```
2c2
< // Swift source code of main.swift Version 9.0.4 at greenchess.com //
---
> // Swift source code of main.swift Version 9.0.3 at greenchess.com //
14,22d13
<    // the castle selection //
<    var CastSel:String = ""
<
<    // the context to write the movie info frame into //
<    var Context:CGContext!
<
<    // the main view image //
<    var MainViewImage:CGImage!
<
35a27,29
>    // the current illustration index //
>    var IllustrationIndex:Int = 0
>
40c34
<    @IBOutlet var GameExplorerImage:BoardImageView!
---
>    @IBOutlet var GameExplorerImage:GameExplorerView!
43c37
<    @IBOutlet var DatabaseExplorerImage:BoardImageView!
---
>    @IBOutlet var DatabaseExplorerImage:GameExplorerView!
61,63c55,58
<    let GameInfoDict:[Int:String] = [1:"Checkmate and Black wins", 2:"Checkmate and White wins",
<                                     3:"Stalemate", 4:"Draw detected by threefold repetition", 7:
<                                     "Draw detected by insufficient material for mate"]
---
>    let GameInfoDictionary:[Int:String] = [1:"Checkmate and Black wins", 2:
>                                           "Checkmate and White wins", 3:"Stalemate", 4:
>                                           "Draw detected by threefold repetition", 7:
>                                           "Draw detected by insufficient material for mate"]
66,70c61,65
<    let TimedGameDict:[Int:String] = [1:"Game Over and Black Wins", 2:"Game Over and White Wins",
<                                      3:"Game Over and Stalemate",
<                                      4:"Game Over by Draw from Threefold Repetition",
<                                      5:"Black Wins on Time", 6:"White Wins on Time", 7:
<                                      "Game Over by Draw from Insufficient Material for Mate"]
---
>    let TimedGameDictionary:[Int:String] = [1:"Game Over and Black Wins", 2:
>                                            "Game Over and White Wins", 3:"Game Over and Stalemate",
>                                            4:"Game Over by Draw from Threefold Repetition",
>                                            5:"Black Wins on Time", 6:"White Wins on Time", 7:
>                                            "Game Over by Draw from Insufficient Material for Mate"]
103c98
<    var MenuItemsDict:Int = 0
---
>    var MenuItemsDictionary:Int = 0
104a100,102
>    // the current board and piece set //
>    var Set:Int = 0
>
114,116d111
<    // the stockfish engine timer //
<    var StockfishTimer:Timer = Timer()
<
122a118,120
>    // the cg image data //
>    var CGImageData:CFData? = nil
>
140a139,141
>    // the position list index //
>    var PositionListIndex:Int = 0
>
172a174,176
>
>    // the movie frame index //
>    var FrameIndex:Int = 0
206a211,225
>    // an index in a list //
>    var Index:Int = 0
>
>    // the annotation glyph index //
>    var GlyphIndex:Int = 0
>
>    // an index in a list //
>    var OpeningBookIndex:Int = 0
>
>    // an index in a list //
>    var BookIndex:Int = 0
>
>    // an index in a list //
>    var MoveListCountIndex:Int = 0
>
238c257
<    var GameState:Int = 0
---
>    var CurrentGameState:Int = 0
244,245c263,264
<    var PositionImages:[CGImage] = []
<
---
>    var SavedPositionImages:[NSImage] = []
299a319,324
>    // the buttons to be displayed in the alert window //
>    var AlertButtons:[String] = []
>
>    // the components of a filename //
>    var FilenameComponents:[String] = []
>
325a351,356
>
>    // the engine filename document type extension //
>    var FilenameExtension:String = ""
>
>    // the name of a game file to open //
>    var GameFilename:String = ""
478c509
<    @IBOutlet var WhiteKingsideCastle:NSToolbarItem!
---
>    @IBOutlet var WhiteKingsideCastleToolbarButton:NSToolbarItem!
481c512
<    @IBOutlet var BlackKingsideCastle:NSToolbarItem!
---
>    @IBOutlet var BlackKingsideCastleToolbarButton:NSToolbarItem!
484c515
<    @IBOutlet var WhiteQueensideCastle:NSToolbarItem!
---
>    @IBOutlet var WhiteQueensideCastleToolbarButton:NSToolbarItem!
487c518
<    @IBOutlet var BlackQueensideCastle:NSToolbarItem!
---
>    @IBOutlet var BlackQueensideCastleToolbarButton:NSToolbarItem!
634c665,666
<    let FENCharacters:[String] = ["k", "q", "r", "b", "n", "p", "", "P", "N", "B", "R", "Q", "K"]
---
>    let FENLayoutCharacters:[String] = ["k", "q", "r", "b", "n", "p", "", "P", "N", "B", "R", "Q",
>                                        "K"]
639a672,677
>    // list of possible kingside castles in pgn //
>    let KCastles:[String] = ["O-O", "O-O+", "O-O#"]
>
>    // list of possible queenside castles in pgn //
>    let QCastles:[String] = ["O-O-O", "O-O-O+", "O-O-O#"]
>
659,663c697,701
<    let AboutDict:[String:String] = ["greenchess":"greenchess.com", "support":
<                                     "greenchess.com/contact", "forum":"greenchess.com/forum",
<                                     "stockfish":"stockfishchess.org/download", "privacy":
<                                     "greenchess.com/privacy", "license":
<                                     "gnu.org/licenses/gpl-3.0.en.html"]
---
>    let AboutDictionary:[String:String] = ["greenchess":"greenchess.com", "support":
>                                           "greenchess.com/contact", "forum":"greenchess.com/forum",
>                                           "stockfish":"stockfishchess.org/download",
>                                           "privacy":"greenchess.com/privacy", "license":
>                                           "gnu.org/licenses/gpl-3.0.en.html"]
686c724
<        StartNewGame()
---
>        StartNewGame(true)
689c727
<        if Inf.Vals("AutosaveGame").VB {ExtractFromGameData(GameData, Tasks:"All")}
---
>        if Inf.Vals("AutosaveGame").VB {ExtractFromGameData(true)}
723c761
<    func SetWindowBackgroundColor()
---
>    func SetWindowBackgroundColor(_ Refresh:Bool)
743a782,784
>
>        // refresh the board and pieces //
>        if Refresh {RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)}
746,748d786
<    // update the window background color and refresh the board and tables //
<    func UpdateWindowBackgroundColor() {SetWindowBackgroundColor(); Refresh("BoardTables")}
<
782c820
<    // the window did enter fullscreen //
---
>    // if the window did enter fullscreen //
807,812c845,850
<        // return if the dictionary item exists //
<        if AboutDict[AboutToolbar!.selectedItemIdentifier!.rawValue] == nil {return}
<
<        // open the corresponding webpage //
<        NSWorkspace.shared.open(URL(string:"https://" + AboutDict[AboutToolbar!
<                                .selectedItemIdentifier!.rawValue]!)!)
---
>        // if the dictionary item exists //
>        if AboutDictionary[AboutToolbar!.selectedItemIdentifier!.rawValue] != nil
>        {
>            // open the corresponding webpage //
>            NSWorkspace.shared.open(URL(string:"https://" + AboutDictionary[AboutToolbar!
>                                    .selectedItemIdentifier!.rawValue]!)!)
814,815c852,854
<        // deselect the selected toolbar item //
<        AboutToolbar.selectedItemIdentifier = nil
---
>            // deselect the selected toolbar item //
>            AboutToolbar.selectedItemIdentifier = nil
>        }
828,829c867,886
<         let Cell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"PGN"),
<                                  owner:self) as! NSTableCellView
---
>         let TableCell = View.makeView(withIdentifier:
>                                       NSUserInterfaceItemIdentifier(rawValue:"PGN"), owner:self)
>                                       as! NSTableCellView
>
>         // if hiding the illustrations //
>         if Inf.Vals("HideIllustrations").VB
>         {
>             // assign a blank image to the table cell //
>             TableCell.imageView!.image = NSImage()
>         }
>         // will show the illustrations //
>         else
>         {
>             // process the illustration for the specified row //
>             ProcessIllustration(Row + 1, Set:0)
>
>             // set the table cell image to the illustration //
>             TableCell.imageView!.image = BPComp.GetNSImage(Inf.Vals("IllustrationSize").VI,
>                                                            Set:0)
>         }
831,835c888,889
<         // update the table cell image and text field //
<         (Cell.imageView!.image, Cell.textField!.attributedStringValue) = (Inf
<         .Vals("HideIllustrations").VB ? NSImage() : FrameNSImage(Row + 1, Size:CGFloat(Inf
<         .Vals("IllustrationSize").VI), UpdateBoard:false, Table:true), PGNMovesData
<         .GetText(Row))
---
>         // assign the attributed string to the table cell text field //
>         TableCell.textField!.attributedStringValue = PGNMovesData.GetText(Row)
838c892
<         return Cell
---
>         return TableCell
844,845c898,899
<         let Cell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"Info"),
<                                  owner:self) as! NSTableCellView
---
>         let TableCell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:
>                                       "Info"), owner:self) as! NSTableCellView
848c902
<         Cell.textField!.attributedStringValue = GameInfoData.GameInfoList[Row]
---
>         TableCell.textField!.attributedStringValue = GameInfoData.GameInfoList[Row]
851c905
<         return Cell
---
>         return TableCell
857,858c911,912
<         let Cell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"Games"),
<                                  owner:self) as! NSTableCellView
---
>         let TableCell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:
>                                       "Games"), owner:self) as! NSTableCellView
860,863c914,927
<         // return the attributed string at the row index in the game list //
<         Cell.textField!.attributedStringValue = (GameListData.SelectedRow == Row) ? GameListData
<                                                 .SelectedRow(Row,GameListFont:GameListFont) :
<                                                 GameListData.GetRow(Row)
---
>         // if the selected row is equal to the row index make font color white //
>         if GameListData.SelectedRow == Row
>         {
>             // return the attributed string at the row index in the game list with white font //
>             TableCell.textField!.attributedStringValue = GameListData.SelectedRow(Row,
>                                                          GameListFont:GameListFont)
>         }
>         // the selected row is not equal to the row index so make chosen font color //
>         else
>         {
>             // return the attributed string at the row index in the game list //
>             TableCell.textField!.attributedStringValue = GameListData.GetRow(Row)
>         }
866c930
<         return Cell
---
>         return TableCell
876,879c940,954
<     // return the table view dynamic height //
<     return (TableView == PGNMovesTableView) ? min(PGNMovesData.RowHeight(Row), 2000) : ((TableView
<            == GameInfoTableView) ? GameInfoData.RowHeight(Row) : ((TableView == GameListTableView)
<            ? GameListData.RowHeight(Row) : 30))
---
>     // switch over the table view to return the height //
>     switch TableView
>     {
>         // if is the pgn moves table view //
>         case PGNMovesTableView : return min(PGNMovesData.RowHeight(Row), 2000)
>
>         // if is the game info table view //
>         case GameInfoTableView : return GameInfoData.RowHeight(Row)
>
>         // if is the game list table view //
>         case GameListTableView : return GameListData.RowHeight(Row)
>
>         // the default case //
>         default : return 30
>     }
922,926c997,1001
<     // if the window is not currently open //
<     if (HideMenuItem.title == "Reopen") {Inf.TurnOnVB("VB", Key:"MainWindowHidden"); return}
<
<     // make the main window visible //
<     MainWindow.makeKeyAndOrderFront(nil)
---
>     // if the main window is open //
>     if Open
>     {
>         // make the main window visible //
>         MainWindow.makeKeyAndOrderFront(nil)
928,929c1003,1013
<     // set the hide menu item title to hide //
<     HideMenuItem.title = "Hide"
---
>         // set the hide menu item title to hide //
>         HideMenuItem.title = "Hide"
>
>         // set that the main window is not hidden //
>         Inf.TurnOffVB("VB", Key:"MainWindowHidden")
>     }
>     // the main window is closed //
>     else
>     {
>         // set the hide menu item title to reopen //
>         HideMenuItem.title = "Reopen"
931,932c1015,1017
<     // set that the main window is not hidden //
<     Inf.TurnOffVB("VB", Key:"MainWindowHidden")
---
>         // set that the main window is hidden //
>         Inf.TurnOnVB("VB", Key:"MainWindowHidden")
>     }
967,972c1052,1069
<     // return false if the filename doesn't have a document extension type //
<     if ((Filename.components(separatedBy:".")).filter {$0 != ""}).count < 2 {return false}
<
<     // return false if not a pgn file type //
<     if (((Filename.components(separatedBy:".")).filter {$0 != ""}).last!).lowercased() != "pgn"
---
>     // assign the game filename //
>     GameFilename = Filename
>
>     // convert the game filename into its components //
>     FilenameComponents = GameFilename.components(separatedBy:".")
>
>     // if the filename has a document file type extension //
>     if FilenameComponents.count >= 2
>     {
>         // get the filename extension type //
>         FilenameExtension = FilenameComponents[FilenameComponents.count - 1]
>
>         // if the file is not a pgn file, return //
>         if FilenameExtension.lowercased() != "pgn" {return false}
>     }
>     // filename doesn't have a document extension type //
>     else
>     {
974c1071
<     // return false //
---
>         // return that can't open this type of file format //
979c1076
<     OpenGame(Filename)
---
>     OpenGame()
1004c1101,1111
<         if (Inf.Vals("MoveExists").VB) {MovingPiece.Reset(); Refresh("All"); return}
---
>         if (Inf.Vals("MoveExists").VB)
>         {
>             // set the moving piece to none //
>             MovingPiece.Reset()
>
>             // refresh the board and pieces //
>             RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
>
>             // exit animation since move does not exist //
>             return
>         }
1011c1118,1119
<         MovingPiece.PrepareAnimation(BPData.Start, End:BPData.End, Rate:Inf.Vals("AnimationRate").VI)
---
>         MovingPiece.PrepareAnimation(BPData.StartSquare, AnimEnd:BPData.EndSquare, AnimationRate:
>                                      Inf.Vals("AnimationRate").VI)
1013,1014c1121,1122
<     // refresh the view //
<     Refresh("View")
---
>     // refresh the board and pieces //
>     RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
1031,1033c1139,1144
<     // if move is illegal, reset to legal and replay the moves //
<     if MovingPiece.Illegal() {MovingPiece.ResetIllegal(); Refresh("All"); return}
---
>     // if undoing an illegal move //
>     if MovingPiece.Illegal()
>     {
>         // set that no longer undoing an illegal move //
>         MovingPiece.ResetIllegal()
1035,1036c1146,1153
<     // update the illustration data //
<     UpdateIllustrationData(Attributes.Inf.GetCount())
---
>         // replay the moves //
>         ReplayMoves(true)
>     }
>     // doing a legal move //
>     else
>     {
>         // perform castling and enpassant move if had one //
>         PerformCastleEnpassantIfNeeded()
1038,1039c1155,1159
<     // refresh the view and tables //
<     Refresh("ViewTables")
---
>         // update the illustration data //
>         UpdateIllustrationData(Attributes.Inf.GetCount())
>
>         // refresh the board and pieces //
>         RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
1041,1042c1161,1162
<     // set that the chess engine is done thinking //
<     Inf.TurnOffVB("VB", Key:"EngineThinking")
---
>         // set that the chess engine is done thinking //
>         Inf.TurnOffVB("VB", Key:"EngineThinking")
1044,1046c1164,1170
<     // set the go stop menu item title to engine //
<     GoStopMenuItem.title = "Engine"
---
>         // set the go stop menu item title to engine //
>         GoStopMenuItem.title = "Engine"
>
>         // print the game information //
>         PrintGameInfo(true)
>     }
1075a1200,1202
>     // play the game sound of a click //
>     GameSounds.Play("click")
>
1077c1204
<     TogglePref("HideGameClockInPanel")
---
>     Inf.ToggleVB("VBPref", Key:"HideGameClockInPanel")
1089,1091c1216,1220
<     // update the time clock hidden states //
<     (WhiteTimeClock.isHidden, BlackTimeClock.isHidden) = (Inf.Vals("HideGameClockInPanel").VB,
<     Inf.Vals("HideGameClockInPanel").VB)
---
>     // update the white time clock hidden state //
>     WhiteTimeClock.isHidden = Inf.Vals("HideGameClockInPanel").VB
>
>     // update the black time clock hidden state //
>     BlackTimeClock.isHidden = Inf.Vals("HideGameClockInPanel").VB
1135,1140c1264,1271
<     // update the window size menu item states and other variables//
<     (HideInfoMenuItem.state, LargeInfoMenuItem.state, SmallInfoMenuItem.state, HideClock,
<     InFullScreen, Frame) = (Ctr.UpdatedState(Inf.Vals("HideInfoPanel").VB), Ctr.UpdatedState(Inf
<     .Vals("LargeInfoPanel").VB), Ctr.UpdatedState(Inf.Vals("SmallInfoPanel").VB), BoolTo().CG(Inf
<     .Vals("HideGameClockInPanel").VB), BoolTo().CG(Inf.Vals("FullScreen").VB), CGRect(x:0, y:0,
<     width:660, height:660))
---
>     // update the hide info menu item state //
>     HideInfoMenuItem.state = Ctr.UpdatedState(Inf.Vals("HideInfoPanel").VB)
>
>     // update the large info menu item state //
>     LargeInfoMenuItem.state = Ctr.UpdatedState(Inf.Vals("LargeInfoPanel").VB)
>
>     // update the small info menu item state //
>     SmallInfoMenuItem.state = Ctr.UpdatedState(Inf.Vals("SmallInfoPanel").VB)
1141a1273,1278
>     // get the current hide game clock in panel value //
>     HideClock = BoolTo().CG(Inf.Vals("HideGameClockInPanel").VB)
>
>     // if in fullscreen mode or not //
>     InFullScreen = BoolTo().CG(Inf.Vals("FullScreen").VB)
>
1145c1282,1285
<
---
>
>
>     // set the main board frame //
>     Frame = CGRect(x:0, y:0, width:660, height:660)
1149,1156c1289,1305
<         // update the frames for the small info panel //
<         (WhiteTimeClock.frame, BlackTimeClock.frame, PGNMovesTableView.enclosingScrollView!.frame,
<         GameInfoTableView.enclosingScrollView!.frame) = (CGRect(x:705, y:280, width:100, height:
<         100), CGRect(x:830, y:280, width:100, height:100), CGRect(x:675, y:15, width:290, height:
<         250 + HideClock * 60), CGRect(x:675, y:395 - HideClock * 60, width:290, height:250 +
<         HideClock * 60))
---
>         // update the white time clock frame //
>         WhiteTimeClock.frame = CGRect(x:705, y:280, width:100, height:100)
>
>         // update the black time clock frame //
>         BlackTimeClock.frame = CGRect(x:830, y:280, width:100, height:100)
>
>         // update the pgn moves table view frame //
>         PGNMovesTableView.enclosingScrollView?.frame = CGRect(x:675, y:15, width:290, height:250 +
>                                                               HideClock * 60)
>
>         // update the game info table view frame //
>         GameInfoTableView.enclosingScrollView?.frame = CGRect(x:675, y:395 - HideClock * 60,
>                                                               width:290, height:250 + HideClock *
>                                                               60)
1159,1164c1307,1319
<         // update the frames for the large info panel //
<         (WhiteTimeClock.frame, BlackTimeClock.frame, PGNMovesTableView.enclosingScrollView!.frame,
<         GameInfoTableView.enclosingScrollView!.frame) = (CGRect(x:1005, y:545 + InFullScreen * 35,
<         width:100, height:100), CGRect(x:1130, y:545 + InFullScreen * 35, width:100, height:100),
<         CGRect(x:675, y:15, width:290, height:630 + InFullScreen * 35), CGRect(x:975, y:15, width:
<         290, height:515 + HideClock * 115 + InFullScreen * 35))
---
>         // update the white time clock frame //
>         WhiteTimeClock.frame = CGRect(x:1005, y:545 + InFullScreen * 35, width:100, height:100)
>
>         // update the black time clock frame //
>         BlackTimeClock.frame = CGRect(x:1130, y:545 + InFullScreen * 35, width:100, height:100)
>
>         // update the pgn moves table view frame //
>         PGNMovesTableView.enclosingScrollView?.frame = CGRect(x:675, y:15, width:290, height:630 +
>                                                               InFullScreen * 35)
>
>         // update the game info table view frame //
>         GameInfoTableView.enclosingScrollView?.frame = CGRect(x:975, y:15, width:290, height:515 +
>                                                               HideClock * 115 + InFullScreen * 35)
1242c1397
<             // if in game gallery window, review the gallery //
---
>             // if the game gallery window is visible, review the gallery //
1246c1401,1404
<         case 53 : if Inf.Vals("FullScreen").VB {MainWindow.toggleFullScreen(nil)}
---
>         case 53 :
>
>             // if in fullscreen mode, toggle the main window out of fullscreen mode //
>             if Inf.Vals("FullScreen").VB {MainWindow.toggleFullScreen(nil)}
1259,1263c1417,1418
<     // exit if the edit menu doesn't exist //
<     if EditMenu == nil {return}
<
<     // loop over the edit menu items to remove //
<     while EditMenu!.submenu!.numberOfItems > 8 {EditMenu!.submenu!.removeItem(at:8)}
---
>     // if the edit menu exists, loop over the edit menu items to remove //
>     if EditMenu != nil {for _ in 1...3 {EditMenu!.submenu!.removeItem(at:8)}}
1270c1425
<     if BPComp.SelectedSquare {MovingPiece.Reset(); BPComp.InitSelectedSquare(); Refresh("All")}
---
>     if BPComp.SelectedSquare {MovingPiece.Reset(); BPComp.InitSelectedSquare(); ReplayMoves(true)}
1283,1288c1438,1485
<     // set the menu items dictionary //
<     MenuItemsDict = (Inf.Vals("InPanel").VB || MovingPiece.HaveMovingPiece() || MovingPiece
<                     .Animating()) ? 3 : ((HaveAttachedSheet() || Inf.Vals("MainWindowHidden").VB)
<                     ? 2 : (Inf.Vals("Diagramming").VB ? 5 : (Inf.Vals("InPositionEditor").VB ? 6 :
<                     (Inf.Vals("EngineThinking").VB ? 1 : (Inf.Vals("AnalyzingGame").VB ? 4 : (Inf
<                     .Vals("PlayingTimedGame").VB ? 7 : 0))))))
---
>     // if in a panel or moving or core animating //
>     if Inf.Vals("InPanel").VB || MovingPiece.HaveMovingPiece() || MovingPiece.Animating()
>     {
>         // set the menu items dictionary to use to 3 //
>         MenuItemsDictionary = 3
>     }
>     // if have an attached sheet or the main window is hidden //
>     else if HaveAttachedSheet() || Inf.Vals("MainWindowHidden").VB
>     {
>         // set the menu items dictionary to use to 2 //
>         MenuItemsDictionary = 2
>     }
>     // if diagramming //
>     else if Inf.Vals("Diagramming").VB
>     {
>         // set the menu items dictionary to use to 5 //
>         MenuItemsDictionary = 5
>     }
>     // if in the position editor //
>     else if Inf.Vals("InPositionEditor").VB
>     {
>         // set the menu items dictionary to use to 6 //
>         MenuItemsDictionary = 6
>     }
>     // if the engine is thinking //
>     else if Inf.Vals("EngineThinking").VB
>     {
>         // set the menu items dictionary to use to 1 //
>         MenuItemsDictionary = 1
>     }
>     // if analyzing a game //
>     else if Inf.Vals("AnalyzingGame").VB
>     {
>         // set the menu items dictionary to use to 4 //
>         MenuItemsDictionary = 4
>     }
>     // if playing a timed game //
>     else if Inf.Vals("PlayingTimedGame").VB
>     {
>         // set the menu items dictionary to use to 7 //
>         MenuItemsDictionary = 7
>     }
>     // the default case //
>     else
>     {
>         // set the menu items dictionary to use to 0 //
>         MenuItemsDictionary = 0
>     }
1291,1292c1488,1490
<     return MenuItemStates.MenuItemState(MenuItem.tag, Dictionary:MenuItemsDict, FullScreen:Inf
<            .Vals("FullScreen").VB, HumanOrGameOverOrTimed:Inf.Vals("HumanOrGameOverOrTimed").VB)
---
>     return MenuItemStates.MenuItemState(MenuItem.tag, Dictionary:MenuItemsDictionary, FullScreen:
>                                         Inf.Vals("FullScreen").VB, HumanOrGameOverOrTimed:Inf
>                                         .Vals("HumanOrGameOverOrTimed").VB)
1298,1302c1496,1500
<     // mark move as invalid and exit if not able to find the move //
<     if !BoardFindMove(Board, Move) {Inf.TurnOffVB("VB", Key:"MoveExists"); return}
<
<     // make the move on the board //
<     BoardMakeMove(Board, 0)
---
>     // if were able to find the move //
>     if BoardFindMove(Board, Move)
>     {
>         // make the move on the board //
>         BoardMakeMove(Board, 0)
1304,1305c1502,1503
<     // update the capture and threaten highlights //
<     UpdateCaptureThreatenHighlightLists()
---
>         // update the capture and threaten highlights //
>         UpdateCaptureThreatenHighlightLists()
1307,1308c1505,1513
<     // set that the move exists and was able to be found from possible moves //
<     Inf.TurnOnVB("VB", Key:"MoveExists")
---
>         // set that the move exists and was able to be found from possible moves //
>         Inf.TurnOnVB("VB", Key:"MoveExists")
>     }
>     // if not able to find the move //
>     else
>     {
>         // set that move does not exist //
>         Inf.TurnOffVB("VB", Key:"MoveExists")
>     }
1311,1313d1515
<   // stop the stockfish engine from computing //
<   @objc func StopStockfish() {Stockfish.WriteStockfishToStop()}
<
1317,1318c1519,1520
<     // not playing against a uci engine opponent //
<     if !Inf.Vals("UCIEngineOpponent").VB
---
>     // if playing against the stockfish uci engine //
>     if Inf.Vals("UCIEngineOpponent").VB
1320,1329c1522,1523
<         // get best chess move from the chess engine with max ply depth of 100 //
<         MovingPiece.SaveUCIMove(String(validatingUTF8:EngineFindBestMove(Engine, Board,
<                                 CInt(DynamicSearchTime()), 100))!)
<
<         // exit function //
<         return
<     }
<
<     // if multithreading the stockfish engine, use maximum number of threads //
<     if Inf.Vals("MultithreadUCIEngine").VB {Stockfish.WriteStockfishMaxThreads()}
---
>         // if multithreading the stockfish engine, use maximum number of threads //
>         if Inf.Vals("MultithreadUCIEngine").VB {Stockfish.WriteStockfishMaxThreads()}
1331,1332c1525,1526
<     // if not multithreading the stockfish engine, use only one thread //
<     if !Inf.Vals("MultithreadUCIEngine").VB {Stockfish.WriteStockfishSingleThread()}
---
>         // if not multithreading the stockfish engine, use only one thread //
>         if !Inf.Vals("MultithreadUCIEngine").VB {Stockfish.WriteStockfishSingleThread()}
1334,1335c1528,1529
<     // set the uci engine difficulty option //
<     Stockfish.WriteStockfishSkillLevel(Inf.Vals("UCIEngineDifficulty").VI)
---
>         // set the uci engine difficulty option //
>         Stockfish.WriteStockfishSkillLevel(Inf.Vals("UCIEngineDifficulty").VI)
1337,1338c1531,1532
<     // set the uci engine hash option //
<     Stockfish.WriteStockfishHashSize(Inf.Vals("UCIEngineHash").VI)
---
>         // set the uci engine hash option //
>         Stockfish.WriteStockfishHashSize(Inf.Vals("UCIEngineHash").VI)
1340,1341c1534,1535
<     // set the uci engine to use nnue //
<     Stockfish.WriteStockfishUseNNUE()
---
>         // set the uci engine to use nnue //
>         Stockfish.WriteStockfishUseNNUE()
1343,1345c1537,1539
<     // write to stockfish the current position using the fen string and the move list //
<     Stockfish.WriteStockfishPositionFromFEN(Attributes.FENString(), MoveList:Attributes.Inf
<                                             .GetVSList("Moves"))
---
>         // write to stockfish the current position using the fen string and the move list //
>         Stockfish.WriteStockfishPositionFromFEN(Attributes.FENString(), MoveList:Attributes.Inf
>                                                 .GetVSList("Moves"))
1347,1370c1541,1562
<     // write a move command to the stockfish engine with a max depth of 200 plies //
<     Stockfish.WriteStockfishToMakeNewMove(200)
<
<     // run on the main queue //
<     DispatchQueue.main.async
<     {
<         // set the stockfish timer //
<         self.StockfishTimer = Timer.scheduledTimer(timeInterval:Double(OnlyOnePossibleMove(self
<                               .Board) ? 1 : (self.Inf.Vals("PlayingTimedGame").VB ? 5 : self.Inf
<                               .Vals("SearchTime").VI)), target:self, selector:#selector(self
<                               .StopStockfish), userInfo:nil, repeats:false)
<
<         // add the stockfish timer to the main run loop //
<         RunLoop.current.add(self.StockfishTimer, forMode:RunLoop.Mode.common)
<     }
<
<     // get the best uci move from the stockfish engine //
<     MovingPiece.SaveUCIMove(Stockfish.FindBestMove())
<
<     // invalidate the stockfish timer since move already finished //
<     StockfishTimer.invalidate()
<
<     // initialize the stockfish principle variation string to be empty //
<     StockfishPV = ""
---
>         // if there is only one possible move on the board //
>         if OnlyOnePossibleMove(Board)
>         {
>             // tell the stockfish engine to search for the best move in 1 second //
>             Stockfish.WriteStockfishToMakeNewMove(1)
>         }
>         // there is more than one possible move available to choose from //
>         else
>         {
>             // if playing a timed game //
>             if Inf.Vals("PlayingTimedGame").VB
>             {
>                 // tell engine to make a move in 5 seconds //
>                 Stockfish.WriteStockfishToMakeNewMove(5)
>             }
>             // not playing a timed game //
>             else
>             {
>                 // tell uci engine to make a move for the specified search time //
>                 Stockfish.WriteStockfishToMakeNewMove(Inf.Vals("SearchTime").VI)
>             }
>         }
1372,1373c1564,1580
<     // if the principal variation exists in the information string from stockfish //
<     if (Stockfish.GetInfoString().range(of:" pv ") != nil) && Inf.Vals("ShowStockfishAnalysis").VB
---
>         // get the best uci move from the stockfish engine //
>         MovingPiece.SaveUCIMove(Stockfish.FindBestMove())
>
>         // initialize the stockfish principle variation string to be empty //
>         StockfishPV = ""
>
>         // if the principal variation exists in the information string from stockfish //
>         if (Stockfish.GetInfoString().range(of:" pv ") != nil) && Inf.Vals("ShowStockfishAnalysis")
>             .VB
>         {
>             // get the principal variation from the stockfish information string //
>             StockfishPV = String(Stockfish.GetInfoString().suffix(from:(Stockfish.GetInfoString()
>                                  .range(of:" pv ")?.upperBound)!))
>         }
>     }
>     // playing against the greenchess engine //
>     else
1375,1377c1582,1584
<         // get the principal variation from the stockfish information string //
<         StockfishPV = String(Stockfish.GetInfoString().suffix(from:(Stockfish.GetInfoString()
<                              .range(of:" pv ")?.upperBound)!))
---
>         // get best chess move from the chess engine with max ply depth of 100 //
>         MovingPiece.SaveUCIMove(String(validatingUTF8:EngineFindBestMove(Engine, Board,
>                                 CInt(DynamicSearchTime()), 100))!)
1385c1592,1593
<     if Attributes.NoMoves() && Attributes.IsDefaultFENAndWhite() && !Inf.Vals("UseEcoBook").VB
---
>     if Attributes.NoMoves() && Attributes.IsDefaultFENLayoutAndWhite() && !Inf.Vals("UseEcoBook")
>         .VB
1437c1645
<   // redo the next move that had been made or undone //
---
>   // redo the last move that had been made or undone //
1441c1649,1675
<     if Attributes.Inf.RedoToNextVS() {Refresh("All"); StoreAutosaveData("")}
---
>     if Attributes.Inf.RedoToNextVS() {ReplayMoves(true); StoreAutosaveData("")}
>   }
>
>   // update the list of captured pieces //
>   func StoreCapturedPiece(_ CapturedPiece:Int)
>   {
>     // if captured piece is a white pawn, knight, bishop, rook, or queen //
>     if (1...5).contains(CapturedPiece)
>     {
>         // increment the captured white pieces by one for the specified captured piece type //
>         BPData.IncrementCapturedWhitePieces(5 - CapturedPiece)
>     }
>     // if captured piece is a black pawn, knight, bishop, rook, or queen //
>     else if (1...5).contains(-CapturedPiece)
>     {
>         // increment the captured black pieces by one for the specified piece type //
>         BPData.IncrementCapturedBlackPieces(CapturedPiece + 5)
>     }
>     // if there was an enpassant capture //
>     else if BoardIsEnpassant(Board)
>     {
>         // if the board is currently white to move next, increment white pieces by one pawn //
>         if BoardIsWhite(Board) {BPData.IncrementCapturedWhitePieces(4)}
>
>         // if the board is currently black to move next, increment black pieces by one pawn //
>         if !BoardIsWhite(Board) {BPData.IncrementCapturedBlackPieces(4)}
>     }
1451,1452c1685
<         BPData.StoreCapture(BoardIsEnpassant(Board) ? (BoardIsWhite(Board) ? 1 : -1) : BPData
<                             .EndSquareValue())
---
>         StoreCapturedPiece(BPData.EndSquareValue())
1457,1474d1689
<
<     // if the moving piece is illegal, reset the moving piece and exit //
<     if MovingPiece.IsIllegal() {MovingPiece.Reset(); return}
<
<     // perform an enpassant move if needed //
<     if BoardIsEnpassant(Board) {BPData.RemovePawnCapturedEnPassant()}
<
<     // if there was a kingside castling move by white, move the rook to correct position //
<     if MovingPiece.Castl("e1g1", Piece:6) {BPData.RFU(7, 0, 0); BPData.RFU(5, 0, 4)}
<
<     // if there was a queenside castling move by white, move the rook to correct position //
<     if MovingPiece.Castl("e1c1", Piece:6) {BPData.RFU(0, 0, 0); BPData.RFU(3, 0, 4)}
<
<     // if there was a kingside castling move by black, move the rook to correct position //
<     if MovingPiece.Castl("e8g8", Piece:-6) {BPData.RFU(7, 7, 0); BPData.RFU(5, 7, -4)}
<
<     // if there was a queenside castling move by black, move the rook to correct position //
<     if MovingPiece.Castl("e8c8", Piece:-6) {BPData.RFU(0, 7, 0); BPData.RFU(3, 7, -4)}
1480,1481c1695,1696
<   // make an alert from the given buttons and the user query message //
<   func MakeAlert(_ Sound:String, Buttons:[String], Query:String)
---
>   // promote a pawn //
>   func PromotePawn()
1483,1484c1698,1702
<     // play the game sound //
<     GameSounds.Play(Sound)
---
>     // update the moving piece to the new location on the board //
>     UpdateDestinationSquare()
>
>     // refresh the board and pieces //
>     RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
1486,1487c1704,1714
<     // make the allert //
<     MakeAlert(Buttons, Query:Query)
---
>     // set that currently querying for a pawn promotion //
>     Inf.TurnOnVB("VB", Key:"QueryPawnPromotion")
>
>     // create the needed alert buttons //
>     AlertButtons = ["Queen", "Rook", "Bishop", "Knight"]
>
>     // play the piece thump sound thread //
>     GameSounds.Play("thump")
>
>     // alert the user as to which type of piece they want to promote pawn to //
>     MakeAlert(AlertButtons, Query:"Promote Pawn To")
1503,1509c1730,1764
<     for TheButton in Buttons {Alert.addButton(withTitle:TheButton)}
<
<     // set the alert icon //
<     (Alert.messageText, Alert.icon) = (Query, Inf.Vals("QueryQuitApplication").VB ? MessageIcon :
<     (Inf.Vals("InPositionEditor").VB ? (FENLayoutIsLegal() ? ValidatedIcon : WarningIcon) : (Inf
<     .Vals("QuerySavePosition").VB ? FrameNSImage(Attributes.Inf.GetCount(), Size:100, UpdateBoard:
<     false, Table:false) : (Inf.Vals("QueryLoadPosition").VB ? WarningIcon : MessageIcon))))
---
>     for Index in 0 ..< Buttons.count {Alert.addButton(withTitle:Buttons[Index])}
>
>     // set the alert query message //
>     Alert.messageText = Query
>
>     // if querying to quit the application //
>     if Inf.Vals("QueryQuitApplication").VB
>     {
>         // set the alert icon to the message icon //
>         Alert.icon = MessageIcon
>     }
>     // if currently in the position editor //
>     else if Inf.Vals("InPositionEditor").VB
>     {
>         // if the fen layout is legal for the board position, set the validated icon //
>         Alert.icon = FENLayoutIsLegal() ? ValidatedIcon : WarningIcon
>     }
>     // if query save position //
>     else if Inf.Vals("QuerySavePosition").VB
>     {
>         // set the snapshot image to the current board //
>         Alert.icon = BPComp.GetNSImage(0)
>     }
>     // if loading the saved position which doesn't currently exist //
>     else if Inf.Vals("QueryLoadPosition").VB
>     {
>         // set the alert icon //
>         Alert.icon = WarningIcon
>     }
>     // is a default query //
>     else
>     {
>         // set the alert icon to show a message icon //
>         Alert.icon = MessageIcon
>     }
1520c1775
<     return (MainWindow.attachedSheet != nil) ? MainWindow.attachedSheet! == Window : false
---
>     return MainWindow.attachedSheet != nil ? MainWindow.attachedSheet! == Window : false
1526,1530c1781,1794
<     // update the game time for white and black //
<     (ElapsedWhiteTime, ElapsedBlackTime) = (TimedGameOffset, TimedGameOffset)
<
<     // refresh the clocks //
<     for Clock in [WhiteTimeClock, BlackTimeClock, Ctr.TimeClockPreview] {Clock!.RefreshAll = true}
---
>     // update the game time for white //
>     ElapsedWhiteTime = TimedGameOffset
>
>     // update the game time for black //
>     ElapsedBlackTime = TimedGameOffset
>
>     // set that need to update the white time clock //
>     WhiteTimeClock.RefreshAll = true
>
>     // set that need to update the black time clock //
>     BlackTimeClock.RefreshAll = true
>
>     // set that need to refresh the time clock preview //
>     Ctr.TimeClockPreview.RefreshAll = true
1548,1549c1812,1813
<         // for all the clocks //
<         for TheClock in [WhiteTimeClock, BlackTimeClock]
---
>         // if the alpha value of the game clock is greater than 0.2 //
>         if WhiteTimeClock.alphaValue > 0.2
1551,1552c1815,1819
<             // decrement the clock alpha value //
<             TheClock!.alphaValue -= (TheClock!.alphaValue > 0.2) ? 0.02 : 0
---
>             // decrement the alpha value of the white time clock //
>             WhiteTimeClock.alphaValue -= 0.02
>
>             // decrement the alpha value of the black time clock //
>             BlackTimeClock.alphaValue -= 0.02
```

```
            BlackTimeClock.alphaValue = 0.02
1558,1560c1825,1826
<           // for all the clocks //
<           for TheClock in [WhiteTimeClock, BlackTimeClock]
---
>           // need to update game clock if alpha value less than 1 //
>           else if WhiteTimeClock.alphaValue < 1
1562,1563c1828,1832
<               // increment the clock alpha value //
<               TheClock.alphaValue += (TheClock.alphaValue < 1) ? 0.02 : 0
---
>               // increment the alpha value of the white time clock //
>               WhiteTimeClock.alphaValue += 0.02
>
>               // increment the alpha value of the black time clock //
>               BlackTimeClock.alphaValue += 0.02
1587,1589c1856,1860
<           // calculate the time remaining for white and black //
<           (RemainingWhiteTime, RemainingBlackTime) = (216000 - ElapsedWhiteTime, 216000 -
ElapsedBlackTime)
---
>           // calculate the time remaining for white //
>           RemainingWhiteTime = 216000 - ElapsedWhiteTime
>
>           // calculate the time remaining for black //
>           RemainingBlackTime = 216000 - ElapsedBlackTime
1592c1863
<           if ElapsedWhiteTime == 216000 {GameState = 5}
---
>           if ElapsedWhiteTime == 216000 {CurrentGameState = 5}
1595c1866
<           if ElapsedBlackTime == 216000 {GameState = 6}
---
>           if ElapsedBlackTime == 216000 {CurrentGameState = 6}
1598c1869
<           if GameState > 0
---
>           if CurrentGameState > 0
1602a1874,1876
>               // set the alert button to okay //
>               AlertButtons = ["Okay"]
>
1604c1878
<               if (GameState == 5) || (GameState == 6) {GameSounds.Play("warning")}
---
>               if (CurrentGameState == 5) || (CurrentGameState == 6) {GameSounds.Play("warning")}
1607c1881
<               if TimedGameDict[GameState] != nil
---
>               if TimedGameDictionary[CurrentGameState] != nil
1610c1884
<                   MakeAlert(["Okay"], Query:TimedGameDict[GameState]!)
---
>                   MakeAlert(AlertButtons, Query:TimedGameDictionary[CurrentGameState]!)
1641,1645d1914
<       // initialize variables to empty //
<       (GameListSearchBox.stringValue, GameState, RemainingWhiteTime, RemainingBlackTime,
TimedGameOffset, TotalGameTime, TimedGameColor, TimedGameTime,TimedGameOpponent) = ("", 0, 0,
0, 0, 0, 0, 0, 0)
1646a1916,1939
>       // set the current game state to zero //
>       CurrentGameState = 0
>
>       // initialize the remaining white time to zero //
>       RemainingWhiteTime = 0
>
>       // initialize the remaining black time to zero //
>       RemainingBlackTime = 0
>
>       // set the timed game offset to zero //
>       TimedGameOffset = 0
>
>       // set total game time to zero //
>       TotalGameTime = 0
>
>       // set the timed game color to zero //
>       TimedGameColor = 0
>
>       // set the timed game time to zero //
>       TimedGameTime = 0
>
>       // set the timed game opponent to zero //
>       TimedGameOpponent = 0
>
1654a1948,1950
>
>       // clear the game list search box text field //
>       GameListSearchBox.stringValue = ""
1658c1954
<   func StartNewGame()
---
>   func StartNewGame(_ Display:Bool)
1678,1679c1974,1978
<       // refresh all //
<       Refresh("All")
---
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:true, Display:Display)
>
>       // print the initial game information //
>       PrintGameInfo(Display)
1710,1713c2009,2014
<           // get the file and rank from the FEN enpassant target square location //
<           (File, Rank) = (SquareLabel.FileIndex(Attributes.Inf.Vals("FENEnpassantSquare").VS.first(),
SquareLabel.RankIndex(Attributes.Inf.Vals("FENEnpassantSquare").VS.last())
---
>           // get the file from the FEN enpassant target square location //
>           File = SquareLabel.FileIndex(Attributes.Inf.Vals("FENEnpassantSquare").VS.first())
>
>           // get the rank from the FEN enpassant target square location //
>           Rank = SquareLabel.RankIndex(Attributes.Inf.Vals("FENEnpassantSquare").VS.last())
1746,1747c2047,2051
<       // initialize the number of white and black pieces that were potentially promoted //
<       (PromotedWhitePieces, PromotedBlackPieces) = (0, 0)
---
>       // initialize the number of white pieces that were potentially promoted //
>       PromotedWhitePieces = 0
>
>       // initialize the number of black pieces that were potentially promoted //
>       PromotedBlackPieces = 0
1762c2066
<       for File in 0...7 {for Rank in 0...7 {PiecesInFENLayout[BPData.RFV(File, Rank) + 6] += 1)}}
---
>       for File in 0 ..< 8 {for Rank in 0 ..< 8 {PiecesInFENLayout[BPData.RFV(File, Rank) + 6] += 1)}}
1764,1765c2068,2081
<       // loop over all the files to see if have any illegally placed pawns //
<       for File in 0...7 {if BPData.RFAE(File, 0, 1) || BPData.RFAE(File, 7, 1)
---
>       // loop over all the files //
>       for File in 0 ..< 8
>       {
>           // if have any pawns on the first and eighth ranks which is illegal, set layout illegal //
>           if BPData.RFAE(File, 0, 1) || BPData.RFAE(File, 7, 1)
>           {
>               // set the legal fen layout to false //
>               Inf.TurnOffVB("VR", Key:"LegalFENLayout")
>           }
>       }
>
>       // if number of kings is wrong or number of pawns is wrong, set layout illegal //
>       if (PiecesInFENLayout[0] != 1) || (PiecesInFENLayout[12] != 1) || (PiecesInFENLayout[5] > 8)
>           || (PiecesInFENLayout[7] > 8)
1769,1777c2085
<       }}
<
<       // set the max number of pieces that could have potentially been promoted by white and black //
<       (AllowedWhitePromotedPieces, AllowedBlackPromotedPieces, PromotedWhitePieces,
PromotedBlackPieces) = (8 - PiecesInFENLayout[7], 8 - PiecesInFENLayout[5], max(0,
PiecesInFENLayout[8] - 2) + max(0, PiecesInFENLayout[9] - 2) + max(0, PiecesInFENLayout[10] -
2) + max(0, PiecesInFENLayout[11] - 1), max(0, PiecesInFENLayout[1] - 1) + max(0,
PiecesInFENLayout[2] - 2) + max(0, PiecesInFENLayout[3] - 2) + max(0, PiecesInFENLayout[4] -
2))
---
>       }
1779c2087,2101
<       // if white or black promoted more than number of pieces that could promote, or wrong count //
---
>       // the max number of pieces that could have potentially been promoted by white //
>       AllowedWhitePromotedPieces = 8 - PiecesInFENLayout[7]
>
>       // the max number of pieces that could have potentially been promoted by black //
>       AllowedBlackPromotedPieces = 8 - PiecesInFENLayout[5]
>
>       // the number of pieces that were possibly promoted by white //
>       PromotedWhitePieces = max(0, PiecesInFENLayout[8] - 2) + max(0, PiecesInFENLayout[9] - 2) +
>                             max(0, PiecesInFENLayout[10] - 2) + max(0, PiecesInFENLayout[11] - 1)
>
>       // the number of pieces that were possibly promoted by black //
>       PromotedBlackPieces = max(0, PiecesInFENLayout[1] - 1) + max(0, PiecesInFENLayout[2] - 2) +
>                             max(0, PiecesInFENLayout[3] - 2) + max(0, PiecesInFENLayout[4] - 2)
>
>       // if white or black promoted more than number of pieces that could promote //
1781,1782c2103
<           AllowedBlackPromotedPieces) || (PiecesInFENLayout[0] != 1) || (PiecesInFENLayout[12] != 1)
<           || (PiecesInFENLayout[5] > 8) || (PiecesInFENLayout[7] > 8)
---
>           AllowedBlackPromotedPieces)
1818c2139
<       for File in 0...7 {for Rank in 0...7 {if BPData.RFAE(File, Rank, 0) {TotalPieces += 1}}}
---
>       for File in 0 ..< 8 {for Rank in 0 ..< 8 {if BPData.RFAE(File, Rank, 0) {TotalPieces += 1}}}
1835a2157,2162
>           // create the buttons for the query if want to save the edited position //
>           AlertButtons = ["Save", "Edit", "Exit"]
>
>           // play the sound of a switch //
>           GameSounds.Play("switch")
>
1837c2164
<           MakeAlert("switch", Buttons:["Save", "Edit", "Exit"], Query:"Position Validated")
---
>           MakeAlert(AlertButtons, Query:"Position Validated")
1842,1843c2169,2176
<           // tell that the edited board position is illegal and they need to fix it //
<           MakeAlert("warning", Buttons:["Edit", "Exit"], Query:"Invalid Position")
---
>           // create the button to give user option to edit or exit position editor //
>           AlertButtons = ["Edit", "Exit"]
>
>           // play the sound of a warning //
>           GameSounds.Play("warning")
>
>           // tell that the edited board position is illegal and they need to fix it //
>           MakeAlert(AlertButtons, Query:"Invalid Position")
1854c2187
<       for Rank in (0...7).reversed()
---
>       for Rank in (0 ..< 8).reversed()
1860c2193
<           for File in 0...7
---
>           for File in 0 ..< 8
1882c2215,2216
<                   Attributes.Inf.AddTo("FENPref", Key:"FENLayout", AddOn:FENCharacters[PieceType + 6])
---
>                   Attributes.Inf.AddTo("FENPref", Key:"FENLayout", AddOn:FENLayoutCharacters[PieceType
>                                        + 6])
1901,1936d2234
<       // initilize the slider and buttons //
<       (Ctr.WindowTextureOpacitySlider.intValue, Ctr.IllustrationSizeSlider.intValue, Ctr
.DisplayIllustrationsButton.state, Ctr.ClockBackgroundVisibilitySlider.intValue, Ctr
.AnimationRateSlider.intValue, Ctr.GalleryAnimationRateSlider.intValue, Ctr
.UCIEngineDifficultySlider.intValue, Ctr.UCIEngineHashSlider.intValue, Ctr.SearchTimeSlider
.intValue, Ctr.MovieFPSSlider.intValue, GameSounds.SoundOff, Ctr.GameSoundsButton.state, Ctr
.ShowStockfishAnalysisButton.state, Ctr.DisplayClockButton.state, Ctr.EcoBookButton.state,
Ctr.AutosaveGameButton.state, Ctr.MultithreadUCIEngineButton.state, PreferencesToolbar
.selectedItemIdentifier) = (Int32(Inf.Vals("WindowTextureOpacity").VI), Int32(Inf
.Vals("IllustrationSize").VI), Ctr.UpdatedState(!Inf.Vals("HideIllustrations").VB), Int32(Inf
.Vals("ClockBackgroundVisibility").VI), Int32(Inf.Vals("AnimationRate").VI), Int32(Inf
.Vals("GalleryAnimationRate").VI), Int32(Inf.Vals("UCIEngineDifficulty").VI), Int32(Inf
.Vals("UCIEngineHash").VI), Int32(Inf.Vals("SearchTime").VI), Int32(Inf
.Vals("MovieFPSFromSlider").VI), Inf.Vals("SoundOff").VB, Ctr.UpdatedState(!Inf
.Vals("SoundOff").VB), Ctr.UpdatedState(Inf.Vals("ShowStockfishAnalysis").VB), Ctr
.UpdatedState(!Inf.Vals("HideGameClockInPanel").VB), Ctr.UpdatedState(Inf.Vals("UseEcoBook")
.VB), Ctr.UpdatedState(Inf.Vals("AutosaveGame").VB), Ctr.UpdatedState(Inf
.Vals("MultithreadUCIEngine").VB), NSToolbarItem.Identifier(rawValue:Inf
.IntStr("PreferencesSelection")))
<
<       // read in the move list font //
<       MoveListFont.Read(Ctr.MoveListColorView, DefaultHue:27, TheFont:Ctr.MoveListFontButton, Sizes:
                         Ctr.MoveListSizesButton, Bold:Ctr.MoveListBoldButton, TheScale:1, VertOffset:Inf
                         .Vals("IllustrationSize").VI, UseVertOffset:!Inf.Vals("HideIllustrations").VB,
                         Tag:"MoveList")
<
<       // read in the game info font //
<       GameInfoFont.Read(Ctr.GameInfoColorView, DefaultHue:27, TheFont:Ctr.GameInfoFontButton,
                         Sizes:Ctr.GameInfoSizesButton, Bold:Ctr.GameInfoBoldButton, TheScale:1,
                         VertOffset:0, UseVertOffset:false, Tag:"GameInfo")
<
<       // read in the game list font //
<       GameListFont.Read(Ctr.GameListColorView, DefaultHue:27, TheFont:Ctr.GameListFontButton,
                         Sizes:Ctr.GameListSizesButton, Bold:Ctr.GameListBoldButton, TheScale:1,
                         VertOffset:0, UseVertOffset:false, Tag:"GameList")
1939a2238,2240
>       // update the window texture opacity slider //
>       Ctr.WindowTextureOpacitySlider.intValue = Int32(Inf.Vals("WindowTextureOpacity").VI)
>
1944c2245
<           SetWindowBackgroundColor()
---
>           SetWindowBackgroundColor(false)
1945a2247,2249
>           // update the illustration size slider //
>           Ctr.IllustrationSizeSlider.intValue = Int32(Inf.Vals("IllustrationSize").VI)
>
1947a2252,2254
>           // update the display illustrations button state //
>           Ctr.DisplayIllustrationsButton.state = Ctr.UpdatedState(!Inf.Vals("HideIllustrations").VB)
>
1948a2256,2271
>           // read in the move list font //
>           MoveListFont.Read(Ctr.MoveListColorView, DefaultHue:27, TheFont:Ctr.MoveListFontButton, Sizes:
>                             Ctr.MoveListSizesButton, Bold:Ctr.MoveListBoldButton, TheScale:1,
>                             VertOffset:Inf.Vals("IllustrationSize").VI, UseVertOffset:!Inf
>                             .Vals("HideIllustrations").VB, Tag:"MoveList")
>
>           // read in the game info font //
>           GameInfoFont.Read(Ctr.GameInfoColorView, DefaultHue:27, TheFont:Ctr.GameInfoFontButton,
>                             Sizes:Ctr.GameInfoSizesButton, Bold:Ctr.GameInfoBoldButton, TheScale:1,
>                             VertOffset:0, UseVertOffset:false, Tag:"GameInfo")
>
>           // read in the game list font //
>           GameListFont.Read(Ctr.GameListColorView, DefaultHue:27, TheFont:Ctr.GameListFontButton,
>                             Sizes:Ctr.GameListSizesButton, Bold:Ctr.GameListBoldButton, TheScale:1,
>                             VertOffset:0, UseVertOffset:false, Tag:"GameList")
>
1963a2287,2289
>           // update the clock background visibility slider //
>           Ctr.ClockBackgroundVisibilitySlider.intValue = Int32(Inf.Vals("ClockBackgroundVisibility").VI)
>
1972a2299,2301
>           // update the board animation rate slider //
>           Ctr.AnimationRateSlider.intValue = Int32(Inf.Vals("AnimationRate").VI)
>
1975a2305,2307
>           // update the gallery animation rate slider //
>           Ctr.GalleryAnimationRateSlider.intValue = Int32(Inf.Vals("GalleryAnimationRate").VI)
>
1978a2311,2313
>           // update the uci engine difficulty slider //
>           Ctr.UCIEngineDifficultySlider.intValue = Int32(Inf.Vals("UCIEngineDifficulty").VI)
>
1981a2317,2319
>           // update the uci engine hash slider //
>           Ctr.UCIEngineHashSlider.intValue = Int32(Inf.Vals("UCIEngineHash").VI)
>
1984a2323,2325
>           // update the search time slider //
>           Ctr.SearchTimeSlider.intValue = Int32(Inf.Vals("SearchTime").VI)
>
1987a2329,2331
>           // update the movie frames per second slider //
>           Ctr.MovieFPSSlider.intValue = Int32(Inf.Vals("MovieFPSFromSlider").VI)
>
1990a2335,2346
>           // retrieve the sound off preference //
>           GameSounds.SoundOff = Inf.Vals("SoundOff").VB
>
>           // update the game sounds button //
>           Ctr.GameSoundsButton.state = Ctr.UpdatedState(!Inf.Vals("SoundOff").VB)
>
>           // update the show the stockfish analysis button //
>           Ctr.ShowStockfishAnalysisButton.state = Ctr.UpdatedState(Inf.Vals("ShowStockfishAnalysis").VB)
>
>           // update the display clock button state //
>           Ctr.DisplayClockButton.state = Ctr.UpdatedState(!Inf.Vals("HideGameClockInPanel").VB)
>
1994,1995c2350,2351
<           // update the progress circle color //
<           ProgressCircle.UpdateCircleColor(Ctr.WindowBackgroundColorView.GetCurrentHue())
---
>           // update the use eco book button //
>           Ctr.EcoBookButton.state = Ctr.UpdatedState(Inf.Vals("UseEcoBook").VB)
1997,1998c2353,2354
<           // update to the current preferences selection //
<           PreferencesAction(self)
---
>           // update the autosave game button //
>           Ctr.AutosaveGameButton.state = Ctr.UpdatedState(Inf.Vals("AutosaveGame").VB)
2000,2001c2356,2360
<           // set the opponent to play against with greenchess as the default //
<           UpdateOpponent(Inf.Vals("Opponent").VS)
---
>           // update the multithread uci engine button //
>           Ctr.MultithreadUCIEngineButton.state = Ctr.UpdatedState(Inf.Vals("MultithreadUCIEngine").VB)
>
>           // update the progress circle color //
>           ProgressCircle.UpdateCircleColor(Ctr.WindowBackgroundColorView.GetCurrentHue())
2004c2363
<       for PositionListIndex in 1...25 {if (Attributes.Initialize("\(PositionListIndex)")
---
>       for PositionListIndex in 1...5
2006,2007c2365,2375
<           // convert the current game to game data //
<           ConvertToGameData()
---
>           // initialize game attributes for the given position index //
>           Attributes.Initialize("\(PositionListIndex)")
>
>           // if the position image is not empty //
>           if Attributes.PositionImage.size != NSZeroSize
>           {
>               // append the saved position image //
>               SavedPositionImages.append(Attributes.PositionImage)
>
>               // convert the current game to game data //
>               ConvertToGameData()
2009,2010c2377,2390
<           // create a new element at the end of the saved positions data array //
<           SavedPositionData.append(DataNew())
---
>               // create a new element at the end of the saved positions data array //
>               SavedPositionData.append(DataNew())
>
>               // copy the game data to the end of the saved positions data array //
>               DataCopy(SavedPositionData[SavedPositionData.count - 1], GameData)
>           }
>       }
>
>       // set the initial preferences toolbar item //
>       PreferencesToolbar.selectedItemIdentifier = NSToolbarItem.Identifier(rawValue:Inf
>                                                   .IntStr("PreferencesSelection"))
>
>       // update to the current preferences selection //
>       PreferencesAction(self)
2012,2015d2391
<           // copy the game data to the end of the saved positions data array //
<           DataCopy(SavedPositionData.last, GameData)
<       }}
<
2017,2018c2393,2394
<       _ = Attributes.Initialize("")
---
>
>       Attributes.Initialize("")
2021,2027d2396
<   }
<
<   // get the position images //
<   func GetPositionImages()
<   {
<       // save the current game //
<       Attributes.SaveSubset(Attributes.Inf.GetCount())
2029,2044c2398,2399
<       // clear the position images //
<       PositionImages.removeAll()
<
<       // loop over the save position data //
<       for PositionData in SavedPositionData
<       {
<           // extract the position data //
<           ExtractFromGameData(PositionData, Tasks:"Replay")
<
<           // append the corresponding position image for the data //
<           PositionImages.append(FrameOfImage(Attributes.Inf.GetCount(), Size:800, UpdateBoard:false,
                                Table:false))
<       }
<
<       // reset the game data to the original values //
<       ResetGameData()
---
>       // set the opponent to play against with greenchess as the default //
>       UpdateOpponent(Inf.Vals("Opponent").VS)
2065,2069c2420,2437
<       // update the game count view //
<       GameCountView.stringValue = (GameListData.numberOfRows(in:NSTableView()) == 0) || !Inf
                                   .Vals("ReadInGameList").VB) ? "No Games" : (GameListData
                                   .numberOfRows(in:NSTableView()) == 1) ? "1 Game" :
                                   "\(GameListData.numberOfRows(in:NSTableView())) Games")
---
>       // if the game list is empty or haven't read in a game list //
>       if (GameListData.numberOfRows(in:NSTableView()) == 0) || (!Inf.Vals("ReadInGameList").VB)
>       {
>           // display that have no games in the game list //
>           GameCountView.stringValue = "No Games"
>       }
>       // if only have one game in the game list //
>       else if GameListData.numberOfRows(in:NSTableView()) == 1
>       {
>           // display that only have one game in the game list //
>           GameCountView.stringValue = "1 Game"
>       }
>       // have more than one game in the game list //
>       else
>       {
>           // display number of games in the current game list //
>           GameCountView.stringValue = "\(GameListData.numberOfRows(in:NSTableView())) Games"
>       }
2075,2080c2443,2453
<       // exit if not currently searching the game list //
<       if (!Inf.Vals("SearchingGameList").VB {return}
<
<       // initialize the previously selected row to zero and get the search string from search box //
<       (PreviouslySelectedRow, SearchString) = (0, GetString().FromTextField(GameListSearchBox
<       .stringValue))
<
<       // condense the game collection data which matches the given search string //
<       GameListData.CondenseGameList(SearchString)
---
>       // if currently searching the game list //
>       if Inf.Vals("SearchingGameList").VB
>       {
>           // initialize the previously selected row to zero //
>           PreviouslySelectedRow = 0
>
>           // get the search string from the game list search box when it is updated //
>           SearchString = GetString().FromTextField(GameListSearchBox.stringValue)
>
>           // condense the game collection data which matches the given search string //
>           GameListData.CondenseGameList(SearchString)
2086,2087c2455,2456
<       // update the game list table view //
<       UpdateGameList()
---
>           // update the game list table view //
>           UpdateGameList()
2088,2089c2458,2460
<       // display the current game count //
<       DisplayGameCount()
---
>
>           // display the current game count //
>           DisplayGameCount()
2117,2118c2480,2492
<       // set the initial rank and file values on the board //
<       (Rank, File) = (7, 0)
---
>       // set the initial rank value on the board //
>       Rank = 7
>
>       // set the initial file value on the board //
>       File = 0
2121c2495
<       for Index in 0..<Attributes.Inf.Vals("FENLayout").VS.count
---
>       for Index in 0 ..< Attributes.Inf.Vals("FENLayout").VS.count
2149c2523
<               for _ in 0..<Int(FENPiece)
---
>               for _ in 0 ..< Int(FENPiece)!
2170c2544
<           if File == 8 {(File, Rank) = (0, Rank - 1)}
---
>           if File == 8 {File = 0; Rank -= 1}
2191c2565
<       for Index in 0..<Attributes.Inf.Vals("FENCastlingRights").VS.count
---
>       for Index in 0 ..< Attributes.Inf.Vals("FENCastlingRights").VS.count
2219,2233c2593,2626
<       "MainWindowHidden":false, "GameExplorerWindowOpen":false, "StartClock":false,
        "FullScreen":false, "InPanel":false, "SecondPositionMenu":false, "ValidFENString":true,
        "RecordingMovie":false, "PlayingEngineMatch":false, "ValidFENEnpassantSquare":false,
        "LegalFENLayout":false, "MoveExists":false, "EditingNonemptyPosition":false,
        "InPositionEditor":false, "InDatabaseExplorer":false, "QuerySavePosition":false,
        "QueryLoadPosition":false, "SearchingGameList":false, "ReadInGameList":false,
        "EnteringPreferences":false, "InGameGallery":false, "QueryFENPromotion":false,
        "QueryStartNewGame":false, "QueryTimedGameOver":false, "QueryEditPosition":false,
        "QueryEngineTurn":false, "QueryPawnPromotion":false, "InDiagrammingMode":false,
        "QueryApplicationUpdate":false, "EngineThinking":false, "HumanOpponent":false,
        "UCIEngineOpponent":false, "UCIEngineOpponentSwitch":false, "DiagrammingMode":false,
        "ReviewingGame":false, "EnteringFEN":false, "AnalyzingGame":false, "AnnotatingGame":false,
        "ExploringGame":false, "PlayingTimedGame":false, "WillSaveEditedPosition":false,
        "WhiteKingsideCastle":false, "WhiteQueensideCastle":false, "BlackKingsideCastle":false,
        "BlackQueensideCastle":false]
---
>       "MainWindowHidden":false, "UpdateReview":false, "GameExplorerWindowOpen":false,
>       "StartClock":false, "FullScreen":false, "InPanel":false, "SecondPositionMenu":false,
>       "ValidFENString":true, "RecordingMovie":false, "PlayingEngineMatch":false,
>       "ValidFENEnpassantSquare":false, "LegalFENLayout":false, "MoveExists":false,
>       "EditingNonemptyPosition":false, "InPositionEditor":false, "InGameExplorer":
>       false, "QuerySavePosition":false, "QueryLoadPosition":false, "SearchingGameList":
>       false, "ReadInGameList":false, "EnteringPreferences":false, "InGameGallery":false,
>       "QueryFENPromotion":false, "QueryStartNewGame":false, "QueryTimedGameOver":false,
>       "QueryEditPosition":false, "QueryEngineTurn":false, "QueryEngineThinking":
>       false, "HumanOpponent":false, "EngineOpponent":false, "UCIEngineOpponent":false,
>       "Diagramming":false, "ReviewingGame":false, "AnalyzingGame":false, "AnnotatingGame":
>       false, "EnteringFEN":false, "ExploringGame":false, "WillAnalyzeGame":false,
>       "WillSaveEditedPosition":false, "WhiteKingsideCastle":false, "WhiteQueensideCastle":
>       false, "BlackKingsideCastle":false, "BlackQueensideCastle":false]
2239,2241c2632,2634
<       "MultithreadUCIEngine":true, "UseEcoBook":true, "AutosaveGame":false,
        "ShowStockfishAnalysis":false, "HideInPanel":false, "SmallInFonts":false,
        "LargeInFonts":false, "SoundOff":false]
---
>       "MultithreadUCIEngine":true, "UseEcoBook":false, "AutosaveGame":false,
>       "ShowStockfishAnalysis":false, "HideInPanel":false, "SmallInFonts":false,
>       "LargeInFonts":true, "SoundOff":false]
2245,2259c2619,2651
<       .WindowBackgroundColorView.MaxHue()], "WindowTextureOpacity":[0, 0, 100],
        "IllustrationSize":[250, 150, 250], "GameClockHue":[27, 0, Ctr.ClockColorView.MaxHue()],
        "GameClockBackgroundHue":[82, 0, Ctr.ClockBackgroundColorView.MaxHue()],
        "ClockBackgroundVisibility":[30, 0, 100], "NumberOfGridRows":[3, 2, 3], "AnimationRate":
        [75, 1, 100], "GalleryAnimationRate":[75, 1, 100], "UCIEngineDifficulty":[20, 0, 20],
        "UCIEngineHash":[16, 16, 512], "MovieFPSFromSlider":[10, 0, 100], "SavedPositionSlot":
        [1, 1, 5], "SearchTime":[5, 5, 120], "PreferencesSelection":[20, 20, 28]])
<
<       // loop over all the windows //
<       for TheWindow in [MainWindow, GameGalleryWindow, PreferencesWindow]
<       {
<           // set that can draw concurrently //
<           TheWindow!.allowsConcurrentViewDrawing = true
<       }
---
>       .WindowBackgroundColorView.MaxHue()], "WindowTextureOpacity":[0, 0, 100],
>       "IllustrationSize":[250, 150, 250], "GameClockHue":[27, 0, Ctr.ClockColorView
>       .MaxHue()], "GameClockBackgroundHue":[82, 0, Ctr.ClockBackgroundColorView.MaxHue()],
>       "ClockBackgroundVisibility":[30, 0, 100], "NumberOfGridRows":[3, 2, 3],
>       "AnimationRate":[75, 1, 100], "GalleryAnimationRate":[75, 1, 100],
>       "UCIEngineDifficulty":[20, 0, 20], "UCIEngineHash":[16, 16, 512],
>       "MovieFPSFromSlider":[10, 0, 100], "SavedPositionSlot":[1, 1, 5], "SearchTime":
>       [5, 5, 120], "PreferencesSelection":[20, 20, 28]])
>
>       // make the main window the first responder //
>       MainWindow.initialFirstResponder = self
>
>       // allow the view to draw concurrently //
>       canDrawConcurrently = true
>
>       // set that the main window allows concurrent view drawing //
>       MainWindow.allowsConcurrentViewDrawing = true
>
>       // set that the game gallery window allows concurrent drawing //
>       GameGalleryWindow.allowsConcurrentViewDrawing = true
>
>       // set that the game preferences window allows concurrent drawing //
>       PreferencesWindow.allowsConcurrentViewDrawing = true
>
>       // set the go stop menu item title //
>       GoStopMenuItem.title = "Engine"
>
>       // set the hide menu item title //
>       HideMenuItem.title = "Hide"
>
>       // set the timed game menu item title //
>       TimedGameMenuItem.title = "Timed Game"
2262c2654
<
---
>
2264a2656
>       addTrackingRect(Dims.Board.Bounds, owner:self, userData:nil, assumeInside:false)
2269,2279c2661,2677
<       addTrackingRect(Dims.MainBoardBounds, owner:self, userData:nil, assumeInside:false)
<       // set button values, menu titles, first responders, concurrent drawing, and data sources //
<       (PGNMovesTableView.dataSource, GameListTableView.dataSource, GameInfoTableView.dataSource,
        PGNMovesTableView.backgroundColor, GameListTableView.backgroundColor, GameInfoTableView
        .backgroundColor, MainWindow
        .initialFirstResponder, canDrawConcurrently, GoStopMenuItem.title, HideMenuItem.title,
        TimedGameMenuItem.title, PositionEditorEP.image, EditorHotspotPoint, GameListTableView
        .doubleAction, GameAnnotationTextField.font, GameAnnotationTextField.alignment, AppNameView
        .stringValue) = (PGNMovesData, GameListData, GameInfoData, NSColor.white
        .withAlphaComponent(0.35), NSColor.white.withAlphaComponent(0.35), self, true, "Engine",
        "Hide", "Timed Game", BPComp.Highlights.EnpassantSquareNSImage, CGPoint(x:Dims.Square.Width *
        0.5, y:Dims.Square.Width * 0.5), #selector(LoadGameFromListButtonAction(_:)), NSFont
        .systemFont(ofSize:20), .justified, "GreenChess 9.0.4")
---
>       // set the position editor ep icon image //
>       PositionEditorEP.image = BPComp.Highlights.GetEnpassantSquareNSImage()
>
>       // set the cursor hotspot point //
>       EditorHotspotPoint = CGPoint(x:Dims.ViewSquareSize(2) * 0.5, y:Dims.ViewSquareSize(2) * 0.5)
>
>       // set the double click action command for the game list table view //
>       GameListTableView.doubleAction = #selector(LoadGameFromListButtonAction(_:))
>
>       // set the game annotation text field font to be system font size 20 //
>       GameAnnotationTextField.font = NSFont.systemFont(ofSize:20)
>
>       // set the game annotation text field alignment to be justified //
>       GameAnnotationTextField.alignment = .justified
>
>       // the app name and version //
>       AppNameView.stringValue = "GreenChess 9.0.3"
2290,2296c2688,2696
>       // loop over all the titles //
>       for TheTitle in ["GreenChess", "Human", "Stockfish"]
>       {
>           // add the title to the timed game opponent button //
>           TimedGameOpponentButton.addItem(withTitle:TheTitle)
>       }
---
>       // add greenchess to the timed game popup button //
>       TimedGameOpponentButton.addItem(withTitle:"GreenChess")
>
>       // add human opponent to the timed game popup button //
>       TimedGameOpponentButton.addItem(withTitle:"Human")
>
>       // add uci engine to timed game popup button //
>       TimedGameOpponentButton.addItem(withTitle:"Stockfish")
2318c2718
<
---
>
2324c2724,2733
<
---
>
>       // set the data source for the pgn moves table view //
>       PGNMovesTableView.dataSource = PGNMovesData
>
>       // set the data source for the game list //
>       GameListTableView.dataSource = GameListData
>
>       // set the game info data source //
>       GameInfoTableView.dataSource = GameInfoData
2333c2742,2748
<
---
>
>       // set the pgn moves table view background color to show main window color //
>       PGNMovesTableView.backgroundColor = NSColor.white.withAlphaComponent(0.35)
>
>       // set the game info table view background color to show main window color //
>       GameInfoTableView.backgroundColor = NSColor.white.withAlphaComponent(0.35)
2336,2339c2751,2755
<                   GameGallerySnapshot, GameGalleryErase, GameGalleryBlueColor, GameGalleryCyanColor,
                    GameGalleryMagentaColor, GameGalleryGreenColor, GameGalleryRedColor,
                    GameGalleryYellowColor, GameGalleryBlackColor, GameGalleryWhiteColor,
                    GameGalleryUndoPath, GameGallerySlider], PlayButton:GameGalleryReviewButton)
---
>                   GameGallerySnapshot, GameGalleryErase, GameGalleryBlueColor,
>                   GameGalleryCyanColor, GameGalleryMagentaColor, GameGalleryGreenColor,
>                   GameGalleryRedColor, GameGalleryYellowColor, GameGalleryBlackColor,
>                   GameGalleryWhiteColor, GameGalleryUndoPath, GameGallerySlider],
>                   PlayButton:GameGalleryReviewButton)
2343,2345c2759,2761
<                   EnterDiagramModeButton, GameExplorerSnapshotButton, GameExplorerMovieButton,
                    GameExplorerSlider, GameExplorerLeft, GameExplorerRight], PlayButton:
                    GameReviewButton)
---
>                   EnterDiagramModeButton, GameExplorerSnapshotButton,
>                   GameExplorerMovieButton, GameExplorerSlider, GameExplorerLeft,
>                   GameExplorerRight], PlayButton:GameReviewButton)
2348,2349c2764,2765
<       Sketch.Initialize(self as NSView, Width:Dims.Board.Width, Height:Dims.Board.WidthRet,
                         Length:Dims.Board.WidthRet, YOffset:0, Margin:Dims.Border.BoundsRet.midX)
---
>       Sketch.Initialize(self as NSView, Width:Dims.GetBoardSize(2), Height:Dims.GetBoardSize(2),
>                         Length:Dims.GetBoardSize(2), YOffset:0, Margin:Dims.Border.Width)
2353c2769
<   func WriteMovieTags(_ Tags:[String]) {for TheTag in Tags {WriteMovieTag(TheTag)}}
---
>   func WriteMovieTags(_ Tags:[String]) {for Index in 0 ..< Tags.count {WriteMovieTag(Tags[Index])}}
2358,2365c2774,2775
<       // create the info frame context, set movie info attributes, and initial tag position //
<       (MovieInfoAttributes, MovieInfoTagPosition, Context) = ([NSAttributedString.Key.font:NSFont
        .boldSystemFont(ofSize:22), NSAttributedString.Key.foregroundColor:NSColor.black], 706,
        CoreGraphics().CreateContext(800, Height:800))
<
<       // write the last frame to the movie into frame //
<       Context.draw(FrameOfImage(Attributes.Inf.GetCount(), Size:400, UpdateBoard:false, Table:
                     false), in:CGRect(x:0, y:0, width:800, height:800), byTiling:false)
<
<       // process illustration for the last movie //
<       ProcessIllustration(Attributes.Inf.GetCount(), Set:1)
<
2367,2368c2777,2778
<       // set the current context to write to //
<       NSGraphicsContext.current = NSGraphicsContext(cgContext:Context, flipped:false)
---
>       // set the current context to the movie sized context //
>       NSGraphicsContext.current = NSGraphicsContext(cgContext:BPComp.GetContext(1), flipped:false)
2369a2780,2786
>
>       // set the movie info attributes //
>       MovieInfoAttributes = [NSAttributedString.Key.font:NSFont.boldSystemFont(ofSize:22),
>                              NSAttributedString.Key.foregroundColor:NSColor.black]
>
>       // set the initial movie info tag position //
>       MovieInfoTagPosition = 706
2375,2376c2792,2793
<       Adaptor!.append(PixelBufferFromCGImage(Context!.makeImage()!), withPresentationTime:
                        CMTimeMake(value:0, timescale:10))
---
>       Adaptor!.append(PixelBufferFromCGImage(BPComp.GetContext(1).makeImage()!),
>                       withPresentationTime:CMTimeMake(value:0, timescale:10))
2382,2394c2799,2812
<       // return if the pgn tag is empty //
<       if Attributes.Inf.VSIsEmpty(Key) {return}
<
<       // add the label to the attributed string //
<       AttributedMovieTag = NSAttributedString(string:Attributes.AppTag(Key).first!, attributes:
                             MovieInfoAttributes as? [NSAttributedString.Key:Any])
<
<       // set the text background color to white //
<       NSColor.white.setFill()
<
<       // draw the text background //
<       NSBezierPath(roundedRect:CGRect(x:42, y:MovieInfoTagPosition - 4, width:AttributedMovieTag
                     .size().width + 24, height:33)), xRadius:16.5, yRadius:16.5).fill()
---
>       // if the pgn tag is not empty //
>       if !Attributes.Inf.VSIsEmpty(Key)
>       {
>           // add the label to the attributed string //
>           AttributedMovieTag = NSAttributedString(string:Attributes.AppTag(Key).first!, attributes:
>                                MovieInfoAttributes as? [NSAttributedString.Key:
>                                Any])
>
>           // set the text background color to white //
>           NSColor.white.setFill()
>
>           // draw the text background //
>           NSBezierPath(roundedRect:CGRect(x:42, y:MovieInfoTagPosition - 4, width:AttributedMovieTag
>                        .size().width + 24, height:33)), xRadius:16.5, yRadius:16.5).fill()
2396,2397c2814,2815
<       // draw the attributed string for the given tag //
<       AttributedMovieTag.draw(at:CGPoint(x:54, y:MovieInfoTagPosition))
---
>           // draw the attributed string for the given tag //
>           AttributedMovieTag.draw(at:CGPoint(x:54, y:MovieInfoTagPosition))
2399,2400c2817,2819
<       // decrement the movie info tag position //
<       MovieInfoTagPosition -= 90.91
---
>           // decrement the movie info tag position //
>           MovieInfoTagPosition -= 90.91
>       }
2411,2413d2829
<
<       // exit function //
<       return
2415,2426c2831,2858
<       // if in diagramming mode //
<       if Inf.Vals("Diagramming").VB {Sketch.Cursor.set(); return}
<
<       // if in the position editor //
<       if Inf.Vals("InPositionEditor").VB {PosCursor.set(); return}
<
<       // if not in diagramming or position mode and mouse down //
<       if Moving {NSCursor.pointingHand.set(); return}
<
<       // set the mouse to the open hand cursor //
<       NSCursor.openHand.set()
---
>       // the mouse location is on the board //
>       else
>       {
>           // if in diagramming mode //
>           if Inf.Vals("Diagramming").VB
>           {
>               // set the cursor to the diagram cursor //
>               Sketch.Cursor.set()
>           }
>           // if in the position editor //
>           else if Inf.Vals("InPositionEditor").VB
>           {
>               // set current cursor to the position cursor //
>               PosCursor.set()
>           }
>           // if not in diagramming or position mode and mouse down //
>           else if Moving
>           {
>               // set the mouse to the pointing hand cursor //
>               NSCursor.pointingHand.set()
>           }
>           // not in diagramming or position mode and not mouse down //
>           else
>           {
>               // set the mouse to the open hand cursor //
>               NSCursor.openHand.set()
>           }
>       }
2461c2893
<       // placing a chess piece //
---
>       // placing a chess piece piece //
2475,2476c2907,2908
<       // refresh the view //
<       Refresh("View")
---
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
2507c2939
<       MovingPiece.UpdateLoc(Event.locationInWindow)
---
>       MovingPiece.UpdateLocation(Event.locationInWindow)
2540,2541c2972,2973
<           // refresh the view //
<           Refresh("View")
---
>           // update the board and pieces //
>           RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
2569,2570c3001,3002
<       // refresh the view //
<       Refresh("View")
---
>       // update the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
2574,2577c3006,3054
<           .foreverDuration, mode:RunLoop.Mode.eventTracking, handler:{(Event, Done) in guard let
            Event = Event else {return}; if Event.type == .leftMouseDragged {UpdateCursor(Event
            .locationInWindow, Moving:true); MovingPiece.UpdateLocAndPos(Event)} else
            {TheMouseUp(Event); Done.pointee = true}})
---
>                   .foreverDuration, mode:RunLoop.Mode.eventTracking, handler:{(Event, Done)
>                   in guard let Event = Event else {return}; if Event.type ==
>                   .leftMouseDragged {UpdateCursor(Event.locationInWindow, Moving:true)
>                   MovingPiece.UpdateLocationAndPosition(Event)} else {TheMouseUp(Event)
>                   Done.pointee = true}})
>   }
>
>   // perform a castle and enpassant if needed //
>   func PerformCastleEnpassantIfNeeded()
>   {
>       // if an en passant move occurred, remove the pawn captured en passant //
>       if BoardIsEnpassant(Board) {BPData.RemovePawnCapturedEnPassant()}
>
>       // if there was a white kingside castle //
>       if KCastles.contains(String(validatingUTF8:BoardPGNLabel(Board))!) && BoardIsBlack(Board)
>       {
>           // assign UCI notation for white kingside castle //
>           MovingPiece.SaveUCIMove("h1f1")
>       }
>       // if there was a white queenside castle //
>       else if QCastles.contains(String(validatingUTF8:BoardPGNLabel(Board))!) && BoardIsBlack(Board)
>       {
>           // assign UCI notation for white queenside castle //
>           MovingPiece.SaveUCIMove("a1d1")
>       }
>       // if there was a black kingside castle //
>       else if KCastles.contains(String(validatingUTF8:BoardPGNLabel(Board))!) && BoardIsWhite(Board)
>       {
>           // assign UCI notation for black kingside castle //
>           MovingPiece.SaveUCIMove("h8f8")
>       }
>       // if there was a black queenside castle //
>       else if QCastles.contains(String(validatingUTF8:BoardPGNLabel(Board))!) && BoardIsWhite(Board)
>       {
>           // assign UCI notation for black queenside castle //
>           MovingPiece.SaveUCIMove("a8d8")
>       }
>       // move was not a castling move //
>       else
>       {
>           // exit function since was not a castling move //
>           return
>       }
>
>       // convert the UCI move to origin and destination squares //
>       ProcessUCIMove()
>
>       // put the piece that moved onto the destination square //
>       UpdateDestinationSquare()
2611a3089,3091
>   // undo the illegal move //
<   func UndoIllegalMove() {MovingPiece.SetIllegal(); BPData.SwapBoardSquares(); RunCoreAnimation()}
---
2613c3093
<   func PointOutOfBounds(_ Point:CGPoint) -> Bool {return !Dims.Board.Bounds.contains(Point)}
---
>   func PointOutOfBounds(_ Point:CGPoint) -> Bool {return !Dims.MainBoardBounds.contains(Point)}
2635,2639c3115,3116
<       // set the move to be illegal //
<       MovingPiece.SetIllegal()
<
<       // run the core animation //
<       RunCoreAnimation()
---
>       // undo the illegal move //
>       UndoIllegalMove()
2646,2665c3123
<       if MovingPiece.PawnCanPromote()
<       {
<           // update the moving piece to the new location on the board //
<           UpdateDestinationSquare()
<
<           // refresh the view //
<           Refresh("View")
<
<           // set that currently querying for a pawn promotion //
<           Inf.TurnOnVB("VB", Key:"QueryPawnPromotion")
<
<           // alert the user as to which type of piece they want to promote pawn to //
<           MakeAlert("thump", Buttons:["Queen", "Rook", "Bishop", "Knight"], Query:"Promote Pawn To")
<
<           // exit the function //
<           return
<       }
<
<       // finish up the move //
<       FinishUpMove()
---
>       if MovingPiece.PawnCanPromote() {PromotePawn()} else {FinishUpMove()}
2685a3144,3146
>
>       // perform castle and enpassant if needed //
>       PerformCastleEnpassantIfNeeded()
2689,2690c3150,3151
```

```
<       // refresh the view and tables //
<       Refresh("ViewTables")
---
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
2697a3159,3161
>
>       // print the game information to the screen //
>       PrintGameInfo(true)
2703c3167,3177
<   @IBAction func StockfishAnal(_:AnyObject) {TogglePref("ShowStockfishAnalysis"); Refresh("All")}
---
>   @IBAction func ShowStockfishAnalysisButtonAction(_:AnyObject)
>   {
>       // switch the stockfish analysis option //
>       Inf.ToggleVB("VBPref", Key:"ShowStockfishAnalysis")
>
>       // play the sound of a button click //
>       GameSounds.Play("click")
>
>       // print game info //
>       PrintGameInfo(true)
>   }
2709c3183
<       TogglePref("SoundOff")
---
>       Inf.ToggleVB("VBPref", Key:"SoundOff")
2712a3187,3189
>
>       // play the sound of a button click //
>       GameSounds.Play("click")
2721,2722c3198,3199
<       // if currently not diagramming and the main view image exists //
<       if !Inf.Vals("Diagramming").VB && (MainViewImage != nil)
---
>       // if currently not diagramming //
>       if !Inf.Vals("Diagramming").VB
2725c3202
<           NSGraphicsContext.current?.cgContext.draw(MainViewImage, in:bounds, byTiling:false)
---
>           NSGraphicsContext.current?.cgContext.draw(BPComp.GetCGImage(2), in:bounds, byTiling:false)
2758,2760c3235,3237
<   // the snapshot tasks //
<   func SnapshotTasks()
---
>   // take a snapshot of the gallery //
>   @IBAction func GallerySnapshot(_:AnyObject)
2770,2774d3246
<
<       // set the allowed file types to jpeg and set the snapshot name //
<       (SavePanel.allowedFileTypes, SavePanel.nameFieldStringValue) = (["jpeg"],
<       "GreenChessSnapshot")
<   }
2776,2780c3248,3249
<   // take a snapshot of the gallery //
<   @IBAction func GallerySnapshot(_:AnyObject)
<   {
<       // do the snapshot tasks //
<       SnapshotTasks()
---
>       // set the allowed file types to jpeg //
>       SavePanel.allowedFileTypes = ["jpeg"]
2781a3251,3253
>
>       // set the snapshot file name to GreenChessSnapshot //
>       SavePanel.nameFieldStringValue = "GreenChessSnapshot"
2784,2786c3256,3258
<                   .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self.SavePanel
<                   .orderOut(nil); self.SaveGallerySnapshot(self.SavePanel.url!.path)} else {self
<                   .ReopenPreviousSheet()}})
---
>                               .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK
>                               {self.SavePanel.orderOut(nil); self.SaveGallerySnapshot(self
>                               .SavePanel.url!.path)} else {self.ReopenPreviousSheet()}})
2789,2790c3261,3262
<   // the movie tasks //
<   func MovieTasks()
---
>   // make a gallery movie //
>   @IBAction func GalleryMovie(_:AnyObject)
2800,2809d3271
<
<       // set the allowed file type and set the movie file name //
<       (SavePanel.allowedFileTypes, SavePanel.nameFieldStringValue) = (["mov"], "GreenChessMovie")
<   }
<
<   // make a gallery movie //
<   @IBAction func GalleryMovie(_:AnyObject)
<   {
<       // do the movie tasks //
<       MovieTasks()
2810a3273,3278
>       // set the allowed file types to mov //
>       SavePanel.allowedFileTypes = ["mov"]
>
>       // set the movie file name to GreenChessMovie //
>       SavePanel.nameFieldStringValue = "GreenChessMovie"
2813,2814c3281,3283
<                   .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self.SavePanel
<                   .orderOut(nil); self.WriteGalleryMovie()} else {self.ReopenPreviousSheet()}})
---
>                               .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK
>                               {self.SavePanel.orderOut(nil); self.WriteGalleryMovie()} else {self
>                               .ReopenPreviousSheet()}})
2820,2821c3289,3302
<       // do the movie tasks //
<       MovieTasks()
---
>       // play the sound of clicking on the game to movie icon //
>       GameSounds.Play("movie")
>
>       // close the attached sheet if have one //
>       CloseAttachedSheet()
>
>       // create a panel to save the game movie //
>       SavePanel = NSSavePanel()
>
>       // set the allowed type for a movie to be mov //
>       SavePanel.allowedFileTypes = ["mov"]
>
>       // set the default movie name //
>       SavePanel.nameFieldStringValue = "GreenChessMovie"
2825,2827c3306,3309
<                   .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self.SavePanel
<                   .orderOut(nil); self.Inf.TurnOnVB("VB",Key:"RecordingMovie"); self.WriteGameMovie()}
<                   else {self.ReopenPreviousSheet()}})
---
>                               .Play("buttonclick"); if Result == NSApplication.ModalResponse.OK
>                               {self.SavePanel.orderOut(nil); self.Inf.TurnOnVB("VB",Key:
>                               "RecordingMovie"); self.WriteGameMovie()} else {self
>                               .ReopenPreviousSheet()}})
2833,2835c3315,3323
<       // get the base address of the cgimage data and reset the pixel buffer //
<       (BaseAddress, PixelBuffer) = (CFDataGetBytePtr((Image.dataProvider?.data)!), nil)
---
>       // get the cgimage data //
>       CGImageData = Image.dataProvider?.data
>
>       // get the base address of the cgimage data //
>       BaseAddress = CFDataGetBytePtr(CGImageData!)
>
>       // reset the pixel buffer //
>       PixelBuffer = nil
2841,2842c3329,3330
<                               .release()), Unmanaged<CFData>.passRetained((Image.dataProvider?
<                               .data)!).toOpaque(), nil, &PixelBuffer)
---
>                               .release()), Unmanaged<CFData>.passRetained(CGImageData!)
>                               .toOpaque(), nil, &PixelBuffer)
2858,2862c3346,3352
<       // assign the video writer and video input //
<       (VidInput, VideoWriter) = (AVAssetWriterInput(mediaType:AVMediaType.video, outputSettings:
<       [AVVideoCodecKey:AVVideoCodecType.h264, AVVideoWidthKey:NSNumber(value:Int(VidWidth)),
<       AVVideoHeightKey:NSNumber(value:Int(VidHeight))]), try? AVAssetWriter(outputURL:SavePanel
<       .url!, fileType:AVFileType.mov))
---
>       // assign the video writer //
>       VideoWriter = try? AVAssetWriter(outputURL:SavePanel.url!, fileType:AVFileType.mov)
>
>       // assign the video input //
>       VidInput = AVAssetWriterInput(mediaType:AVMediaType.video, outputSettings:[AVVideoCodecKey:
>                               AVVideoCodecType.h264, AVVideoWidthKey:NSNumber(value:
>                               Int(VidWidth)), AVVideoHeightKey:NSNumber(value:Int(VidHeight))])
2893,2894c3383,3393
<       // reset the values to nil //
<       (Adaptor, VidInput, VideoWriter, PixelBuffer) = (nil, nil, nil, nil)
---
>       // reset the adaptor //
>       Adaptor = nil
>
>       // reset the vid input //
>       VidInput = nil
>
>       // reset the video writer //
>       VideoWriter = nil
>
>       // reset the pixel buffer //
>       PixelBuffer = nil
2907c3406
<       for FrameIndex in 0...GameGalleryView.MaxPosition
---
>       for FrameIndex in 0 ..< GameGalleryView.MaxPosition + 1
2914,2915c3413,3414
<                   withPresentationTime:CMTimeMake(value:Int64(CGFloat(FrameIndex * 10) / MovieFPS),
<                   timescale:10))
---
>                   withPresentationTime:CMTimeMake(value:Int64(CGFloat(FrameIndex * 10) /
>                   MovieFPS), timescale:10))
2927c3426,3428
<           properties:[:])!.write(to:URL(fileURLWithPath:Filename), options:[.atomic])
---
>                               properties:[:])!.write(to:
>                               URL(fileURLWithPath:Filename), options
>                               :[.atomic])
2940,2942c3441,3443
<       Adaptor!.append(PixelBufferFromCGImage(FrameCGImage(FrameNumber, Size:400, UpdatesBoard:
<                   false, Table:false), withPresentationTime:CMTimeMake(value:Int64(50 +
<                   CGFloat(FrameNumber * 10) / MovieFPS), timescale:10))
---
>       Adaptor!.append(PixelBufferFromCGImage(BPComp.GetCGImage(2)), withPresentationTime:
>                   CMTimeMake(value:Int64(50 + CGFloat(FrameNumber * 10) / MovieFPS),
>                   timescale:10))
2949c3450
<       PrepareToWriteAMovie(800, VidHeight:800)
---
>       PrepareToWriteAMovie(Dims.GetBoardSize(1), VidHeight:Dims.GetBoardSize(1))
2964,2965c3465,3478
<       // do the snapshot tasks //
<       SnapshotTasks()
---
>       // close the attached sheet if have one //
>       CloseAttachedSheet()
>
>       // play the game sound of a camera taking a photo //
>       GameSounds.Play("camera")
>
>       // create a save panel for the board snapshot image //
>       SavePanel = NSSavePanel()
>
>       // set the allowed type of file format for a saved board snapshot image //
>       SavePanel.allowedFileTypes = ["jpeg"]
>
>       // set the default filename to save board snapshot as //
>       SavePanel.nameFieldStringValue = "GreenChessSnapshot"
2969,2971c3482,3485
<                   .GameSounds.Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self
<                   .SavePanel.orderOut(nil); self.SaveBoardSnapshot(self.SavePanel.url!.path)} else
<                   {self.ReopenPreviousSheet()}})
---
>                               .GameSounds.Play("buttonclick"); if Result == NSApplication
>                               .ModalResponse.OK {self.SavePanel.orderOut(nil); self
>                               .SaveBoardSnapshot(self.SavePanel.url!.path)} else {self
>                               .ReopenPreviousSheet()}})
2976a3491,3497
>       // if in the game explorer or the database explorer //
>       if Inf.Vals("GameExplorerWindowOpen").VB || Inf.Vals("InDatabaseExplorer").VB
>       {
>           // process the current illustration //
>           ProcessIllustration(IllustrationIndex, Set:2)
>       }
>
2978,2982c3499,3500
<       SnapshotImageRep = NSBitmapImageRep(cgImage:Inf.Vals("Diagramming").VB ? Sketch.Snapshot() :
<                   FrameCGImage((Inf.Vals("GameExplorerWindowOpen").VB ? Int(GameExplorerSlider
<                   .intValue) : (Inf.Vals("InDatabaseExplorer").VB ?
<                   Int(DatabaseGameExplorerSlider.intValue) : Attributes.Inf.GetCount()),
<                   Size:400, UpdatesBoard:false, Table:false))
---
>       SnapshotImageRep = Inf.Vals("Diagramming").VB ? NSBitmapImageRep(cgImage:Sketch.Snapshot()) :
>                   NSBitmapImageRep(cgImage:BPComp.GetCGImage(2))
2989c3507,3508
<           .write(to:URL(fileURLWithPath:Filename), options:[.atomic])
---
>                               .write(to:URL(fileURLWithPath:Filename), options:
>                               [.atomic])
3017,3019c3536,3541
<       // initialize number of move plies and add header information to pgn string //
<       (Plies, PGN) = (BoolTo().Num(Attributes.BlackStarts()), Attributes.GetPGNGameHeader())
---
>       // initialize number of move plies //
>       Plies = BoolTo().Num(Attributes.BlackStarts())
>
>       // add header information to pgn string //
>       PGN = Attributes.GetPGNGameHeader()
3021c3543
<       for Index in 0..<Attributes.Inf.GetCount()
---
>       for Index in 0 ..< Attributes.Inf.GetCount()
3069c3591
<       for Index in 0..<Attributes.Inf.GetCount()
---
>       for Index in 0 ..< Attributes.Inf.GetCount()
3100c3622
<       for TheBook in OpeningBooks {ReadOpeningBook(TheBook)}
---
>       for BookIndex in 0 ..< OpeningBooks.count {ReadOpeningBook(OpeningBooks[BookIndex])}
3111c3633
<       for OpeningBookIndex in 0..<Int(DatabaseGetDataListSize(OpeningBookGameList))
---
>       for OpeningBookIndex in 0 ..< Int(DatabaseGetDataListSize(OpeningBookGameList))
3121,3122c3643,3644
<           // loop over the move list for this game in the opening book up to the first 8 moves //
<           for MoveListCountIndex in 0..<min(8, OpeningBookMoveList.count)
---
>           // loop over the move list for this game in the opening book up to the first 10 moves //
>           for MoveListCountIndex in 0 ..< min(10, OpeningBookMoveList.count)
3142,3144c3664,3669
<       // set the slider minimum and maximum values //
<       (Slider.minValue, Slider.maxValue) = (Double(MinV), max(Slider.minValue, Double(MaxV)))
---
>       // set the slider minimum value //
>       Slider.minValue = Double(MinV)
>
>       // set the slider maximum value //
>       Slider.maxValue = max(Slider.minValue, Double(MaxV))
3150c3675
<   func OpenGame(_ Filename:String)
---
>   func OpenGame()
3156c3681
<       DatabaseReadGameCollection(GameList, Filename)
---
>       DatabaseReadGameCollection(GameList, GameFilename)
3161a3687,3689
>
>       // set that doesn't allow multiple row selection //
>       GameListTableView.allowsMultipleSelection = false
3181,3183c3709,3713
<       // clear the search string and update the search string to the game list search box //
<       (GameListTableView.allowsMultipleSelection, SearchString, GameListSearchBox.stringValue) =
<       (false, "", "")
---
>       // clear the search string //
>       SearchString = ""
>
>       // update the search string to the game list search box //
>       GameListSearchBox.stringValue = SearchString
3187c3717
<   @IBAction func CancelLoadGameFromListButtonAction(_:AnyObject) {CloseListWindow()}
---
>   @IBAction func CancelLoadGameFromListButtonAction(_:AnyObject) {CloseGameListWindow()}
3213c3743
<       CloseListWindow()
---
>       CloseGameListWindow()
3219c3749
<       if (0..<Int(DatabaseGetDataListSize(GameList))).contains(GameListTableView.selectedRow)
---
>       if (0 ..< Int(DatabaseGetDataListSize(GameList))).contains(GameListTableView.selectedRow)
3221,3222c3751,3753
<           // start a new game //
<           StartNewGame()
---
>           // get the selected game to display //
>           GameData = DatabaseGetGameData(GameData, GameList, CInt(GameListData
>                   .GetGameIndex(GameListTableView.selectedRow)))
3224,3226c3755,3758
<           // extract and display the game information from the game data list //
<           ExtractFromGameData(DatabaseGetGameData(GameData, GameList, CInt(GameListData
<                   .GetGameIndex(GameListTableView.selectedRow))), Tasks:"All")
---
>
>           // display the selected game from the game list //
>           DisplaySelectedGameFromGameList()
>       }
>   }
3228,3229c3760,3764
<           // store the autosave data //
<           StoreAutosaveData("")
---
>   // display the selected game from the game list //
>   func DisplaySelectedGameFromGameList()
>   {
>       // start a new game //
>       StartNewGame(false)
3231,3233c3766,3773
<       // initialize the game clock //
<       InitializeGameClock()
<   }
---
>       // extract and display the game information from the game data list //
>       ExtractFromGameData(true)
>
>       // store the autosave data //
>       StoreAutosaveData("")
>
>       // initialize the game clock //
>       InitializeGameClock()
3270,3271c3810,3815
<           // set there was no database loaded yet and set pgn move database text field to be empty //
<           (DatabaseGameDescription.stringValue, PGNMoveDatabase.stringValue) = ("No Database Loaded",
<           "")
---
>           // set there was no database loaded yet //
>           DatabaseGameDescription.stringValue = "No Database Loaded"
>
>           // set the pgn move database text field to be empty //
>           PGNMoveDatabase.stringValue = ""
3280,3282c3822,3823
<           // update the database game explorer image //
<           UpdateExplorerImage(DatabaseGameExplorerSlider, TextView:PGNMoveDatabase, ImageView:
<                   DatabaseExplorerImage, Shift:0)
---
>           // update the database explorer image with the current board image //
>           DatabaseExplorerImage.Update(BPComp.GetCGImage(2))
3334,3339c3875,3881
<       // turn on the white pawn toolbar item and lighten the table views //
<       (MainToolbar.selectedItemIdentifier, PGNMoveTableView.enclosingScrollView!.alphaValue,
<       GameInfoTableView.enclosingScrollView!.alphaValue) = (NSToolbarItem.Identifier(rawValue:
<       "204"), 0.25, 0.25)
<
<       // save the edited position to false and second position menu to false //
---
>       // lighten the pgn moves table view //
>       PGNMoveTableView.enclosingScrollView?.alphaValue = 0.35
>
>       // lighten the game info table view //
>       GameInfoTableView.enclosingScrollView?.alphaValue = 0.35
>
>       // save edited position to false and second position menu to false //
3343a3886,3888
>
>       // turn on the white pawn item //
>       MainToolbar.selectedItemIdentifier = NSToolbarItem.Identifier(rawValue:"204")
3345,3346c3890,3891
<       // refresh the view and tables //
<       Refresh("ViewTables")
---
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
3410c3955
<           StartNewGame()
---
>           StartNewGame(false)
3432,3433c3977,3978
<       // refresh all //
<       Refresh("All")
---
>       // replay all the moves //
>       ReplayMoves(true)
3435,3436c3980,3985
<       // unhide the table views //
<       (PGNMoveTableView.enclosingScrollView!.alphaValue, GameInfoTableView.enclosingScrollView!
<       .alphaValue) = (1, 1)
---
>
>       // unhide the pgn moves table view //
>       PGNMoveTableView.enclosingScrollView?.alphaValue = 1
>
>       // unhide the game info table view //
>       GameInfoTableView.enclosingScrollView?.alphaValue = 1
3457,3459c4005,4007
<                   .GameSounds.Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self
<                   .AppendPanel.orderOut(nil); self.AppendGame(self.AppendPanel.url!.path)}
<                   .ReopenPreviousSheet()}})
---
>                               .GameSounds.Play("buttonclick"); if Result == NSApplication
>                               .ModalResponse.OK {self.AppendPanel.orderOut(nil); self
>                               .AppendGame(self.AppendPanel.url!.path)} else {self
>                               .ReopenPreviousSheet()}})
3476,3478c4024,4027
<                   .GameSounds.Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self
<                   .OpenPanel.orderOut(nil); self.OpenGame(self.OpenPanel.url!.path)} else {self
<                   .ReopenPreviousSheet()}})
---
>                               .GameSounds.Play("buttonclick"); if Result == NSApplication
>                               .ModalResponse.OK {self.OpenPanel.orderOut(nil); self.GameFilename =
>                               self.OpenPanel.url!.path; self.OpenGame()} else {self
>                               .ReopenPreviousSheet()}})
3490,3491c4039,4043
<       // set the allowed type of file format for a saved game and the filename to save as //
<       (SavePanel.allowedFileTypes, SavePanel.nameFieldStringValue) = (["pgn"], "GreenChessGame")
---
>       // set the allowed type of file format for a saved game //
>       SavePanel.allowedFileTypes = ["pgn"]
>
>       // set the default filename to save game as //
>       SavePanel.nameFieldStringValue = "GreenChessGame"
3495,3497c4047,4049
<                   .GameSounds.Play("buttonclick"); if Result == NSApplication.ModalResponse.OK {self
<                   .SavePanel.orderOut(nil); self.SaveGame(self.SavePanel.url!.path)} else {self
<                   .ReopenPreviousSheet()}})
---
>                               .GameSounds.Play("buttonclick"); if Result == NSApplication
>                               .ModalResponse.OK {self.SavePanel.orderOut(nil); self.SaveGame(self
>                               .SavePanel.url!.path)} else {self.ReopenPreviousSheet()}})
3506,3507c4058,4059
<       // refresh all //
<       Refresh("All")
---
>       // replay the moves //
>       ReplayMoves(true)
3574,3575c4126,4127
<       // refresh all //
<       Refresh("All")
---
>       // replay the moves //
>       ReplayMoves(true)
3591,3592c4143,4144
<       // refresh all //
<       Refresh("All")
---
>       // replay the moves to the current game position //
>       ReplayMoves(true)
3604,3605c4156,4157
<       // refresh all //
<       Refresh("All")
---
>       // replay the moves //
>       ReplayMoves(true)
3610d4162
<
3612c4164
<   func ExtractFromGameData(_ GameData:UnsafeMutableRawPointer, Tasks:String)
---
>   func ExtractFromGameData(_ Display:Bool)
3620,3621c4172,4173
<       // update the pgn tags //
<       if Tasks == "All" {PGNTags.UpdateGameInfoFromGameData(GameData)}
---
>       // update the pgn tags if will display //
>       if Display {PGNTags.UpdateGameInfoFromGameData(GameData)}
3623,3624c4175,4176
<       // refresh the tasks //
<       Refresh(Tasks)
---
>       // replay all the moves for this current game //
>       ReplayMoves(Display)
3629c4180,4187
<   func CloseListWindow() {Inf.TurnOffVB("VB", Key:"SearchingGameList"); CloseSheet(GameListWindow)}
---
>   func CloseGameListWindow()
>   {
>       // set searching game list to false //
>       Inf.TurnOffVB("VB", Key:"SearchingGameList")
>
>       // close the game list windows //
>       CloseSheet(GameListWindow)
>   }
3683,3686c4242,4247
<       // set the white and black time clocks to refresh all //
<       (WhiteTimeClock.RefreshAll, BlackTimeClock.RefreshAll) = (true, true)
---
>       // set that the white time clock to refresh all //
>       WhiteTimeClock.RefreshAll = true
>
>       // set the black time clock to refresh all //
>       BlackTimeClock.RefreshAll = true
3690a4253,4255
>
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
3695,3696c4260,4265
<       // refresh the board and tables or all //
<       Refresh(Inf.Vals("InPositionEditor").VB ? "BoardTables" : "All")
---
>       // update the move list font with the illustration size and if have illustrations or not //
>       MoveListFont.UpdateFont(Inf.Vals("IllustrationSize").VI, UseVertOffset:Inf
>                   .Vals("HideIllustrations").VB)
>
>       // update the move list //
>       UpdateMoveList(true)
3711,3712c4280,4281
<       // refresh the views //
<       Refresh("All")
---
>       // print current game information //
>       PrintGameInfo(true)
3727a4297,4306
>
>       // set the number of layers for the gallery //
>       GameGalleryView.Layers = NumberOfLayers
>
>       // set the number of frames for the gallery //
>       GameGalleryView.NumberOfFrames = Attributes.Inf.GetCount() + 1
>
>       // set the gallery frame size //
>       GameGalleryView.FrameSize = MainWindow.frame.size.width / CGFloat(Int(MainWindow.frame.size
>                   .width / ApproxSize))
3731,3734d4309
<
<       // set the game gallery view frame size //
<       (GameGalleryView.Layers, GameGalleryView.FrameSize) = (NumberOfLayers, MainWindow.frame.size
<       .width / CGFloat(Int(MainWindow.frame.size.width / ApproxSize)))
3737,3743c4312,4324
<                   GameGalleryView.FrameSize * CGFloat(GameGalleryView.Layers) + 90))
<
<       // set the number of layers, frames, and frame size for the gallery //
<       (GameGalleryView.NumberOfFrames, GameGalleryView.frame, GameGallerySlider.frame, GalleryToolbar
<       .selectedItemIdentifier) = (Attributes.Inf.GetCount() + 1, CGRect(x:0, y:50, width:MainWindow
<       .frame.size.width, height:GameGalleryView.FrameSize * CGFloat(GameGalleryView.Layers)),
<       CGRect(x:15, y:13, width:MainWindow.frame.size.width - 30, height:25), nil)
---
>                               GameGalleryView.FrameSize * CGFloat(GameGalleryView.Layers) +
>                               90))
>
>       // set the frame of the game gallery view to fit the size of the main window //
>       GameGalleryView.frame = CGRect(x:0, y:50, width:MainWindow.frame.size.width, height:
>                               GameGalleryView.FrameSize * CGFloat(GameGalleryView.Layers))
>
>       // set the frame of the game gallery slider to fit the size of the main window //
>       GameGallerySlider.frame = CGRect(x:15, y:13, width:MainWindow.frame.size.width - 30, height:
>                               25)
>
>       // deselect the gallery diagram color button //
>       GalleryToolbar.selectedItemIdentifier = nil
3779,3782c4360,4368
<   @IBAction func UpdateMoveListFont(_:AnyObject) {Refresh("Tables")}
<
<   // update the game info font //
<   @IBAction func UpdateGameInfoFont(_:AnyObject) {Refresh("Tables")}
---
>   @IBAction func UpdateMoveListFont(_:AnyObject)
>   {
>       // update the move list font //
>       MoveListFont.UpdateFont(Inf.Vals("IllustrationSize").VI, UseVertOffset:Inf
>                   .Vals("HideIllustrations").VB)
>
>       // update the move list //
>       UpdateMoveList(true)
>   }
3790,3791c4376,4387
<       // refresh the board //
<       Refresh("Board")
---
>       // refresh the board and pieces //
>       RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
>   }
>
>   // update the game info font //
>   @IBAction func UpdateGameInfoFont(_:AnyObject)
>   {
>       // update the game info font //
>       GameInfoFont.UpdateFont(0, UseVertOffset:false)
>
>       // update the game info list //
>       UpdateGameInfoList(true)
3837,3838c4433,4438
<       // update the preference previews //
<       UpdatePreviews(true, Diagram:true)
---
>       // create the smaller diagram image //
>       RefreshBoardAndPieces(0, UpdateBoard:true, Display:false)
>
>       // update the board and diagram previews //
>       Ctr.UpdateBoardPreviews(BPComp.GetNSImage(400, Set:2), DiagramImage:BPComp
>                   .GetNSImage(Inf.Vals("IllustrationSize").VI, Set:0))
3846,3851c4446,4451
<       // return if selected preference tab doesn't exist in the preference controls dictionary //
<       if Ctr.PrefControls[Inf.Vals("PreferencesSelection").VI] == nil {return}
<
<       // if the previously selected preference tab exists in the pref controls dictionary //
<       if Ctr.PrefControls[PreviousPreferencesSelection] != nil
---
>       // update the board preview when in the board preferences tab //
>       Inf.Update("VB", VBItems:["UpdatePreview":Inf.VIIsEqual("PreferencesSelection", VI:21)])
>
>       // if the selected preference tab exists in the preference controls dictionary //
>       if Ctr.PrefControls[Inf.Vals("PreferencesSelection").VI] != nil
3853,3855c4453,4458
<           // hide the previous preferences selection //
<           Ctr.PrefControls[PreviousPreferencesSelection]!.UpdateState(true)
<       }
---
>           // if the previously selected preference tab exists in the pref controls dictionary //
>           if Ctr.PrefControls[PreviousPreferencesSelection] != nil
>           {
>               // hide the previous preferences selection //
>               Ctr.PrefControls[PreviousPreferencesSelection]!.UpdateState(true)
>           }
3857,3858c4460,4461
<       // display the current preferences selection //
<       Ctr.PrefControls[Inf.Vals("PreferencesSelection").VI]!.UpdateState(false)
---
>           // display the current preferences selection //
>           Ctr.PrefControls[Inf.Vals("PreferencesSelection").VI]!.UpdateState(false)
3860,3861c4463,4465
<       // save the previous preferences selection //
<       PreviousPreferencesSelection = Inf.Vals("PreferencesSelection").VI
---
>           // save the previous preferences selection //
>           PreviousPreferencesSelection = Inf.Vals("PreferencesSelection").VI
>       }
3870,3871c4474,4479
<       // update the preference previews //
<       UpdatePreviews(true, Diagram:true)
---
>       // create the smaller diagram image //
>       RefreshBoardAndPieces(0, UpdateBoard:true, Display:false)
>
>       // update the board and diagram previews //
>       Ctr.UpdateBoardPreviews(BPComp.GetNSImage(400, Set:2), DiagramImage:BPComp.GetNSImage(Inf
>                   .Vals("IllustrationSize").VI, Set:0))
3931c4539
<       for TheItem in MainToolbar.items {TheItem.isEnabled = true}
---
>       for Index in 0 ..< MainToolbar.items.count {MainToolbar.items[Index].isEnabled = true}
3960c4568
<       PositionLayers.Animate(LoadPosition, Image:PositionImages[LoadPosIndex])
---
>       PositionLayers.Animate(LoadPosition, Image:PositionImages[LoadPosIndex]!)
3973c4581
<       PositionLayers.Animate(LoadPosition, Image:SavedPositionImages[LoadPosIndex])
---
>       PositionLayers.Animate(LoadPosition, Image:SavedPositionImages[LoadPosIndex]!)
3994c4602,4608
<
---
>
>
>           // play the sound of a warning message //
>           GameSounds.Play("warning")
>
>           // set the alert button to be okay //
>           AlertButtons = ["Okay"]
3996c4610
<           MakeAlert(["Okay"], Query:"No Saved Boards")
---
>           MakeAlert(AlertButtons, Query:"No Saved Boards")
4002,4004d4615
<       // get the position images //
<       GetPositionImages()
<
4006c4617
<       PositionLayers.UpdateLayer(PositionImages[LoadPosIndex])
---
>       PositionLayers.UpdateLayer(SavedPositionImages[LoadPosIndex])
4016a4628,4633
>
>       // update the small sized board //
>       RefreshBoardAndPieces(2, UpdateBoard:false, Display:false)
>
>       // save the snapshot image to game attributes //
>       Attributes.PositionImage = BPComp.GetNSImage(0)
4026c4643,4646
<
---
>
>
>       // append the snapshot image to saved position images //
>       SavedPositionImages.append(BPComp.GetNSImage(0))
4034c4654
<           Inf.Update("VIPref", VIItems:["SavedPositionSlot":Inf.Vals("SavedPositionSlot").VI % 25 +
---
>           Inf.Update("VIPref", VIItems:["SavedPositionSlot":Inf.Vals("SavedPositionSlot").VI % 5 +
4040c4660,4663
<
---
>
>
>       // set the alert button to be okay //
>       AlertButtons = ["Okay"]
4042c4665
<       MakeAlert(["Okay"], Query:"Saved Board")
---
>       MakeAlert(AlertButtons, Query:"Saved Board")
4047a4671,4673
>       // copy the saved position data into the game data //
>       DataCopy(GameData, SavedPositionData[LoadPosIndex])
>
4049c4675
<       StartNewGame()
---
>       StartNewGame(false)
4051,4052c4677,4678
<       // extract and display the game information from the game data list //
<       ExtractFromGameData(SavedPositionData[LoadPosIndex], Tasks:"All")
---
>       // extract and display the game information from the game data list //
>       ExtractFromGameData(true)
4088c4714
<       StartNewGame()
---
>       StartNewGame(true)
4111,4118c4737,4757
<       // get the index of the timed game opponent, color, and index //
<       (TimedGameOpponent, TimedGameColor, TimedGameTime) = (TimedGameOpponentButton
<       .indexOfSelectedItem, TimedGameColorButton.indexOfSelectedItem, TimedGameTimeButton
<       .indexOfSelectedItem)
<
<       // update the opponent time //
<       UpdateOpponent((TimedGameOpponent == 0) ? "GreenChess" : ((TimedGameOpponent == 1) ?
<       "Human" : "Stockfish"))
---
>       // get the index of the timed game opponent //
>       TimedGameOpponent = TimedGameOpponentButton.indexOfSelectedItem
>
>       // get the index of the timed game color button //
>       TimedGameColor = TimedGameColorButton.indexOfSelectedItem
>
>       // get the index of the timed game time control button //
>       TimedGameTime = TimedGameTimeButton.indexOfSelectedItem
>
>       // switch over the chosen opponent //
>       switch TimedGameOpponent
>       {
>           // set opponent to be the greenchess engine //
>           case 0 : UpdateOpponent("GreenChess")
>
>           // set opponent to be a human //
>           case 1 : UpdateOpponent("Human")
>
>           // default to the stockfish engine //
>           default : UpdateOpponent("Stockfish")
>       }
4123,4125c4762,4769
<       // calculate the timed game offset from the six o'clock position and the remaining times //
<       (TimedGameOffset, RemainingWhiteTime, RemainingBlackTime) = (216000 - TotalGameTime,
<       TotalGameTime, TotalGameTime)
---
>       // calculate the timed game offset from the six o'clock position //
>       TimedGameOffset = 216000 - TotalGameTime
>
>       // set the total amount of time remaining for white //
>       RemainingWhiteTime = TotalGameTime
>
>       // set the total amount of time remaining for black //
>       RemainingBlackTime = TotalGameTime
4140,4143c4784,4789
<       // remove the timed game offset from the elapsed times //
<       (ElapsedWhiteTime, ElapsedBlackTime) = (ElapsedWhiteTime - TimedGameOffset, ElapsedBlackTime
<       - TimedGameOffset)
<
---
>       // remove the timed game offset from the elapsed white time //
>       ElapsedWhiteTime -= TimedGameOffset
>
>       // remove the timed game offset from the elapsed black time //
>       ElapsedBlackTime -= TimedGameOffset
4159,4165c4805,4813
<       // loop over all the clocks //
<       for TheClock in [WhiteTimeClock, BlackTimeClock, Ctr.TimeClockPreview]
<       {
<           // set the clock to refresh //
<           TheClock!.RefreshAll = true
<       }
---
>       // set that need to refresh the white time clock //
>       WhiteTimeClock.RefreshAll = true
>
>       // set that need to refresh the black time clock //
>       BlackTimeClock.RefreshAll = true
>
>       // set that need to refresh the time clock preview //
>       Ctr.TimeClockPreview.RefreshAll = true
4173,4182c4821,4874
<       // set the window title //
<       MainWindow.title = "GreenChess" + (Inf.Vals("Diagramming").VB ?
<                       " (Game Explorer Diagram Mode)" : (Inf.Vals("EnteringFEN").VB ?
<                       " (FEN Converter)" : (Inf.Vals("InPositionEditor").VB ?
<                       " (Position Editor Mode)" : (Inf.Vals("InGameGallery").VB ? " (Gallery)" :
<                       (Inf.Vals("AnnotatingGame").VB ? " Game Annotator Mode)" :
<                       (Inf.Vals("PlayingTimedGame").VB ? " (Timed Game Mode)" :
<                       (Inf.Vals("GameExplorerWindowOpen").VB ? " (Game Explorer Mode)" :
<                       (Inf.Vals("AnalyzingGame").VB ? " (Game Explorer Analysis Mode)" :
<                       "")))))))
---
>       // if currently diagramming //
>       if Inf.Vals("Diagramming").VB
>       {
>           // set the main window title to game explorer diagram mode //
>           MainWindow.title = "GreenChess (Game Explorer Diagram Mode)"
>       }
>       // if entering fen string //
>       else if Inf.Vals("EnteringFEN").VB
>       {
>           // update the main window title to fen converter //
>           MainWindow.title = "GreenChess (FEN Converter)"
>       }
>       // if in the position editor //
>       else if Inf.Vals("InPositionEditor").VB
>       {
>           // update the main window title to position editor mode //
>           MainWindow.title = "GreenChess (Position Editor Mode)"
>       }
>       // if in the gallery //
>       else if Inf.Vals("InGameGallery").VB
>       {
>           // update the main window title to gallery mode //
>           MainWindow.title = "GreenChess (Gallery)"
>       }
>       // if annotating a game //
>       else if Inf.Vals("AnnotatingGame").VB
>       {
>           // update the main window title to game annotator mode //
>           MainWindow.title = "GreenChess (Game Annotator Mode)"
>       }
>       // if playing a timed game //
>       else if Inf.Vals("PlayingTimedGame").VB
>       {
>           // update the main window title to timed game mode //
>           MainWindow.title = "GreenChess (Timed Game Mode)"
>       }
>       // if the game explorer window is open //
>       else if Inf.Vals("GameExplorerWindowOpen").VB
>       {
>           // update the main window title to game explorer mode //
>           MainWindow.title = "GreenChess (Game Explorer Mode)"
>       }
>       // if analyzing a game //
>       else if Inf.Vals("AnalyzingGame").VB
>       {
>           // update the main window title to game explorer analysis mode //
>           MainWindow.title = "GreenChess (Game Explorer Analysis Mode)"
>       }
>       // if not in the database explorer and not searching game list //
>       else if !Inf.Vals("InDatabaseExplorer").VB && !Inf.Vals("SearchingGameList").VB
>       {
>           // update the main window title to greenchess //
>           MainWindow.title = "GreenChess"
>       }
4220,4223c4912,4916
<           // set pgn moves view to be empty and the text field values //
<           (PGNMoveAnnotatorView.stringValue, GameAnnotationTextField.isEditable,
<           GameAnnotationTextField.isSelectable, GameAnnotationTextField.string) = ("", false, false,
<           "There are currently no moves to annotate.")
---
>           // set pgn moves view to be empty //
>           PGNMoveAnnotatorView.stringValue = ""
>
>           // print that there are currently no moves to annotate //
>           GameAnnotationTextField.string = "There are currently no moves to annotate."
4225,4226c4918,4919
<           // update the left annotation field to the first frame //
<           AnnotationLayers.UpdateLayer(0, ImageFrames[0])
---
>           // make the game annotation text field editing temporarily unavailable //
>           GameAnnotationTextField.isEditable = false
4228,4229c4921,4922
<           // update the middle annotation layer to the first frame //
<           AnnotationLayers.UpdateLayer(1, ImageFrames[0])
---
>           // make the game annotation text field selection temporarily unavailable //
>           GameAnnotationTextField.isSelectable = false
4234,4236c4927,4932
<           // make the game annotation text field editable and selectable //
<           (GameAnnotationTextField.isEditable, GameAnnotationTextField.isSelectable) = (true, true)
<
---
>           // make the game annotation text field editable //
>           GameAnnotationTextField.isEditable = true
>
>           // make the game annotation text field selectable //
>           GameAnnotationTextField.isSelectable = true
4312,4313c5008,5009
<       // refresh all //
<       Refresh("All")
---
>       // replay all the moves //
>       ReplayMoves(true)
4421,4422c5117,5121
<       // initialize number of left and right parentheses to zero //
<       (LeftParentheses, RightParentheses) = (0, 0)
---
>       // initialize number of left parentheses to zero //
>       LeftParentheses = 0
>
>       // initialize number of right parentheses to zero //
>       RightParentheses = 0
4425c5124
```

```
<       for Index in 0..<AnnotationString.count
---
>       for Index in 0 ..< AnnotationString.count
4437,4439c5136,5140
<          // remove all the left and right parentheses from the move annotation //
<          AnnotationString = (AnnotationString.replacingOccurrences(of:"(", with:""))
<                               .replacingOccurrences(of:")", with:"")
---
>          // remove all the left parentheses from the move annotation //
>          AnnotationString = AnnotationString.replacingOccurrences(of:"(", with:"")
>
>          // remove all the right parentheses from the move annotation //
>          AnnotationString = AnnotationString.replacingOccurrences(of:")", with:"")
4441,4444c5142,5147
<
<       // remove all the left and right brackets //
<       AnnotationString = (AnnotationString.replacingOccurrences(of:"{", with:""))
<                               .replacingOccurrences(of:"}", with:"")
---
>
>
>       // remove all the left brackets //
>       AnnotationString = AnnotationString.replacingOccurrences(of:"{", with:"")
>
>       // remove all the right brackets //
>       AnnotationString = AnnotationString.replacingOccurrences(of:"}", with:"")
4469,4474c5172,5178
<       // return if didn't read in the game database already //
<       if !Inf.Vals("ReadInGameList").VB {return}
<
<       // update the database game explorer image //
<       UpdateExplorerImage(DatabaseGameExplorerSlider, TextView:PGNMoveDatabase, ImageView:
<                           DatabaseExplorerImage, Shift:0)
---
>       // if read in the game database already //
>       if Inf.Vals("ReadInGameList").VB
>       {
>          // update the database game explorer image //
>          UpdateExplorerImage(DatabaseGameExplorerSlider, TextView:PGNMoveDatabase, ImageView:
>                              DatabaseExplorerImage, Shift:0)
>       }
4487a5192,5194
>       // get the selected game to display //
>       GameData = DatabaseGetGameData(GameData, GameList, CInt(DatabaseExplorerSlider.intValue))
>
4489,4490c5196
<       ExtractFromGameData(DatabaseGetGameData(GameData, GameList, CInt(DatabaseExplorerSlider
<                           .intValue)), Tasks:"Replay")
---
>       ExtractFromGameData(false)
4527c5233,5236
<       // update the game annotation string //
---
>       // update the game annotator slider to the new move location //
>       GameAnnotatorSlider.intValue = Int32(MoveLocation)
>
>       // get the annotation string to display //
4529c5238
<
---
>
4532a5242,5244
>       // send the current move annotation to the game annotator text field //
>       GameAnnotationTextField.string = AnnotationString
>
4536,4538c5248,5249
<       // update the game annotation slider, text field, and the pgn move view //
<       (GameAnnotatorSlider.intValue, GameAnnotationTextField.string, PGNMoveAnnotatorView
<       .stringValue) = (Int32(MoveLocation), AnnotationString, PGNMove)
---
>       // if the pgn move string index is within range //
>       PGNMoveAnnotatorView.stringValue = PGNMove
4561c5272
<    func UpdateExplorerImage(_ Slider:NSSlider, TextView:NSTextField, ImageView:BoardImageView,
---
>    func UpdateExplorerImage(_ Slider:NSSlider, TextView:NSTextField, ImageView:GameExplorerView,
4567a5279,5281
>       // process the illustration for set 1 //
>       ProcessIllustration(Int(Slider.intValue), Set:1)
>
4569c5283
<       ImageView.Update(FrameCGImage(Int(Slider.intValue), Size:450, UpdateBoard:false, Table:false))
---
>       ImageView.Update(BPComp.GetCGImage(1))
4616,4621c5330,5337
<       // loop over all the clocks //
<       for TheClock in [WhiteTimeClock, BlackTimeClock, Ctr.TimeClockPreview]
<       {
<          // update the clock background visibility //
<          TheClock!.UpdateClockBackgroundOpacity(Inf.Vals("ClockBackgroundVisibility").VI)
<       }
---
>       // update the white time clock background color opacity //
>       WhiteTimeClock.UpdateClockBackgroundOpacity(Inf.Vals("ClockBackgroundVisibility").VI)
>
>       // update the black time clock background color opacity //
>       BlackTimeClock.UpdateClockBackgroundOpacity(Inf.Vals("ClockBackgroundVisibility").VI)
>
>       // update the time clock preview background opacity //
>       Ctr.TimeClockPreview.UpdateClockBackgroundOpacity(Inf.Vals("ClockBackgroundVisibility").VI)
4634,4635c5350,5351
<       // update the window background color from the current window color //
<       UpdateWindowBackgroundColor()
---
>       // set the window background color from the current window color //
>       SetWindowBackgroundColor(true)
4668,4672c5382,5386
<       // update the preference previews //
<       UpdatePreviews(false, Diagram:true)
<
<       // refresh all //
<       Refresh("All")
---
>       // update the board and diagram previews //
>       Ctr.UpdateBoardPreviews(BPComp.GetNSImage(400, Set:2), DiagramImage:BPComp.GetNSImage(Inf
>                              .Vals("IllustrationSize").VI, Set:0))
4767,4771c5481,5487
<       // update the illustration size view string and the preview frame //
<       (Ctr.IllustrationSizeView.stringValue, Ctr.DiagramPreview.frame) = ("Diagram size is " +
<       Inf.IntStr("IllustrationSize"), CGRect(x:505 - Inf.Fraction(2, Key:"IllustrationSize"), y:490
<       - Inf.Fraction(2, Key:"IllustrationSize"), width:Inf.VF("IllustrationSize"), height:Inf
<       .VF("IllustrationSize")))
---
>       // update the illustration size view //
>       Ctr.IllustrationSizeView.stringValue = "Diagram size is " + Inf.IntStr("IllustrationSize")
>
>       // update the diagram preview frame //
>       Ctr.DiagramPreview.frame = CGRect(x:505 - Inf.Fraction(2, Key:"IllustrationSize"), y:490 -
>                                         Inf.Fraction(2, Key:"IllustrationSize"), width:Inf
>                                         .VF("IllustrationSize"), height:Inf.VF("IllustrationSize"))
4834,4836c5550,5561
<       // update the opponent toolbar item image //
<       OpponentToolbarItem.image = (Opponent == "Human") ? HumanOpponentIcon : ((Opponent ==
<       "GreenChess") ? GreenChessOpponentIcon : UCIEngineOpponentIcon)
---
>       // switch over the opponent //
>       switch Opponent
>       {
>          // if playing against a human //
>          case "Human" : OpponentToolbarItem.image = HumanOpponentIcon
>
>          // if playing against the greenchess engine //
>          case "GreenChess" : OpponentToolbarItem.image = GreenChessOpponentIcon
>
>          // default to the stockfish engine //
>          default : OpponentToolbarItem.image = UCIEngineOpponentIcon
>       }
4842,4846c5567,5575
<       // update the opponent states //
<       (HumanOpponentMenuItem.state, EngineOpponentMenuItem.state, UCIEngineOpponentMenuItem.state) =
<       (Ctr.UpdatedState(Inf.Vals("HumanOpponent").VB), Ctr.UpdatedState(Inf.Vals("EngineOpponent")
<       .VB), Ctr.UpdatedState(Inf.Vals("UCIEngineOpponent").VB))
<
---
>       // update the state of the human opponent menu item //
>       HumanOpponentMenuItem.state = Ctr.UpdatedState(Inf.Vals("HumanOpponent").VB)
>
>       // update the state of the greenchess opponent menu item //
>       EngineOpponentMenuItem.state = Ctr.UpdatedState(Inf.Vals("EngineOpponent").VB)
>
>       // update the state of chess engine menu item //
>       UCIEngineOpponentMenuItem.state = Ctr.UpdatedState(Inf.Vals("UCIEngineOpponent").VB)
>
4901c5630
<    func RefreshTableViews()
---
>    func PrintGameInfo(_ Display:Bool)
4903,4909c5631
<       // update the move list font //
<       MoveListFont.UpdateFont(Inf.Vals("IllustrationSize").VI, UseVertOffset:!Inf
<                               .Vals("HideIllustrations").VB)
<
<       // update the game info font //
<       GameInfoFont.UpdateFont(0, UseVertOffset:false)
<
4916,4923c5638,5651
<       // set the current game state //
<       GameState = (BoardIsCheckmate(Board) && BoardIsWhite(Board)) ? 1 : ((BoardIsCheckmate(Board)
<                   && BoardIsBlack(Board)) ? 2 : (BoardIsStalemate(Board) ? 3 :
<                   (BoardIsThreefoldRepetition(Board) ? 4 : (BoardInsufficientMaterialToMate(Board)
<                   ? 7 : 0))))
<
<       // if the current game state is zero //
<       if GameState == 0
---
>       // if have checkmate and white to move next //
>       if BoardIsCheckmate(Board) && BoardIsWhite(Board)
>       {
>          // set current game state to show that black won //
>          CurrentGameState = 1
>       }
>       // if have checkmate and black to move next //
>       else if BoardIsCheckmate(Board) && BoardIsBlack(Board)
>       {
>          // set current game state to show that white won //
>          CurrentGameState = 2
>       }
>       // if have stalemate //
>       else if BoardIsStalemate(Board)
4924a5653,5673
>          // set current game state to show that had a stalemate //
>          CurrentGameState = 3
>       }
>       // if have threefold repetition //
>       else if BoardIsThreefoldRepetition(Board)
>       {
>          // set current game state to show that detected draw by threefold repetition //
>          CurrentGameState = 4
>       }
>       // if have insufficient material to checkmate with //
>       else if BoardInsufficientMaterialToMate(Board)
>       {
>          // set current game state to show that detected draw by insufficient material //
>          CurrentGameState = 7
>       }
>       // game has not ended yet //
>       else
>       {
>          // set the current game state to show that game not over yet //
>          CurrentGameState = 0
>
4937c5686,5690
<       if GameInfDict[GameState] != nil {GameInfoList.append(GameInfDict[GameState]!)}
---
>       if GameInfoDictionary[CurrentGameState] != nil
>       {
>          // append the current game state from the game info dictionary //
>          GameInfoList.append(GameInfoDictionary[CurrentGameState]!)
>       }
4940a5694,5696
>
>       // display game information to gui //
>       UpdateGameInfoList(Display)
4956c5712
<       for Index in 0..<Attributes.Inf.AnalysisCount()
---
>       for Index in 0 ..< Attributes.Inf.AnalysisCount()
4968c5724
<             for GlyphIndex in 0..<GlyphCode.count
---
>             for GlyphIndex in 0 ..< GlyphCode.count
4996c5752
<       // reset the stockish analysis string to empty //
---
>       // resent the stockish analysis string to empty //
4999,5015c5755,5756
<       // refresh the pgn moves list //
<       PGNMovesData.RefreshPGNMovesList(PGNMovesList, MoveListFont:MoveListFont)
<
<       // refresh the game info list //
<       GameInfoData.RefreshGameInfoList(GameInfoList, GameInfoFont:GameInfoFont)
<
<       // reload the game data to the pgn moves table view //
<       PGNMovesTableView.reloadData()
<
<       // reload the game info list //
<       GameInfoTableView.reloadData()
<
<       // scroll to the last row of the pgn moves table view //
<       PGNMovesTableView.scrollRowToVisible(PGNMovesData.LastRow())
<
<       // scroll to the first row in the game info table view //
<       GameInfoTableView.scrollRowToVisible(0)
---
>       // update the move list //
>       UpdateMoveList(Display)
5037a5779,5807
>    // update the game info list //
>    func UpdateGameInfoList(_ Display:Bool)
>    {
>       // refresh the game info list //
>       GameInfoData.RefreshGameInfoList(GameInfoList, GameInfoFont:GameInfoFont)
>
>       // if will display to the screen //
>       if Display{GameInfoTableView.reloadData(); GameInfoTableView.scrollRowToVisible(0)}
>    }
>
>    // update the move list //
>    func UpdateMoveList(_ Display:Bool)
>    {
>       // refresh the pgn moves list //
>       PGNMovesData.RefreshPGNMovesList(PGNMovesList, MoveListFont:MoveListFont,
>                                        DisplayIllustrations:!Inf.Vals("HideIllustrations").VB,
>                                        IllustrationSize:Inf.Vals("IllustrationSize").VI)
>
>       // if will display to the screen //
>       if Display
>       {
>          // reload the game data to the pgn moves table view //
>          PGNMovesTableView.reloadData()
>
>          // scroll the pgn moves table view to the last move made //
>          PGNMovesTableView.scrollRowToVisible(PGNMovesData.LastRow())
>       }
>    }
>
5071,5075c5841,5857
<       // set the position editor moving piece and the position editor cursor image //
<       (PositionMovingPiece, PosCursor) = (Type, (Type == 0) ? NSCursor.init(image:Eraser, hotSpot:
<       EditorHotspotPoint) : ((Type == 10) ? NSCursor.init(image:PositionEditorEP.image!, hotSpot:
<       EditorHotspotPoint) : NSCursor.init(image:BPComp.GetPieceNSImage(Type), hotSpot:
<       EditorHotspotPoint)))
---
>       // set the position editor moving piece to chosen type //
>       PositionMovingPiece = Type
>
>       // switch over the piece type //
>       switch Type
>       {
>          // set the position cursor to the eraser icon //
>          case 0 : PosCursor = NSCursor.init(image:Eraser, hotSpot:EditorHotspotPoint)
>
>          // set the position cursor to the en passant icon //
>          case 10 : PosCursor = NSCursor.init(image:PositionEditorEP.image!, hotSpot:
>                                              EditorHotspotPoint)
>
>          // default to a piece icon //
>          default : PosCursor = NSCursor.init(image:BPComp.GetPieceNSImage(Type), hotSpot:
>                                              EditorHotspotPoint)
>       }
5079c5861
<    @IBAction func PositionEditorEraser(_:AnyObject) {GameSounds.Play("click"); SetCursor(0)}
---
>    @IBAction func PositionEditorEraserAction(_:AnyObject) {GameSounds.Play("click"); SetCursor(0)}
5119,5123c5901,5905
<       // return if the selected items isn't in the position pieces dictionary //
<       if PositionPieces[MainToolbar!.selectedItemIdentifier!.rawValue] == nil {return}
<
<       // set the position cursor to the current piece type //
<       SetCursor(PositionPieces[MainToolbar!.selectedItemIdentifier!.rawValue]!)
---
>       // if the selected items is in the position pieces dictionary //
>       if PositionPieces[MainToolbar!.selectedItemIdentifier!.rawValue] != nil
>       {
>          // set the position cursor to the current piece type //
>          SetCursor(PositionPieces[MainToolbar!.selectedItemIdentifier!.rawValue]!)
5125,5126c5907,5909
<       // play the game sound of a click //
<       GameSounds.Play("click")
---
>          // play the game sound of a click //
>          GameSounds.Play("click")
>       }
5135,5140c5918,5935
<       // set the castle selection //
<       CastSel = MainToolbar!.selectedItemIdentifier!.rawValue
<
<       // toggle the corresponding castle selection //
<       Inf.ToggleVB("VB", Key:((CastSel == "217") ? "WhiteKing" : ((CastSel == "218") ?
<          "WhiteQueen" : ((CastSel == "219") ? "BlackKing" : "BlackQueen"))) + "sideCastle")
---
>       // switch over the main toolbar items for castling options //
>       switch MainToolbar!.selectedItemIdentifier!.rawValue
>       {
>          // toggle white can kingside castle option //
>          case "217" : Inf.ToggleVB("VB", Key:"WhiteKingsideCastle")
>
>          // toggle white can queenside castle option //
>          case "218" : Inf.ToggleVB("VB", Key:"WhiteQueensideCastle")
>
>          // toggle black can kingside castle option //
>          case "219" : Inf.ToggleVB("VB", Key:"BlackKingsideCastle")
>
>          // toggle black can queenside castle option //
>          case "220" : Inf.ToggleVB("VB", Key:"BlackQueensideCastle")
>
>          // the default case //
>          default : break
>       }
5148,5150d5942
<
<    // toggle the preference //
<    func TogglePref(_ Key:String) {GameSounds.Play("click"); Inf.ToggleVB("VBPref", Key:Key)}
5153c5945,5952
<    @IBAction func MultithreadUCIEngineButton(_:AnyObject) {TogglePref("MultithreadUCIEngine")}
---
>    @IBAction func MultithreadUCIEngineButtonAction(_:AnyObject)
>    {
>       // play the sound of a button click //
>       GameSounds.Play("click")
>
>       // toggle the multithread uci engine state //
>       Inf.ToggleVB("VBPref", Key:"MultithreadUCIEngine")
>    }
5156c5955,5965
<    @IBAction func AutosaveGame(_:AnyObject) {TogglePref("AutosaveGame"); StoreAutosaveData("")}
---
>    @IBAction func AutosaveGameButtonAction(_:AnyObject)
>    {
>       // play the sound of a button click //
>       GameSounds.Play("click")
>
>       // switch the autosave game state //
>       Inf.ToggleVB("VBPref", Key:"AutosaveGame")
>
>       // store the autosave data //
>       StoreAutosaveData("")
>    }
5159c5968,5975
<    @IBAction func EcoBookButtonAction(_:AnyObject) {TogglePref("UseEcoBook")}
---
>    @IBAction func EcoBookButtonAction(_:AnyObject)
>    {
>       // play the sound of a button click //
>       GameSounds.Play("click")
>
>       // switch whether currently using the chess opening book or not //
>       Inf.ToggleVB("VBPref", Key:"UseEcoBook")
>    }
5162c5978,5985
<    @IBAction func DisplayIllus(_:AnyObject) {TogglePref("HideIllustrations"); Refresh("All")}
---
>    @IBAction func DisplayIllustrationsAction(_:AnyObject)
>    {
>       // play the sound of a button click //
>       GameSounds.Play("click")
>
>       // switch the display illustrations state //
>       Inf.ToggleVB("VBPref", Key:"HideIllustrations")
>    }
5239,5240c6062,6064
<                            .Vals("GalleryAnimationRate").VI) / 35, target:self, selector:
<                            #selector(GalleryGoRight), userInfo:nil, repeats:true)
---
>                                           .Vals("GalleryAnimationRate").VI) / 35,
>                                           target:self, selector:#selector(GalleryGoRight),
>                                           userInfo:nil, repeats:true)
5284,5285c6108,6110
<                         .Vals("AnimationRate").VI) / 20, target:self, selector:
<                         #selector(GameExplorerGoRight), userInfo:nil, repeats:true)
---
>                                       .Vals("AnimationRate").VI) / 20, target:self,
>                                       selector:#selector(GameExplorerGoRight), userInfo:
>                                       nil, repeats:true)
5296c6121
<    func SetCastLabel(_ ToolbarItem:NSToolbarItem, On:Bool, Kingside:Bool)
---
>    func UpdateCastleItemLabel(_ ToolbarItem:NSToolbarItem, On:Bool, Kingside:Bool)
5311,5313c6136,6140
<          // update the white moves toolbar button image and label //
<          (WhiteMovesToolbarButton.image, WhiteMovesToolbarButton.label) = (BPComp
<          .GetPieceNSImage(1), "White Moves")
---
>          // update the white moves toolbar button image //
>          WhiteMovesToolbarButton.image = BPComp.GetPieceNSImage(1)
>
>          // set the white moves toolbar button label to "white moves" //
>          WhiteMovesToolbarButton.label = "White Moves"
5318,5320c6145,6149
<          // update the white moves toolbar button image and label //
<          (WhiteMovesToolbarButton.image, WhiteMovesToolbarButton.label) = (BPComp
<          .GetPieceNSImage(-1), "Black Moves")
---
>          // update the white moves toolbar button image //
>          WhiteMovesToolbarButton.image = BPComp.GetPieceNSImage(-1)
>
>          // update the white moves toolber button label //
>          WhiteMovesToolbarButton.label = "Black Moves"
5324c6153
<       for TheItem in MainToolbar.items {if PositionPieces[TheItem.itemIdentifier.rawValue] != nil
---
>       for Index in 0 ..< MainToolbar.items.count
5326,5328c6155,6162
<          // update the item image to the correct piece set and piece type //
<          TheItem.image = BPComp.GetPieceNSImage(PositionPieces[TheItem.itemIdentifier.rawValue]!)
<       }}
---
>          // if the toolbar item is in the position pieces dictionary //
>          if PositionPieces[MainToolbar.items[Index].itemIdentifier.rawValue] != nil
>          {
>             // update the item image to the correct piece set and piece type //
>             MainToolbar.items[Index].image = BPComp.GetPieceNSImage(PositionPieces[MainToolbar
>                                              .items[Index].itemIdentifier.rawValue]!)
>          }
>       }
5331c6165,6166
<       SetCastLabel(WhiteKingsideCastle, On:Inf.Vals("WhiteKingsideCastle").VB, Kingside:true)
---
>       UpdateCastleItemLabel(WhiteKingsideCastleToolbarButton, On:Inf.Vals("WhiteKingsideCastle").VB,
>                             Kingside:true)
5334c6169,6170
<       SetCastLabel(WhiteQueensideCastle, On:Inf.Vals("WhiteQueensideCastle").VB, Kingside:false)
---
>       UpdateCastleItemLabel(WhiteQueensideCastleToolbarButton, On:Inf.Vals("WhiteQueensideCastle")
>                             .VB, Kingside:false)
5337c6173,6174
<       SetCastLabel(BlackKingsideCastle, On:Inf.Vals("BlackKingsideCastle").VB, Kingside:true)
---
>       UpdateCastleItemLabel(BlackKingsideCastleToolbarButton, On:Inf.Vals("BlackKingsideCastle").VB,
>                             Kingside:true)
5340c6177,6178
<       SetCastLabel(BlackQueensideCastle, On:Inf.Vals("BlackQueensideCastle").VB, Kingside:false)
---
>       UpdateCastleItemLabel(BlackQueensideCastleToolbarButton, On:Inf.Vals("BlackQueensideCastle")
>                             .VB, Kingside:false)
5351c6189,6192
<
---
>
>
>       // create the buttons for the start new game query //
>       AlertButtons = ["Yes", "No"]
5353c6194
<       MakeAlert(["Yes", "No"], Query:"New Game?")
---
>       MakeAlert(AlertButtons, Query:"New Game?")
5448c6289
<             StartNewGame()
---
>             StartNewGame(true)
5523a6365,6367
>
>       // set the alert buttons //
>       AlertButtons = ["Quit", "Cancel"]
5526c6370
<       MakeAlert(["Quit", "Cancel"], Query:"Quit GreenChess?")
---
>       MakeAlert(AlertButtons, Query:"Quit GreenChess?")
5532,5536c6376,6380
<       // return if not undoing to the previous move //
<       if !Attributes.Inf.UndoToPreviousVS() {return}
<
<       // refresh all //
<       Refresh("All")
---
>       // undo to the previous move //
>       if Attributes.Inf.UndoToPreviousVS()
>       {
>          // replay all the moves excluding the last move //
>          ReplayMoves(true)
5538,5539c6382,6383
<       // compact the transposition table //
<       EngineCompactTransTable(Engine, false)
---
>          // compact the transposition table //
>          EngineCompactTransTable(Engine, false)
5541,5542c6385,6387
<       // store the autosave game data //
<       StoreAutosaveData("")
---
>          // store the autosave game data //
>          StoreAutosaveData("")
>       }
5547a6393,6395
>       // if in the game gallery or annotating a game //
>       Set = (Inf.Vals("InGameGallery").VB || Inf.Vals("AnnotatingGame").VB) ? 0 : 1
>
5559a6408,6410
>          // process the illustration for the current frame number //
>          ProcessIllustration(FrameNumber, Set:Set)
>
5564,5565c6415,6416
<          if Inf.Vals("AnnotatingGame").VB {Frames.append(FrameCGImage(FrameNumber, Size:250,
<             UpdateBoard:false, Table:false))}
---
>          if Inf.Vals("AnnotatingGame").VB {Frames.append(BPComp.GetCGImage(Set))}
5568,5569c6418
<          if Inf.Vals("InGameGallery").VB {GameGalleryView.AddFrame(FrameCGImage(FrameNumber, Size:
<                      GameGalleryView.FrameSize * 0.5, UpdateBoard:false, Table:false))}
---
>          if Inf.Vals("InGameGallery").VB {GameGalleryView.AddFrame(BPComp.GetCGImage(Set))}
5583c6432
<       for Index in 0..<Attributes.Inf.AnalysisCount()
---
>       for Index in 0 ..< Attributes.Inf.AnalysisCount()
5613c6462
<       for Index in 0..<Attributes.Inf.GetCount() {MakeMoveOnBoard(Attributes.Inf.Vals(Index).VS)}
---
>       for Index in 0 ..< Attributes.Inf.GetCount() {MakeMoveOnBoard(Attributes.Inf.Vals(Index).VS)}
5615c6464
<
---
>
5617c6466
<    func ReplayMoves()
---
>    func ReplayMoves(_ Display:Bool)
5623c6472
<       for Index in 0..<Attributes.Inf.GetCount()
---
>       for Index in 0 ..< Attributes.Inf.GetCount()
5639c6488,6491
<
---
>
>
>          // perform castling and enpassant move if had one //
>          PerformCastleEnpassantIfNeeded()
5642a6495,6500
>
>       // if will display to the screen, refresh the board and pieces for main sized board //
>       if Display {RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)}
>
>       // print the game info //
>       PrintGameInfo(Display)
5645,5665c6503,6504
<    // get the board and pieces frame as an nsimage //
<    func FrameNSImage(_ Index:Int, Size:CGFloat, UpdateBoard:Bool, Table:Bool) -> NSImage
<    {
<       // return the board and pieces frame as an nsimage //
<       return CoreGraphics().CGImageToNSImage(FrameCGImage(Index, Size:Size, UpdateBoard:
<                                              UpdateBoard, Table:Table), Size:Size)
<    }
<
<    // get the board and pieces frame as a cgimage //
<    func FrameCGImage(_ Index:Int, Size:CGFloat, UpdateBoard:Bool, Table:Bool) -> CGImage
<    {
<       // return the board and pieces frame as a cgimage //
<       return BPComp.Recomposite(Size, BPData:((Attributes.Inf.GetCount() == Index) && !Table) ?
<                     BPData : PGNMovesData.GetData(Index), InPositionEditor:Inf
<                     .Vals("InPositionEditor").VB, FENEnpassantSquare:Attributes.Inf
<                     .Vals("FENEnpassantSquare").VS, EnpassantSquare:EnpassantSquare,
<                     RankAndFile:MovingPiece.Location, UpdateBoard:UpdateBoard)
<    }
<
<    // update the board preview images //
<    func UpdatePreviews(_ Board:Bool, Diagram:Bool)
---
>    // process an illustration //
>    func ProcessIllustration(_ IllusIndex:Int, Set:Int)
5667,5684c6506,6512
<       // refresh the board //
<       Refresh("Board")
<
<       // update the board preview if need to //
<       if Board
<       {
<          // update the board preview image //
<          Ctr.BoardPreview.image = FrameNSImage(Attributes.Inf.GetCount(), Size:400, UpdateBoard:
<                                               false, Table:false)
<       }
<
<       // return if don't need to update the diagram preview //
<       if !Diagram {return}
<
<       // update the diagram preview image //
<       Ctr.DiagramPreview.image = FrameNSImage(Attributes.Inf.GetCount(), Size:Inf
<                                               .VF("IllustrationSize"), UpdateBoard:false, Table:
<                                               false)
---
>       // save the current illustration index //
>       IllustrationIndex = IllusIndex
>
>       // recompose the pieces into the context //
>       BPComp.Recomposite(Set, BPData:PGNMovesData.GetData(IllusIndex), InPositionEditor:false,
>                          FENEnpassantSquare:"", EnpassantSquare:APoint(), RankAndFile:APoint(),
>                          UpdateBoard:false)
5694,5695c6522,6524
<       BPData.UpdateCaptureThreatLists(String(validatingUTF8:BoardGetListOfCaptureSquares(Board))!
<              .components(separatedBy:"*"), ThreatSquares:String(validatingUTF8:
---
>       BPData.UpdateCaptureThreatenHighlightLists(String(validatingUTF8:
>              BoardGetListOfCaptureSquares(Board))!.components(separatedBy:"*"),
>              ListOfThreatenSquares:String(validatingUTF8:
5701,5703c6530,6532
<
<    // refresh the graphics //
<    func Refresh(_ Tasks:String)
---
>
>    // refresh the board and pieces //
>    func RefreshBoardAndPieces(_ Set:Int, UpdateBoard:Bool, Display:Bool)
5705,5712c6534,6546
<       // if need to replay the moves //
<       if Tasks.contains("Replay") || Tasks.contains("All") {ReplayMoves()}
<
<       // if need to recomposite the board and refresh the view //
<       if Tasks.contains("Board") {RefreshView(true)}
<
<       // if just need to refresh the view //
<       if Tasks.contains("View") || Tasks.contains("All") {RefreshView(false)}
---
>       // recomposite the pieces onto the board //
>       BPComp.Recomposite(Set, BPData:BPData, InPositionEditor:Inf.Vals("InPositionEditor").VB,
>                          FENEnpassantSquare:Attributes.Inf.Vals("FENEnpassantSquare").VS,
>                          EnpassantSquare:EnpassantSquare, RankAndFile:MovingPiece.Location,
>                          UpdateBoard:UpdateBoard)
>
>       // if need to update the preview and entering preferences //
>       if Inf.Vals("UpdatePreview").VB && Inf.Vals("EnteringPreferences").VB
>       {
>          // update the board and diagram previews //
>          Ctr.UpdateBoardPreviews(BPComp.GetNSImage(400, Set:2), DiagramImage:BPComp
>                                 .GetNSImage(Inf.Vals("IllustrationSize").VI, Set:0))
>       }
5714,5720d6547
<       // if need to refresh all the table views //
<       if Tasks.contains("Tables") || Tasks.contains("All") {RefreshTableViews()}
<    }
<
<    // refresh the main view //
<    func RefreshView(_ UpdateBoard:Bool)
<    {
5725,5726c6552
<          Sketch.SetBackground(FrameCGImage(Attributes.Inf.GetCount(), Size:660, UpdateBoard:false,
<                               Table:false))
---
>          Sketch.SetBackground(BPComp.GetCGImage(2))
5730,5732c6555
<
<          // return //
<          return
5734,5740c6557,6562
<
<       // update the main view image //
<       MainViewImage = FrameCGImage(Attributes.Inf.GetCount(), Size:660, UpdateBoard:UpdateBoard,
<                                    Table:false)
<
<       // update the display if need to //
<       display(self.bounds)
---
>
>       // update the display if needed //
>       else if Display
>       {
>          // update the display //
>          display(bounds)
>       }
```