```
// Mario Kart 64 Course Info Version 0.1 (4 protothreads)
// ----------------------------------------------------------
// Jason Bunk
// ----------------------------------------------------------
// 3rd graphics function (1)
// ----------------------------------------------------------
// ---- 3rd set of object models (1)
// ----------------------------------------------------------
// ---- 3rd set of texture objects (1)
// ----------------------------------------------------------
// ---- 3rd set of texture offsets (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex coordinates (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex connections/edges/polygons, colors, etc. (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex normals (1)
// ----------------------------------------------------------
// ---- 3rd set of texture offsets (1) (which first to be animated)
// ----------------------------------------------------------
// ---- 3rd set of texture objects (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex coordinates (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex connections/edges/polygons, colors, etc. (1)
// ----------------------------------------------------------
// ---- 3rd set of vertex normals (1)
// ----------------------------------------------------------
// ---- 3rd set of texture offsets (1
```

```
function Get_Number_Of_Dimensions
  (Dimension : Dimension_Type)
  return Dimension_Count;
pragma Inline(Get_Number_Of_Dimensions);

function Get_Length
  (Dimension : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Length);

function Get_Stride
  (Dimension : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Stride);

function Get_First_Index
  (Dimension : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_First_Index);

function Get_Last_Index
  (Dimension : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Last_Index);

function Get_Position
  (Dimension : Dimension_Type;
   Index : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Position);

function Get_Index
  (Dimension : Dimension_Type;
   Position : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Index);

function Get_Offset
  (Dimension : Dimension_Type;
   Index : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Offset);

function Get_Index
  (Dimension : Dimension_Type;
   Offset : Dimension_Type;
   Number : Dimension_Count)
  return Dimension_Type;
pragma Inline(Get_Index
```

```
// MARK: - Extensions
// the default fen string //
var DefaultFENLayout:String = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR"

// the shared user default preferences //
var AppPreferencesUserDefaults = UserDefaults.standard

// image data //
var ImageDataCache = Data()

// a position image //
var PositionImage:UIImage = UIImage()

// the move list string for the game //
var MoveListString:String = ""

// the annotation list string for the game //
var AnnotationListString:String = ""

// the fen layout string to use for the position editor //
var FENLayout:String = ""

// the starting move number for the game from the fen string //
var FENStartMoveNumber:String = ""

// the fen color to move string to use for the position editor //
var FENColor:String = ""

// the fen castling rights to use for the position editor //
var FENCastlingRights:String = ""

// the fen enpassant target square to use for the position editor //
var FENEnpassantSquare:String = ""

// the name of the person playing as white //
var WhiteName:String = ""

// the name of the person playing as black //
var BlackName:String = ""

// the location where the game was played //
var EvtLocation:String = ""

// the title of the tournament event //
var EventTitle:String = ""

// the round number of the game in the tournament //
var RoundNumber:String = ""

// the end result of the game //
var EndResult:String = ""

// the date the game was played on //
var DateGamePlayed:String = ""

// the time control used for the game //
var TimeControl:String = ""

// the opening for the game //
var Opening:String = ""

// the section for the game //
var Section:String = ""

// the board number for the game //
var BoardNumber:String = ""

// the white elo for the game //
var WhiteElo:String = ""

// the black elo for the game //
var BlackElo:String = ""

// the white uscf for the game //
var WhiteUSCF:String = ""

// the black uscf for the game //
var BlackUSCF:String = ""

// the annotator for the game //
var Annotator:String = ""

// the eco for the game //
var ECO:String = ""

// initialize to empty attributes //
func Initialize()
{
    // initialize the move list string //
    MoveListString = ""

    // initialize the annotation list string //
    AnnotationListString = ""

    // initialize the fen layout string //
    FENLayout = DefaultFENLayout

    // initialize the fen color string //
    FENColor = "w"

    // initialize the fen castling rights string //
    FENCastlingRights = "KQkq"

    // initialize the fen enpassant square string //
    FENEnpassantSquare = "-"

    // initialize the fen start move number string //
    FENStartMoveNumber = "1"

    // initialize name of white player string //
    WhiteName = ""

    // initialize name of black player string //
    BlackName = ""

    // initialize site location string //
    EvtLocation = ""

    // initialize event title string //
    EventTitle = ""

    // initialize round number string //
    RoundNumber = ""

    // initialize end result string //
    EndResult = ""

    // initialize date game played string //
    DateGamePlayed = ""

    // initialize time control string //
    TimeControl = ""

    // initialize the opening string //
    Opening = ""

    // initialize the section string //
    Section = ""

    // initialize the board number string //
    BoardNumber = ""

    // initialize white elo string //
    WhiteElo = ""

    // initialize black elo string //
    BlackElo = ""

    // initialize white uscf string //
    WhiteUSCF = ""

    // initialize black uscf string //
    BlackUSCF = ""

    // initialize annotator string //
    Annotator = ""

    // initialize eco string //
    ECO = ""

    // initialize the position image //
    PositionImage = UIImage()
}

// initialize the game attributes for the given tag //
func Initialize(_ Tag:String)
{
    // retrieve the move list string if was stored //
    MoveListString = PreferenceString("MoveListString" + Tag, Default:"")

    // retrieve the annotation list string if was stored //
    AnnotationListString = PreferenceString("AnnotationListString" + Tag, Default:"")

    // retrieve the fen layout if was stored //
    FENLayout = PreferenceString("FENLayout" + Tag, Default:(defaultFENLayout)

    // retrieve the fen color if was stored //
    FENColor = PreferenceString("FENColor" + Tag, Default:"w")

    // retrieve the fen castling rights if was stored //
    FENCastlingRights = PreferenceString("FENCastlingRights" + Tag, Default:"KQkq")

    // retrieve the fen enpassant square if was stored //
    FENEnpassantSquare = PreferenceString("FENEnpassantSquare" + Tag, Default:"-")

    // retrieve the fen start move number if was saved //
    FENStartMoveNumber = PreferenceString("FENStartMoveNumber" + Tag, Default:"1")

    // retrieve name of white player if was saved //
    WhiteName = PreferenceString("WhiteName" + Tag, Default:"")

    // retrieve name of black player if was saved //
    BlackName = PreferenceString("BlackName" + Tag, Default:"")

    // retrieve site location if was saved //
    EvtLocation = PreferenceString("EvtLocation" + Tag, Default:"")

    // retrieve event title if was saved //
    EventTitle = PreferenceString("EventTitle" + Tag, Default:"")

    // retrieve round number if was saved //
    RoundNumber = PreferenceString("RoundNumber" + Tag, Default:"")

    // retrieve end result if was saved //
    EndResult = PreferenceString("EndResult" + Tag, Default:"")

    // retrieve date game played if was saved //
    DateGamePlayed = PreferenceString("DateGamePlayed" + Tag, Default:"")

    // retrieve time control if was saved //
    TimeControl = PreferenceString("TimeControl" + Tag, Default:"")

    // retrieve the opening if was saved //
    Opening = PreferenceString("Opening" + Tag, Default:"")

    // retrieve the section if was saved //
    Section = PreferenceString("Section" + Tag, Default:"")

    // retrieve the board number if was saved //
    BoardNumber = PreferenceString("BoardNumber" + Tag, Default:"")

    // retrieve white elo if was saved //
    WhiteElo = PreferenceString("WhiteElo" + Tag, Default:"")

    // retrieve black elo if was saved //
    BlackElo = PreferenceString("BlackElo" + Tag, Default:"")

    // retrieve white uscf if was saved //
    WhiteUSCF = PreferenceString("WhiteUSCF" + Tag, Default:"")

    // retrieve black uscf if was saved //
    BlackUSCF = PreferenceString("BlackUSCF" + Tag, Default:"")

    // retrieve annotator if was saved //
    Annotator = PreferenceString("Annotator" + Tag, Default:"")

    // retrieve eco if was saved //
    ECO = PreferenceString("ECO" + Tag, Default:"")

    // if the tag is not empty //
    if !Tag.isEmpty
    {
        // retrieve the position image //
        PositionImage = PreferenceImage("PositionImage" + Tag, Default:UIImage())
    }
}

// store the autosave data //
func StoreAutosaveData(_ Tag:String)
{
    // store the move list to preferences //
    AppPreferencesSave.setMoveListString(_forKey:"MoveListString" + Tag)

    // store the annotation list to preferences //
    AppPreferencesSave.setAnnotationListString(_forKey:"AnnotationListString" + Tag)

    // store the fen layout to preferences //
    AppPreferencesSave.setFENLayout(_forKey:"FENLayout" + Tag)

    // store the fen color to preferences //
    AppPreferencesSave.setFENColor(_forKey:"FENColor" + Tag)

    // store the fen castling rights to preferences //
    AppPreferencesSave.setFENCastlingRights(_forKey:"FENCastlingRights" + Tag)

    // store the fen enpassant square to preferences //
    AppPreferencesSave.setFENEnpassantSquare(_forKey:"FENEnpassantSquare" + Tag)

    // store the fen start move number to preferences //
    AppPreferencesSave.setFENStartMoveNumber(_forKey:"FENStartMoveNumber" + Tag)

    // store the white player name to preferences //
    AppPreferencesSave.setWhiteName(_forKey:"WhiteName" + Tag)

    // store the black player name to preferences //
    AppPreferencesSave.setBlackName(_forKey:"BlackName" + Tag)

    // store the site location to preferences //
    AppPreferencesSave.setEvtLocation(_forKey:"EvtLocation" + Tag)

    // store the event title to preferences //
    AppPreferencesSave.setEventTitle(_forKey:"EventTitle" + Tag)

    // store the round number to preferences //
    AppPreferencesSave.setRoundNumber(_forKey:"RoundNumber" + Tag)

    // store the end result to preferences //
    AppPreferencesSave.setEndResult(_forKey:"EndResult" + Tag)

    // store the date game played to preferences //
    AppPreferencesSave.setDateGamePlayed(_forKey:"DateGamePlayed" + Tag)

    // store the time control to preferences //
    AppPreferencesSave.setTimeControl(_forKey:"TimeControl" + Tag)

    // store the opening to preferences //
    AppPreferencesSave.setOpening(_forKey:"Opening" + Tag)

    // store the section to preferences //
    AppPreferencesSave.setSection(_forKey:"Section" + Tag)

    // store the board number to preferences //
    AppPreferencesSave.setBoardNumber(_forKey:"BoardNumber" + Tag)

    // store the white elo to preferences //
    AppPreferencesSave.setWhiteElo(_forKey:"WhiteElo" + Tag)

    // store the black elo to preferences //
    AppPreferencesSave.setBlackElo(_forKey:"BlackElo" + Tag)

    // store the white uscf to preferences //
    AppPreferencesSave.setWhiteUSCF(_forKey:"WhiteUSCF" + Tag)

    // store the black uscf to preferences //
    AppPreferencesSave.setBlackUSCF(_forKey:"BlackUSCF" + Tag)

    // store the annotator to preferences //
    AppPreferencesSave.setAnnotator(_forKey:"Annotator" + Tag)

    // store the eco to preferences //
    AppPreferencesSave.setECO(_forKey:"ECO" + Tag)

    // if the tag is not empty //
    if !Tag.isEmpty
    {
        // save the position image to preferences //
        AppPreferencesSave.setNSKeyedArchiver.archivedDataWithRootObject:(PositionImage)_forKey:
                                    "PositionImage" + Tag)
    }

}

// get the user preference for the specified key //
func PreferenceImage(_ Key:String, Default:UIImage) -> UIImage
{
    // if the value for the key already exists //
    if AppPreferences.boolForKey("Exists")
    {
        // if the data to unarchive exists //
        if NSKeyedUnarchiver.unarchiveObjectWithAppPreferences.object(forKey:Key) as! Data) !=
                                    nil
        {
            // return the saved image //
            return NSKeyedUnarchiver.unarchiveObject(with:AppPreferences.object(forKey:Key) as!
                                    Data) as! UIImage
        }
        // 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 //
        AppPreferences.setTrue(_forKey:Key + "Exists")

        // get the image data for the default image //
        ImageData = NSKeyedArchiver.archivedDataWithRootObject:(Default)

        // save the user preference for the image key //
        AppPreferences.setImageData(_forKey:Key)

        // return the default image //
        return Default
    }
}

// get the user saved preference for this key //
func PreferenceString(_ Key:String, Default:String) -> String
{
    // if the user saved preference exists //
    if AppPreferences.boolForKey(Key + "Exists")
    {
        // return the user saved preference for this key //
        return AppPreferences.string(forKey:Key)
    }
    // the user saved preference does not yet exist //
    else
    {
        // set that the user preference now exists for this key //
        AppPreferences.setTrue(_forKey:Key + "Exists")

        // set the default value for this key //
        AppPreferences.setDefault(_forKey:Key)

        // return the default value for this key //
        return Default
    }
}
}

// the core graphics class //
class CoreGraphics
{
    // the core graphics context //
    var Context:CGContext!

    // the radius of a circle //
    var Radius:CGFloat = 0

    // the diameter of a circle //
    var Diameter:CGFloat = 0

    // the width of a square //
    var Width:CGFloat = 0

    // the half width of a square //
    var HalfWidth:CGFloat = 0

    // create the context //
    func CreateContext(_ Width:CGFloat, Height:CGFloat, Scale:CGFloat) -> CGContext
    {
        // return a context for the given width, height, and scale //
        return CGContext(data:nil, width:Int(Width * Scale), height:Int(Height * Scale),
                         bitsPerComponent:8, bytesPerRow:0, space:CGColorSpaceCreateDeviceRGB(),
                         bitmapInfo:CGImageAlphaInfo.premultiplied1stFirst.rawValue)
    }

    // create the layer //
    func CreateLayer(_ Width:CGFloat, Height:CGFloat, Scale:CGFloat) -> CGLayer
    {
        // return a layer for the given width, height, and scale //
        return CGLayer(CreateContext(Width, Height:Height, Scale:Scale), size:CGSize(width:Width *
                       Scale, height:Height * Scale), auxiliaryInfo:nil)
    }

    // get a cgimage from an uiimage //
    func UIImageToCGImage(_ Image:UIImage, Size:CGFloat, Scale:CGFloat) -> CGImage
    {
        // create the context for the given image, size, and scale
        Context = CreateContext(Size, Height:Size, Scale:Scale)

        // draw the image into the context //
        Context.drawImage(CGRectMake(0, 0, CGFloat(ceilf(Float(Size))), CGFloat(ceilf(Float(Size)))),
                          image:Image.cgImage!, isMirrored:false)

        // return the cgimage //
        return Context.makeImage()
    }

    // create a square cgimage //
    func CreateSquare(_ Size:CGFloat, Fraction:CGFloat, Scale:CGFloat, Color:NSColor) -> CGImage
    {
        // set the width of the square //
        Width = Size * Scale

        // set the half width of the square //
        HalfWidth = Width * 0.5

        // create the context from the given width, height, and scale //
        Context = CreateContext(Width, Height:Width, Scale:1)

        // translate the context origin to the center of the context //
        Context.translateBy(x:HalfWidth, y:HalfWidth)

        // set the fill color for the square //
        Context.setFillColor(Color.cgColor)

        // fill the rectangle with the fill color //
        Context.fillRect(CGRectMake(-HalfWidth * Fraction, y:-HalfWidth * Fraction, width:Width * Fraction,
                         height:Width * Fraction))

        // return a cgimage from the context //
        return Context.makeImage()
    }

    // create a circle cgimage //
    func CreateCircle(_ Size:CGFloat, Fraction:CGFloat, Scale:CGFloat, Color:NSColor) -> CGImage
    {
        // set the diameter of the circle //
        Diameter = Size * Scale

        // set the radius of the circle //
        Radius = Diameter * 0.5

        // create the context from the given diameter and scale //
        Context = CreateContext(Diameter, Height:(Diameter), Scale:1)

        // translate the context origin to the center of the context //
        Context.translateBy(x:Radius, y:Radius)

        // set the fill color //
        Context.setFillColor(Color.cgColor)

        // fill in the ellipse with the given fill color //
        Context.fillEllipseIn(CGRectMake(-Radius * Fraction, y:-Radius * Fraction, width:Diameter *
                              Fraction, height:Diameter * Fraction))

        // return a cgimage from the context //
        return Context.makeImage()
    }

    // convert a cgimage to an uiimage //
    func CGImageToUIImage(_ Image:CGImage, Size:CGFloat) -> UIImage
    {
        // return an uiimage from a cgimage from the given image and size //
        return UIImage(cgImage:Image, size:CGSize(width:Size, height:Size))
    }

    // convert a context to an uiimage //
    func ContextToUIImage(_ ImageContext:CGContext, Size:CGFloat) -> UIImage
    {
        // return an uiimage from a context //
        return UIImage(cgImage:ImageContext.makeImage(), size:CGSize(width:Size, height:Size))
    }
}

// the marker pens class //
class MarkerPens
{
    // the draw colors for the markers //
    var DrawColorsNSColor = [NSColor.red, NSColor.green, NSColor.blue, NSColor.white,
                             NSColor.black, NSColor.cyan, NSColor.magenta, NSColor.yellow]

    // the marker cursor names //
    var MarkerNames:[String] = ["marker_red_icon", "marker_green_icon", "marker_blue_icon",
                                "marker_white_icon", "marker_black_icon", "marker_cyan_icon",
                                "marker_magenta_icon", "marker_yellow_icon"]

    // diagram cursor for the given marker color //
    func Cursor(_ ColorIndx:Int) -> NSCursor
    {
        // return the diagram cursor for the given marker color //
        return NSCursor.init(image:NSImage(named:MarkerNames[ColorIndx]), hotSpot:CGPoint(x:2, y:30))
    }

    // diagram color for the given marker color //
    func Color(_ ColorIndx:Int) -> NSColor
    {
        // return the diagram cursor for the given marker color //
        return DrawColors[ColorIndx].cgColor
    }
```