```
2c2
< // Swift source code of main.swift Version 9.1.9 at greenchess.com //
---
> // Swift source code of main.swift Version 9.1.8 at greenchess.com //
15c15,16
<                   NSApplicationDelegate, NSMenuItemValidation
---
>                   NSApplicationDelegate, NSMenuItemValidation,
>                   NSToolbarItemValidation
17,19d17
<     // the vertical y offset of the board view //
<     var YOffset:Double = 105
<
44c42,45
<
---
>
>     // the castle selection //
>     var CastSel:String = ""
>
69a71,73
>     // the position view layers //
>     var PositionLayers:PositionViewLayers = PositionViewLayers()
>
108a113,118
>     // whether to hide the game clock in the panel or not //
>     var HideClock:CGFloat = 0
>
>     // whether in full screen mode or not //
>     var InFullScreen:CGFloat = 0
>
113c123
<     var PosMovingPiece:Int = 0
---
>     var PositionMovingPiece:Int = 0
170c180
<     var PosImages:[Image] = []
---
>     var PositionImages:[CGImage] = []
220a231,239
>     // the position editor hotspot point //
>     var EditorHotspotPoint:CGPoint = CGPoint()
>
>     // the rank of the enpassant square //
>     var EnpassantSquare:APoint = APoint()
>
>     // the eraser icon for the position editor //
>     var Eraser:NSImage = NSImage(named:"eraser_icon")!
>
222,225c241
<     var MessageIcon:NSImage = CoreGraphics()
<                               .CGImageToNSImage(ImageRenderer(content:
<                               IconImage("message.fill", .blue)).cgImage!, Size:
<                               500)
---
>     var MessageIcon:NSImage = NSImage(named:"message_icon")!
228,231c244
<     var ValidatedIcon:NSImage = CoreGraphics()
<                                 .CGImageToNSImage(ImageRenderer(content:
<                                 IconImage("checkmark.message.fill", .green))
<                                 .cgImage!, Size:500)
---
>     var ValidatedIcon:NSImage = NSImage(named:"validated_icon")!
234,238c247,257
<     var WarningIcon:NSImage = CoreGraphics()
<                               .CGImageToNSImage(ImageRenderer(content:
<                               IconImage("exclamationmark.message.fill", .red))
<                               .cgImage!, Size:500)
<
---
>     var WarningIcon:NSImage = NSImage(named:"warning_icon")!
>
>     // the human opponent icon //
>     var HumanOpponentIcon:NSImage = NSImage(named:"human_opponent")!
>
>     // the greenchess opponent icon //
>     var GreenChessOpponentIcon = NSImage(named:"greenchess_opponent")!
>
>     // the uci engine opponent icon //
>     var UCIEngineOpponentIcon = NSImage(named:"uciengine_opponent")!
>
276a296,301
>
>     // the main toolbar //
>     @IBOutlet var MainToolbar:NSToolbar!
>
>     // the opponent toolbar item //
>     @IBOutlet var OpponentToolbarItem:NSToolbarItem!
278,279c303,316
<     // the fen string //
<     var FENString:String = ""
---
>     // position editor query button of whether white is to move next //
>     @IBOutlet var WhiteMovesToolbarButton:NSToolbarItem!
>
>     // position editor button of whether white can still kingside castle //
>     @IBOutlet var WhiteKingsideCastle:NSToolbarItem!
>
>     // position editor button of whether black can still kingside castle //
>     @IBOutlet var BlackKingsideCastle:NSToolbarItem!
>
>     // position editor button of whether white can still queenside castle //
>     @IBOutlet var WhiteQueensideCastle:NSToolbarItem!
>
>     // position editor button of whether black can still queenside castle //
>     @IBOutlet var BlackQueensideCastle:NSToolbarItem!
280a318,323
>     // the position editor enpassant square button //
>     @IBOutlet var PositionEditorEP:NSToolbarItem!
>
>     // the position editor second menu button //
>     @IBOutlet var PositionEditorSecondMenu:NSToolbarItem!
>
282c325
<     var LoadPositionWindow:AWindow = AWindow()
---
>     @IBOutlet var LoadPositionWindow:AWindow!
288c331
<     var FENWindow:AWindow = AWindow()
---
>     @IBOutlet var FENWindow:AWindow!
294c337
<     var TimedGameWindow:AWindow = AWindow()
```

```
---
>    @IBOutlet var TimedGameWindow:AWindow!
346a390,404
>    // the load position view //
>    @IBOutlet var LoadPositionView:NSView!
>
>    // the enter fen to convert text field //
>    @IBOutlet var EnterFENTextField:NSTextField!
>
>    // the timed game opponent popup button //
>    @IBOutlet var TimedGameOpponentButton:NSPopUpButton!
>
>    // the timed game color popup button //
>    @IBOutlet var TimedGameColorButton:NSPopUpButton!
>
>    // the timed game time control popup button //
>    @IBOutlet var TimedGameTimeButton:NSPopUpButton!
>
380,381c438,441
<    var PosPieces:[Int:Int] = [2:0, 3:1, 4:2, 5:3, 6:4, 7:5, 8:6, 9:-1, 10:-2,
<                              11:-3, 12:-4, 13:-5, 14:-6]
---
>    var PosPieces:[String:Int] = ["204":1, "205":2, "206":3, "207":4, "208":5,
>                                 "209":6, "210":-1, "211":-2, "212":-3, "213":-4,
>                                 "214":-5, "215":-6, "217":6, "218":5, "219":-6,
>                                 "220":-5]
383,394d442
<    // the enpassant squares label dictionary //
<    var EnpassantSquares:[Int:String] = [1:"a3", 2:"b3", 3:"c3", 4:"d3", 5:"e3",
<                                        6:"f3", 7:"g3", 8:"h3", 9:"a6", 10:"b6",
<                                        11:"c6", 12:"d6", 13:"e6", 14:"f6", 15:
<                                        "g6", 16:"h6"]
<
<    // the enpassant squares selection dictionary //
<    var EnpassantSelection:[String:Int] = ["-":0, "a3":1, "b3":2, "c3":3, "d3":4,
<                                          "e3":5, "f3":6, "g3":7, "h3":8, "a6":9,
<                                          "b6":10, "c6":11, "d6":12, "e6":13,
<                                          "f6":14, "g6":15, "h6":16]
<
405,407d452
<       // make the main window transparent to hide it //
<       MainWindow.alphaValue = 0
<
446c491,504
<
---
>
>    // undo the diagram path //
>    @MainActor @IBAction func UndoPath(_:AnyObject) {Sketch.UndoPath()}
>
>    // update the diagram color //
>    @MainActor @IBAction func UpdateDiagramColor(_:AnyObject)
>    {
>       // update the sketch diagram color //
>       Sketch.UpdateColor(MainToolbar!.selectedItemIdentifier!.rawValue)
>    }
>
>    // clear the diagram the user drew on //
>    @MainActor @IBAction func ClearDiagram(_:AnyObject) {Sketch.Refresh(true)}
>
493c551
<       return CGRect(x:0, y:740, width:Window.frame.size.width, height:0)
---
>       return CGRect(x:0, y:700, width:Window.frame.size.width, height:0)
503a562,564
>
>       // center the main window //
>       MainWindow.center()
529c590,600
<
---
>
>    // the window will begin sheet delegate //
>    @MainActor func windowWillBeginSheet(_ Notif:Notification)
>    {
>       // update the window title //
>       UpdateWindowTitle()
>    }
>
>    // the window did end sheet delegate //
>    @MainActor func windowDidEndSheet(_ Notif:Notification) {UpdateWindowTitle()}
>
536c607
<                  .Vals("GameExplorerWindowOpen").VB ? Int(round(GCUI
---
>                  .Vals("GameExplorerWindowOpen").VB ? Int(GCUI
538,541c609,611
<                  .GetCount()))) : (Inf.Vals("InDatabaseExplorer").VB ?
<                  Int(round(GCUI.Objs["DataExpMove"]!.Val * Double(GCUI
<                  .Attributes.Inf.GetCount()))) : -1), Size:1320, Update:
<                  false))
---
>                  .GetCount())) : (Inf.Vals("InDatabaseExplorer").VB ?
>                  Int(GCUI.Objs["DataExpMove"]!.Val * Double(GCUI.Attributes
>                  .Inf.GetCount())) : -1), Size:1320, Update:false))
716a787,793
>    }
>
>    // validate a toolbar item //
>    @MainActor func validateToolbarItem(_ Item:NSToolbarItem) -> Bool
>    {
>       // return if the toolbar item is enabled //
>       return Item.isEnabled
914,932c991,997
<       // run on the main thread //
<       Task {await MainActor.run
<       {
<          // set the window content size //
<          MainWindow.setContentSize(CGSize(width:660 + BoolTo().CG(Inf
<                     .Vals("SmallInfoPanel").VB) * 320 + BoolTo().CG(Inf
<                     .Vals("LargeInfoPanel").VB) * 630, height:800))
<
<          // center the main window //
<          MainWindow.center()
<
<          // update the window size menu item states and clock origins //
<          (frame, HideInfoMenuItem.state, LargeInfoMenuItem.state,
<          SmallInfoMenuItem.state, GCUI.GetClockView("WhiteClockView").frame
<          .origin, GCUI.GetClockView("BlackClockView").frame.origin) = (CGRect(x:
<          0, y:YOffset, width:660, height:660), UpdatedState(Inf
<          .Vals("HideInfoPanel").VB), UpdatedState(Inf.Vals("LargeInfoPanel")
<          .VB), UpdatedState(Inf.Vals("SmallInfoPanel").VB), CGPoint(x:205, y:2),
<          CGPoint(x:350, y:2))
---
```

```
>       // update the window size menu item states and other variables //
>       (HideInfoMenuItem.state, LargeInfoMenuItem.state, SmallInfoMenuItem.state,
>       HideClock, InFullScreen, frame) = (UpdatedState(Inf.Vals("HideInfoPanel")
>       .VB), UpdatedState(Inf.Vals("LargeInfoPanel").VB), 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))
934,951c999,1002
<          // if have a small info panel //
<          if Inf.Vals("SmallInfoPanel").VB
<          {
<             // update the frames for the small info panel //
<             (GCUI.Objs["GameInfoTableView"]!.Place, GCUI
<             .Objs["MoveListTableView"]!.Place) = (Placement(XP:670, YP:392.5,
<             Width:300, Height:372.5), Placement(XP:670, YP:10, Width:300,
<             Height:372.5))
<          }
<          // if have a large info panel //
<          else if Inf.Vals("LargeInfoPanel").VB
<          {
<             // update the frames for the large info panel //
<             (GCUI.Objs["GameInfoTableView"]!.Place, GCUI
<             .Objs["MoveListTableView"]!.Place) = (Placement(XP:980, YP:10,
<             Width:300, Height:755), Placement(XP:670, YP:10, Width:300, Height:
<             755))
<          }
---
>       // set the window content size //
>       MainWindow.setContentSize(CGSize(width:660 + BoolTo().CG(Inf
>                 .Vals("SmallInfoPanel").VB) * 320 + BoolTo().CG(Inf
>                 .Vals("LargeInfoPanel").VB) * 620, height:729))
953,969c1004,1028
<          // initialize the move tracker //
<          Tracker.Initialize(self, Margin:Dims.Border.Width, Bounds:bounds,
<                         YOffset:YOffset)
<
<          // refresh the views //
<          RefreshViews(["GameInfoTableView":false, "MoveListTableView":false,
<                     "PosEditorToolbar":false, "DiagramToolbar":false,
<                     "AnalysisToolbar":false, "ExitGameToolbar":false])
<
<          // set the main window to be visible //
<          MainWindow.alphaValue = 1
<
<          // refresh the table views //
<          GCUI.RefreshViews(["MoveListTableView":false, "GameInfoTableView":
<                        false])
<       }}
<
---
>       // if have a small info panel //
>       if Inf.Vals("SmallInfoPanel").VB
>       {
>          // update the frames for the small info panel //
>          (GCUI.GetClockView("WhiteClockView").frame.origin, GCUI
>          .GetClockView("BlackClockView").frame.origin, GCUI
>          .Objs["GameInfoTableView"]!.Place, GCUI.Objs["MoveListTableView"]!
>          .Place) = (CGPoint(x:695, y:280), CGPoint(x:845, y:280), Placement(XP:
>          670, YP:390 - HideClock * 55, Width:300, Height:260 + HideClock * 55),
>          Placement(XP:670, YP:10, Width:300, Height:260 + HideClock * 55))
>       }
>       // if have a large info panel //
>       else if Inf.Vals("LargeInfoPanel").VB
>       {
>          // update the frames for the large info panel //
>          (GCUI.GetClockView("WhiteClockView").frame.origin, GCUI
>          .GetClockView("BlackClockView").frame.origin, GCUI
>          .Objs["GameInfoTableView"]!.Place, GCUI.Objs["MoveListTableView"]!
>          .Place) = (CGPoint(x:1000, y:550 + InFullScreen * 10), CGPoint(x:1145,
>          y:550 + InFullScreen * 10), Placement(XP:975, YP:10 + InFullScreen *
>          25 * 0 - HideClock * 0, Width:295, Height:530 + HideClock * 110 +
>          InFullScreen * 10), Placement(XP:670, YP:10 + InFullScreen * 25 * 0,
>          Width:295, Height:640 + InFullScreen * 10))
>       }
>
971c1030
<       GCUI.RefreshViews(["MoveListTableView":false, "GameInfoTableView":false])
---
>       RefreshViews(["GameInfoTableView":false, "MoveListTableView":false])
1002,1006c1061,1062
<               // go right in the position images index //
<               LoadPosIndex = min(PosImages.count - 1, LoadPosIndex + 1)
<
<               // update the load position image //
<               GCUI.Objs["LoadPosImage"]!.TheImage = PosImages[LoadPosIndex]
---
>               // go to the right in the load position window view //
>               PositionButtonRightAction(self)
1038,1042c1094,1095
<               // go left in the position images index //
<               LoadPosIndex = max(LoadPosIndex - 1, 0)
<
<               // update the load position image //
<               GCUI.Objs["LoadPosImage"]!.TheImage = PosImages[LoadPosIndex]
---
>               // go to the left in the load position window view //
>               PositionButtonLeftAction(self)
1353a1407
>
1355c1409
<       else if MovingPiece.Castl("e1c1", Piece:6)
---
>       if MovingPiece.Castl("e1c1", Piece:6)
1362a1417
>
1364c1419
<       else if MovingPiece.Castl("e8g8", Piece:-6)
---
>       if MovingPiece.Castl("e8g8", Piece:-6)
1371a1427
>
1373c1429
<       else if MovingPiece.Castl("e8c8", Piece:-6)
---
>       if MovingPiece.Castl("e8c8", Piece:-6)
1458,1459c1514,1515
<             // hide the clock //
<             (TheClock.isHidden, TheClock.alphaValue) = (true, 0)
---
>             // decrement the clock alpha value //
>             TheClock.alphaValue -= (TheClock.alphaValue > 0.2) ? 0.02 : 0
1470,1473c1526,1527
<          // update the clock hidden status //
<          (TheClock.isHidden, TheClock.alphaValue) = (Inf
<          .Vals("HideGameClockInPanel").VB, Inf.Vals("HideGameClockInPanel").VB
<          ? 0 : 1)
---
>          // increment the clock alpha value //
>          TheClock.alphaValue += (TheClock.alphaValue < 1) ? 0.02 : 0
1557c1611,1612
<       TotalGameTime) = (0, 0, 0, 0, 0)
---
```

```
>       TotalGameTime, TimedGameColor, TimedGameTime,TimedGameOpponent) = (0, 0,
>       0, 0, 0, 0, 0, 0)
1624a1680,1683
>
>                // save the enpassant square //
>                EnpassantSquare.Set(File, Y:Rank, Dims:Dims, Rotated:GCUI.BPComp
>                                   .Inf.Vals("RotateBoard").VB)
1740c1799
<    @MainActor func PositionEditorSave()
---
>    @MainActor @IBAction func PositionEditorSaveAction(_:AnyObject)
2062,2070d2120
<
<       // initialize the load position window //
<       LoadPositionWindow.Update(655, Height:685)
<
<       // initialize the timed game window //
<       TimedGameWindow.Update(550, Height:100)
<
<       // initialize the enter fen window //
<       FENWindow.Update(660, Height:100)
2085c2135
<       AboutWindow.Update(660, Height:525)
---
>       AboutWindow.Update(660, Height:530)
2097,2111c2147,2162
<           "ValidFENString":true, "PlayingEngineMatch":false,
<           "ValidFENEnpassantSquare":false, "LegalFENLayout":false, "MoveExists":
<           false, "EditingNonemptyPosition":false, "InPositionEditor":false,
<           "InDatabaseExplorer":false, "QuerySavePosition":false,
<           "QueryLoadPosition":false, "EnteringPreferences":false,
<           "InGameGallery":false, "QueryPawnPromotion":false,
<           "QueryStartNewGame":false, "QueryTimedGameOver":false,
<           "QuerySaveEditedPosition":false, "QueryQuitApplication":false,
<           "EngineThinking":false, "HumanOpponent":false, "EngineOpponent":false,
<           "UCIEngineOpponent":false, "Diagramming":false, "ReviewingGame":false,
<           "ExploringGame":false, "AnnotatingGame":false, "PlayingTimedGame":
<           false, "EnteringFEN":false, "AnalyzingGame":false,
<           "WillSaveEditedPosition":false, "WhiteKingsideCastle":false,
<           "WhiteQueensideCastle":false, "BlackKingsideCastle":false,
<           "BlackQueensideCastle":false, "QueryInvalidPGNFile":false])
---
>           "SecondPositionMenu":false, "ValidFENString":true,
>           "PlayingEngineMatch":false, "ValidFENEnpassantSquare":false,
>           "LegalFENLayout":false, "MoveExists":false, "EditingNonemptyPosition":
>           false, "InPositionEditor":false, "InDatabaseExplorer":false,
>           "QuerySavePosition":false, "QueryLoadPosition":false,
>           "EnteringPreferences":false, "InGameGallery":false,
>           "QueryPawnPromotion":false, "QueryStartNewGame":false,
>           "QueryTimedGameOver":false, "QuerySaveEditedPosition":false,
>           "QueryQuitApplication":false, "EngineThinking":false, "HumanOpponent":
>           false, "EngineOpponent":false, "UCIEngineOpponent":false,
>           "Diagramming":false, "ReviewingGame":false, "ExploringGame":false,
>           "AnnotatingGame":false, "PlayingTimedGame":false, "EnteringFEN":false,
>           "AnalyzingGame":false, "WillSaveEditedPosition":false,
>           "WhiteKingsideCastle":false, "WhiteQueensideCastle":false,
>           "BlackKingsideCastle":false, "BlackQueensideCastle":false,
>           "QueryInvalidPGNFile":false])
2127c2178
<           265], "GameClockHue":[27, 0, GCUI.GetColorView("ClockColorView")
---
>           250], "GameClockHue":[27, 0, GCUI.GetColorView("ClockColorView")
2145,2146c2196,2200
<       canDrawConcurrently, GoStopMenuItem.title, HideMenuItem.title) = (true,
<       true, true, self, true, "Engine", "Hide")
---
>       canDrawConcurrently, GoStopMenuItem.title, HideMenuItem.title,
>       TimedGameMenuItem.title, PositionEditorEP.image, EditorHotspotPoint) =
>       (true, true, true, self, true, "Engine", "Hide", "Timed Game", GCUI.BPComp
>       .Highlights.EnpassantSquareNSImage, CGPoint(x:Dims.Square.Width * 0.5, y:
>       Dims.Square.Width * 0.5))
2147a2202,2217
>       // add the default toolbar items //
>       AddToolbarItems([100])
>
>       // remove all items in the timed game opponent popup button //
>       TimedGameOpponentButton.removeAllItems()
>
>       // 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 the position layers to the position view //
>       PositionLayers.AddLayers(LoadPositionView)
>
2168c2238
<
---
>
2171c2241,2244
<
---
>
>       // center the main window //
>       MainWindow.center()
>
2174,2176c2247,2253
<                        .Board.WidthRet, Length:Dims.Board.WidthRet, YOffset:
<                        YOffset, Margin:Dims.Border.BoundsRet.midX)
<
---
>             .Board.WidthRet, Length:Dims.Board.WidthRet, YOffset:0, Margin:
>             Dims.Border.BoundsRet.midX)
>
>       // initialize the mouse tracker //
>       Tracker.Initialize(self, Margin:Dims.Border.Width, Bounds:bounds,
>                         YOffset:0)
>
2179,2181d2255
<
<       // refresh the position editor buttons //
<       RefreshPosEditorButtons(false)
2183,2195d2256
<
<    // refresh the position editor buttons //
<    @MainActor func RefreshPosEditorButtons(_ Visible:Bool)
<    {
<       // run on the main thread //
<       Task {await MainActor.run
<       {
<          // update the visibility of the position editor buttons //
<          GCUI.UpdateVisibility(["PosEditorToolbar", "WhiteCastling",
<                              "BlackCastling", "EPSquare", "ColorToMove"],
<                              Visible:Visible)
<       }}
<    }
2235c2296,2303
<       MovingPiece.Set(PosMovingPiece)
---
```

```
>          MovingPiece.Set(PositionMovingPiece)
>
>          // if placing an enpassant square //
>          if MovingPiece.PlacingEnpassantSquare()
>          {
>             // set the chosen en passant square //
>             GCUI.Attributes.Inf.Update("FENPref", VSItems:["FENEnpassantSquare":
>                                            MovingPiece.GetSquareLabel()])
2237,2238c2305,2326
<          // add the piece to the chosen square on the board //
<          BPData.RFU(MovingPiece.Location, MovingPiece.MovingPiece)
---
>             // set the enpassant square file //
>             EnpassantSquare.Set(MovingPiece.Location)
>          }
>          // not placing an en passant square //
>          else
>          {
>             // if moving piece is the eraser //
>             if MovingPiece.NoMovingPiece()
>             {
>                // if the eraser lands on the enpassant square //
>                if GCUI.Attributes.Inf.VSIsEqual("FENEnpassantSquare", VS:
>                                                          MovingPiece.GetSquareLabel())
>                {
>                   // set the fen enpassant square to - //
>                   GCUI.Attributes.Inf.Update("FENPref", VSItems:
>                                                   ["FENEnpassantSquare":"-"])
>                }
>             }
>
>             // add the piece to the chosen square on the board //
>             BPData.RFU(MovingPiece.Location, MovingPiece.MovingPiece)
>          }
2579,2580c2667,2670
<                  .Vals("InPositionEditor").VB, RankAndFile:MovingPiece.Location,
<                  UpdateBoard:Update, Info:false)
---
>                  .Vals("InPositionEditor").VB, FENEnpassantSquare:GCUI.Attributes
>                  .Inf.Vals("FENEnpassantSquare").VS, EnpassantSquare:
>                  EnpassantSquare, RankAndFile:MovingPiece.Location, UpdateBoard:
>                  Update, Info:false)
2745c2835
<                for MoveListCountIndex in 0..<min(8, OpeningBookMoveList.count)
---
>                for MoveListCountIndex in 0..<min(6, OpeningBookMoveList.count)
2821,2828c2911,2915
<          // run on the main thread //
<          Task {await MainActor.run
<          {
<             // condense the search list to values matching the search string //
<             GCUI.Objs[Key]!.GameList = GetString().FromTextField(Entry).isEmpty ?
<                                             GameListData : GameListData.filter
<                                             {$0.lowercased().contains(GetString()
<                                             .FromTextField(Entry).lowercased())}
---
>          // condense the search list to values matching the search string //
>          GCUI.Objs[Key]!.GameList = GetString().FromTextField(Entry).isEmpty ?
>                                          GameListData : GameListData.filter
>                                          {$0.lowercased().contains(GetString()
>                                          .FromTextField(Entry).lowercased())}
2830,2836c2917,2922
<             // reset the selected row to be zero //
<             GCUI.Objs[Key]!.Selection = 0
<
<             // refresh the views //
<             RefreshViews(["GameListTableView":false, "ListToolbar":GCUI.Objs[Key]!
<                         .GameList.isEmpty])
<          }}
---
>          // reset the selected row to be zero //
>          GCUI.Objs[Key]!.Selection = 0
>
>          // refresh the views //
>          RefreshViews(["GameListTableView":false, "ListToolbar":GCUI.Objs[Key]!
>                      .GameList.isEmpty])
2887,2943d2972
<    }
<
<    // play timed game action //
<    @MainActor func PlayTimedGameAction(_ Selection:Int)
<      {
<          // switch over the selection //
<          switch Selection
<          {
<             // exit the timed game window //
<             case 0 : CloseSheet(TimedGameWindow)
<
<             // play the timed game from the user selections //
<             case 1 :
<
<                // update the exit timed game toolbar visibility //
<                GCUI.UpdateVisibility(["ExitGameToolbar"], Visible:true)
<
<                // start a new game //
<                StartNewGame("All")
<
<                // set that currently playing a timed game and to start the clock //
<                Inf.Update("VB", VBItems:["PlayingTimedGame":true, "StartClock":
<                            true])
<
<                // save the use eco book value //
<                Inf.SaveValue("VBPref", Key:"UseEcoBook")
<
<                // set that currently using opening book //
<                Inf.TurnOnVB("VBPref", Key:"UseEcoBook")
<
<                // update the opponent type //
<                UpdateOpponent((TimedGameOpponent == 0) ? "GreenChess" :
<                              ((TimedGameOpponent == 1) ? "Human" : "Stockfish"))
<
<                // calculate the total game time //
<                TotalGameTime = 9000 * (TimedGameTime + 1)
<
<                // calc timed game offset from 6 o'clock pos and remaining times //
<                (TimedGameOffset, RemainingWhiteTime, RemainingBlackTime) = (216000
<                - TotalGameTime, TotalGameTime, TotalGameTime)
<
<                // initialize the game clock //
<                InitializeGameClock()
<
<                // close the timed game window //
<                CloseSheet(TimedGameWindow)
<
```

```
<                 // if user plays black against engine, start engine move //
<              if (TimedGameColor == 1) && !Inf.Vals("HumanOpponent").VB
<              {
<                 // start the engine move //
<                 StartEngineMove()
<              }
<
<           // the default case //
<           default : break
<        }
2946,3066d2974
<     // the enter fen string action //
<     @MainActor func EnterFENAction(_ Selection:Int)
<     {
<        // close the fen window //
<        CloseSheet(FENWindow)
<
<        // if the fen string is not empty //
<        if !FENString.isEmpty
<        {
<           // save the current board layout //
<           SaveBoardLayout()
<
<           // extract the entered fen string //
<           GCUI.Attributes.ExtractEnteredFENString(FENString)
<
<           // set the initial layout //
<           SetInitialLayout()
<
<           // send the fen to convert to the position editor //
<           StartPosEditor()
<        }
<
<        // set entering fen to false //
<        Inf.TurnOffVB("VB", Key:"EnteringFEN")
<     }
<
<     // update the position editor //
<     @MainActor func UpdatePosEditor(_ Key:String, Selection:Int)
<     {
<        // switch over the key //
<        switch Key
<        {
<           // update the white castling selection //
<           case "WhiteCastling" : Inf.Update("VB", VBItems:["WhiteKingsideCastle":
<                                  (Selection == 1 || Selection == 3) ? true :
<                                  false, "WhiteQueensideCastle": (Selection == 2
<                                  || Selection == 3) ? true : false])
<
<           // update the black castling selection //
<           case "BlackCastling" : Inf.Update("VB", VBItems:["BlackKingsideCastle":
<                                  (Selection == 1 || Selection == 3) ? true :
<                                  false, "BlackQueensideCastle": (Selection == 2
<                                  || Selection == 3) ? true : false])
<
<           // update the enpassant square selection //
<           case "EPSquare" : GCUI.Attributes.Inf.Update("FENPref", VSItems:
<                             ["FENEnpassantSquare":EnpassantSquares[Selection] !=
<                             nil ? EnpassantSquares[Selection]! : "-"])
<
<           // update the color to move selection //
<           case "ColorToMove" : GCUI.Attributes.Inf.Update("FENPref", VSItems:
<                               ["FENColor":(Selection == 0) ? "w" : "b"])
<
<           // the default case //
<           default : break
<        }
<     }
<
<     // update the timed game info //
<     @MainActor func UpdateTimedGameInfo(_ Key:String, Selection:Int)
<     {
<        // set the timed game color //
<        TimedGameColor = GCUI.Objs["TimedGameColor"]!.Selection
<
<        // set the timed game opponent //
<        TimedGameOpponent = GCUI.Objs["TimedGameOpponent"]!.Selection
<
<        // set the timed game time //
<        TimedGameTime = GCUI.Objs["TimedGameTime"]!.Selection
<     }
<
<     // the load position action //
<     @MainActor func LoadPositionAction(_ Selection:Int)
<     {
<        // switch over the selection //
<        switch Selection
<        {
<           // exit the load position window and close the load position window //
<           case 0 : CloseSheet(LoadPositionWindow)
<
<           // load the selected position //
<           case 1 :
<
<              // start a new game //
<              StartNewGame("")
<
<              // extract and display the game information from game data list //
<              ExtractGameData(SavedPositionData[LoadPosIndex], Tasks:"All")
<
<              // Initialize the game clock //
<              InitializeGameClock()
<
<              // store the autosave data //
<              StoreAutoData("")
<
<              // close the load position window //
<              CloseSheet(LoadPositionWindow)
<
<           // go left in the load position window //
<           case 2 :
<
<              // go left in the position images index //
<              LoadPosIndex = max(LoadPosIndex - 1, 0)
<
<              // set the load position image to the selected index //
<              GCUI.Objs["LoadPosImage"]!.TheImage = PosImages[LoadPosIndex]
<
<           // go right in the load position window //
<           case 3 :
<
<              // go right in the position images index //
<              LoadPosIndex = min(PosImages.count - 1, LoadPosIndex + 1)
<
<              // set the load position image to the selected index //
<              GCUI.Objs["LoadPosImage"]!.TheImage = PosImages[LoadPosIndex]
<
<           // the default case //
<           default : break
<        }
<     }
<
```

```
3103,3189d3010
<    // the position editor action //
<    @MainActor func PositionAction(_ Selection:Int)
<    {
<       // switch over the selection //
<       switch Selection
<       {
<          // the exit button //
<          case 0 :
<
<             // refresh the game explorer toolbar //
<             RefreshViews(["GameExpToolbar":Inf.Vals("ReviewingGame").VB])
<
<             // exit the position editor //
<             ExitPositionEditor()
<
<          // save the edited position //
<          case 1 : PositionEditorSave()
<
<          // set the cursor to the piece currently being placed in the editor //
<          case 2...14 : SetCursor(PosPieces[Selection]!)
<
<          // the default case //
<          default : break
<       }
<    }
<
<    // the game analysis action //
<    @MainActor func AnalysisAction(_ Selection:Int)
<    {
<       // update the visibility of the game analysis toolbar //
<       GCUI.UpdateVisibility(["AnalysisToolbar"], Visible:false)
<
<       // exit the game anlysis or timed game mode //
<       ExitAnalysisOrTimedGame()
<    }
<
<    // exit the the timed game action //
<    @MainActor func ExitGameAction(_ Selection:Int)
<    {
<       // update the visibility of the exit timed game toolbar //
<       GCUI.UpdateVisibility(["ExitGameToolbar"], Visible:false)
<
<       // exit the game analysis or timed game mode //
<       ExitAnalysisOrTimedGame()
<    }
<
<    // the board diagram action //
<    @MainActor func DiagramAction(_ Selection:Int)
<    {
<       // switch over the selection //
<       switch Selection
<       {
<          // exit the board diagram mode //
<          case 0 :
<
<             // refresh the game explorer toolbar //
<             RefreshViews(["GameExpToolbar":Inf.Vals("ReviewingGame").VB])
<
<             // update the visibility of the diagram toolbar //
<             GCUI.UpdateVisibility(["DiagramToolbar"], Visible:false)
<
<             // the start game explorer menu item action //
<             StartGameExplorer()
<
<          // take the board snapshot action //
<          case 1 : BoardSnapshotAction()
<
<          // refresh the board sketch //
<          case 2 : Sketch.Refresh(true)
<
<          // update the sketch path //
<          case 3 : Sketch.UndoPath()
<
<          // update the marker drawing color //
<          case 4...7 :
<
<             // set that currently diagramming //
<             Inf.TurnOnVB("VB", Key:"Diagramming")
<
<             // set the current diagram cursor //
<             Sketch.UpdateColor(Selection + 1)
<
<          // the default case //
<          default : break
<       }
<    }
<
3245,3246c3066,3069
<          case 5...8 : GCUI.GetGalleryView("GalleryView")
<                            .UpdateGalleryColor(Selection)
---
>          case 5...8 :
>
>             // update the sketch diagram color //
>             GCUI.GetGalleryView("GalleryView").UpdateGalleryColor(Selection)
3269c3092
<          case 0 :
---
>          case 0 : ExitGameExplorer()
3271,3283d3093
<             // invalidate the game explorer review timer //
<             GameExplorerTimer.invalidate()
<
<             // set game explorer window open, exploring, reviewing to false //
<             Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false,
<                       "ExploringGame":false, "ReviewingGame":false])
<
<             // close the game explorer window sheet //
<             CloseSheet(GameExplorerWindow)
<
<             // refresh all the views //
<             Refresh("All")
<
3301,3315c3111,3112
<          case 3 :
<
<             // update the game analysis toolbar visibility //
<             GCUI.UpdateVisibility(["AnalysisToolbar"], Visible:true)
<
<             // set that game explorer window closed and analyzing game //
<             Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false,
<                       "AnalyzingGame":true])
<
<             // close the game explorer window sheet //
<             CloseSheet(GameExplorerWindow)
<
<             // save the game subset //
<             SaveGameSubset()
<
---
```

```
>           case 3 : EnterAnalysisMode()
>
3317,3324c3114
<           case 4 :
<
<               // set explorer window false, analyzing, editing nonempty to true //
<               Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false,
<                         "AnalyzingGame":true, "EditingNonemptyPosition":true])
<
<               // close the game explorer window sheet //
<               CloseSheet(GameExplorerWindow)
---
>           case 4 : ExplorerPositionEditor()
3326,3330c3116,3117
<               // save the game subset //
<               SaveGameSubset()
<
<               // enter the position editor //
<               StartPosEditor()
---
>           // enter the diagram mode //
>           case 5 : EnterDiagramMode()
3332,3356d3118
<           // enter the board diagram mode //
<           case 5 :
<
<               // set the current marker color selection to red //
<               GCUI.Objs["DiagramToolbar"]!.Selection = 4
<
<               // update the board diagram toolbar
<               GCUI.UpdateVisibility(["DiagramToolbar"], Visible:true)
<
<               // update explorer window to not open and that diagramming //
<               Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false,
<                         "Diagramming":true])
<
<               // set the diagram cursor to red highlighter //
<               Sketch.UpdateColor(5)
<
<               // close the game explorer window sheet //
<               CloseSheet(GameExplorerWindow)
<
<               // save the game subset //
<               SaveGameSubset()
<
<               // refresh the sketch //
<               Sketch.Refresh(true)
<
3478,3483d3239
<       // set the position editor piece selection to white pawn //
<       GCUI.Objs["PosEditorToolbar"]!.Selection = 3
<
<       // refresh the position editor buttons //
<       RefreshPosEditorButtons(true)
<
3509,3539d3264
<
<       // set the white castling picker selection //
<       GCUI.Objs["WhiteCastling"]!.Selection = (Inf.Vals("WhiteKingsideCastle")
<                                                 .VB && Inf
<                                                 .Vals("WhiteQueensideCastle").VB)
<                                             ? 3 : (Inf
<                                             .Vals("WhiteKingsideCastle").VB ?
<                                             1 : (Inf
<                                             .Vals("WhiteQueensideCastle").VB
<                                             ? 2 : 0))
<
<       // set the black castling picker selection //
<       GCUI.Objs["BlackCastling"]!.Selection = (Inf.Vals("BlackKingsideCastle")
<                                             .VB && Inf
<                                             .Vals("BlackQueensideCastle").VB)
<                                             ? 3 : (Inf
<                                             .Vals("BlackKingsideCastle").VB ?
<                                             1 : (Inf
<                                             .Vals("BlackQueensideCastle").VB
<                                             ? 2 : 0))
<
<       // set the en passant square picker selection //
<       GCUI.Objs["EPSquare"]!.Selection = (EnpassantSelection[GCUI.Attributes.Inf
<                                        .Vals("FENEnpassantSquare").VS] != nil)
<                                        ? EnpassantSelection[GCUI.Attributes
<                                        .Inf.Vals("FENEnpassantSquare").VS]! :
<                                        0
<
<       // set the color to move picker selection //
<       GCUI.Objs["ColorToMove"]!.Selection = (GCUI.Attributes.Inf
<                                            .Vals("FENColor").VS == "w") ? 0 : 1
3544,3546c3269,3272
<       // lighten the table views //
<       (GCUI.Objs["GameInfoTableView"]!.ObjView.alphaValue, GCUI
<       .Objs["MoveListTableView"]!.ObjView.alphaValue) = (0.25, 0.25)
---
>       // turn on the white pawn toolbar item and lighten the table views //
>       (MainToolbar.selectedItemIdentifier, GCUI.Objs["GameInfoTableView"]!
>       .ObjView.alphaValue, GCUI.Objs["MoveListTableView"]!.ObjView.alphaValue) =
>       (NSToolbarItem.Identifier(rawValue:"204"), 0.25, 0.25)
3548,3549c3274,3276
<       // save edited position to false //
<       Inf.Update("VB", VBItems:["WillSaveEditedPosition":false])
---
>       // save edited position to false and second position menu to false //
>       Inf.Update("VB", VBItems:["WillSaveEditedPosition":false,
>                 "SecondPositionMenu":false])
3555a3283,3285
>
>       // add the position editor piece items to the toolbar //
>       AddPositionEditorPieceItems()
3651,3660c3381,3383
<
<       // refresh the position editor buttons //
<       RefreshPosEditorButtons(false)
<
<       // if currently in game analysis mode //
<       if Inf.Vals("AnalyzingGame").VB
<       {
<          // update the visibility of the game analysis toolbar //
<          GCUI.UpdateVisibility(["AnalysisToolbar"], Visible:true)
<       }
---
>
>       // add the specified toolbar items //
>       AddToolbarItems([Inf.Vals("AnalyzingGame").VB ? 300 : 100])
3740a3464,3503
>
>    // enter the diagram mode //
>    @MainActor func EnterDiagramMode()
>    {
>       // update game explorer window not open and that currently diagramming //
>       Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false, "Diagramming":
>                 true])
>
>       // add the diagram items to the toolbar //
>       AddToolbarItems(Array(300...311))
>
>       // set the main toolbar selected item to 304 for red color //
>       MainToolbar.selectedItemIdentifier = NSToolbarItem.Identifier(rawValue:
>                                                                  "304")
>
>       // set the diagram cursor to red highlighter //
>       Sketch.UpdateColor("304")
>
>       // close the game explorer window sheet //
>       CloseSheet(GameExplorerWindow)
>
>       // save the game subset //
>       SaveGameSubset()
>
>       // refresh the sketch //
>       Sketch.Refresh(true)
>    }
>
>    // enter the position editor directly from the game explorer //
>    @MainActor func ExplorerPositionEditor()
>    {
>       // set game explorer window false, analyzing, editing nonempty pos true //
>       Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false, "AnalyzingGame":
>                 true, "EditingNonemptyPosition":true])
>
>       // close the game explorer window sheet //
>       CloseSheet(GameExplorerWindow)
>
>       // save the game subset //
>       SaveGameSubset()
3741a3505,3525
>       // enter the position editor //
>       StartPosEditor()
>    }
>
>    // enter the game analysis mode //
>    @MainActor func EnterAnalysisMode()
>    {
>       // set that game explorer window currently closed and analyzing game //
>       Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false, "AnalyzingGame":
>                 true])
>
>       // add the analysis items to the main toolbar //
>       AddToolbarItems([300])
>
>       // close the game explorer window sheet //
>       CloseSheet(GameExplorerWindow)
>
>       // save the game subset //
>       SaveGameSubset()
>    }
>
3746,3747c3530,3531
<       GCUI.Attributes.SaveSubset(Int(round(GCUI.Objs["GameExpMove"]!.Val *
<                                 Double(GCUI.Attributes.Inf.GetCount()))))
---
```

```
>        GCUI.Attributes.SaveSubset(Int(GCUI.Objs["GameExpMove"]!.Val * Double(GCUI
>                                  .Attributes.Inf.GetCount())))
3751a3536,3548
> 
>     // exit the game explorer //
>     @MainActor func ExitGameExplorer()
>     {
>        // invalidate the game explorer review timer //
>        GameExplorerTimer.invalidate()
> 
>        // set game explorer window open, exploring game, reviewing all false //
>        Inf.Update("VB", VBItems:["GameExplorerWindowOpen":false, "ExploringGame":
>                  false, "ReviewingGame":false])
> 
>        // close the game explorer window sheet //
>        CloseSheet(GameExplorerWindow)
3752a3550,3553
>        // refresh all the views //
>        Refresh("All")
>     }
> 
3860c3661,3665
< 
---
> 
>        // refresh the gallery toobar //
>        RefreshViews(["GalleryToolbar":GCUI.GetGalleryView("GalleryView").Inf
>                    .Vals("Reviewing").VB])
> 
3865,3867c3670,3672
<                          70))
< 
<        // set frame for gallery view //
---
>                          40))
> 
>        // set frame for gallery and gallery toolbar selected item identifier //
3872,3875d3676
<        // refresh the gallery toolbar //
<        RefreshViews(["GalleryToolbar":GCUI.GetGalleryView("GalleryView").Inf
<                    .Vals("Reviewing").VB])
< 
4036a3838,3924
>     }
> 
>     // add a flexible space item to the main toolbar //
>     @MainActor func AddFlexibleSpace()
>     {
>        // add a flexible space item to the main toolbar //
>        MainToolbar.insertItem(withItemIdentifier:NSToolbarItem
>                              .Identifier(rawValue:"NSToolbarFlexibleSpaceItem"),
>                              at:MainToolbar.items.count)
>     }
> 
>     // add an item in the main toolbar //
>     @MainActor func InsertItem(_ ItemTag:String)
>     {
>        // add the specified item to the main toolbar //
>        MainToolbar.insertItem(withItemIdentifier:NSToolbarItem
>                    .Identifier(rawValue:ItemTag), at:MainToolbar.items.count)
> 
>        // add a flexible space to the main toolbar //
>        AddFlexibleSpace()
>     }
> 
>     // add toolbar items to the main toolbar //
>     @MainActor func AddToolbarItems(_ Indices:[Int])
>     {
>        // remove all the toolbar items //
>        while !MainToolbar.items.isEmpty
>        {
>           // remove the toolbar item at the location //
>           MainToolbar.removeItem(at:MainToolbar.items.count - 1)
>        }
> 
>        // add a flexible space to the main toolbar //
>        AddFlexibleSpace()
> 
>        // add all the toolbar items //
>        for Index in Indices {InsertItem("\(Index)")}
> 
>        // enable all the toolbar items //
>        for TheItem in MainToolbar.items {TheItem.isEnabled = true}
> 
>        // update the window title //
>        UpdateWindowTitle()
>     }
> 
>     // add the position editor piece toolbar items //
>     @MainActor func AddPositionEditorPieceItems()
>     {
>        // add the position editor toolbar items //
>        AddToolbarItems([300] + Array(200...215))
> 
>        // set position editor second menu button image to the going right icon //
>        PositionEditorSecondMenu.image = NSImage(named:NSImage
>                                                .goRightTemplateName)
> 
>        // update the position editor toolbar buttons //
>        UpdatePositionEditorToolbarButtons()
>     }
> 
>     // go left in the list of saved positions //
>     @MainActor @IBAction func PositionButtonLeftAction(_:AnyObject)
>     {
>        // if there is only one saved position, return //
>        if SavedPositionData.count == 1 {return}
> 
>        // if at beginning of saved position list, update load position index //
>        LoadPosIndex = (LoadPosIndex == 0) ? SavedPositionData.count - 1 :
>                       LoadPosIndex - 1
> 
>        // animate the position image changing //
>        PositionLayers.Animate(LoadPositionView, Image:
>                              PositionImages[LoadPosIndex])
>     }
> 
>     // go right button in list of saved positions //
>     @MainActor @IBAction func PositionButtonRightAction(_:AnyObject)
>     {
>        // if there is only one saved position, return //
>        if SavedPositionData.count == 1 {return}
> 
>        // if the load position index is at the end of the list //
>        LoadPosIndex = (LoadPosIndex == (SavedPositionData.count - 1)) ? 0 :
>                       LoadPosIndex + 1
```

```
>
>       // animate the position image changing //
>       PositionLayers.Animate(LoadPositionView, Image:
>                              PositionImages[LoadPosIndex])
4037a3926,3938
>
>    // exit the load position window //
>    @MainActor @IBAction func ExitLoadPositionWindow(_:AnyObject) {ExitLoadPos()}
>
>    // exit load position //
>    @MainActor func ExitLoadPos()
>    {
>       // remove the position layer animations //
>       PositionLayers.RemoveAnimations()
>
>       // close the load position window sheet //
>       CloseSheet(LoadPositionWindow)
>    }
4055,4057d3955
<       // refresh the load position toolbar //
<       RefreshViews(["LoadPosToolbar":true])
<
4062c3960
<       PosImages.removeAll()
---
>       PositionImages.removeAll()
4071c3969
<          PosImages.append(BoardImage(-1, Size:1310, Update:false))
---
>          PositionImages.append(BoardImage(-1, Size:900, Update:false))
4077,4079c3975,3977
<       // update the load position image //
<       GCUI.Objs["LoadPosImage"]!.TheImage = PosImages[LoadPosIndex]
<
---
>       // update the position layer //
>       PositionLayers.UpdateLayer(PositionImages[LoadPosIndex])
>
4116a4015,4033
>    // the load position button action //
>    @MainActor @IBAction func LoadPositionButtonAction(_:AnyObject)
>    {
>       // start a new game //
>       StartNewGame("")
>
>       // extract and display the game information from the game data list //
>       ExtractGameData(SavedPositionData[LoadPosIndex], Tasks:"All")
>
>       // Initialize the game clock //
>       InitializeGameClock()
>
>       // store the autosave data //
>       StoreAutoData("")
>
>       // exit the load position window //
>       ExitLoadPos()
>    }
>
4145a4063,4113
>    // start a timed game //
>    @MainActor @IBAction func StartTimedGame(_:AnyObject)
>    {
>       // start a new game //
>       StartNewGame("All")
>
>       // add the timed game items to the main toolbar //
>       AddToolbarItems([300])
>
>       // set that currently playing a timed game and to start the clock //
>       Inf.Update("VB", VBItems:["PlayingTimedGame":true, "StartClock":true])
>
>       // update the window title //
>       UpdateWindowTitle()
>
>       // save the use eco book value //
>       Inf.SaveValue("VBPref", Key:"UseEcoBook")
>
>       // set that currently using opening book //
>       Inf.TurnOnVB("VBPref", Key:"UseEcoBook")
>
>       // 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"))
>
>       // calculate the total game time //
>       TotalGameTime = 9000 * (TimedGameTime + 1)
>
>       // calc timed game offset from 6 o'clock position and remaining times //
>       (TimedGameOffset, RemainingWhiteTime, RemainingBlackTime) = (216000 -
>       TotalGameTime, TotalGameTime, TotalGameTime)
>
>       // initialize the game clock //
>       InitializeGameClock()
>
>       // close the timed game window //
>       CloseSheet(TimedGameWindow)
>
>       // if user plays black against engine, start engine move //
>       if (TimedGameColor == 1) && !Inf.Vals("HumanOpponent").VB
>       {
>          // start the engine move //
>          StartEngineMove()
>       }
>    }
>
4158c4126,4129
<
---
```

```
>
>       // reset game menu item to play timed game //
>       TimedGameMenuItem.title = "Timed Game"
>
4165a4137,4142
>
>       // update the window title //
>       UpdateWindowTitle()
>
>       // add the toolbar items //
>       AddToolbarItems([100])
4167a4145,4161
>    // update the window title //
>    @MainActor func UpdateWindowTitle()
>    {
>       // 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)" : ""))))))))
>    }
>
4171,4173d4164
<       // refresh the timed game toolbar //
<       RefreshViews(["EnterFENStringToolbar":true])
<
4184,4186d4174
<       // refresh the timed game toolbar //
<       RefreshViews(["PlayTimedGameToolbar":true])
<
4236,4237c4224,4225
<    // update the entered fen string //
<    @MainActor func UpdateEnteredFEN(_ Key:String, Entry:String)
---
>    // load the entered fen to convert //
>    @MainActor @IBAction func LoadEnteredFEN(_:AnyObject)
4239,4240c4227,4251
<       // update the entered fen string //
<       FENString = GetString().FromTextField(Entry)
---
>       // if the entered fen string is empty //
>       if GetString().FromTextField(EnterFENTextField.stringValue).isEmpty
>       {
>          // close the enter fen window sheet //
>          CloseSheet(FENWindow)
>
>          // exit function //
>          return
>       }
>
>       // save the current board layout //
>       SaveBoardLayout()
>
>       // extract the entered fen string //
>       GCUI.Attributes.ExtractEnteredFENString(GetString()
>                       .FromTextField(EnterFENTextField.stringValue))
>
>       // close the enter fen window sheet //
>       CloseSheet(FENWindow)
>
>       // set the initial layout //
>       SetInitialLayout()
>
>       // send the fen to convert to the position editor //
>       StartPosEditor()
4243,4244c4254,4255
<    // exit the current analysis or timed game mode //
<    @MainActor func ExitAnalysisOrTimedGame()
---
>    // exit the enter fen window //
>    @MainActor @IBAction func ExitFENWindow(_:AnyObject)
4245a4257,4273
>       // set entering fen to false //
>       Inf.TurnOffVB("VB", Key:"EnteringFEN")
>
>       // close the fen window //
>       CloseSheet(FENWindow)
>    }
>
>    // exit the timed game window //
>    @MainActor @IBAction func ExitTimedGameWindow(_:AnyObject)
>    {
>       // closed the  timed game window sheet //
>       CloseSheet(TimedGameWindow)
>    }
>
>    // exit the current mode //
>    @MainActor @IBAction func ExitCurrentMode(_:AnyObject)
>    {
4249,4256c4277,4287
<       // set stop engines to true and playing engine match move to false //
<       Inf.Update("VB", VBItems:["StopEngines":true, "PlayingEngineMatch":false])
<
<       // tell the uci engine to stop //
<       Stockfish.WriteStockfishToStop()
<
<       // make the chess engine stop //
<       EngineMakeEngineStop(Engine)
---
>       // if currently diagramming //
>       if Inf.Vals("Diagramming").VB
>       {
>          // the game explorer menu item action //
>          StartGameExplorer()
>       }
>       // if currently in the position editor //
>       else if Inf.Vals("InPositionEditor").VB
>       {
>          // exit the position editor //
>          ExitPositionEditor()
4258,4259c4289,4297
<       // remove all the moving piece animations //
<       MovingPiece.removeAllAnimations()
---
>          // if analyzing the game, add the analysis toolbar items //
>          if Inf.Vals("AnalyzingGame").VB {AddToolbarItems([300])}
>       }
>       // if analyzing the game //
>       else if Inf.Vals("AnalyzingGame").VB || Inf.Vals("PlayingTimedGame").VB
>       {
>          // set stop engines to true and playing engine match move to false //
>          Inf.Update("VB", VBItems:["StopEngines":true, "PlayingEngineMatch":
>                      false])
4261,4262c4299,4306
<       // set engine thinking to false //
<       Inf.TurnOffVB("VB", Key:"EngineThinking")
---
```

```
>           // tell the uci engine to stop //
>           Stockfish.WriteStockfishToStop()
>
>           // make the chess engine stop //
>           EngineMakeEngineStop(Engine)
>
>           // remove all the moving piece animations //
>           MovingPiece.removeAllAnimations()
4264,4265c4308,4312
<        // set the go stop menu item title to engine //
<        GoStopMenuItem.title = "Engine"
---
>           // set engine thinking to false //
>           Inf.TurnOffVB("VB", Key:"EngineThinking")
>
>           // set the go stop menu item title to engine //
>           GoStopMenuItem.title = "Engine"
4267,4268c4314,4315
<        // stop the progress circle animation //
<        ProgressCircle.StopProgressAnimation()
---
>           // stop the progress circle animation //
>           ProgressCircle.StopProgressAnimation()
4270,4271c4317,4318
<        // remove the animation layer from the super layer //
<        MovingPiece.HideAnimation()
---
>           // remove the animation layer from the super layer //
>           MovingPiece.HideAnimation()
4273,4274c4320,4321
<        // reset the moving piece //
<        MovingPiece.Reset()
---
>           // reset the moving piece //
>           MovingPiece.Reset()
4276,4277c4323,4324
<        // the game explorer menu item action //
<        if Inf.Vals("AnalyzingGame").VB {StartGameExplorer()}
---
>           // the game explorer menu item action //
>           if Inf.Vals("AnalyzingGame").VB {StartGameExplorer()}
4279,4280c4326,4335
<        // if playing a timed game, remove the timed game offset //
<        if Inf.Vals("PlayingTimedGame").VB {RemoveTimedGameOffset()}
---
>           // if playing a timed game //
>           if Inf.Vals("PlayingTimedGame").VB
>           {
>              // remove the timed game offset //
>              RemoveTimedGameOffset()
>
>              // add the specified toolbar item //
>              AddToolbarItems([100])
>           }
>        }
4320a4376,4378
>
>        // add the default items to the toolbar //
>        AddToolbarItems([100])
4384c4442
<           ExtractGameData(DatabaseGetGameData(GameData, GameList, CInt(round(GCUI
---
>           ExtractGameData(DatabaseGetGameData(GameData, GameList, CInt(GCUI
4386c4444
<                          1)))), Tasks:"Replay")
---
>                          1))), Tasks:"Replay")
4443,4444c4501,4502
<        ProcessPGNMove(Int(round(GCUI.Objs[Key]!.Val * Double(GCUI.Attributes.Inf
<                      .GetCount()))) - 1)
---
>        ProcessPGNMove(Int(GCUI.Objs[Key]!.Val * Double(GCUI.Attributes.Inf
>                      .GetCount())) - 1)
4450,4453c4508,4511
<           GCUI.Objs[Key + "Image"]!.TheImage = BoardImage(Int(round(GCUI
<                                                  .Objs[Key]!.Val * Double(GCUI
<                                                  .Attributes.Inf.GetCount())) *
<                                                  ((GameListData.isEmpty && Inf
---
>           GCUI.Objs[Key + "Image"]!.TheImage = BoardImage(Int(GCUI.Objs[Key]!.Val
>                                                  * Double(GCUI.Attributes.Inf
>                                                  .GetCount()) * ((GameListData
>                                                  .isEmpty && Inf
4494,4495c4552,4553
<        // run on the main thread to set the window background color //
<        Task {await MainActor.run {SetWindowBackgroundColor(false)}}
---
>        // update the window background color //
>        SetWindowBackgroundColor(false)
4509c4567
<               GameListData[Int(round(Value * Double(GameListData.count - 1)))]
---
>               GameListData[Int(Value * Double(GameListData.count - 1))]
4533,4535c4591,4593
<           GCUI.Objs[Key + "View"]!.Selection = max(0, Int(round(Double(GCUI
<                                                    .Attributes.Inf.GetCount() - 1)
<                                                    * Value)))
---
>           GCUI.Objs[Key + "View"]!.Selection = max(0, Int(Double(GCUI
>                                                    .Attributes.Inf.GetCount() -
>                                                    1) * Value))
4579,4580c4637,4638
<        // run on the main thread //
<        Task {await MainActor.run
---
>        // if currently in the game gallery //
>        if Inf.Vals("InGameGallery").VB
4582,4590c4640,4644
<           // if currently in the game gallery //
<           if Inf.Vals("InGameGallery").VB
<           {
<              // update the game gallery view //
<              GCUI.GetGalleryView("GalleryView")
<                   .UpdateCurrentPosition(Int(round(Value * Double(GCUI
<                                          .FramesMovie.MaxPosition))))
<           }
<        }}
---
>           // update the game gallery view //
>           GCUI.GetGalleryView("GalleryView").UpdateCurrentPosition(Int(Value *
>                                               Double(GCUI.FramesMovie
>                                               .MaxPosition)))
>        }
4715a4770,4775
>        // update the opponent toolbar item image //
>        OpponentToolbarItem.image = (Opponent == "Human") ? HumanOpponentIcon :
>                                     ((Opponent == "GreenChess") ?
>                                     GreenChessOpponentIcon :
>                                     UCIEngineOpponentIcon)
>
```

```
4960a5021,5030
>    // swap the color to move next //
>    @MainActor @IBAction func SwapColorToMoveButtonAction(_:AnyObject)
>    {
>       // swap the color to move //
>       GCUI.Attributes.SwapColorToMove()
>
>       // update the position editor toolbar items //
>       UpdatePositionEditorToolbarButtons()
>    }
>
4965,4969c5035,5075
<       (PosMovingPiece, PosCursor) = (Type, NSCursor.init(image:(Type == 0) ?
<       Graphics.CGImageToNSImage(ImageRenderer(content:IconImage("xmark.square",
<       .red)).cgImage!, Size:Dims.Square.Width) : Graphics.CGImageToNSImage(GCUI
<       .BPComp.GetPieceCGImage(Type), Size:Dims.Square.Width), hotSpot:CGPoint(x:
<       Dims.Square.Width * 0.5, y: Dims.Square.Width * 0.5)))
---
>       (PositionMovingPiece, PosCursor) = (Type, (Type == 0) ? NSCursor
>       .init(image:Eraser, hotSpot:EditorHotspotPoint) : ((Type == 10) ? NSCursor
>       .init(image:self.PositionEditorEP.image!, hotSpot:EditorHotspotPoint) :
>       NSCursor.init(image:GCUI.BPComp.GetPieceNSImage(Type), hotSpot:
>       EditorHotspotPoint)))
>    }
>
>    // the position editor eraser action //
>    @MainActor @IBAction func PositionEditorEraser(_:AnyObject) {SetCursor(0)}
>
>    // the position editor en passant square action //
>    @MainActor @IBAction func PositionEditorEPAction(_:AnyObject) {SetCursor(10)}
>
>    // the position editor second menu action //
>    @MainActor @IBAction func PositionEditorSecondMenuAction(_:AnyObject)
>    {
>       // if in the second position editor menu //
>       if Inf.Vals("SecondPositionMenu").VB
>       {
>          // set that no longer in the second position editor menu //
>          Inf.TurnOffVB("VB", Key:"SecondPositionMenu")
>
>          // add the position editor piece items //
>          AddPositionEditorPieceItems()
>       }
>       // in the first position editor menu //
>       else
>       {
>          // set that in the second position editor menu //
>          Inf.TurnOnVB("VB", Key:"SecondPositionMenu")
>
>          // add the castling toolbar items to the toolbar //
>          AddToolbarItems([300, 200, 201, 216, 217, 218, 219, 220])
>
>          // set position editor second menu button image to going left icon //
>          PositionEditorSecondMenu.image = NSImage(named:NSImage
>                                                .goLeftTemplateName)
>
>          // update the position editor toolbar buttons //
>          UpdatePositionEditorToolbarButtons()
>       }
4970a5077,5101
>
>    // the position pieces action for the position editor //
>    @MainActor @IBAction func PositionPiecesAction(_:AnyObject)
>    {
>       // return if the selected items isn't in the position pieces dictionary //
>       if PosPieces[MainToolbar!.selectedItemIdentifier!.rawValue] == nil
>       {
>          // exit the function //
>          return
>       }
>
>       // set the position cursor to the current piece type //
>       SetCursor(PosPieces[MainToolbar!.selectedItemIdentifier!.rawValue]!)
>    }
>
>    // update the castle button action //
>    @MainActor @IBAction func UpdateCastleButtonAction(_:AnyObject)
>    {
>       // 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")
4971a5103,5109
>       // deselect the selected toolbar item //
>       MainToolbar.selectedItemIdentifier = nil
>
>       // update the position editor toolbar buttons //
>       UpdatePositionEditorToolbarButtons()
>    }
>
5077a5216,5267
>    }
>
>    // update the castle item labels on the toolbar //
>    @MainActor func SetCastLabel(_ ToolbarItem:NSToolbarItem, On:Bool, Kingside:
>                                Bool)
>    {
>       // update the castle item labels on the toolbar //
>       ToolbarItem.label = (Kingside ? "King" : "Queen") + "side Castle " + (On
>                           ? "On" : "Off")
>    }
>
>    // update the position editor toolbar buttons //
>    @MainActor func UpdatePositionEditorToolbarButtons()
>    {
>       // if not currently in the position editor, return //
>       if !Inf.Vals("InPositionEditor").VB {return}
>
>       // set white moves button image and label based on color to move next //
>       (WhiteMovesToolbarButton.image, WhiteMovesToolbarButton.label) = (GCUI
>       .BPComp.GetPieceNSImage(GCUI.Attributes.WhiteStarts() ? 1 : -1), (GCUI
>       .Attributes.WhiteStarts() ? "White" : "Black") + " Moves")
>
>       // loop over all the toolbar items //
>       for TheItem in MainToolbar.items
>       {
>          // if the piece exists //
>          if self.PosPieces[TheItem.itemIdentifier.rawValue] != nil
>          {
>             // update the item image to the correct piece set and piece type //
>             TheItem.image = GCUI.BPComp.GetPieceNSImage(PosPieces[TheItem
>                                .itemIdentifier.rawValue]!)
>          }
>       }
>
>       // update the white kingside castle toolbar button label //
>       SetCastLabel(WhiteKingsideCastle, On:Inf.Vals("WhiteKingsideCastle").VB,
>                   Kingside:true)
>
>       // update the white queenside castle toolbar button label //
>       SetCastLabel(WhiteQueensideCastle, On:Inf.Vals("WhiteQueensideCastle")
>                   .VB, Kingside:false)
>
>       // update the black kingside castle toolbar button label //
>       SetCastLabel(BlackKingsideCastle, On:Inf.Vals("BlackKingsideCastle").VB,
>                   Kingside:true)
>
>       // update the black queenside castle toolbar button label //
>       SetCastLabel(BlackQueensideCastle, On:Inf.Vals("BlackQueensideCastle")
>                   .VB, Kingside:false)
>
>       // update position editor cursor //
>       SetCursor(PositionMovingPiece)
5392c5582
<             Update:false), 0.5 * (265 - Inf.VF("IllustrationSize")))
---
>             Update:false), 0.5 * (250 - Inf.VF("IllustrationSize")))
```