```
2c2
< // Swift source code of Classes.swift Version 9.0.4 at greenchess.com //
---
> // Swift source code of Classes.swift Version 9.0.3 at greenchess.com //
55c55,65
<    init(_ TheEnd:Int, Inf:Int, Analy:Int) {(End, InfoEnd, AnalysisEnd) = (TheEnd, Inf, Analy)}
---
>    init(_ End:Int, Info:Int, Analysis:Int)
>    {
>       // initialize the end key //
>       self.End = End
>
>       // initialize the information end key //
>       self.InfoEnd = Info
>
>       // initialize the analysis end key //
>       self.AnalysisEnd = Analysis
>    }
163c173
<       for (Key, VDefMinMaxInt) in VDefMinMaxIntItems {if VDefMinMaxInt.count == 3
---
>       for (Key, VDefMinMaxInt) in VDefMinMaxIntItems
165,166c175,179
<          // set the integer value default //
<          Item(Tag, Key:Key)?.IVals.VDI = VDefMinMaxInt[0]
---
>          // if there are three elements in the integer array //
>          if VDefMinMaxInt.count == 3
>          {
>             // set the integer value default //
>             Item(Tag, Key:Key)?.IVals.VDI = VDefMinMaxInt[0]
168,169c181,182
<          // set the integer value minimum //
<          Item(Tag, Key:Key)?.IVals.VMinInt = VDefMinMaxInt[1]
---
>             // set the integer value minimum //
>             Item(Tag, Key:Key)?.IVals.VMinInt = VDefMinMaxInt[1]
171,173c184,187
<          // set the integer value maximum //
<          Item(Tag, Key:Key)?.IVals.VMaxInt = VDefMinMaxInt[2]
<       }}
---
>             // set the integer value maximum //
>             Item(Tag, Key:Key)?.IVals.VMaxInt = VDefMinMaxInt[2]
>          }
>       }
183c197
<       for Index in Start..<End
---
>       for Index in Start ..< End
207c221
<       Keys = InfoKeys(Keys.End - Shift, Inf:Keys.End - Shift, Analy:Keys.End - Shift)
---
>       Keys = InfoKeys(Keys.End - Shift, Info:Keys.End - Shift, Analysis:Keys.End - Shift)
217c231
<       Keys = InfoKeys(Keys.End, Inf:Keys.End, Analy:Keys.AnalysisEnd - Shift)
---
>       Keys = InfoKeys(Keys.End, Info:Keys.End, Analysis:Keys.AnalysisEnd - Shift)
322c336
<       BackupKeys = InfoKeys(Keys.End, Inf:Keys.InfoEnd, Analy:Keys.InfoEnd)
---
>       BackupKeys = InfoKeys(Keys.End, Info:Keys.InfoEnd, Analysis:Keys.InfoEnd)
325c339
<       for Index in 0..<Keys.InfoEnd {Item(Tag, Key:Index)?.IValsBackup = Vals(Index)}
---
>       for Index in 0 ..< Keys.InfoEnd {Item(Tag, Key:Index)?.IValsBackup = Vals(Index)}
342c356
<       for Index in 0..<Keys.InfoEnd {Item(Tag, Key:Index)?.IVals = ValsBackup(Index)}
---
>       for Index in 0 ..< Keys.InfoEnd {Item(Tag, Key:Index)?.IVals = ValsBackup(Index)}
431,433c445,450
<       // update the information keys to new values and the index to zero //
<       (Keys, Index) = (InfoKeys(Keys.End, Inf:Keys.End, Analy:Keys.End), 0)
<
---
>       // update the information keys to new values //
>       Keys = InfoKeys(Keys.End, Info:Keys.End, Analysis:Keys.End)
>
>       // set index to zero //
>       Index = 0
>
455c472
<             Update(Value.Tag, VSItems:[Key:Prefs.PrefString(Key, Default:Vals(Key).VDS)])
---
>             Update(Value.Tag, VSItems:[Key:Prefs.PreferenceString(Key, Default:Vals(Key).VDS)])
461c478
<             Update(Value.Tag, VBItems:[Key:Prefs.PrefBool(Key, Default:Vals(Key).VDB)])
---
>             Update(Value.Tag, VBItems:[Key:Prefs.PreferenceBool(Key, Default:Vals(Key).VDB)])
467,468c484,485
<             Update(Value.Tag, VIItems:[Key:Prefs.PrefInt(Key, Default:Vals(Key).VDI, Min:Vals(Key)
<                   .VMinInt, Max:Vals(Key).VMaxInt)])
---
>             Update(Value.Tag, VIItems:[Key:Prefs.PreferenceInt(Key, Default:Vals(Key).VDI, Min:
>                   Vals(Key).VMinInt, Max:Vals(Key).VMaxInt)])
477c494
<       for (Key, Value) in Info {if Tags.contains(Value.Tag)
---
>       for (Key, Value) in Info
479,481c496,503
<          // update the preference string for the specified suffix //
<          Update(Value.Tag, VSItems:[Key:Prefs.PrefString(Key + Suffix, Default:Vals(Key).VDS)])
<       }}
---
>          // if the info item contains a tag in the tags array //
>          if Tags.contains(Value.Tag)
>          {
>             // update the preference string for the specified suffix //
>             Update(Value.Tag, VSItems:[Key:Prefs.PreferenceString(Key + Suffix, Default:Vals(Key)
>                   .VDS)])
>          }
>       }
491c513
<       for Index in 0..<Keys.End
---
>       for Index in 0 ..< Keys.End
533c555
<    func SaveSubset(_ End:Int) {Keys = InfoKeys(End, Inf:End, Analy:End)}
---
>    func SaveSubset(_ End:Int) {Keys = InfoKeys(End, Info:End, Analysis:End)}
545c567
<       Keys = InfoKeys(Keys.End + 1, Inf:Keys.End + 1, Analy:Keys.End + 1)
---
>       Keys = InfoKeys(Keys.End + 1, Info:Keys.End + 1, Analysis:Keys.End + 1)
558c580
<          Keys = InfoKeys(Keys.End + 1, Inf:Keys.InfoEnd, Analy:Keys.End + 1)
---
>          Keys = InfoKeys(Keys.End + 1, Info:Keys.InfoEnd, Analysis:Keys.End + 1)
563,565c585,590
<
<       // return that were not able to redo to the next value string //
<       return false
---
>       // the end key is out of range of the stored information //
>       else
>       {
>          // return that were not able to redo to the next value string //
>          return false
>       }
575c600
<          Keys = InfoKeys(Keys.End - 1, Inf:Keys.InfoEnd, Analy:Keys.End - 1)
---
>          Keys = InfoKeys(Keys.End - 1, Info:Keys.InfoEnd, Analysis:Keys.End - 1)
580,582c605,610
<
<       // return that were not able to undo to the previous value string //
<       return false
---
>       // the end key is already at zero //
>       else
>       {
>          // return that were not able to undo to the previous value string //
>          return false
>       }
592c620
<       for Index in 0..<Keys.End {List.append(Vals(Index).VS)}
---
>       for Index in 0 ..< Keys.End {List.append(Vals(Index).VS)}
600c628
< class BoardImageView:NSView
---
> class GameExplorerView:NSView
667c695,696
<    func SquareFromUCI(_ UCIMove:String, Start:Bool, Dims:BoardDimensions, Rotated:Bool) -> APoint
---
>    func SquareFromUCIMove(_ UCIMove:String, StartSquare:Bool, Dims:BoardDimensions, Rotated:Bool) ->
>                       APoint
673c702
<       if Start && (UCIMove.count >= 2)
---
>       if StartSquare && (UCIMove.count >= 2)
680c709
<       else if !Start && (UCIMove.count >= 4)
---
>       else if !StartSquare && (UCIMove.count >= 4)
699c728
<    let PromoteDict:[Character:Int] = ["n":2, "b":3, "r":4, "q":5]
---
>    let ExtractPromotionDictionary:[Character:Int] = ["n":2, "b":3, "r":4, "q":5]
750,752d778
<    // check if have a castling move or not //
<    func Castl(_ Move:String, Piece:Int) -> Bool {return (UCIMove == Move) && (Piece == MovingPiece)}
<
756,760c782,789
<       // return if piece is not in the promotion dictionary //
<       if PromoteToDictionary[ReturnCode] == nil {return}
<
<       // set the moving piece from the button return code //
<       Set(BoardIsWhite(Board) ? 5 - Int(ReturnCode) : Int(ReturnCode) - 5)
---
>       // if the piece to promote to is in the promotion dictionary //
>       if PromoteToDictionary[ReturnCode] != nil
>       {
>          // set the moving piece from the button return code //
>          Set(5 - Int(ReturnCode))
>
>          // if black is currently moving, make the moving piece black //
>          if BoardIsBlack(Board) {MovingPiece *= -1}
762,763c791,793
<       // update the last move on the move list to show a promotion occurred //
<       UCIMove += PromoteToDictionary[ReturnCode]!
---
>          // update the last move on the move list to show a promotion occurred //
>          UCIMove += PromoteToDictionary[ReturnCode]!
>       }
767c797
<    func PrepareAnimation(_ Start:APoint, End:APoint, Rate:Int)
---
>    func PrepareAnimation(_ AnimStart:APoint, AnimEnd:APoint, AnimationRate:Int)
772,778c802,821
<       // create the animation from and to values, position, and duration //
<       (MoveAnimation.fromValue, MoveAnimation.toValue, position, MoveAnimation.duration) =
<       (NSValue(point:IllegalMove ? End.GetCoord() : Start.GetCoord()), NSValue(point:IllegalMove ?
<       Start.GetCoord() : End.GetCoord()), IllegalMove ? Start.GetCoord() : End.GetCoord(),
<       IllegalMove ? 0.25 : CFTimeInterval(sqrt(pow(End.XF - Start.XF, 2) + pow(End.YF - Start.YF,
<       2)) * CGFloat(101 - Rate) / 50))
<
---
>       // create the animation fromvalue //
>       MoveAnimation.fromValue = NSValue(point:AnimStart.GetCoord())
>
>       // remove the animation effects //
>       CATransaction.setDisableActions(true)
>
>       // create the animation tovalue //
>       MoveAnimation.toValue = NSValue(point:AnimEnd.GetCoord())
>
>       // remove the animation effects //
>       CATransaction.setDisableActions(true)
>
>       // set the animation final position //
>       position = AnimEnd.GetCoord()
>
>       // if undoing an illegal move //
>       MoveAnimation.duration = IllegalMove ? 0.25 : CFTimeInterval(sqrt(pow(AnimEnd.XF - AnimStart
>                                                 .XF, 2) + pow(AnimEnd.YF - AnimStart.YF, 2)) *
>                                                 CGFloat(101 - AnimationRate) / 50)
>
792,793c835,842
<       // set the moving piece image contents and unhide the piece //
<       (contents, isHidden) = (BPComp.GetPieceNSImage(MovingPiece), false)
---
>       // set the moving piece image contents //
>       contents = BPComp.GetPieceNSImage(MovingPiece)
>
>       // remove the animation effects //
>       CATransaction.setDisableActions(true)
>
>       // unhide the moving piece contents //
>       isHidden = false
799,801c848,863
<       // extract a pawn promotion if needed //
<       PromotionType = (PromoteDict[UCIMove.last!] != nil) ? ((UCIMove.prefix(4).last! == "1") ?
<                       -PromoteDict[UCIMove.last!]! : PromoteDict[UCIMove.last!]!) : 0
---
>       // initialize promotion type to zero //
>       PromotionType = 0
>
>       // if the uci move contains a pawn promotion //
>       if UCIMove.count == 5
>       {
>          // if the piece promotion to exists //
>          if ExtractPromotionDictionary[UCIMove.last!] != nil
>          {
>             // update the promoted piece type //
>             PromotionType = ExtractPromotionDictionary[UCIMove.last!]!
>
>             // if pawn moved to the first rank, set to be a black piece //
>             if UCIMove.prefix(4).last! == "1" {PromotionType *= -1}
>          }
>       }
805c867
<    func UpdateLoc(_ MouseLocation:CGPoint)
---
>    func UpdateLocation(_ MouseLocation:CGPoint)
818c880
<    func Reset() {(MovingPiece, PromotionType) = (0, 0)}
---
>    func Reset() {MovingPiece = 0; PromotionType = 0}
821c883
<    func UpdatePos() {CATransaction.setDisableActions(true); position = Location.GetCGPoint()}
---
>    func UpdatePosition() {CATransaction.setDisableActions(true); position = Location.GetCGPoint()}
824c886,893
<    func UpdateLocAndPos(_ Event:NSEvent) {UpdateLoc(Event.locationInWindow); UpdatePos()}
---
>    func UpdateLocationAndPosition(_ Event:NSEvent)
>    {
>       // update the current location of the moving piece //
>       UpdateLocation(Event.locationInWindow)
>
>       // update the current position of the moving piece //
>       UpdatePosition()
>    }
870c939
<    // hide the piece animating and update to the promoted type if needed //
---
>    // hide the piece animating and update to the promoted type if neeeded //
874c943
<    func UnHideMovingPiece() {UpdatePos(); SetContents()}
---
>    func UnHideMovingPiece() {UpdatePosition(); SetContents()}
880c949
<       UpdateLocAndPos(Event)
---
>       UpdateLocationAndPosition(Event)
882,884c951,955
<       // add the end square to the uci move and hide the moving piece layer contents //
<       (UCIMove, isHidden) = (UCIMove + SquareLabel.FileString(Location.XI) + SquareLabel
<       .RankString(Location.YI), true)
---
>       // add the end square to the uci move //
>       UCIMove += SquareLabel.FileString(Location.XI) +  SquareLabel.RankString(Location.YI)
>
>       // hide the moving piece layer contents //
>       isHidden = true
890,892c961,974
<       // set delegate, board and piece compositor, hidden value, shadow opacity value, and bounds //
<       (MoveAnimation.delegate, self.BPComp, isHidden, shadowOpacity, bounds) = (Delegate, BPComp,
<       true, 0.5, CGRect(x:0, y:0, width:Dims.Square.Width, height:Dims.Square.Width))
---
>       // set the animation delegate to be from the main board view //
>       MoveAnimation.delegate = Delegate
>
>       // set the board and piece compositor //
>       self.BPComp = BPComp
>
>       // set the moving piece layer contents to be hidden //
>       isHidden = true
>
>       // set the moving piece layer contents shadow opacity to 1 //
>       shadowOpacity = 1
>
>       // set the moving piece layer contents bounds //
>       bounds = CGRect(x:0, y:0, width:Dims.ViewSquareSize(2), height:Dims.ViewSquareSize(2))
917,918c999,1009
<       // update the variables //
<       (self.Created, self.Bounds, self.Image, Empty) = (Created, Bounds, Image, false)
---
>       // save date the display image was created //
>       self.Created = Created
>
>       // save the bounds of the display image //
>       self.Bounds = Bounds
>
>       // save the display image //
>       self.Image = Image
>
>       // set that image is not empty //
>       Empty = false
967c1058
<    func ResetImages() {(DImage1.Empty, DImage2.Empty) = (true, true)}
---
>    func ResetImages() {DImage1.Empty = true; DImage2.Empty = true}
1067a1159,1161
>
>    // a list index //
>    var Index:Int = 0
1072,1073c1166,1167
<    // the visible context //
<    var VisibleContext:CGContext!
---
>    // the full context //
>    var FullContext:CGContext!
1085c1179
<    func Snapshot() -> CGImage {Refresh(false); return VisibleContext.makeImage()!}
---
>    func Snapshot() -> CGImage {Refresh(false); return FullContext.makeImage()!}
1106,1108c1200,1207
<       // clear the diagram, background, and visible context layers //
<       (Diagram, Background, VisibleContext) = (CoreGraphics().CreateLayer(1, Height:1),
<       CoreGraphics().CreateLayer(1, Height:1), CoreGraphics().CreateContext(1, Height:1))
---
>       // clear the diagram layer //
>       Diagram = CoreGraphics().CreateLayer(1, Height:1)
>
>       // clear the background layer //
>       Background = CoreGraphics().CreateLayer(1, Height:1)
>
>       // clear the full context //
>       FullContext = CoreGraphics().CreateContext(1, Height:1)
1126c1225
<       if Diagram != nil {VisibleContext.draw(Diagram, at:CGPoint(x:XOffset, y:0))}
---
>       if Diagram != nil {FullContext.draw(Diagram, at:CGPoint(x:XOffset, y:0))}
1132c1231
<       SketchImages.Update(ViewBounds, Image:VisibleContext.makeImage())
---
>       SketchImages.Update(ViewBounds, Image:FullContext.makeImage())
1148c1247
<       for ThePath in Paths
---
>       for Index in 0 ..< Paths.count
1154c1253
<          Diagram.context?.setStrokeColor(ThePath.Color)
---
>          Diagram.context?.setStrokeColor(Paths[Index].Color)
1157c1256
<          Diagram.context?.addLines(between:ThePath.Points)
---
>          Diagram.context?.addLines(between:Paths[Index].Points)
1182,1183c1281,1288
<       // reset the line segment start index, drawing and previous point //
<       (SegmentStart, Drawing, PreviousPoint) = (0, true, NSEvent.mouseLocation)
---
>       // reset the line segment start index to zero //
>       SegmentStart = 0
>
>       // set that currently drawing to the screen //
>       Drawing = true
>
>       // initialize the previous point //
>       PreviousPoint = NSEvent.mouseLocation
1187,1188c1302,1306
<       // adjust the y origin by the y offset and set that currently drawing a path //
<       (Origin.y, Drawing) = (Origin.y + YOffset, true)
---
>       // adjust the y origin by the y offset //
>       Origin.y += YOffset
>
>       // set that currently drawing a path //
>       Drawing = true
1202,1204c1310,1313
<                      .foreverDuration, mode:RunLoop.Mode.eventTracking, handler:{(Event, Done) in
<       guard let Event = Event else {return}; if Event.type == .leftMouseUp {if Path
<       .Points.count > 1 {Paths.append(Path)}; Done.pointee = true}})
---
>                      .foreverDuration, mode:RunLoop.Mode.eventTracking, handler:{(Event,
>                      Done) in guard let Event = Event else {return}; if Event.type ==
>                      .leftMouseUp {if Path.Points.count > 1 {Paths.append(Path)};
>                      Done.pointee = true}})
1214c1323
<       while Drawing {if (SketchImages.ReadyImage())
---
>       while Drawing
1216,1217c1325,1332
<          // set that currently displaying image to the screen //
<          Displaying = true
---
>          // if not currently displaying to the screen //
>          if !Displaying
>          {
>             // if have an image ready to display to the screen //
>             if SketchImages.ReadyImage()
>             {
>                // set that currently displaying image to the screen //
>                Displaying = true
1219,1221c1334,1338
<          // display the image to the screen on the main thread //
<          DispatchQueue.main.async {self.View.display(self.SketchImages.GetBounds())}
<       }}
---
>                // display the image to the screen on the main thread //
>                DispatchQueue.main.async {self.View.display(self.SketchImages.GetBounds())}
>             }
>          }
>       }
1228c1345
<       while Drawing {if (SketchImages.EmptyImage() && (Path.isEmpty
---
>       while Drawing
1230,1234c1347,1351
<          // expand the end line segment index to draw //
<          SegmentEnd = Path.Points.count - 1
<
<          // begin the draw path //
<          Diagram.context?.beginPath()
---
>          // if there is an empty image available to draw into and if there are points in the path //
>          if SketchImages.EmptyImage() && !Path.Points.isEmpty
>          {
>             // set the end line segment index to draw //
>             SegmentEnd = Path.Points.count - 1
1236,1237c1353,1354
<          // add the path segment to the diagram layer //
<          Diagram.context?.addLines(between:Array(Path.Points[SegmentStart...SegmentEnd]))
---
>             // begin the draw path //
>             Diagram.context?.beginPath()
1239,1243c1356,1357
<          // if the bounding box is not nil //
<          if Diagram.context?.path?.boundingBox != nil
<          {
<             // set the segment start to the end value and the bounding box of //
<             (SegmentStart, Bounds) = (SegmentEnd, (Diagram.context?.path?.boundingBox)!)
---
>             // add the path segment to the diagram layer //
>             Diagram.context?.addLines(between:Array(Path.Points[SegmentStart...SegmentEnd]))
1245,1247c1359,1370
<             // expand the bounds to encompass the entire line segment to draw //
<             Bounds = CGRect(x:Int(Bounds.minX + XOffset) - 4, y:Int(Bounds.minY) - 4, width:
<                      Int(Bounds.width) + 8, height:Int(Bounds.height) + 8)
---
>             // if the bounding box is not nil //
>             if Diagram.context?.path?.boundingBox != nil
>             {
>                // set the segment start to the end value //
>                SegmentStart = SegmentEnd
>
>                // get the bounding box //
>                Bounds = (Diagram.context?.path?.boundingBox)!
>
>                // expand the bounds to encompass the entire line segment to draw //
>                Bounds = CGRect(x:Int(Bounds.minX + XOffset) - 4, y:Int(Bounds.minY) - 4, width:
>                         Int(Bounds.width) + 8, height:Int(Bounds.height) + 8)
1249,1250c1372,1373
<             // stroke the path to the diagram layer //
<             Diagram.context?.strokePath()
---
>                // stroke the path to the diagram layer //
>                Diagram.context?.strokePath()
1252,1253c1375,1376
<             // create the context to encompass the specified bounds //
<             Context = CoreGraphics().CreateContext(Bounds.width, Height:Bounds.height)
---
>                // create the context to encompass the specified bounds //
>                Context = CoreGraphics().CreateContext(Bounds.width, Height:Bounds.height)
1255,1259c1378,1388
<             // if the diagram layer exists //
<             if Diagram != nil
<             {
<                // draw the diagram into the context at the specified point //
<                Context.draw(Diagram, at:CGPoint(x:XOffset - Bounds.minX, y:-Bounds.minY))
---
>                // if the diagram layer exists //
>                if Diagram != nil
>                {
>                   // draw the diagram into the context at the specified point //
>                   Context.draw(Diagram, at:CGPoint(x:XOffset - Bounds.minX, y:-Bounds.minY))
>                }
>
>                // save the image to an available empty image slot //
>                SketchImages.Update(CGRect(x:Bounds.minX + 0.5, y:Bounds.minY + 0.5, width:Bounds
>                         .width + 0.5, height:Bounds.height + 0.5), Image:Context
>                         .makeImage()!)
1261,1264d1389
<
<             // save the image to an available empty image slot with the specified bounds //
<             SketchImages.Update(CGRect(x:Bounds.minX + 0.5, y:Bounds.minY + 0.5, width:Bounds.width
<                      + 0.5, height:Bounds.height + 0.5), Image:Context.makeImage()!)
1266c1391
<       }}
---
>       }
1294,1300c1419,1439
<       // set all the diagram values //
<       (View, self.YOffset, self.XOffset, self.Margin, ViewBounds, Background, Diagram,
<       VisibleContext) = (View, YOffset, 0, Margin, CGRect(x:0, y:0, width:Width * 0.5, height:Height
<       * 0.5), CoreGraphics().CreateLayer(Width, Height:Height), CoreGraphics()
<       .CreateLayer(max(Width, Length), Height:max(Width, Length)), CoreGraphics().CreateContext(Width, Height:
<       Height))
---
>    {
>       // set the view //
>       self.View = View
>
>       // set the y offset //
>       self.YOffset = YOffset
>
>       // set the margin //
>       self.Margin = Margin
>
>       // set the x offset to zero //
>       XOffset = 0
>
>       // set the screen view bounds //
>       ViewBounds = CGRect(x:0, y:0, width:Width * 0.5, height:Height * 0.5)
>
>       // create the background layer //
>       Background = CoreGraphics().CreateLayer(Width, Height:Height)
>
>       // create the diagram layer //
>       Diagram = CoreGraphics().CreateLayer(max(Width, Length), Height:max(Width, Length))
1301a1441,1443
>
>       // create the full context //
>       FullContext = CoreGraphics().CreateContext(Width, Height:Height)
1347a1490,1492
>
>    // an index in a list //
>    var Index:Int = 0
1350,1352c1495,1497
<    let SoundNames:[String] = ["sound_thump", "sound_wrongsquare", "sound_pop", "sound_click",
<                               "sound_clear", "sound_warning", "sound_switch", "sound_buttonclick",
<                               "sound_camera", "sound_marker", "sound_movie"]
---
>    let GameSoundNames:[String] = ["sound_thump", "sound_wrongsquare", "sound_pop", "sound_click",
>                                   "sound_clear", "sound_warning", "sound_switch",
>                                   "sound_buttonclick", "sound_camera", "sound_marker", "sound_movie"]
1355,1357c1500,1502
<    let SoundDict:[String:Int] = ["thump":0, "wrongsquare":1, "pop":2, "click":3, "clear":4,
<                                  "warning":5, "switch":6, "buttonclick":7, "camera":8, "marker":9,
<                                  "movie":10]
---
>    let SoundDictionary:[String:Int] = ["thump":0, "wrongsquare":1, "pop":2, "click":3, "clear":4,
>                                        "warning":5, "switch":6, "buttonclick":7, "camera":8,
>                                        "marker":9, "movie":10]
1368,1370d1512
<       // return if the sound doesn't exist //
<       if SoundDict[Sound] == nil {return}
<
1373,1375d1514
<
<       // set the game sound selection //
<       GameSound = GameSounds[SoundDict[Sound]!]
1377,1378c1516,1523
<       // set the game sound volume //
<       GameSound.volume = SoundVolumes[SoundDict[Sound]!]
---
>       // if the game sound exists //
>       if SoundDictionary[Sound] != nil
>       {
>          // set the game sound selection //
>          GameSound = GameSounds[SoundDictionary[Sound]!]
>
>          // set the game sound volume //
>          GameSound.volume = SoundVolumes[SoundDictionary[Sound]!]
1380,1381c1525,1527
<       // play the game sound //
<       GameSound.play()
---
>          // play the game sound //
>          GameSound.play()
>       }
1384,1385c1530,1539
<    // override the default initializer to add the game sounds //
<    override init() {for TheSound in SoundNames {GameSounds.append(NSSound(named:TheSound)!)}}
---
>    // override the default initializer //
>    override init()
>    {
>       // loop over all the game sounds //
>       for Index in 0 ..< GameSoundNames.count
>       {
>          // add the corresponding game sound from the game sound names //
>          GameSounds.append(NSSound(named:GameSoundNames[Index])!)
>       }
>    }
1398c1552
<    var TheDateFormat:DateFormatter = DateFormatter()
---
>    var TheDateFormatter:DateFormatter = DateFormatter()
1419c1573
<    @IBOutlet var WhiteWon:NSButton!
---
>    @IBOutlet var WhiteWonButton:NSButton!
1422c1576
<    @IBOutlet var BlackWon:NSButton!
---
>    @IBOutlet var BlackWonButton:NSButton!
1458c1612
<    func GetDateGamePlayed() -> String {return TheDateFormat.string(from:GameDate.dateValue)}
---
>    func GetDateGamePlayed() -> String {return TheDateFormatter.string(from:GameDate.dateValue)}
1461,1464c1615
<    func GetCurrentDate() -> String {return TheDateFormat.string(from:Date())}
<
<    // toggle the key //
<    func ToggleKey(_ Key:String) {GameSounds.Play("click"); Inf.ToggleVB("VB", Key:Key)}
---
>    func GetCurrentDate() -> String {return TheDateFormatter.string(from:Date())}
1466,1478d1616
<    // toggle the result //
<    func ToggleRes(_ Key:String, Offs:[String], Buttons:[NSButton])
<    {
<       // toggle the key //
<       ToggleKey(Key)
<
<       // turn the specified button states off //
<       for TheButton in Buttons {TheButton.state = NSControl.StateValue.off}
<
<       // set specified keys to false //
<       for TheKey in Off {Inf.Update("VB", VBItems:[TheKey:false])}
<    }
<
1482,1483c1620,1633
<       // toggle the result //
<       ToggleRes("WhiteWonGame", Offs:["DrawnGame", "BlackWonGame"], Buttons:[BlackWon, DrawButton])
---
>       // play the game sound of a click //
>       GameSounds.Play("click")
>
>       // turn the black won button off //
>       BlackWonButton.state = NSControl.StateValue.off
>
>       // turn the draw button off //
>       DrawButton.state = NSControl.StateValue.off
>
>       // change the state of whether white won the game or not //
>       Inf.ToggleVB("VB", Key:"WhiteWonGame")
>
>       // set drawn game and black won game to false //
>       Inf.Update("VB", VBItems:["DrawnGame":false, "BlackWonGame":false])
1489,1490c1639,1652
<       // toggle the result //
<       ToggleRes("BlackWonGame", Offs:["DrawnGame", "WhiteWonGame"], Buttons:[WhiteWon, DrawButton])
---
>       // play the game sound of a click //
>       GameSounds.Play("click")
>
>       // turn the white won button off //
>       WhiteWonButton.state = NSControl.StateValue.off
>
>       // turn the draw button off //
>       DrawButton.state = NSControl.StateValue.off
>
>       // change the state of whether white won the game or not //
>       Inf.ToggleVB("VB", Key:"BlackWonGame")
>
>       // set drawn game and white won game to false //
>       Inf.Update("VB", VBItems:["DrawnGame":false, "WhiteWonGame":false])
1496,1497c1658,1671
<       // toggle the result //
<       ToggleRes("DrawnGame", Offs:["WhiteWonGame", "BlackWonGame"], Buttons:[WhiteWon, BlackWon])
---
>       // play the game sound of a click //
>       GameSounds.Play("click")
>
>       // turn the white won button off //
>       WhiteWonButton.state = NSControl.StateValue.off
>
>       // turn the black won button off //
>       BlackWonButton.state = NSControl.StateValue.off
>
>       // change the state of whether have drawn game or not //
>       Inf.ToggleVB("VB", Key:"DrawnGame")
>
>       // set white won game and black won game to false //
>       Inf.Update("VB", VBItems:["WhiteWonGame":false, "BlackWonGame":false])
1506,1507c1680,1687
<       // set the specified buttons to off //
<       for Button in [WhiteWon, BlackWon, DrawButton] {Button.state = NSControl.StateValue.off}
---
>       // set the white won button state to off //
>       WhiteWonButton.state = NSControl.StateValue.off
>
>       // set the black won button state to off //
>       BlackWonButton.state = NSControl.StateValue.off
>
>       // set the draw button state to off //
>       DrawButton.state = NSControl.StateValue.off
1509,1510c1689,1690
<       // specify the date format to use and set the date to the current date //
<       (TheDateFormat.dateFormat, GameDate.dateValue) = ("yyyy.MM.dd", Date())
---
>       // specify the date format to use //
>       TheDateFormatter.dateFormat = "yyyy.MM.dd"
1512,1521c1692,1738
<       // loop over the specified text fields //
<       for TheField in [WhiteNameTextField, BlackNameTextField, SiteLocationTextField,
<                        EventTitleTextField, RoundNumberTextField, TimeControlTextField,
<                        OpeningTextField, SectionTextField, BoardNumberTextField, WhiteEloTextField,
<                        BlackEloTextField, WhiteUSCFTextField, BlackUSCFTextField,
<                        AnnotatorTextField, ECOTextField]
<       {
<          // set the specified field string value to empty //
<          TheField!.stringValue = ""
<       }
---
>       // clear the whitename text field //
>       WhiteNameTextField.stringValue = ""
>
>       // clear the blackname text field //
>       BlackNameTextField.stringValue = ""
>
>       // clear the site location text field //
>       SiteLocationTextField.stringValue = ""
>
>       // clear the event title text field //
>       EventTitleTextField.stringValue = ""
>
>       // clear the round number text field //
>       RoundNumberTextField.stringValue = ""
>
>       // clear the time control text field //
>       TimeControlTextField.stringValue = ""
>
>       // clear the opening text field //
>       OpeningTextField.stringValue = ""
>
>       // clear the section text field //
>       SectionTextField.stringValue = ""
>
>       // clear the board number text field //
>       BoardNumberTextField.stringValue = ""
>
>       // clear the white elo text field //
>       WhiteEloTextField.stringValue = ""
>
>       // clear the black elo text field //
>       BlackEloTextField.stringValue = ""
>
>       // clear the white uscf text field //
>       WhiteUSCFTextField.stringValue = ""
>
>       // clear the black uscf text field //
>       BlackUSCFTextField.stringValue = ""
>
>       // clear the annotator text field //
>       AnnotatorTextField.stringValue = ""
>
>       // clear the eco text field //
>       ECOTextField.stringValue = ""
>
>       // set the game date to the current date //
>       GameDate.dateValue = Date()
1527,1543c1744,1763
<       // update all the text fields //
<       (WhiteNameTextField.stringValue, BlackNameTextField.stringValue, TimeControlTextField
<       .stringValue, OpeningTextField.stringValue, SectionTextField.stringValue, BoardNumberTextField
<       .stringValue, WhiteEloTextField.stringValue, BlackEloTextField.stringValue, WhiteUSCFTextField
<       .stringValue, BlackUSCFTextField.stringValue, AnnotatorTextField.stringValue, ECOTextField
<       .stringValue, SiteLocationTextField.stringValue, EventTitleTextField.stringValue,
<       RoundNumberTextField.stringValue) = (String(validatingUTF8:DataWhitePlayer(GameData))!,
<       String(validatingUTF8:DataBlackPlayer(GameData))!, String(validatingUTF8:
<       DataGetTimeControl(GameData))!, String(validatingUTF8:DataGetOpening(GameData))!,
<       String(validatingUTF8:DataGetSection(GameData))!, String(validatingUTF8:
<       DataGetBoardNumber(GameData))!, String(validatingUTF8:DataGetWhiteElo(GameData))!,
<       String(validatingUTF8:DataGetBlackElo(GameData))!, String(validatingUTF8:
<       DataGetWhiteUSCF(GameData))!, String(validatingUTF8:DataGetBlackUSCF(GameData))!,
<       String(validatingUTF8:DataGetAnnotator(GameData))!, String(validatingUTF8:
<       DataGetECO(GameData))!, String(validatingUTF8:DataGetSite(GameData))!,
<       String(validatingUTF8:DataGetEvent(GameData))!, String(validatingUTF8:
<       DataGetRound(GameData))!)
---
>       // update the white name text field //
>       WhiteNameTextField.stringValue = String(validatingUTF8:DataWhitePlayer(GameData))!
>
>       // update the black name text field //
>       BlackNameTextField.stringValue = String(validatingUTF8:DataBlackPlayer(GameData))!
>
>       // update the time control text field //
>       TimeControlTextField.stringValue = String(validatingUTF8:DataGetTimeControl(GameData))!
>
>       // update the opening text field //
>       OpeningTextField.stringValue = String(validatingUTF8:DataGetOpening(GameData))!
>
>       // update the section text field //
>       SectionTextField.stringValue = String(validatingUTF8:DataGetSection(GameData))!
>
>       // update the board number text field //
>       BoardNumberTextField.stringValue = String(validatingUTF8:DataGetBoardNumber(GameData))!
>
>       // update the white elo text field //
>       WhiteEloTextField.stringValue = String(validatingUTF8:DataGetWhiteElo(GameData))!
1545,1546c1765,1790
<       // switch over the game result value //
<       switch String(validatingUTF8:DataGetResult(GameData))!
---
>       // update the black elo text field //
>       BlackEloTextField.stringValue = String(validatingUTF8:DataGetBlackElo(GameData))!
>
>       // update the white uscf text field //
>       WhiteUSCFTextField.stringValue = String(validatingUTF8:DataGetWhiteUSCF(GameData))!
>
>       // update the black uscf text field //
>       BlackUSCFTextField.stringValue = String(validatingUTF8:DataGetBlackUSCF(GameData))!
>
>       // update the annotator text field //
>       AnnotatorTextField.stringValue = String(validatingUTF8:DataGetAnnotator(GameData))!
>
>       // update the eco text field //
>       ECOTextField.stringValue = String(validatingUTF8:DataGetECO(GameData))!
>
>       // update the site location text field //
>       SiteLocationTextField.stringValue = String(validatingUTF8:DataGetSite(GameData))!
>
>       // update the event title text field //
>       EventTitleTextField.stringValue = String(validatingUTF8:DataGetEvent(GameData))!
>
>       // update the round number text field //
>       RoundNumberTextField.stringValue = String(validatingUTF8:DataGetRound(GameData))!
>
>       // if white won the game //
>       if String(validatingUTF8:DataGetResult(GameData))! == "1-0"
>       {
1549,1553c1793,1800
<          case "1-0" : WhiteWon.state = NSControl.StateValue.on
<
<                       // set that white won the game //
<                       Inf.TurnOnVB("VB", Key:"WhiteWonGame")
<
---
>          WhiteWonButton.state = NSControl.StateValue.on
>
>          // set that white won the game //
>          Inf.TurnOnVB("VB", Key:"WhiteWonGame")
>       }
>       // if black won the game //
>       else if String(validatingUTF8:DataGetResult(GameData))! == "0-1"
>       {
1555,1559c1802,1809
<          case "0-1" : BlackWon.state = NSControl.StateValue.on
<
<                       // set that black won the game //
<                       Inf.TurnOnVB("VB", Key:"BlackWonGame")
<
---
>          BlackWonButton.state = NSControl.StateValue.on
>
>          // set that black won the game //
>          Inf.TurnOnVB("VB", Key:"BlackWonGame")
>       }
>       // if the game ended in a draw //
>       else if String(validatingUTF8:DataGetResult(GameData))! == "1/2-1/2"
>       {
1561,1567c1811,1814
<          case "1/2-1/2" : DrawButton.state = NSControl.StateValue.on
<
<                           // set that the game was drawn //
<                           Inf.TurnOnVB("VB", Key:"DrawnGame")
<
<          // the default case //
<          default : break
---
>          DrawButton.state = NSControl.StateValue.on
>
>          // set that the game was drawn //
>          Inf.TurnOnVB("VB", Key:"DrawnGame")
1570,1573c1817,1823
<       // exit if the date format is invalid //
<       if TheDateFormat.date(from:String(validatingUTF8:DataGetDate(GameData))!) == nil {return}
<
<       // update the date picker calendar //
<       GameDate.dateValue = TheDateFormat.date(from:String(validatingUTF8:DataGetDate(GameData))!)!
---
>       // if the date format is valid //
>       if TheDateFormatter.date(from:String(validatingUTF8:DataGetDate(GameData))!) != nil
>       {
>          // update the date picker calendar //
>          GameDate.dateValue = TheDateFormatter.date(from:String(validatingUTF8:
>                               DataGetDate(GameData))!)!
>       }
1756c2005
<    @IBOutlet var HighlightLastMove:NSButton!
---
>    @IBOutlet var HighlightLastMoveButton:NSButton!
1759c2008
<    @IBOutlet var HighlightCaptures:NSButton!
---
>    @IBOutlet var HighlightCapturedPiecesButton:NSButton!
1762c2011
<    @IBOutlet var HighlightThreats:NSButton!
---
>    @IBOutlet var HighlightThreatenedPiecesButton:NSButton!
1765c2014
<    @IBOutlet var MoveBySelectingSquares:NSButton!
---
>    @IBOutlet var MoveBySelectingSquaresButton:NSButton!
1768c2017
<    @IBOutlet var DisplayCaptures:NSButton!
---
>    @IBOutlet var DisplayCapturedPiecesButton:NSButton!
1817c2066,2069
<
---
>
>    // the list of board preference controls //
>    var BoardControlsList:[AnyObject] = []
```

```
1826a2079,2081
>     // the list of font preference controls //
>     var FontControlsList:[AnyObject] = []
>
1840a2096,2105
>
>     // update the board previews //
>     func UpdateBoardPreviews(_ BoardImage:NSImage, DiagramImage:NSImage)
>     {
>        // update the board preview image //
>        BoardPreview.image = BoardImage
>
>        // update the diagram preview image for the illustratons //
>        DiagramPreview.image = DiagramImage
>     }
1852,1857c2117,2133
<        // set the color view identifiers and controls //
<        (BoardLabelsColorView.Identifier, ClockColorView.Identifier, ClockBackgroundColorView
<        .Identifier, GameInfoColorView.Identifier, MoveListColorView.Identifier, GameListColorView
<        .Identifier, WindowBackgroundColorView.Identifier, PrefControls) = (0, 1, 2, 3, 4, 5, 6,
<        [20:BackgroundControls, 21:BoardControls, 22:FontControls, 23:StockfishControls, 24:
<        SpeedControls, 25:ClockControls, 26:MovieControls, 27:GalleryControls, 28:DiagramControls])
---
>        // set indentifier for board labels color view //
>        BoardLabelsColorView.Identifier = 0
>
>        // set identifier for clock color view //
>        ClockColorView.Identifier = 1
>
>        // set identifier for clock background color view //
>        ClockBackgroundColorView.Identifier = 2
>
>        // set identifier for game info color view //
>        GameInfoColorView.Identifier = 3
>
>        // set identifier for move list color view //
>        MoveListColorView.Identifier = 4
>
>        // set identifier for game list color view //
>        GameListColorView.Identifier = 5
1859,1873c2135,2136
<        // initialize the board controls //
<        BoardControls.Initialize([BoardPreview, BoardTexturesButton, PieceSetButton,
<                     AutosaveGameButton, GameSoundsButton, EcoBookButton, HighlightLastMove,
<                     HighlightCaptures, HighlightThreats, MoveBySelectingSquares, DisplayCaptures,
<                     RotateBoardButton, BorderTextureView, BorderTextureSlider, BorderVisibilityView,
<                     BorderVisibilitySlider, SquareBorderView, SquareBorderOpacitySlider,
<                     BoardRoughnessView, BoardRoughnessSlider, BoardTintColorView,
<                     BoardTintColorSlider, BoardTintVisibilityView, BoardTintVisibilitySlider],
<                     Frames:[[30, 215, 400, 400], [439, 590, 200, 25], [439, 550, 200, 25], [440,
<                     505, 215, 25], [440, 470, 215, 25], [440, 435, 215, 25], [440, 400, 215, 25],
<                     [440, 365, 215, 25], [440, 330, 215, 25], [440, 295, 215, 25], [440, 260, 215,
<                     25], [440, 225, 215, 25], [440, 164, 215, 25], [30, 165, 400, 25], [440, 134,
<                     215, 25], [30, 135, 400, 25], [440, 104, 215, 25], [30, 105, 400, 25], [440, 74,
<                     215, 25], [30, 75, 400, 25], [440, 44, 215, 25], [30, 45, 400, 25], [440, 14,
<                     215, 25], [30, 15, 400, 25]])
---
>        // set identifier for window background color view //
>        WindowBackgroundColorView.Identifier = 6
1874a2138,2172
>        // set the preference controls dictionary //
>        PrefControls = [20:BackgroundControls, 21:BoardControls, 22:FontControls, 23:
>                        StockfishControls, 24:SpeedControls, 25:ClockControls, 26:MovieControls,
>                        27:GalleryControls, 28:DiagramControls]
>
>        // set the board controls list //
>        BoardControlsList = [BoardPreview, BoardTexturesButton, PieceSetButton, AutosaveGameButton,
>                             GameSoundsButton, EcoBookButton, HighlightLastMoveMadeButton,
>                             HighlightCapturedPiecesButton, HighlightThreatenedPiecesButton,
>                             MoveBySelectingSquaresButton, DisplayCapturedPiecesButton,
>                             RotateBoardButton, BorderTextureView, BorderTextureSlider,
>                             BorderVisibilityView, BorderVisibilitySlider, SquareBorderView,
>                             SquareBorderOpacitySlider, BoardRoughnessView, BoardRoughnessSlider,
>                             BoardTintColorView, BoardTintColorSlider, BoardTintVisibilityView,
>                             BoardTintVisibilitySlider]
>
>        // initialize the board controls //
>        BoardControls.Initialize(BoardControlsList, Frames:[[30, 215, 400, 400], [439, 590, 200, 25],
>                                 [439, 550, 200, 25], [440, 505, 215, 25], [440, 470, 215, 25],
>                                 [440, 435, 215, 25], [440, 400, 215, 25], [440, 365, 215, 25],
>                                 [440, 330, 215, 25], [440, 295, 215, 25], [440, 260, 215, 25],
>                                 [440, 225, 215, 25], [440, 164, 215, 25], [30, 165, 400, 25],
>                                 [440, 134, 215, 25], [30, 135, 400, 25], [440, 104, 215, 25],
>                                 [30, 105, 400, 25], [440, 74, 215, 25], [30, 75, 400, 25],
>                                 [440, 44, 215, 25], [30, 45, 400, 25], [440, 14, 215, 25],
>                                 [30, 15, 400, 25]])
>
>        // set the font controls list //
>        FontControlsList = [BoardLabelsField, BoardLabelsFontButton, BoardLabelsBoldButton,
>                            BoardLabelsSizesButton, BoardLabelsColorView, GameInfoField,
>                            GameInfoFontButton, GameInfoBoldButton, GameInfoSizesButton,
>                            GameInfoColorView, MoveListField, MoveListFontButton, MoveListBoldButton,
>                            MoveListSizesButton, MoveListColorView, GameListField, GameListFontButton,
>                            GameListBoldButton, GameListSizesButton, GameListColorView]
>
1876,1886c2174,2180
<        FontControls.Initialize([BoardLabelsField, BoardLabelsFontButton, BoardLabelsBoldButton,
<                    BoardLabelsSizesButton, BoardLabelsColorView, GameInfoField, GameInfoFontButton,
<                    GameInfoBoldButton, GameInfoSizesButton, GameInfoColorView, MoveListField,
<                    MoveListFontButton, MoveListBoldButton, MoveListSizesButton, MoveListColorView,
<                    GameListField, GameListFontButton, GameListBoldButton, GameListSizesButton,
<                    GameListColorView], Frames:[[230, 595, 200, 25], [230, 570, 200, 25], [230, 545,
<                    200, 25], [230, 520, 200, 25], [30, 480, 600, 40], [230, 445, 200, 25], [230,
<                    420, 200, 25], [230, 395, 200, 25], [230, 370, 200, 25], [30, 330, 600, 40],
<                    [230, 295, 200, 25], [230, 270, 200, 25], [230, 245, 200, 25], [230, 220, 200,
<                    25], [30, 180, 600, 40], [230, 145, 200, 25], [230, 120, 200, 25], [230, 95, 200,
<                    25], [230, 70, 200, 25], [30, 30, 600, 40]])
---
>        FontControls.Initialize(FontControlsList, Frames:[[230, 595, 200, 25], [230, 570, 200, 25],
>                                [230, 545, 200, 25], [230, 520, 200, 25], [30, 480, 600, 40],
>                                [230, 445, 200, 25], [230, 420, 200, 25], [230, 395, 200, 25],
>                                [230, 370, 200, 25], [30, 330, 600, 40], [230, 295, 200, 25],
>                                [230, 270, 200, 25], [230, 245, 200, 25], [230, 220, 200, 25],
>                                [30, 180, 600, 40], [230, 145, 200, 25], [230, 120, 200, 25],
>                                [230, 95, 200, 25], [230, 70, 200, 25], [30, 30, 600, 40]])
1890,1892c2184,2187
<                            WindowBackgroundColorView, WindowTextureOpacitySlider,
<                            WindowTextureOpacityView], Frames:[[230, 420, 200, 200], [230, 330, 200,
<                            25], [30, 290, 600, 40], [30, 205, 600, 25], [30, 180, 600, 25]])
---
>                                     WindowBackgroundColorView, WindowTextureOpacitySlider,
>                                     WindowTextureOpacityView], Frames:[[230, 420, 200, 200],
>                                     [230, 330, 200, 25], [30, 290, 600, 40], [30, 205, 600, 25],
>                                     [30, 180, 600, 25]])
1896,1899c2191,2196
<                     ClockColorView, GameClockBackgroundColorField, ClockBackgroundColorView,
<                     ClockBackgroundVisibilitySlider, ClockBackgroundOpacityView], Frames:[[280, 515,
<                     100, 100], [30, 550, 200, 25], [230, 430, 200, 25], [30, 390, 600, 40], [230,
<                     310, 200, 25], [30, 270, 600, 40], [30, 180, 600, 25], [30, 155, 600, 25]])
---
>                               ClockColorView, GameClockBackgroundColorField,
>                               ClockBackgroundColorView, ClockBackgroundVisibilitySlider,
>                               ClockBackgroundOpacityView], Frames:[[280, 515, 100, 100],
>                               [30, 550, 200, 25], [230, 430, 200, 25], [30, 390, 600, 40],
>                               [230, 310, 200, 25], [30, 270, 600, 40], [30, 180, 600, 25],
>                               [30, 155, 600, 25]])
1903,1904c2200,2201
<                     SearchTimeView], Frames:[[30, 595, 600, 25], [30, 570, 600, 25], [30, 505, 600,
<                     25], [30, 480, 600, 25]])
---
>                               SearchTimeView], Frames:[[30, 595, 600, 25], [30, 570, 600, 25],
>                               [30, 505, 600, 25], [30, 480, 600, 25]])
1907,1909c2204,2206
<        DiagramControls.Initialize([DiagramPreview, DisplayIllustrationsButton,
<                       IllustrationSizeSlider, IllustrationSizeView], Frames:[[30, 520, 250, 250],
<                       [30, 480, 300, 25], [30, 270, 600, 25], [30, 245, 600, 25]])
---
>        DiagramControls.Initialize([DiagramPreview, DisplayIllustrationsButton, IllustrationSizeSlider,
>                                  IllustrationSizeView], Frames:[[30, 520, 250, 250],
>                                  [30, 480, 300, 25], [30, 270, 600, 25], [30, 245, 600, 25]])
1913,1916c2210,2213
<                          UCIEngineHashSlider, UCIEngineHashView, MultithreadUCIEngineButton,
<                          ShowStockfishAnalysisButton], Frames: [[30, 595, 600, 25], [30, 570, 600,
<                          25], [30, 505, 600, 25], [30, 480, 600, 25], [30, 390, 200, 25], [30, 300,
<                          200, 25]])
---
>                                    UCIEngineHashSlider, UCIEngineHashView,
>                                    MultithreadUCIEngineButton, ShowStockfishAnalysisButton], Frames:
>                                    [[30, 595, 600, 25], [30, 570, 600, 25], [30, 505, 600, 25],
>                                    [30, 480, 600, 25], [30, 390, 200, 25], [30, 300, 200, 25]])
1920,1921c2217,2218
<                       GalleryAnimationRateView], Frames:[[170, 590, 320, 25], [30, 505, 600, 25],
<                       [30, 480, 600, 25]])
---
>                                  GalleryAnimationRateView], Frames:[[170, 590, 320, 25],
>                                  [30, 505, 600, 25], [30, 480, 600, 25]])
1924,1925c2221,2222
<        MovieControls.Initialize([MovieFPSSlider, MovieFPSView], Frames:[[30, 595, 600, 25], [30, 570,
<                                 600, 25]])
---
>        MovieControls.Initialize([MovieFPSSlider, MovieFPSView], Frames:[[30, 595, 600, 25],
>                                 [30, 570, 600, 25]])
1949c2246,2255
<
---
>
>     // the number of captured pieces to display in the board //
>     var CapturedPieceCount:Int = 0
>
>     // a rank index in a list //
>     var Rank:Int = 0
>
>     // a file index in a list //
>     var File:Int = 0
>
1955a2262,2267
>     // a loop index //
>     var Index:Int = 0
>
>     // the current board and piece set //
>     var Set:Int = 0
>
1958c2270,2279
<
---
>
>
>     // the board cgimages //
>     var CGImagesBoard:[CGImage] = []
>
>     // the core graphics contexts for rendering //
>     var CGContexts:[CGContext] = []
>
>     // the board and pieces images //
>     var CGBoardAndPieces:[CGImage] = []
>
1997,1999d2317
<
<     // the recomposite background image //
<     var RecompositeBackground:CGImage!
2001,2004c2319,2320
<     // recomposite the board and the pieces //
<     func Recomposite(_ Size:CGFloat, BPData:ChessBoardData, InPositionEditor:Bool,
<                     FENEnpassantSquare:String, EnpassantSquare:APoint, RankAndFile:APoint,
<                     UpdateBoard:Bool) -> CGImage
---
>     // get the nsimage for the given board and pieces set and the specified size //
>     func GetNSImage(_ Size:Int, Set:Int) -> NSImage
2006,2019c2322,2324
<        // set the context to draw images into //
<        let Context = CoreGraphics().CreateContext(Size * 2, Height:Size * 2)
<
<        // scale the context coordinates to draw the images into //
<        Context.scaleBy(x:Size / 660, y:Size / 660)
<
<        // if need to update the board or the recomposite background image is nil //
<        if UpdateBoard || (RecompositeBackground == nil)
<        {
<           // draw the chess board into the context //
<           Context.draw(GetBoardImage(), in:Dims.Board.BoundsRet, byTiling:false)
<
<           // set the current context to draw into //
<           NSGraphicsContext.current = NSGraphicsContext(cgContext:Context, flipped:false)
---
>        // return the nsimage for the given board and pieces set //
>        return CoreGraphics().CGImageToNSImage(CGBoardAndPieces[Set], Size:CGFloat(Size))
>     }
2021,2022c2326,2327
<           // set chess board border color //
<           GetBorderTexture().set()
---
>     // get the context for the given board set //
>     func GetContext(_ Set:Int) -> CGContext {return CGContexts[Set]}
2024,2050c2329,2330
<           // draw the chess board border //
<           Dims.BorderBounds.stroke()
<
<           // loop over all the board labels //
<           for Index in 0...7
<           {
<              // set the file label for the board //
<              BoardLabel = BoardLabelsFont.GetString(SquareLabel.FileString(Index, Rotated:Inf
<                                                    .Vals("RotateBoard").VB), White:false)
<
<              // draw the file label at the specified point //
<              BoardLabel.draw(at:CGPoint(x:Dims.Square.BoundsRet.midX + CGFloat(Index) * Dims.Square
<                             .WidthRet - BoardLabel.size().width * 0.5, y:max(Dims.Border.BoundsRet
<                             .midX - BoardLabel.size().height * 0.5, 0)))
<
<              // set the rank label for the board //
<              BoardLabel = BoardLabelsFont.GetString(SquareLabel.RankString(Index, Rotated:Inf
<                                                    .Vals("RotateBoard").VB), White:false)
<
<              // draw the rank label at the specified point //
<              BoardLabel.draw(at:CGPoint(x:max(Dims.Border.BoundsRet.midX - BoardLabel.size().width *
<                             0.5, 0), y:Dims.Square.BoundsRet.midX + CGFloat(Index) * Dims.Square
<                             .WidthRet - BoardLabel.size().height * 0.5))
<           }
<
<           // set the board roughness texture fill color //
<           GetBoardTexture().setFill()
---
>     // get the cgimage for the given board and pieces set //
>     func GetCGImage(_ Set:Int) -> CGImage {return CGBoardAndPieces[Set]}
2052,2053c2332,2337
<           // fill the context with the board roughness texture color //
<           Dims.InnerBounds.BoundsRet.fill()
---
>     // get the nsimage for the given board and pieces set //
>     func GetNSImage(_ Set:Int) -> NSImage
>     {
>        // return the nsimage for the given board and pieces set //
>        return CoreGraphics().CGImageToNSImage(CGBoardAndPieces[Set], Size:Dims.ViewBoardSize(Set))
>     }
2055,2066c2339,2347
<           // loop over all the square border paths //
<           for Index in 0..<SquareBorders.Paths.count
<           {
<              // set the square border color //
<              SquareBorders.Colors[Index].set()
<
<              // stroke the square border path with the given color //
<              SquareBorders.Paths[Index].stroke()
<           }
<
<           // save the recomposite background image //
<           RecompositeBackground = Context.makeImage()
---
>     // recomposite the board and the pieces //
>     func Recomposite(_ Set:Int, BPData:ChessBoardData, InPositionEditor:Bool, FENEnpassantSquare:
>                     String, EnpassantSquare:APoint, RankAndFile:APoint, UpdateBoard:Bool)
>     {
>        // if need to update the board //
>        if UpdateBoard
>        {
>           // recomposite the board //
>           RecompositeBoard()
2068,2069c2349,2350
<        // don't need to update the recomposite background image //
<        else if RecompositeBackground != nil
---
>        // if not updating the board //
>        else
2071,2072c2352,2353
<           // draw the recomposite background image into the context //
<           Context.draw(RecompositeBackground, in:Dims.Board.BoundsRet, byTiling:false)
---
>           // draw the previously saved board image into the context //
>           CGContexts[Set].draw(CGImagesBoard[Set], in:Dims.GetViewBounds(Set), byTiling:false)
2074,2076c2355,2357
<
<        // if in position editor and fen enpassant square is not empty and on main sized board //
<        if InPositionEditor && (FENEnpassantSquare != "-") && (Size == 660)
---
>
>        // if in the position editor //
>        if InPositionEditor
2078,2080c2359,2366
<           // draw the en passant square in the rectangle //
<           Context.draw(Highlights.EnpassantSquareImage, in:Dims.GetRect(Inf.Vals("RotateBoard").VB,
<                        Point:EnpassantSquare), byTiling:false)
---
>           // if the fen enpassant square is not empty on the main sized board //
>           if (FENEnpassantSquare != "-") && (Set == 2)
>           {
>              // draw the en passant square in the rectangle //
>              CGContexts[Set].draw(Highlights.EnpassantSquareImage, in:Dims.GetRect(Inf
>                                   .Vals("RotateBoard").VB, Set:Set, Point:EnpassantSquare),
>                                   byTiling:false)
>           }
2083c2369
<        else if !InPositionEditor
---
>        else
2086c2372
<           if Inf.Vals("HighlightCapturedPieces").VB {for TheCapSquare in BPData.CaptureSquares
---
>           if Inf.Vals("HighlightCapturedPieces").VB
2088,2091c2374,2382
<              // display the captured highlight image on this square //
<              Context.draw(Highlights.CaptureSquareImage, in:Dims.GetRect(Inf.Vals("RotateBoard").VB,
<                           Square:TheCapSquare), byTiling:false)
<           }}
---
>              // loop over the list of squares where captures can occur //
>              for Index in 0 ..< BPData.ListOfCaptureSquares.count
>              {
>                 // display the captured highlight image on this square //
>                 CGContexts[Set].draw(Highlights.GetCaptureSquareImage(Set), in:Dims.GetRect(Inf
>                                      .Vals("RotateBoard").VB, Set:Set, Square:BPData
>                                      .ListOfCaptureSquares[Index]), byTiling:false)
>              }
>           }
2094c2385
<           if Inf.Vals("HighlightThreatenedPieces").VB {for TheThreatSquare in BPData.ThreatSquares
---
>           if Inf.Vals("HighlightThreatenedPieces").VB
2096,2099c2387,2395
<              // draw the threatened highlight in the rectangle //
<              Context.draw(Highlights.ThreatenedSquareImage, in:Dims.GetRect(Inf.Vals("RotateBoard").VB,
<                           Square:TheThreatSquare), byTiling:false)
<           }}
---
>              // loop over the list of squares where pieces can be threatened //
>              for Index in 0 ..< BPData.ListOfThreatenedSquares.count
>              {
>                 // draw the threatened highlight in the rectangle //
>                 CGContexts[Set].draw(Highlights.GetThreatenedSquareImage(Set), in:Dims.GetRect(Inf
>                                      .Vals("RotateBoard").VB, Set:Set, Square:BPData
>                                      .ListOfThreatenedSquares[Index]), byTiling:false)
>              }
>           }
2102c2398
<           if BPData.SquaresNotEqual() && (Inf.Vals("HighlightLastMoveMade").VB || (Size < 660))
---
>           if BPData.SquaresNotEqual() && (Inf.Vals("HighlightLastMoveMade").VB || (Set == 0))
2105,2106c2401,2403
<              Context.draw(Highlights.LastMoveImage, in:Dims.GetRect(Inf.Vals("RotateBoard").VB,
<                           Point:BPData.StartSquare), byTiling:false)
---
>              CGContexts[Set].draw(Highlights.GetLastMoveImage(Set), in:Dims.GetRect(Inf
>                                   .Vals("RotateBoard").VB, Set:Set, Point:BPData.StartSquare),
>                                   byTiling:false)
2109,2110c2406,2408
<              Context.draw(Highlights.LastMoveImage, in:Dims.GetRect(Inf.Vals("RotateBoard").VB,
<                           Point:BPData.EndSquare), byTiling:false)
---
>              CGContexts[Set].draw(Highlights.GetLastMoveImage(Set), in:Dims.GetRect(Inf
>                                   .Vals("RotateBoard").VB, Set:Set, Point:BPData.EndSquare),
>                                   byTiling:false)
2121,2122c2411,2412
<        // if not hiding the captured pieces in the board //
<        if !Inf.Vals("HideCapturedPiecesInBoard").VB
---
>        // if for the large sized board //
>        if Set == 2
2124,2131c2414,2415
<           // set the captured pieces background color to match that of the main window //
<           Context.setFillColor(WindowTexture.ScaledTexture.cgColor)
<
<           // if have captured pieces, fill the background color for the captured pieces //
<           if !BPData.Captures.isEmpty {Context.fill(Dims.CapturesBounds.BoundsRet)}
<
<           // loop over the list of captured pieces to display //
<           for Index in 0..<BPData.Captures.count
---
>           // if move by selecting squares and have a selected square for main sized board //
>           if SelectedSquare
2133,2135c2417,2420
<              // draw the captured piece into the context //
<              Context.draw(GetPieceCGImage(BPData.Captures[Index]), in:Dims.GetCaptureRect(Index),
<                           byTiling:false)
---
>              // draw the piece into the rectangle //
>              CGContexts[Set].draw(Highlights.SelectedSquareImage, in:Dims.GetRect(Inf
>                                   .Vals("RotateBoard").VB, Set:Set, Point:RankAndFile), byTiling:
>                                   false)
2137,2142c2422,2487
<           }
<        }
<
<        // loop over all files and ranks //
<        for File in 0...7 {for Rank in 0...7 {if BPData.RFV(File, Rank) != 0
<        {
<           // draw the chess piece image onto the board //
<           Context.draw(GetPieceCGImage(BPData.RFV(File, Rank)), in:Dims.GetRect(Inf
<                        .Vals("RotateBoard").VB, Rank:Rank, File:File), byTiling:false)
<        }}}
---
>           // if not hiding the captured pieces in the board //
>           if !Inf.Vals("HideCapturedPiecesInBoard").VB
>           {
>              // set the captured pieces background color to match that of the main window //
>              CGContexts[Set].setFillColor(WindowTexture.ScaledTexture.cgColor)
>
>              // compute the total number of captured pieces //
>              CapturedPieceCount = BPData.CapturedWhitePieces.reduce(0, +) + BPData
>                                   .CapturedBlackPieces.reduce(0, +)
>
>              // if have captured pieces, fill the background color for the captured pieces //
>              if CapturedPieceCount > 0 {CGContexts[Set].fill(Dims.GetCapturesRectBackground())}
>
>              // reset the captured pieces count back to zero //
>              CapturedPieceCount = 0
>
>              // loop over the list of captured white pieces to display //
>              for Index in 0 ..< BPData.CapturedWhitePieces.count
>              {
>                 // loop over the number of captured pieces for this particular piece type //
>                 for _ in 0 ..< BPData.CapturedWhitePieces[Index]
>                 {
>                    // draw the captured piece in the view //
>                    CGContexts[Set].draw(GetPieceCGImage(5 - Index, Set:0), in:Dims
>                                         .GetCaptureRect(CapturedPieceCount), byTiling:false)
>
>                    // increment the captured piece count //
>                    CapturedPieceCount += 1
>                 }
>              }
>
>              // loop over the list of captured black pieces to display //
>              for Index in 0 ..< BPData.CapturedBlackPieces.count
>              {
>                 // loop over the number of captured pieces for this particular piece type //
>                 for _ in 0 ..< BPData.CapturedBlackPieces[Index]
>                 {
>                    // draw the captured piece in the view //
>                    CGContexts[Set].draw(GetPieceCGImage(Index - 5, Set:0), in:Dims
>                                         .GetCaptureRect(CapturedPieceCount), byTiling:false)
>
>                    // increment the captured piece count //
>                    CapturedPieceCount += 1
>                 }
>              }
>           }
>        }
>
>        // loop over all files //
>        for File in 0 ..< 8
>        {
>           // loop over all ranks //
>           for Rank in 0 ..< 8
>           {
>              // if there is a piece on this square //
>              if BPData.RFV(File, Rank) != 0
>              {
>                 // draw the chess piece image into the board //
>                 CGContexts[Set].draw(GetPieceCGImage(BPData.RFV(File, Rank), Set:Set), in:Dims
>                                      .GetRect(Inf.Vals("RotateBoard").VB, Set:Set, Rank:Rank, File:
>                                      File), byTiling:false)
>              }
>           }
>        }
2148,2149c2489,2559
<        // return the board and pieces cgimage //
<        return Context.makeImage()!
---
>        // update the board and pieces image //
>        CGBoardAndPieces[Set] = CGContexts[Set].makeImage()!
>     }
>
>     // recomposite the board //
>     func RecompositeBoard()
>     {
>        // set the current context to the main board sized context //
>        NSGraphicsContext.current = NSGraphicsContext(cgContext:CGContexts[2], flipped:false)
>
>        // draw the current chess board image to the current context //
>        NSGraphicsContext.current?.cgContext.draw(GetBoardImage(), in:Dims.GetViewBounds(2), byTiling:
>                                                  false)
>
>        // set chess board border color //
>        GetBorderTexture().set()
>
>        // draw the chess board border //
>        Dims.BorderRect.stroke()
>
>        // loop over all the board labels //
>        for Index in 0 ..< 8
>        {
>           // set the file label for the board //
>           BoardLabel = BoardLabelsFont.GetString(SquareLabel.FileString(Index, Rotated:Inf
>                                                 .Vals("RotateBoard").VB), White:false)
>
>           // draw the file label at the specified point //
>           BoardLabel.draw(at:CGPoint(x:Dims.SquareCenter + CGFloat(Index) * Dims.GetSquareSize(2) -
>                           BoardLabel.size().width * 0.5, y:max(Dims.HalfBorderWidth - BoardLabel
>                           .size().height * 0.5, 0)))
>
>           // set the rank label for the board //
>           BoardLabel = BoardLabelsFont.GetString(SquareLabel.RankString(Index, Rotated:Inf
>                                                 .Vals("RotateBoard").VB), White:false)
>
>           // draw the rank label at the specified point //
>           BoardLabel.draw(at:CGPoint(x:max(Dims.HalfBorderWidth - BoardLabel.size().width * 0.5, 0),
>                           y:Dims.SquareCenter + CGFloat(Index) * Dims.GetSquareSize(2) - BoardLabel
>                           .size().height * 0.5))
>        }
>
>        // set the board roughness texture fill color //
>        GetBoardTexture().setFill()
>
>        // fill the context with the board roughness texture color //
>        Dims.InnerBounds.BoundsRect.fill()
>
>        // loop over all the square border paths //
>        for Index in 0 ..< SquareBorders.GetSquareBorderPathsCount()
>        {
>           // set the square border color //
>           SquareBorders.GetSquareBorderColor(Index).set()
>
>           // stroke the square border path with the given color //
>           SquareBorders.GetSquareBorderPath(Index).stroke()
>        }
>
>        // convert the recomposited board to a cgimage //
>        CGImagesBoard[2] = CGContexts[2].makeImage()!
>
>        // loop over the smaller board sizes //
>        for Set in 0 ..< Dims.GetBoardSizeCount() - 1
>        {
>           // draw the large image into the smaller board //
>           CGContexts[Set].draw(CGImagesBoard[Dims.GetBoardSizeCount() - 1], in:Dims
>                                .GetViewBounds(Set), byTiling:false)
>
>           // save the current sized board image //
>           CGImagesBoard[Set] = CGContexts[Set].makeImage()!
>        }
2156,2170c2568,2569
<        // save the window texture and update board texture and board rendering values //
<        (self.WindowTexture, Ctr.BoardRoughnessSlider.intValue, Ctr.BoardTintColorSlider.intValue,
<        Ctr.BoardTintVisibilitySlider.intValue, Ctr.BorderTextureSlider.intValue, Ctr
<        .BorderVisibilitySlider.intValue, Ctr.SquareBorderOpacitySlider.intValue, Ctr
<        .HighlightLastMove.state, Ctr.HighlightCaptures.state, Ctr.HighlightThreats.state, Ctr
<        .RotateBoardButton.state, Ctr.DisplayCaptures.state, Ctr.MoveBySelectingSquares.state) =
<        (WindowTexture, Int32(Inf.Vals("BoardRoughness").VI), Int32(Inf.Vals("BoardTintHue").VI),
<        Int32(Inf.Vals("BoardTintVisibility").VI), Int32(Inf.Vals("BorderTextureColor").VI), Int32(Inf
<        .Vals("BorderVisibility").VI), Int32(Inf.Vals("SquareBorder").VI), Ctr.UpdatedState(Inf
<        .Vals("HighlightLastMoveMade").VB), Ctr.UpdatedState(Inf.Vals("HighlightCapturedPieces").VB),
<        Ctr.UpdatedState(Inf.Vals("HighlightThreatenedPieces").VB), Ctr.UpdatedState(Inf
<        .Vals("RotateBoard").VB), Ctr.UpdatedState(!Inf.Vals("HideCapturedPiecesInBoard").VB), Ctr
<        .UpdatedState(Inf.Vals("MoveBySelectingSquares").VB))
---
>        // save the window texture //
>        self.WindowTexture = WindowTexture
2186c2585,2588
<
---
>
>
>        // update the board roughness slider //
>        Ctr.BoardRoughnessSlider.intValue = Int32(Inf.Vals("BoardRoughness").VI)
2189a2592,2594
>
>        // update the board tint color slider //
>        Ctr.BoardTintColorSlider.intValue = Int32(Inf.Vals("BoardTintHue").VI)
2192a2598,2600
>
>        // update the board tint visibility slider //
>        Ctr.BoardTintVisibilitySlider.intValue = Int32(Inf.Vals("BoardTintVisibility").VI)
2195c2603,2606
<
---
>
>
>        // update the chess board border texture slider //
>        Ctr.BorderTextureSlider.intValue = Int32(Inf.Vals("BorderTextureColor").VI)
2198a2610,2612
>
>        // update the chess board border visibility slider //
>        Ctr.BorderVisibilitySlider.intValue = Int32(Inf.Vals("BorderVisibility").VI)
2201c2615,2618
<
---
>
>
>        // update the square border opacity slider //
>        Ctr.SquareBorderOpacitySlider.intValue = Int32(Inf.Vals("SquareBorder").VI)
2204a2622,2643
>
>        // update the highlight last move made button //
>        Ctr.HighlightLastMoveMadeButton.state = Ctr.UpdatedState(Inf.Vals("HighlightLastMoveMade").VB)
>
>        // update the highlight captured pieces button //
>        Ctr.HighlightCapturedPiecesButton.state = Ctr.UpdatedState(Inf
>                                                 .Vals("HighlightCapturedPieces").VB)
>
>        // update the highlight threatened pieces button //
>        Ctr.HighlightThreatenedPiecesButton.state = Ctr.UpdatedState(Inf
>                                                   .Vals("HighlightThreatenedPieces").VB)
>
>        // update the rotate board button state //
>        Ctr.RotateBoardButton.state = Ctr.UpdatedState(Inf.Vals("RotateBoard").VB)
>
>        // update the display captured pieces button state //
>        Ctr.DisplayCapturedPiecesButton.state = Ctr.UpdatedState(!Inf
>                                               .Vals("HideCapturedPiecesInBoard").VB)
>
>        // update the move by selecting squares button //
>        Ctr.MoveBySelectingSquaresButton.state = Ctr.UpdatedState(Inf.Vals("MoveBySelectingSquares")
>                                                .VB)
2212,2213c2651,2652
<     // toggle the key //
<     func ToggleKey(_ Key:String)
---
>     // rotate the board view //
>     @IBAction func RotateBoardAction(_:AnyObject)
2215c2654
<        // play the sound of a button click //
---
>        // play the game sound of a click //
2216a2656,2658
>
>        // toggle the rotate board status //
>        Inf.ToggleVB("VBPref", Key:"RotateBoard")
2218,2222c2660,2661
<        // toggle the key //
<        Inf.ToggleVB("VBPref", Key:Key)
<
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2225,2227d2663
<     // rotate the board view //
<     @IBAction func RotateBoardAction(_:AnyObject) {ToggleKey("RotateBoard")}
<
2229c2665,2675
<     @IBAction func HideCapturedPiecesAction(_:AnyObject) {ToggleKey("HideCapturedPiecesInBoard")}
---
>     @IBAction func HideCapturedPiecesInBoardAction(_:AnyObject)
>     {
>        // play the sound of a button click //
>        GameSounds.Play("click")
>
>        // toggle the hide captured pieces in board status //
>        Inf.ToggleVB("VBPref", Key:"HideCapturedPiecesInBoard")
>
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
>     }
2232c2678,2685
<     @IBAction func MoveBySelectingSquaresAction(_:AnyObject) {ToggleKey("MoveBySelectingSquares")}
---
>     @IBAction func MoveBySelectingSquaresAction(_:AnyObject)
>     {
>        // switch the move by selecting squares mode //
>        Inf.ToggleVB("VBPref", Key:"MoveBySelectingSquares")
>
>        // play the sound of a button click //
>        GameSounds.Play("click")
>     }
2241c2694,2704
<     @IBAction func HighlightCapturedPiecesAction(_:AnyObject) {ToggleKey("HighlightCapturedPieces")}
---
>     @IBAction func HighlightCapturedPiecesButtonAction(_:AnyObject)
>     {
>        // play the sound of a button click //
>        GameSounds.Play("click")
>
>        // toggle the highlight captured pieces status //
>        Inf.ToggleVB("VBPref", Key:"HighlightCapturedPieces")
>
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
>     }
2244c2707,2717
<     @IBAction func HighlightLastMoveMadeAction(_:AnyObject) {ToggleKey("HighlightLastMoveMade")}
---
>     @IBAction func HighlightLastMoveMadeButtonAction(_:AnyObject)
>     {
>        // play the sound of a button click //
>        GameSounds.Play("click")
>
>        // toggle the highlight last move made status //
>        Inf.ToggleVB("VBPref", Key:"HighlightLastMoveMade")
>
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
>     }
2247c2720,2730
<     @IBAction func HighlightThreatPiecesAction(_:AnyObject) {ToggleKey("HighlightThreatenedPieces")}
---
>     @IBAction func HighlightThreatenedPiecesButtonAction(_:AnyObject)
>     {
>        // play the sound of a button click //
>        GameSounds.Play("click")
>
>        // toggle the highlight threatened pieces status //
>        Inf.ToggleVB("VBPref", Key:"HighlightThreatenedPieces")
>
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
>     }
2268,2269c2751,2752
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2292,2294c2775,2779
<        // update the board tint opacity and opacity view//
<        (TintColorOpacity, Ctr.BoardTintVisibilityView.stringValue) = (Inf.Fraction(400, Key:
<        "BoardTintVisibility"), "Tint " + Inf.Perc("BoardTintVisibility"))
---
>        // update the board tint opacity //
>        TintColorOpacity = Inf.Fraction(400, Key:"BoardTintVisibility")
>
>        // update the board tint opacity view //
>        Ctr.BoardTintVisibilityView.stringValue = "Tint " + Inf.Perc("BoardTintVisibility")
2310,2312c2795,2799
<        // update the chess board border opacity and border visibility //
<        (BorderOpacity, Ctr.BorderVisibilityView.stringValue) = (Inf.Fraction(100, Key:
<        "BorderVisibility"), "Border " + Inf.Perc("BorderVisibility"))
---
>        // update the chess board border opacity //
>        BorderOpacity = Inf.Fraction(100, Key:"BorderVisibility")
>
>        // display the chess board border visibility //
>        Ctr.BorderVisibilityView.stringValue = "Border " + Inf.Perc("BorderVisibility")
2343c2830,2834
<     func GetPieceCGImage(_ Piece:Int) -> CGImage {return PieceImages.GetPieceCGImage(Piece)}
---
>     func GetPieceCGImage(_ Piece:Int, Set:Int) -> CGImage
>     {
>        // return the piece cgimage //
>        return PieceImages.GetPieceCGImage(Piece, Set:Set)
>     }
2357,2358c2848,2849
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2370,2371c2861,2862
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2383,2384c2874,2875
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2397,2398c2888,2889
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2410,2411c2901,2902
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2420,2421c2911,2912
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:true, Display:true)
2433,2434c2924,2925
<        // update the preference previews //
<        Board.UpdatePreviews(true, Diagram:false)
---
>        // refresh the board and pieces //
>        Board.RefreshBoardAndPieces(2, UpdateBoard:false, Display:true)
2452a2944,2957
>
>        // loop over all the board size sets //
>        for Set in 0 ..< Dims.GetBoardSizeCount()
>        {
>           // append a context //
>           CGContexts.append(CoreGraphics().CreateContext(Dims.GetBoardSize(Set), Height:Dims
>                             .GetBoardSize(Set)))
>
>           // append a board and pieces cgimage //
>           CGBoardAndPieces.append(CGContexts.last!.makeImage()!)
>
>           // append a board cgimage //
>           CGImagesBoard.append(CGContexts.last!.makeImage()!)
>        }
2501c3006
<        for Object in Items {switch Object
---
>        for Object in Items
2503,2504c3008,3012
<           // update is enabled in the case where the toolbar object is a toolbar item //
<           case let ToolbarObject as NSToolbarItem : ToolbarObject.isEnabled = Enabled
---
>           // switch on the current object //
>           switch Object
>           {
>              // update is enabled in the case where the toolbar object is a toolbar item //
>              case let ToolbarObject as NSToolbarItem : ToolbarObject.isEnabled = Enabled
2506,2507c3014,3015
<           // update is enabled in the case where the toolbar object is a slider //
<           case let ToolbarObject as NSSlider : ToolbarObject.isEnabled = Enabled
---
>              // update is enabled in the case where the toolbar object is a slider //
>              case let ToolbarObject as NSSlider : ToolbarObject.isEnabled = Enabled
2509,2511c3017,3020
<           // the default case //
<           default : break
<        }}
---
>              // the default case //
>              default : break
>           }
>        }
2513,2514c3022,3023
<        // set the play button image //
<        PlayButton.image = Enabled ? PlayIcon : PauseIcon
---
>        // if the buttons are to be enabled //
>        if Enabled {PlayButton.image = PlayIcon} else {PlayButton.image = PauseIcon}
2518c3027
<     func Initialize(_ Items:[AnyObject], PlayButton:NSToolbarItem)
---
>     func Initialize(_ ToolbarItems:[AnyObject], ThePlayButton:NSToolbarItem)
2520,2522c3029,3034
<        // save the toolbar items and the play button //
<        (self.Items, self.PlayButton) = (Items, PlayButton)
---
>        // save the toolbar items //
>        Items = ToolbarItems
>
>        // save the play button //
>        PlayButton = ThePlayButton
2530a3043,3045
>     // the preferences vertical adjustment //
>     var PrefV:CGFloat = 0
>
2544c3059
<        for Object in Items {switch Object
---
>        for Object in Items
2546,2547c3061,3065
<           // update is hidden in the case where the object is a popup button //
<           case let PrefObject as NSPopUpButton : PrefObject.isHidden = Hidden
---
>           // switch on the current object //
>           switch Object
>           {
>              // update is hidden in the case where the object is a popup button //
>              case let PrefObject as NSPopUpButton : PrefObject.isHidden = Hidden
2549,2550c3067,3068
<           // update is hidden in the case where the object is an image view //
<           case let PrefObject as NSImageView : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is an image view //
>              case let PrefObject as NSImageView : PrefObject.isHidden = Hidden
2552,2553c3070,3071
<           // update is hidden in the case where the object is a slider //
<           case let PrefObject as NSSlider : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is a slider //
>              case let PrefObject as NSSlider : PrefObject.isHidden = Hidden
2555,2556c3073,3074
<           // update is hidden in the case where the object is a text field //
<           case let PrefObject as NSTextField : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is a text field //
>              case let PrefObject as NSTextField : PrefObject.isHidden = Hidden
2558,2559c3076,3077
<           // update is hidden in the case where the object is a button //
<           case let PrefObject as NSButton : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is a button //
>              case let PrefObject as NSButton : PrefObject.isHidden = Hidden
2561,2562c3079,3080
<           // update is hidden in the case where the object is a colors view //
<           case let PrefObject as ColorsView : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is a colors view //
>              case let PrefObject as ColorsView : PrefObject.isHidden = Hidden
2564,2565c3082,3083
<           // update is hidden in the case where the object is a time clock display //
<           case let PrefObject as TimeClockDisplay : PrefObject.isHidden = Hidden
---
>              // update is hidden in the case where the object is a time clock display //
>              case let PrefObject as TimeClockDisplay : PrefObject.isHidden = Hidden
2567,2569c3085,3088
<           // the default case //
<           default : break
<        }}
---
>              // the default case //
>              default : break
>           }
>        }
2575,2577c3094,3099
<        // save the controller items and set index to zero //
<        (Items, Index) = (ControlItems, 0)
```

```
<
---
>      // save the controller items //
>      Items = ControlItems
>
>      // initialize index to zero //
>      Index = 0
2582c3104
<          if Index < Frames.count {if Frames[Index].count == 4
---
>          if Index < Frames.count
2584,2587c3106,3113
<              // set the frame from the extracted frame components //
<              Frame = CGRect(x:Frames[Index][0], y:Frames[Index][1], width:Frames[Index][2],height:
                              Frames[Index][3])
<          }}
---
>              // if the selected frames has enough components to make a frame from //
>              if Frames[Index].count == 4
>              {
>                  // set the frame from the extracted frame components //
>                  Frame = CGRect(x:Frames[Index][0], y:Frames[Index][1] + PrefV, width:
                              Frames[Index][2], height:Frames[Index][3])
>              }
>          }
2631c3157,3160
<
---
>
>      // image data for the saved image //
>      var ImageData:Data = Data()
>
2640a3170,3177
>      // save an image for the specified key //
>      func Save(_ Image:NSImage, Key:String)
>      {
>          // save the image data to preferences //
>          Prefs.set(try? NSKeyedArchiver.archivedData(withRootObject:Image, requiringSecureCoding:
                   true), forKey:Key)
>      }
>
2642c3179
<      func PrefInt(_ Key:String, Default:Int, Min:Int, Max:Int) -> Int
---
>      func PreferenceInt(_ Key:String, Default:Int, Min:Int, Max:Int) -> Int
2651,2653c3188,3192
<          // set that the user preference exists now //
<          Prefs.set(true, forKey:Key + "Exists")
---
>          // the user preference doesn't exist yet //
>          else
>          {
>              // set that the user preference exists now //
>              Prefs.set(true, forKey:Key + "Exists")
2655,2656c3194,3195
<          // set the default value for the specified key //
<          Prefs.set(Default, forKey:Key)
---
>              // set the default value for the specified key //
>              Prefs.set(Default, forKey:Key)
2658,2659c3197,3199
<          // return the default value for the key //
<          return Default
---
>
>              // return the default value for the key //
>              return Default
>          }
2663c3203
<      func PrefBool(_ Key:String, Default:Bool) -> Bool
---
>      func PreferenceBool(_ Key:String, Default:Bool) -> Bool
2665,2669c3205,3215
<          // return the saved user preference if exists //
<          if Prefs.bool(forKey:Key + "Exists") {return Prefs.bool(forKey:Key)}
<
<          // set that the user preference exists now //
<          Prefs.set(true, forKey:Key + "Exists")
---
>          // if the user preference exists //
>          if Prefs.bool(forKey:Key + "Exists")
>          {
>              // return the saved user preference //
>              return Prefs.bool(forKey:Key)
>          }
>          // the user preference does not yet exist //
>          else
>          {
>              // set that the user preference exists now //
>              Prefs.set(true, forKey:Key + "Exists")
2671,2672c3217,3218
<          // set the default value for the specified key //
<          Prefs.set(Default, forKey:Key)
---
>              // set the default value for the specified key //
>              Prefs.set(Default, forKey:Key)
2674,2675c3220,3222
<          // return the default value for this key //
<          return Default
---
>              // return the default value for this key //
>              return Default
>          }
2679c3226
<      func PrefString(_ Key:String, Default:String) -> String
---
>      func PreferenceString(_ Key:String, Default:String) -> String
2681,2685c3228,3238
<          // return the saved user preference if exists //
<          if Prefs.bool(forKey:Key + "Exists") {return Prefs.string(forKey:Key)!}
<
<          // set that the user preference now exists for this key //
<          Prefs.set(true, forKey:Key + "Exists")
---
>          // if the user saved preference exists //
>          if Prefs.bool(forKey:Key + "Exists")
>          {
>              // return the user saved preference for this key //
>              return Prefs.string(forKey:Key)!
>          }
>          // the user saved preference does not yet exist //
>          else
>          {
>              // set that the user preference now exists for this key //
>              Prefs.set(true, forKey:Key + "Exists")
2687,2688c3240,3241
<          // set the default value for this key //
<          Prefs.set(Default, forKey:Key)
---
>              // set the default value for this key //
>              Prefs.set(Default, forKey:Key)
2690,2691c3243,3283
<          // return the default value //
<          return Default
---
>              // return the default value for this key //
>              return Default
>          }
>      }
>
>      // get the user preference for the specified key //
>      func PreferenceImage(_ Key:String, Default:NSImage) -> NSImage
>      {
>          // if the value for the key already exists //
>          if Prefs.bool(forKey:Key + "Exists")
>          {
>              // if the data to unarchive exists //
>              if let ImageToUnarchive = (try? (NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(Prefs
                              .object(forKey:Key) as! Data) as? NSImage)
>              {
>                  // return the image //
>                  return ImageToUnarchive
>              }
>              // the data to unarchive doesn't exist //
>              else
>              {
>                  // return the default image value //
>                  return Default
>              }
>          }
>          // value for the key doesn't exist yet //
>          else
>          {
>              // set that the value for the key currently exists now //
>              Prefs.set(true, forKey:Key + "Exists")
>
>              // get the image data for the default image //
>              ImageData = try! NSKeyedArchiver.archivedData(withRootObject:Default,
                                    requiringSecureCoding:true) as Data
>
>              // save the user preference for the image key //
>              Prefs.set(ImageData, forKey:Key)
>
>              // return the default image //
>              return Default
>          }
2721a3314,3316
>      // a loop index //
>      var Index:Int = 0
>
2756,2758c3351,3355
<          // set the prefix string, color view, and font values //
<          (Prefix, ColorView, FontTypeButton, SizesIndexButton, BoldButton, Scale) = (Tag, View,
            TheFont, Sizes, Bold, TheScale)
---
>          // set the prefix string //
>          Prefix = Tag
>
>          // set the color view //
>          ColorView = View
2769a3367,3378
>
>          // set the font type button //
>          FontTypeButton = TheFont
>
>          // set the size index button //
>          SizesIndexButton = Sizes
>
>          // set the bold button //
>          BoldButton = Bold
>
>          // set the scale //
>          Scale = TheScale
2774c3383
<          for TheFont in Fonts {FontTypeButton.addItem(withTitle:TheFont)}
---
>          for Index in 0 ..< Fonts.count {FontTypeButton.addItem(withTitle:Fonts[Index])}
2797a3407,3409
>
>          // whether to use a paragraph vertical offset or not //
>          UseParagraphOffset = UseVertOffset
2806,2811c3418,3424
<          // set the font size and font bold string and other variables //
<          (Size, FontBoldString, UseParagraphOffset, ParagraphStyle.paragraphSpacing) = (Scale * (16 +
              3 * Scale * Inf.VP(Prefix + "Sizes")), Inf.Vals(Prefix + "Bold").VB ? " Bold" : "",
              UseVertOffset, CGFloat(max(0, VertOffset - 10)))
<
<          // set the current font //
---
>          // set the font size //
>          Size = Scale * (16 + 3 * Scale * Inf.VP(Prefix + "Sizes"))
>
>          // set the font bold string //
>          FontBoldString = Inf.Vals(Prefix + "Bold").VB ? " Bold" : ""
>
>          // if the font exists //
2816,2823c3429,3444
<          // set the font attributes with and without vertical offset //
<          (VerticallyOffsetAttributes, Attributes, WhiteFontAttributes) = ([NSAttributedString.Key
              .font:CurrentFont, NSAttributedString.Key.foregroundColor:ColorView.GetCurrentHue(),
              NSAttributedString.Key.paragraphStyle:ParagraphStyle], [NSAttributedString.Key.font:
              CurrentFont, NSAttributedString.Key.foregroundColor:ColorView.GetCurrentHue()],
              [NSAttributedString.Key.font:CurrentFont, NSAttributedString.Key.foregroundColor:NSColor
              .white])
---
>          // set the paragraph style paragraph spacing //
>          ParagraphStyle.paragraphSpacing = CGFloat(max(0, VertOffset - 10))
>
>          // set the vertically offset font attributes //
>          VerticallyOffsetAttributes = [NSAttributedString.Key.font:CurrentFont, NSAttributedString.Key
                          .foregroundColor:ColorView.GetCurrentHue(), NSAttributedString
                          .Key.paragraphStyle:ParagraphStyle]
>
>          // set the font attributes without the vertical offset in the paragraph style //
>          Attributes = [NSAttributedString.Key.font:CurrentFont, NSAttributedString.Key
              .foregroundColor:ColorView.GetCurrentHue()]
>
>          // get the white font attributes for the case of a selected table row //
>          WhiteFontAttributes = [NSAttributedString.Key.font:CurrentFont, NSAttributedString.Key
                          .foregroundColor:NSColor.white]
2829,2832c3450,3470
<          // return the attributed string based on type of string needed //
<          return NSAttributedString(string:Text, attributes:White ? WhiteFontAttributes :
              (UseParagraphOffset ? VerticallyOffsetAttributes : Attributes)) as? [NSAttributedString
              .Key:Any])
---
>          // if the text color is to be white as in a selected row //
>          if White
>          {
>              // return the attributed string with white color for a selected row in a table //
>              return NSAttributedString(string:Text, attributes:WhiteFontAttributes as?
                                    [NSAttributedString.Key:Any])
>          }
>          // if the text is to be vertically offset //
>          else if UseParagraphOffset
>          {
>              // return the attributed string for vertically offset text //
>              return NSAttributedString(string:Text, attributes:VerticallyOffsetAttributes as?
                          [NSAttributedString.Key:Any])
>          }
>          // if the text won't be vertically offset //
>          else
>          {
>              // return the attributed string with no vertical offset //
>              return NSAttributedString(string:Text, attributes:Attributes as? [NSAttributedString.Key:
                              Any])
>          }
2954a3593,3595
>      // a loop count //
>      var Count:Int = 0
>
2967,2969d3607
<          // update layer opacities //
<          for TheLayer in Layers {TheLayer.opacity = AnimateProgress ? 1 : 0}
<
2973,2975c3611,3616
<          // update the progress circle index //
<          Index = ((Index + 1) >= Circles.count) ? 0 : Index + 1
<
---
>          // increment the progress circle index //
>          Index += 1
>
>          // if the index is past or equal to the progress circles count, reset the index to zero //
>          if Index >= Circles.count {Index = 0}
>
2977c3618
<          for Count in 0..<Layers.count
---
>          for Count in 0 ..< Layers.count
2986a3628,3643
>          // if the progress circle size is not zero //
>          else if Index != 0
>          {
>              // set the progress circle size to zero //
>              Index = 0
>
>              // loop over the layers //
>              for Count in 0 ..< Layers.count
>              {
>                  // remove the animation effects //
>                  CATransaction.setDisableActions(true)
>
>                  // set the layers to zero size //
>                  Layers[Count].contents = Circles[Count]
>              }
>          }
2990c3647,3651
<      func AddLayer(_ View:NSView) {for TheLayer in Layers {View.layer?.addSublayer(TheLayer)}}
---
>      func AddLayer(_ View:NSView)
>      {
>          // loop over the layers //
>          for Count in 0 ..< Layers.count {View.layer?.addSublayer(Layers[Count])}
>      }
2999c3660
<          for Index in 0...99
---
>          for Index in 0 ..< 100
3002c3663
<              Circles.append(CoreGraphics().CreateShape(Dims.Square.WidthRet, Fraction:
---
>              Circles.append(CoreGraphics().CreateShape(Dims.GetSquareSize(2), Fraction:
3012c3673
<          for TheLayer in Layers
---
>          for Count in 0 ..< Layers.count
3014,3016c3675,3681
<              // set the layer position and bounds //
<              (TheLayer.position, TheLayer.bounds) = (CGPoint(x:Dims.Board.bounds.midX, y:Dims.Board
                  .bounds.midY), CGRect(x:0, y:0, width:Dims.Square.Width, height:Dims.Square.Width))
---
>              // set the layer position //
>              Layers[Count].position = CGPoint(x:Dims.ViewBoardSize(2) * 0.5, y:Dims.ViewBoardSize(2) *
                          0.5)
>
>              // set the layer bounds //
>              Layers[Count].bounds = CGRect(x:0, y:0, width:Dims.ViewSquareSize(2), height:Dims
                          .ViewSquareSize(2))
3023a3689,3691
>      // a loop index //
>      var Index:Int = 0
>
3031c3699,3703
<      func AddLayers(_ View:NSView) {for TheLayer in Layers {View.layer?.addSublayer(TheLayer)}}
---
>      func AddLayers(_ View:NSView)
>      {
>          // loop over all the layers //
>          for Index in 0 ..< Layers.count {View.layer?.addSublayer(Layers[Index])}
>      }
3034c3706
<      func UpdateLayer(_ Image:CGImage)
---
>      func UpdateLayer(_ Image:NSImage)
3039,3040c3711,3718
<          // copy the current saved position into position layer one and set layer two opacity to 0 //
<          (Layers[0].contents, Layers[1].opacity) = (Image, 0)
---
>          // copy the current saved position into position layer one //
>          Layers[0].contents = Image
>
>          // remove the animation effects //
>          CATransaction.setDisableActions(true)
>
>          // set position layer two opacity to zero //
>          Layers[1].opacity = 0
3044c3722
<      func RemoveAnimations() {for TheLayer in Layers {TheLayer.removeAllAnimations()}}
---
>      func RemoveAnimations() {for Index in 0 ..< Layers.count {Layers[Index].removeAllAnimations()}}
3047c3725
<      func Animate(_ View:NSView, Image:CGImage)
---
>      func Animate(_ View:NSView, Image:NSImage)
3070,3072c3748,3761
<          // set the position layer values //
<          (Layers[0].contents, PositionAnimation.fromValue, PositionAnimation.toValue, Layers[1]
              .opacity) = (Image, 1, 0, 0)
---
>          // set the position layer one contents to the new saved position image //
>          Layers[0].contents = Image
>
>          // set animation opacity to start at 1 //
>          PositionAnimation.fromValue = 1
>
>          // set animation opacity to end at 0 //
>          PositionAnimation.toValue = 0
>
>          // remove the animation effects //
>          CATransaction.setDisableActions(true)
>
>          // set ending position layer two opacity to zero //
>          Layers[1].opacity = 0
3081,3083c3770,3774
<          // set the bounds of the position layer and the position //
<          (Layers[Index].bounds, Layers[Index].position) = (CGRect(x:0, y:0, width:400, height:400),
              CGPoint(x:200, y:200))
---
>          // set the size of the position layer //
>          Layers[Index].bounds = CGRect(x:0, y:0, width:400, height:400)
>
>          // set the position layer position //
>          Layers[Index].position = CGPoint(x:200, y:200)
3100a3792,3797
>
>      // index along the x axis //
>      var X:Int = 0
>
>      // index along the y axis //
>      var Y:Int = 0
3130,3133c3827,3831
<          // set the window texture from the pattern image and create window texture preview image //
<          (Texture, ScaledTexture, Preview) = (NSColor(patternImage:Pattern), NSColor(patternImage:
              NSImage(cgImage:Pattern.cgImage(forProposedRect:nil, context:nil, hints:nil)!, size:
              CGSize(width:80, height:80))), NSImage(size:CGSize(width:200, height:200)))
---
>          // set the window texture from the pattern image //
>          Texture = NSColor(patternImage:Pattern)
>
>          // create the window texture preview image //
>          Preview = NSImage(size:CGSize(width:200, height:200))
3146a3844,3848
>
>
>          // set the scaled window texture from the pattern image //
>          ScaledTexture = NSColor(patternImage:NSImage(cgImage:Pattern.cgImage(forProposedRect:nil,
                          context:nil, hints:nil)!, size:CGSize(width:80, height:80)))
>
3160,3161c3862,3863
<          // loop over the x index and the y indices //
<          for X in 0...9 {for Y in 0...9
---
>          // loop over the x index //
>          for X in 0 ..< 10
3163,3164c3865,3869
<              // set the fill color to light gray //
<              NSColor.lightGray.setFill()
---
>              // loop over the y index //
>              for Y in 0 ..< 10
>              {
>                  // set the fill color to light gray //
>                  NSColor.lightGray.setFill()
3166,3168c3871,3873
<              // make a light gray rectangle //
<              NSBezierPath(rect:CGRect(x:CGFloat(X) * 4, y:CGFloat(Y) * 4 + 2.5, width:4, height:1))
                  .fill()
---
>                  // make a light gray rectangle //
>                  NSBezierPath(rect:CGRect(x:CGFloat(X) * 4, y:CGFloat(Y) * 4 + 2.5, width:4, height:1))
                      .fill()
3170,3172c3875,3877
<              // make a light gray rectangle //
<              NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 2.5, y:CGFloat(Y) * 4, width:1, height:4))
                  .fill()
---
>                  // make a light gray rectangle //
>                  NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 2.5, y:CGFloat(Y) * 4, width:1, height:4))
                      .fill()
3174,3175c3879,3880
<              // set the fill color to gray //
<              NSColor.gray.setFill()
---
>                  // set the fill color to gray //
>                  NSColor.gray.setFill()
3177,3179c3882,3884
<              // make a gray rectangle //
<              NSBezierPath(rect:CGRect(x:CGFloat(X) * 4, y:CGFloat(Y) * 4 + 0.5, width:4, height:1))
                  .fill()
---
>                  // make a gray rectangle //
>                  NSBezierPath(rect:CGRect(x:CGFloat(X) * 4, y:CGFloat(Y) * 4 + 0.5, width:4, height:1))
                      .fill()
3181,3183c3886,3888
<              // make a gray rectangle //
<              NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 0.5, y:CGFloat(Y) * 4, width:1, height:4))
                  .fill()
---
>                  // make a gray rectangle //
>                  NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 0.5, y:CGFloat(Y) * 4, width:1, height:4))
                      .fill()
3185,3186c3890,3891
<              // set the fill color to light gray //
<              NSColor.lightGray.setFill()
---
>                  // set the fill color to light gray //
>                  NSColor.lightGray.setFill()
3188,3191c3893,3909
<              // make a light gray rectangle //
<              NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + (((X + Y) % 2 == 0) ? 0.5 : 2.5), y:CGFloat(Y)
                  * 4 + (((X + Y) % 2 == 0) ? 2.5 : 0.5), width:1, height:1)).fill()
<          }}
---
>                  // if on an even square //
>                  if ((X + Y) % 2) == 0
>                  {
>                      // make a light gray rectangle //
>                      NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 0.5, y:CGFloat(Y) * 4 + 2.5, width:1,
                              height:1)).fill()
>                  }
>                  // if it is an odd square //
>                  else
>                  {
>                      // make a light gray rectangle //
>                      NSBezierPath(rect:CGRect(x:CGFloat(X) * 4 + 2.5, y:CGFloat(Y) * 4 + 0.5, width:1,
                              height:1)).fill()
>                  }
>              }
>          }
3202a3920,3922
>
>      // a loop index //
>      var Index:Int = 0
3211,3220c3931,3961
<          // set the annotation layer values //
<          (Layers[1].bounds, Layers[1].position, Layers[0].bounds, Layers[0].position, Layers[0]
              .anchorPoint, Layers[0].transform, Layers[2].bounds, Layers[2].position, Layers[2].anchorPoint,
              Layers[2].transform = (CGRect(x:0, y:0, width:250, height:250), CGPoint(x:330, y:125),
              CGRect(x:0, y:0, width:250, height:250), CGPoint(x:204, y:125), CGPoint(x:1, y:0.5),
              CATransform3DMakeRotation(Perspective, 0.35, 0, 1, 0), CGRect(x:0, y:0, width:250,
              height:250), CGPoint(x:456, y:125), CGPoint(x:0, y:0.5),
              CATransform3DMakeRotation(Perspective, -0.35, 0, 1, 0))
---
>          // set the annotation board bounds middle slot //
>          Layers[1].bounds = CGRect(x:0, y:0, width:250, height:250)
>
>          // set the annotation board position for middle slot //
>          Layers[1].position = CGPoint(x:330, y:125)
>
>          // set the annotation board bounds for left slot //
>          Layers[0].bounds = CGRect(x:0, y:0, width:250, height:250)
>
>          // set the annotation board position for left slot //
>          Layers[0].position = CGPoint(x:204, y:125)
>
>          // set the annotation board anchor point for left slot //
>          Layers[0].anchorPoint = CGPoint(x:1, y:0.5)
>
>          // set the annotation board transform for 3d rotation and perspective for left slot //
>          Layers[0].transform = CATransform3DMakeRotation(Perspective, 0.35, 0, 1, 0)
>
>          // set the annotation board bounds for right slot //
>          Layers[2].bounds = CGRect(x:0, y:0, width:250, height:250)
>
>          // set the annotation board position for right slot //
>          Layers[2].position = CGPoint(x:456, y:125)
>
>          // set the annotation board anchor point for right slot //
>          Layers[2].anchorPoint = CGPoint(x:0, y:0.5)
>
>          // set the annotation board transform for 3d rotation and perspective for right slot //
>          Layers[2].transform = CATransform3DMakeRotation(Perspective, -0.35, 0, 1, 0)
3226,3227c3967,3968
<          // loop over all the layers from the start index and on //
<          for TheLayer in Layers[Start...]
---
>          // loop over all the layers //
>          for Index in Start ..< Layers.count
3233c3974
<              TheLayer.contents = NSImage(size:CGSize(width:250, height:250))
---
>              Layers[Index].contents = NSImage(size:CGSize(width:250, height:250))
3238c3979,3983
<      func AddLayers(_ View:NSView) {for TheLayer in Layers {View.layer?.addSublayer(TheLayer)}}
---
>      func AddLayers(_ View:NSView)
>      {
>          // loop over all the layers //
>          for Index in 0 ..< Layers.count {View.layer?.addSublayer(Layers[Index])}
>      }
3257,3258c4002,4003
<      // the capture square image //
<      var CaptureSquareImage:CGImage!
---
>      // the capture square images //
>      var CaptureSquareImages:[CGImage] = []
3260,3261c4005,4006
<      // the threaten square image //
<      var ThreatenSquareImage:CGImage!
---
>      // the threaten square images //
>      var ThreatenSquareImages:[CGImage] = []
3263,3264c4008,4009
<      // the last move image //
<      var LastMoveImage:CGImage!
---
>      // the last move images //
>      var LastMoveImages:[CGImage] = []
3269c4014
<      // the enpassant square cgimage //
---
>      // the enpassant square image //
3271,3273d4015
<
<      // the enpassant square nsimage //
<      var EnpassantSquareNSImage:NSImage!
3278,3291c4020,4062
<          // set the board highlight squares //
<          (LastMoveImage, CaptureSquareImage, ThreatenSquareImage, SelectedSquareImage,
              EnpassantSquareImage) = (CoreGraphics().CreateShape(Dims.Square.WidthRet, Fraction:0.9, Color:
              NSColor(red:1, green:1, blue:0.65, alpha:1), Square:false), CoreGraphics().CreateShape(Dims
              .Square.WidthRet, Fraction:0.95, Color:NSColor(red:1, green:0.3, blue:0.3, alpha:1), Square:
              false), CoreGraphics().CreateShape(Dims.Square.WidthRet, Fraction:0.95, Color:NSColor(red:1,
              green:0.6, blue:0, alpha:1), Square:false), CoreGraphics().CreateShape(Dims.Square.WidthRet,
              Fraction:0.97, Color:NSColor(red:0.2, green:0.7, blue:1, alpha:1), Square:true),
              CoreGraphics().CreateShape(Dims.Square.WidthRet, Fraction:0.95, Color:NSColor(red:0.45, green:
              0.45, blue:1, alpha:1), Square:false))
<
<          // set the enpassant square nsimage //
<          EnpassantSquareNSImage = CoreGraphics().CGImageToNSImage(EnpassantSquareImage, Size:Dims
                                              .Square.Width)
---
>          // loop over all the board size sets //
>          for Index in 0 ..< Dims.GetBoardSizeCount()
>          {
>              // append a last move image //
>              LastMoveImages.append(CoreGraphics().CreateShape(Dims.GetSquareSize(Index), Fraction:0.9,
                              Color:NSColor(red:1, green:1, blue:0.65, alpha:1), Square:false))
>
>              // append a capture square image //
>              CaptureSquareImages.append(CoreGraphics().CreateShape(Dims.GetSquareSize(Index), Fraction:
                          0.95, Color:NSColor(red:1, green:0.3, blue:0.3, alpha:1), Square:
                          false))
>
>              // append a threaten square image //
>              ThreatenSquareImages.append(CoreGraphics().CreateShape(Dims.GetSquareSize(Index), Fraction:
                          0.95, Color:NSColor(red:1, green:0.6, blue:0, alpha:1), Square:
                          false))
>          }
>
>          // create the selected square image //
>          SelectedSquareImage = CoreGraphics().CreateShape(Dims.GetSquareSize(2), Fraction:0.97, Color:
                                      NSColor(red:0.2, green:0.7, blue:1, alpha:1),
                                      Square:true)
>
>          // create the enpassant square image //
>          EnpassantSquareImage = CoreGraphics().CreateShape(Dims.GetSquareSize(2), Fraction:0.95, Color:
                                          NSColor(red:0.45, green:0.45, blue:1, alpha:
                                          1), Square:false)
>      }
>
>      // get the last move image for the specified board set //
>      func GetLastMoveImage(_ Set:Int) -> CGImage {return LastMoveImages[Set]}
>
>      // get the capture square image for the specified board set //
>      func GetCaptureSquareImage(_ Set:Int) -> CGImage {return CaptureSquareImages[Set]}
>
>      // get the threaten square image for the specified board set //
>      func GetThreatenSquareImage(_ Set:Int) -> CGImage {return ThreatenSquareImages[Set]}
>
>      // get the enpassant square nsimage //
>      func GetEnpassantSquareNSImage() -> NSImage
>      {
>          // return the enpassant square nsimage //
>          return CoreGraphics().CGImageToNSImage(EnpassantSquareImage, Size:Dims.ViewSquareSize(2))
3297a4069,4077
>      // a loop index //
>      var Index:Int = 0
>
>      // a rank on the board //
>      var Rank:Int = 0
>
>      // a file on the board //
>      var File:Int = 0
>
3299c4079
<      var Origin:CGPoint = CGPoint()
---
>      var SquareOrigin:CGPoint = CGPoint()
3302c4082
<      var Offset:CGFloat = 0
---
>      var SquareBorderOffset:CGFloat = 0
3308c4088
<      var Colors:[NSColor] = [NSColor.black, NSColor.white, NSColor.gray]
---
>      var SquareBorderColors:[NSColor] = [NSColor.black, NSColor.white, NSColor.gray]
3311c4091
<      var Paths:[NSBezierPath] = [NSBezierPath(), NSBezierPath(), NSBezierPath()]
---
>      var SquareBorderPaths:[NSBezierPath] = [NSBezierPath(), NSBezierPath(), NSBezierPath()]
3317c4097
<          for Index in 0..<Paths.count
---
>          for Index in 0 ..< SquareBorderPaths.count
3319,3323c4099,4106
<              // set the square border path line width and line cap style //
<              (Paths[Index].lineWidth, Paths[Index].lineCapStyle) = (2, .square)
<
<              // loop over all the files and ranks //
<              for File in 0...7 {for Rank in 0...7
---
>              // set the square border path line width //
>              SquareBorderPaths[Index].lineWidth = 2
>
>              // set the square border path line cap style //
>              SquareBorderPaths[Index].lineCapStyle = .square
>
>              // loop over all the files //
>              for File in 0 ..< 8
3325,3328c4108,4113
<                  // set the current square origin and the square border offset from the edge //
<                  (Origin, Offset) = (CGPoint(x:CGFloat(File) * Dims.Square.WidthRet + Dims.Border
                      .WidthRet, y:CGFloat(Rank) * Dims.Square.WidthRet + Dims.Border.WidthRet), CGFloat
                      (Index) * 2)
---
>                  // loop over all the ranks //
>                  for Rank in 0 ..< 8
>                  {
>                      // set the current square origin //
>                      SquareOrigin = CGPoint(x:CGFloat(File) * Dims.GetSquareSize(2) + Dims.BorderWidth,
                                  y:CGFloat(Rank) * Dims.GetSquareSize(2) + Dims.BorderWidth)
3330,3335c4115,4116
<                  // move to the lower left corner of the square border //
<                  Paths[Index].move(to:CGPoint(x:Origin.x + Offset, y:Origin.y + Offset))
<
<                  // add a line to the upper left corner of the square border //
<                  Paths[Index].line(to:CGPoint(x:Origin.x + Offset, y:Origin.y + Dims.Square.WidthRet -
                      Offset))
---
>                      // set the square border offset from the edge //
>                      SquareBorderOffset = CGFloat(Index) * 2
3337,3339c4118,4125
<                  // add a line to the upper right corner of the square border //
<                  Paths[Index].line(to:CGPoint(x:Origin.x + Dims.Square.WidthRet - Offset, y:Origin.y +
                      Dims.Square.WidthRet - Offset))
---
>                      // move to the lower left corner of the square border //
>                      SquareBorderPaths[Index].move(to:CGPoint(x:SquareOrigin.x + SquareBorderOffset, y:
                                  SquareOrigin.y + SquareBorderOffset))
>
>                      // add a line to the upper left corner of the square border //
>                      SquareBorderPaths[Index].line(to:CGPoint(x:SquareOrigin.x + SquareBorderOffset, y:
                                  SquareOrigin.y + Dims.GetSquareSize(2) -
                                  SquareBorderOffset))
3341,3343c4127,4130
<                  // add a line to the lower right corner of the square border //
<                  Paths[Index].line(to:CGPoint(x:Origin.x + Dims.Square.WidthRet - Offset, y:Origin.y +
                      Offset))
---
>
>                      // add a line to the upper right corner of the square border //
>                      SquareBorderPaths[Index].line(to:CGPoint(x:SquareOrigin.x + Dims.GetSquareSize(2) -
                                  SquareBorderOffset, y:SquareOrigin.y + Dims
                                  .GetSquareSize(2) - SquareBorderOffset))
3345,3370c4132,4143
<                  // add a line to the lower left corner of the square border //
<                  Paths[Index].line(to:CGPoint(x:Origin.x + Offset, y:Origin.y + Offset))
<              }}
<          }
<      }
<
<      // update the square border colors //
<      func UpdateSquareBorderColors(_ Opacity:CGFloat)
<      {
<          // set the opacity colors for the square border rectangles //
<          (Colors[0], Colors[1], Colors[2]) = (NSColor.black.withAlphaComponent(Opacity), NSColor.white
              .withAlphaComponent(Opacity), NSColor.gray.withAlphaComponent(Opacity))
<      }
< }
<
< // shape dimensions //
< class Dimensions
< {
<      // the width //
<      var Width:CGFloat = 0
<
<      // the retina width //
<      var WidthRet:CGFloat = 0
<
<      // the height //
<      var Height:CGFloat = 0
---
>
>                      // add a line to the lower right corner of the square border //
>                      SquareBorderPaths[Index].line(to:CGPoint(x:SquareOrigin.x + Dims.GetSquareSize(2) -
                                  SquareBorderOffset, y:SquareOrigin.y +
                                  SquareBorderOffset))
>
>                      // add a line to the lower left corner of the square border //
>                      SquareBorderPaths[Index].line(to:CGPoint(x:SquareOrigin.x + SquareBorderOffset, y:
                                  SquareOrigin.y + SquareBorderOffset))
>                  }
>              }
>          }
>      }
3372,3373c4145,4149
<      // the retina height //
<      var HeightRet:CGFloat = 0
---
>      // update the square border colors //
>      func UpdateSquareBorderColors(_ Opacity:CGFloat)
>      {
>          // set the opacity for the outer rectangle black color //
>          SquareBorderColors[0] = NSColor.black.withAlphaComponent(Opacity)
3375,3382c4151,4152
<      // the bounds //
<      var Bounds:CGRect = CGRect()
<
<      // the retina bounds //
<      var BoundsRet:CGRect = CGRect()
<
<      // the default initializer //
<      init() {}
---
>          // set the opacity for the middle rectangle white color //
>          SquareBorderColors[1] = NSColor.white.withAlphaComponent(Opacity)
3384,3390c4154,4155
<      // the initializer //
<      init (_ XDim:CGFloat, YDim:CGFloat, X:CGFloat, Y:CGFloat)
<      {
<          // set the dimension values //
<          (Width, WidthRet, Height, HeightRet, Bounds, BoundsRet) = (XDim, XDim * 2, YDim, YDim * 2,
              CGRect(x:X, y:Y, width:XDim, height:YDim), CGRect(x:X * 2, y:Y * 2, width:XDim * 2, height:
              YDim * 2))
---
>          // set the opacity for the inner rectangle gray color //
>          SquareBorderColors[2] = NSColor.gray.withAlphaComponent(Opacity)
3391a4157,4165
>
>      // get the square border paths count //
>      func GetSquareBorderPathsCount() -> Int {return SquareBorderPaths.count}
>
>      // get the square border path for the specified index //
>      func GetSquareBorderPath(_ Index:Int) -> NSBezierPath {return SquareBorderPaths[Index]}
>
>      // get the square border color for the specified index //
>      func GetSquareBorderColor(_ Index:Int) -> NSColor {return SquareBorderColors[Index]}
3397,3398c4171,4172
<      // the board dimensions //
<      var Board:Dimensions = Dimensions(660, YDim:660, X:0, Y:0)
---
>      // a loop index //
>      var Index:Int = 0
3400,3401c4174,4175
<      // the border dimensions //
<      var Border:Dimensions = Dimensions(30, YDim:30, X:0, Y:0)
---
>      // a rank on the board //
>      var Rank:Int = 0
3403,3404c4177,4178
<      // the square dimensions //
<      var Square:Dimensions = Dimensions(75, YDim:75, X:30, Y:30)
---
>      // a file on the board //
>      var File:Int = 0
3406,3410c4180,4205
<      // the inner board bounds dimensions //
<      var InnerBounds:Dimensions = Dimensions(600, YDim:600, X:30, Y:30)
<
<      // the capture bounds dimensions //
<      var CapturesBounds:Dimensions = Dimensions(660, YDim:29, X:0, Y:631)
---
>      // the possible board sizes //
>      var BoardSizes:[CGFloat] = [440, 800, 1320]
>
>      // the possible square sizes for the different board sizes //
>      var SquareSizes:[CGFloat] = []
>
>      // the border width for the largest board //
>      var BorderWidth:CGFloat = 0
>
>      // the half border width for the largest board //
>      var HalfBorderWidth:CGFloat = 0
>
>      // the naview bounds //
>      var ViewBounds:[CGRect] = []
>
>      // the main board bounds //
>      var MainBoardBounds:CGRect = CGRect()
>
>      // the inner board bounds //
>      var InnerBoardBounds:CGRect = CGRect()
>
>      // the distance to a board square center //
>      var SquareCenter:CGFloat = 0
>
>      // the view distance to a board square center //
>      var ViewSquareCenter:CGFloat = 0
3412,3414d4206
<      // the border bounds dimensions //
<      var BorderBounds:NSBezierPath = NSBezierPath(rect:CGRect(x:30, y:30, width:1260, height:1260))
<
3416c4208
<      var Squares:[CGRect] = []
---
>      var SquareRects:[CGRect] = []
3419,3420c4211,4221
<      var Captures:[CGRect] = []
<
---
>      var CapturesRects:[CGRect] = []
>
>      // the board border bezier path //
>      var BorderRectangle:NSBezierPath = NSBezierPath()
>
>      // the capture pieces rectangle background //
>      var CapturesRectBackground:CGRect = CGRect()
>
>      // a point //
>      var Point:APoint = APoint()
>
3424,3425c4225,4226
<          // loop over all the ranks and files //
<          for Rank in 0...7 {for File in 0...7
---
>          // loop over all the board size sets //
>          for Index in 0 ..< BoardSizes.count
3427,3431c4228,4249
<              // append a square rectangle //
<              Squares.append(CGRect(x:CGFloat(File) * Square.WidthRet + Border.WidthRet, y:CGFloat(Rank)
                  * Square.WidthRet + Border.WidthRet, width:Square.WidthRet, height:Square
                  .WidthRet))
<          }}
---
>              // set the border width //
>              BorderWidth = 60 * BoardSizes[Index] / BoardSizes[2]
>
>              // append the square size //
>              SquareSizes.append((BoardSizes[Index] - 2 * BorderWidth) / 8)
>
>              // append the chess board view bounds //
>              ViewBounds.append(CGRect(x:0, y:0, width:BoardSizes[Index], height:BoardSizes[Index]))
>
>              // loop over all the ranks //
>              for Rank in 0 ..< 8
>              {
>                  // loop over all the files //
>                  for File in 0 ..< 8
>                  {
>                      // append a square rectangle //
>                      SquareRects.append(CGRect(x:CGFloat(File) * SquareSizes[Index] + BorderWidth,
                                  y:CGFloat(Rank) * SquareSizes[Index] + BorderWidth, width:
                                  SquareSizes[Index], height:SquareSizes[Index]))
>                  }
>              }
>          }
3434c4252
<          for Index in 0...29
---
>          for Index in 0 ..< 30
3437c4255
<              Captures.append(CGRect(x:CGFloat(Index) * 44, y:1262, width:56, height:56))
---
>              CapturesRects.append(CGRect(x:CGFloat(Index) * 44, y:1262, width:56, height:56))
3439c4257,4282
<
```

```
---
>     // the half board border width for the largest board //
>     HalfBorderWidth = BorderWidth * 0.5
>
>     // set the distance to the square center //
>     SquareCenter = BorderWidth + SquareSizes[2] * 0.5
>
>     // set the view distance to the square center //
>     ViewSquareCenter = SquareCenter * 0.5
>
>     // set the rectangle for the captured pieces background //
>     CapturesRectBackground = CGRect(x:0, y:1262, width:GetBoardSize(2), height:58)
>
>     // set the main board bounds //
>     MainBoardBounds = CGRect(x:HalfBorderWidth, y:HalfBorderWidth, width:ViewBoundsSize(2) -
>                              BorderWidth, height:ViewBoundsSize(2) - BorderWidth)
>
>     // set the inner board bounds //
>     InnerBoardBounds = CGRect(x:BorderWidth, y:BorderWidth, width:GetBoardSize(2) - 2 *
>                               BorderWidth, height:GetBoardSize(2) - 2 * BorderWidth)
>
>     // set the board border bezier path rectangle //
>     BorderRectangle = NSBezierPath(rect:CGRect(x:HalfBorderWidth, y:HalfBorderWidth, width:
>                                    GetBoardSize(2) - BorderWidth, height:GetBoardSize(2) -
>                                    BorderWidth))
>
3441c4284
<         BorderBounds.lineWidth = Border.WidthRet
---
>         BorderRectangle.lineWidth = BorderWidth
3445,3446c4288,4310
<     func GetCaptureRect(_ Index:Int) -> CGRect {return Captures[Index]}
---
>     func GetCaptureRect(_ Index:Int) -> CGRect {return CapturesRect[Index]}
>
>     // get the view bounds for the specified board set //
>     func GetViewBounds(_ Set:Int) -> CGRect {return ViewBounds[Set]}
>
>     // get the captured pieces rectangle //
>     func GetCapturesRectBackground() -> CGRect {return CapturesRectBackground}
>
>     // get the number of board sizes //
>     func GetBoardSizeCount() -> Int {return BoardSizes.count}
>
>     // get the view square size for the specified board set //
>     func ViewSquareSize(_ Set:Int) -> CGFloat {return SquareSizes[Set] / 2}
>
>     // the view board size for the specified board set //
>     func ViewBoardSize(_ Set:Int) -> CGFloat {return BoardSizes[Set] / 2}
>
>     // get the square size for the specified board set //
>     func GetSquareSize(_ Set:Int) -> CGFloat {return SquareSizes[Set]}
>
>     // the board size for the specified board set //
>     func GetBoardSize(_ Set:Int) -> CGFloat {return BoardSizes[Set]}
3448c4312
<     func GetRect(_ Rotated:Bool, Point:APoint) -> CGRect
---
>     func GetRect(_ Rotated:Bool, Set:Int, Point:APoint) -> CGRect
3450,3451c4314,4316
<         // return the square rectangle based on if the board is rotated or not //
<         return Rotated ? Squares[63 - Point.YI * 8 - Point.XI] : Squares[Point.YI * 8 + Point.XI]
---
>         // if the board is rotated //
>         return Rotated ? SquareRects[63 - Point.YI * 8 - Point.XI + 64 * Set] : SquareRects[Point.YI
>                          * 8 + Point.XI + 64 * Set]
3455c4320
<     func GetRect(_ Rotated:Bool, Rank:Int, File:Int) -> CGRect
---
>     func GetRect(_ Rotated:Bool, Set:Int, Rank:Int, File:Int) -> CGRect
3456a4322,4327
>         // set the x value for the point //
>         Point.XI = File
>
>         // set the y value for the point //
>         Point.YI = Rank
>
3458c4329
<         return GetRect(Rotated, Point:APoint(File:File, Y:Rank))
---
>         return GetRect(Rotated, Set:Set, Point:Point)
3462c4333
<     func GetRect(_ Rotated:Bool, Square:String) -> CGRect
---
>     func GetRect(_ Rotated:Bool, Set:Int, Square:String) -> CGRect
3463a4335,4340
>         // get the x value from the square number on the board //
>         Point.XI = (Int(Square) == nil) ? 0 : Int(Square)! % 8
>
>         // get the y value from the square number on the board //
>         Point.YI = (Int(Square) == nil) ? 0 : Int(Square)! / 8
>
3465,3466c4342
<         return GetRect(Rotated, Point:APoint(X:(Int(Square) == nil) ? 0 : Int(Square)! % 8, Y:
<                        (Int(Square) == nil) ? 0 : Int(Square)! / 8))
---
>         return GetRect(Rotated, Set:Set, Point:Point)
3472a4349,4360
>     // a loop index //
>     var Index:Int = 0
>
>     // the file index //
>     var File:Int = 0
>
>     // the rank index //
>     var Rank:Int = 0
>
>     // the board dimensions //
>     let Dims:BoardDimensions = BoardDimensions()
>
3490,3493c4378,4381
<
<     // the board dimensions //
<     let Dims:BoardDimensions = BoardDimensions()
<
---
>
>     // the context //
>     var Context:CGContext!
3498c4386
<         let Context = CoreGraphics().CreateContext(Dims.Board.WidthRet, Height:Dims.Board.WidthRet)
---
>         Context = CoreGraphics().CreateContext(Dims.GetBoardSize(2), Height:Dims.GetBoardSize(2))
3501c4389
<         for Index in 0..<Squares.count / 2
---
>         for Index in 0 ..< Squares.count / 2
3503,3504c4391,4392
<             // loop over all the ranks and files //
<             for Rank in 0...7 {for File in 0...7
---
>             // loop over all ranks //
>             for Rank in 0 ..< 8
3506,3510c4394,4414
<                 // draw the light or dark square into the board context //
<                 Context.draw(CoreGraphics().NSImageToCGImage(NSImage(named:Squares[Index * 2 + (Rank +
<                              File) % 2])!), in:Dims.Square.WidthRet), in:Dims.GetRect(false,
<                              Rank:Rank, File:File), byTiling:false)
<             }}
---
>                 // loop over all files //
>                 for File in 0 ..< 8
>                 {
>                     // if is a light square //
>                     if ((Rank + File) % 2) == 1
>                     {
>                         // draw the light square //
>                         Context.draw(CoreGraphics().NSImageToCGImage(NSImage(named:Squares[Index * 2])!,
>                                      Size:Dims.GetSquareSize(2)), in:Dims.GetRect(false, Set:2, Rank:Rank,
>                                      File:File), byTiling:false)
>                     }
>                     // if is a dark square //
>                     else
>                     {
>                         // draw the dark square //
>                         Context.draw(CoreGraphics().NSImageToCGImage(NSImage(named:Squares[Index * 2 +
>                                      1])!, Size:Dims.GetSquareSize(2)), in:Dims.GetRect(false, Set:2,
>                                      Rank:Rank, File:File), byTiling:false)
>                     }
>                 }
>             }
3512c4416
<         // append the chess board image to the board images //
---
>         // create a grid of light and dark chess board squares //
3523,3524c4427,4428
<         // loop over all the chess board names and add the button title //
<         for TheTitle in Boards {Button.addItem(withTitle:TheTitle)}
---
>         // loop over all the chess board names //
>         for Index in 0 ..< Boards.count {Button.addItem(withTitle:Boards[Index])}
3533a4438,4440
>     // the board dimensions //
>     let Dims:BoardDimensions = BoardDimensions()
>
3535,3537c4442,4451
<     let Pieces:[String] = ["whitepawn", "whiteknight", "whitebishop", "whiterook", "whitequeen",
<                            "whiteking", "blackpawn", "blackknight", "blackbishop", "blackrook",
<                            "blackqueen", "blackking"]
---
>     let Pieces:[String] = ["whitepawn1", "whiteknight1", "whitebishop1", "whiterook1", "whitequeen1",
>                            "whiteking1", "blackpawn1", "blackknight1", "blackbishop1", "blackrook1",
>                            "blackqueen1", "blackking1", "whitepawn2", "whiteknight2", "whitebishop2",
>                            "whiterook2", "whitequeen2", "whiteking2", "blackpawn2", "blackknight2",
>                            "blackbishop2", "blackrook2", "blackqueen2", "blackking2", "whitepawn3",
>                            "whiteknight3", "whitebishop3", "whiterook3", "whitequeen3", "whiteking3",
>                            "blackpawn3", "blackknight3", "blackbishop3", "blackrook3", "blackqueen3",
>                            "blackking3", "whitepawn4", "whiteknight4", "whitebishop4", "whiterook4",
>                            "whitequeen4", "whiteking4", "blackpawn4", "blackknight4", "blackbishop4",
>                            "blackrook4", "blackqueen4", "blackking4"]
3547a4462,4467
>     // a list index //
>     var Index:Int = 0
>
>     // the board set //
>     var Set:Int = 0
>
3550,3552d4469
<
<     // the board dimensions //
<     let Dims:BoardDimensions = BoardDimensions()
3557,3558c4474,4475
<         // loop over all the chess piece names to append the nsimage //
<         for Num in 1...4 {for Piece in Pieces {NSImages.append(NSImage(named:Piece + "\(Num)")!)}}
---
>         // loop over all the chess piece names //
>         for Index in 0 ..< Pieces.count {NSImages.append(NSImage(named:Pieces[Index])!)}
3560,3561c4477,4478
<         // loop over all the board sets and images //
<         for TheImage in NSImages
---
>         // loop over all the board sets //
>         for Set in 0 ..< Dims.GetBoardSizeCount()
3563,3564c4480,4486
<             // append a cgimage for the specified dimensions //
<             CGImages.append(CoreGraphics().NSImageToCGImage(TheImage, Size:Dims.Square.WidthRet))
---
>             // loop over all the images //
>             for Index in 0 ..< NSImages.count
>             {
>                 // append a cg image for the specified dimensions //
>                 CGImages.append(CoreGraphics().NSImageToCGImage(NSImages[Index], Size:Dims
>                                 .GetSquareSize(Set)))
>             }
3574,3575c4496,4497
<         // loop over all the piece sets and add the button title //
<         for TheTitle in Sets {Button.addItem(withTitle:TheTitle)}
---
>         // loop over all the piece sets //
>         for Index in 0 ..< Sets.count {Button.addItem(withTitle:Sets[Index])}
3589c4511
<     func GetPieceCGImage(_ Piece:Int) -> CGImage
---
>     func GetPieceCGImage(_ Piece:Int, Set:Int) -> CGImage
3592c4514,4515
<             return Piece > 0 ? CGImages[Piece + PieceSet * 12 - 1] : CGImages[PieceSet * 12 - Piece + 5]
---
>             return Piece > 0 ? CGImages[Piece + PieceSet * 12 + Set * 48 - 1] : CGImages[PieceSet * 12
>                                                                                 + Set * 48 - Piece + 5]
3619c4562
<         for Index in 0..<DictionaryCount
---
>         for Index in 0 ..< DictionaryCount
3646c4569
<         for Index in 0..<DictionaryCount * 2
---
>         for Index in 0 ..< DictionaryCount * 2
3684,3686d4606
<     // all the captured pieces //
<     var Captures:[Int] = []
<
3691c4611
<     var CaptureSquares:[String] = []
---
>     var ListOfCaptureSquares:[String] = []
3693,3694c4613,4614
<     // the list of threatened squares //
<     var ThreatSquares:[String] = []
---
>     // the list of threaten squares //
>     var ListOfThreatenSquares:[String] = []
3703c4623
<     var Start:APoint = APoint()
---
>     var StartSquare:APoint = APoint()
3706,3707c4626,4636
<     var End:APoint = APoint()
---
>     var EndSquare:APoint = APoint()
>
>     // a temporary square //
>     var TempSquare:APoint = APoint()
>
>     // the list of captured white pieces to display //
>     var CapturedWhitePieces:[Int] = []
>
>     // the list of captured black pieces to display //
>     var CapturedBlackPieces:[Int] = []
3714,3718c4643,4647
<     // store the captured piece //
<     func StoreCapture(_ Piece:Int)
<     {
<         // store the captured piece if there was one //
<         if Piece != 0 {Captures.append(Piece)}
---
>     // increment the captured white pieces //
>     func IncrementCapturedWhitePieces(_ Index:Int) {CapturedWhitePieces[Index] += 1}
>
>     // increment the captured black pieces //
>     func IncrementCapturedBlackPieces(_ Index:Int) {CapturedBlackPieces[Index] += 1}
3720,3723d4648
<         // sort the captured pieces //
<         Captures = Captures.sorted(by:>).filter {$0 > 0} + Captures.sorted().filter {$0 < 0}
<     }
<
3728c4653,4654
<         Start.Set(SquareLabel.SquareFromUCI(UCIMove, Start:true, Dims:Dims, Rotated:Rotated))
---
>         StartSquare.Set(SquareLabel.SquareFromUCI(UCIMove, Start:true, Dims:Dims,
>                         Rotated:Rotated))
3731c4657,4658
<         End.Set(SquareLabel.SquareFromUCI(UCIMove, Start:false, Dims:Dims, Rotated:Rotated))
---
>         EndSquare.Set(SquareLabel.SquareFromUCI(UCIMove, Start:false, Dims:Dims,
>                       Rotated:Rotated))
3735c4662,4663
<     func UpdateCaptureThreatLists(_ CaptureSquares:[String], ThreatSquares:[String])
---
>     func UpdateCaptureThreatenHighlightLists(_ ListOfCaptureSquares:[String], ListOfThreatenSquares:
>                                              [String])
3737,3738c4665,4669
<         // store the list of capture squares and threatened squares //
<         (self.CaptureSquares, self.ThreatSquares) = (CaptureSquares, ThreatSquares)
---
>         // store the list of capture squares //
>         self.ListOfCaptureSquares = ListOfCaptureSquares
>
>         // remove the last element from the list of capture squares since it is empty //
>         _ = self.ListOfCaptureSquares.popLast()
3740,3741c4671,4675
<         // remove the last element from the list of capture and threaten squares since empty //
<         (_, _) = (self.CaptureSquares.popLast(), self.ThreatSquares.popLast())
---
>         // store the list of capture squares //
>         self.ListOfThreatenSquares = ListOfThreatenSquares
>
>         // remove the last element from the list threaten squares since it is empty //
>         _ = self.ListOfThreatenSquares.popLast()
3750,3751c4684,4688
<         // clear the captures //
<         Captures.removeAll()
---
>         // initialize the captured white pieces to display //
>         CapturedWhitePieces = Array(repeating:0, count:5)
>
>         // initialize the captured black pieces to display //
>         CapturedBlackPieces = Array(repeating:0, count:5)
3753c4690,4693
<
---
>
>
>     // swap the start and end squares for the move //
>     func SwapBoardSquares() {TempSquare.Set(StartSquare); SquaresEqual(); EndSquare.Set(TempSquare)}
3755c4695
<     func SquaresNotEqual() -> Bool {return !Start.Equals(End)}
---
>     func StartEndSquaresNotEqual() -> Bool {return !StartSquare.Equals(EndSquare)}
3758c4698
<     func SquaresEqual() {Start.Set(End)}
---
>     func SquaresEqual() {StartSquare.Set(EndSquare)}
3789c4729
<     func EndSquareValue() -> Int {return RFV(End)}
---
>     func EndSquareValue() -> Int {return RFV(EndSquare)}
3792c4732
<     func StartSquareValue() -> Int {return RFV(Start)}
---
>     func StartSquareValue() -> Int {return RFV(StartSquare)}
3795c4735
<     func RemovePawnCapturedEnPassant() {RFV(End.XI, Start.YI, 0)}
---
>     func RemovePawnCapturedEnPassant() {RFV(EndSquare.XI, StartSquare.YI, 0)}
3798c4738
<     func EndSquareValue(_ MovingPiece:Int) {RFV(End, MovingPiece)}
---
>     func EndSquareValue(_ MovingPiece:Int) {RFV(EndSquare, MovingPiece)}
3801c4741
<     func RemoveStartSquarePiece() {RFV(Start.XI, Start.YI, 0)}
---
>     func RemoveStartSquarePiece() {RFV(StartSquare.XI, StartSquare.YI, 0)}
3827,3829c4767,4780
<         // store the board data values //
<         (Layout, CaptureSquares, ThreatSquares, Captures) = (BPData.Layout, BPData.CaptureSquares,
<         BPData.ThreatSquares, BPData.Captures)
---
>         // store the board layout from the board and pieces data //
>         Layout = BPData.Layout
>
>         // store the list of capture squares from the board and pieces data //
>         ListOfCaptureSquares = BPData.ListOfCaptureSquares
>
>         // store the list of threaten squares from the board and pieces data //
>         ListOfThreatenSquares = BPData.ListOfThreatenSquares
>
>         // store the captured white pieces //
>         CapturedWhitePieces = BPData.CapturedWhitePieces
>
>         // store the captured black pieces //
>         CapturedBlackPieces = BPData.CapturedBlackPieces
3832c4783
<         Start.Set(BPData.Start)
---
>         StartSquare.Set(BPData.StartSquare)
3835c4786
<         End.Set(BPData.End)
---
>         EndSquare.Set(BPData.EndSquare)
3875,3880d4825
<     // the initializer //
<     init() {}
<
<     // the initializer //
<     init(_ X:Int, Y:Int) {XI = X; YI = Y; XF = CGFloat(X); YF = CGFloat(Y)}
3884,3885c4830,4871
<         // update the indices and float values //
<         (XI, YI, XF, YF) = (max(min(X, 7), 0), max(min(Y, 7), 0), CGFloat(X), CGFloat(Y))
---
>         // set the index for x //
>         XI = max(min(X, 7), 0)
>
>         // set the index for y //
>         YI = max(min(Y, 7), 0)
>
>         // set the cgfloat value of x //
>         XF = CGFloat(X)
>
>         // set the cgfloat value of y //
>         YF = CGFloat(Y)
>
>         // if the board is rotated //
>         if Rotated
>         {
>             // set the x coordinate of the square center in display points //
>             XC = CGFloat(7 - XI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
3887,3889c4847,4858
<         // set the x and y coordinates of the square center in display points //
<         (XC, YC) = (CGFloat(Rotated ? 7 - XI : XI) * Dims.Square.Width + Dims.Square.Bounds.midX,
<                     CGFloat(Rotated ? 7 - YI : YI) * Dims.Square.Width + Dims.Square.Bounds.midY)
---
>             // set the y coordinate of the square center in display points //
>             YC = CGFloat(7 - YI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
>         }
>         // board is not rotated //
>         else
>         {
>             // set the x coordinate of the square center in display points //
>             XC = CGFloat(XI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
>
>             // set the y coordinate of the square center in display points //
>             YC = CGFloat(YI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
>         }
3895,3907c4864,4883
<         // set the bounded x and y locations of the mouse in display points //
<         (XL, YL) = (min(max(Location.x, Dims.Square.Bounds.midX), Dims.Board.Width - Dims.Square
<                     .Bounds.midX), min(max(Location.y, Dims.Square.Bounds.midY), Dims.Board.Width - Dims.Square
<                     .Bounds.midY))
<
<         // set the x and y indices of the possible square value from 0 to 7 //
<         (XI, YI) = (max(min(Int((XL - Dims.Border.Width) / Dims.Square.Width), 7), 0), max(min(Int((YL
<                     - Dims.Border.Width) / Dims.Square.Width), 7), 0))
<
<         // set the x and y coordinates of the square center in display points //
<         (XC, YC) = (CGFloat(XI) * Dims.Square.Width + Dims.Square.Bounds.midX, CGFloat(YI) * Dims
<                     .Square.Width + Dims.Square.Bounds.midX)
---
>         // set the bounded x location of the mouse in display points //
>         XL = min(max(Location.x, Dims.ViewSquareCenter), Dims.ViewBoardSize(2) - Dims
>              .ViewSquareCenter)
>
>         // set the bounded y location of the mouse in display points //
>         YL = min(max(Location.y, Dims.ViewSquareCenter), Dims.ViewBoardSize(2) - Dims
>              .ViewSquareCenter)
>
>         // set the x index of the possible square value from 0 to 7 //
>         XI = max(min(Int((XL - Dims.HalfBorderWidth) / Dims.ViewSquareSize(2)), 7), 0)
>
>         // set the y index of the possible square value from 0 to 7 //
>         YI = max(min(Int((YL - Dims.HalfBorderWidth) / Dims.ViewSquareSize(2)), 7), 0)
>
>         // set the x coordinate of the square center in display points //
>         XC = CGFloat(XI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
>
>         // set the y coordinate of the square center in display points //
>         YC = CGFloat(YI) * Dims.ViewSquareSize(2) + Dims.ViewSquareCenter
3911,3912c4887,4891
<         // save the cgfloat value of xi and yi //
<         (XF, YF) = (CGFloat(XI), CGFloat(YI))
---
>         // save the cgfloat value of xi //
>         XF = CGFloat(XI)
>
>         // save the cgfloat value of yi //
>         YF = CGFloat(YI)
3916c4895
<     func Set(_ P:APoint)
---
>     func Set(_ Point:APoint)
3918,3919c4897,4919
<         // initialize the variables //
<         (XI, YI, XF, YF, XC, YC, XL, YL) = (P.XI, P.YI, P.XF, P.YF, P.XC, P.YC, P.XL, P.YL)
---
>         // store the xi value //
>         XI = Point.XI
>
>         // store the yi value //
>         YI = Point.YI
>
>         // store the xf value //
>         XF = Point.XF
>
>         // store the yf value //
>         YF = Point.YF
>
>         // store the xc value //
>         XC = Point.XC
>
>         // store the yc value //
>         YC = Point.YC
>
>         // store the xl value //
>         XL = Point.XL
>
>         // store the yl value //
>         YL = Point.YL
3925a4926,4928
>     // a loop index //
>     var Index:Int = 0
>
3942c4945
<         for TheInfo in SavedGameInfoList {if TheInfo.isEmpty
---
>         for Index in 0 ..< SavedGameInfoList.count
3944,3949c4947,4958
<             // append the attributed move string //
<             GameInfoList.append(GameInfoFont.GetString("\n" + TheInfo + "\n", White:false))
<
<             // append the row height //
<             Heights.append(GameInfoFont.GetStringHeight(GameInfoList.last!, Width:250))
<         }}
---
>             // if the saved game info list value at this index is nonempty //
>             if !SavedGameInfoList[Index].isEmpty
>             {
>                 // append the attributed move string //
>                 GameInfoList.append(GameInfoFont.GetString("\n" + SavedGameInfoList[Index] + "\n",
>                                     White:false))
>
>                 // append the row height //
>                 Heights.append(GameInfoFont.GetStringHeight(GameInfoList[GameInfoList.count - 1],
>                                Width:250))
>             }
>         }
3964a4974,4976
>     // a loop index //
>     var Index:Int = 0
>
3966c4978
<     var PGNS:[NSAttributedString] = []
---
>     var PGNMovesList:[NSAttributedString] = []
3975c4987,4991
<     func GetViewText(_ Row:Int) -> NSAttributedString {return PGNS[min(Row, PGNS.count - 1)]}
---
>     func GetViewText(_ Row:Int) -> NSAttributedString
>     {
>         // return the text attributed string value at the specified row //
>         return PGNMovesList[min(Row, PGNMovesList.count - 1)]
>     }
3980c4996
<         // while data bounds exceeded by the row //
---
>         // if data bounds exceeded by the row //
3988c5004,5005
<     func RefreshPGNS(_ SavedPGNS:[String], MoveListFont:FontSet)
---
>     func RefreshPGNMovesList(_ SavedPGNMovesList:[String], MoveListFont:FontSet,
>                              DisplayIllustrations:Bool, IllustrationSize:Int)
3991c5008
<         PGNS.removeAll()
---
>         PGNMovesList.removeAll()
3997c5014
<         for TheMove in SavedPGNS
---
>         for Index in 0 ..< SavedPGNMovesList.count
4000c5017
<             PGNS.append(MoveListFont.GetString("\n" + TheMove, White:false))
---
>             PGNMovesList.append(MoveListFont.GetString("\n" + SavedPGNMovesList[Index], White:false))
4003c5020
<             Heights.append(MoveListFont.GetStringHeight(PGNS.last!, Width:250) + 25)
---
>             Heights.append(MoveListFont.GetStringHeight(PGNMovesList[Index], Width:250) + 25)
4011c5028
<     func LastRow() -> Int {return max(0, PGNS.count - 1)}
---
>     func LastRow() -> Int {return max(0, PGNMovesList.count - 1)}
4014c5031
<     func numberOfRows(in TableView:NSTableView) -> Int {return PGNS.count}
---
>     func numberOfRows(in TableView:NSTableView) -> Int {return PGNMovesList.count}
4022c5039,5042
<
---
>
>
>     // the index in a list //
>     var Index:Int = 0
4027c5047
<     var Indices:[Int] = []
---
>     var GameListIndices:[Int] = []
4030c5050
<     var List:[String] = []
---
>     var GameList:[String] = []
4033c5053
<     var AttributedList:[NSAttributedString] = []
---
>     var AttributedGameList:[NSAttributedString] = []
4036c5056
<     var ListOriginal:[String] = []
---
>     var GameListOriginal:[String] = []
4039c5059
<     var IndicesOriginal:[Int] = []
---
>     var GameListIndicesOriginal:[Int] = []
4045c5065
<     func DatabaseSize() -> Int {return ListOriginal.count}
---
>     func DatabaseSize() -> Int {return GameListOriginal.count}
4051c5071
<     func numberOfRows(in TableView:NSTableView) -> Int {return List.count}
---
>     func numberOfRows(in TableView:NSTableView) -> Int {return GameList.count}
4054c5074
<     func StoreGameList(_ TheList:[String])
---
>     func StoreGameList(_ GameListToStore:[String])
4057c5077
<         List = TheList
---
>         GameList = GameListToStore
4060,4063c5080,5083
<         _ = List.popLast()
<
<         // set the list of game indices //
<         Indices = Array(0..<Int(List.count))
---
>         _ = GameList.popLast()
>
>         // clear the game list indices //
>         GameListIndices.removeAll()
4065,4066c5085,5092
<         // save the original game list and indices //
<         (ListOriginal, IndicesOriginal) = (List, Indices)
---
>         // loop over the entire game list //
>         for Index in 0 ..< GameListToStore.count {GameListIndices.append(Index)}
>
>         // save the original game list //
>         GameListOriginal = GameList
>
>         // save the original game list indices //
>         GameListIndicesOriginal = GameListIndices
4072,4073c5098,5109
<         // reset the list if the search string is empty //
<         if SearchString.isEmpty {(List, Indices) = (ListOriginal, IndicesOriginal); return}
---
>         // if the search string is empty //
>         if SearchString.isEmpty
>         {
>             // set the current game list to the original game list //
>             GameList = GameListOriginal
>
>             // set the current game list indices to the original game list indices //
>             GameListIndices = GameListIndicesOriginal
>
>             // exit function since don't need to condense the game list //
>             return
>         }
4076c5112
<         List.removeAll()
---
>         GameList.removeAll()
4079c5115
<         Indices.removeAll()
---
>         GameListIndices.removeAll()
4082c5118
<         for Index in 0..<ListOriginal.count
---
>         for Index in 0 ..< GameListOriginal.count
4085c5121
<             if ListOriginal[Index].lowercased().range(of:SearchString.lowercased()) != nil
---
>             if GameListOriginal[Index].lowercased().range(of:SearchString.lowercased()) != nil
4088c5124
<                 List.append(ListOriginal[Index])
---
>                 GameList.append(GameListOriginal[Index])
4091c5127
<                 Indices.append(IndicesOriginal[Index])
---
>                 GameListIndices.append(GameListIndicesOriginal[Index])
4097c5133
<     func GetGameIndex(_ Row:Int) -> Int {return Indices[Row]}
---
>     func GetGameIndex(_ Row:Int) -> Int {return GameListIndices[Row]}
4103c5139
<         AttributedList.removeAll()
---
>         AttributedGameList.removeAll()
4109c5145
<         for TheGame in List
---
>         for Index in 0 ..< GameList.count
4112c5148,5149
<             AttributedList.append(GameListFont.GetString("\n" + TheGame + "\n", White:false))
---
>             AttributedGameList.append(GameListFont.GetString("\n" + GameList[Index] + "\n", White:
>                                       false))
4115c5152
<             Heights.append(GameListFont.GetStringHeight(AttributedList.last!, Width:605))
---
>             Heights.append(GameListFont.GetStringHeight(AttributedGameList[Index], Width:605))
4120c5157
<     func GetRow(_ Row:Int) -> NSAttributedString {return AttributedList[Row]}
---
>     func GetRow(_ Row:Int) -> NSAttributedString {return AttributedGameList[Row]}
4126c5163
<         return GameListFont.GetString(AttributedList[Row].string, White:true)
---
>         return GameListFont.GetString(AttributedGameList[Row].string, White:true)
4130c5167
<     func GameDescription(_ GameNumber:Int) -> String {return ListOriginal[GameNumber]}
---
>     func GameDescription(_ GameNumber:Int) -> String {return GameListOriginal[GameNumber]}
4165a5203,5205
>     // a loop index //
>     var Index:Int = 0
>
4235,4236c5275,5279
<         // update the clock background color opacity and set refresh all to true //
<         (BackgroundColorOpacity, RefreshAll) = (CGFloat(Opacity) / 100, true)
---
>         // update the clock background color opacity //
>         BackgroundColorOpacity = CGFloat(Opacity) / 100
>
>         // set that the clock face needs to be refreshed //
>         RefreshAll = true
4249,4252c5292,5299
<         // set the clock colors //
<         (LightenedColor, DarkenedColor, BezelColor) = (BlendedBackgroundColor.blended(withFraction:
<         0.05, of:NSColor.white)!, BlendedBackgroundColor.blended(withFraction:0.05, of:NSColor
<         .black)!, BlendedBackgroundColor.blended(withFraction:0.1, of:NSColor.white)!)
---
>         // set the lightened color from the blended background color and white //
>         LightenedColor = BlendedBackgroundColor.blended(withFraction:0.05, of:NSColor.white)!
>
>         // set the darkened color from the blended background color and black //
>         DarkenedColor = BlendedBackgroundColor.blended(withFraction:0.05, of:NSColor.black)!
>
>         // set the bezel color by mixing in white //
>         BezelColor = BlendedBackgroundColor.blended(withFraction:0.1, of:NSColor.white)!
4255c5302
<         for Index in 0..<Circles.count
---
>         for Index in 0 ..< Circles.count
4266,4267c5313,5317
<         // allow concurrent drawing, and set clock radius and diameter //
<         (canDrawConcurrently, Radius, Diameter) = (true, frame.size.width, frame.size.width * 2)
---
>         // allow concurrent drawing //
>         canDrawConcurrently = true
>
>         // set the clock radius //
>         Radius = frame.size.width
4269c5319,5322
<         // translate the clock origin to be the center of the view //
---
>         // set the clock diameter //
>         Diameter = Radius * 2
>
>         // translate the clock origin to be the middle of the view //
4271,4275c5324,5326
<
<         // set the view and bezel bounds //
<         (ClockFaceLayer, ViewBounds, BezelBounds) = (CoreGraphics().CreateLayer(Diameter, Height:
<         Diameter), CGRect(x:-Radius, y:-Radius, width:Diameter, height:Diameter), CGRect(x:5 - Radius,
<         y:5 - Radius, width:Diameter - 10, height:Diameter - 10))
---
>
>         // create the clock face layer //
>         ClockFaceLayer = CoreGraphics().CreateLayer(Diameter, Height:Diameter)
4279c5330,5336
<
---
>
>
>         // set the view bounds //
>         ViewBounds = CGRect(x:-Radius, y:-Radius, width:Diameter, height:Diameter)
>
>         // set the clock bezel bounds //
>         BezelBounds = CGRect(x:5 - Radius, y:5 - Radius, width:Diameter - 10, height:Diameter - 10)
4281c5338
<         for Index in 0...59
---
>         for Index in 0 ..< 60
4283c5340
<             // set the current radians //
---
>             // get the current radians //
4286,4287c5343,5344
<             // set the current cosine and sine //
<             (Cosine, Sine) = (cos(Radians), sin(Radians))
---
>             // get the current cosine //
>             Cosine = cos(Radians)
4289,4292c5346,5354
<             // set the start and end points of the tick mark //
<             (StartPoint, EndPoint) = (CGPoint(x:0.93 * Radius * Cosine, y:0.93 * Radius * Sine),
<             CGPoint(x:0.97 * Radius * Cosine, y:0.97 * Radius * Sine))
---
>             // get the current sine //
>             Sine = sin(Radians)
>
>             // set the start point of the tick mark //
>             StartPoint = CGPoint(x:0.93 * Radius * Cosine, y:0.93 * Radius * Sine)
>
>             // set the end point of the tick mark //
>             EndPoint = CGPoint(x:0.97 * Radius * Cosine, y:0.97 * Radius * Sine)
4321c5383
<         for Index in 0...599
---
>         for Index in 0 ..< 600
4326,4327c5388,5392
<             // set the cosine and sine //
<             (Cosine, Sine) = (cos(Radians), sin(Radians))
---
>             // set the cosine of the radians //
>             Cosine = cos(Radians)
>
>             // set the sine of the radians //
>             Sine = sin(Radians)
4337,4339c5402,5404
<                        min(SecondsOrigin.last!.y, SecondsEndpoint.last!.y) - 1, width:abs(SecondsEndpoint
<                        .last!.x - SecondsOrigin.last!.x) + 2, height:abs(SecondsEndpoint.last!.y -
<                        SecondsOrigin.last!.y) + 2))
---
>                            min(SecondsOrigin.last!.y, SecondsEndpoint.last!.y) - 1, width:
>                            abs(SecondsEndpoint.last!.x - SecondsOrigin.last!.x) + 2, height:
>                            abs(SecondsEndpoint.last!.y - SecondsOrigin.last!.y) + 2))
4346,4350c5411,5416
<         // return if the clock face layer doesn't exist //
<         if ClockFaceLayer == nil {return}
<
<         // draw the clock face layer into the current context //
<         NSGraphicsContext.current?.cgContext.draw(ClockFaceLayer, in:bounds)
---
>         // is the clock face layer exists //
>         if ClockFaceLayer != nil
>         {
>             // draw the clock face layer into the current context //
>             NSGraphicsContext.current?.cgContext.draw(ClockFaceLayer, in:bounds)
>         }
4405c5471
<         for Index in 0..<Circles.count
---
>         for Index in 0 ..< Circles.count
4457,4458c5523,5524
<         // update the elapsed time and radians //
<         (ElapsedTime, Radians) = (Time, CGFloat.pi * CGFloat(Time) / 18000)
---
>         // update the elapsed time //
>         ElapsedTime = Time
4459a5526,5528
>
>         // set the radians for the minute hand //
>         Radians = CGFloat.pi * CGFloat(ElapsedTime) / 18000
4489c5558
<                    width:abs(MinutesEndpoint.x) + 2, height:abs(MinutesEndpoint.y) + 2))
---
>                        width:abs(MinutesEndpoint.x) + 2, height:abs(MinutesEndpoint.y) + 2))
4524c5593
<     var NumFrames:Int = 0
---
>     var NumberOfFrames:Int = 0
4558a5628,5630
>     // a list index //
>     var Index:Int = 0
>
4629c5701
<             Points.append(CGPoint(x:CGFloat(-NumOfFrames) * FrameSize, y:0))
---
>             Points.append(CGPoint(x:CGFloat(-NumberOfFrames) * FrameSize, y:0))
4632c5704
<             FrameStripContext.draw(Frame, in:CGRect(x:CGFloat(NumOfFrames) * FrameSize, y:0, width:
---
>             FrameStripContext.draw(Frame, in:CGRect(x:CGFloat(NumberOfFrames) * FrameSize, y:0, width:
4636c5708
<             NumOfFrames += 1
---
>             NumberOfFrames += 1
4727,4730c5799,5804
<         // set the frame strip image and max position //
<         (FrameStripImage, MaxPosition) = (FrameStripContext.makeImage(), max(NumOfFrames -
<         FramesWide * Layers, 0))
---
>         // create the frame strip image //
>         FrameStripImage = FrameStripContext.makeImage()
>
>         // set the max position //
>         MaxPosition = max(NumberOfFrames - FramesWide * Layers, 0)
4732c5806
<         for Index in 0..<Layers
---
>         for Index in 0 ..< Layers
4736,4737c5810,5812
<                      CGFloat(Layers - 1 - Index) * FrameSize, width:CGFloat(FrameStripContext.width),
<                      height:CGFloat(FrameStripContext.height)), byTiling:false)
---
>                                      CGFloat(Layers - 1 - Index) * FrameSize, width:
>                                      CGFloat(FrameStripContext.width),
>                                      CGFloat(FrameStripContext.height)), byTiling:false)
4755c5830,5833
<
---
>
>
>         // allow concurrent drawing //
>         canDrawConcurrently = true
4759,4762c5837,5838
<         // update the frame size to retina, curent position, and height and width //
<         (canDrawConcurrently, FrameSize, CurrentPosition, Height, Width, TrackingRectangle) = (true,
<         FrameSize * 2, 0, frame.size.height * 2, frame.size.width * 2, addTrackingRect(bounds, owner:
<         self, userData:nil, assumeInside:false))
---
>         // update the frame size to retina display resolution //
>         FrameSize *= 2
4763a5840,5848
>
>         // set the current position to zero //
>         CurrentPosition = 0
>
>         // set the view height //
>         Height = frame.size.height * 2
>
>         // set the view width //
>         Width = frame.size.width * 2
4765c5850
<         Sketch.Initialize(self as NSView, Width:Width, Height:Height, LengthCGFloat(NumOfFrames)
---
>         Sketch.Initialize(self as NSView, Width:Width, Height:Height, LengthCGFloat(NumberOfFrames)
```

```
4767c5932
<
---
>
4771,4777c5856,5858
<       // create the frame strip context, the context, and get number of frames wide //
<       (FrameStripContext, Context, FramesWide) = (CoreGraphics().CreateContext(CGFloat(NumOfFrames)
<       * FrameSize, Height:FrameSize), CoreGraphics().CreateContext(Width, Height:Height), Int(Width
<       / FrameSize))
<
<       // reset number of frames to zero //
<       NumOfFrames = 0
---
>       // initialize the frame strip context //
>       FrameStripContext = CoreGraphics().CreateContext(CGFloat(NumberOfFrames) * FrameSize, Height:
>                                                        FrameSize)
4778a5860,5865
>       // create the context //
>       Context = CoreGraphics().CreateContext(Width, Height:Height)
>
>       // get the number of frames that can span the width of the view //
>       FramesWide = Int(Width / FrameSize)
>
4780a5868,5870
>
>       // add a tracking rectangle for the view //
>       TrackingRectangle = addTrackingRect(bounds, owner:self, userData:nil, assumeInside:false)
4783a5874,5876
>
>       // reset number of frames to zero //
>       NumberOfFrames = 0
4797c5890
<       var CurrentRect:CGRect = CGRect()
---
>       var CurrentHueRect:CGRect = CGRect()
4860c5953
<           case 6 : Board.UpdateWindowBackgroundColor()
---
>           case 6 : Board.SetWindowBackgroundColor(true)
4888,4891c5981,5998
<       // get the current rectangle based on the current hue //
<       CurrentRect = (CurrentHue == 0) ? CGRect(x:0.5, y:1, width:Width, height:38) : ((CurrentHue ==
<                      Hues.count - 1) ? CGRect(x:CGFloat(CurrentHue) * Width - 0.5, y:1, width:Width,
<                      height:38) : CGRect(x:CGFloat(CurrentHue) * Width, y:1, width:Width, height:38))
---
>       // if the current hue is zero //
>       if CurrentHue == 0
>       {
>           // set the first current hue rectangle //
>           CurrentHueRect = CGRect(x:0.5, y:1, width:Width, height:38)
>       }
>       // if the current hue is the last hue //
>       else if CurrentHue == (Hues.count - 1)
>       {
>           // set the last current hue rectangle //
>           CurrentHueRect = CGRect(x:CGFloat(CurrentHue) * Width - 0.5, y:1, width:Width, height:38)
>       }
>       // is between the first and last possible hue //
>       else
>       {
>           // set the current hue rectangle //
>           CurrentHueRect = CGRect(x:CGFloat(CurrentHue) * Width, y:1, width:Width, height:38)
>       }
4897c6004
<       NSBezierPath(rect:CurrentRect).fill()
---
>       NSBezierPath(rect:CurrentHueRect).fill()
4903c6010
<       NSBezierPath(rect:CurrentRect).stroke()
---
>       NSBezierPath(rect:CurrentHueRect).stroke()
4913c6020
<       for Index in 0...119
---
>       for Index in 0 ..< 120
4939,4942c6046,6054
<       // set the current hue, hue bar width, and set current graphics context to the hue layer //
<       (CurrentHue, Width, NSGraphicsContext.current) = (ChosenHue, frame.size.width / CGFloat(Hues
<       .count), NSGraphicsContext(cgContext:HueLayer.context!, flipped:false))
<
---
>       // set the current hue //
>       CurrentHue = ChosenHue
>
>       // set the current hue bar width //
>       Width = frame.size.width / CGFloat(Hues.count)
>
>       // set the current graphics context to be the hue layer context //
>       NSGraphicsContext.current = NSGraphicsContext(cgContext:HueLayer.context!, flipped:false)
>
4944c6056
<       for Index in 0..<Hues.count
---
>       for Index in 0 ..< Hues.count
4967a6080,6082
>   // loop index //
>   var Index:Int = 0
>
4978a6094,6096
>
>   // the raw entered fen string array //
>   var RawEnteredFENStringArray:[String] = []
4979a6098,6100
>   // a position image //
>   var PositionImage:NSImage = NSImage()
>
4982a6104,6106
>   // the info list //
>   var InfoList:[String] = []
>
5007c6131
<   func IsDefaultFENAndWhite() -> Bool
---
>   func IsDefaultFENLayoutAndWhite() -> Bool
5016,5022c6140,6154
<       // if the fen array is not of size six, set to fen defaults and return //
<       if FENStringArray.count != 6 {SetFENDefaults(); return}
<
<       // save the fen items into the info dictionary //
<       Inf.Update("FENPref", VSItems:["FENLayout":FENStringArray[0], "FENColor":FENStringArray[1]
<                  .lowercased(), "FENCastlingRights":FENStringArray[2], "FENEnpassantSquare":
<                  FENStringArray[3].lowercased(), "FENStartMoveNumber":FENStringArray[5]])
---
>       // if the fen string array is of the correct size //
>       if FENStringArray.count == 6
>       {
>           // save the fen items into the info dictionary //
>           Inf.Update("FENPref", VSItems:["FENLayout":FENStringArray[0], "FENColor":
>                      FENStringArray[1].lowercased(), "FENCastlingRights":FENStringArray[2],
>                      "FENEnpassantSquare":FENStringArray[3].lowercased(), "FENStartMoveNumber":
>                      FENStringArray[5]])
>       }
>       // the fen string was incomplete //
>       else
>       {
>           // set the default fen strings //
>           SetFENDefaults()
>       }
5075c6207,6221
<       EnteredFENStringArray = (EnteredFENString.components(separatedBy:" ")).filter {$0 != ""}
---
>
>       RawEnteredFENStringArray = EnteredFENString.components(separatedBy:" ")
>
>       // clear the entered fen string array //
>       EnteredFENStringArray.removeAll()
>
>       // loop over all the items in the raw entered fen string array //
>       for Index in 0 ..< RawEnteredFENStringArray.count
>       {
>           // if the raw entered string array element is not empty //
>           if !RawEnteredFENStringArray[Index].isEmpty
>           {
>               // copy over the nonempty element to the entered fen string array //
>               EnteredFENStringArray.append(RawEnteredFENStringArray[Index])
>           }
>       }
5078c6224
<       for Index in 0..<FENComponents.count {if EnteredFENStringArray.count == (Index + 1)
---
>       for Index in 0 ..< FENComponents.count
5080,5082c6226,6232
<           // append the corresponding fen string component to the entered fen string array //
<           EnteredFENStringArray.append(FENComponents[Index])
<       }}
---
>           // if the entered fen string array count is equal to index plus one //
>           if EnteredFENStringArray.count == (Index + 1)
>           {
>               // append the corresponding fen string component to the entered fen string array //
>               EnteredFENStringArray.append(FENComponents[Index])
>           }
>       }
5093,5101c6243,6253
<           DStng(DataBlackPlayer(GameData)), "TimeControl":DStng(DataGetTimeControl(GameData)),
<           "Opening":DStng(DataGetOpening(GameData)), "Section":DStng(DataGetSection(GameData)),
<           "BoardNumber":DStng(DataGetBoardNumber(GameData)), "WhiteElo":
<           DStng(DataGetWhiteElo(GameData)), "BlackElo":DStng(DataGetWhiteElo(GameData)),
<           "WhiteUSCF":DStng(DataGetWhiteUSCF(GameData)), "BlackUSCF":
<           DStng(DataGetBlackUSCF(GameData)), "Annotator":DStng(DataGetAnnotator(GameData)), "ECO":
<           DStng(DataGetECO(GameData)), "SiteLocation":DStng(DataGetSite(GameData)), "EventTitle":
<           DStng(DataGetEvent(GameData)), "RoundNumber":DStng(DataGetRound(GameData)), "EndResult":
<           DStng(DataGetResult(GameData)),"DateGamePlayed":DStng(DataGetDate(GameData))]}
---
>               DStng(DataBlackPlayer(GameData)), "TimeControl":DStng(DataGetTimeControl(GameData)),
>               "Opening":DStng(DataGetOpening(GameData)), "Section":
>               DStng(DataGetSection(GameData)), "BoardNumber":DStng(DataGetBoardNumber(GameData)),
>               "WhiteElo":DStng(DataGetWhiteElo(GameData)), "BlackElo":
>               DStng(DataGetWhiteElo(GameData)), "WhiteUSCF":DStng(DataGetWhiteUSCF(GameData)),
>               "BlackUSCF":DStng(DataGetBlackUSCF(GameData)), "Annotator":
>               DStng(DataGetAnnotator(GameData)), "ECO":DStng(DataGetECO(GameData)),
>               "SiteLocation":DStng(DataGetSite(GameData)), "EventTitle":
>               DStng(DataGetEvent(GameData)), "RoundNumber":DStng(DataGetRound(GameData)),
>               "EndResult":DStng(DataGetResult(GameData)),"DateGamePlayed":
>               DStng(DataGetDate(GameData))]}
5161,5169c6313,6322
<           PGNStr(PGNTags.BlackNameTextField), "SiteLocation":PGNStr(PGNTags.SiteLocationTextField),
<           "EventTitle":PGNStr(PGNTags.EventTitleTextField), "RoundNumber":PGNStr(PGNTags
<           .RoundNumberTextField), "TimeControl":PGNStr(PGNTags.TimeControlTextField), "Opening":
<           PGNStr(PGNTags.OpeningTextField), "Section":PGNStr(PGNTags.SectionTextField),
<           "BoardNumber":PGNStr(PGNTags.BoardNumberTextField), "WhiteElo":PGNStr(PGNTags
<           .WhiteEloTextField), "BlackElo":PGNStr(PGNTags.BlackEloTextField), "WhiteUSCF":
<           PGNStr(PGNTags.WhiteUSCFTextField), "BlackUSCF":PGNStr(PGNTags.BlackUSCFTextField),
<           "Annotator":PGNStr(PGNTags.AnnotatorTextField), "ECO":PGNStr(PGNTags.ECOTextField),
<           "DateGamePlayed":PGNTags.GetDateGamePlayed()]}
---
>               PGNStr(PGNTags.BlackNameTextField), "SiteLocation":PGNStr(PGNTags
>               .SiteLocationTextField), "EventTitle":PGNStr(PGNTags.EventTitleTextField),
>               "RoundNumber":PGNStr(PGNTags.RoundNumberTextField), "TimeControl":PGNStr(PGNTags
>               .TimeControlTextField), "Opening":PGNStr(PGNTags.OpeningTextField), "Section":
>               PGNStr(PGNTags.SectionTextField), "BoardNumber":PGNStr(PGNTags
>               .BoardNumberTextField), "WhiteElo":PGNStr(PGNTags.WhiteEloTextField), "BlackElo":
>               PGNStr(PGNTags.BlackEloTextField), "WhiteUSCF":PGNStr(PGNTags.WhiteUSCFTextField),
>               "BlackUSCF":PGNStr(PGNTags.BlackUSCFTextField), "Annotator":PGNStr(PGNTags
>               .AnnotatorTextField), "ECO":PGNStr(PGNTags.ECOTextField), "DateGamePlayed":
>               PGNTags.GetDateGamePlayed()]}
5196,5200c6349,6353
<           GetTag("RoundNumber") + GetTag("WhiteName") + GetTag("BlackName") + GetTag("EndResult")
<           + GetTag("TimeControl") + GetTag("WhiteElo") + GetTag("BlackElo") + GetTag("WhiteUSCF")
<           + GetTag("BlackUSCF") + GetTag("Annotator") + GetTag("ECO") + GetTag("Opening") +
<           GetTag("Section") + GetTag("BoardNumber") + "[SetUp \"1\"]\n" + "[FEN \"" + FENString()
<           + "\"]\n\n"
---
>           GetTag("RoundNumber") + GetTag("WhiteName") + GetTag("BlackName") +
>           GetTag("EndResult") + GetTag("TimeControl") + GetTag("WhiteElo") +
>           GetTag("BlackElo") + GetTag("WhiteUSCF") + GetTag("BlackUSCF") + GetTag("Annotator") +
>           GetTag("ECO") + GetTag("Opening") + GetTag("Section") + GetTag("BoardNumber") +
>           "[SetUp \"1\"]\n" + "[FEN \"" + FENString() + "\"]\n\n"
5206,5211c6359,6371
<       // return the game information list from the pgn tags //
<       return AppTag("EventTitle") + AppTag("SiteLocation") + AppTag("DateGamePlayed") +
<              AppTag("RoundNumber") + AppTag("WhiteName") + AppTag("BlackName") + AppTag("EndResult")
<              + AppTag("TimeControl") + AppTag("WhiteElo") + AppTag("BlackElo") + AppTag("WhiteUSCF")
<              + AppTag("BlackUSCF") + AppTag("Annotator") + AppTag("ECO") + AppTag("Opening") +
<              AppTag("Section") + AppTag("BoardNumber")
---
>       // clear the game info list //
>       InfoList.removeAll()
>
>       // create the game information list from the pgn tags //
>       InfoList = AppTag("EventTitle") + AppTag("SiteLocation") + AppTag("DateGamePlayed") +
>                  AppTag("RoundNumber") + AppTag("WhiteName") + AppTag("BlackName") +
>                  AppTag("EndResult") + AppTag("TimeControl") + AppTag("WhiteElo") +
>                  AppTag("BlackElo") + AppTag("WhiteUSCF") + AppTag("BlackUSCF") +
>                  AppTag("Annotator") + AppTag("ECO") + AppTag("Opening") +  AppTag("Section") +
>                  AppTag("BoardNumber")
>
>       // return the game information list //
>       return InfoList
5232a6393,6395
>       // initialize the position image //
>       PositionImage = NSImage()
>
5258c6421
<   func Initialize(_ Suffix:String) -> Bool
---
>   func Initialize(_ Suffix:String)
5267,5268c6430,6435
<       // return if the move list string is empty and is the default fen layout
<       return Inf.VSIsEmpty("MoveListString") && Inf.VSIsDefault("FENLayout")
---
>       // if the suffix is not empty //
>       if !Suffix.isEmpty
>       {
>           // retrieve the position image //
>           PositionImage = Prefs.PreferenceImage("PositionImage" + Suffix, Default:NSImage())
>       }
5278a6446,6448
>
>       // if the suffix is not empty, save the position image to preferences //
>       if !Suffix.isEmpty {Prefs.Save(PositionImage, Key:"PositionImage" + Suffix)}
5284a6455,6457
>       // the half width of a square or circle //
>       var HalfWidth:CGFloat = 0
>
5321a6495,6497
>       // set the half width of the square //
>       HalfWidth = Width * 0.5
>
5326c6502
<       Context.translateBy(x:Width * 0.5, y:Width * 0.5)
---
>
>       Context.translateBy(x:HalfWidth, y:HalfWidth)
5332,5333c6508,6509
<       Rectangle = CGRect(x:-Width * 0.5 * Fraction, y:-Width * 0.5 * Fraction, width:Width *
<                          Fraction, height:Width * Fraction)
---
>       Rectangle = CGRect(x:-HalfWidth * Fraction, y:-HalfWidth * Fraction, width:Width * Fraction,
>                          height:Width * Fraction)
```