```
2c2
< // Swift source code of main.swift Version 8.0.0 at greenchess.com //
---
> // Swift source code of main.swift Version 7.0.7 at greenchess.com //
23a24,31
>    // the game info at the corresponding row index in the table view //
>    func tableView(_ TableView:NSTableView, objectValueFor TableColumn:NSTableColumn?, row
>                   RowIndex:Int) -> Any?
>    {
>       // return the game info corresponding to the row index in the table view //
>       return GameInfoList[RowIndex]
>    }
>
48c56
<             Height = Int(GameInfoList[Index].size().height) + 5
---
>             Height = Int(GameInfoList[Index].size().height) + 2
61c69
<             Heights.append((Int(GameInfoList[Index].size().width / 200) + 1) * Height)
---
>             Heights.append((Int(GameInfoList[Index].size().width / 190) + 1) * Height)
84,89d91
<    // the paragraph style to use //
<    var ParagraphStyle:NSMutableParagraphStyle = NSMutableParagraphStyle()
<
<    // the maximum number of rows to display //
<    var MaxRows:Int = 0
<
101a104,111
>    // the pgn move at the corresponding row index in the table view //
>    func tableView(_ TableView:NSTableView, objectValueFor TableColumn:NSTableColumn?, row
>                   RowIndex:Int) -> Any?
>    {
>       // return the pgn move corresponding to the row index in the table view //
>       return PGNMovesList[RowIndex]
>    }
>
105,107d114
<       // set the paragraph alignment to be justified //
<       ParagraphStyle.alignment = .justified
<
123,124c130
<                             .foregroundColor:Color, NSAttributedStringKey.paragraphStyle:
<                             ParagraphStyle]))
---
>                             .foregroundColor:Color]))
130c136
<             Height = Int(PGNMovesList[Index].size().height) + 5
---
>             Height = Int(PGNMovesList[Index].size().height) + 2
132c138
<
---
>
143c149
<             Heights.append((Int(PGNMovesList[Index].size().width / 200) + 1) * Height)
---
>             Heights.append((Int(PGNMovesList[Index].size().width / 190) + 1) * Height)
158,159c164,165
<       // return the number of rows in the pgn moves list to display //
<       return min(PGNMovesList.count * 2, MaxRows)
---
>       // return the number of moves in the pgn moves list //
>       return PGNMovesList.count
334c340
<             Heights.append((Int(AttributedGameList[Index].size().width / 600) + 1) * Height)
---
>             Heights.append((Int(AttributedGameList[Index].size().width / 550) + 1) * Height)
338a345,363
>    // the game at the corresponding row index in the table view //
>    func tableView(_ TableView:NSTableView, objectValueFor TableColumn:NSTableColumn?, row
>                   RowIndex:Int) -> Any?
>    {
>       // if the selected row is equal to the row index make font color white //
>       if SelectedRow == RowIndex
>       {
>          // return the attributed string at the row index in the game list //
>          return NSAttributedString(string:GameList[RowIndex], attributes:[NSAttributedStringKey
>                                   .font:Font, NSAttributedStringKey.foregroundColor:NSColor.white])
>       }
>       // 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 //
>          return AttributedGameList[RowIndex]
>       }
>    }
>
2078,2104d2102
<    // if the game explorer window is open //
<    var GameExplorerWindowOpen:Bool = false
<
<    // if found any invalid moves //
<    var FoundInvalidMoves:Bool = false
<
<    // whether should close the main window or not //
<    var ShouldCloseMainWindow:Bool = false
<
<    // whether have stockfish analysis to show or not //
<    var StockfishAnalysisToDisplay:Bool = false
<
<    // the illustration size for the move list //
<    var IllustrationSize:CGFloat = 150
<
<    // the table row //
<    var TableRow:Int = 0
<
<    // a table cell view //
<    var TableCell:NSTableCellView = NSTableCellView()
<
<    // the max frame for the illustrations //
<    var MaxIllustrationFrame:Int = 0
<
<    // whether already called awake from nib //
<    var Awoke:Bool = false
<
2150,2152d2147
<    // the illustrations for the pgn moves list //
<    var Illustrations:[NSImage] = []
<
2165,2167d2159
<    // the sized board composite image //
<    var SizedBoardComposite:NSImage = NSImage()
<
2363,2364c2355,2356
<    // initialize the previously selected row in the game list table view //
<    var PreviouslySelectedRow:Int = 0
---
>    // initialize the previous selected row data //
>    var PreviousSelectedRow:Int = 0
3092c3084
<
```

```
---
>
3189,3191d3180
<    // a temporary move list //
<    var TempMoveList:[String] = []
<
4050,4062d4038
<       // if already called the awake from nib function //
<       if Awoke
<       {
<          // exit function since awake from nib was already called //
<          return
<       }
<       // if haven't called awake from nib yet //
<       else
<       {
<          // set that awoke from nib has been called //
<          Awoke = true
<       }
<
4251a4228,4230
>
>       // set the main window alpha value to zero //
>       MainWindow.alphaValue = 0
4262,4271c4241,4243
<
<       // if should close the main window //
<       if ShouldCloseMainWindow
<       {
<          // close the main window //
<          MainWindow.close()
<       }
<
<       // set that no longer need to close the main window //
<       ShouldCloseMainWindow = false
---
>
>       // set the main window alpha to one //
>       MainWindow.alphaValue = 1
4287,4295d4258
<
<       // remove all the frames //
<       Frames.removeAll()
<
<       // remove all the illustrations //
<       Illustrations.removeAll()
<
<       // set stockfish analysis to display to false //
<       StockfishAnalysisToDisplay = false
4300,4301c4263,4268
<       // replay all the moves //
<       ReplayMoves(0, End:MoveList.count, Display:false)
---
>       // if not in the position editor //
>       if !InPositionEditor
>       {
>          // replay all the moves //
>          ReplayMoves(0, End:MoveList.count, Display:false)
>       }
4306a4274,4276
>       // set the main window alpha value to one //
>       MainWindow.alphaValue = 1
>
4315a4286,4288
>
>       // set the main window alpha value to zero //
>       MainWindow.alphaValue = 0
4361,4460d4333
<
<    // return the view for the specified row in the table view //
<    func tableView(_ View:NSTableView, viewFor Column:NSTableColumn?, row Row:Int) -> NSView?
<    {
<       // if the current view is the pgn moves table view //
<       if View == PGNMovesTableView
<       {
<          // the modified table row to account for max rows //
<          TableRow = PGNMovesData.PGNMovesList.count * 2 - min(PGNMovesData.PGNMovesList.count * 2,
<                     PGNMovesData.MaxRows) + Row
<
<          // create a table cell //
<          TableCell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"PGN"),
<                                    owner:self) as! NSTableCellView
<
<          // if on an even row //
<          if TableRow % 2 == 0
<          {
<             // set the table cell text to empty //
<             TableCell.textField!.stringValue = ""
<
<             // if index is less than the illustrations count //
<             if TableRow / 2 < Illustrations.count
<             {
<                // set the table cell image view to the illustration //
<                TableCell.imageView!.image = Illustrations[TableRow / 2]
<             }
<             // if is the stockfish analysis string //
<             else if TableRow / 2 == PGNMovesList.count - 1 && StockfishAnalysisToDisplay
<             {
<                // set the table cell image view to the stockfish icon //
<                TableCell.imageView!.image = UCIEngineOpponentIcon
<             }
<             // index is out of bounds //
<             else
<             {
<                // set the table cell image view to be empty //
<                TableCell.imageView!.image = NSImage()
<             }
<          }
<          // if is an odd row //
<          else
<          {
<             // set the table cell text field attributed string value to the pgn move //
<             TableCell.textField!.attributedStringValue = PGNMovesData.PGNMovesList[TableRow / 2]
<
<             // set the table cell image view to be an empty image //
<             TableCell.imageView!.image = NSImage()
<          }
<
<          // return the cell //
<          return TableCell
<       }
<       // if is the game info table view //
<       else if View == GameInfoTableView
<       {
<          // create a table cell //
<          TableCell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"Info"),
<                                    owner:self) as! NSTableCellView
<
<          // set the table cell attributed string value //
<          TableCell.textField!.attributedStringValue = GameInfoData.GameInfoList[Row]
<
```

```
<          // return the table cell //
<          return TableCell
<       }
<       // if is the game list table view //
<       else if View == GameListTableView
<       {
<          // create a table cell //
<          TableCell = View.makeView(withIdentifier:NSUserInterfaceItemIdentifier(rawValue:"Games"),
<                                  owner:self) as! NSTableCellView
<
<          // 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 //
<             TableCell.textField!.attributedStringValue = NSAttributedString(string:GameListData
<                                                       .GameList[Row], attributes:
<                                                       [NSAttributedStringKey.font:GameListData
<                                                       .Font, NSAttributedStringKey
<                                                       .foregroundColor:NSColor.white])
<          }
<          // 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.AttributedGameList[Row]
<          }
<
<          // return the table cell //
<          return TableCell
<       }
<       // not a valid table view //
<       else
<       {
<          // return a nil value //
<          return nil
<       }
<    }
4465,4466c4338,4339
<       // if is the pgn moves table views //
<       if TableView == PGNMovesTableView
---
>       // if is the pgn moves table view and within array bounds //
>       if TableView == PGNMovesTableView && Row < PGNMovesList.count
4468,4493c4341,4342
<          // the modified table row to account for max rows //
<          TableRow = PGNMovesData.PGNMovesList.count * 2 - min(PGNMovesData.PGNMovesList.count * 2,
<                     PGNMovesData.MaxRows) + Row
<
<          // if is an even row //
<          if TableRow % 2 == 0
<          {
<             // if is the stockfish analysis string //
<             if TableRow / 2 == PGNMovesList.count - 1 && StockfishAnalysisToDisplay
<             {
<                // return the row height of 26 //
<                return 26
<             }
<             // if is an illustration //
<             else
<             {
<                // return the row size for a board illustration //
<                return IllustrationSize
<             }
<          }
<          // if an odd row //
<          else
<          {
<             // return the row height for the pgn move table to adjust for text annotations //
<             return min(CGFloat(PGNMovesData.RowHeight(TableRow / 2)), 660)
<          }
---
>          // return the row height for the pgn move table to adjust to annotation size //
>          return CGFloat(PGNMovesData.RowHeight(Row))
4495,4496c4344,4345
<       // if is the game info table view //
<       else if TableView == GameInfoTableView
---
>       // if is the game info table view and within array bounds //
>       else if TableView == GameInfoTableView && Row < GameInfoList.count
4498c4347
<          // return the row height for the game info to adjust to string size //
---
>          // return the row height for the game info table to adjust to string size //
4501,4507c4350
<       // if is the game list table view //
<       else if TableView == GameListTableView
<       {
<          // return the row height for the game list //
<          return CGFloat(GameListData.RowHeight(Row))
<       }
<       // not a valid table view //
---
>       // if is the game list view //
4510,4511c4353,4354
<          // return a row height of 30 //
<          return 30
---
>          // return the row height for the game list table view //
>          return CGFloat(GameListData.RowHeight(Row))
4518,4524d4360
<       // if not searching the game list //
<       if !SearchingGameList
<       {
<          // return since not searching the game list //
<          return
<       }
<
4529c4365
<       GameListTableView.reloadData(forRowIndexes:IndexSet(integer:PreviouslySelectedRow),
---
>       GameListTableView.reloadData(forRowIndexes:IndexSet(integer:PreviousSelectedRow),
4537c4373
<       PreviouslySelectedRow = GameListTableView.selectedRow
---
>       PreviousSelectedRow = GameListTableView.selectedRow
4757c4593
<
---
>
4927a4764,4766
>          // print the game information //
>          PrintGameInfo()
>
```

```
4933,4938d4771
<
<          // update the pieces layer for an illustration frame //
<          UpdatePiecesLayer(IllustrationSize)
<
<          // append the pieces layer as a frame //
<          Frames.append(PiecesLayer)
4948,4950d4780
<
<             // print the game information //
<             PrintGameInfo()
4979,4981d4808
<
<             // print the game information //
<             PrintGameInfo()
5029,5037d4855
<       // add a timer to run the hide info menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheHideInfoMenuItemAction), userInfo:nil, repeats:false), forMode:
<                          RunLoopMode.commonModes)
<    }
<
<    // the hide info menu item action //
<    @objc func TheHideInfoMenuItemAction()
<    {
5045,5053d4862
<       // add a timer to run the small info menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheSmallInfoMenuItemAction), userInfo:nil, repeats:false), forMode:
<                          RunLoopMode.commonModes)
<    }
<
<    // the small info menu item action //
<    @objc func TheSmallInfoMenuItemAction()
<    {
5061,5069d4869
<       // add a timer to run the large info menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheLargeInfoMenuItemAction), userInfo:nil, repeats:false), forMode:
<                          RunLoopMode.commonModes)
<    }
<
<    // the large info menu item action //
<    @objc func TheLargeInfoMenuItemAction()
<    {
5077,5085d4876
<       // add a timer the run the hide captured pieces in panel menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheHideCapturedPiecesInPanelMenuItemAction), userInfo:nil, repeats:
<                          false), forMode:RunLoopMode.commonModes)
<    }
<
<    // the hide captured pieces in panel menu item action //
<    @objc func TheHideCapturedPiecesInPanelMenuItemAction()
<    {
5099,5107d4889
<       // add a timer to run the hide game clock in panel menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheHideGameClockInPanelMenuItemAction), userInfo:nil, repeats:false),
<                          forMode:RunLoopMode.commonModes)
<    }
<
<    // the hide game clock in panel menu item action //
<    @objc func TheHideGameClockInPanelMenuItemAction()
<    {
5165,5204d4946
<    // the window should close delegate //
<    func windowShouldClose(_ Window:NSWindow) -> Bool
<    {
<       // if is the main window //
<       if Window == MainWindow
<       {
<          // set that should close the main window //
<          ShouldCloseMainWindow = true
<       }
<
<       // return that should close the main window //
<       return true
<    }
<
<    // the custom windows to use when entering full screen //
<    func customWindowsToEnterFullScreen(for Window:NSWindow) -> [NSWindow]?
<    {
<       // return the main window //
<       return [MainWindow]
<    }
<
<    // start the custom animation when entering full screen //
<    func window(_ Window:NSWindow, startCustomAnimationToEnterFullScreenWithDuration Duration:
<                TimeInterval)
<    {
<    }
<
<    // the custom windows to use when exit full screen //
<    func customWindowsToExitFullScreen(for Window:NSWindow) -> [NSWindow]?
<    {
<       // return the main window //
<       return [MainWindow]
<    }
<
<    // start the custom animation when exiting full screen //
<    func window(_ Window:NSWindow, startCustomAnimationToExitFullScreenWithDuration Duration:
<                TimeInterval)
<    {
<    }
<
5329c5071
<       PGNMovesTableView.backgroundColor = NSColor(red:1, green:1, blue:1, alpha:0.4)
---
>       PGNMovesTableView.backgroundColor = NSColor(red:1, green:1, blue:1, alpha:0.6)
5332c5074
<       GameInfoTableView.backgroundColor = NSColor(red:1, green:1, blue:1, alpha:0.4)
---
```

```
>       GameInfoTableView.backgroundColor = NSColor(red:1, green:1, blue:1, alpha:0.6)
5783,5791d5524
<       // add a timer to run the redo last move menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                           .TheRedoLastMoveMenuItemAction), userInfo:nil, repeats:false), forMode:
<                           RunLoopMode.commonModes)
<    }
<
<    // the redo last move menu item action //
<    @objc func TheRedoLastMoveMenuItemAction()
<    {
6269,6274d6001
<
<       // remove all the frames //
<       Frames.removeAll()
<
<       // remove all the illustrations //
<       Illustrations.removeAll()
7549a7277,7282
>
>       // deselect the previouly selected row if any //
>       GameListTableView.deselectRow(GameListTableView.selectedRow)
>
>       // store the selected row to the game data //
>       GameListData.StoreSelectedRow(GameListTableView.selectedRow)
7553c7286
<
---
>
7567,7569d7299
<          // initialize the previously selected row to zero //
<          PreviouslySelectedRow = 0
<
8395,8396c8125,8126
<    // create the sized board composite //
<    func CreateSizedBoardComposite(_ Width:CGFloat)
---
>    // create the gallery grid board composite //
>    func CreateGalleryGridBoardComposite(_ Width:Double)
8398,8405c8128,8142
<       // initialize the sized board composite //
<       SizedBoardComposite = NSImage(size:NSSize(width:Width, height:Width))
<
<       // lock the sized board composite focus//
<       SizedBoardComposite.lockFocus()
<
<       // draw the board composite into the sized board composite //
<       BoardComposite.draw(in:NSMakeRect(0, 0, Width, Width))
---
>       // initialize the gallery grid view board composite //
>       GalleryGridView.BoardComposite = NSImage(size:NSSize(width:CGFloat(Width), height:
>                                               CGFloat(Width)))
>
>       // lock the gallery grid view board composite //
>       GalleryGridView.BoardComposite.lockFocus()
>
>       // loop over all the board layers //
>       for Index in 0 ..< BoardLayers.count
>       {
>          // draw the board layer into the gallery grid view board composite //
>          BoardLayers[Index].draw(in:NSMakeRect(0, 0, CGFloat(Width), CGFloat(Width)), from:
>                                 NSZeroRect, operation:NSCompositingOperation.sourceOver, fraction:
>                                 CGFloat(BoardLayerOpacities[Index]))
>       }
8407,8408c8144,8145
<       // unlock the sized board composite focus //
<       SizedBoardComposite.unlockFocus()
---
>       // unlock the gallery grid view board composite //
>       GalleryGridView.BoardComposite.unlockFocus()
8486,8487c8223,8224
<       // loop over all nine board layers //
<       for _ in 0 ... 8
---
>       // loop over all ten board layers //
>       for _ in 0 ... 9
8504c8241
<       AppNameView.stringValue = "GreenChess 8.0.0"
---
>       AppNameView.stringValue = "GreenChess 7.0.7"
9017,9018c8754
<       if HighlightCapturedPieces && !ReviewingGame && !InPositionEditor && !CoreAnimating &&
<          Width != IllustrationSize
---
>       if HighlightCapturedPieces && !ReviewingGame && !InPositionEditor && !CoreAnimating
9030,9031c8766
<       if HighlightThreatenedPieces && !ReviewingGame && !InPositionEditor && !CoreAnimating &&
<          Width != IllustrationSize
---
>       if HighlightThreatenedPieces && !ReviewingGame && !InPositionEditor && !CoreAnimating
9043,9044c8778,8779
<       if (StartSquare != EndSquare) && ((HighlightLastMoveMade && !InPositionEditor) ||
<          CreatingGalleryGrid || Width == IllustrationSize)
---
>       if (StartSquare != EndSquare) && (HighlightLastMoveMade && !InPositionEditor ||
>          CreatingGalleryGrid)
9598,9603c9333,9334
<             // if the move is not an invalid move //
<             if MoveList[Index] != "Invalid Move"
<             {
<                // add the move made to the engine command //
<                EngineCommand += " \(MoveList[Index])"
<             }
---
>             // add the moves made to the engine command //
>             EngineCommand += " \(MoveList[Index])"
9915,9920d9645
<
<       // update the pieces layer for the illustrations //
<       UpdatePiecesLayer(IllustrationSize)
<
<       // append the pieces layer to the frames //
<       Frames.append(PiecesLayer)
10029c9754
<          BoardComposite.draw(in:CGRect(x:0, y:0, width:Width, height:Width))
---
>          BoardComposite.draw(in:self.bounds)
10032c9757
<          PiecesLayer.draw(in:CGRect(x:0, y:0, width:Width, height:Width))
---
```

```
>           PiecesLayer.draw(in:self.bounds)
10702,10710d10426
<        // add a timer to rotate the board //
<        RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .RotateTheBoard), userInfo:nil, repeats:false), forMode:RunLoopMode
<                          .commonModes)
<     }
<
<     // rotate the board //
<     @objc func RotateTheBoard()
<     {
10720,10724c10436,10437
<        // replay all the moves //
<        ReplayMoves(0, End:MoveList.count, Display:false)
<
<        // update the move list //
<        UpdateMoveList()
---
>        // update the board and captured pieces //
>        UpdateBoardAndCapturedPieces()
10901,10906c10614,10615
<           // if the move exists //
<           if MoveExists
<           {
<              // store the current move and annotation into the game data //
<              DataStoreMove(GameData, BoardPGNLabel(GameDataBoard), AnnotationList[Index])
<           }
---
>           // store the current move and annotation into the game data //
>           DataStoreMove(GameData, BoardPGNLabel(GameDataBoard), AnnotationList[Index])
11050a10760,10775
>        // run on the main thread //
>        DispatchQueue.main.async
>        {
>           // clear the search string //
>           self.SearchString = ""
>
>           // update the search string to the game list search box //
>           self.GameListSearchBox.stringValue = self.SearchString
>
>           // deselect the previously selected row in the game list table view //
>           self.GameListTableView.deselectRow(self.GameListTableView.selectedRow)
>
>           // store the selected row to the game data //
>           self.GameListData.StoreSelectedRow(self.GameListTableView.selectedRow)
>        }
>
11085,11087d10809
<           // initialize the previously selected row to zero //
<           PreviouslySelectedRow = 0
<
11112,11117d10833
<
<        // clear the search string //
<        SearchString = ""
<
<        // update the search string to the game list search box //
<        GameListSearchBox.stringValue = SearchString
11415,11423d11130
<        // add a timer to run the database explorer menu item action //
<        RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:
<                          #selector(GreenChess.TheDatabaseExplorerMenuItemAction), userInfo:nil,
<                          repeats:false), forMode:RunLoopMode.commonModes)
<     }
<
<     // the database explorer menu item action //
<     @objc func TheDatabaseExplorerMenuItemAction()
<     {
11448,11449c11155,11156
<           // update the move list //
<           UpdateMoveList()
---
>           // update the board //
>           UpdateBoardAndCapturedPieces()
11804,11806d11510
<
<        // reset the game data //
<        ResetGameData()
11809,11810c11513,11514
<     // reset the game data //
<     func ResetGameData()
---
>     // exit the game analysis mode //
>     func ExitAnalysisMode()
11825,11827d11528
<        // set that the game explorer window is currently closed //
<        GameExplorerWindowOpen = false
<
11861,11862c11562,11563
<        // save the game subset //
<        SaveGameSubset()
---
>        // update the board and captured pieces //
>        UpdateBoardAndCapturedPieces()
11881,11883d11581
<        // set that the game explorer window is currently closed //
<        GameExplorerWindowOpen = false
<
11908,11914d11605
<
<        // save the game subset //
<        SaveGameSubset()
<
<        // set that done entering the position editor from the game explorer //
<        EnteringPositionEditorFromGameExplorer = false
<     }
11916,11918d11606
<     // save the game subset //
<     func SaveGameSubset()
<     {
11938a11627,11629
>
>        // set that done entering the position editor from the game explorer //
>        EnteringPositionEditorFromGameExplorer = false
11952,11954d11642
<
<        // set that the game explorer window is currently closed //
<        GameExplorerWindowOpen = false
12021,12026d11708
<
<        // set the maximum value of the database game explorer slider //
<        DatabaseGameExplorerSlider.maxValue = Double(MoveList.count)
<
<        // set the current value of the database game explorer slider //
<        DatabaseGameExplorerSlider.intValue = Int32(MoveList.count)
12030,12065d11711
<
<        // clear the temporary move list //
<        TempMoveList.removeAll()
<
<        // set that initially haven't found any invalid moves //
<        FoundInvalidMoves = false
```

```
<
<       // loop over the entire move list //
<       for Index in 0 ..< MoveList.count
<       {
<          // if the move is not an invalid move //
<          if MoveList[Index] != "Invalid Move"
<          {
<             // add the move to the temporary move list //
<             TempMoveList.append(MoveList[Index])
<
<             // set that found invalid moves //
<             FoundInvalidMoves = true
<          }
<       }
<
<       // copy the temporary move list back into the move list //
<       MoveList = TempMoveList
<
<       // if found invalid moves //
<       if FoundInvalidMoves
<       {
<          // set the maximum value of the database game explorer slider //
<          DatabaseGameExplorerSlider.maxValue = Double(MoveList.count)
<
<          // set the current value of the database game explorer slider //
<          DatabaseGameExplorerSlider.intValue = Int32(MoveList.count)
<
<          // replay all the moves for this current game //
<          ReplayMoves(0, End:MoveList.count, Display:false)
<       }
12127,12128c11773,11774
<       // replay all the moves //
<       ReplayMoves(0, End:MoveList.count, Display:false)
---
>       // update the board and captured pieces //
>       UpdateBoardAndCapturedPieces()
12143,12144c11789,11790
<       // replay all the moves //
<       ReplayMoves(0, End:MoveList.count, Display:false)
---
>       // update the board and captured pieces //
>       UpdateBoardAndCapturedPieces()
12160a11807,11809
>
>       // update board //
>       UpdateBoardAndCapturedPieces()
12326,12328d11974
<
<       // replay all the moves //
<       ReplayMoves(0, End:MoveList.count, Display:false)
13641,13649d13286
<       // add a timer to run the preferences menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:
<                           #selector(GreenChess.ThePreferencesMenuItemAction), userInfo:nil,
<                           repeats:false), forMode:RunLoopMode.commonModes)
<    }
<
<    // the preferences menu item action //
<    @objc func ThePreferencesMenuItemAction()
<    {
13653,13655d13289
<       // update the move list //
<       UpdateMoveList()
<
13669,13671d13302
<       // set that currently searching the game list window //
<       SearchingGameList = true
<
13674,13676d13304
<
<       // reload the data rows //
<       ReloadDataRows()
13679a13308,13310
>
>       // set that currently searching the game list window //
>       SearchingGameList = true
14074a13706,13710
>    // the empty opponent item action //
>    @IBAction func OpponentItemAction(_:AnyObject)
>    {
>    }
>
14513,14521d14148
<       // add a timer to run the game explorer menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:
<                           #selector(GreenChess.TheGameExplorerMenuItemAction), userInfo:nil,
<                           repeats:false), forMode:RunLoopMode.commonModes)
<    }
<
<    // the game explorer menu item action //
<    @objc func TheGameExplorerMenuItemAction()
<    {
14529,14530c14156,14157
<       // set that the game explorer window is currently open //
<       GameExplorerWindowOpen = true
---
>       // open the game explorer window as a sheet on the main window //
>       MainWindow.beginSheet(GameExplorerWindow, completionHandler:nil)
14535,14536c14162,14163
<          // reset the game data //
<          ResetGameData()
---
>          // exit the game analysis mode //
>          ExitAnalysisMode()
14574,14576d14200
<
<       // open the game explorer window as a sheet on the main window //
<       MainWindow.beginSheet(GameExplorerWindow, completionHandler:nil)
14756a14381,14386
>
>       // set the maximum value of the database game explorer slider //
>       DatabaseGameExplorerSlider.maxValue = Double(MoveList.count)
>
>       // set the current value of the database game explorer slider //
>       DatabaseGameExplorerSlider.intValue = Int32(MoveList.count)
15494c15124,15125
<       if GameExplorerWindowOpen || InDatabaseExplorer || AnnotatingGame || ReviewingGame
---
>       if IsAttached(GameExplorerWindow) || IsAttached(DatabaseExplorerWindow) || ReviewingGame ||
>          IsAttached(GameAnnotatorWindow)
15505,15507d15135
<
<       // initialize stockfish analysis to display to false //
<       StockfishAnalysisToDisplay = false
15511,15513d15138
<
<       // clear the game info list //
<       GameInfoList.removeAll()
15527a15153,15155
>       // clear the game info list //
>       GameInfoList.removeAll()
>
```

```
15578a15207,15209
>             // initialize game info list //
>             GameInfoList.append("Move ")
>
15580c15211
<          if GameExplorerWindowOpen || InDatabaseExplorer || AnnotatingGame || ReviewingGame
---
>          if IsAttached(GameExplorerWindow) || InDatabaseExplorer || AnnotatingGame || ReviewingGame
15583,15584c15214,15217
<             GameInfoList.append(String(format:"Move %d", (PlyCount + MoveOffset) / 2 +
<                               Int(Attributes.FENStartMoveNumber)!))
---
>             GameInfoList[GameInfoList.count - 1] += NSNumber(value:Int32((PlyCount + MoveOffset) /
>                                                          2 + Int(Attributes
>                                                          .FENStartMoveNumber)!) as Int32)
>                                                          .stringValue
15586c15219
<          // not exploring the game, in database explorer, annotating a game, or reviewing game //
---
>          // not reviewing a game //
15590,15591c15223,15226
<             GameInfoList.append(String(format:"Move %d", (MoveList.count + MoveOffset) / 2 +
<                               Int(Attributes.FENStartMoveNumber)!))
---
>             GameInfoList[GameInfoList.count - 1] += NSNumber(value:Int32((MoveList.count +
>                                                          MoveOffset) / 2 + Int(Attributes
>                                                          .FENStartMoveNumber)!) as Int32)
>                                                          .stringValue
15598c15233
<             GameInfoList[0] += " and White to move"
---
>             GameInfoList[GameInfoList.count - 1] += " and White to move"
15604c15239
<             GameInfoList[0] += " and Black to move"
---
>             GameInfoList[GameInfoList.count - 1] += " and Black to move"
15611c15246
<             GameInfoList[0] += " out of check"
---
>             GameInfoList[GameInfoList.count - 1] += " out of check"
15672a15308,15310
>       // clear the pgn move //
>       PGNMove = ""
>
15707,15710d15344
<
<             // set the pgn move number //
<             PGNMove = String(format:"%d.", (MoveIndex + MoveOffset) / 2 + Int(Attributes
<                            .FENStartMoveNumber)!)
15712,15713c15346,15347
<             // if black is moving //
<             if (MoveIndex + MoveOffset) % 2 == 1
---
>             // if have a case where black has moved first from fen notation //
>             if MoveIndex == 0 && MoveOffset == 1
15715,15716c15349,15350
<                // add .. to pgn move for black //
<                PGNMove += ".."
---
>                // assign the pgn move string //
>                PGNMove = "\(Attributes.FENStartMoveNumber)..."
15719,15721c15353,15373
<             // add the pgn label to the pgn move //
<             PGNMove += " \(PGNLabel)"
<
---
>             // if is the beginning of a new white/black move sequence //
>             if (MoveIndex + MoveOffset) % 2 == 0
>             {
>                // set the pgn move string //
>                PGNMove = String(format:"%d. \(PGNLabel) ", (MoveIndex + MoveOffset) / 2 +
>                               Int(Attributes.FENStartMoveNumber)!)
>             }
>             // if not at the beginning of a new white/black move sequence //
>             else if MoveIndex != ListEnd
>             {
>                // set the pgn move string //
>                // set the pgn move string //
>                PGNMove += " \(PGNLabel) "
>             }
>             // if in the first move of the stockfish analysis move list //
>             else if MoveIndex == ListEnd
>             {
>                // set the pgn move string //
>                PGNMove = String(format:"%d... \(PGNLabel) ", (MoveIndex + MoveOffset) / 2 +
>                                Int(Attributes.FENStartMoveNumber)!)
>             }
>
15732c15384
<                PGNMove += " " + AnnotationString
---
>                PGNMove += AnnotationString
15734,15735c15386,15391
<                // add the pgn move string to the pgn move list array //
<                PGNMovesList.append(PGNMove)
---
>                // if not at the beginning of a new white/black move sequence or at end //
>                if (MoveIndex + MoveOffset) % 2 == 1 || MoveIndex == ListEnd - 1
>                {
>                   // add the pgn move string to the pgn move list array //
>                   PGNMovesList.append(PGNMove)
>                }
15738c15394,15395
<             else
---
>             else if (MoveIndex + MoveOffset) % 2 == 1 || MoveIndex == ListEnd + StockfishMoveList
>                   .count - 1
15741c15398
<                StockfishAnalysisString += "(" + PGNMove + ") "
---
>                StockfishAnalysisString += PGNMove + " "
15744c15401
<          // the move does not exist //
---
>          // the move did not exist //
15753,15758c15410,15412
<
<             // set the pgn move string to invalid move //
<             PGNMove = "Invalid Move"
<
<             // add the pgn move string to the pgn move list array //
<             PGNMovesList.append(PGNMove)
---
>
>             // set that pgn move list contains invalid data //
>             PGNMovesList.append("PGN Move Was Formatted Incorrectly")
15766,15768d15419
<          // set stockfish analysis to display to true //
<          StockfishAnalysisToDisplay = true
<
15770c15421
<          PGNMovesList.append(StockfishAnalysisString)
---
>          PGNMovesList.append("Stockfish Analysis:  " + StockfishAnalysisString)
15784c15435
<
---
>
15798,15807d15448
<
<       // select the previously selected row //
<       GameListTableView.selectRowIndexes(IndexSet(integer:PreviouslySelectedRow),
<                                          byExtendingSelection:false)
<
<       // scroll to the previously selected row //
<       GameListTableView.scrollRowToVisible(PreviouslySelectedRow)
<
<       // store the selected row to the game data //
<       GameListData.StoreSelectedRow(self.GameListTableView.selectedRow)
15836,15867c15477
<
<       // create a sized board composite //
<       CreateSizedBoardComposite(IllustrationSize)
<
<       // update the move list frames //
<       UpdateMoveListFrames()
<
<       // if the game explorer, database explorer, game annotator, or prefs window is attached //
<       if GameExplorerWindowOpen || InDatabaseExplorer || AnnotatingGame || EnteringPreferences
<       {
<          // set the max pgn moves data rows to be 8 //
<          PGNMovesData.MaxRows = 8
<
<          // hide the vertical scroller in the game info table view //
<          GameInfoTableView.enclosingScrollView?.hasVerticalScroller = false
<
<          // hide the vertical scroller in the pgn moves table view //
<          PGNMovesTableView.enclosingScrollView?.hasVerticalScroller = false
<       }
<       // no attached game explorer, database explorer, game annotator, or preferences window //
<       else
<       {
<          // set the max pgn moves data rows to be 10000 //
<          PGNMovesData.MaxRows = 10000
<
<          // unhide the vertical scroller in the game info table view //
<          GameInfoTableView.enclosingScrollView?.hasVerticalScroller = true
<
<          // unhide the vertical scroller in the pgn moves table view //
<          PGNMovesTableView.enclosingScrollView?.hasVerticalScroller = true
<       }
<
---
>
15875,15876c15485
<          PGNMovesTableView.scrollRowToVisible(min(PGNMovesList.count * 2, PGNMovesData.MaxRows) -
<                                               1)
---
>          PGNMovesTableView.scrollRowToVisible(PGNMovesList.count - 1)
16375c15984
<
---
>
16388c15997
<
---
>
```

```
17214,17222d16822
<       // add a timer to run the undo menu item action //
<       RunLoop.current.add(Timer(timeInterval:0.01, target:self, selector:#selector(GreenChess
<                          .TheUndoMoveMenuItemAction), userInfo:nil, repeats:false), forMode:
<                          RunLoopMode.commonModes)
<    }
<
<    // the undo move menu item action //
<    @objc func TheUndoMoveMenuItemAction()
<    {
17271,17327d16870
<
<    // update the move list frames //
<    func UpdateMoveListFrames()
<    {
<       // if the game explorer window is attached //
<       if GameExplorerWindowOpen
<       {
<          // set the maximum frame to be the current game explorer slider value //
<          MaxIllustrationFrame = Int(GameExplorerSlider.intValue)
<       }
<       // if the database explorer window is attached //
<       else if IsAttached(DatabaseExplorerWindow)
<       {
<          // set the maximum frame to be the current database game explorer slider value //
<          MaxIllustrationFrame = Int(DatabaseGameExplorerSlider.intValue)
<       }
<       // if the game annotator window is attached //
<       else if IsAttached(GameAnnotatorWindow)
<       {
<          // set the maximum frame to be the current game annotator slider value //
<          MaxIllustrationFrame = Int(GameAnnotatorSlider.intValue)
<       }
<       // no database explorer, annotator, or game explorer sheet attached //
<       else
<       {
<          // set the max frame to the move list count //
<          MaxIllustrationFrame = MoveList.count
<       }
<
<       // remove all the frames //
<       Illustrations.removeAll()
<
<       // loop over frames up to the max illustration frame //
<       for Index in 0 ..< MaxIllustrationFrame
<       {
<          // append a new illustration //
<          Illustrations.append(NSImage(size:NSMakeSize(IllustrationSize, IllustrationSize)))
<
<          // unlock the illustration focus //
<          Illustrations[Index].lockFocus()
<
<          // draw the sized board composite into the illustration //
<          SizedBoardComposite.draw(in:CGRect(x:0, y:0, width:IllustrationSize, height:
<                                  IllustrationSize))
<
<          // if index is less than frames count //
<          if Index < Frames.count
<          {
<             // draw the current pieces layer into the illustration //
<             Frames[Index].draw(in:CGRect(x:0, y:0, width:IllustrationSize, height:
<                               IllustrationSize))
<          }
<
<          // unlock the illustration focus //
<          Illustrations[Index].unlockFocus()
<       }
<    }
17357c16900
<          if MoveExists
---
>          if self.MoveExists
17399,17403c16942,16946
<       // save the frames to the gallery grid view //
<       GalleryGridView.Frames = Frames
<
<       // create the sized board composite for the width of a frame //
<       CreateSizedBoardComposite(CGFloat(GalleryGridView.FrameWidth))
---
>       // run on the main thread //
>       DispatchQueue.main.async
>       {
>          // save the frames to the gallery grid view //
>          self.GalleryGridView.Frames = self.Frames
17405,17406c16948,16949
<       // assign the sized board composite to the gallery grid view board composite //
<       GalleryGridView.BoardComposite = SizedBoardComposite
---
>          // create the gallery grid board composite //
>          self.CreateGalleryGridBoardComposite(self.GalleryGridView.FrameWidth)
17408,17409c16951,16952
<       // create the game gallery or grid image //
<       GalleryGridView.CreateGalleryGridImage(InGameGrid)
---
>          // create the game gallery or grid image //
>          self.GalleryGridView.CreateGalleryGridImage(self.InGameGrid)
17411,17412c16954,16955
<       // update the game gallery or grid view //
<       GalleryGridView.UpdateGalleryGridView(GalleryGridSlider.doubleValue)
---
>          // update the game gallery or grid view //
>          self.GalleryGridView.UpdateGalleryGridView(self.GalleryGridSlider.doubleValue)
17414,17416d16956
<       // run on the main thread //
<       DispatchQueue.main.async
<       {
17422d16961
<       }
17424,17425c16963,16965
<       // set that done creating the game gallery //
<       CreatingGalleryGrid = false
---
>          // set that done creating the game gallery //
>          self.CreatingGalleryGrid = false
>       }
17431,17437d16970
<       // if in the position editor //
<       if InPositionEditor
<       {
<          // return since don't want to replay the moves from within the position editor //
<          return
<       }
<
17446,17448d16978
<
<          // remove all the frames //
<          Frames.removeAll()
17506,17520d17035
<
<                // update the pieces layer for the illustrations //
<                UpdatePiecesLayer(IllustrationSize)
<
<                // append the pieces player to the frames //
<                Frames.append(PiecesLayer)
<             }
<             // the move doesn't exist //
<             else
<             {
<                // append an empty image for the illustrations //
<                Frames.append(NSImage(size:NSMakeSize(IllustrationSize, IllustrationSize)))
<
<                // set the move to be invalid move since it doesn't exist //
<                MoveList[Index] = "Invalid Move"
```