount++;

int iii = KingOnTable[ik].KingThinking[0].RowColumnKing[j][0];

int jjj = KingOnTable[ik].KingThinking[0].RowColumnKing[j][1];

Color aa = a;

int[,] Tab = CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]);

int Ord = Order;

KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].AStarGreedyString = this;

Task array = Task.Factory.StartNew(() => KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, ii, jj, a, CloneATable(KingOnTable[ik].KingThinking[0].TableListKing[j]), Order, false, FOUND, LeafAStarGreedy));

//KingOnTable[ik].KingThinking[0].AStarGreedy[KingOnTable[ik].KingThinking[0].AStarGreedy.Count - 1].InitiateAStarGreedyt(iAStarGreedy, iii, jjj, aa, Tab, Ord, false, FOUND, LeafAStarGreedy);

//array.Start();

/\*if (!AllDraw.Blitz)

{

Object ttttt = new Object(); lock (ttttt) { tHA.Add(array); }

}

else

{

Object ttttt = new Object(); lock (ttttt) { array.Wait(); }

}\*/

//array.Name = "K" + i.ToString();

Do = true;

}

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

});

/\*if (tHA.Count > 1)

{

Task array = Task.Factory.StartNew(() => Parallel.ForEach(tHA, items => Task.WaitAny(items)));

Task.WaitAll(array);

//array.Start();

}\*/

Object O2 = new Object();

lock (O2)

{

TaskEnd++;

}

}

return Do;

}

bool FullGameThinkingTreeKingGray(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//int ik;

//King.

try

{

Parallel.For(0, KingMidle, ik =>

{

if (KingOnTable != null && KingOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeKing(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeSoldierBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

try

{

Parallel.For(SodierMidle, SodierHigh, ik =>

{

if (SolderesOnTable != null && SolderesOnTable[ik] != null)

{

//Soldier.

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeSoldier(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeElephantBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Elephant

try

{

Parallel.For(ElefantMidle, ElefantHigh, ik =>

{

if (ElephantOnTable != null && ElephantOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeElephant(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeHourseBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Hourse.

try

{

Parallel.For(HourseMidle, HourseHight, ik =>

{

if (HoursesOnTable != null && HoursesOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeHourse(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeCastleBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Castles.

try

{

Parallel.For(CastleMidle, CastleHigh, ik =>

{

if (CastlesOnTable != null && CastlesOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeCastle(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeMinisterBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//Minister.

try

{

Parallel.For(MinisterMidle, MinisterHigh, ik =>

{

if (MinisterOnTable != null && MinisterOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeMinister(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

}

return Do;

}

bool FullGameThinkingTreeKingBrown(Color a, int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

bool Do = false;

Object O1 = new Object();

lock (O1)

{

//King.

try

{

Parallel.For(KingMidle, KingHigh, ik =>

{

if (KingOnTable != null && KingOnTable[ik] != null)

{

Object O = new Object();

lock (O)

{

Do = FullGameThinkingTreeKing(ik, a, Order, iAStarGreedy, ii, jj, ik1, j1, FOUND, LeafAStarGreedy);

}

}

});

}

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

{

//if (JI == 0)

//if (JI == 1)

//if (JI == 2)

//if (JI == 3)

//if (JI == 4)

//if (JI == 5)

}

}

return Do;

}

bool FullGameThinkingTree(int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

//List<Task> tH = new List<Task>();

bool Do = false;

//Initiatye Variables.

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

Color a;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

////Order \*= -1;

//Index = -1;

//jindex = -1;

//Kind =

Object O = new Object();

lock (O)

{

if (AllDraw.Blitz)

FullGameMakimgBlitz(ref Index, ref jindex, Order, LeafAStarGreedy);

}

/\*Object OO = new Object();

lock (OO)

{

if (Order == -1)

{

//Index[0] = -1;

//Soldeir

//Initiatye Variables.

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

Task array1 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeSoldierGray(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND));

//array1.Start();

Object tttt1 = new Object(); lock (tttt1) { TH.Add(array1); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

Task array2 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeElephantGray(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND));

//array2.Start();

Object tttt2 = new Object(); lock (tttt2) { TH.Add(array2); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

Task array3 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeHourseGray(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND));

///array3.Start();

Object tttt3 = new Object(); lock (tttt3) { TH.Add(array3); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

Task array4 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeCastleGray(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND));

//array4.Start();

Object tttt4 = new Object(); lock (tttt4) { TH.Add(array4); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

Task array5 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeMinisterGray(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND));

//array5.Start();

Object tttt5 = new Object(); lock (tttt5) { TH.Add(array5); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

Task array6 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeKingGray(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND));

//array6.Start();

Object tttt6 = new Object(); lock (tttt6) { TH.Add(array6); }

}

//For Brown Order Blitz Game Calculate Maximum Table Inclusive AStarGreedy First Game Search.

else

{

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

Task array1 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeSoldierBrown(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND));

//array1.Start();

Object tttt1 = new Object(); lock (tttt1) { TH.Add(array1); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

Task array2 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeElephantBrown(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND));

//array2.Start();

Object tttt2 = new Object(); lock (tttt2) { TH.Add(array2); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

Task array3 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeHourseBrown(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND));

///array3.Start();

Object tttt3 = new Object(); lock (tttt3) { TH.Add(array3); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

Task array4 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeCastleBrown(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND));

//array4.Start();

Object tttt4 = new Object(); lock (tttt4) { TH.Add(array4); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

Task array5 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeMinisterBrown(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND));

//array5.Start();

Object tttt5 = new Object(); lock (tttt5) { TH.Add(array5); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

Task array6 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeKingBrown(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND));

//array6.Start();

Object tttt6 = new Object(); lock (tttt6) { TH.Add(array6); }

}

Parallel.ForEach(TH, items => Task.WaitAny(items));

}

\*/

if (Order == -1)

{

//Index[0] = -1;

//Soldeir

//Initiatye Variables.

Parallel.Invoke(() =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

Do |= this.FullGameThinkingTreeSoldierGray(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND, LeafAStarGreedy);

//Task array1 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeSoldierGray(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND));

//array1.Start();

//Object tttt1 = new Object(); lock (tttt1) { TH.Add(array1); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

Do |= this.FullGameThinkingTreeElephantGray(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND, LeafAStarGreedy);

//Task array2 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeElephantGray(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND));

//array2.Start();

//Object tttt2 = new Object(); lock (tttt2) { TH.Add(array2); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

Do |= this.FullGameThinkingTreeHourseGray(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND, LeafAStarGreedy);

//Task array3 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeHourseGray(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND));

///array3.Start();

//Object tttt3 = new Object(); lock (tttt3) { TH.Add(array3); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

Do |= this.FullGameThinkingTreeCastleGray(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND, LeafAStarGreedy);

//Task array4 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeCastleGray(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND));

//array4.Start();

//Object tttt4 = new Object(); lock (tttt4) { TH.Add(array4); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

Do |= this.FullGameThinkingTreeMinisterGray(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND, LeafAStarGreedy);

//Task array5 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeMinisterGray(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND));

//array5.Start();

//Object tttt5 = new Object(); lock (tttt5) { TH.Add(array5); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

Do |= this.FullGameThinkingTreeKingGray(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND, LeafAStarGreedy);

//Task array6 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeKingGray(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND));

//array6.Start();

//Object tttt6 = new Object(); lock (tttt6) { TH.Add(array6); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

});

}

//For Brown Order Blitz Game Calculate Maximum Table Inclusive AStarGreedy First Game Search.

else

{

Parallel.Invoke(() =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

Do |= this.FullGameThinkingTreeSoldierBrown(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND, LeafAStarGreedy);

//Task array1 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeSoldierBrown(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND));

//array1.Start();

//Object tttt1 = new Object(); lock (tttt1) { TH.Add(array1); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

Do |= this.FullGameThinkingTreeElephantBrown(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND, LeafAStarGreedy);

//Task array2 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeElephantBrown(a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND));

//array2.Start();

//Object tttt2 = new Object(); lock (tttt2) { TH.Add(array2); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

Do |= this.FullGameThinkingTreeHourseBrown(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND, LeafAStarGreedy);

//Task array3 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeHourseBrown(a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND));

///array3.Start();

//Object tttt3 = new Object(); lock (tttt3) { TH.Add(array3); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

Do |= this.FullGameThinkingTreeCastleBrown(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND, LeafAStarGreedy);

//Task array4 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeCastleBrown(a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND));

//array4.Start();

//Object tttt4 = new Object(); lock (tttt4) { TH.Add(array4); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

Do |= this.FullGameThinkingTreeMinisterBrown(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND, LeafAStarGreedy);

//Task array5 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeMinisterBrown(a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND));

//array5.Start();

//Object tttt5 = new Object(); lock (tttt5) { TH.Add(array5); }

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}, () =>

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

Do |= this.FullGameThinkingTreeKingBrown(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND, LeafAStarGreedy);

//Task array6 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeKingBrown(a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND));

//array6.Start();

//Object tttt6 = new Object(); lock (tttt6) { TH.Add(array6); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

});

}

return Do;

}

bool FullGameThinkingTreeObject(int Order, int iAStarGreedy, int ii, int jj, int ik1, int j1, bool FOUND, int LeafAStarGreedy)

{

//List<Task> tH = new List<Task>();

bool Do = false;

//Initiatye Variables.

int DummyOrder = Order;

int DummyCurrentOrder = ChessRules.CurrentOrder;

Color a;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

////Order \*= -1;

//Index = -1;

//jindex = -1;

//Kind =

Object O = new Object();

lock (O)

{

if (AllDraw.Blitz)

FullGameMakimgBlitz(ref Index, ref jindex, Order, LeafAStarGreedy);

}

if (Order == -1)

{

Parallel.For(0, MaxGrayMidle(), i =>

{

Parallel.Invoke(() =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < SodierMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

int i1 = i;

Do |= FullGameThinkingTreeSoldier(i1, a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND, LeafAStarGreedy);

//Task array1 = Task.Factory.StartNew(() => Do |= this.FullGameThinkingTreeSoldierGray(a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND));

//array1.Start();

//Object tttt1 = new Object(); lock (tttt1) { TH.Add(array1); }

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < ElefantMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

int i2 = i;

Do |= this.FullGameThinkingTreeElephant(i2, a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < HourseMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

int i3 = i;

Do |= this.FullGameThinkingTreeHourse(i3, a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < CastleMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

int i4 = i;

Do |= this.FullGameThinkingTreeCastle(i4, a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < MinisterMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

int i5 = i;

Do |= this.FullGameThinkingTreeMinister(i5, a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

Object ooo = new Object();

lock (ooo)

{

if (i < KingMidle)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

//Order \*= -1;

//ChessRules.CurrentOrder \*= -1;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

int i6 = i;

Do |= this.FullGameThinkingTreeKing(i6, a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND, LeafAStarGreedy);

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

});

});

}

//For Brown Order Blitz Game Calculate Maximum Table Inclusive AStarGreedy First Game Search.

else

{

Parallel.For(MinBrownMidle(), MaxGrayMidle(), i =>

{

Parallel.Invoke(() =>

{

Object ooo = new Object();

lock (ooo)

{

if (i >= SodierMidle && i < SodierHigh)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii1 = ii, jj1 = jj, ik11 = ik1, j11 = j1;

int Ord1 = Order;

Color a1 = a;

int iAStarGreedy1 = iAStarGreedy;

int i1 = i;

Do |= this.FullGameThinkingTreeSoldier(i1, a1, Ord1, iAStarGreedy1, ii1, jj1, ik11, j11, FOUND, LeafAStarGreedy);

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}

}, () =>

{

if (i >= ElefantMidle && i < ElefantHigh)

{

Object ooo = new Object();

lock (ooo)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii2 = ii, jj2 = jj, ik12 = ik1, j12 = j1;

int Ord2 = Order;

Color a2 = a;

int iAStarGreedy2 = iAStarGreedy;

int i2 = i;

Do |= this.FullGameThinkingTreeElephant(i2, a2, Ord2, iAStarGreedy2, ii2, jj2, ik12, j12, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}, () =>

{

if (i >= HourseMidle && i < HourseHight)

{

Object O1 = new Object();

lock (O1)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii3 = ii, jj3 = jj, ik13 = ik1, j13 = j1;

int Ord3 = Order;

Color a3 = a;

int iAStarGreedy3 = iAStarGreedy;

int i3 = i;

Do |= this.FullGameThinkingTreeHourse(i3, a3, Ord3, iAStarGreedy3, ii3, jj3, ik13, j13, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}, () =>

{

if (i >= CastleMidle && i < CastleHigh)

{

Object ooo = new Object();

lock (ooo)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii4 = ii, jj4 = jj, ik14 = ik1, j14 = j1;

int Ord4 = Order;

Color a4 = a;

int iAStarGreedy4 = iAStarGreedy;

int i4 = i;

Do |= this.FullGameThinkingTreeCastle(i4, a4, Ord4, iAStarGreedy4, ii4, jj4, ik14, j14, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}, () =>

{

if (i >= MinisterMidle && i < MinisterHigh)

{

Object ooo = new Object();

lock (ooo)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii5 = ii, jj5 = jj, ik15 = ik1, j15 = j1;

int Ord5 = Order;

Color a5 = a;

int iAStarGreedy5 = iAStarGreedy;

int i5 = i;

Do |= this.FullGameThinkingTreeMinister(i5, a5, Ord5, iAStarGreedy5, ii5, jj5, ik15, j15, FOUND, LeafAStarGreedy);

//Initiatye Variables.

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

}, () =>

{

if (i >= KingMidle && i < KingHigh)

{

Object ooo = new Object();

lock (ooo)

{

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int ii6 = ii, jj6 = jj, ik16 = ik1, j16 = j1;

int Ord6 = Order;

Color a6 = a;

int iAStarGreedy6 = iAStarGreedy;

int i6 = i;

Do |= this.FullGameThinkingTreeKing(i6, a6, Ord6, iAStarGreedy6, ii6, jj6, ik16, j16, FOUND, LeafAStarGreedy);

Order = DummyOrder;

ChessRules.CurrentOrder = DummyCurrentOrder;

}

}

});

});

}

return Do;

}

int[,] CloneATable(int[,] Tab)

{

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

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

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

Table[i, j] = Tab[i, j];

return Table;

}

int MaxOfThreeHuristic(double \_1, double \_2, double \_3, double \_4, double \_5, double \_6)

{

double[] Less = new double[6];

Less[0] = \_1;

Less[1] = \_2;

Less[2] = \_3;

Less[3] = \_4;

Less[4] = \_5;

Less[5] = \_6;

int Value = -1;

double Les = Double.MinValue;

for (int i = 0; i < 6; i++)

{

if (Less[i] > Les)

{

Les = Less[i];

Value = i;

}

}

return Value;

}

//best movement indexes founder method.

List<List<double>> FoundOfBestMovments(int AStarGreedy, ref List<double> i, ref List<double> j, ref List<double> k, AllDraw Dummy, Color a, int Order)

{

//initiate local variables.

List<List<double>> p = new List<List<double>>();

for (int ii = 0; ii < 6; ii++)

{

List<double> pl = new List<double>();

p.Add(pl);

}

Less = Double.MinValue; ;

List<AllDraw> DummyList = new List<AllDraw>();

DummyList.Add(Dummy);

MaxHuristicAStarGreedytBackWard.Clear();

//found best movment depend of max huristic.

Dummy.HuristicAStarGreedySearch(0, a, Order, false);

//proccess from a stored global variable decicion making.

if (MaxHuristicAStarGreedytBackWard[0][1] != -1)//soldier.

{

i.Add(MaxHuristicAStarGreedytBackWard[0][2]);

j.Add(MaxHuristicAStarGreedytBackWard[0][3]);

k.Add(MaxHuristicAStarGreedytBackWard[0][4]);

p[0].Add(MaxHuristicAStarGreedytBackWard[0][2]);

}

else if (MaxHuristicAStarGreedytBackWard[0][5] != -1)//Elephant

{

i.Add(MaxHuristicAStarGreedytBackWard[0][6]);

j.Add(MaxHuristicAStarGreedytBackWard[0][7]);

k.Add(MaxHuristicAStarGreedytBackWard[0][8]);

p[1].Add(MaxHuristicAStarGreedytBackWard[0][6]);

}

else if (MaxHuristicAStarGreedytBackWard[0][9] != -1)//Hourse

{

i.Add(MaxHuristicAStarGreedytBackWard[0][10]);

j.Add(MaxHuristicAStarGreedytBackWard[0][11]);

k.Add(MaxHuristicAStarGreedytBackWard[0][12]);

p[2].Add(MaxHuristicAStarGreedytBackWard[0][10]);

}

else if (MaxHuristicAStarGreedytBackWard[0][13] != -1)//Castles.

{

i.Add(MaxHuristicAStarGreedytBackWard[0][14]);

j.Add(MaxHuristicAStarGreedytBackWard[0][15]);

k.Add(MaxHuristicAStarGreedytBackWard[0][16]);

p[3].Add(MaxHuristicAStarGreedytBackWard[0][14]);

}

else if (MaxHuristicAStarGreedytBackWard[0][17] != -1)//Minister

{

i.Add(MaxHuristicAStarGreedytBackWard[0][18]);

j.Add(MaxHuristicAStarGreedytBackWard[0][19]);

k.Add(MaxHuristicAStarGreedytBackWard[0][20]);

p[4].Add(MaxHuristicAStarGreedytBackWard[0][18]);

}

else if (MaxHuristicAStarGreedytBackWard[0][21] != -1)//King.

{

i.Add(MaxHuristicAStarGreedytBackWard[0][22]);

j.Add(MaxHuristicAStarGreedytBackWard[0][23]);

k.Add(MaxHuristicAStarGreedytBackWard[0][24]);

p[5].Add(MaxHuristicAStarGreedytBackWard[0][22]);

}

//not found

return p;

}

//Copying of Items of Enemy Non Move and Current Moved.

public AllDraw CopyRemeiningItems(AllDraw ADummy, int Order)

{

//Initiate Local Variables.

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

Dummy.TableList.Add(TableList[0]);

Dummy.SolderesOnTable = new DrawSoldier[SodierHigh];

Dummy.ElephantOnTable = new DrawElefant[ElefantHigh];

Dummy.HoursesOnTable = new DrawHourse[HourseHight];

Dummy.CastlesOnTable = new DrawCastle[CastleHigh];

Dummy.MinisterOnTable = new DrawMinister[MinisterHigh];

Dummy.KingOnTable = new DrawKing[KingHigh];

//For All Sodiers Movments.

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

{

try

{

//Construction of Current Solders.

Dummy.SolderesOnTable[i] = new DrawSoldier(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, SolderesOnTable[i].Row, SolderesOnTable[i].Column, SolderesOnTable[i].color, SolderesOnTable[i].Table, SolderesOnTable[i].Order, false, SolderesOnTable[i].Current);

}

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

}

//For All Elephant Objects.

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

{

try

{

//Construction of Curren Elephant.

Dummy.ElephantOnTable[i] = new DrawElefant(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, ElephantOnTable[i].Row, ElephantOnTable[i].Column, ElephantOnTable[i].color, ElephantOnTable[i].Table, ElephantOnTable[i].Order, false, ElephantOnTable[i].Current);

}

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

}

//for All Hourse Objects.

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

{

try

{

//Construction of Hourse Objects.

Dummy.HoursesOnTable[i] = new DrawHourse(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, HoursesOnTable[i].Row, HoursesOnTable[i].Column, HoursesOnTable[i].color, HoursesOnTable[i].Table, HoursesOnTable[i].Order, false, HoursesOnTable[i].Current);

}

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

}

//For All Castles Objects.

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

{

try

{

//Construction of Castles Objects.

Dummy.CastlesOnTable[i] = new DrawCastle(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, CastlesOnTable[i].Row, CastlesOnTable[i].Column, CastlesOnTable[i].color, CastlesOnTable[i].Table, CastlesOnTable[i].Order, false, CastlesOnTable[i].Current);

}

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

}

//For All Minister Objects.

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

{

try

{

//Construction of Current Minister.

Dummy.MinisterOnTable[i] = new DrawMinister(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, MinisterOnTable[i].Row, MinisterOnTable[i].Column, MinisterOnTable[i].color, MinisterOnTable[i].Table, MinisterOnTable[i].Order, false, MinisterOnTable[i].Current);

}

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

}

//For All King Objects.

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

{

try

{

//Construction of Kings Objects.

Dummy.KingOnTable[i] = new DrawKing(CurrentAStarGredyMax, MovementsAStarGreedyHuristicFoundT, IgnoreSelfObjectsT, UsePenaltyRegardMechnisamT, BestMovmentsT, PredictHuristicT, OnlySelfT, AStarGreedyHuristicT, ArrangmentsChanged, KingOnTable[i].Row, KingOnTable[i].Column, KingOnTable[i].color, KingOnTable[i].Table, KingOnTable[i].Order, false, KingOnTable[i].Current);

}

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

}

//Gray Order.

if (Order == 1)

{

//For Gray Soders Objects.

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

{

try

{

//Clone a Movments.

ADummy.SolderesOnTable[i].Clone(ref Dummy.SolderesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.SolderesOnTable[i] = null; }

}

//For Gray Elephant.

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

{

try

{

//Clone a Movments.

ADummy.ElephantOnTable[i].Clone(ref Dummy.ElephantOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.ElephantOnTable[i] = null; }

}

//For Gray Hourses.

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

{

try

{

//Clone a Movments.

ADummy.HoursesOnTable[i].Clone(ref Dummy.HoursesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.HoursesOnTable[i] = null; }

}

//For Gray Castles.

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

{

try

{

//Clone a Movments.

ADummy.CastlesOnTable[i].Clone(ref Dummy.CastlesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.CastlesOnTable[i] = null; }

}

//For Gray Ministers.

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

{

try

{

//Clone a Movments.

ADummy.MinisterOnTable[i].Clone(ref Dummy.MinisterOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.MinisterOnTable[i] = null; }

}

//For Gray King.

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

{

try

{

//Clone a Movments.

ADummy.KingOnTable[i].Clone(ref Dummy.KingOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.KingOnTable[i] = null; }

}

//For All Solders.

}

else//For Order Brown.

{

{

//For Brown Solders.

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

{

try

{

//Clone a Movments.

ADummy.SolderesOnTable[i].Clone(ref Dummy.SolderesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.SolderesOnTable[i] = null; }

}

//For All Brown Elephants.

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

{

try

{

//Clone a Enemy.

ADummy.ElephantOnTable[i].Clone(ref Dummy.ElephantOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.ElephantOnTable[i] = null; }

}

//For All Brown Hourses.

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

{

try

{

//Clone a Enemy.

ADummy.HoursesOnTable[i].Clone(ref Dummy.HoursesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.HoursesOnTable[i] = null; }

}

//For Brown Castles.

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

{

try

{

//Clone a Movments.

ADummy.CastlesOnTable[i].Clone(ref Dummy.CastlesOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.CastlesOnTable[i] = null; }

}

//For Gray Minsters.

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

{

try

{

//Clone a Enemy.

ADummy.MinisterOnTable[i].Clone(ref Dummy.MinisterOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.MinisterOnTable[i] = null; }

}

//For Brown Kings.

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

{

try

{

//Clone a Enemy.

ADummy.KingOnTable[i].Clone(ref Dummy.KingOnTable[i]);

}

catch (Exception t) { Log(t); Dummy.KingOnTable[i] = null; }

}

}

}

//Return Constructed Tables.

return Dummy;

}

public bool TableZero(int[,] Ta)

{

bool Zerro = true;

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

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

if (Ta[i, j] != 0)

Zerro = false;

return Zerro;

}

void CheckedMateConfiguratiionSoldier(int Order, int i, bool Regrad)

{

for (int j = 0; j < SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder.Count; j++)

{

try

{

if (SolderesOnTable[i].SoldierThinking[0].LearningVarsObject.Count == SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder.Count)

if (SolderesOnTable[i].SoldierThinking[0].LearningVarsObject[j][1] && (!SolderesOnTable[i].SoldierThinking[0].LearningVarsObject[j][4]))

{

SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j].Initiate();

//if(Regrad)

//SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j].LearningAlgorithmRegard();

//else

if (!Regrad)

SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j].LearningAlgorithmPenalty();

SolderesOnTable[i].SoldierThinking[0].HuristicPenaltyValuePerform(SolderesOnTable[i].SoldierThinking[0].PenaltyRegardListSolder[j], Order, ref SolderesOnTable[i].SoldierThinking[0].HuristicListSolder[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiionElephant(int Order, int i, bool Regrad)

{

for (int j = 0; j < ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant.Count; j++)

{

try

{

if (ElephantOnTable[i].ElefantThinking[0].LearningVarsObject.Count == ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant.Count)

if (ElephantOnTable[i].ElefantThinking[0].LearningVarsObject[j][1] && (!ElephantOnTable[i].ElefantThinking[0].LearningVarsObject[j][4]))

{

ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].Initiate();

//if(Regrad)

//ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmRegard();

//else

if (!Regrad)

ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j].LearningAlgorithmPenalty();

ElephantOnTable[i].ElefantThinking[0].HuristicPenaltyValuePerform(ElephantOnTable[i].ElefantThinking[0].PenaltyRegardListElefant[j], Order, ref ElephantOnTable[i].ElefantThinking[0].HuristicListElefant[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiionHourse(int Order, int i, bool Regrad)

{

for (int j = 0; j < HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse.Count; j++)

{

try

{

if (HoursesOnTable[i].HourseThinking[0].LearningVarsObject.Count == HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse.Count)

if (HoursesOnTable[i].HourseThinking[0].LearningVarsObject[j][1] && (!HoursesOnTable[i].HourseThinking[0].LearningVarsObject[j][4]))

{

HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].Initiate();

//if(Regrad)

//HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmRegard();

//else

if (!Regrad)

HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j].LearningAlgorithmPenalty();

HoursesOnTable[i].HourseThinking[0].HuristicPenaltyValuePerform(HoursesOnTable[i].HourseThinking[0].PenaltyRegardListHourse[j], Order, ref HoursesOnTable[i].HourseThinking[0].HuristicListHourse[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiionCastle(int Order, int i, bool Regrad)

{

for (int j = 0; j < CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle.Count; j++)

{

try

{

if (CastlesOnTable[i].CastleThinking[0].LearningVarsObject.Count == CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle.Count)

if (CastlesOnTable[i].CastleThinking[0].LearningVarsObject[j][1] && (!CastlesOnTable[i].CastleThinking[0].LearningVarsObject[j][4]))

{

CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].Initiate();

//if(Regrad)

//CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmRegard();

//else

if (!Regrad)

CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j].LearningAlgorithmPenalty();

CastlesOnTable[i].CastleThinking[0].HuristicPenaltyValuePerform(CastlesOnTable[i].CastleThinking[0].PenaltyRegardListCastle[j], Order, ref CastlesOnTable[i].CastleThinking[0].HuristicListCastle[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiionMinister(int Order, int i, bool Regrad)

{

for (int j = 0; j < MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister.Count; j++)

{

try

{

if (MinisterOnTable[i].MinisterThinking[0].LearningVarsObject.Count == MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister.Count)

if (MinisterOnTable[i].MinisterThinking[0].LearningVarsObject[j][1] && (!MinisterOnTable[i].MinisterThinking[0].LearningVarsObject[j][4]))

{

MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].Initiate();

//if(Regrad)

//MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmRegard();

//else

if (!Regrad)

MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j].LearningAlgorithmPenalty();

MinisterOnTable[i].MinisterThinking[0].HuristicPenaltyValuePerform(MinisterOnTable[i].MinisterThinking[0].PenaltyRegardListMinister[j], Order, ref MinisterOnTable[i].MinisterThinking[0].HuristicListMinister[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiionking(int Order, int i, bool Regrad)

{

for (int j = 0; j < KingOnTable[i].KingThinking[0].PenaltyRegardListKing.Count; j++)

{

try

{

if (KingOnTable[i].KingThinking[0].LearningVarsObject.Count == KingOnTable[i].KingThinking[0].PenaltyRegardListKing.Count)

if (KingOnTable[i].KingThinking[0].LearningVarsObject[j][1] && (!KingOnTable[i].KingThinking[0].LearningVarsObject[j][4]))

{

KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].Initiate();

//if(Regrad)

//KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmRegard();

//else

if (!Regrad)

KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j].LearningAlgorithmPenalty();

KingOnTable[i].KingThinking[0].HuristicPenaltyValuePerform(KingOnTable[i].KingThinking[0].PenaltyRegardListKing[j], Order, ref KingOnTable[i].KingThinking[0].HuristicListKing[j][0], true);

}

}

catch (Exception t)

{

Log(t);

}

}

}

void CheckedMateConfiguratiion(int Order)

{

if (ThinkingChess.LearningVarsCheckedMateOccured && ThinkingChess.LearningVarsCheckedMateOccuredOneCheckedMate)

{

if (Order == 1)

{

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

if (SolderesOnTable != null && SolderesOnTable[i] != null)

try

{

CheckedMateConfiguratiionSoldier(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (ElephantOnTable != null && ElephantOnTable[i] != null)

try

{

CheckedMateConfiguratiionElephant(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (HoursesOnTable != null && HoursesOnTable[i] != null)

try

{

CheckedMateConfiguratiionHourse(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (CastlesOnTable != null && CastlesOnTable[i] != null)

try

{

CheckedMateConfiguratiionCastle(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (MinisterOnTable != null && MinisterOnTable[i] != null)

try

{

CheckedMateConfiguratiionMinister(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (KingOnTable != null && KingOnTable[i] != null)

try

{

CheckedMateConfiguratiionking(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

}

else

{

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

if (SolderesOnTable != null && SolderesOnTable[i] != null)

try

{

CheckedMateConfiguratiionSoldier(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (ElephantOnTable != null && ElephantOnTable[i] != null)

try

{

CheckedMateConfiguratiionElephant(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (HoursesOnTable != null && HoursesOnTable[i] != null)

try

{

CheckedMateConfiguratiionHourse(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (CastlesOnTable != null && CastlesOnTable[i] != null)

try

{

CheckedMateConfiguratiionCastle(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (MinisterOnTable != null && MinisterOnTable[i] != null)

try

{

CheckedMateConfiguratiionMinister(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

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

if (KingOnTable != null && KingOnTable[i] != null)

try

{

CheckedMateConfiguratiionking(Order, i, true);

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

if (Order == 1)

{

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

if (SolderesOnTable != null && SolderesOnTable[i] != null)

try

{

CheckedMateConfiguratiionSoldier(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (ElephantOnTable != null && ElephantOnTable[i] != null)

try

{

CheckedMateConfiguratiionElephant(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (HoursesOnTable != null && HoursesOnTable[i] != null)

try

{

CheckedMateConfiguratiionHourse(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (CastlesOnTable != null && CastlesOnTable[i] != null)

try

{

CheckedMateConfiguratiionCastle(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (MinisterOnTable != null && MinisterOnTable[i] != null)

try

{

CheckedMateConfiguratiionMinister(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (KingOnTable != null && KingOnTable[i] != null)

try

{

CheckedMateConfiguratiionking(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

}

else

{

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

if (SolderesOnTable != null && SolderesOnTable[i] != null)

try

{

CheckedMateConfiguratiionSoldier(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (ElephantOnTable != null && ElephantOnTable[i] != null)

try

{

CheckedMateConfiguratiionElephant(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (HoursesOnTable != null && HoursesOnTable[i] != null)

try

{

CheckedMateConfiguratiionHourse(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (CastlesOnTable != null && CastlesOnTable[i] != null)

try

{

CheckedMateConfiguratiionCastle(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (MinisterOnTable != null && MinisterOnTable[i] != null)

try

{

CheckedMateConfiguratiionMinister(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

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

if (KingOnTable != null && KingOnTable[i] != null)

try

{

CheckedMateConfiguratiionking(Order, i, false);

}

catch (Exception t)

{

Log(t);

}

}

}

}

//Main Initiate Thinking Method.

public int[,] Initiate(int ii, int jj, Color a, int[,] Table, int Order, bool TB, bool FOUND, int LeafAStarGreedy)

{

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

int Current = ChessRules.CurrentOrder;

int DummyOrder = Order;

Object O = new Object();

lock (O)

{

AllDraw.ActionStringReady = false;

SignKiller = Double.MaxValue / (System.Math.Pow(6 \* 32, AllDraw.MaxAStarGreedy) \* 64 \* 32);

ThinkingChess.LearningVarsCheckedMateOccured = false;

ThinkingChess.LearningVarsCheckedMateOccuredOneCheckedMate = false;

RegardOccurred = false;

TaskBegin = 0;

TaskEnd = 0;

MaxDuringLevelThinkingCreation = System.Convert.ToInt32(AllDraw.THIScomboBoxMaxLevelText);

MinThinkingTreeDepth = 0;

//ThinkingChess.NumbersOfAllNode = 0;

ThinkingChess.FoundFirstMating = 0;

//Monitor Log File Appending ZFirst Line.

String state1 = "\n\t=====================================================================================================================================================================";

String state2 = "\n\tMovment Number:" + AllDraw.MovmentsNumber;

//String R = File.ReadAllText(Root + "\\Database\\Monitor.html");

//R = R.Replace("</body>", "");

//File.WriteAllText(Root + "\\Database\\Monitor.html", R);

////File.AppendAllText(Root + "\\Database\\Monitor.html", "\n\t" + state1 + "<br/>");

//File.AppendAllText(AllDraw.Root + "\\Database\\Monitor.html", state2 + "<br/>");

//File.AppendAllText(Root + "\\Database\\Monitor.html", "\n\t" + "</body>");

OutPut += state1;

OutPut += state2;

//Initiate Local and Global Variables.

//ThinkingChess.Sign = 1;

CurrentHuristic = Double.MinValue; ;

//SetprogressBarRefregitzValue(THIS.progressBarVerify, 0);

//THIS.progressBarVerify.Invalidate();

//SetprogressBarUpdate(THIS.progressBarVerify);

MaxHuristicxT = Double.MinValue;

DrawCastle.MaxHuristicxB = Double.MinValue;

DrawElefant.MaxHuristicxE = Double.MinValue;

DrawHourse.MaxHuristicxH = Double.MinValue;

DrawKing.MaxHuristicxK = Double.MinValue;

DrawMinister.MaxHuristicxM = Double.MinValue;

DrawSoldier.MaxHuristicxS = Double.MinValue;

MovementsAStarGreedyHuristicFoundT = false;

DrawTable = false;

ChessRules.CheckBrownObjectDangourFirstTimesOcured = false;

ChessRules.CheckGrayObjectDangourFirstTimesOcured = false;

}

//If There is Not AStarGreedy Huristic Boolean Chacked.

if (!AllDraw.AStarGreadyFirstSearch)

{

/\*AllDraw.StoreADraw.Clear();

int[,] Tab = null;

int[,] TablInit = null;

TableList.Add(Table);

ThinkingChess.NotSolvedKingDanger = false;

LoopHuristicIndex = 0;

//For All Pssible One.

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

{

//If Gray Order.

if (Order == 1)

{

OutPut = "\r\nChess Thinking AStarGreedy " + i.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else

{

OutPut = "\r\nChess Thinking AStarGreedy " + i.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

//Initaite Local Variables.

TablInit = new int[8, 8];

if (Order == 1)

a = Color.Gray;

else

a = Color.Brown;

int In = 0;

//Determine a Random Solders Objects.

do

{

//When Order is Gray Random is on Gray.

if (Order == 1)

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

else

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

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

//Initiate a DFept On Movments.

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

//Initaite a Local Varibales of Huristics.

Less = Double.MinValue;

//For Greater Than Zero ADraw Count Varibale.

//If Repetedly Movments Occurred.

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

{

//If Gray Order.

if (Order == 1)

{

OutPut = "\r\nGenetic Algorithm Begin AStarGreedy " + i.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else

{

OutPut = "\r\nGenetic Algirithm Begin AStarGreedy " + i.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

//Genetic Algorithm.

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

//Found of Table.

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

//Gray Order.

if (Order == 1)

{

OutPut = "\r\nGenetic Algorithm Finsished AStarGreedy " + i.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else

{

OutPut = "\r\nGenetic Algirithm Finished AStarGreedy " + i.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

}

//If Table Found.

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);

AStarGreedy++;

}

}

//Initaite Global Order Varibales By Local Varibales.

Order = DummyOrder;

ChessRules.CurrentOrder = Current;

DrawTable = true;

FoundATable = true;

return;\*/

}

else

{

//Initiate Local Varibales.

TableHuristic = null;

RW1 = -1;

CL1 = -1;

Ki1 = -1;

RW2 = -1;

CL2 = -1;

Ki2 = -1;

RW3 = -1;

CL3 = -1;

Ki3 = -1;

RW4 = -1;

CL4 = -1;

Ki4 = -1;

RW5 = -1;

CL5 = -1;

Ki5 = -1;

RW6 = -1;

CL6 = -1;

Ki6 = -1;

MaxHuristicAStarGreedytBackWard.Clear();

int[,] Tab = null;

if (!FOUND)

{

Object O7 = new Object();

lock (O7)

{

ThinkingChess.NotSolvedKingDanger = false;

}

LoopHuristicIndex = 0;

Less = Double.MinValue; ;

}

//THIS.Invoke((MethodInvoker)delegate()

{

MaxAStarGreedy = System.Convert.ToInt32(AllDraw.THIScomboBoxMaxLevelText);

AllDraw.AStarGreedyiLevelMax = MaxAStarGreedy;

AStarGreedyiLevelMax = System.Convert.ToInt32(AllDraw.THIScomboBoxMaxLevelText);

AllDraw.MaxAStarGreedyHuristicProgress = 6;

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

AllDraw.MaxAStarGreedyHuristicProgress += AllDraw.MaxAStarGreedyHuristicProgress \* 6;

//THIS.progressBarVerify.Maximum = 999999999;

increasedProgress = (int)((double)999999999 / (double)(AllDraw.MaxAStarGreedyHuristicProgress));

AStarGreedytMaxCount = (double)MaxAStarGreedy;

//SetprogressBarRefregitzValue(THIS.progressBarVerify, increasedProgress);

}//);

ChessRules.CurrentOrder = Current;

Order = DummyOrder;

int iiii = ii, jjjj = jj, Ord = Order;

int MaxAStarGreedy1 = MaxAStarGreedy;

int[,] Tabl = CloneATable(Table);

Color aaa = a;

InitiateAStarGreedytObject(MaxAStarGreedy1, iiii, jjjj, aaa, Tabl, Ord, false, FOUND, LeafAStarGreedy);

MinThinkingTreeDepth = MaxAStarGreedy - MinThinkingTreeDepth;

//Initaite Local Varibales.

Tab = new int[8, 8];

Less = Double.MinValue;

ChessRules.CurrentOrder = Current;

Order = DummyOrder;

//Thread.Sleep(1000);

Object OO = new Object();

lock (OO)

{

OutPut = "\r\nMinimum Thinking Tree Depth:" + MinThinkingTreeDepth.ToString() + "!";

}

TableHuristic = HuristicAStarGreedySearch(0, a, Order, false);

if (TableHuristic == null || ((TableZero(TableHuristic))))

{

try

{

Object OOOO = new Object();

lock (OOOO)

{

OutPut = "\r\nTable Zero.Possibly Full Penalty!";

}

//THIS.RefreshBoxText();

bool aa = UsePenaltyRegardMechnisamT;

UsePenaltyRegardMechnisamT = false;

//THISDummy = THISDummy.RemovePenalltyFromFirstBranches(Order);

RemovePenalltyFromFirstBranches(Order);

MaxAStarGreedy = 1;

AStarGreedyiLevelMax = 1;

Less = Double.MinValue;

//TableHuristic = THISDummy.HuristicAStarGreedySearchPenalties(0, a, Order, false);

//TableHuristic = THISDummy.HuristicAStarGreedySearch(0, a, Order, false);

TableHuristic = HuristicAStarGreedySearch(0, a, Order, false);

//THISDummy.UsePenaltyRegardMechnisamT = aa;

UsePenaltyRegardMechnisamT = aa;

}

catch (Exception t)

{

Log(t);

}

}

//If Table Found.

if (TableHuristic != null)

{

Object OOOOO = new Object();

lock (OOOOO)

{

if (Order == 1)

{

OutPut = "\r\nHuristic Find Best Movements AStarGreedy " + AStarGreedy.ToString() + " By Bob!";

//THIS.RefreshBoxText();

}

else

{

OutPut = "\r\nHuristic Find Best Movements AStarGreedy " + AStarGreedy.ToString() + " By Alice!";

//THIS.RefreshBoxText();

}

}

Order = DummyOrder;

ChessRules.CurrentOrder = Current;

}

else

{

//Clear AStarGreedy Varibales.

AllDraw.StoreADraw.Clear();

TableCurrent.Clear();

AStarGreedy = 0;

}

Order = DummyOrder;

ChessRules.CurrentOrder = Current;

//THISDummy.Dispose();

DrawTable = true;

FoundATable = true;

}

return TableHuristic;

}

//Identification of Illegal AStarGreedy First and Common Hurist Movments.

public bool IsEnemyThingsinStable(int[,] TableHuristic, int[,] TableAction, int Order)

{

//Iniatiet Local Variables.

int[,] Cromosom1 = TableHuristic;

int[,] Cromosom2 = TableAction;

bool and = true;

bool Find = false;

//bool Hit = false;

int FindNumber = 0;

int

CromosomRowFirst = -1, CromosomColumnFirst = -1,

CromosomRow = -1, CromosomColumn = -1;

//Initiate Local Variables.

//For All Table Home

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

{

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

{

//Gray Order.

if (Order == 1)

{

//Situation 11.

if (and)

{

//All The Brown Object Ignored.

if (Cromosom1[i, j] < 0 && Cromosom2[i, j] < 0)

continue;

}

else///Situation 2.

{

//All The Brown Ojects Ignored.

if (Cromosom1[i, j] < 0 || Cromosom2[i, j] < 0)

continue;

}

}

else//Brown Order.

{

//Situation 1.

if (and)

{

//All The Gray Objects Ignored.

if (Cromosom1[i, j] > 0 && Cromosom2[i, j] > 0)

continue;

}

else

{

//All The Gray Objects Ignored.

if (Cromosom1[i, j] > 0 || Cromosom2[i, j] > 0)

continue;

}

}

if (!(ArrangmentsChanged))

{

{

if (Order == 1 && j == 6 && i > 0 && i < 7)

{

if (((Cromosom2[i, j + 1] > 0) || (Cromosom2[i + 1, j + 1] > 0 && Cromosom1[i + 1, j + 1] < 0) || (Cromosom2[i - 1, j + 1] > 0 && Cromosom1[i - 1, j + 1] < 0) && Cromosom1[i, j] == 1))

{

CromosomRowFirst = i;

CromosomColumnFirst = j;

if (Cromosom2[i, j + 1] > 0)

{

CromosomRow = i;

CromosomColumn = j + 1;

}

else if (Cromosom2[i + 1, j + 1] > 0 && Cromosom1[i + 1, j + 1] < 0)

{

CromosomRow = i + 1;

CromosomColumn = j + 1;

}

else if (Cromosom2[i - 1, j + 1] > 0 && Cromosom1[i - 1, j + 1] < 0)

{

CromosomRow = i - 1;

CromosomColumn = j + 1;

}

Find = true;

FindNumber++;

AllDraw.SodierConversionOcuured = true;

}

}

else

if (Order == -1 && j == 1 && i > 0 && i < 7)

{

if (((Cromosom2[i, j - 1] < 0) || (Cromosom2[i + 1, j - 1] < 0 && Cromosom1[i + 1, j - 1] > 0) || (Cromosom2[i - 1, j - 1] < 0 && Cromosom1[i - 1, j - 1] < 0) && Cromosom1[i, j] == -1))

{

CromosomRowFirst = i;

CromosomColumnFirst = j;

if (Cromosom2[i, j - 1] > 0)

{

CromosomRow = i;

CromosomColumn = j - 1;

}

else if (Cromosom2[i + 1, j - 1] > 0 && Cromosom1[i + 1, j - 1] < 0)

{

CromosomRow = i + 1;

CromosomColumn = j - 1;

}

else if (Cromosom2[i - 1, j - 1] > 0 && Cromosom1[i - 1, j - 1] < 0)

{

CromosomRow = i - 1;

CromosomColumn = j - 1;

}

FindNumber++;

AllDraw.SodierConversionOcuured = true;

}

}

//Castles King Validity Condition.

if (Order == 1 && j == 0)

{

//Small Gray Castles King.

if (i == 6 && Cromosom2[i, j] == 6 && Cromosom2[i - 1, j] == 4 && Cromosom1[i, j] != 6 && Cromosom1[i - 1, j] != 4)

{

CromosomRowFirst = i - 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.SmallKingCastleGray = true;

CastlesKing = true;

}

else //Big Briges King Gray.

if (i == 2 && Cromosom2[i, j] == 6 && Cromosom2[i + 1, j] == 4 && Cromosom1[i, j] != 6 && Cromosom1[i + 1, j] != 4)

{

CromosomRowFirst = i + 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.BigKingCastleGray = true;

CastlesKing = true;

}

}

else if (j == 7)

{

//Small Castles King Brown.

if (i == 6 && Cromosom2[i, j] == -6 && Cromosom2[i - 1, j] == -4 && Cromosom1[i, j] != -6 && Cromosom1[i - 1, j] != -4)

{

CromosomRowFirst = i - 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.SmallKingCastleBrown = true;

CastlesKing = true;

}

else//Big Castles King Brown.

if (i == 2 && Cromosom2[i, j] == -6 && Cromosom2[i + 1, j] == -4 && Cromosom1[i, j] != -6 && Cromosom1[i + 1, j] != -4)

{

CromosomRowFirst = i + 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.BigKingCastleBrown = true;

CastlesKing = true;

}

}

}

}

else

{

{

if (Order == 1 && j == 1 && i > 0 && i < 7)

{

if (((Cromosom2[i, j - 1] > 0) || (Cromosom2[i + 1, j - 1] > 0 && Cromosom1[i + 1, j - 1] < 0) || (Cromosom2[i - 1, j - 1] > 0 && Cromosom1[i - 1, j - 1] < 0) && Cromosom1[i, j] == 1))

{

CromosomRowFirst = i;

CromosomColumnFirst = j;

if (Cromosom2[i, j - 1] > 0)

{

CromosomRow = i;

CromosomColumn = j - 1;

}

else if (Cromosom2[i + 1, j - 1] > 0 && Cromosom1[i + 1, j - 1] < 0)

{

CromosomRow = i + 1;

CromosomColumn = j - 1;

}

else if (Cromosom2[i - 1, j - 1] > 0 && Cromosom1[i - 1, j - 1] < 0)

{

CromosomRow = i - 1;

CromosomColumn = j - 1;

}

FindNumber++;

AllDraw.SodierConversionOcuured = true;

}

}

else

if (Order == -1 && j == 6 && i > 0 && i < 7)

{

if (((Cromosom2[i, j + 1] < 0) || (Cromosom2[i + 1, j + 1] < 0 && Cromosom1[i + 1, j + 1] > 0) || (Cromosom2[i - 1, j + 1] < 0 && Cromosom1[i - 1, j + 1] < 0) && Cromosom1[i, j] == -1))

{

CromosomRowFirst = i;

CromosomColumnFirst = j;

if (Cromosom2[i, j + 1] > 0)

{

CromosomRow = i;

CromosomColumn = j + 1;

}

else if (Cromosom2[i + 1, j + 1] > 0 && Cromosom1[i + 1, j + 1] < 0)

{

CromosomRow = i + 1;

CromosomColumn = j + 1;

}

else if (Cromosom2[i - 1, j + 1] > 0 && Cromosom1[i - 1, j + 1] < 0)

{

CromosomRow = i - 1;

CromosomColumn = j + 1;

}

Find = true;

FindNumber++;

AllDraw.SodierConversionOcuured = true;

}

}

//Castles King Validity Condition.

if (Order == 1 && j == 7)

{

//Small Gray Castles King.

if (i == 6 && Cromosom2[i, j] == 6 && Cromosom2[i - 1, j] == 4 && Cromosom1[i, j] != 6 && Cromosom1[i - 1, j] != 4)

{

CromosomRowFirst = i - 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.SmallKingCastleGray = true;

CastlesKing = true;

}

else //Big Briges King Gray.

if (i == 2 && Cromosom2[i, j] == 6 && Cromosom2[i + 1, j] == 4 && Cromosom1[i, j] != 6 && Cromosom1[i + 1, j] != 4)

{

CromosomRowFirst = i + 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.BigKingCastleGray = true;

CastlesKing = true;

}

}

else if (j == 0)

{

//Small Castles King Brown.

if (i == 6 && Cromosom2[i, j] == -6 && Cromosom2[i - 1, j] == -4 && Cromosom1[i, j] != -6 && Cromosom1[i - 1, j] != -4)

{

CromosomRowFirst = i - 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.SmallKingCastleBrown = true;

CastlesKing = true;

}

else//Big Castles King Brown.

if (i == 2 && Cromosom2[i, j] == -6 && Cromosom2[i + 1, j] == -4 && Cromosom1[i, j] != -6 && Cromosom1[i + 1, j] != -4)

{

CromosomRowFirst = i + 3;

CromosomColumnFirst = j;

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

ChessRules.BigKingCastleBrown = true;

CastlesKing = true;

}

}

}

}

//When To Same Location Tbles are Different in Gen.

if (Cromosom1[i, j] != Cromosom2[i, j])

{

//When Cromosom 2 is Empty.

if (Cromosom2[i, j] == 0)

{

//Initiate Location of Table.

continue;

}

else

{

//Situation 1.0

if (and)

{

//When Cromosom1 Current Location is Empty.

if (Cromosom1[i, j] == 0)

{

//Initiate Location of Gen.

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

continue;

}

}

}

//Store Location of Gen and Calculate Gen Numbers.

CromosomRow = i;

CromosomColumn = j;

Find = true;

FindNumber++;

}

}

}

//If Gen Foundation is Valid.

if (((FindNumber == 1 || FindNumber == 2) && Find) || CastlesKing || AllDraw.SodierConversionOcuured)

return Find;

//Gen Not Found.

return false;

}

List<int[]> WhereNumbers(String Tag)

{

List<int[]> TagList = new List<int[]>();

for (int i = 0; i < Tag.Length; i++)

{

if (i + 1 < Tag.Length)

{

for (int j = i + 1; j < i + RefrigtzDLL.AllDraw.MaxAStarGreedy.ToString().Length + 1; j++)

{

try

{

int A = System.Convert.ToInt32(Tag.Substring(i, j - i));

if (A >= 0 && A <= AllDraw.MaxAStarGreedy)

{

int[] Loc = new int[2];

Loc[0] = i;

Loc[1] = j - i;

TagList.Add(Loc);

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

return TagList;

}

String CreateHtmlTag(String Tag)

{

Object O = new Object();

lock (O)

{

//List<int[]> List = new List<int[]>();

//List = WhereNumbers(Tag);

//for (int i = 0; i < List.Count; i++)

//Tag = Tag.Replace(Tag.Substring(List[i][0], List[i][1]), "<font Color=\"Gold\">" + Tag.Substring(List[i][0], List[i][1]) + "</font>");

if (Tag.Contains("Thinking"))

Tag = Tag.Replace("Thinking", "<font Color=\"Green\">" + "Thinking" + "</font>");

if (Tag.Contains("Perception"))

Tag = Tag.Replace("Perception", "<font Color=\"Green\">" + "Perception" + "</font>");

if (Tag.Contains("Bob"))

Tag = Tag.Replace("Bob", "<font Color=\"Gray\">" + "Bob" + "</font>");

if (Tag.Contains("Alice"))

Tag = Tag.Replace("Alice", "<font Color=\"Brown\">" + "Brown" + "</font>");

if (Tag.Contains("AstarGreedy "))

Tag = Tag.Replace("AstarGreedy ", "<font Color=\"Yellow\">" + "AstarGreedy " + "</font>");

if (Tag.Contains("Level"))

Tag = Tag.Replace("Level", "<font Color=\"Blue\">" + "Level" + "</Font>");

/\* if (Tag.Contains("Soldeir"))

Tag = Tag.Replace("Soldeir", "<font Color=\"Silver\">" + "Soldeir" + "</font>");

else

if (Tag.Contains("Elephant"))

Tag = Tag.Replace("Elephant", "<font Color=\"Silver\">" + "Elephant" + "</font>");

else

if (Tag.Contains("Hourse"))

Tag = Tag.Replace("Hourse", "<font Color=\"Silver\">" + "Hourse" + "</font>");

else

if (Tag.Contains("Castle"))

Tag = Tag.Replace("Castle", "<font Color=\"Silver\">" + "Castle" + "</font>");

else

if (Tag.Contains("Minister"))

Tag = Tag.Replace("Minister", "<font Color=\"Silver\">" + "Minister" + "</font>");

else

if (Tag.Contains("King"))

Tag = Tag.Replace("King", "<font Color=\"Silver\">" + "King" + "</font>");

\*/

String R = "<font Color=\"Red\">" + Tag + "</font>";

return R;

}

}

}

}

//End of Documentation.