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

\* Main class of Form Inherited of Form Main in the C#.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ramin Edjlal\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Mouse Event Handling Error\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Some Soldiers Loos Location And Virutuallization at Click\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Some Codes Lines Frizes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Hit things Abnormal behaviour\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Virtualization Things Error\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* No Reason Logically For Equality of 'SoldierP' and 'Soldier'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Problem of Sameness of Hit Enemys Solved.No Reason For Equality of 'Soldier' and 'SoldierP'RS--\*\*\*\*\*\*\*\*\*\*\*\*(-+)

\* No Problem For Hiting\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Castles of Gray Color Conversion to Kings Brown\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Castles Conversion To King Brown Abnormally no Reasonably\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)

\* Color Conversion In Virtualization Hit Enemy\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Need To Find Enemy Detection on Current OrderPlate\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* CheckMate an Check Dosn't Work\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* No Movments By Computer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Illegal Virtualization. The Thinking By 'Alice' (My Computer) RefrigtzDLL.ChessRules Misleading\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* permutative Constant Huristic Results\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* In Existence of Adding Suported Huristic Constant Huristic Result Detection\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* OrderPlate Not Configured\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Non Color Hourse Hit Assignment Misleading(Abnormal)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Undetected Error Table Content Malfunction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* MalFunction Movments Greate than 5 by 'Alice'.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* 'Check' Second Time 'Alice' MalFanction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* 'Check' 'Alice' Mechnisam Failure\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

'\* CheckMate' of Unsatisfied in 'Alice'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* CheckMate Dosn't Recognized.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Virtualization Filed (Not Responding) at Indpencdency State\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Unable To Draw Refrigtz.Timer Content at Tow Picture Box\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Refrigtz.Timer Working Hardly.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Dead Lock In Drawing Images.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Misleading Thread OrderPlate And Time Sharing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Thinkings Taking a lot of Time.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* AntiVirus Protextion and Existance Caused to Reduce Speed of Thinking and lead to Lose.\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* No Programatically Reason For Speed Reduction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Mybe Windows Filrewall Has no been correctly Arranges to reduce speed.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Method on Leave not Work.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* OrderPlate Reader Table MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*(+)

\* Wrong Sysntax To Read.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Some Tables of Hitting Tow Person Are Missing.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\_\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\_)

\* Some Syntaxes at Table Read Are Missing.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\_\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\_)

\* Chess Refregitz Sometimes Not Responding due to Cpu Power Non Ability.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Chess Refregitz Sometimes stop working.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* 1395/1/16\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* OrderPlate MalFunction.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RS\*\*\*\*\*0.12\*\*4\*\*Managements and Cuation Programing\*\*{+}

\* Virtualization Error.No Reason For MalFunctionla Operation of Program at Sysntax and Orde.\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Cause Sensitive Problems of 'Check' And 'CheckMate By 'Alice' is borring at StateCP.\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Table Content Misleading.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Documentation Faulting On Removing Detials.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Loading Games to Continue failed Unreasonly.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC-OK.

\* "Thread is dead.The state can not be accessed.".\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Illegal Syntax Mechanisam Detection By Genetic Algorithm.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC-Ok.

\* 'MaxCurrentMovmentsNumber' Changes Illegal to Reduced.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*{\*}QC-OK.

\* Arranagment Up-Down Changes caused to Born of Some Objects UnSupported Attractively.\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Table TableListAction Genetic Algorithm MalFunctionally Illegal Data.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Castles King Insertion Overllaping Previous Enemy Movments of Person Movments.\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Statisticer Loader and Acter Castles King Overview Failed.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Castles King Continue With Illegal Movments at Statisticer.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC-OK.

\* Calling of All Events From Pressing a Single Button.Mal Function of Database Leading.\*\*\*\*\*\*CU\*\*\*\*\*0.88\*\*1\*\*Risk Control\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<\*>QC\_OK

\* Proccess Calling of Stockfish8 not run Comlpetely.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_OK

\* Can not Send Arguments via Foriegn Sites Studies. Problem Misleading.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_OK

\* Serialization and Deserialization Wrong Config vars conflict.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_OK

\* MalFunctionally break heart of Access Database Configuration Table.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_OK

\* Parallleism occured Misfaulting in all Opetions of this work.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_BAD

\* Mal Function in Drawing some part of objects in somthings of games.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*)QC\_BAD

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Diagnostics;

using System.Threading;

using System.Data.OleDb;

using System.Media;

using System.IO;

using System.Threading.Tasks;

namespace Refrigtz

{

[Serializable]

//Constructor

public partial class FormRefrigtz : Form

{

//Variables That to be Included at RefregitzDLL

//Initiate Variables.

String Out = "";

bool LoadConvertedTable = false;

bool LoadedDLL = false;

bool WaitOn = true;

String comboBoxMaxLevelText = "";

bool RemoveUncomStock = false;

bool TimerIniataite = false;

bool PersonWithStockfish = false;

const string PieceToChar = "kqrnbp PBNRQK";

int StockMoveBase = 0;

int FenCastling = -1;//0 for small castling 1 for big castling -1 for non castling.

int StockMove = 1;

//Initiate Global Variable.

//public bool DisableTemporarlyTimerUpdate = false;

public bool ConvertWait = false;

public double MaxHuristicx = Double.MinValue;

public bool MovementsAStarGreedyHuristicFound = false;

public bool IInoreSelfObjects = false;

public bool UsePenaltyRegardMechnisam = false;

public bool PredictHuristic = false;

public bool OnlySelf = false;

public bool AStarGreedyHuristic = false;

public bool BestMovments = false;

bool Loaded = false;

bool FirstMovmentOnLoad = true;

int DrawNumber = 0;

bool FullGame = false;

static int GrayCountHiiting = 1;

static int BrownCountHiiting = 17;

List<PictureBox> ApList = new List<PictureBox>();

public static int X1 = 0, Y1 = 0;

public static int X2 = 0, Y2 = 0;

bool BobWithStockfishFinished = false;

public static bool Stockfish = false;

bool RefregitzisCurrent = true;

bool SettingPRFALSE = false;

public static bool ProfesionalWithComputer = false;

public bool TimersSet = true;

public static bool GameStarted = false;

public bool ArrangmentsChanged = true;

public static int MaxCurrentMovmentsNumber = 1;

public static bool ErrorTrueMonitorFalse = true;

Thread tM = null;

bool Clicked = true;

public static int MaxAStarGreedyHuristicProgress = 0;

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

public bool MouseClicked = false;

public static bool Blitz = false;

DateTime PreviousTime;

static bool LoadAG = false;

static bool \_1 = false;

static bool \_2 = false;

static bool \_3 = false;

static bool \_4 = false;

static bool Hideag = false;

bool exit = false;

public FormSelect Sec = new FormSelect();

bool AllDrawLoad = false;

Thread AllOperate = null;

static bool PaintingPaused = false;

static bool PaintedPaused = false;

static bool UpdateConfigurationTableVal = false;

static bool NewTable = false;

OleDbConnection bookConn;

OleDbCommand oleDbCmd = new OleDbCommand();

private OleDbCommand oleDbCmdUser = new OleDbCommand();

Image TimerImage = null;

Graphics g1 = null;

Image TimerImage1 = null;

Graphics g2 = null;

Graphics g = null;

Image ChessTable = null;

public static int LastRow = -1;

public static int LastColumn = -1;

public static int NextRow = -1;

public static int NextColumn = -1;

Thread t1 = null;

Thread t2 = null;

Thread t3 = null;

Thread t4 = null;

Thread TTimerSet;

public static bool LoadedTable = false;

bool GrayWinner = false;

bool BrownWiner = false;

public Refrigtz.Timer TimerText = null;

public Refrigtz.Timer GrayTimer = null;

public Refrigtz.Timer BrownTimer = null;

public static int MovmentsNumber = 0;

public static int MovmentsNumberMax = 0;

public static bool EndOfGame = false;

bool Maximize = false;

static int RowP = 0, ColP = 0, RowS = 0, ColS = 0;

bool BobSection = false;

bool AliceSection = false;

public static bool Person = false;

static int CurrentKind = 0;

public static bool StateCC = false;//Computer With Computer

public static bool StateCP = true;//Person With Computer

public static bool StateGe = false;//For Genetic Games.

public RefrigtzDLL.AllDraw Draw;

public static int OrderPlate = 1;

// int RefrigtzDLL.AllDraw.MouseClick;

int Soldier;

int Elefant;

int Hourse;

int Castle;

int Minister;

int King;

float RowClickP = -1, ColumnClickP = -1;

int RowClick = -1, ColumnClick = -1;

float RowRealesedP = -1, ColumnRealeasedP = -1;

float RowRealesed = -1, ColumnRealeased = -1;

public int[,] Table ={

{ -4, -1, 0, 0, 0, 0, 1, 4 },

{ -3, -1, 0, 0, 0, 0, 1, 3 },

{ -2, -1, 0, 0, 0, 0, 1, 2 },

{ -6, -1, 0, 0, 0, 0, 1, 5 },

{ -5, -1, 0, 0, 0, 0, 1, 6 },

{ -2, -1, 0, 0, 0, 0, 1, 2 },

{ -3, -1, 0, 0, 0, 0, 1, 3 },

{ -4, -1, 0, 0, 0, 0, 1, 4 }

};

List<char> fenS = new List<char>("position fen rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1\0");

//FormRefrigtz THIs = null;

String connParam = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + Root + "\\" + "Database\\CurrentBank.accdb;;Persist Security Info=False; Jet OLEDB:Database Password='!HN#BGHHN&N$G$V4'";

public bool LoadTree = false;

Thread tttt = null;

Thread ttt = null;

//Error Handling.

static void Log(Exception ex)

{

try

{

Object a = new Object();

lock (a)

{

string stackTrace = ex.ToString();

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

}

}

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

}

//Constructor No2

public FormRefrigtz(bool AllDra)

{

AllDrawLoad = AllDra;

InitializeComponent();

X1 = textBoxText.Location.X;

Y1 = textBoxText.Location.Y - 30;

X2 = textBoxText.Location.X;

Y2 = textBoxText.Location.Y + 30 + textBoxText.Size.Height;

if (!AllDrawLoad)

{

this.pictureBoxRefrigtz.Size = new Size((this.pictureBoxRefrigtz.Width / 8) \* 8, (this.pictureBoxRefrigtz.Height / 8) \* 8);

t1 = new Thread(new ThreadStart(AliceWithPerson));

t2 = new Thread(new ThreadStart(BobAction));

t3 = new Thread(new ThreadStart(AliceAction));

t4 = new Thread(new ThreadStart(GeneticAction));

tttt = new Thread(new ThreadStart(SetRefregitzDLL));

ttt = new Thread(new ThreadStart(SetNodesCount));

}

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

{

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

{

RefrigtzDLL.AllDraw.TableVeryfy[i, j] = Table[i, j];

RefrigtzDLL.AllDraw.TableVeryfyConst[i, j] = Table[i, j];

}

}

RefrigtzDLL.ThinkingChess.LearniningTable = new LearningMachine.NetworkQuantumLearningKrinskyAtamata(8, 8, 8);

try

{

if (System.IO.File.Exists(Root + "\\Database\\CurrentBank.accdb"))

{

bookConn = new OleDbConnection(connParam);

bookConn.Open();

oleDbCmd.Connection = bookConn;

}

}

catch (FileNotFoundException t)

{

Log(t);

}

}

//Acccess Point

//Syncronization Between Class Dlls.

private void SetTimer()

{

do

{

Thread.Sleep(1);

Object O = new Object();

lock (O)

{

//while (!StateCC && !StateCP && !StateGe)

Refrigtz.Timer.AStarGreadyFirstSearch = RefrigtzDLL.AllDraw.AStarGreadyFirstSearch;

Refrigtz.Timer.AStarGreedytiLevelMax = RefrigtzDLL.AllDraw.AStarGreedyiLevelMax;

Refrigtz.Timer.UseDoubleTime = RefrigtzDLL.AllDraw.UseDoubleTime;

Refrigtz.Timer.StoreAllDrawCount = RefrigtzDLL.AllDraw.StoreADraw.Count;

}

}

while (true);

}

delegate void SetlableRefregitzMaxValueCalBack(Label Refregitz, String value);

private void SetlableRefregitzMaxValue(Label Refregitz, String value)

{

try

{

if (Refregitz.InvokeRequired)

{

SetlableRefregitzMaxValueCalBack d = new SetlableRefregitzMaxValueCalBack(SetlableRefregitzMaxValue);

Refregitz.Invoke(new Action(() => Refregitz.Text = value));

Refregitz.Refresh();

}

else

{

Refregitz.Text = value;

Refregitz.Refresh();

}

}

catch (Exception t)

{

Log(t);

}

}

delegate void SetprogressBarRefregitzValueCalBack(ProgressBar Refregitz, Int32 value);

private void SetprogressBarRefregitzValue(ProgressBar Refregitz, Int32 value)

{

try

{

if (Refregitz.InvokeRequired)

{

SetprogressBarRefregitzValueCalBack d = new SetprogressBarRefregitzValueCalBack(SetprogressBarRefregitzValue);

Refregitz.Invoke(new Action(() => Refregitz.Value = value));

}

else

{

Refregitz.Value = value;

}

}

catch (Exception t)

{

Log(t);

}

}

delegate void SetPictureBoxRefregitzUpdateCalBack(PictureBox Refregitz);

private void SetPrictureBoxRefregitzUpdate(PictureBox Refregitz)

{

try

{

if (Refregitz.InvokeRequired)

{

SetPictureBoxRefregitzUpdateCalBack d = new SetPictureBoxRefregitzUpdateCalBack(SetPrictureBoxRefregitzUpdate);

Refregitz.Invoke(new Action(() => Refregitz.Update()));

}

else

{

Refregitz.Update();

}

}

catch (Exception t)

{

Log(t);

}

}

delegate void SetPictureBoxRefregitzInvalidateCalBack(PictureBox Refregitz);

private void SetPrictureBoxRefregitzInvalidate(PictureBox Refregitz)

{

try

{

if (Refregitz.InvokeRequired)

{

SetPictureBoxRefregitzInvalidateCalBack d = new SetPictureBoxRefregitzInvalidateCalBack(SetPrictureBoxRefregitzInvalidate);

Refregitz.Invoke(new Action(() => Refregitz.Invalidate()));

}

else

{

Refregitz.Invalidate();

}

}

catch (Exception t)

{

Log(t);

}

}

delegate void SetprogressBarRefregitzMaxValueCalBack(ProgressBar Refregitz, Int32 value);

private void SetprogressBarRefregitzMaxValue(ProgressBar Refregitz, Int32 value)

{

try

{

if (Refregitz.InvokeRequired)

{

SetprogressBarRefregitzMaxValueCalBack d = new SetprogressBarRefregitzMaxValueCalBack(SetprogressBarRefregitzMaxValue);

Refregitz.Invoke(new Action(() => Refregitz.Maximum = value));

}

else

{

Refregitz.Maximum = value;

}

}

catch (Exception t)

{

Log(t);

}

}

//Boolean Setting of Illustration at Slected Object Rules.

bool[,] VeryFye(int[,] Table, int Order, Color a)

{

int Cdummy = RefrigtzDLL.ChessRules.CurrentOrder;

if (OrderPlate == 1)

RefrigtzDLL.ChessRules.CurrentOrder = 1;

else

RefrigtzDLL.ChessRules.CurrentOrder = -1;

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

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

{

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

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, Table[(int)RowClickP, (int)ColumnClickP], Table, Order, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, i, j, a, Table[(int)RowClickP, (int)ColumnClickP]))

Tab[i, j] = true;

}

}

RefrigtzDLL.ChessRules.CurrentOrder = Cdummy;

return Tab;

}

//Delegate Of Form Close Visibility.

delegate void SetCloseVisibleCallback();

public void SetCloseVisible()

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

try

{

SetCloseVisibleCallback d = new SetCloseVisibleCallback(SetCloseVisible);

this.Invoke(new Action(() => this.Close()));

}

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

}

else

{

try

{

this.Close();

}

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

}

}

void SetImageOfPictrueBox(ref PictureBox Ap, int

Hit)

{

if (Hit > 0)

{

if (Hit == 1)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\SG.png");

else

if (Hit == 2)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\EG.png");

else

if (Hit == 3)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\HG.png");

else

if (Hit == 4)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\BrG.png");

else

if (Hit == 5)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\MG.png");

}

else

{

if (Hit == -1)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\SB.png");

else

if (Hit == -2)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\EB.png");

else

if (Hit == -3)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\HB.png");

else

if (Hit == -4)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\BrB.png");

else

if (Hit == -5)

Ap.Image = Image.FromFile(Root + "\\Images\\Fit\\Small\\MB.png");

}

//Ap.BringToFront();

}

public void SetObjectInPictureBox(int R, int C)

{

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R, C];

if (Hit > 0)

{

if (GrayCountHiiting == 1)

SetImageOfPictrueBox(ref pictureBox1, Hit);

else

if (GrayCountHiiting == 2)

SetImageOfPictrueBox(ref pictureBox2, Hit);

else

if (GrayCountHiiting == 3)

SetImageOfPictrueBox(ref pictureBox3, Hit);

else

if (GrayCountHiiting == 4)

SetImageOfPictrueBox(ref pictureBox4, Hit);

else

if (GrayCountHiiting == 5)

SetImageOfPictrueBox(ref pictureBox5, Hit);

else

if (GrayCountHiiting == 6)

SetImageOfPictrueBox(ref pictureBox6, Hit);

else

if (GrayCountHiiting == 7)

SetImageOfPictrueBox(ref pictureBox7, Hit);

else

if (GrayCountHiiting == 8)

SetImageOfPictrueBox(ref pictureBox8, Hit);

else

if (GrayCountHiiting == 9)

SetImageOfPictrueBox(ref pictureBox9, Hit);

else

if (GrayCountHiiting == 10)

SetImageOfPictrueBox(ref pictureBox10, Hit);

else

if (GrayCountHiiting == 11)

SetImageOfPictrueBox(ref pictureBox11, Hit);

else

if (GrayCountHiiting == 12)

SetImageOfPictrueBox(ref pictureBox12, Hit);

else

if (GrayCountHiiting == 13)

SetImageOfPictrueBox(ref pictureBox13, Hit);

else

if (GrayCountHiiting == 14)

SetImageOfPictrueBox(ref pictureBox14, Hit);

else

if (GrayCountHiiting == 15)

SetImageOfPictrueBox(ref pictureBox15, Hit);

else

if (GrayCountHiiting == 16)

SetImageOfPictrueBox(ref pictureBox16, Hit);

GrayCountHiiting++;

}

else if (Hit < 0)

{

if (BrownCountHiiting == 17)

SetImageOfPictrueBox(ref pictureBox17, Hit);

else

if (BrownCountHiiting == 18)

SetImageOfPictrueBox(ref pictureBox18, Hit);

else

if (BrownCountHiiting == 19)

SetImageOfPictrueBox(ref pictureBox19, Hit);

else

if (BrownCountHiiting == 20)

SetImageOfPictrueBox(ref pictureBox20, Hit);

else

if (BrownCountHiiting == 21)

SetImageOfPictrueBox(ref pictureBox21, Hit);

else

if (BrownCountHiiting == 22)

SetImageOfPictrueBox(ref pictureBox22, Hit);

else

if (BrownCountHiiting == 23)

SetImageOfPictrueBox(ref pictureBox23, Hit);

else

if (BrownCountHiiting == 24)

SetImageOfPictrueBox(ref pictureBox24, Hit);

else

if (BrownCountHiiting == 25)

SetImageOfPictrueBox(ref pictureBox25, Hit);

else

if (BrownCountHiiting == 26)

SetImageOfPictrueBox(ref pictureBox26, Hit);

else

if (BrownCountHiiting == 27)

SetImageOfPictrueBox(ref pictureBox27, Hit);

else

if (BrownCountHiiting == 28)

SetImageOfPictrueBox(ref pictureBox28, Hit);

else

if (BrownCountHiiting == 29)

SetImageOfPictrueBox(ref pictureBox29, Hit);

else

if (BrownCountHiiting == 30)

SetImageOfPictrueBox(ref pictureBox30, Hit);

else

if (BrownCountHiiting == 31)

SetImageOfPictrueBox(ref pictureBox31, Hit);

else

if (BrownCountHiiting == 32)

SetImageOfPictrueBox(ref pictureBox32, Hit);

BrownCountHiiting++;

}

}

void SetArragngmen()

{

/\* do

{

Thread.Sleep(1000);

ArrangmentsChanged = true;

} while (true);

\*/

}

void SetNodesCount()

{

Double Store = 0;

Double Count = 1;

Int64 TimeElapsed = DateTime.Now.Hour \* 3600000 + DateTime.Now.Minute \* 60000 + DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond;

Int64 FirstTime = DateTime.Now.Hour \* 3600000 + DateTime.Now.Minute \* 60000 + DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond;

do

{

Object O = new Object();

lock (O)

{

Int64 FirstTimeD = DateTime.Now.Hour \* 3600000 + DateTime.Now.Minute \* 60000 + DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond;

Int64 FirstNode = RefrigtzDLL.ThinkingChess.NumbersOfAllNode;

Thread.Sleep(50);

Int64 EndTime = DateTime.Now.Hour \* 3600000 + DateTime.Now.Minute \* 60000 + DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond;

Double S = ((double)RefrigtzDLL.ThinkingChess.NumbersOfAllNode - FirstNode) / (((double)EndTime - (double)FirstTimeD) / 1000.0);

Store = ((Store \* Count) + S) / (Count + 1.0);

Count++;

if (RefrigtzDLL.AllDraw.ActionStringReady)

SetlableRefregitzMaxValue(labelNodesCount, RefrigtzDLL.ThinkingChess.NumbersOfAllNode.ToString() + " at time " + ((int)(Store)).ToString() + "N/s and by Elapsed time " + ((int)((EndTime - TimeElapsed) / 1000)).ToString() + " s by string" + RefrigtzDLL.AllDraw.ActionString + " By CheckMate Count " + RefrigtzDLL.ThinkingChess.FoundFirstMating.ToString() + " For Order " + RefrigtzDLL.AllDraw.OrderPlate.ToString());

else

SetlableRefregitzMaxValue(labelNodesCount, RefrigtzDLL.ThinkingChess.NumbersOfAllNode.ToString() + " at time " + ((int)(Store)).ToString() + "N/s and by Elapsed time " + ((int)((EndTime - TimeElapsed) / 1000)).ToString() + " s" + " By CheckMate Count " + RefrigtzDLL.ThinkingChess.FoundFirstMating.ToString() + " For Order " + RefrigtzDLL.AllDraw.OrderPlate.ToString());

//labelNodesCount.Refresh();

}

} while (true);

}

void SetRefregitzDLL()

{

bool Set = false;

do

{

Object o = new Object();

lock (o)

{

RefrigtzDLL.AllDraw.Root = Root;

RefrigtzDLL.AllDraw.OrderPlate = OrderPlate;

RefrigtzDLL.AllDraw.Blitz = Blitz;

ConvertWait = RefrigtzDLL.AllDraw.ConvertWait;

RefrigtzDLL.AllDraw.Stockfish = Stockfish;

RefrigtzDLL.AllDraw.Person = Person;

RefrigtzDLL.AllDraw.THISSecradioButtonGrayOrderChecked = Sec.radioButtonGrayOrder.Checked;

RefrigtzDLL.AllDraw.THISSecradioButtonBrownOrderChecked = Sec.radioButtonBrownOrder.Checked;

RefrigtzDLL.AllDraw.MovmentsNumber = MovmentsNumber;

while (!(RefrigtzDLL.ChessRules.ObjectHittedRow != -1 && RefrigtzDLL.ChessRules.ObjectHittedColumn != -1) &&

(!RefrigtzDLL.AllDraw.ActionStringReady) &&

(!(RefrigtzDLL.AllDraw.OutPut != "")) &&

(!(RefrigtzDLL.AllDraw.ConvertedKind == -1)) &&

(!StateCC) &&

(!StateCP))

{

Thread.Sleep(1);

}

if (RefrigtzDLL.ChessRules.ObjectHittedRow != -1 && RefrigtzDLL.ChessRules.ObjectHittedColumn != -1)

{

SetObjectInPictureBox(RefrigtzDLL.ChessRules.ObjectHittedRow, RefrigtzDLL.ChessRules.ObjectHittedColumn);

RefrigtzDLL.ChessRules.ObjectHittedRow = -1;

RefrigtzDLL.ChessRules.ObjectHittedColumn = -1;

}

if (RefrigtzDLL.AllDraw.ActionStringReady)

{

SetBoxText(RefrigtzDLL.AllDraw.ActionString);

RefreshBoxText();

RefrigtzDLL.AllDraw.ActionString = "";

RefrigtzDLL.AllDraw.ActionStringReady = false;

}

if (RefrigtzDLL.AllDraw.OutPut != "")

{

SetBoxText(RefrigtzDLL.AllDraw.OutPut);

RefreshBoxText();

RefrigtzDLL.AllDraw.OutPut = "";

}

if (RefrigtzDLL.AllDraw.ConvertedKind == -1)

{

FormُSelectItems.Items = -1;

FormُSelectItems A = new FormُSelectItems();

A.ShowDialog();

while (RefrigtzDLL.AllDraw.ConvertedKind == -1) { RefrigtzDLL.AllDraw.ConvertedKind = FormُSelectItems.Items; Thread.Sleep(10); }

}

if (LoadedDLL)

{

SetcomboBoxText();

LoadedDLL = false;

}

//public static AllDraw THISDummy;

RefrigtzDLL.AllDraw.StateCP = StateCP;

if (StateCP && (!Stockfish))

{

if (OrderPlate == -1 && Sec.radioButtonGrayOrder.Checked)

{

LastRow = RefrigtzDLL.AllDraw.LastRow;

LastColumn = RefrigtzDLL.AllDraw.LastColumn;

NextRow = RefrigtzDLL.AllDraw.NextRow;

NextColumn = RefrigtzDLL.AllDraw.NextRow;

}

else if (OrderPlate == 1 && Sec.radioButtonBrownOrder.Checked)

{

LastRow = RefrigtzDLL.AllDraw.LastRow;

LastColumn = RefrigtzDLL.AllDraw.LastColumn;

NextRow = RefrigtzDLL.AllDraw.NextRow;

NextColumn = RefrigtzDLL.AllDraw.NextRow;

}

}

else

if (StateCP && Stockfish)

{

if (OrderPlate == -1 && Sec.radioButtonGrayOrder.Checked)

{

LastRow = RefrigtzDLL.AllDraw.LastRow;

LastColumn = RefrigtzDLL.AllDraw.LastColumn;

NextRow = RefrigtzDLL.AllDraw.NextRow;

NextColumn = RefrigtzDLL.AllDraw.NextRow;

}

}

//System.Threading.Thread.Sleep(1);

}

} while (true);

}

//Load Refregitz Form.

private void Form1\_Load(object sender, EventArgs e)

{

bool DrawDrawen = false;

//Set and syncronization with dynamic refregitzdll.

tttt = new Thread(new ThreadStart(SetRefregitzDLL));

ttt = new Thread(new ThreadStart(SetNodesCount));

AllOperate = new Thread(new ThreadStart(AllOperations));

BrownTimer = new Refrigtz.Timer(false);

GrayTimer = new Refrigtz.Timer(false);

TimerText = new Refrigtz.Timer(true);

BrownTimer.TimerInitiate();

GrayTimer.TimerInitiate();

TTimerSet = new Thread(new ThreadStart(SetTimer));

TTimerSet.Start();

TimerText.TimerInitiate();

TimerText.StartTime();

if (Blitz)

{

BrownTimer = new Refrigtz.Timer(true);

GrayTimer = new Refrigtz.Timer(true);

if (OrderPlate == 1)

{

GrayTimer.StartTime();

BrownTimer.StopTime();

}

else

{

GrayTimer.StopTime();

BrownTimer.StartTime();

}

}

else if (FullGame)

{

BrownTimer = new Refrigtz.Timer(true);

GrayTimer = new Refrigtz.Timer(true);

if (OrderPlate == 1)

{

GrayTimer.StartTime();

BrownTimer.StopTime();

}

else

{

GrayTimer.StopTime();

BrownTimer.StartTime();

}

}

else if (StateCC)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

if (OrderPlate == 1)

{

BobSection = true;

AliceSection = false;

GrayTimer.StartTime();

}

else

{

BobSection = false;

AliceSection = true;

BrownTimer.StartTime();

}

UpdateConfigurationTableVal = true;

}

else if (Stockfish)

{

BrownTimer = new Refrigtz.Timer(true);

GrayTimer = new Refrigtz.Timer(true);

if (OrderPlate == 1)

{

GrayTimer.StartTime();

BrownTimer.StopTime();

BobSection = true;

AliceSection = false;

GrayTimer.StartTime();

}

else

{

BrownTimer.StartTime();

GrayTimer.StopTime();

BobSection = false;

AliceSection = true;

BrownTimer.StartTime();

}

//

//

}

//Wehn no need to load.

if (File.Exists(Root + "\\Run.txt"))

{

AllDrawLoad = false;

NewTable = false;

}

//Determne of several refrigitz proccess.

if (!LoadAG)

{

List<Process> a = new List<Process>();

a.AddRange(Process.GetProcessesByName("Refrigtz"));

if (a.Count > 1)

{

if (System.Windows.Forms.MessageBox.Show(null, "New Instant Of Refregitz!", "New Instant", MessageBoxButtons.YesNo) == DialogResult.No)

{

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

{

try

{

a[i].Kill();

exitToolStripMenuItem\_Click(sender, e);

}

catch (Exception t) { Log(t); Application.ExitThread(); }

}

}

}

}

//When direcrories not exist.

if (!Directory.Exists(Root + "\\DataBase"))

{

if (!Directory.Exists(Root + "\\DataBase\\MainBank"))

{

Directory.CreateDirectory(Root + "\\DataBase\\MainBank");

File.Move(Root + "\\ChessBank.accdb", Root + "\\DataBase\\MainBank\\ChessBank.accdb");

}

}

if (!Directory.Exists(Root + "\\Images"))

{

if (!Directory.Exists(Root + "\\Images\\Original"))

{

Directory.CreateDirectory(Root + "\\Images\\Original");

}

}

//When file dos't exist.

if (!AllDrawLoad)

{

if (!System.IO.File.Exists(Root + "\\Database\\CurrentBank.accdb"))

{

System.IO.File.Copy(Root + "\\Database\\MainBank\\ChessBank.accdb", Root + "\\Database\\CurrentBank.accdb");

if (!File.Exists(Root + "\\Database\\Monitor.html"))

System.IO.File.Copy(Root + "\\Database\\MainBank\\Monitor\_Log.html", Root + "\\Database\\Monitor.html");

if (File.Exists("List.txt"))

File.Delete("List.txt");

bookConn = new OleDbConnection(connParam);

bookConn.Open();

oleDbCmd.Connection = bookConn;

InsertTableAtDataBase(Table);

CreateConfigurationTable();

}

else

{

//When movments not occured.

if (!NewTable)

{

UpdateConfigurationTableVal = false;

//When Configuration is Allowed Read Configuration.

ReadConfigurationTable();

//Set Configuration To True for some unknown reason!.

UpdateConfigurationTableVal = true;

//Read Last Table and Set MovementNumber

Table = ReadTable(0, ref MovmentsNumber);

//Load AllDraw.asd

DrawDrawen = (new TakeRoot()).Load(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

}

else

{

int iii = 0;

do { iii++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

System.IO.File.Copy(Root + "\\Database\\MainBank\\ChessBank.accdb", Root + "\\Database\\CurrentBank.accdb");

System.IO.File.Copy(Root + "\\Database\\MainBank\\Monitor\_Log.html", Root + "\\Database\\Monitor.html");

InsertTableAtDataBase(Table);

CreateConfigurationTable();

}

}

//Set Level Variable from selection.

comboBoxMaxLevelText = comboBoxMaxLevel.Text;

}

if (File.Exists(Root + "\\Run.txt"))

{

String \_0 = File.ReadAllText(Root + "\\Run.txt");

if (\_0[0] == '1') \_1 = true; else \_1 = false;

if (\_0[1] == '1') \_2 = true; else \_2 = false;

if (\_0[2] == '1') \_3 = true; else \_3 = false;

if (\_0[3] == '1') \_4 = true; else \_4 = false;

File.Delete(Root + "Run.txt");

continueAGameToolStripMenuItem.Visible = false; if (\_1) { computerWithComputerToolStripMenuItem\_Click(sender, e); } else if (\_2) { computerWithComputerToolStripMenuItem1\_Click(sender, e); } else if (\_3) { toolStripMenuItem3\_Click(sender, e); } else if (\_4) { toolStripMenuItem6\_Click(sender, e); }

}

Loaded = true;

LoadedDLL = true;

if (Sec.radioButtonGrayOrder.Checked && OrderPlate == 1)

Person = true;

else

if (Sec.radioButtonBrownOrder.Checked && OrderPlate == -1)

Person = true;

tttt.Start();

ttt.Start();

AllOperate.Start();

if (!DrawDrawen)

{

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

RefrigtzDLL.AllDraw.DepthIterative = 0;

//MessageBox.Show("Draw Not Found.");

}

if (RefrigtzDLL.AllDraw.TableListAction.Count == 0)

RefrigtzDLL.AllDraw.TableListAction.Add(Table);

DrawImageOfMain();

bool A = GrayTimer.TextChanged;

GrayTimer.TextChanged = true;

SetPrictureBoxRefregitzInvalidate(pictureBoxTimerGray);

SetPrictureBoxRefregitzUpdate(pictureBoxTimerGray);

GrayTimer.TextChanged = A;

A = BrownTimer.TextChanged;

BrownTimer.TextChanged = true;

SetPrictureBoxRefregitzInvalidate(pictureBoxTimerBrown);

SetPrictureBoxRefregitzUpdate(pictureBoxTimerBrown);

BrownTimer.TextChanged = A;

var parallelOptions = new ParallelOptions();

parallelOptions.MaxDegreeOfParallelism = Int32.MaxValue;

}

//Reading Table Database.

int[,] ReadTable(int Movment, ref int MoveNumber)

{

RefrigtzDLL.ChessRules.CastleActBrown = false;

bool OneIncreament = false;

int[,] Tab = Table;

int Move = 0;

RefrigtzDLL.AllDraw.TableListAction.Clear();

do

{

try

{

String TableName = Move.ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

bookConn.Close();

bookConn.Open();

oleDbCmd.CommandText = "Select \* From " + TableName;

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

int ii = 0;

while (dr.Read())

{

Tab[0, ii] = System.Convert.ToInt32(dr["a"]);

Tab[1, ii] = System.Convert.ToInt32(dr["b"]);

Tab[2, ii] = System.Convert.ToInt32(dr["c"]);

Tab[3, ii] = System.Convert.ToInt32(dr["d"]);

Tab[4, ii] = System.Convert.ToInt32(dr["e"]);

Tab[5, ii] = System.Convert.ToInt32(dr["f"]);

Tab[6, ii] = System.Convert.ToInt32(dr["g"]);

Tab[7, ii] = System.Convert.ToInt32(dr["h"]);

ii++;

}

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

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

//MaxCurrentMovmentsNumber++;

RefrigtzDLL.AllDraw.TableListAction.Add(TableA);

if (Move == 42)

Move = 42;

if (RefrigtzDLL.AllDraw.TableListAction.Count > 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, TableA, MoveNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, TableA, MoveNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

//while (RefrigtzDLL.ChessRules.ObjectHittedRow != -1 && RefrigtzDLL.ChessRules.ObjectHittedColumn != -1)

{

// Thread.Sleep(100);

}

Move++;

if (Move > 2)

GameStarted = true;

if (Move > 1)

MoveNumber++;

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, TableA, OrderPlate, -1, -1).CheckMate(TableA, OrderPlate)))

{

int iii = 0;

do { iii++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", "Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

break;

}

}

catch (Exception t)

{

try

{

Move++;

String TableName = (Move).ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Select \* From " + TableName;

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

int ii = 0;

while (dr.Read())

{

Tab[0, ii] = System.Convert.ToInt32(dr["a"]);

Tab[1, ii] = System.Convert.ToInt32(dr["b"]);

Tab[2, ii] = System.Convert.ToInt32(dr["c"]);

Tab[3, ii] = System.Convert.ToInt32(dr["d"]);

Tab[4, ii] = System.Convert.ToInt32(dr["e"]);

Tab[5, ii] = System.Convert.ToInt32(dr["f"]);

Tab[6, ii] = System.Convert.ToInt32(dr["g"]);

Tab[7, ii] = System.Convert.ToInt32(dr["h"]);

ii++;

}

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

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

Draw.TableList.Clear();

Draw.TableList.Add(TableA);

Draw.SetRowColumn(0);

RefrigtzDLL.AllDraw.TableListAction.Add(TableA);

if (RefrigtzDLL.AllDraw.TableListAction.Count >= 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, TableA, MoveNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, TableA, MoveNumber, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, TableA, OrderPlate, -1, -1).CheckMate(TableA, OrderPlate)))

{

int iii = 0;

do { iii++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

break;

}

}

catch (Exception tt)

{

Log(tt);

break;

}

}

if (Move > 1)

{

OrderPlate \*= -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

}

} while (MoveNumber < MovmentsNumberMax);

//Draw.TableList.Clear();

//Draw.TableList.Add(Tab);

//Draw.SetRowColumn(0);

return Tab;

}

//Verify Accuarance of Table Games Methos.

bool VerifyTable(String FileName, int Movment, ref int MoveNumber)

{

bool PreviouseCheck = false;

int[,] Tab = Table;

int Move = 1;

int Order = 1;

bool TowCheckFromOneKind = false;

do

{

if (Move > 5000)

break;

try

{

if (Move % 2 == 1)

Order = 1;

else

Order = -1;

String connParam = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + FileName + ";;Persist Security Info=False; Jet OLEDB:Database Password='!HN#BGHHN&N$G$V4'";

String TableName = Move.ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Select \* From " + TableName;

OleDbDataReader dr = null;

try

{

dr = oleDbCmd.ExecuteReader();

}

catch (Exception t)

{

Log(t);

Move++;

break;

}

int ii = 0;

while (dr.Read())

{

Tab[0, ii] = System.Convert.ToInt32(dr["a"]);

Tab[1, ii] = System.Convert.ToInt32(dr["b"]);

Tab[2, ii] = System.Convert.ToInt32(dr["c"]);

Tab[3, ii] = System.Convert.ToInt32(dr["d"]);

Tab[4, ii] = System.Convert.ToInt32(dr["e"]);

Tab[5, ii] = System.Convert.ToInt32(dr["f"]);

Tab[6, ii] = System.Convert.ToInt32(dr["g"]);

Tab[7, ii] = System.Convert.ToInt32(dr["h"]);

ii++;

}

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

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

Draw.TableList.Clear();

Draw.TableList.Add(TableA);

Draw.SetRowColumn(0);

if (!Draw.IsEnemyThingsinStable(TableA, RefrigtzDLL.AllDraw.TableVeryfy, Order))

{

Tab = null;

Tab[0, 0] = -1;

}

else

{

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

{

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

{

RefrigtzDLL.AllDraw.TableVeryfy[i, j] = Tab[i, j];

}

}

}

Move++;

if (Move > 1)

MoveNumber++;

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, TableA, OrderPlate, -1, -1);

if (A.CheckMate(TableA, OrderPlate))

{

int iii = 0;

do { iii++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

break;

}

if (A.CheckGray && TowCheckFromOneKind)

{

Tab = null;

Tab[0, 0] = -1;

}

else

if (Order == 1 && A.CheckGray)

TowCheckFromOneKind = true;

else

TowCheckFromOneKind = false;

if (A.CheckGray && TowCheckFromOneKind)

{

Tab = null;

Tab[0, 0] = -1;

}

else

if (Order == 1 && A.CheckGray)

TowCheckFromOneKind = true;

else

TowCheckFromOneKind = false;

if (A.CheckBrown || A.CheckGray)

{

if (PreviouseCheck)

return false;

else

PreviouseCheck = true;

}

else

PreviouseCheck = false;

}

catch (Exception t)

{

Log(t);

do

{

try

{

Move++;

String connParam = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + FileName + ";;Persist Security Info=False; Jet OLEDB:Database Password='!HN#BGHHN&N$G$V4'";

String TableName = (Move).ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Drop Table " + TableName;

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

//return true;

}

catch (Exception tt)

{

Log(tt);

return false;

}

} while (true);

}

if (Move > 1)

{

OrderPlate \*= -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

}

} while (true);

return true;

}

//Read Specific Table Number.

int[,] ReadTableMovmentNumber()

{

int[,] Tab = Table;

int Move = MovmentsNumber;

RefrigtzDLL.AllDraw.TableListAction.Clear();

try

{

if (Move > 1)

{

if (MovmentsNumber % 2 == 0)

OrderPlate = 1;

else

OrderPlate = -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

}

String TableName = Move.ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Select \* From " + TableName;

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

int ii = 0;

while (dr.Read())

{

Tab[0, ii] = System.Convert.ToInt32(dr["a"]);

Tab[1, ii] = System.Convert.ToInt32(dr["b"]);

Tab[2, ii] = System.Convert.ToInt32(dr["c"]);

Tab[3, ii] = System.Convert.ToInt32(dr["d"]);

Tab[4, ii] = System.Convert.ToInt32(dr["e"]);

Tab[5, ii] = System.Convert.ToInt32(dr["f"]);

Tab[6, ii] = System.Convert.ToInt32(dr["g"]);

Tab[7, ii] = System.Convert.ToInt32(dr["h"]);

ii++;

}

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

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

Move++;

}

catch (Exception t)

{

Log(t);

try

{

Move++;

String TableName = (Move).ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Select \* From " + TableName;

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

int ii = 0;

while (dr.Read())

{

Tab[0, ii] = System.Convert.ToInt32(dr["a"]);

Tab[1, ii] = System.Convert.ToInt32(dr["b"]);

Tab[2, ii] = System.Convert.ToInt32(dr["c"]);

Tab[3, ii] = System.Convert.ToInt32(dr["d"]);

Tab[4, ii] = System.Convert.ToInt32(dr["e"]);

Tab[5, ii] = System.Convert.ToInt32(dr["f"]);

Tab[6, ii] = System.Convert.ToInt32(dr["g"]);

Tab[7, ii] = System.Convert.ToInt32(dr["h"]);

ii++;

}

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

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

{

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

{

TableA[i, j] = Tab[i, j];

}

}

}

catch (Exception tt)

{

Log(tt);

return null;

}

}

return Tab;

}

//Creation of New Tables at DatabaseMethod.

String CreatTable()

{

Begin12:

try

{

String TableName = MovmentsNumber.ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

bookConn.Close();

bookConn.Open();

oleDbCmd.CommandText = "Create Table " + TableName + " (a Number NOT NULL,b Number NOT NULL,c Number NOT NULL,d Number NOT NULL,e Number NOT NULL,f Number NOT NULL,g Number NOT NULL,h Number NOT NULL)";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

return TableName;

}

catch (Exception t)

{

Log(t);

String TableName = MovmentsNumber.ToString();

String Zero = "Table";

for (int i = 0; i < 8 - TableName.Length; i++)

Zero += "0";

TableName = Zero + TableName;

oleDbCmd.CommandText = "Drop Table " + TableName;

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

goto Begin12;

}

}

//Creatiopn of Configuration Table

void CreateConfigurationTable()

{

if (!AllDrawLoad)

{

Begin12:

try

{

oleDbCmd.CommandText = "Create Table Configuration (checkBoxAStarGreedyHuristic Number NOT NULL,checkBoxPredictHuristci Number NOT NULL,checkBoxAStarGreadyFirstSearch Number NOT NULL,checkBoxBestMovments Number NOT NULL,checkBoxOnlySelf Number NOT NULL,radioButtonOriginalImages Number NOT NULL,radioButtonBigFittingImages Number NOT NULL,radioButtonSmallFittingImages Number NOT NULL,checkBoxAStarGreedyMovement Number NOT NULL,checkBoxUseDoubleTime Number NOT NULL,checkBoxUsePenaltyRegradMechnisam Number NOT NULL,checkBoxDynamicProgrammingAStarGreedyt Number NOT NULL,comboBoxMaxTree Number NOT NULL,comboBoxAttack Number NOT NULL,comboBoxObjectDangour Number NOT NULL,comboBoxReducedAttacked Number NOT NULL,comboBoxSupport Number NOT NULL,comboBoxHitting Number NOT NULL,comboBoxMovments Number NOT NULL,ArrangmentsChanged Number NOT NULL,GrayTimer Number NOT NULL,BrownTimer Number NOT NULL,BobSection Number NOT NULL,AliceSection Number NOT NULL,StateCP Number NOT NULL,StateCC Number NOT NULL,StateGe Number NOT NULL,Blitz Number NOT NULL,Person Number NOT NULL,SettingPRFALSE Number NOT NULL,FullGame Number NOT NULL,Stockfish Number NOT NULL,lable1 Text NOT NULL,lable2 Text NOT NULL,MovmentsNumber Number NOT NULL)";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

bookConn.Close();

bookConn.Open();

oleDbCmd.CommandText = "Insert into Configuration (checkBoxAStarGreedyHuristic,checkBoxPredictHuristci,checkBoxAStarGreadyFirstSearch,checkBoxBestMovments,checkBoxOnlySelf,radioButtonOriginalImages,radioButtonBigFittingImages,radioButtonSmallFittingImages,checkBoxAStarGreedyMovement,checkBoxUseDoubleTime,checkBoxUsePenaltyRegradMechnisam,checkBoxDynamicProgrammingAStarGreedyt,comboBoxMaxTree,comboBoxAttack,comboBoxObjectDangour,comboBoxReducedAttacked,comboBoxSupport,comboBoxHitting,comboBoxMovments,ArrangmentsChanged,GrayTimer,BrownTimer,BobSection,AliceSection,StateCP,StateCC,StateGe,Blitz,Person,SettingPRFALSE,FullGame,Stockfish,lable1,lable2,MovmentsNumber) values(" + System.Convert.ToInt32(checkBoxAStarGreedyHuristic.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxPredictHuristci.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxAStarGreadyFirstSearch.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxBestMovments.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxOnlySelf.Checked).ToString() + "," + System.Convert.ToInt32(radioButtonOriginalImages.Checked).ToString() + "," + System.Convert.ToInt32(radioButtonBigFittingImages.Checked).ToString() + "," + System.Convert.ToInt32(radioButtonSmallFittingImages.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxBestMovments.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxUseDoubleTime.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxUsePenaltyRegradMechnisam.Checked).ToString() + "," + System.Convert.ToInt32(checkBoxDynamicProgrammingAStarGreedyt.Checked).ToString() + "," + System.Convert.ToInt32(comboBoxMaxLevel.Text).ToString() + "," + System.Convert.ToInt32(comboBoxAttack.Text).ToString() + "," + System.Convert.ToInt32(comboBoxObjectDangour.Text).ToString() + "," + System.Convert.ToInt32(comboBoxReducedAttacked.Text).ToString() + "," + System.Convert.ToInt32(comboBoxSupport.Text).ToString() + "," + System.Convert.ToInt32(comboBoxKiller.Text).ToString() + "," + System.Convert.ToInt32(comboBoxMovments.Text).ToString() + "," + System.Convert.ToInt32(ArrangmentsChanged).ToString() + "," + GrayTimer.Times.ToString() + "," + BrownTimer.Times.ToString() + "," + System.Convert.ToInt32(BobSection).ToString() + "," + System.Convert.ToInt32(AliceSection).ToString() + "," + System.Convert.ToInt32(StateCP).ToString() + "," + System.Convert.ToInt32(StateCC).ToString() + "," + System.Convert.ToInt32(StateGe).ToString() + "," + System.Convert.ToInt32(Blitz).ToString() + "," + System.Convert.ToInt32(Person).ToString() + "," + System.Convert.ToInt32(SettingPRFALSE).ToString() + "," + System.Convert.ToInt32(FullGame).ToString() + "," + System.Convert.ToInt32(Stockfish).ToString() + ",'" + label1.Text + "','" + label2.Text + "','" + MovmentsNumber.ToString() + "')";

temp = oleDbCmd.ExecuteNonQuery();

}

catch (Exception t)

{

Log(t);

bookConn.Close();

bookConn.Open();

oleDbCmd.CommandText = "Drop Table Configuration";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

goto Begin12;

}

}

}

void UpdateTable(int[,] Tabl, String TableName)

{

//if (UpdateConfigurationTableVal)

{

try

{

oleDbCmd.CommandText = "Drop Table " + TableName;

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

InsertTableAtDataBase(Tabl);

}

catch (Exception t)

{

Log(t);

}

}

}

//Reading of Configuration Table Method.

void ReadConfigurationTable()

{

if (!AllDrawLoad)

{

Begin12:

try

{

oleDbCmd.CommandText = "Select \* from Configuration";

OleDbDataReader dr = null;

dr = oleDbCmd.ExecuteReader();

bool ARead = false;

if (dr.Read())

{

checkBoxAStarGreedyHuristic.Checked = System.Convert.ToBoolean(dr["checkBoxAStarGreedyHuristic"]);

if (checkBoxAStarGreedyHuristic.Checked)

AStarGreedyHuristic = true;

else

AStarGreedyHuristic = false;

checkBoxPredictHuristci.Checked = System.Convert.ToBoolean(dr["checkBoxPredictHuristci"]);

if (checkBoxPredictHuristci.Checked)

PredictHuristic = true;

else

PredictHuristic = false;

checkBoxAStarGreadyFirstSearch.Checked = System.Convert.ToBoolean(dr["checkBoxAStarGreadyFirstSearch"]);

if (checkBoxAStarGreadyFirstSearch.Checked)

AStarGreedyHuristic = true;

else

AStarGreedyHuristic = false;

checkBoxBestMovments.Checked = System.Convert.ToBoolean(dr["checkBoxBestMovments"]);

if (checkBoxBestMovments.Checked)

BestMovments = true;

else

BestMovments = false;

checkBoxOnlySelf.Checked = System.Convert.ToBoolean(dr["checkBoxOnlySelf"]);

if (checkBoxOnlySelf.Checked)

OnlySelf = true;

else

OnlySelf = false;

radioButtonOriginalImages.Checked = System.Convert.ToBoolean(dr["radioButtonOriginalImages"]);

radioButtonBigFittingImages.Checked = System.Convert.ToBoolean(dr["radioButtonBigFittingImages"]);

radioButtonSmallFittingImages.Checked = System.Convert.ToBoolean(dr["radioButtonSmallFittingImages"]);

checkBoxAStarGreedyMovement.Checked = System.Convert.ToBoolean(dr["checkBoxAStarGreedyMovement"]);

checkBoxUseDoubleTime.Checked = System.Convert.ToBoolean(dr["checkBoxUseDoubleTime"]);

checkBoxUsePenaltyRegradMechnisam.Checked = System.Convert.ToBoolean(dr["checkBoxUsePenaltyRegradMechnisam"]);

if (checkBoxUsePenaltyRegradMechnisam.Checked)

UsePenaltyRegardMechnisam = true;

else

UsePenaltyRegardMechnisam = false;

checkBoxDynamicProgrammingAStarGreedyt.Checked = System.Convert.ToBoolean(dr["checkBoxDynamicProgrammingAStarGreedyt"]);

comboBoxMaxLevel.Text = System.Convert.ToString(dr["comboBoxMaxTree"]);

comboBoxAttack.Text = System.Convert.ToString(dr["comboBoxAttack"]);

comboBoxObjectDangour.Text = System.Convert.ToString(dr["comboBoxObjectDangour"]);

comboBoxReducedAttacked.Text = System.Convert.ToString(dr["comboBoxReducedAttacked"]);

comboBoxSupport.Text = System.Convert.ToString(dr["comboBoxSupport"]);

comboBoxKiller.Text = System.Convert.ToString(dr["comboBoxHitting"]);

comboBoxMovments.Text = System.Convert.ToString(dr["comboBoxMovments"]);

ArrangmentsChanged = System.Convert.ToBoolean(dr["ArrangmentsChanged"]);

GrayTimer.Times = System.Convert.ToInt64(dr["GrayTimer"]);

BrownTimer.Times = System.Convert.ToInt64(dr["BrownTimer"]);

BobSection = System.Convert.ToBoolean(dr["BobSection"]);

AliceSection = System.Convert.ToBoolean(dr["AliceSection"]);

StateCP = System.Convert.ToBoolean(dr["StateCP"]);

StateCC = System.Convert.ToBoolean(dr["StateCC"]);

StateGe = System.Convert.ToBoolean(dr["StateGe"]);

Blitz = System.Convert.ToBoolean(dr["Blitz"]);

Person = System.Convert.ToBoolean(dr["Person"]);

SettingPRFALSE = System.Convert.ToBoolean(dr["SettingPRFALSE"]);

FullGame = System.Convert.ToBoolean(dr["FullGame"]);

Stockfish = System.Convert.ToBoolean(dr["Stockfish"]);

label1.Text = System.Convert.ToString(dr["lable1"]);

label2.Text = System.Convert.ToString(dr["lable2"]);

MovmentsNumberMax = System.Convert.ToInt32(dr["MovmentsNumber"]);

ARead = true;

}

if (!ARead)

CreateConfigurationTable();

}

catch (Exception t)

{

Log(t);

try

{

oleDbCmd.CommandText = "Drop Table Configuration";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

}

catch (Exception tt)

{

Log(tt);

CreateConfigurationTable();

}

goto Begin12;

}

}

}

//Updating of Configuration Method.

void UpdateConfigurationTable()

{

//if (UpdateConfigurationTableVal)

{

Begin12:

try

{

if (AStarGreedyHuristic)

checkBoxAStarGreedyHuristic.Checked = true;

else

checkBoxAStarGreedyHuristic.Checked = false;

if (PredictHuristic)

checkBoxPredictHuristci.Checked = true;

else

checkBoxPredictHuristci.Checked = false;

if (AStarGreedyHuristic)

checkBoxAStarGreadyFirstSearch.Checked = true;

else

checkBoxAStarGreadyFirstSearch.Checked = false;

if (BestMovments)

checkBoxBestMovments.Checked = true;

else

checkBoxBestMovments.Checked = false;

if (OnlySelf)

checkBoxOnlySelf.Checked = true;

else

checkBoxOnlySelf.Checked = false;

if (UsePenaltyRegardMechnisam)

checkBoxUsePenaltyRegradMechnisam.Checked = true;

else

checkBoxUsePenaltyRegradMechnisam.Checked = false;

TimersSet = false;

oleDbCmd.CommandText = "Update Configuration Set checkBoxAStarGreedyHuristic=" + System.Convert.ToInt32(checkBoxAStarGreedyHuristic.Checked).ToString() + ",checkBoxPredictHuristci=" + System.Convert.ToInt32(checkBoxPredictHuristci.Checked).ToString() + ",checkBoxAStarGreadyFirstSearch=" + System.Convert.ToInt32(checkBoxAStarGreadyFirstSearch.Checked).ToString() + ",checkBoxBestMovments=" + System.Convert.ToInt32(checkBoxBestMovments.Checked).ToString() + ",checkBoxOnlySelf=" + System.Convert.ToInt32(checkBoxOnlySelf.Checked).ToString() + ",radioButtonOriginalImages=" + System.Convert.ToInt32(radioButtonOriginalImages.Checked).ToString() + ",radioButtonBigFittingImages=" + System.Convert.ToInt32(radioButtonBigFittingImages.Checked).ToString() + ",radioButtonSmallFittingImages=" + System.Convert.ToInt32(radioButtonSmallFittingImages.Checked).ToString() + ",checkBoxAStarGreedyMovement=" + System.Convert.ToInt32(checkBoxAStarGreedyMovement.Checked).ToString() + ",checkBoxUseDoubleTime=" + System.Convert.ToInt32(checkBoxUseDoubleTime.Checked).ToString() + ",checkBoxUsePenaltyRegradMechnisam=" + System.Convert.ToInt32(checkBoxUsePenaltyRegradMechnisam.Checked).ToString() + ",checkBoxDynamicProgrammingAStarGreedyt=" + System.Convert.ToInt32(checkBoxDynamicProgrammingAStarGreedyt.Checked).ToString() + ",comboBoxMaxTree=" + comboBoxMaxLevel.Text + ",comboBoxAttack=" + comboBoxAttack.Text + ",comboBoxObjectDangour=" + comboBoxObjectDangour.Text + ",comboBoxReducedAttacked=" + comboBoxReducedAttacked.Text + ",comboBoxSupport=" + comboBoxSupport.Text + ",comboBoxHitting=" + comboBoxKiller.Text + ",comboBoxMovments=" + comboBoxMovments.Text + ",ArrangmentsChanged=" + System.Convert.ToString(System.Convert.ToInt32(ArrangmentsChanged)) + ",GrayTimer=" + GrayTimer.Times.ToString() + ",BrownTimer=" + BrownTimer.Times.ToString() + ",BobSection = " + System.Convert.ToUInt32(BobSection).ToString() + ",AliceSection =" + System.Convert.ToUInt32(AliceSection).ToString() + ",StateCP = " + System.Convert.ToUInt32(StateCP).ToString() + ",StateCC = " + System.Convert.ToUInt32(StateCC).ToString() + ",StateGe=" + System.Convert.ToUInt32(StateGe).ToString() + ",Blitz=" + System.Convert.ToUInt32(Blitz).ToString() + ",Person=" + System.Convert.ToUInt32(Person).ToString() + ",SettingPRFALSE=" + System.Convert.ToUInt32(SettingPRFALSE).ToString() + ",FullGame=" + System.Convert.ToUInt32(FullGame).ToString() + ",Stockfish=" + System.Convert.ToUInt32(Stockfish).ToString() + ",lable1='" + label1.Text + "',lable2='" + label2.Text + "',MovmentsNumber='" + MovmentsNumber.ToString() + "'";

bookConn.Close();

bookConn.Open();

int temp = oleDbCmd.ExecuteNonQuery();

TimersSet = true;

}

catch (Exception t)

{

//TimersSet = true;

Log(t);

try

{

oleDbCmd.CommandText = "Drop Table Configuration";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

goto Begin12;

}

catch (Exception tt)

{

Log(tt);

}

}

}

}

//Inserting of New Tabler at Database.

void InsertTableAtDataBase(int[,] Table)

{

//TimersSet = false;

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

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

{

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

{

Tab[i, j] = Table[i, j];

}

}

String TableName = CreatTable();

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

{

oleDbCmd.CommandText = "insert into " + TableName + "(a,b,c,d,e,f,g,h) values (" + Tab[0, i].ToString() + "," + Tab[1, i].ToString() + "," + Tab[2, i].ToString() + "," + Tab[3, i].ToString() + "," + Tab[4, i].ToString() + "," + Tab[5, i].ToString() + "," + Tab[6, i].ToString() + "," + Tab[7, i].ToString() + ")";

int temp = 0;

temp = oleDbCmd.ExecuteNonQuery();

}

//MaxCurrentMovmentsNumber++;

if (MovmentsNumber > 1)

GameStarted = true;

}

void DrawObjects()

{

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

{

Object O1 = new Object();

lock (O1)

{

if (Draw.SolderesOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.SolderesOnTable[i].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

}

//}, () =>

//{

//for(int i = 0;i< Draw.ElefantHigh;i++)

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

{

Object O1 = new Object();

lock (O1)

{

if (Draw.ElephantOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.ElephantOnTable[i].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

}

//}, () =>

//{

//for(int i = 0;i< Draw.HourseHight;i++)

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

{

Object O1 = new Object();

lock (O1)

{

if (Draw.HoursesOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.HoursesOnTable[i].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

}

//}, () =>

//{

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

//for(int i = 0;i< Draw.CastleHigh;i++)

{

Object O1 = new Object();

lock (O1)

{

if (Draw.CastlesOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.CastlesOnTable[i].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

//}, () =>

//{

}

//for(int i = 0;i< Draw.MinisterHigh;i++)

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

{

Object O1 = new Object();

lock (O1)

{

if (Draw.MinisterOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.MinisterOnTable[i].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

//}, () =>

}

//{

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

//for(int i = 0;i< Draw.KingHigh;i++)

{

Object O1 = new Object();

lock (O1)

{

if (Draw.KingOnTable[i] != null)

{

try

{

Object O = new Object();

lock (O)

{

Draw.KingOnTable[i].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

}

catch (Exception t)

{

Log(t);

}

}

}

//}

//}

}

}

//Painting of Form Refregitz PictureBox and Tow Refrigtz.Timer Pictue Box on Time.

private void pictureBoxRefrigtz\_Paint(object sender, PaintEventArgs e)

{

Object o = new Object();

lock (o)

{

if (RefrigtzDLL.AllDraw.DrawTable)

{

//System.Threading.Thread.Sleep(100);

if (GameStarted && MovmentsNumber == 0)

buttonChangeArrangment.Visible = false;

//System.Threading.Thread.Sleep(8);

//if (RefrigtzDLL.AllDraw.TableListAction.Count > 3)

//toolStripMenuItemRandomGeneticGames.Enabled = true;

//else

// toolStripMenuItemRandomGeneticGames.Enabled = false;

if (GrayTimer.EndTime)

BrownWiner = true;

if (BrownTimer.EndTime)

GrayWinner = true;

if (GrayWinner || BrownWiner)

{

if (t1 != null)

t1.Abort();

if (t2 != null)

t2.Abort();

if (t3 != null)

t3.Abort();

if (GrayWinner)

SetBoxText("\r\nGray Winees!");

if (BrownWiner)

SetBoxText("\r\nBrown Winees!");

return;

}

try

{

RefrigtzDLL.AllDraw.DrawTable = false;

//ReconstructTable();

pictureBoxRefrigtz.Image = (Image)new Bitmap(pictureBoxRefrigtz.Width, pictureBoxRefrigtz.Height);

ChessTable = (Image)new Bitmap(pictureBoxRefrigtz.Image.Width, pictureBoxRefrigtz.Image.Height);

g = Graphics.FromImage(ChessTable);

g.FillRectangle(new SolidBrush(Color.Yellow), new Rectangle(0, 0, pictureBoxRefrigtz.Width, pictureBoxRefrigtz.Height));

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

for (int i = 0; i < pictureBoxRefrigtz.Image.Width; i += pictureBoxRefrigtz.Image.Width / 8)

for (int j = 0; j < pictureBoxRefrigtz.Image.Height; j += pictureBoxRefrigtz.Image.Height / 8)

{

try

{

if ((i + j) % 2 == 0)

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\Black.jpg"), new Rectangle(i, j, this.pictureBoxRefrigtz.Width / 8, this.pictureBoxRefrigtz.Height / 8));

else

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\White.jpg"), new Rectangle(i, j, this.pictureBoxRefrigtz.Width / 8, this.pictureBoxRefrigtz.Height / 8));

}

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

}

if (RowClickP != -1 && ColumnClickP != -1)

{

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

bool[,] Tab = null;

Tab = VeryFye(Table, OrderPlate, a);

for (int i = 0; i < pictureBoxRefrigtz.Image.Width; i += pictureBoxRefrigtz.Image.Width / 8)

for (int j = 0; j < pictureBoxRefrigtz.Image.Height; j += pictureBoxRefrigtz.Image.Height / 8)

{

try

{

if (Tab != null)

{

if (Tab[i / (pictureBoxRefrigtz.Image.Width / 8), j / (pictureBoxRefrigtz.Image.Height / 8)])

{

g.DrawString("\*", new Font("Times New Roman", 50), new SolidBrush(Color.Red), new Rectangle(i, j, this.pictureBoxRefrigtz.Width / 8, this.pictureBoxRefrigtz.Height / 8));

}

}

}

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

}

}

//System.Threading.Tasks.Parallel.Invoke(() =>

//{

//for(int i = 0;i< Draw.SodierHigh;i++)

DrawObjects();

pictureBoxRefrigtz.Image = ChessTable;

g.Dispose();

}

catch (Exception t)

{

Log(t);

//RunInFront();

}

}

LoadedTable = true;

//UpadatTimer();

/\*while (PaintedPaused && DrawNumber > 50)

{

System.Threading.Thread.Sleep(10);

}

DrawNumber++;

if (DrawNumber >= 50)

DrawNumber = 0;

\*/

AllDrawLoad = true;

//pictureBoxTimerGray.Update();

//pictureBoxTimerGray.Invalidate();

//return;

}

System.Threading.Thread.Sleep(20);

//pictureBoxRefrigtz.Invalidate();

//pictureBoxRefrigtz.Update();

}

void SetRefregitzBicture()

{

}

double CalculateMoveMentHueuristicUser(int[,] Table, int Order, int Row, int Column, int RowS, int ColumnS, Color color)

{

RefrigtzDLL.ThinkingChess th = new RefrigtzDLL.ThinkingChess(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, Row, Column);

double HuristicAttackValue = new double();

double HuristicMovementValue = new double();

double HuristicSelfSupportedValue = new double();

double HuristicObjectDangourCheckMateValue = new double();

double HuristicHittingValue = new double();

double HuristicReducedAttackValue = new double();

double HeuristicDistabceOfCurrentMoveFromEnemyKingValue = new double();

double HeuristicKingSafe = new double();

double HeuristicFromCenter = new double();

double HeuristicKingDangour = new double();

th.CalculateHuristics(true, 0, Table, Row, Column, RowS, ColumnS, color, ref HuristicAttackValue, ref HuristicMovementValue, ref HuristicSelfSupportedValue, ref HuristicObjectDangourCheckMateValue, ref HuristicHittingValue, ref HuristicReducedAttackValue, ref HeuristicDistabceOfCurrentMoveFromEnemyKingValue, ref HeuristicKingSafe, ref HeuristicFromCenter, ref HeuristicKingDangour);

double[] Hu = new double[4];

return HuristicAttackValue + HuristicMovementValue +

HuristicSelfSupportedValue +

HuristicObjectDangourCheckMateValue + HuristicReducedAttackValue + HuristicHittingValue + HeuristicDistabceOfCurrentMoveFromEnemyKingValue + HeuristicKingSafe + HeuristicFromCenter + HeuristicKingDangour;

}

bool CheckMovment(int[,] Table, int i, int j, int ii, int jj, int Order)

{

bool Check = false;

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

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

{

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

{

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

}

}

Tab[ii, jj] = Tab[i, j];

Tab[i, j] = 0;

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, Order);

A.Check(Tab, Order);

if (Order == 1 && A.CheckGray)

Check = true;

if (Order == -1 && A.CheckBrown)

Check = true;

return Check;

}

void MovmentneDraw(int Kind)

{

DrawImageOfMain();

g = Graphics.FromImage(ChessTable);

if (Kind == 7)

{

Draw.KingOnTable[King].Row = RowRealesed;

Draw.KingOnTable[King].Column = ColumnRealeased;

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else

if (Kind == 1)

{

Draw.SolderesOnTable[Soldier].Row = (int)RowRealesed;

Draw.SolderesOnTable[Soldier].Column = (int)ColumnRealeased;

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else

if (Kind == 2)

{

Draw.ElephantOnTable[Elefant].Row = RowRealesed;

Draw.ElephantOnTable[Elefant].Column = ColumnRealeased;

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == 3)

{

Draw.HoursesOnTable[Hourse].Row = RowRealesed;

Draw.HoursesOnTable[Hourse].Column = ColumnRealeased;

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == 4)

{

Draw.CastlesOnTable[Castle].Row = RowRealesed;

Draw.CastlesOnTable[Castle].Column = ColumnRealeased;

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == 5)

{

Draw.MinisterOnTable[Minister].Row = RowRealesed;

Draw.MinisterOnTable[Minister].Column = ColumnRealeased;

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == 6)

{

Draw.KingOnTable[King].Row = RowRealesed;

Draw.KingOnTable[King].Column = ColumnRealeased;

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -7)

{

Draw.KingOnTable[King].Row = RowRealesed;

Draw.KingOnTable[King].Column = ColumnRealeased;

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -1)

{

Draw.SolderesOnTable[Soldier].Row = (int)RowRealesed;

Draw.SolderesOnTable[Soldier].Column = (int)ColumnRealeased;

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -2)

{

Draw.ElephantOnTable[Elefant].Row = RowRealesed;

Draw.ElephantOnTable[Elefant].Column = ColumnRealeased;

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -3)

{

Draw.HoursesOnTable[Hourse].Row = RowRealesed;

Draw.HoursesOnTable[Hourse].Column = ColumnRealeased;

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -4)

{

Draw.CastlesOnTable[Castle].Row = RowRealesed;

Draw.CastlesOnTable[Castle].Column = ColumnRealeased;

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -5)

{

Draw.MinisterOnTable[Minister].Row = RowRealesed;

Draw.MinisterOnTable[Minister].Column = ColumnRealeased;

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

else if (Kind == -6)

{

Draw.KingOnTable[King].Row = RowRealesed;

Draw.KingOnTable[King].Column = ColumnRealeased;

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

}

//pictureBoxRefrigtz.Image = ChessTable;

g.Dispose();

}

void MovmentsCastleKing(int Kind)

{

if (Kind == 7)

{

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnRealeased;

CurrentKind = 7;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 6;

}

else if (Kind == -7)

{

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnRealeased;

CurrentKind = -7;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, King);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -6;

}

}

void SetSyntax()

{

if (RefrigtzDLL.AllDraw.TableListAction.Count > 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

}

void Movements()

{

Object O = new Object();

lock (O)

{

RefrigtzDLL.ChessRules AA = null;

do

{

if (RefrigtzDLL.AllDraw.MouseClick == 1 //|| RefrigtzDLL.AllDraw.MouseClick == 2

)

{

//SetRefregitzBicture();

//System.Threading.Thread.Sleep(50);

}

//else

System.Threading.Thread.Sleep(10);

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

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

{

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

{

TabStor[i, j] = Table[i, j];

}

}

//For Iterative Movewmnt

if (SetMovement((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased))

continue;

if (RowClickP == -1 && ColumnClickP == -1)

continue;

try

{

if (((!StateCC && StateCP) || Blitz) && Person)

{

if (Sec.radioButtonGrayOrder.Checked)

{

if (ColumnClickP == ColumnRealeased && System.Math.Abs(RowClickP - RowRealesed) >= 2 && Table[(int)RowClickP, (int)ColumnClickP] == 6 && (RefrigtzDLL.ChessRules.CastleKingAllowedGray))

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(7);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 7, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 7))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

MovmentsCastleKing(7);

int Castles = -1;

if (!ArrangmentsChanged)

{

if (RowClickP < RowRealesed)

{

Table[(int)RowRealesed - 1, (int)ColumnRealeased] = 4;

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP + 3 && Draw.CastlesOnTable[i].Column == ColumnClick && ColumnClick == 0)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed + 1, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP + 3;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

else

{

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 4;

Table[(int)RowRealesed - 2, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP - 4 && Draw.CastlesOnTable[i].Column == ColumnClickP & ColumnClick == 0)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed - 1, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP - 4;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

}

else

{

if (RowClickP < RowRealesed)

{

Table[(int)RowRealesed - 1, (int)ColumnRealeased] = 4;

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP + 3 && Draw.CastlesOnTable[i].Column == ColumnClick && ColumnClick == 7)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed + 1, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP + 3;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

else

{

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 4;

Table[(int)RowRealesed - 2, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP - 4 && Draw.CastlesOnTable[i].Column == ColumnClickP && ColumnClick == 7)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed - 1, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP - 4;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

}

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

// return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

// return;

}

}

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Draw.CastlesOnTable[Castles].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

GrayTimer.StopTime();

BrownTimer.StartTime();

Person = false;

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

Clicked = false;

RefrigtzDLL.ChessRules.CastleActGray = true;

//TakeRoot.CalculateRootGray(Draw);

}

else

{

if (!ArrangmentsChanged)

{

if (ColumnClickP == 0)

Table[(int)RowClickP, (int)ColumnClickP] = 6;

else

if (ColumnClickP == 7)

Table[(int)RowClickP, (int)ColumnClickP] = -6;

}

else

{

if (ColumnClickP == 7)

Table[(int)RowClickP, (int)ColumnClickP] = 6;

else

if (ColumnClickP == 0)

Table[(int)RowClickP, (int)ColumnClickP] = -6;

}

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, King);

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

////SetRefregitzBicture();

}

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == 1)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(1);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 1))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

RefrigtzDLL.ThingsConverter.ActOfClickEqualTow = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnRealeased;

CurrentKind = 1;

ConvertWait = true;

Draw.SolderesOnTable[Soldier].ConvertOperation((int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate, false, Soldier);

Thread.Sleep(100);

while (ConvertWait) ;

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

if (Draw.SolderesOnTable[Soldier].Convert)

{

Table[(int)RowClickP, (int)ColumnClickP] = 0;

/\* if (Draw.SolderesOnTable[Soldier].ConvertedToMinister)

Table[(int)RowRealesed, (int)ColumnRealeased] = 5;

else if (Draw.SolderesOnTable[Soldier].ConvertedToCastle)

Table[(int)RowRealesed, (int)ColumnRealeased] = 4;

else if (Draw.SolderesOnTable[Soldier].ConvertedToHourse)

Table[(int)RowRealesed, (int)ColumnRealeased] = 3;

else if (Draw.SolderesOnTable[Soldier].ConvertedToElefant)

Table[(int)RowRealesed, (int)ColumnRealeased] = 2;

\*/

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

RefrigtzDLL.ThingsConverter.ClickOcurred = false;

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

else

{

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

Draw.SolderesOnTable[Soldier] = new RefrigtzDLL.DrawSoldier(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, Soldier);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 1;

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

}

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

//return;

}

}

Person = false;

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

{

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

{

Table[i, j] = TableCon[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, Soldier, 1);

Clicked = false;

return;

}

else

{

Draw.SolderesOnTable[Soldier] = new RefrigtzDLL.DrawSoldier(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, Soldier);

Table[(int)RowClickP, (int)ColumnClickP] = 1;

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == 2)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(2);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 2, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 2))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = 2;

Draw.ElephantOnTable[Elefant] = new RefrigtzDLL.DrawElefant(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, Elefant);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 2;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

// return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

// return;

}

}

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Person = false;

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, Elefant, 2);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = 2;

Draw.ElephantOnTable[Elefant] = new RefrigtzDLL.DrawElefant(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, Elefant);

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == 3)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(3);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 3, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 3))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = 3;

Draw.HoursesOnTable[Hourse] = new RefrigtzDLL.DrawHourse(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, Hourse);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 3;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

//return;

}

}

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, Hourse, 3);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = 3;

Draw.HoursesOnTable[Hourse] = new RefrigtzDLL.DrawHourse(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, Hourse);

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else if (Table[(int)RowClickP, (int)ColumnClickP] == 4)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(4);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 4, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 4))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = 4;

Draw.CastlesOnTable[Castle] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, Castle);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 4;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

//return;

}

}

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, Castle, 4);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = 4;

Draw.CastlesOnTable[Castle] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, Castle);

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == 5)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(5);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 5, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 5))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = 5;

Draw.MinisterOnTable[Minister] = new RefrigtzDLL.DrawMinister(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, Minister);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 5;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

//return;

}

}

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, Minister, 5);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = 5;

Draw.MinisterOnTable[Minister] = new RefrigtzDLL.DrawMinister(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, Minister);

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == 6)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(6);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 6, Table, 1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, 6))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, 1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, 1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Gray);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = 6;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = 6;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

//return;

}

}

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, 1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Gray);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, Table, OrderPlate \* -1, King, 6);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = 6;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, King);

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

}

if (Sec.radioButtonBrownOrder.Checked)

{

if (ColumnClickP == ColumnRealeased && System.Math.Abs(RowClickP - RowRealesed) >= 2 && Table[(int)RowClickP, (int)ColumnClickP] == -6 && (RefrigtzDLL.ChessRules.CastleKingAllowedBrown))

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-7);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -7, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -7))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

MovmentsCastleKing(-7);

int Castles = -1;

if (!ArrangmentsChanged)

{

if (RowClickP < RowRealesed)

{

Table[(int)RowRealesed - 1, (int)ColumnRealeased] = -4;

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP + 3 && Draw.CastlesOnTable[i].Column == ColumnClick && ColumnClick == 7)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed + 1, ColumnRealeased, Color.Brown, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP + 3;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

else

{

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = -4;

Table[(int)RowRealesed - 2, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP - 4 && Draw.CastlesOnTable[i].Column == ColumnClickP & ColumnClick == 0)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed - 1, ColumnRealeased, Color.Brown, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP - 4;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

}

else

{

if (RowClickP < RowRealesed)

{

Table[(int)RowRealesed - 1, (int)ColumnRealeased] = -4;

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP + 3 && Draw.CastlesOnTable[i].Column == ColumnClick && ColumnClick == 7)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed + 1, ColumnRealeased, Color.Brown, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP + 3;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

else

{

Table[(int)RowRealesed + 1, (int)ColumnRealeased] = -4;

Table[(int)RowRealesed - 2, (int)ColumnRealeased] = 0;

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

{

if (Draw.CastlesOnTable[i].Row == RowClickP - 4 && Draw.CastlesOnTable[i].Column == ColumnClickP && ColumnClick == 7)

{

Draw.CastlesOnTable[i] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed - 1, ColumnRealeased, Color.Brown, Table, OrderPlate, false, King);

Castles = i;

return;

}

}

if (Castles != -1)

{

Draw.CastlesOnTable[Castles].Row = RowClickP - 4;

Draw.CastlesOnTable[Castles].Column = ColumnClickP;

}

}

}

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

//return;

}

}

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Draw.CastlesOnTable[Castles].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

GrayTimer.StopTime();

BrownTimer.StartTime();

Person = false;

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

RefrigtzDLL.AllDraw.DrawTable = false;

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

InsertTableAtDataBase(Table);

Clicked = false;

//RefrigtzDLL.ChessRules.CastleActBrown = true;

//TakeRoot.CalculateRootGray(Draw);

return;

}

else

{

if (!ArrangmentsChanged)

{

if (ColumnClickP == 0)

Table[(int)RowClickP, (int)ColumnClickP] = 6;

else

if (ColumnClickP == 7)

Table[(int)RowClickP, (int)ColumnClickP] = -6;

}

else

{

if (ColumnClickP == 7)

Table[(int)RowClickP, (int)ColumnClickP] = 6;

else

if (ColumnClickP == 0)

Table[(int)RowClickP, (int)ColumnClickP] = -6;

}

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Gray, Table, OrderPlate, false, King);

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

//SetRefregitzBicture();

}

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -1)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-1);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -1, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -1))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

RefrigtzDLL.ThingsConverter.ActOfClickEqualTow = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -1;

ConvertWait = true;

Draw.SolderesOnTable[Soldier].ConvertOperation((int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate, false, Soldier);

while (ConvertWait) ;

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

if (Draw.SolderesOnTable[Soldier].Convert)

{

Table[(int)RowClickP, (int)ColumnClickP] = 0;

/\* if (Draw.SolderesOnTable[Soldier].ConvertedToMinister)

Table[(int)RowRealesed, (int)ColumnRealeased] = -5;

else if (Draw.SolderesOnTable[Soldier].ConvertedToCastle)

Table[(int)RowRealesed, (int)ColumnRealeased] = -4;

else if (Draw.SolderesOnTable[Soldier].ConvertedToHourse)

Table[(int)RowRealesed, (int)ColumnRealeased] = -3;

else if (Draw.SolderesOnTable[Soldier].ConvertedToElefant)

Table[(int)RowRealesed, (int)ColumnRealeased] = -2;

\*/

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

RefrigtzDLL.ThingsConverter.ClickOcurred = false;

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

else

{

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

Draw.SolderesOnTable[Soldier] = new RefrigtzDLL.DrawSoldier(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, Soldier);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -1;

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

}

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

/// Table = TabStor;

// return;

}

}

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

{

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

{

Table[i, j] = TableCon[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, Soldier, -1);

Clicked = false;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -1;

Draw.SolderesOnTable[Soldier] = new RefrigtzDLL.DrawSoldier(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, Soldier);

Draw.SolderesOnTable[Soldier].DrawSoldierOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -2)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-2);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -2, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -2))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -2;

Draw.ElephantOnTable[Elefant] = new RefrigtzDLL.DrawElefant(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, Elefant);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -2;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

//return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

//return;

}

}

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, Elefant, -2);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -2;

Draw.ElephantOnTable[Elefant] = new RefrigtzDLL.DrawElefant(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, Elefant);

Draw.ElephantOnTable[Elefant].DrawElefantOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -3)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-3);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -3, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -3))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -3;

Draw.HoursesOnTable[Hourse] = new RefrigtzDLL.DrawHourse(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, Hourse);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -3;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

///return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

// return;

}

}

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, Hourse, -3);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -3;

Draw.HoursesOnTable[Hourse] = new RefrigtzDLL.DrawHourse(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, Hourse);

Draw.HoursesOnTable[Hourse].DrawHourseOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -4)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-4);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -4, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -4))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -4;

Draw.CastlesOnTable[Castle] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, Castle);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -4;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

// return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

//return;

}

}

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, Castle, -4);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -4;

Draw.CastlesOnTable[Castle] = new RefrigtzDLL.DrawCastle(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, Castle);

Draw.CastlesOnTable[Castle].DrawCastleOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -5)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-5);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -5, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -5))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -5;

Draw.MinisterOnTable[Minister] = new RefrigtzDLL.DrawMinister(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Brown, Table, OrderPlate, false, Minister);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -5;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

//Table = TabStor;

// return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

// Table = TabStor;

// return;

}

}

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

//TakeRoot.CalculateRootGray(Draw);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, Minister, -5);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -5;

Draw.MinisterOnTable[Minister] = new RefrigtzDLL.DrawMinister(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, Minister);

Draw.MinisterOnTable[Minister].DrawMinisterOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

}

//SetRefregitzBicture();

}

else

if (Table[(int)RowClickP, (int)ColumnClickP] == -6)

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

MovmentneDraw(-6);

}

else

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

if ((new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -6, Table, -1, (int)RowClickP, (int)ColumnClickP)).Rules((int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown, -6))

{

if (CheckMovment(Table, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, -1))

continue;

RefrigtzDLL.AllDraw.Less = CalculateMoveMentHueuristicUser(Table, -1, (int)RowClickP, (int)ColumnClickP, (int)RowRealesed, (int)ColumnRealeased, Color.Brown);

int Hit = Table[(int)RowRealesed, (int)ColumnRealeased];

bool HitVal = false;

if (Hit != 0)

HitVal = true;

LastRow = (int)RowRealesed;

LastColumn = (int)ColumnClickP;

CurrentKind = -6;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowRealesed, ColumnRealeased, Color.Gray, Table, OrderPlate, false, King);

Table[(int)RowClickP, (int)ColumnClickP] = 0;

Table[(int)RowRealesed, (int)ColumnRealeased] = -6;

AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

if (AA.Check(Table, OrderPlate))

{

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

{

// Table = TabStor;

// return;

}

else

if (AA.CheckBrown && OrderPlate == -1)

{

//Table = TabStor;

// return;

}

}

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

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

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

{

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

{

TableCon[i, j] = Table[i, j];

}

}

RefrigtzDLL.AllDraw.TableListAction.Add(TableCon);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

SetSyntax();

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, King, -6);

Clicked = false;

return;

}

else

{

Table[(int)RowClickP, (int)ColumnClickP] = -6;

Draw.KingOnTable[King] = new RefrigtzDLL.DrawKing(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, RowClickP, ColumnClickP, Color.Brown, Table, OrderPlate, false, King);

Draw.KingOnTable[King].DrawKingOnTable(ref g, pictureBoxRefrigtz.Image.Width / 8, pictureBoxRefrigtz.Image.Height / 8);

Clicked = false;

}

//SetRefregitzBicture();

}

}

}

}

}

catch (Exception T)

{

Log(T);

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

//Blitz = true;

//Person = true;

//Clicked = true;

//StateCC = false;

//StateCP = true;

continue;

}

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

}

while (true);

}

}

void Wait()

{

do

{

Thread.Sleep(100);

} while (Clicked);

}

String Alphabet()

{

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()

{

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;

}

String ListToString(List<char> A)

{

String B = "";

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

B += A[i];

return B;

}

bool empty(int Row, int Column)

{

bool S = false;

if (Table[Row, Column] == 0)

S = true;

else

S = false;

return S;

}

int piece\_on(int Row, int Column)

{

return 6 + Table[Row, Column];

}

String Fen()

{

bool StartPos = false;

if (RowRealesed == -1 || ColumnRealeased == -1)

StartPos = true;

int emptyCnt;

String ss = "";

for (int r = 0; r <= 7; ++r)

{

for (int f = 0; f <= 7; ++f)

{

for (emptyCnt = 0; f <= 7 && empty(f, r); ++f)

++emptyCnt;

if (emptyCnt != 0)

ss += emptyCnt;

if (f <= 7)

ss += PieceToChar[piece\_on(f, r)];

}

if (r != 7)

ss += '/';

}

if (!BobSection)

ss += " w ";

else

ss += " b ";

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

ss += "K";

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

ss += "Q";

if (RefrigtzDLL.ChessRules.SmallKingCastleBrown)

ss += "k";

if (RefrigtzDLL.ChessRules.BigKingCastleBrown)

ss += "q";

if (!RefrigtzDLL.ChessRules.CastleKingAllowedGray && !RefrigtzDLL.ChessRules.CastleKingAllowedBrown)

ss += '-';

String S = " - ";

if (!StartPos)

{

if (!BobSection)

{

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 1)

{

S = " ";

S += Alphabet() + ((int)(7 - ColumnRealeased)).ToString();

S += " ";

}

}

else

{

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 1)

{

S = " ";

S += Alphabet() + ((int)(7 - ColumnRealeased)).ToString();

S += " ";

}

}

}

else

{

S = " ";

S += "-";

S += " ";

}

StockMoveBase = MovmentsNumber / 2;

StockMove = MovmentsNumber % 2;

S += (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

ss += S;

//if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

// StockMoveBase++;

//else

// StockMove++;

ss = "position fen " + ss;

return ss;

/\*

int Start = -1;

//When Refregitz Moves

if (!BobSection)

{

if (FenCastling == -1)

{

//When Index first is Solders.

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 1)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + (int)RowClickP] == 'P')

fenS[Start + (int)RowClickP] = '1';

else

fenS[Start + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "P" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - (Dum1 + Dum2 + Dum3 + Dum4) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4, S);

fenS = new List<char>(fs);

S = "w KQkq " + Alphabet() + Number() + " " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 2)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + 9 + (int)RowClickP] == 'N')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "N" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6) - (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "w KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 3)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + 9 + (int)RowClickP] == 'B')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "B" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6) - (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "w KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 4)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + 9 + (int)RowClickP] == 'R')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "R" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6) - (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "w KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 5)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + 9 + (int)RowClickP] == 'Q')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "Q" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6) - (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "w KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 6)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6;

if (fenS[Start + 9 + (int)RowClickP] == 'K')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

String S = ((int)RowRealesed).ToString() + "K" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, (Dum1 + Dum2 + Dum3 + Dum4 + Dum5 + Dum6) - (Dum1 + Dum2 + Dum3 + Dum4 + Dum5) - 1);

fs = fs.Insert(Dum1 + Dum2 + Dum3 + Dum4 + Dum5, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "w KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

}

else

if (FenCastling == 0)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

String S = "w K-kq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (FenCastling == 1)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

String S = "w -Qkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("b "), fs.Length - fs.IndexOf("b ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

}

else

{

if (FenCastling == -1)

{

//When Index first is Solders.

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 1)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1;

if (fenS[Start + (int)RowClickP] == 'p')

fenS[Start + (int)RowClickP] = '1';

else

fenS[Start + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "P" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1 + Dum2, (Dum1 + Dum2 + Dum3) - (Dum1 + Dum2) - 1);

fs = fs.Insert(Dum1 + Dum2, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

S = "b KQkq " + Alphabet() + Number() + " " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 2)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2;

if (fenS[Start + 9 + (int)RowClickP] == 'n')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "N" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1, (Dum1 + Dum2) - (Dum1) - 1);

fs = fs.Insert(Dum1, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "b KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 3)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2;

if (fenS[Start + 9 + (int)RowClickP] == 'b')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "B" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1, (Dum1 + Dum2) - (Dum1) - 1);

fs = fs.Insert(Dum1, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "b KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 4)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2;

if (fenS[Start + 9 + (int)RowClickP] == 'r')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "R" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1, (Dum1 + Dum2) - (Dum1) - 1);

fs = fs.Insert(Dum1, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "b KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 5)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2;

if (fenS[Start + 9 + (int)RowClickP] == 'q')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "Q" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1, (Dum1 + Dum2) - (Dum1) - 1);

fs = fs.Insert(Dum1, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "b KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (System.Math.Abs(Table[(int)RowRealesed, (int)ColumnRealeased]) == 6)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

Start = Dum1 + Dum2;

if (fenS[Start + 9 + (int)RowClickP] == 'k')

fenS[Start + 9 + (int)RowClickP] = '1';

else

fenS[Start + 9 + (int)RowClickP] = System.Convert.ToChar((System.Convert.ToInt32(fenS[Start + 9 + (int)RowClickP]) + 1));

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

String S = ((int)RowRealesed).ToString() + "K" + ((int)(7 - ColumnRealeased)).ToString();

fs = ListToString(fenS);

fs = fs.Remove(Dum1, (Dum1 + Dum2) - (Dum1) - 1);

fs = fs.Insert(Dum1, S);

fenS = new List<char>(fs);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

S = "b KQkq - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ") - 1), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

}

else

if (FenCastling == 0)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

String S = "b KQk- - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

else

if (FenCastling == 1)

{

String fs = ListToString(fenS);

int Dum1 = 0, Dum2 = 0, Dum3 = 0, Dum4 = 0, Dum5 = 0, Dum6 = 0;

String S = "b KQ-q - " + (StockMoveBase).ToString() + " " + ((int)StockMove).ToString() + "\n";

fs = ListToString(fenS);

Dum1 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum1, fs.Length - Dum1);

Dum2 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum2, fs.Length - Dum2);

Dum3 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum3, fs.Length - Dum3);

Dum4 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum4, fs.Length - Dum4);

Dum5 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum5, fs.Length - Dum5);

Dum6 = fs.Substring(0, fs.IndexOf("/")).Length + 1;

fs = fs.Substring(Dum6, fs.Length - Dum6);

fs = ListToString(fenS); fenS = new List<char>(fs);

fs = fs.Replace(fs.Substring(fs.IndexOf("w "), fs.Length - fs.IndexOf("w ")), S);

fenS = new List<char>(fs);

if (MovmentsNumber % 2 == 0 && MovmentsNumber != 0)

StockMoveBase++;

else

StockMove++;

}

}

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

\*/

//return fenS.ToString();

}

private static StringBuilder sortOutput = null;

private static int numOutputLines = 0;

private static void SortOutputHandler(object sendingProcess,

DataReceivedEventArgs outLine)

{

// Collect the sort command output.

if (!String.IsNullOrEmpty(outLine.Data))

{

numOutputLines++;

// Add the text to the collected output.

sortOutput.Append(Environment.NewLine +

"[" + numOutputLines.ToString() + "] - " + outLine.Data);

}

}

int SetRowColumn(String A)

{

try

{

if (A[0] == 'a')

RowClickP = 0;

else

if (A[0] == 'b')

RowClickP = 1;

else

if (A[0] == 'c')

RowClickP = 2;

else

if (A[0] == 'd')

RowClickP = 3;

else

if (A[0] == 'e')

RowClickP = 4;

else

if (A[0] == 'f')

RowClickP = 5;

else

if (A[0] == 'g')

RowClickP = 6;

else

if (A[0] == 'h')

RowClickP = 7;

ColumnClickP = 7 - ((System.Convert.ToInt32(A[1]) - 48) - 1);

;

if (A[2] == 'a')

RowRealesed = 0;

else

if (A[2] == 'b')

RowRealesed = 1;

else

if (A[2] == 'c')

RowRealesed = 2;

else

if (A[2] == 'd')

RowRealesed = 3;

else

if (A[2] == 'e')

RowRealesed = 4;

else

if (A[2] == 'f')

RowRealesed = 5;

else

if (A[2] == 'g')

RowRealesed = 6;

else

if (A[2] == 'h')

RowRealesed = 7;

ColumnRealeased = 7 - ((System.Convert.ToInt32(A[3]) - 48) - 1);

if (A.Length == 5)

{

if (A[4] == 'p')

return -1;

else

if (A[4] == 'n')

return -3;

else

if (A[4] == 'b')

return -2;

else

if (A[4] == 'r')

return -4;

else

if (A[4] == 'q')

return -5;

else

if (A[4] == 'P')

return 1;

else

if (A[4] == 'N')

return 3;

else

if (A[4] == 'B')

return 2;

else

if (A[4] == 'R')

return 4;

else

if (A[4] == 'Q')

return 5;

}

}

catch (Exception t)

{

Log(t);

return -1;

}

return 0;

}

void SetRowColumnReleasedAndClickP()

{

if (MovmentsNumber >= 1)

{

int[,] Table1 = new int[8, 8];

int[,] Table2 = new int[8, 8];

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

{

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

{

Table1[i, j] = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][i, j];

}

}

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

{

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

{

Table2[i, j] = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][i, j];

}

}

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

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

{

if (Table1[i, j] != Table2[i, j])

{

if (Table1[5, 7] == -6 && Table1[7, 7] == -4 && Table2[5, 7] == 0 && Table2[7, 7] == 0)

{

RefrigtzDLL.ChessRules.SmallKingCastleBrown = true;

FenCastling = 1;

}

if (Table1[5, 7] == -6 && Table1[0, 7] == -4 && Table2[5, 7] == 0 && Table2[0, 7] == 0)

{

RefrigtzDLL.ChessRules.BigKingCastleBrown = true;

FenCastling = 2;

}

else

if (Table2[i, j] == 0 && Table1[i, j] != 0)

{

RowClickP = i;

ColumnClickP = j;

}

else if (Table2[i, j] != 0 && Table1[i, j] == 0)

{

RowRealesed = i;

ColumnRealeased = j;

}

}

}

}

}

bool WaitOnMovmentOccured(String Preveios, ref String Next)

{

Next = "";

try

{

if (File.Exists("output.txt"))

Next = File.ReadAllText("output.txt");

if (Preveios == Next)

return true;

}

catch (Exception t)

{

Log(t);

Thread.Sleep(1000);

//goto Again;

}

if (Preveios == Next)

return true;

//WaitOn = false;

return false;

}

bool SetMovement(int i, int j, int ii, int jj)

{

if (RefrigtzDLL.AllDraw.MouseClick == 2)

{

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, Table[i, j], Table, OrderPlate, i, j);

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

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

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

{

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

{

Tab[ik, jk] = Table[ik, jk];

}

}

if (A.Rules(i, j, ii, jj, a, Tab[i, j]))

{

Tab[ii, jj] = Tab[i, j];

Tab[i, j] = 0;

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

Draw.FoundOfCurrentTableNode(Table, OrderPlate, ref THIS, ref FOUND);

if (FOUND)

{

Table[ii, jj] = Table[i, j];

Table[i, j] = 0;

Draw = THIS;// OrderPlate = Draw.OrderP;

RefrigtzDLL.AllDraw.TableListAction.Add(Tab);

Person = false;

if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch && RefrigtzDLL.AllDraw.StoreADraw.Count > 0)

{

RefrigtzDLL.AllDraw.StoreADraw.RemoveAt(0);

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Clear();

RefrigtzDLL.AllDraw.StoreADraw[0].TableList.Add(Table);

RefrigtzDLL.AllDraw.StoreADraw[0].SetRowColumn(0);

}

RefrigtzDLL.AllDraw.Less += CalculateMoveMentHueuristicUser(Table, -1, (int)RowRealesed, (int)ColumnRealeased, (int)RowRealesedP, (int)ColumnRealeasedP, Color.Brown);

if (RefrigtzDLL.AllDraw.TableListAction.Count > 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

OrderPlate = OrderPlate \* -1;

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

MovmentsNumber++;

BrownTimer.StopTime();

GrayTimer.StartTime();

InsertTableAtDataBase(Table);

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, King, -6);

Clicked = false;

SetBoxText("\r\nDraw Found");

RefreshBoxText();

// SetRefregitzBicture();

return true;

}

else

return false;

}

else

{

if (System.Math.Abs(Tab[i, j]) == 1)

{

Draw.SolderesOnTable[Soldier].Row = (int)RowClickP;

Draw.SolderesOnTable[Soldier].Column = (int)ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 2)

{

Draw.ElephantOnTable[Elefant].Row = (int)RowClickP;

Draw.ElephantOnTable[Elefant].Column = (int)ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 3)

{

Draw.HoursesOnTable[Hourse].Row = RowClickP;

Draw.HoursesOnTable[Hourse].Column = ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 1)

{

Draw.SolderesOnTable[Soldier].Row = (int)RowClickP;

Draw.SolderesOnTable[Soldier].Column = (int)ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 4)

{

Draw.CastlesOnTable[Castle].Row = (int)RowClickP;

Draw.CastlesOnTable[Castle].Column = (int)ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 5)

{

Draw.MinisterOnTable[Minister].Row = (int)RowClickP;

Draw.MinisterOnTable[Minister].Column = (int)ColumnClickP;

}

else

if (System.Math.Abs(Tab[i, j]) == 6)

{

Draw.KingOnTable[King].Row = (int)RowClickP;

Draw.KingOnTable[King].Column = (int)ColumnClickP;

}

SetBoxText("\r\nWait...");

RefreshBoxText();

//if (!FirstMovmentOnLoad && System.Convert.ToInt32(comboBoxMaxLevel.Text) > 2) Draw.InitiateAStarGreedytOneNode(0, (int)RowRealesed, (int)ColumnRealeased, Color.Gray, Table, OrderPlate \* -1, King, -6);

Clicked = true;

// SetRefregitzBicture();

return true;

}

}

return false;

}

void ComputerByComputerAliceAsStockFish(ref Process proc)

{

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

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

{

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

{

Tab[i, j] = Table[i, j];

}

}

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

if (OrderPlate == 1)

{

SetBoxText("\r\nStockfish Number " + MovmentsNumber + " By Bob!");

RefreshBoxText();

}

else

{

SetBoxText("\r\nStockfish Number " + MovmentsNumber + " By Alice!");

RefreshBoxText();

}

// RefregitzisCurrent = false;

String Pre = "";

if (File.Exists("output.txt"))

Pre = File.ReadAllText("output.txt");

StreamWriter sw = proc.StandardInput;

string input = "go depth " + comboBoxMaxLevelText + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

String wr = "";

WaitOn = true;

do

{

try

{

Thread.Sleep(100);

input = "wr" + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

Thread.Sleep(100);

WaitOn = WaitOnMovmentOccured(Pre, ref wr);

}

catch (Exception t)

{

Log(t);

}

} while (WaitOn);

if (wr == "e8c8")

{

FenCastling = 1;

RefrigtzDLL.ChessRules.BigKingCastleBrown = true;

}

else

if (wr == "e8g8")

{

RefrigtzDLL.ChessRules.SmallKingCastleBrown = true;

FenCastling = 0;

}

else

FenCastling = -1;

int Pro = 0;

if (FenCastling == -1)

{

do

{

Pro = SetRowColumn(wr);

} while (Pro == -1);

File.AppendAllText("List.txt", wr + "-");

if (Pro == 0)

{

Tab[(int)RowRealesed, (int)ColumnRealeased] = Tab[(int)RowClickP, (int)ColumnClickP];

Tab[(int)RowClickP, (int)ColumnClickP] = 0;

}

else

{

Tab[(int)RowRealesed, (int)ColumnRealeased] = Pro;

Tab[(int)RowClickP, (int)ColumnClickP] = 0;

}

}

else

if (FenCastling == 1)

{

Tab[0, 0] = 0;

Tab[4, 0] = 0;

Tab[3, 0] = -6;

Tab[4, 0] = -4;

}

else

if (FenCastling == 0)

{

Tab[7, 0] = 0;

Tab[4, 0] = 0;

Tab[6, 0] = -6;

Tab[5, 0] = -4;

}

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

if (FenCastling != -1)

{

RefrigtzDLL.ChessRules.BigKingCastleBrown = false;

RefrigtzDLL.ChessRules.SmallKingCastleBrown = false;

}

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

sw = proc.StandardInput;

input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

if (OrderPlate == 1)

this.SetBoxText("\r\nThinking Finished by Bob!");

else

this.SetBoxText("\r\nThinking Finished by Alice!");

RefreshBoxText();

RefrigtzDLL.AllDraw.TableListAction.Add(Tab);

if (RefrigtzDLL.AllDraw.TableListAction.Count >= 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Tab, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Tab, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

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

{

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

{

Table[i, j] = Tab[i, j];

}

}

SetDrawFounding(ref FOUND, ref THIS, false);

OrderPlate \*= -1;

BobSection = true;

//Draw.TableList.Clear();

//Draw.TableList.Add(Tab);

//Draw.SetRowColumn(0);

InsertTableAtDataBase(Tab);

BrownTimer.StopTime();

GrayTimer.StartTime();

SetBoxTextWrite(Out);

DrawImageOfMain();

}

void ComputerByComputerBobAsRefregitz(ref Process proc)

{

if (t3.IsAlive) new Syncronization(t3, 1); ;

BobAction();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

FenCastling = 1;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

FenCastling = 0;

}

else

FenCastling = -1;

RowClickP = RefrigtzDLL.AllDraw.LastRow;

ColumnClickP = RefrigtzDLL.AllDraw.LastColumn;

RowRealesed = RefrigtzDLL.AllDraw.NextRow;

ColumnRealeased = RefrigtzDLL.AllDraw.NextColumn;

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

//Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

SetBoxTextWrite(Out);

DrawImageOfMain();

}

public void SetDrawFounding(ref bool FOUND, ref RefrigtzDLL.AllDraw THIS, bool First)

{

Object O = new Object();

lock (O)

{

FOUND = false;

THIS = null;

Color a = Color.Brown;

//if (First)

//Draw.FoundOfCurrentTableNode(Table, OrderPlate \* -1, ref THIS, ref FOUND);

//else

Draw.FoundOfCurrentTableNode(Table, OrderPlate, ref THIS, ref FOUND);

if (FOUND)

{

Draw = THIS;

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

//

//

SetBoxText("\r\nDraw Found");

RefreshBoxText();

}

else

{

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

RefrigtzDLL.AllDraw.DepthIterative = 0;

SetBoxText("\r\nDraw Not Found");

RefreshBoxText();

}

}

}

//All Operation of Thinking Handling.

void AllOperations()

{

while (!AllDrawLoad || RefrigtzDLL.AllDraw.TableListAction.Count == 0) { Thread.Sleep(100); }

//Fen();

String FolderLocation = Root;

sortOutput = new StringBuilder("");

ProcessStartInfo start = new ProcessStartInfo();

start.FileName = FolderLocation + "\\" + "stockfish8.exe";

start.UseShellExecute = false;

start.RedirectStandardOutput = true;

start.RedirectStandardInput = true;

start.RedirectStandardError = true;

start.CreateNoWindow = true;

start.ErrorDialog = false; // Run the external process & wait for it to finish

Process proc = new Process();

//proc.StartInfo = start;

//proc.StartInfo.RedirectStandardOutput = true;

//proc.StartInfo.RedirectStandardInput = true;

//proc.StartInfo.RedirectStandardError = true;

proc.OutputDataReceived += new DataReceivedEventHandler(SortOutputHandler);

proc.ErrorDataReceived += new DataReceivedEventHandler(SortOutputHandler);

proc = Process.Start(start);

proc.BeginOutputReadLine();

proc.BeginErrorReadLine();

if (File.Exists("output.txt"))

File.Delete("output.txt");

if ((MovmentsNumber > 0) && Stockfish)

{

/\*

SetRowColumnReleasedAndClickP();

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

\*/

if (OrderPlate == -1)

{

/\*RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

String fens = Fen();

StreamWriter sw = proc.StandardInput;

sw = proc.StandardInput;

String input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

\*/

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

MessageBox.Show("No Konwledgs to begin with stockfish! Please delete one node of last table and continue");

Application.ExitThread();

Application.Exit();

}

}

if (OrderPlate == 1)

BobSection = true;

else

BobSection = false;

do

{

//if (RefrigtzDLL.AllDraw.THISDummy != null)

// RefrigtzDLL.AllDraw.THISDummy.Clone(Draw);

if (MovmentsNumber == 0)

{

MovmentsNumberMax = MovmentsNumber;

if (Stockfish)

{

GrayTimer.Sign = 1;

BrownTimer.Sign = 1;

if (OrderPlate == 1)

{

GrayTimer.StartTime();

BrownTimer.StopTime();

BobSection = true;

AliceSection = false;

GrayTimer.StartTime();

}

else

{

BrownTimer.StartTime();

GrayTimer.StopTime();

BobSection = false;

AliceSection = true;

BrownTimer.StartTime();

}

}

}

RefrigtzDLL.AllDraw.SyntaxToWrite = "";

try

{

if (RefrigtzDLL.ChessGeneticAlgorithm.NoGameFounf)

{

SetBoxText("No Game Could be continued!");

RefreshBoxText();

}

if (RefrigtzDLL.AllDraw.MouseClick == 0 && !RefrigtzDLL.ThinkingChess.ThinkingRun)

{

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, 1, Table, OrderPlate, -1, -1);

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

RefrigtzDLL.ChessRules AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, OrderPlate);

//if (!UsePenaltyRegardMechnisamT)

if (AA.CheckMate(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], OrderPlate))

{

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

{

EndOfGame = true;

return;

}

else

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

{

EndOfGame = true;

return;

}

}

if (A.CheckMate(Table, OrderPlate) || A.Pat(Table, OrderPlate, a))

{

if (A.CheckMateGray || A.CheckMateBrown || EndOfGame || A.PatkGray || A.PatBrown)

{

StateCC = false;

StateCP = false;

Person = false;

if (A.CheckMateGray || A.CheckMateBrown)

SetBoxText("\r\nCheckMate!");

else if (A.PatkGray || A.PatBrown)

SetBoxText("\r\nPat!");

RefreshBoxText();

if (AllOperate.IsAlive)

new Syncronization(AllOperate, 1);

}

else

{

if (A.CheckMateGray && OrderPlate == 1)

if (A.CheckGray || A.CheckBrown)

{

if (OrderPlate == 1)

SetBoxText("\r\nGray OrderPlate!Check!");

else

SetBoxText("\r\nBrown OrderPlate!Check!");

}

}

}

}

if (Sec.radioButtonGrayOrder.Checked)

{

if (StateCC)

{

if (Stockfish)

{

if (ArrangmentsChanged)

{

if (Blitz)

{

if (BobSection && OrderPlate == 1)//Gray is Refregitz.

ComputerByComputerBobAsRefregitz(ref proc);

else

if (!BobSection && OrderPlate == -1)//Brow is Stockfish.

ComputerByComputerAliceAsStockFish(ref proc);

}

else

if (FullGame)

{

if (BobSection && OrderPlate == 1)//Gray is Refregitz.

ComputerByComputerBobAsRefregitz(ref proc);

else

if (!BobSection && OrderPlate == -1)//Brow is Stockfish.

ComputerByComputerAliceAsStockFish(ref proc);

}

}

else

MessageBox.Show("Mirror Objects Please!");

}

else//Not Stockfish

{

if (Blitz)

{

if (BobSection && OrderPlate == 1)

{

AliceSection = true;

BobSection = false;

BobAction();

}

else

{

Clicked = true;

if (AliceSection && OrderPlate == -1)

{

AliceSection = false;

BobSection = true;

AliceAction();

}

}

}

else

if (FullGame)

{

if (BobSection && OrderPlate == 1)

{

AliceSection = true;

BobSection = false;

BobAction();

}

else

{

Clicked = true;

if (AliceSection && OrderPlate == -1)

{

AliceSection = false;

BobSection = true;

AliceAction();

}

}

}

}

}

else

if (StateCP)

{

if (Stockfish)

{

if (ArrangmentsChanged)

{

if (BobSection && OrderPlate == 1)

{

Clicked = true;

GrayTimer.StartTime();

BrownTimer.StopTime();

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

//

//

//

RefrigtzDLL.AllDraw.MouseClick = 0;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Color a = Color.Brown;

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

SetDrawFounding(ref FOUND, ref THIS, false);

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

FenCastling = 1;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

FenCastling = 0;

}

else

FenCastling = -1;

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

//Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

}

else

if (!BobSection && OrderPlate == -1)

MessageBox.Show("No Knowledge!");

}

else

MessageBox.Show("Mirror the Objects please!");

}

else

{

if (Blitz)

{

if (BobSection && OrderPlate == 1)

{

bool SCC = StateCC;

bool SCP = StateCP;

bool SCG = StateGe;

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

//

//

//

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Color a = Color.Brown;

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

//SetDrawFounding(ref FOUND, ref THIS, true);

BobSection = false;

StateCC = SCC;

StateCP = SCP;

StateGe = SCG;

}

else

if (!BobSection && OrderPlate == -1)

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

Person = true;

AliceWithPerson();

BobSection = true;

}

}

else

if (FullGame)

{

if (BobSection && OrderPlate == 1)

{

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

//

//

//

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

BobSection = false;

}

else

if (!BobSection && OrderPlate == -1)

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

Person = true;

AliceWithPerson();

BobSection = true;

}

}

}

}

else

if (StateGe)

{

if (Blitz)

{

GeneticAction();

}

else

if (FullGame)

{

GeneticAction();

}

}

}

else

if (Sec.radioButtonBrownOrder.Checked)

{

if (StateCC)

{

if (Stockfish)

{

if (ArrangmentsChanged)

{

if (Blitz)

{

if (BobSection && OrderPlate == 1)//Gray is Refregitz.

ComputerByComputerBobAsRefregitz(ref proc);

else

if (!BobSection && OrderPlate == -1)//Brow is Stockfish.

ComputerByComputerAliceAsStockFish(ref proc);

}

else

if (FullGame)

{

if (BobSection && OrderPlate == 1)//Gray is Refregitz.

ComputerByComputerBobAsRefregitz(ref proc);

else

if (!BobSection && OrderPlate == -1)//Brow is Stockfish.

ComputerByComputerAliceAsStockFish(ref proc);

}

}

else

MessageBox.Show("Mirror Objects Please!");

}

else//Not Stockfish

{

if (Blitz)

{

if (BobSection && OrderPlate == 1)

{

AliceSection = true;

BobSection = false;

BobAction();

}

else

{

Clicked = true;

if (AliceSection && OrderPlate == -1)

{

AliceSection = false;

BobSection = true;

AliceAction();

}

}

}

else

if (FullGame)

{

if (BobSection && OrderPlate == 1)

{

AliceSection = true;

BobSection = false;

BobAction();

}

else

{

Clicked = true;

if (AliceSection && OrderPlate == -1)

{

AliceSection = false;

BobSection = true;

AliceAction();

}

}

}

}

}

else

if (StateCP)

{

if (Stockfish)

{

if (ArrangmentsChanged)

{

if (BobSection && OrderPlate == 1)

{

Clicked = true;

GrayTimer.StartTime();

BrownTimer.StopTime();

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

FenCastling = 1;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

FenCastling = 0;

}

else

FenCastling = -1;

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

//Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

}

else

if (!BobSection && OrderPlate == -1)

MessageBox.Show("No Knowledge!");

}

else

MessageBox.Show("Mirror the Objects please!");

}

else

if (Blitz)

{

if (!BobSection && OrderPlate == -1)

{

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

//Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound,IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

//Draw.TableList.Clear();

//Draw.TableList.Add(Table);

//Draw.SetRowColumn(0);

BobSection = true;

}

else

if (BobSection && OrderPlate == 1)

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

Person = true;

BobWithPerson();

BobSection = false;

}

}

else

if (FullGame)

{

if (!BobSection && OrderPlate == -1)

{

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

BobSection = true;

}

else

if (BobSection && OrderPlate == 1)

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

Person = true;

AliceWithPerson();

BobSection = false;

}

}

}

else

if (StateGe)

{

if (Blitz)

{

GeneticAction();

}

else

if (FullGame)

{

GeneticAction();

}

}

}

/\* if (((StateCP) || Person || Blitz) && (!StateCC))

{

try

{

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

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

if (RowClickP != -1 && ColumnClickP != -1)

Tab = VeryFye(Table, OrderPlate, a);

if ((RowRealesed >= 0) && (RowRealesed < 8) && (ColumnRealeased >= 0) && (ColumnRealeased < 8) && ((int)(this.pictureBoxRefrigtz.Width / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Width / 8) < 8) && ((int)(this.pictureBoxRefrigtz.Height / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Height / 8) < 8))

{

if ((RowRealesed + ColumnRealeased) % 2 == 0)

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\Black.jpg"), new Rectangle((int)RowRealesed, (int)ColumnRealeased, (int)(this.pictureBoxRefrigtz.Width / 8), (int)(this.pictureBoxRefrigtz.Height / 8)));

else

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\White.jpg"), new Rectangle((int)RowRealesed, (int)ColumnRealeased, (int)(this.pictureBoxRefrigtz.Width / 8), (int)(this.pictureBoxRefrigtz.Height / 8)));

}

}

catch (Exception t)

{ //RunInFront();

Log(t);

}

int Or = 1;

if (Sec.radioButtonBrownOrder.Checked)

Or = -1;

if (Sec.radioButtonGrayOrder.Checked)

{

if (!Stockfish)

{

if ((StateCP || Blitz) && (OrderPlate == Or))

{

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

{

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound,IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

}

else

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

if ((StateCP) || Blitz)

{

if (Sec.radioButtonGrayOrder.Checked && OrderPlate == -1)

{

Person = true;

AliceWithPerson();

}

}

}

}

else//Stockfish with Person

{

if (ArrangmentsChanged)

{

if (BobSection)

{

Clicked = true;

GrayTimer.StartTime();

BrownTimer.StopTime();

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound,IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

FenCastling = 1;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

FenCastling = 0;

}

else

FenCastling = -1;

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

//Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

}

else

if (!BobSection)

{

}

}

else

{

Stockfish = false;

StateCC = false;

StateCP = false;

MessageBox.Show("Mirror Objects please!");

}

}

}

else if (Sec.radioButtonBrownOrder.Checked)

{

if ((StateCP || Blitz) && (OrderPlate == Or))

{

Person = false;

RefrigtzDLL.AllDraw.FoundATable = false;

Clicked = true;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

{

if (tM == null)

tM = new Thread(new ThreadStart(Movements));

if (!tM.IsAlive)

{

tM.Start();

}

Wait();

Clicked = false;

RefrigtzDLL.AllDraw.MouseClick = 0;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

SetBoxText("\r\nYour Ready!");

RefreshBoxText();

}

}

else

{

if (tM != null)

{

try

{

tM.Abort();

tM.Join();

tM = null;

}

catch (Exception t)

{

Log(t);

}

}

if ((StateCP) || Blitz)

{

if (Sec.radioButtonBrownOrder.Checked && OrderPlate == 1)

{

Person = true;

BobWithPerson();

}

}

}

}

}

else

if (StateCC)

{

if (!Stockfish)

{

if (Sec.radioButtonGrayOrder.Checked)

{

if (BobSection)

{

Clicked = true;

BobAction();

}

else

{

Clicked = true;

if (AliceSection)

{

if (t2.IsAlive) new Syncronization(t2, 1); ;

AliceSection = false;

BobSection = true;

AliceAction();

}

}

}

else

if (Sec.radioButtonBrownOrder.Checked)

{

if (AliceSection)

{

Clicked = true;

AliceAction();

AliceSection = false;

BobSection = true;

}

else

if (BobSection)

{

AliceSection = true;

BobSection = false;

BobAction();

}

}

}

else

{

if (ArrangmentsChanged)

{

if (Sec.radioButtonGrayOrder.Checked)

{

if (RefregitzisCurrent && BobSection)

{

if (t3.IsAlive) new Syncronization(t3, 1); ;

BobAction();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

FenCastling = 1;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

FenCastling = 0;

}

else

FenCastling = -1;

RowClickP = FormRefrigtz.LastRow;

ColumnClickP = FormRefrigtz.LastColumn;

RowRealesed = FormRefrigtz.NextRow;

ColumnRealeased = FormRefrigtz.NextColumn;

String fens = Fen();

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

StreamWriter sw = proc.StandardInput;

string input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

//Thread.Sleep(1500);

BobSection = false;

MovmentsNumber++;

GrayTimer.StopTime();

BrownTimer.StartTime();

}

else if (!BobSection)

{

if (OrderPlate == 1)

{

SetBoxText("\r\nStockfish Number " + MovmentsNumber + " By Bob!");

RefreshBoxText();

}

else

{

SetBoxText("\r\nStockfish Number " + MovmentsNumber + " By Alice!");

RefreshBoxText();

}

// RefregitzisCurrent = false;

String Pre = "";

if (File.Exists("output.txt"))

Pre = File.ReadAllText("output.txt");

StreamWriter sw = proc.StandardInput;

string input = "go depth " + comboBoxMaxLevelText + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

WaitOn = true;

do

{

try

{

Thread.Sleep(1000);

input = "wr" + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

WaitOn = WaitOnMovmentOccured(Pre);

}

catch (Exception t)

{

Log(t);

}

} while (WaitOn);

while (WaitOn) { Thread.Sleep(100); }

String wr = File.ReadAllText("output.txt");

if (wr == "e8c8")

{

FenCastling = 1;

RefrigtzDLL.ChessRules.BigKingCastleBrown = true;

}

else

if (wr == "e8g8")

{

RefrigtzDLL.ChessRules.SmallKingCastleBrown = true;

FenCastling = 0;

}

else

FenCastling = -1;

int Pro = 0;

if (FenCastling == -1)

{

Pro = SetRowColumn(wr);

if (Pro == 0)

{

Table[(int)RowRealesed, (int)ColumnRealeased] = Table[(int)RowClickP, (int)ColumnClickP];

Table[(int)RowClickP, (int)ColumnClickP] = 0;

}

else

{

Table[(int)RowRealesed, (int)ColumnRealeased] = Pro;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

}

}

else

if (FenCastling == 1)

{

Table[0, 0] = 0;

Table[4, 0] = 0;

Table[3, 0] = -6;

Table[4, 0] = -4;

}

else

if (FenCastling == 0)

{

Table[7, 0] = 0;

Table[4, 0] = 0;

Table[6, 0] = -6;

Table[5, 0] = -4;

}

if (RefrigtzDLL.ChessRules.BigKingCastleGray)

{

RefrigtzDLL.ChessRules.BigKingCastleGray = false;

}

else

if (RefrigtzDLL.ChessRules.SmallKingCastleGray)

{

RefrigtzDLL.ChessRules.SmallKingCastleGray = false;

}

String fens = Fen();

if (FenCastling != -1)

{

RefrigtzDLL.ChessRules.BigKingCastleBrown = false;

RefrigtzDLL.ChessRules.SmallKingCastleBrown = false;

}

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

sw = proc.StandardInput;

input = fens + "\r\n";

sw.BaseStream.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

sw.Flush();

RefrigtzDLL.AllDraw.TableListAction.Add(Table);

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound,IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

this.SetBoxText("\r\nThinking Finished by Bob!");

RefreshBoxText();

if (RefrigtzDLL.AllDraw.TableListAction.Count >= 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0,OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0,OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

OrderPlate \*= -1;

BobSection = true;

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

InsertTableAtDataBase(Table);

GrayTimer.StartTime();

BrownTimer.StopTime();

}

}

}

else

{

Stockfish = false;

StateCC = false;

StateCP = false;

MessageBox.Show("Mirror Objects please!");

}

}

}

else

if (StateGe)

{

GeneticAction();

}

\*/

while ((!StateCP && !StateCC && !StateGe && !Blitz))

Thread.Sleep(1000);

{

if (RefrigtzDLL.AllDraw.MouseClick == 0 && RowClickP != -1)

{

//HitRecustruct();

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

RowRealesedP = -1;

ColumnClickP = -1;

}

else

{

if (RefrigtzDLL.AllDraw.MouseClick >= 2)

{

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

}

}

RefrigtzDLL.AllDraw.RedrawTable = false;

}

}

catch (Exception t)

{

Log(t);

}

Thread.Sleep(100);

} while (true);

}

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 <= RefrigtzDLL.AllDraw.MaxAStarGreedy)

{

int[] Loc = new int[2];

Loc[0] = i;

Loc[1] = j - i;

TagList.Add(Loc);

}

}

catch (Exception 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><br/>";

return R;

}

}

//Deligation of Control Threading.

delegate void SetTextBoxWriteTextCallback(String state);

public void SetBoxTextWrite(String state)

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

try

{

//SetTextBoxWriteTextCallback d = new SetTextBoxWriteTextCallback(SetBoxTextWrite);

String R = File.ReadAllText(Root + "\\Database\\Monitor.html");

R = R.Replace("</body>", "");

File.AppendAllText(Root + "\\Database\\Monitor.html", "\n\t" + state);

File.AppendAllText(Root + "\\Database\\Monitor.html", "\n\t" + "</body>");

Out = "";

}

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

}

else

{

try

{

Out = "";

}

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

}

this.RefreshBoxText();

}

}

delegate void SetTextBoxTextCallback(String state);

public void SetBoxText(String state)

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

try

{

String A = TimerText.ReturnTime();

if (OrderPlate == -1)

A = TimerText.ReturnTime();

SetTextBoxTextCallback d = new SetTextBoxTextCallback(SetBoxText);

this.Invoke(new Action(() => textBoxText.AppendText(state + " At Time " + A)));

state = CreateHtmlTag(state);

Out += state + " At Time " + A + "\n\t";

}

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

}

else

{

try

{

String A = GrayTimer.ReturnTime();

if (OrderPlate == -1)

A = BrownTimer.ReturnTime();

textBoxText.AppendText(state + " At Time " + A);

state = CreateHtmlTag(state);

Out += state + " At Time " + A + "\n\t";

}

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

}

this.RefreshBoxText();

}

}

delegate void SetcomboBoxTextCallback();

public void SetcomboBoxText()

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

Invoke((MethodInvoker)delegate ()

{

RefrigtzDLL.AllDraw.THIScomboBoxMaxLevelText = comboBoxMaxLevel.Text;

});

}

}

delegate void RefreshhTextBoxTextCallback();

public void RefreshBoxText()

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

try

{

//RefreshhTextBoxTextCallback d = new RefreshhTextBoxTextCallback(RefreshBoxText);

//this.Invoke(new Action(() => textBoxText.Refresh()));

textBoxText.Refresh();

}

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

}

else

{

try

{

this.textBoxStatistic.Refresh();

}

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

}

}

}

delegate void SetTextBoxStatisticCallback(String state);

public void SetBoxStatistic(String state)

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

try

{

SetTextBoxStatisticCallback d = new SetTextBoxStatisticCallback(SetBoxStatistic);

this.Invoke(new Action(() => textBoxStatistic.AppendText(state)));

}

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

}

else

{

try

{

textBoxStatistic.AppendText(state);

}

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

}

this.RefreshBoxStatistic();

}

}

delegate void RefreshhTextBoxStatisticCallback();

public void RefreshBoxStatistic()

{

Object O = new Object();

lock (O)

{

// InvokeRequired required compares the thread ID of the

// calling thread to the thread ID of the creating thread.

// If these threads are different, it returns true.

if (this.InvokeRequired)

{

RefreshhTextBoxStatisticCallback d = new RefreshhTextBoxStatisticCallback(RefreshBoxStatistic);

this.Invoke(new Action(() => textBoxStatistic.Refresh()));

}

else

{

textBoxStatistic.Refresh();

}

}

}

//The State of Alice with Person Thinking.

void AliceWithPerson()

{

Object O = new Object();

lock (O)

{

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

RefrigtzDLL.ThinkingChess.ThinkingRun = true;

PaintedPaused = true;

try

{

Begin1:

LoadConvertedTable = false;

Color a = Color.Gray;

bool StoreStateCC = false, StoreStateCP = false, StoreStateGe = false;

OpBeforeThinking(ref a, ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

//SetDrawFounding(ref FOUND, ref THIS, true);

// if (!FirstMovmentOnLoad)

// FOUND = true;

int LeafAStarGrteedy = 0;

Table = Draw.Initiate(1, 4, a, Table, OrderPlate, false, FOUND, LeafAStarGrteedy);

LoadConvertedTable = true;

PaintedPaused = false;

this.SetBoxText("\r\nThinking Finished!");

try

{

if (TableZero(Table))

OpTableZero();

}

catch (Exception t)

{

Log(t);

goto Begin1;

}

FOUND = false;

THIS = null;

SetDrawFounding(ref FOUND, ref THIS, false);

SetAndConfirmSyntax();

OpAfterAllTinking(ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

OrderPlate \*= -1;

SetBoxTextWrite(Out);

DrawImageOfMain();

}

catch (Exception t)

{

Log(t);

}

}

}

//The State of Bob with Person Thinking.

void BobWithPerson()

{

Object O = new Object();

lock (O)

{

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

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

LoadConvertedTable = false;

RefrigtzDLL.ThinkingChess.ThinkingRun = true;

try

{

Begin1:

Color a = Color.Gray;

bool StoreStateCC = false, StoreStateCP = false, StoreStateGe = false;

OpBeforeThinking(ref a, ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

//SetDrawFounding(ref FOUND, ref THIS, true);

int LeafAStarGrteedy = 0;

Table = Draw.Initiate(1, 4, a, Table, OrderPlate, false, FOUND, LeafAStarGrteedy);

try

{

if (TableZero(Table))

OpTableZero();

}

catch (Exception t)

{

Log(t);

goto Begin1;

}

SetAndConfirmSyntax();

FOUND = false;

THIS = null;

SetDrawFounding(ref FOUND, ref THIS, false);

OpAfterAllTinking(ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

OrderPlate \*= -1;

SetBoxTextWrite(Out);

DrawImageOfMain();

}

catch (Exception t)

{

Log(t);

this.SetBoxText("\r\nError!");

}

}

}

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 OpBeforeThinking(ref Color a, ref bool StoreStateCC, ref bool StoreStateCP, ref bool StoreStateGe)

{

Object O = new Object();

lock (O)

{

StoreStateCC = StateCC;

StoreStateCP = StateCP;

StoreStateGe = StateGe;

//AliceSection = false;

MovmentsNumber++;

if (OrderPlate == 1)

{

SetBoxText("\r\n Movments Number " + MovmentsNumber.ToString() + " is Gray OrderPlate!");

RefreshBoxText();

}

else

{

SetBoxText("\r\n Movments Number " + MovmentsNumber.ToString() + " is Brown OrderPlate!");

RefreshBoxText();

}

if (OrderPlate == 1)

{

this.SetBoxText("\r\nThinking Begin By Bob!");

RefreshBoxText();

}

else

{

this.SetBoxText("\r\nThinking Begin By Alice!");

RefreshBoxText();

}

RefreshBoxText();

a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

}

}

void OpTableZero()

{

Object O = new Object();

lock (O)

{

//Saved Midle Target.

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

checkBoxUsePenaltyRegradMechnisam.Checked = false;

SettingPRFALSE = true;

UpdateConfigurationTable();

System.IO.File.AppendAllText("CheckSum.btt", "\n\tInstallation Begine On " + DateTime.Now.ToString());

String FolderLocation = Root;

int exitCode = 0;

// Prepare the process to run

ProcessStartInfo start = new ProcessStartInfo();

// Prepare the process to run

// Enter in the command line arguments, everything you would enter after the executable name itself

start.Arguments = "";

// Enter the executable to run, including the complete path

start.FileName = "\"" + FolderLocation + "\\" + "LoadRP.exe" + "\"";

// Do you want to show a console window?

start.WindowStyle = ProcessWindowStyle.Normal;

start.CreateNoWindow = true;

start.UseShellExecute = true;

// Run the external process & wait for it to finish

using (Process proc = Process.Start(start))

{

proc.WaitForExit(20000);

// Retrieve the app's exit code

exitCode = proc.ExitCode;

}

Application.Exit();

}

}

void SetAndConfirmSyntax()

{

Object O = new Object();

lock (O)

{

RefrigtzDLL.AllDraw.TableListAction.Add(Table);

if (RefrigtzDLL.AllDraw.TableListAction.Count >= 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

}

}

void OpAfterAllTinking(ref bool StoreStateCC, ref bool StoreStateCP, ref bool StoreStateGe)

{

Object O = new Object();

lock (O)

{

InsertTableAtDataBase(Table);

StateCC = StoreStateCC;

StateCP = StoreStateCP;

StateGe = StoreStateGe;

RefrigtzDLL.ThinkingChess.ThinkingRun = false;

UpdateConfigurationTable();

if (OrderPlate == 1)

{

this.SetBoxText("\r\nThinking Finished By Bob!");

RefreshBoxText();

}

else

{

this.SetBoxText("\r\nThinking Finished By Alice!");

RefreshBoxText();

}

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

Clicked = false;

}

}

//Alice Section of Computer by Computer Thinking.

void AliceAction()

{

Object O = new Object();

lock (O)

{

RefrigtzDLL.ThinkingChess.ThinkingRun = true;

Begin4:

Color a = Color.Gray;

bool StoreStateCC = false, StoreStateCP = false, StoreStateGe = false;

OpBeforeThinking(ref a, ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

//SetDrawFounding(ref FOUND, ref THIS, true);

int LeafAStarGrteedy = 0;

Table = Draw.Initiate(1, 4, a, Table, OrderPlate, false, FOUND, LeafAStarGrteedy);

LoadConvertedTable = true;

//StateCP = false;

try

{

if (TableZero(Table))

OpTableZero();

}

catch (Exception t)

{

Log(t);

goto Begin4;

}

SetAndConfirmSyntax();

FOUND = false;

THIS = null;

SetDrawFounding(ref FOUND, ref THIS, false);

OpAfterAllTinking(ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

OrderPlate \*= -1;

BobSection = true;

SetBoxTextWrite(Out);

DrawImageOfMain();

}

}

void DrawImageOfMain()

{

Object O = new Object();

lock (O)

{

RefrigtzDLL.AllDraw.DrawTable = true;

SetPrictureBoxRefregitzInvalidate(pictureBoxRefrigtz);

SetPrictureBoxRefregitzUpdate(pictureBoxRefrigtz);

}

}

void GeneticAction()

{

RefrigtzDLL.ThinkingChess.ThinkingRun = true;

StateGe = false;

MovmentsNumber++;

if (OrderPlate == 1)

SetBoxText("\r\n Movments Number " + MovmentsNumber.ToString() + " is Gray OrderPlate!");

else

SetBoxText("\r\n Movments Number " + MovmentsNumber.ToString() + " is Brown OrderPlate!");

RefreshBoxText();

Begin4:

if (OrderPlate == 1)

SetBoxText("\r\nThinking Begin By Bob!");

else

SetBoxText("\r\nThinking Begin By Alice!");

RefreshBoxText();

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

Draw.InitiateGenetic(1, 4, a, Table, OrderPlate, false);

try

{

//Table = Draw.TableList[0];

}

catch (Exception t)

{

Log(t);

goto Begin4;

}

OrderPlate = OrderPlate \* -1;

Draw.SetRowColumn(0);

RefrigtzDLL.ChessRules.CurrentOrder = OrderPlate;

this.SetBoxText("\r\nThinking Finished By Alice!");

RefreshBoxText();

RefrigtzDLL.AllDraw.TableListAction.Add(Table);

if (RefrigtzDLL.AllDraw.TableListAction.Count >= 1)

{

RefrigtzDLL.ChessGeneticAlgorithm R = new RefrigtzDLL.ChessGeneticAlgorithm(MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

if (R.FindGenToModified(RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2], RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1], RefrigtzDLL.AllDraw.TableListAction, 0, OrderPlate, true))

{

bool HitVal = false;

int Hit = RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRow, R.CromosomColumn];

if (Hit != 0)

HitVal = true;

bool Convert = false;

if (OrderPlate == 1)

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == 1)

{

if (R.CromosomColumn == 7)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleGray || RefrigtzDLL.ChessRules.BigKingCastleGray) && (!RefrigtzDLL.ChessRules.CastleActGray))

RefrigtzDLL.ChessRules.CastleActGray = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActGray, Convert);

}

else

{

if (RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 1][R.CromosomRow, R.CromosomColumn] == -1)

{

if (R.CromosomColumn == 0)

Convert = true;

}

if ((RefrigtzDLL.ChessRules.SmallKingCastleBrown || RefrigtzDLL.ChessRules.BigKingCastleBrown) && (!RefrigtzDLL.ChessRules.CastleActBrown))

RefrigtzDLL.ChessRules.CastleActBrown = true;

RefrigtzDLL.AllDraw.SyntaxToWrite = (new RefrigtzDLL.ChessRules(0, OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged)).CreateStatistic(ArrangmentsChanged, Table, MovmentsNumber + 1, RefrigtzDLL.AllDraw.TableListAction[RefrigtzDLL.AllDraw.TableListAction.Count - 2][R.CromosomRowFirst, R.CromosomColumnFirst], R.CromosomColumn, R.CromosomRow, HitVal, Hit, RefrigtzDLL.ChessRules.CastleActBrown, Convert);

}

SetBoxStatistic(RefrigtzDLL.AllDraw.SyntaxToWrite);

RefreshBoxStatistic();

}

}

if (OrderPlate == 1)

{

BrownTimer.StopTime();

GrayTimer.StartTime();

}

else

{

GrayTimer.StopTime();

BrownTimer.StartTime();

}

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

StateGe = true;

InsertTableAtDataBase(Table);

RefrigtzDLL.ThinkingChess.ThinkingRun = false;

if (t4.IsAlive)

new Syncronization(t4, 1); ;

}

//Bob Section of Computer By Computer Thinking.

void BobAction()

{

Object O = new Object();

lock (O)

{

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

LoadConvertedTable = false;

RefrigtzDLL.ThinkingChess.ThinkingRun = true;

Begin2:

Color a = Color.Gray;

bool StoreStateCC = false, StoreStateCP = false, StoreStateGe = false;

OpBeforeThinking(ref a, ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

bool FOUND = false;

RefrigtzDLL.AllDraw THIS = null;

//SetDrawFounding(ref FOUND, ref THIS, true);

int LeafAStarGrteedy = 0;

Table = Draw.Initiate(1, 4, a, Table, OrderPlate, false, FOUND, LeafAStarGrteedy);

LoadConvertedTable = true;

StateCC = StoreStateCC;

StateCP = StoreStateCP;

StateGe = StoreStateGe;

try

{

////Table = Draw.TableList[0];

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

{

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

{

TableC[i, j] = Table[i, j];

}

}

if (TableZero(Table))

OpTableZero();

}

catch (Exception t)

{

Log(t);

goto Begin2;

}

SetAndConfirmSyntax();

FOUND = false;

THIS = null;

SetDrawFounding(ref FOUND, ref THIS, false);

OpAfterAllTinking(ref StoreStateCC, ref StoreStateCP, ref StoreStateGe);

BobWithStockfishFinished = true;

OrderPlate \*= -1;

SetBoxTextWrite(Out);

DrawImageOfMain();

}

}

public void SetObjectNumbers(int[,] TabS)

{

Draw.SodierMidle = 0;

Draw.SodierHigh = 0;

Draw.ElefantMidle = 0;

Draw.ElefantHigh = 0;

Draw.HourseMidle = 0;

Draw.HourseHight = 0;

Draw.CastleMidle = 0;

Draw.CastleHigh = 0;

Draw.MinisterMidle = 0;

Draw.MinisterHigh = 0;

Draw.KingMidle = 0;

Draw.KingHigh = 0;

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

{

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

{

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

{

Draw.SodierMidle++;

Draw.SodierHigh++;

}

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

{

Draw.ElefantMidle++;

Draw.ElefantHigh++;

}

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

{

Draw.HourseMidle++;

Draw.HourseHight++;

}

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

{

Draw.CastleMidle++;

Draw.CastleHigh++;

}

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

{

Draw.MinisterMidle++;

Draw.MinisterHigh++;

}

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

{

Draw.KingMidle++;

Draw.KingHigh++;

}

else

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

{

Draw.SodierHigh++;

}

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

{

Draw.ElefantHigh++;

}

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

{

Draw.HourseHight++;

}

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

{

Draw.CastleHigh++;

}

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

{

Draw.MinisterHigh++;

}

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

{

Draw.KingHigh++;

}

}

}

}

//Hit Reconstruction of Table.

/\*void HitRecustruct()

{

if (RefrigtzDLL.ChessRules.ExistInDestinationEnemy)

{

if (System.Math.Abs(CurrentKind) == 1)

{

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.SolderesOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.ElephantOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.HoursesOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.CastlesOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.MinisterOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.SolderesOnTable[Soldier].Row == Draw.KingOnTable[i].Row && Draw.SolderesOnTable[Soldier].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

if (System.Math.Abs(CurrentKind) == 2)

{

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.SolderesOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.ElephantOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.HoursesOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.CastlesOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.MinisterOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.ElephantOnTable[Elefant].Row == Draw.KingOnTable[i].Row && Draw.ElephantOnTable[Elefant].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

if (System.Math.Abs(CurrentKind) == 3)

{

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.SolderesOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.ElephantOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.HoursesOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.CastlesOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.MinisterOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.HoursesOnTable[Hourse].Row == Draw.KingOnTable[i].Row && Draw.HoursesOnTable[Hourse].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

if (System.Math.Abs(CurrentKind) == 4)

{

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.SolderesOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.ElephantOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.HoursesOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.CastlesOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.MinisterOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.CastlesOnTable[Castle].Row == Draw.KingOnTable[i].Row && Draw.CastlesOnTable[Castle].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

if (System.Math.Abs(CurrentKind) == 5)

{

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.SolderesOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.ElephantOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.HoursesOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.CastlesOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.MinisterOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.MinisterOnTable[Minister].Row == Draw.KingOnTable[i].Row && Draw.MinisterOnTable[Minister].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

if (System.Math.Abs(CurrentKind) == 6)

{

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.SolderesOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.SolderesOnTable[i].Column)

{

Draw.SolderesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.ElephantOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.ElephantOnTable[i].Column)

{

Draw.ElephantOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.HoursesOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.HoursesOnTable[i].Column)

{

Draw.HoursesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.CastlesOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.CastlesOnTable[i].Column)

{

Draw.CastlesOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.MinisterOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.MinisterOnTable[i].Column)

{

Draw.MinisterOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

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

{

try

{

if (Draw.KingOnTable[King].Row == Draw.KingOnTable[i].Row && Draw.KingOnTable[King].Column == Draw.KingOnTable[i].Column)

{

Draw.KingOnTable[i] = null;

Table[(int)RowClickP, (int)ColumnClickP] = 0;

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

}

\*/

//About Tool Strip Calling.

private void aboutToolStripMenuItem1\_Click(object sender, EventArgs e)

{

AboutBoxChessRefregitz ChessRefrigitz = new AboutBoxChessRefregitz();

ChessRefrigitz.ShowDialog();

ChessRefrigitz.Dispose();

}

//Mouse Click Form Refregitz pictureBox Event Handling.

private void pictureBoxRefrigtz\_MouseClick(object sender, MouseEventArgs e)

{

System.Threading.Thread.Sleep(10);

MouseClicked = true;

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

{

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

{

try

{

if (System.Math.Abs(e.X - i \* (pictureBoxRefrigtz.Image.Width / 8)) < pictureBoxRefrigtz.Image.Width / 8 && System.Math.Abs(e.Y - j \* (pictureBoxRefrigtz.Image.Height / 8)) < pictureBoxRefrigtz.Image.Height / 8)

{

//if (!RefrigtzDLL.ThingsConverter.ClickOcurred)

{

if (RefrigtzDLL.AllDraw.MouseClick == 0)

{

RowClickP = i;

ColumnClickP = j;

for (int ii = 0; ii < Draw.SodierHigh; ii++)

{

try

{

if ((Draw.SolderesOnTable[ii].Row == i & Draw.SolderesOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 1)

{

Soldier = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

for (int ii = 0; ii < Draw.ElefantHigh; ii++)

{

try

{

if ((Draw.ElephantOnTable[ii].Row == i & Draw.ElephantOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 2)

{

Elefant = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

for (int ii = 0; ii < Draw.HourseHight; ii++)

{

try

{

if ((Draw.HoursesOnTable[ii].Row == i & Draw.HoursesOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 3)

{

Hourse = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

for (int ii = 0; ii < Draw.CastleHigh; ii++)

{

try

{

if ((Draw.CastlesOnTable[ii].Row == i & Draw.CastlesOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 4)

{

Castle = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

for (int ii = 0; ii < Draw.MinisterHigh; ii++)

{

try

{

if ((Draw.MinisterOnTable[ii].Row == i & Draw.MinisterOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 5)

{

Minister = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

for (int ii = 0; ii < Draw.KingHigh; ii++)

{

try

{

if ((Draw.KingOnTable[ii].Row == i & Draw.KingOnTable[ii].Column == j) && System.Math.Abs(Table[i, j]) == 6)

{

King = ii;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Selected.");

RefreshBoxText();

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

else

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

RowRealesed = i;

ColumnRealeased = j;

RowClick = i;

ColumnClick = j;

RefrigtzDLL.AllDraw.MouseClick++;

SetBoxText("\r\nObject Released.");

RefreshBoxText();

}

else

{

//Needing for Objects for fen string of stockfish

if (!Stockfish)

{

RowClick = -1;

ColumnClick = -1;

RowClickP = -1;

ColumnClickP = -1;

RowRealesed = -1;

ColumnRealeased = -1;

}

RefrigtzDLL.AllDraw.MouseClick = 0;

SetBoxText("\r\nObject Cleared.");

RefreshBoxText();

}

}

}

return;

}

}

catch (Exception t)

{

Log(t);

}

}

}

}

//Mouse Movments of FormRefregitz PictureBox Event Handling.

private void pictureBoxRefrigtz\_MouseMove(object sender, MouseEventArgs e)

{

DrawImageOfMain();

try

{

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

{

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

{

if ((e != null) && (pictureBoxRefrigtz != null) && (pictureBoxRefrigtz.Image != null) && (System.Math.Abs(e.X - i \* (pictureBoxRefrigtz.Image.Width / 8)) < pictureBoxRefrigtz.Image.Width / 8) && (System.Math.Abs(e.Y - j \* (pictureBoxRefrigtz.Image.Height / 8)) < pictureBoxRefrigtz.Image.Height / 8))

{

if (RefrigtzDLL.AllDraw.MouseClick == 1)

{

ChessTable = pictureBoxRefrigtz.Image;

g = Graphics.FromImage(ChessTable);

for (int ii = 0; ii < pictureBoxRefrigtz.Image.Width; ii += pictureBoxRefrigtz.Image.Width / 8)

for (int jj = 0; jj < pictureBoxRefrigtz.Image.Height; jj += pictureBoxRefrigtz.Image.Height / 8)

{

try

{

if ((i >= 0) && (i < 8) && (j >= 0) && (j < 8) && ((int)(this.pictureBoxRefrigtz.Width / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Width / 8) < 8) && ((int)(this.pictureBoxRefrigtz.Height / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Height / 8) < 8)

)

{

if ((ii + jj) % 2 == 0)

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\Black.jpg"), new Rectangle((int)ii, (int)jj, (int)(this.pictureBoxRefrigtz.Width / 8), (int)(this.pictureBoxRefrigtz.Height / 8)));

else

g.DrawImage(Image.FromFile(Root + "\\Images\\Program\\White.jpg"), new Rectangle((int)ii, (int)jj, (int)(this.pictureBoxRefrigtz.Width / 8), (int)(this.pictureBoxRefrigtz.Height / 8)));

}

}

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

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

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

if (RowClickP != -1 && ColumnClickP != -1)

Tab = VeryFye(Table, OrderPlate, a);

if (Tab != null)

{

if (Tab[ii / (int)(pictureBoxRefrigtz.Image.Width / 8), jj / (int)(pictureBoxRefrigtz.Image.Height / 8)])

{

if (((int)(this.pictureBoxRefrigtz.Width / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Width / 8) < 8) && ((int)(this.pictureBoxRefrigtz.Height / 8) >= 0) && ((int)(this.pictureBoxRefrigtz.Height / 8) < 8 && ((int)(this.pictureBoxRefrigtz.Height) >= jj) && ((int)(this.pictureBoxRefrigtz.Width) >= ii)))

{

g.DrawString("\*", new Font("Times New Roman", 50), new SolidBrush(Color.Red), new Rectangle(new Point(ii, jj), new Size((int)(this.pictureBoxRefrigtz.Width / 8), (int)(this.pictureBoxRefrigtz.Height / 8))));

}

}

}

}

pictureBoxRefrigtz.Image = ChessTable;

g.Dispose();

if (RowRealesed == -1 && ColumnRealeased == -1 & RowRealesedP == -1 && ColumnRealeasedP == -1)

{

RowRealesed = i;

ColumnRealeased = j;

RowRealesedP = i;

ColumnRealeasedP = j;

}

}

RowRealesedP = RowRealesed;

ColumnRealeasedP = ColumnRealeased;

RowRealesed = i;

ColumnRealeased = j;

return;

}

}

}

}

catch (Exception t)

{

Log(t);

}

}

//Computer By Computer tool Strip Menu Item Event Handlimng.

private void computerWithComputerToolStripMenuItem\_Click(object sender, EventArgs e)

{

StateCC = true;

if (OrderPlate == 1)

{

BobSection = true;

AliceSection = false;

GrayTimer.StartTime();

}

else

{

BobSection = false;

AliceSection = true;

BrownTimer.StartTime();

}

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

}

//Computer by Person Illegal name tool Strip Evnt Handling.

private void computerWithComputerToolStripMenuItem1\_Click(object sender, EventArgs e)

{

BobSection = false;

AliceSection = false;

StateCC = false;

StateGe = false;

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount \* 2 + 1).ToString();

if (OrderPlate == 1 && Sec.radioButtonGrayOrder.Checked)

{

Person = true;

StateCP = true;

GrayTimer.StartTime();

}

else if (Sec.radioButtonBrownOrder.Checked && OrderPlate == -1)

{

Person = false;

StateCP = true;

BrownTimer.StartTime();

}

else

{

StateCP = true;

Person = false;

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "You";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "You";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

}

//Button Next Game Analysis Click Event Handling.

private void buttonNext\_Click(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

if (MovmentsNumber < MaxCurrentMovmentsNumber)

{

//Increased a movments.

MovmentsNumber++;

//Read Increased Movments.

Table = ReadTableMovmentNumber();

//Clear Table List of Draw.

Draw.TableList.Clear();

//Add Table to Table List.

Draw.TableList.Add(Table);

//Constructed a Draw All.

Draw.SetRowColumn(0);

//OutPut.

SetBoxText("\r\nMovments Number " + MovmentsNumber.ToString() + " Fronted.");

//Refresh TextBox.

RefreshBoxText();

//Sound a Music.

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

}

}

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

}

}

//Previous Game Analysis Click Event Handling

private void buttonPrevious\_Click(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

if (MovmentsNumber >= 0)

{

MovmentsNumber--;

//Read Current Table List

Table = ReadTableMovmentNumber();

//Clear Current TableList.

Draw.TableList.Clear();

//Add Table To Table List

Draw.TableList.Add(Table);

//Construction of All Things and Thinkings.

Draw.SetRowColumn(0);

//Out Put

SetBoxText("\r\nMovments Number " + MovmentsNumber.ToString() + " Backed.");

//Refresh TextBox.

RefreshBoxText();

//Sound a Music.

using (SoundPlayer soundClick = new SoundPlayer(Root + "\\Music\\Click6.wav"))

{

soundClick.Play();

soundClick.Dispose();

}

//Decreased a Movments.

}

}

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

}

}

//Mouse Double Click pictureBoxRefregitz Click Event Handlimng.

private void pictureBoxRefrigtz\_MouseDoubleClick(object sender, MouseEventArgs e)

{

Thread.Sleep(10);

if (!Maximize)

{

RowP = pictureBoxRefrigtz.Width;

ColP = pictureBoxRefrigtz.Height;

RowS = this.Width;

ColS = this.Height;

this.MaximumSize = new Size(1000, 700);

pictureBoxRefrigtz.MaximumSize = new Size(900, 600);

Maximize = true;

}

else

{

this.MaximumSize = new Size(RowS, ColS);

pictureBoxRefrigtz.MaximumSize = new Size(RowP, ColP);

RowP = pictureBoxRefrigtz.Width;

ColP = pictureBoxRefrigtz.Height;

RowS = this.Width;

ColS = this.Height;

Maximize = false;

}

}

//New Game tool Strip Event Handling.

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

NewTable = true;

this.Hide();

StateCC = false;

BobSection = false;

AliceSection = false;

StateCP = false;

Person = false;

FormRefrigtz New = new FormRefrigtz(false);

New.ShowDialog();

New.Dispose();

}

//Leave FormRefregitz Event Handling.

private void FormRefrigtz\_Leave(object sender, EventArgs e)

{

try

{

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

bookConn.Close();

oleDbCmd.Dispose();

bookConn.Dispose();

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -1, Table, OrderPlate, -1, -1);

RefrigtzDLL.ChessRules AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -1, Table, OrderPlate, -1, -1);

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

if (OrderPlate == 1)

A.CheckMate(Table, OrderPlate);

else

A.CheckMate(Table, OrderPlate);

AA.Pat(Table, OrderPlate, a);

bool Exitting = false;

if (OrderPlate == 1 && (A.CheckMateGray || AA.PatBrown))

Exitting = true;

else

if (OrderPlate == -1 && (A.CheckMateBrown || AA.PatkGray))

Exitting = true;

if (Exitting || RemoveUncomStock)

{

try

{

File.Delete("Run.txt");

}

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

{

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

try

{

if (TTimerSet != null)

TTimerSet.Abort();

if (AllOperate != null)

AllOperate.Abort();

if (t1 != null)

t1.Abort();

if (t2 != null)

t2.Abort();

if (t3 != null)

t3.Abort();

if (t4 != null)

t4.Abort();

if (AllOperate != null)

AllOperate.Abort();

if (tttt != null)

tttt.Abort();

if (ttt != null)

ttt.Abort();

GrayTimer.StopTime();

BrownTimer.StopTime();

TimerText.StopTime();

StateCC = false;

StateCP = false;

StateGe = false;

Person = false;

}

catch (Exception t)

{

Log(t);

}

if (!Directory.Exists(Root + "\\DataBase\\Games"))

Directory.CreateDirectory(Root + "\\DataBase\\Games");

int i = 0;

do { i++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + i.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", Root + "\\Database\\Games\\CurrentBank" + i.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Application.Exit();

return;

}

}

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Application.Exit();

}

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

Sec.Dispose();

//Draw.THIS.Dispose();

//THIs.Dispose();

this.Dispose();

}

//Exit ToolStrip Event Handling.

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

bookConn.Close();

oleDbCmd.Dispose();

bookConn.Dispose();

RefrigtzDLL.ChessRules A = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -1, Table, OrderPlate, -1, -1);

RefrigtzDLL.ChessRules AA = new RefrigtzDLL.ChessRules(0, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged, -1, Table, OrderPlate, -1, -1);

Color a = Color.Gray;

if (OrderPlate == -1)

a = Color.Brown;

if (OrderPlate == 1)

A.CheckMate(Table, OrderPlate);

else

A.CheckMate(Table, OrderPlate);

AA.Pat(Table, OrderPlate, a);

bool Exitting = false;

if (OrderPlate == 1 && (A.CheckMateGray || AA.PatBrown))

Exitting = true;

else

if (OrderPlate == -1 && (A.CheckMateBrown || AA.PatkGray))

Exitting = true;

if (Exitting || RemoveUncomStock)

{

try

{

File.Delete("Run.txt");

}

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

{

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

try

{

if (TTimerSet != null)

TTimerSet.Abort();

if (AllOperate != null)

AllOperate.Abort();

if (t1 != null)

t1.Abort();

if (t2 != null)

t2.Abort();

if (t3 != null)

t3.Abort();

if (t4 != null)

t4.Abort();

if (AllOperate != null)

AllOperate.Abort();

if (tttt != null)

tttt.Abort();

if (ttt != null)

ttt.Abort();

GrayTimer.StopTime();

BrownTimer.StopTime();

TimerText.StopTime();

StateCC = false;

StateCP = false;

StateGe = false;

Person = false;

}

catch (Exception t)

{

Log(t);

}

if (!Directory.Exists(Root + "\\DataBase\\Games"))

Directory.CreateDirectory(Root + "\\DataBase\\Games");

int i = 0;

do { i++; } while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + i.ToString() + ".accdb"));

System.IO.File.Copy(Root + "\\Database\\CurrentBank.accdb", Root + "\\Database\\Games\\CurrentBank" + i.ToString() + ".accdb");

System.IO.File.Delete(Root + "\\Database\\CurrentBank.accdb");

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Application.Exit();

return;

}

}

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Application.Exit();

}

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

}

public RefrigtzDLL.AllDraw RootFound()

{

try

{

if (Draw != null)

{

while (Draw.AStarGreedyString != null)

{

Draw = Draw.AStarGreedyString;

}

}

}

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

return Draw;

}

//AStarGreedy Huristic Checkbox Checked Event Handling.

private void checkBoxAStarGreedyHuristic\_CheckedChanged(object sender, EventArgs e)

{

RefrigtzDLL.AllDraw.DrawTable = false;

//if (!DisableTemporarlyTimerUpdate)

{

//RunInBackGround();

if (checkBoxAStarGreedyHuristic.Checked)

AStarGreedyHuristic = true;

else

AStarGreedyHuristic = false;

if (MovmentsNumber == 0)

{

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

//UpdateConfigurationTable();

//RunInFront();

}

}

//radio Button Checked Box Checked Event Handling.

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

//RunInBackGround();

if (radioButtonOriginalImages.Checked)

RefrigtzDLL.AllDraw.ImagesSubRoot = RefrigtzDLL.AllDraw.ImageRoot + "\\Original\\";

//UpdateConfigurationTable();

//RunInFront();

}

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

}

}

//Big Fitting Checked Box Checked Event Handling.

private void radioButtonBigFittingImages\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

//RunInBackGround();

if (radioButtonBigFittingImages.Checked)

RefrigtzDLL.AllDraw.ImagesSubRoot = RefrigtzDLL.AllDraw.ImageRoot + "\\Fit\\Big\\";

//UpdateConfigurationTable();

//RunInFront();

}

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

}

}

//Samll Fitting Radio Button Checked Event Handling.

private void radioButtonSmallFittingImages\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

//RunInBackGround();

if (radioButtonSmallFittingImages.Checked)

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

//UpdateConfigurationTable();

//RunInFront();

}

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

}

}

//Genetic Algorithm Game tool Strip Event Handling.

private void toolStripMenuItem3\_Click(object sender, EventArgs e)

{

StateGe = true;

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

}

//Stop Button Click Event Handling.

private void buttonStop\_Click(object sender, EventArgs e)

{

//RunInBackGround();

SetBoxText("\r\nAll Stop!.");

}

//AStarGreedy First Search Checked BOX Checked Event Handling.

private void checkBoxAStarGreadyFirstSearch\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

/\*try

{

RefrigtzDLL.AllDraw.AStarGreadyFirstSearch = true;

if (checkBoxAStarGreadyFirstSearch.Checked)

checkBoxUseDoubleTime.Visible = true;

else checkBoxUseDoubleTime.Visible = false;

}

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

\*/

}

}

//Hardes Games tool Strip Event Handling.

private void hardestToolStripMenuItem\_Click(object sender, EventArgs e)

{

checkBoxAStarGreedyHuristic.Checked = true;

checkBoxPredictHuristci.Checked = true;

checkBoxAStarGreadyFirstSearch.Checked = true;

checkBoxBestMovments.Checked = false;

checkBoxOnlySelf.Checked = false;

checkBoxUsePenaltyRegradMechnisam.Checked = true;

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

//Medum Game tool Strip Event Handling .

private void medumToolStripMenuItem\_Click(object sender, EventArgs e)

{

checkBoxAStarGreedyHuristic.Checked = true;

checkBoxPredictHuristci.Checked = true;

checkBoxAStarGreadyFirstSearch.Checked = false;

checkBoxBestMovments.Checked = false;

checkBoxOnlySelf.Checked = false;

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

//Easest tool Strip Event Handling .

private void easestToolStripMenuItem\_Click(object sender, EventArgs e)

{

checkBoxAStarGreedyHuristic.Checked = false;

checkBoxPredictHuristci.Checked = false;

checkBoxAStarGreadyFirstSearch.Checked = false;

RefrigtzDLL.AllDraw.AStarGreadyFirstSearch = false;

checkBoxBestMovments.Checked = false;

checkBoxOnlySelf.Checked = true;

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

//Maximum Size Form Refregitz Evemnt Handling Operation.

private void FormRefrigtz\_MaximumSizeChanged(object sender, EventArgs e)

{

if (t1.IsAlive)

new Syncronization(t1, 1); ;

if (t2.IsAlive)

new Syncronization(t2, 1); ;

if (t3.IsAlive)

new Syncronization(t3, 1); ;

if (t4.IsAlive)

new Syncronization(t4, 1); ;

System.Threading.Thread.Sleep(10);

if (t1.IsBackground)

new Syncronization(t1, 3);

if (t2.IsBackground)

new Syncronization(t2, 3);

if (t3.IsBackground)

new Syncronization(t3, 3);

if (t4.IsBackground)

new Syncronization(t4, 3);

}

//Leave toll Strips Event Handling Operation.

private void menuStripChessRefrigitz\_Leave(object sender, EventArgs e)

{

if (t1 != null)

t1.Abort();

if (t2 != null)

t2.Abort();

if (t3 != null)

t3.Abort();

if (t4 != null)

t4.Abort();

}

//Puased Click Event Handling

private void buttonPause\_Click(object sender, EventArgs e)

{

if (PaintingPaused)

{

try

{

PaintingPaused = true;

//RunInBackGround();

buttonPauseStart.Text = "Start";

}

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

}

else

{

try

{

//RunInFront();

PaintingPaused = false;

buttonPauseStart.Text = "Stop";

}

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

}

}

//Run In Backgroud Thread Handling Method.

void RunInBackGround()

{

if (t1.IsAlive)

new Syncronization(t1, 1); ;

if (t2.IsAlive)

new Syncronization(t2, 1); ;

if (t3.IsAlive)

new Syncronization(t3, 1); ;

if (t4.IsAlive)

new Syncronization(t4, 1); ;

}

//Run In Front Thread Handling Method.

void RunInFront()

{

if (StateCC && AliceSection && t3.IsBackground)

new Syncronization(t3, 3);

if (StateCC && BobSection && t2.IsBackground)

new Syncronization(t2, 3);

if (((StateCP && !Person) || Blitz) && t1.IsBackground)

new Syncronization(t1, 3);

if (t4.IsAlive && t4.IsBackground)

new Syncronization(t4, 3);

}

private void continueAGameToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

try

{

File.Delete(Root + "\\Run.txt");

}

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

FormKindOfGameContinue R = new FormKindOfGameContinue();

R.ShowDialog();

try

{

File.Delete(Root + "\\DataBase\\CurrentBank.accdb");

}

catch (Exception tt)

{

Log(tt);

}

File.Copy(Root + "\\Database\\Games\\" + R.comboBoxDatabase.Text, Root + "\\Database\\CurrentBank.accdb");

try

{

File.Delete(Root + "\\DataBase\\Games\\" + R.comboBoxDatabase.Text);

}

catch (Exception ttt)

{

Log(ttt);

}

\_1 = R.radioButtonComputerByComputer.Checked;

\_2 = R.radioButtonComputerByPerson.Checked;

\_3 = R.radioButtonGeneticGame.Checked;

\_4 = R.radioButtonBlitz.Checked;

String a = "";

if (\_1) a += 1; else a += 0;

if (\_2) a += 1; else a += 0;

if (\_3) a += 1; else a += 0;

if (\_4) a += 1; else a += 0;

File.WriteAllText(Root + "\\Run.txt", a);

String FolderLocation = Root;

int exitCode = 0;

// Prepare the process to run

ProcessStartInfo start = new ProcessStartInfo();

// Prepare the process to run

// Enter in the command line arguments, everything you would enter after the executable name itself

start.Arguments = "";

// Enter the executable to run, including the complete path

start.FileName = "\"" + FolderLocation + "\\" + "Run.exe" + "\"";

// Do you want to show a console window?

start.WindowStyle = ProcessWindowStyle.Hidden;

start.CreateNoWindow = true;

start.UseShellExecute = true;// Do not 'Run the external process & wait for it to finish'

using (Process proc = Process.Start(start))

{

proc.WaitForExit();

// Retrieve the app's exit code

exitCode = proc.ExitCode;

}

Application.Exit();

}

catch (Exception p)

{

Log(p);

}

}

private void startGameToolStripMenuItem\_Click(object sender, EventArgs e)

{

}

private void progressBarVerify\_CursorChanged(object sender, EventArgs e)

{

if (PreviousTime != null)

{

DateTime Remaining = (new RefregitzReader.RefregitzReader(null)).ConvertRefregitzStringToDateTime(((1 - ((float)progressBarVerify.Value / (float)progressBarVerify.Maximum)) \* (DateTime.Now.Ticks / 1000 - PreviousTime.Ticks / 1000)).ToString());

labelTimesRemaining.Text = (Remaining.Day.ToString() + ":" + Remaining.Minute.ToString() + ":" + Remaining.Second.ToString()) + " Remaining";

labelTimesRemaining.Update();

labelTimesRemaining.Invalidate();

}

else

PreviousTime = DateTime.Now;

Thread.Sleep(100);

}

private void progressBarVerify\_Click(object sender, EventArgs e)

{

progressBarVerify\_CursorChanged(sender, e);

}

private void progressBarVerify\_Validated(object sender, EventArgs e)

{

progressBarVerify\_Click(sender, e);

}

private void toolStripMenuItem6\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

//BobSection = false;

//AliceSection = false;

StateCP = true;

StateCC = false;

StateGe = false;

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount).ToString();

if (OrderPlate == 1)

{

GrayTimer.StartTime();

Blitz = true;

}

else if (OrderPlate == -1)

{

BrownTimer.StartTime();

Blitz = true;

}

else

{

Blitz = false;

}

if (Sec.radioButtonGrayOrder.Checked && OrderPlate == 1)

Person = true;

else

if (Sec.radioButtonBrownOrder.Checked && OrderPlate == -1)

Person = true;

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "You";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "You";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

}

private void pictureBoxRefrigtz\_MouseLeave(object sender, EventArgs e)

{

MouseClicked = false;

}

private void checkBoxIgnoreSelf\_CheckedChanged(object sender, EventArgs e)

{

/\*if (checkBoxIgnoreSelf.Checked)

RefrigtzDLL.ThinkingChess.IInoreSelfObjects = true;

else

RefrigtzDLL.ThinkingChess.IInoreSelfObjects = false;

\*/

}

//AStarGreedy Movments Event Handl;ing Operations Method.

private void checkBoxAStarGreedyMovement\_Enter(object sender, EventArgs e)

{

if (checkBoxAStarGreedyMovement.Checked)

{

RefrigtzDLL.AllDraw.SodierMovments = 4;

RefrigtzDLL.AllDraw.ElefantMovments = 16;

RefrigtzDLL.AllDraw.HourseMovments = 8;

RefrigtzDLL.AllDraw.CastleMovments = 16;

RefrigtzDLL.AllDraw.MinisterMovments = 32;

RefrigtzDLL.AllDraw.KingMovments = 8;

}

else

{

RefrigtzDLL.AllDraw.SodierMovments = 1;

RefrigtzDLL.AllDraw.ElefantMovments = 1;

RefrigtzDLL.AllDraw.HourseMovments = 1;

RefrigtzDLL.AllDraw.CastleMovments = 1;

RefrigtzDLL.AllDraw.MinisterMovments = 1;

RefrigtzDLL.AllDraw.KingMovments = 1;

}

}

//Double Time Checked Box Checked Event Handling.

private void checkBoxUseDoubleTime\_CheckedChanged(object sender, EventArgs e)

{

try

{

//RunInBackGround();

/\*if (RefrigtzDLL.AllDraw.AStarGreadyFirstSearch)

{

checkBoxUseDoubleTime.Visible = true;

if (checkBoxUseDoubleTime.Checked)

{

RefrigtzDLL.AllDraw.UseDoubleTime = true;

}

}

else

{

checkBoxUseDoubleTime.Visible = false;

RefrigtzDLL.AllDraw.UseDoubleTime = false;

}

\*/

//UpdateConfigurationTable();

//RunInFront();

}

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

}

//Penalty Regrad Mechansiam Checed Box Event Handling Operation Method.

private void checkBoxUsePenaltyRegradMechnisam\_CheckedChanged(object sender, EventArgs e)

{

RefrigtzDLL.AllDraw.DrawTable = false;

{

try

{

//RunInBackGround();

if (checkBoxUsePenaltyRegradMechnisam.Checked)

UsePenaltyRegardMechnisam = true;

else

UsePenaltyRegardMechnisam = false;

if (MovmentsNumber == 0)

{

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

//UpdateConfigurationTable();

//RunInFront();

}

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

}

}

//Dynamic Programming AStarGreedy First checked Event Handling.

private void checkBoxDynamicProgrammingAStarGreedyt\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

try

{

/\* //RunInBackGround();

if (checkBoxDynamicProgrammingAStarGreedyt.Checked)

RefrigtzDLL.AllDraw.DynamicAStarGreedytPrograming = true;

else

RefrigtzDLL.AllDraw.DynamicAStarGreedytPrograming = false;

//UpdateConfigurationTable();

//RunInFront();

\*/

}

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

}

}

//Help Event Handling Show Method.

private void toolStripMenuItem3\_Click\_1(object sender, EventArgs e)

{

Help.ShowHelp(this, "Help.chm");

}

object sender;

EventArgs e;

void Veryfi()

{

int Max = 0;

do

{

Max++;

} while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + Max.ToString() + ".accdb"));

int iii = 0;

do

{

progressBarVerify\_Validated(sender, e);

progressBarVerify.Value = (int)(((double)iii / (double)Max) \* 100.0);

iii++;

String FileName = Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb";

//Read Last Table and Set MovementNumber

MovmentsNumber = 0;

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

{

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

{

RefrigtzDLL.AllDraw.TableVeryfy[i, j] = RefrigtzDLL.AllDraw.TableVeryfyConst[i, j];

}

}

try

{

VerifyTable(FileName, 0, ref MovmentsNumber);

}

catch (Exception t)

{ Log(t); }

} while (System.IO.File.Exists(Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"));

//MovmentsNumber = 0;

}

//Verify tool Srtip Games Folder Databases Event Handling Operation.

private void verifyToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.sender = sender;

this.e = e;

//Thread t = new Thread(new ThreadStart(Veryfi));

// t.Start();

}

private void clearToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (File.Exists(Root + "ErrorProgramRun.txt") || File.Exists(Root + "\\Run.txt"))

{

this.SetBoxText("Clearing...\n\n\n");

if (File.Exists(Root + "ErrorProgramRun.txt"))

File.Delete(Root + "\\ErrorProgramRun.txt");

if (File.Exists(Root + "\\Run.txt"))

File.Delete(Root + "\\Run.txt");

this.SetBoxText("Finished.");

}

}

private void checkBoxPredictHuristci\_CheckedChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

//RunInBackGround();

/\* if (checkBoxPredictHuristci.Checked)

RefrigtzDLL.ThinkingChess.PredictHuristic = true;

else

RefrigtzDLL.ThinkingChess.PredictHuristic = false;

\*/

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxMaxTree\_SelectedIndexChanged(object sender, EventArgs e)

{

comboBoxMaxLevelText = comboBoxMaxLevel.Text;

//if (!DisableTemporarlyTimerUpdate)

{

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void repairToolStripMenuItem\_Click(object sender, EventArgs e)

{

int iii = 1, Max = 1;

int t = DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond;

do

{

if (System.IO.File.Exists(FormRefrigtz.Root + "\\Database\\Games\\CurrentBank" + Max.ToString() + ".accdb"))

Max++;

if (DateTime.Now.Second \* 1000 + DateTime.Now.Millisecond - t > 2000) break;

} while (true);

progressBarVerify.Maximum = Max;

do

{

iii++;

try

{

progressBarVerify.Value = iii;

progressBarVerify.Update();

}

catch (Exception t6)

{

Log(t6);

progressBarVerify.Value = Max;

progressBarVerify.Update();

}

if (System.IO.File.Exists(FormRefrigtz.Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb"))

{

try

{

File.Copy(FormRefrigtz.Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb", Root + "\\CurrentBank.accdb");

}

catch (Exception t1)

{

Log(t1);

}

try

{

File.Delete(FormRefrigtz.Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

}

catch (Exception t2)

{

Log(t2);

}

try

{

CreateConfigurationTable();

}

catch (Exception t3)

{

Log(t3);

}

try

{

File.Copy(Root + "\\CurrentBank.accdb", FormRefrigtz.Root + "\\Database\\Games\\CurrentBank" + iii.ToString() + ".accdb");

}

catch (Exception t4)

{

Log(t4);

}

try

{

File.Delete(Root + "\\CurrentBank.accdb");

}

catch (Exception t5)

{

Log(t5);

}

}

} while (iii < Max);

}

private void comboBoxAttack\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignAttack = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxObjectDangour\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignObjectDangour = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxReducedAttacked\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignReducedAttacked = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxSupport\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignSupport = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxHitting\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignKiller = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void comboBoxMovments\_SelectedIndexChanged(object sender, EventArgs e)

{

//if (!DisableTemporarlyTimerUpdate)

{

RefrigtzDLL.AllDraw.SignMovments = System.Convert.ToInt32(((ComboBox)(sender)).Text);

//RunInBackGround();

//UpdateConfigurationTable();

//RunInFront();

}

}

private void errorOpenToolStripMenuItem\_Click(object sender, EventArgs e)

{

ErrorTrueMonitorFalse = true;

(new FormTXT()).Show();

}

private void monitorOpenToolStripMenuItem\_Click(object sender, EventArgs e)

{

ErrorTrueMonitorFalse = false;

(new FormTXT()).Show();

}

private void toolStripMenuItem7\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

folderBrowserDialogBackup.ShowDialog();

File.Copy(FormRefrigtz.Root + "\\Database\\CurrentBank.accdb", folderBrowserDialogBackup.SelectedPath + "\\CurrentBank.accdb");

File.Copy(FormRefrigtz.Root + "\\Database\\Monitor.html", folderBrowserDialogBackup.SelectedPath + "\\Monitor.html");

MessageBox.Show("Backup Finished.");

}

catch (Exception t)

{

Log(t);

MessageBox.Show(t.ToString());

}

}

private void toolStripMenuItem7\_Click\_1(object sender, EventArgs e)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

BobSection = false;

AliceSection = false;

StateCP = true;

StateCC = false;

StateGe = false;

comboBoxMaxLevel.Text = "2";

if (OrderPlate == 1 && Sec.radioButtonGrayOrder.Checked)

{

Person = true;

StateCP = true;

GrayTimer.StartTime();

}

else if (Sec.radioButtonBrownOrder.Checked && OrderPlate == -1)

{

Person = false;

StateCP = true;

BrownTimer.StartTime();

}

else

{

StateCP = true;

Person = false;

}

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

}

private void pictureBoxRefrigtz\_Click(object sender, EventArgs e)

{

}

private void toolStripMenuItem8\_Click(object sender, EventArgs e)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

StateCC = true;

if (OrderPlate == 1)

{

BobSection = true;

AliceSection = false;

GrayTimer.StartTime();

}

else

{

BobSection = false;

AliceSection = true;

BrownTimer.StartTime();

}

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

}

private void buttonChangeArrangment\_Click(object sender, EventArgs e)

{

RefrigtzDLL.AllDraw.DrawTable = false;

/\*if (MovmentsNumber == 0)

{

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

{

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

{

Table[i, j] \*= -1;

}

}

RefrigtzDLL.AllDraw.TableListAction.Clear();

RefrigtzDLL.AllDraw.TableListAction.Add(Table);

ArrangmentsChanged = !ArrangmentsChanged;

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

UpdateConfigurationTableVal = true;

//UpdateConfigurationTable();

try

{

UpdateTable(Table, "Table00000000");

}

catch (Exception t)

{

Log(t);

InsertTableAtDataBase(Table);

}

}

SetPrictureBoxRefregitzUpdate(pictureBoxTimerBrown);

SetPrictureBoxRefregitzInvalidate(pictureBoxTimerBrown);

\*/

//DisableTemporarlyTimerUpdate = false;

}

private void pictureBoxTimerGray\_Paint(object sender, PaintEventArgs e)

{

Object o = new Object();

lock (o)

{

//UpadatTimer();

if (GrayTimer.TextChanged)

{

try

{

Image TimerImageGray = (Image)new Bitmap(pictureBoxTimerGray.Width, pictureBoxTimerGray.Height);

g1 = Graphics.FromImage(TimerImageGray);

g1.FillRectangle(new SolidBrush(Color.Gray), new Rectangle(new Point(0, 0), new Size(pictureBoxTimerGray.Width, pictureBoxTimerGray.Height)));

g1.DrawString(GrayTimer.ReturnTime(), new Font("Times New Roman", 30), new SolidBrush(Color.Black), new PointF(5, 5));

pictureBoxTimerGray.Image = TimerImageGray;

g1.Dispose();

//Thread.Sleep(20);

}

catch (Exception t)

{

//pictureBoxTimerGray.Update();

// pictureBoxTimerGray.Invalidate();

//RunInFront();

Log(t);

}

}

System.Threading.Thread.Sleep(1000);

pictureBoxTimerBrown.Invalidate();

pictureBoxTimerBrown.Update();

}

}

void UpadatTimer()

{

//Thread.Sleep(1000);

//while (DisableTemporarlyTimerUpdate) ;

{

try

{

// while (!TimersSet) ;

UpdateConfigurationTable();

}

catch (Exception t)

{

Log(t);

}

}

}

private void pictureBoxTimerBrown\_Paint(object sender, PaintEventArgs e)

{

Object o = new Object();

lock (o)

{

//UpadatTimer();

if (BrownTimer.TextChanged)

{

try

{

Image TimerImageBrown = (Image)new Bitmap(pictureBoxTimerBrown.Width, pictureBoxTimerBrown.Height);

g2 = Graphics.FromImage(TimerImageBrown);

g2.FillRectangle(new SolidBrush(Color.Brown), new Rectangle(new Point(0, 0), new Size(pictureBoxTimerBrown.Width, pictureBoxTimerBrown.Height)));

g2.DrawString(BrownTimer.ReturnTime(), new Font("Times New Roman", 30), new SolidBrush(Color.Black), new PointF(5, 5));

pictureBoxTimerBrown.Image = TimerImageBrown;

g2.Dispose();

//Thread.Sleep(20);

}

catch (Exception t)

{

Log(t);

//pictureBoxTimerBrown.Update();

//pictureBoxTimerBrown.Invalidate();

//RunInFront();

}

}

System.Threading.Thread.Sleep(1000);

pictureBoxRefrigtz.Invalidate();

pictureBoxRefrigtz.Update();

}

}

private void checkBoxBestMovments\_CheckedChanged(object sender, EventArgs e)

{

}

private void checkBoxOnlySelf\_CheckedChanged(object sender, EventArgs e)

{

}

private void checkBoxAStarGreedyMovement\_CheckedChanged(object sender, EventArgs e)

{

}

private void toolStripMenuItem9\_Click(object sender, EventArgs e)

{

ProfesionalWithComputer = true;

comboBoxMaxLevel.Text = "1";

comboBoxMaxLevel.Enabled = false;

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

BobSection = false;

AliceSection = false;

StateCP = false;

StateCC = false;

StateGe = false;

if (OrderPlate == 1)

{

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

BrownTimer.StartTime();

}

else

{

Blitz = false;

}

Blitz = true;

UpdateConfigurationTableVal = true;

}

private void toolStripMenuItem10\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}//AStarGreedyHuristic = false;

if (OrderPlate == 1)

{

FullGame = true;

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

FullGame = true;

BrownTimer.StartTime();

}

//UsePenaltyRegardMechnisam = false;

PredictHuristic = false;

TimerIniataite = true;

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "Refregitz";

label2.Text = "stockfish-8";

}

else

{

label2.Text = "Refregitz";

label1.Text = "stockfish-8";

}

Stockfish = true;

FullGame = true;

Blitz = false;

StateCC = true;

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

//BobSection = true;

//RefregitzisCurrent = false;

}

private void toolStripMenuItem1\_Click(object sender, EventArgs e)

{

}

private void toolStripMenuItem11\_Click(object sender, EventArgs e)

{

UsePenaltyRegardMechnisam = false;

UpdateConfigurationTable();

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

BobSection = false;

AliceSection = false;

StateCP = false;

StateCC = false;

StateGe = false;

comboBoxMaxLevel.Text = "4";

if (OrderPlate == 1)

{

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

BrownTimer.StartTime();

}

else

{

Blitz = false;

}

Blitz = true;

UpdateConfigurationTableVal = true;

}

private void toolStripMenuItem12\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

//BobSection = false;

//AliceSection = false;

StateCP = true;

StateCC = false;

StateGe = false;

FullGame = false;

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount \* 2 + 1).ToString();

if (OrderPlate == 1)

{

Blitz = true;

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

Blitz = true;

BrownTimer.StartTime();

}

else

{

Blitz = false;

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "You";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "You";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

}

private void toolStripMenuItem13\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

//BobSection = false;

//AliceSection = false;

StateCP = true;

StateCC = false;

StateGe = false;

Blitz = false;

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount \* 2 + 1).ToString();

if (OrderPlate == 1)

{

FullGame = true;

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

FullGame = true;

BrownTimer.StartTime();

}

else

{

FullGame = false;

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "You";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "You";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

}

private void opneToolStripMenuItem\_Click(object sender, EventArgs e)

{

}

private void pictureBoxTimerBrown\_Click(object sender, EventArgs e)

{

}

private void pictureBoxTimerGray\_Click(object sender, EventArgs e)

{

}

private void FormRefrigtz\_FormClosing(object sender, FormClosingEventArgs e)

{

(new TakeRoot()).Save(this, ref LoadTree, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

}

private void toolStripMenuItem14\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

//BobSection = false;

//AliceSection = false;

StateCP = true;

StateCC = false;

StateGe = false;

Blitz = false;

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount).ToString();

if (OrderPlate == 1)

{

FullGame = true;

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

FullGame = true;

BrownTimer.StartTime();

}

else

{

FullGame = false;

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "You";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "You";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

}

private void buttonCalculateRootGray\_Click(object sender, EventArgs e)

{

}

private void pictureBox3\_Click(object sender, EventArgs e)

{

}

private void toolStripMenuItem15\_Click(object sender, EventArgs e)

{

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

StateCP = false;

StateGe = false;

Blitz = false;

StateCC = true;

FullGame = true;

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

comboBoxMaxLevel.Text = (PlatformHelper.ProcessorCount \* 2 + 1).ToString();

if (OrderPlate == 1)

{

//BobSection = true;

//AliceSection = false;

GrayTimer.StartTime();

}

else

{

//BobSection = false;

//AliceSection = true;

BrownTimer.StartTime();

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "Refregiz";

label2.Text = "Refregiz";

}

else

{

label1.Text = "Refregiz";

label2.Text = "Refregiz";

}

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

UpdateConfigurationTableVal = true;

}

private void toolStripMenuItem16\_Click(object sender, EventArgs e)

{

//UsePenaltyRegardMechnisam = false;

TimerIniataite = true;

label1.Text = "Refregitz";

label2.Text = "stockfish-8";

Stockfish = true;

Blitz = true;

StateCP = true;

//BobSection = true;

}

private void toolStripMenuItem17\_Click(object sender, EventArgs e)

{

UsePenaltyRegardMechnisam = true;

checkBoxUsePenaltyRegradMechnisam.Checked = true;

TimerIniataite = true;

label1.Text = "You";

label2.Text = "stockfish-8";

Stockfish = true;

Person = true;

StateCP = true;

//BobSection = true;

PersonWithStockfish = true;

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

private void toolStripMenuItem18\_Click(object sender, EventArgs e)

{

//AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

PredictHuristic = false;

TimerIniataite = true;

label1.Text = "Refregitz";

label2.Text = "stockfish-8";

Stockfish = true;

FullGame = true;

Blitz = false;

StateCC = true;

//BobSection = true;

UsePenaltyRegardMechnisam = false;

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

}

private void toolStripMenuItem19\_Click(object sender, EventArgs e)

{

UsePenaltyRegardMechnisam = false;

AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

PredictHuristic = false;

TimerIniataite = true;

if (OrderPlate == 1)

{

Blitz = true;

GrayTimer.StartTime();

}

else if (OrderPlate == -1)

{

Blitz = true;

BrownTimer.StartTime();

}

if (Sec.radioButtonGrayOrder.Checked)

{

label1.Text = "Refregitz";

label2.Text = "stockfish-8";

}

else

{

label2.Text = "Refregitz";

label1.Text = "stockfish-8";

}

Stockfish = true;

Blitz = true;

StateCC = true;

StateCP = false;

StateGe = false;

BobSection = true;

if (!LoadTree)

{

RefrigtzDLL.AllDraw THIS = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

bool FOUND = false;

SetDrawFounding(ref FOUND, ref THIS, false);

}

UpdateConfigurationTableVal = true;

UpdateConfigurationTable();

}

private void toolStripMenuItem20\_Click(object sender, EventArgs e)

{

//AStarGreedyHuristic = false;

checkBoxAStarGreedyHuristic.Checked = false;

UsePenaltyRegardMechnisam = false;

checkBoxUsePenaltyRegradMechnisam.Checked = false;

if (MovmentsNumber == 0)

{

GrayTimer = new Refrigtz.Timer(true);

BrownTimer = new Refrigtz.Timer(true);

GrayTimer.TimerInitiate();

BrownTimer.TimerInitiate();

}

PredictHuristic = false;

TimerIniataite = true;

label1.Text = "Refregitz";

label2.Text = "stockfish-8";

Stockfish = true;

Blitz = true;

StateCC = true;

//BobSection = true;

UsePenaltyRegardMechnisam = false;

if (!AllDrawLoad)

{

Draw = new RefrigtzDLL.AllDraw(OrderPlate, MovementsAStarGreedyHuristicFound, IInoreSelfObjects, UsePenaltyRegardMechnisam, BestMovments, PredictHuristic, OnlySelf, AStarGreedyHuristic, ArrangmentsChanged);

Draw.TableList.Clear();

Draw.TableList.Add(Table);

Draw.SetRowColumn(0);

}

}

}

}

//End of Documentation.