```
// Copyright(C) 2025 Gordon Klok
import Sys;
import Sys.Math;
import Sys.Draw;

// the list of capture keysms
var KeyCaptureSetup[keysms, keycount];

// the list of keysms keycount
var KeyCaptureSetup[keysms, keycount] = [];

// the layout ...
var KeyCaptureSetup[layout] = [screenwidth/2, screenheight, screen.h];

// the colours space for a delta value to be applied
var ColoursSpace[colours, count];
// the colours space for a delta value to be applied
var ColoursSpace[colours] = [];
var ColoursSpace[count] = 0;
var ColoursSpace.Add(colour, delta) = ThisObject.ColoursSpace.Insert(ThisObject.ColoursSpace.Count(), colour, delta);

// the ball
var Ball[x, dx, y, dy, r, colour, ...];
var Ball[x] = screenwidth/2;
var Ball[dx] = screenwidth/200 * Sys.Math.Random(-1, 1);
var Ball[y] = screenheight/2;
var Ball[dy] = screenheight/200 * Sys.Math.Random(-1, 1);
var Ball[r] = screenwidth/100;
var Ball[colour] = ColoursSpace[Sys.Math.Random(0, ThisObject.ColoursSpace.Count() - 1)];

// the paddle
var Paddle[x, dx, y, dy, w, h, colour, ...];
var Paddle[x] = screenwidth/2;
var Paddle[dx] = 0;
var Paddle[y] = screenheight - screenheight/20;
var Paddle[dy] = 0;
var Paddle[w] = screenwidth/10;
var Paddle[h] = screenheight/40;
var Paddle[colour] = ColoursSpace[Sys.Math.Random(0, ThisObject.ColoursSpace.Count() - 1)];

// the paddle speed
var PaddleSpeed = screenwidth/100;

// the keys
var Keys[keyleft, keyright, ...];
var Keys[keyleft] = 0;
var Keys[keyright] = 0;

// the keys state
var KeyStates[keyleft, keyright, ...];
var KeyStates[keyleft] = 0;
var KeyStates[keyright] = 0;

// the keys pressed
var KeyPressed[keyleft, keyright, ...];
var KeyPressed[keyleft] = 0;
var KeyPressed[keyright] = 0;

// the keys released
var KeyReleased[keyleft, keyright, ...];
var KeyReleased[keyleft] = 0;
var KeyReleased[keyright] = 0;

// the keys down
var KeyDown[keyleft, keyright, ...];
var KeyDown[keyleft] = 0;
var KeyDown[keyright] = 0;

// the keys up
var KeyUp[keyleft, keyright, ...];
var KeyUp[keyleft] = 0;
var KeyUp[keyright] = 0;

// the keys value
var KeyValue[keyleft, keyright, ...];
var KeyValue[keyleft] = 0;
var KeyValue[keyright] = 0;

// the keys value of a delta
var KeyDelta[keyleft, keyright, ...];
var KeyDelta[keyleft] = 0;
var KeyDelta[keyright] = 0;

// the keys value of a
var KeyA[keyleft, keyright, ...];
var KeyA[keyleft] = 0;
var KeyA[keyright] = 0;

// the keys value of b
var KeyB[keyleft, keyright, ...];
var KeyB[keyleft] = 0;
var KeyB[keyright] = 0;

// the keys value of c
var KeyC[keyleft, keyright, ...];
var KeyC[keyleft] = 0;
var KeyC[keyright] = 0;

// the keys value of d
var KeyD[keyleft, keyright, ...];
var KeyD[keyleft] = 0;
var KeyD[keyright] = 0;

// the keys value of e
var KeyE[keyleft, keyright, ...];
var KeyE[keyleft] = 0;
var KeyE[keyright] = 0;

// the keys value of f
var KeyF[keyleft, keyright, ...];
var KeyF[keyleft] = 0;
var KeyF[keyright] = 0;

// the keys value of g
var KeyG[keyleft, keyright, ...];
var KeyG[keyleft] = 0;
var KeyG[keyright] = 0;

// the keys value of h
var KeyH[keyleft, keyright, ...];
var KeyH[keyleft] = 0;
var KeyH[keyright] = 0;

// the keys value of i
var KeyI[keyleft, keyright, ...];
var KeyI[keyleft] = 0;
var KeyI[keyright] = 0;

// the keys value of j
var KeyJ[keyleft, keyright, ...];
var KeyJ[keyleft] = 0;
var KeyJ[keyright] = 0;

// the keys value of k
var KeyK[keyleft, keyright, ...];
var KeyK[keyleft] = 0;
var KeyK[keyright] = 0;

// the keys value of l
var KeyL[keyleft, keyright, ...];
var KeyL[keyleft] = 0;
var KeyL[keyright] = 0;

// the keys value of m
var KeyM[keyleft, keyright, ...];
var KeyM[keyleft] = 0;
var KeyM[keyright] = 0;

// the keys value of n
var KeyN[keyleft, keyright, ...];
var KeyN[keyleft] = 0;
var KeyN[keyright] = 0;

// the keys value of o
var KeyO[keyleft, keyright, ...];
var KeyO[keyleft] = 0;
var KeyO[keyright] = 0;

// the keys value of p
var KeyP[keyleft, keyright, ...];
var KeyP[keyleft] = 0;
var KeyP[keyright] = 0;

// the keys value of q
var KeyQ[keyleft, keyright, ...];
var KeyQ[keyleft] = 0;
var KeyQ[keyright] = 0;

// the keys value of r
var KeyR[keyleft, keyright, ...];
var KeyR[keyleft] = 0;
var KeyR[keyright] = 0;

// the keys value of s
var KeyS[keyleft, keyright, ...];
var KeyS[keyleft] = 0;
var KeyS[keyright] = 0;

// the keys value of t
var KeyT[keyleft, keyright, ...];
var KeyT[keyleft] = 0;
var KeyT[keyright] = 0;

// the keys value of u
var KeyU[keyleft, keyright, ...];
var KeyU[keyleft] = 0;
var KeyU[keyright] = 0;

// the keys value of v
var KeyV[keyleft, keyright, ...];
var KeyV[keyleft] = 0;
var KeyV[keyright] = 0;

// the keys value of w
var KeyW[keyleft, keyright, ...];
var KeyW[keyleft] = 0;
var KeyW[keyright] = 0;

// the keys value of x
var KeyX[keyleft, keyright, ...];
var KeyX[keyleft] = 0;
var KeyX[keyright] = 0;

// the keys value of y
var KeyY[keyleft, keyright, ...];
var KeyY[keyleft] = 0;
var KeyY[keyright] = 0;

// the keys value of z
var KeyZ[keyleft, keyright, ...];
var KeyZ[keyleft] = 0;
var KeyZ[keyright] = 0;

// the keys value of space
var KeySpace[keyleft, keyright, ...];
var KeySpace[keyleft] = 0;
var KeySpace[keyright] = 0;

// the keys value of enter
var KeyEnter[keyleft, keyright, ...];
var KeyEnter[keyleft] = 0;
var KeyEnter[keyright] = 0;

// the keys value of tab
var KeyTab[keyleft, keyright, ...];
var KeyTab[keyleft] = 0;
var KeyTab[keyright] = 0;

// the keys value of esc
var KeyEsc[keyleft, keyright, ...];
var KeyEsc[keyleft] = 0;
var KeyEsc[keyright] = 0;

// the keys value of backspace
var KeyBackspace[keyleft, keyright, ...];
var KeyBackspace[keyleft] = 0;
var KeyBackspace[keyright] = 0;

// the keys value of delete
var KeyDelete[keyleft, keyright, ...];
var KeyDelete[keyleft] = 0;
var KeyDelete[keyright] = 0;

// the keys value of insert
var KeyInsert[keyleft, keyright, ...];
var KeyInsert[keyleft] = 0;
var KeyInsert[keyright] = 0;

// the keys value of home
var KeyHome[keyleft, keyright, ...];
var KeyHome[keyleft] = 0;
var KeyHome[keyright] = 0;

// the keys value of end
var KeyEnd[keyleft, keyright, ...];
var KeyEnd[keyleft] = 0;
var KeyEnd[keyright] = 0;

// the keys value of pgup
var KeyPgUp[keyleft, keyright, ...];
var KeyPgUp[keyleft] = 0;
var KeyPgUp[keyright] = 0;

// the keys value of pgdn
var KeyPgDn[keyleft, keyright, ...];
var KeyPgDn[keyleft] = 0;
var KeyPgDn[keyright] = 0;

// the keys value of up
var KeyUp[keyleft, keyright, ...];
var KeyUp[keyleft] = 0;
var KeyUp[keyright] = 0;

// the keys value of down
var KeyDown[keyleft, keyright, ...];
var KeyDown[keyleft] = 0;
var KeyDown[keyright] = 0;

// the keys value of left
var KeyLeft[keyleft, keyright, ...];
var KeyLeft[keyleft] = 0;
var KeyLeft[keyright] = 0;

// the keys value of right
var KeyRight[keyleft, keyright, ...];
var KeyRight[keyleft] = 0;
var KeyRight[keyright] = 0;

// the keys value of f1
var KeyF1[keyleft, keyright, ...];
var KeyF1[keyleft] = 0;
var KeyF1[keyright] = 0;

// the keys value of f2
var KeyF2[keyleft, keyright, ...];
var KeyF2[keyleft] = 0;
var KeyF2[keyright] = 0;

// the keys value of f3
var KeyF3[keyleft, keyright, ...];
var KeyF3[keyleft] = 0;
var KeyF3[keyright] = 0;

// the keys value of
```

```
// the number of frames in the entity
int numFrames(int ent);

// the number of joints in the entity
int numJoints(int ent);

// the current frame position
int getCurrentFrame(int ent);

// the time of the current frame
int getCurrentFrameTime(int ent);

// the current animation
int getCurrentAnim(int ent);

// the frame rate
int getFrameRate(int ent);

// the time of one frame
int getFrameTime(int ent);

// the bone position
void getBonePosition( int ent, int bone, int frame, int time, vec3_t offset, vec3_t axis[3] );

// the bone position relative to parent
void getBoneAngles( int ent, int bone, int frame, int time, vec3_t angles );

// the bounding rectangle for a frame
void getBoneBounds( int ent, int bone, int frame, int time, float scale, vec3_t mins, vec3_t maxs );

// the tag position and orientation
int getTag( int ent, int frame, int tagindex, int time, vec3_t offset, vec3_t axis[3] );

// the number of tags
int numTags( int ent );

// the name of a tag
const char *getTagName( int ent, int tagindex );

// find a tag by name
int getTagNum( int ent, const char *name );

// the number of surfaces
int numSurfaces( int ent );

// the name of a surface
const char *getSurfaceName( int ent, int surfaceindex );

// the number of shaders on a surface
int getNumShaders( int ent, int surfaceindex );

// the name of a shader on a surface
const char *getShaderName( int ent, int surfaceindex, int shaderindex );

// the hit location on the model
int getHitLocation( int ent, int frame, int time, vec3_t start, vec3_t end );

// the current position in the world
void getPosition( int ent, int time, vec3_t offset );

// the current velocity
void getVelocity( int ent, int time, vec3_t velocity );

// the current angles
void getAngles( int ent, int time, vec3_t angles );

// the current bounding box
void getBoundingBox( int ent, int time, int frame, vec3_t mins, vec3_t maxs );

// the current origin
void getOrigin( int ent, int time, int frame, int bone, vec3_t offset, vec3_t axis[3] );

// the current axis
void getAxis( int ent, int time, int frame, int bone, vec3_t axis[3] );

// the width of the model
int getWidth( int ent, int time, int frame );

// the height of the model
int getHeight( int ent, int time, int frame );

// the depth of the model
int getDepth( int ent, int time, int frame );

// the current animation
int getCurrentAnim( int ent );

// the current frame
int getCurrentFrame( int ent );

// the time of the current frame
int getCurrentFrameTime( int ent );

// the current time
int getTime( int ent );

// the current origin
void getOrigin( int ent, int time, vec3_t offset );

// the current angles
void getAngles( int ent, int time, vec3_t angles );

// the current velocity
void getVelocity( int ent, int time, vec3_t velocity );

// the current bounding box
void getBoundingBox( int ent, int time, int frame, vec3_t mins, vec3_t maxs );

// the current position
void getPosition( int ent, int time, vec3_t offset );

// the current axis
void getAxis( int ent, int time, int frame, int bone, vec3_t axis[3] );

// the current frame
int getCurrentFrame( int ent );

// the current frame time
int getCurrentFrameTime( int ent );

// the current animation
int getCurrentAnim( int ent );

// the current time
int getTime( int ent );

// the current origin
void getOrigin( int ent, int time, int frame, int bone, vec3_t offset, vec3_t axis[3]
```

```
// a position image //
var PositionImage = NSImage()

// 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 WhiteNameString = ""

// the name of the person playing as black //
var BlackNameString = ""

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

// the title of the tournament event //
var EventTitleString = ""

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

// the end result of the game //
var EndResultString = ""

// the date the game was played on //
var DateGamePlayedString = ""

// the time control used for the game //
var TimeControlString = ""

// the opening for the game //
var OpeningString = ""

// the section for the game //
var SectionString = ""

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

// the white elo for the game //
var WhiteEloString = ""

// the black elo for the game //
var BlackEloString = ""

// the white uscf for the game //
var WhiteUSCFString = ""

// the black uscf for the game //
var BlackUSCFString = ""

// the annotator for the game //
var AnnotatorString = ""

// the eco for the game //
var ECOString = ""

// 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 //
    SiteLocation = ""

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

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

    // initialize and 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 = NSImage()
}

// 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 //
    SiteLocation = PreferenceString("SiteLocation" + 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 and 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:"")

    // retrieve the position image //
    PositionImage = PreferenceImage("PositionImage" + Tag, Default:NSImage())
}

// store the autosave data //
func StoreAutosaveData(_ Tag:String)
{
    // store the move list to preferences //
    AppPreference.set(MoveListString, forKey:"MoveListString" + Tag)

    // store the annotation list to preferences //
    AppPreference.set(AnnotationListString, forKey:"AnnotationListString" + Tag)

    // store the fen layout to preferences //
    AppPreference.set(FENLayout, forKey:"FENLayout" + Tag)

    // store the fen color to preferences //
    AppPreference.set(FENColor, forKey:"FENColor" + Tag)

    // store the fen castling rights to preferences //
    AppPreference.set(FENCastlingRights, forKey:"FENCastlingRights" + Tag)

    // store the fen enpassant square to preferences //
    AppPreference.set(FENEnpassantSquare, forKey:"FENEnpassantSquare" + Tag)

    // store the fen start move number to preferences //
    AppPreference.set(FENStartMoveNumber, forKey:"FENStartMoveNumber" + Tag)

    // store the white player name to preferences //
    AppPreference.set(WhiteName, forKey:"WhiteName" + Tag)

    // store the black player name to preferences //
    AppPreference.set(BlackName, forKey:"BlackName" + Tag)

    // store the site location to preferences //
    AppPreference.set(SiteLocation, forKey:"SiteLocation" + Tag)

    // store the event title to preferences //
    AppPreference.set(EventTitle, forKey:"EventTitle" + Tag)

    // store the round number to preferences //
    AppPreference.set(RoundNumber, forKey:"RoundNumber" + Tag)

    // store the end result to preferences //
    AppPreference.set(EndResult, forKey:"EndResult" + Tag)

    // store the date game played to preferences //
    AppPreference.set(DateGamePlayed, forKey:"DateGamePlayed" + Tag)

    // store the time control to preferences //
    AppPreference.set(TimeControl, forKey:"TimeControl" + Tag)

    // store the opening to preferences //
    AppPreference.set(Opening, forKey:"Opening" + Tag)

    // store the section to preferences //
    AppPreference.set(Section, forKey:"Section" + Tag)

    // store the board number to preferences //
    AppPreference.set(BoardNumber, forKey:"BoardNumber" + Tag)

    // store the white elo to preferences //
    AppPreference.set(WhiteElo, forKey:"WhiteElo" + Tag)

    // store the black elo to preferences //
    AppPreference.set(BlackElo, forKey:"BlackElo" + Tag)

    // store the white uscf to preferences //
    AppPreference.set(WhiteUSCF, forKey:"WhiteUSCF" + Tag)

    // store the black uscf to preferences //
    AppPreference.set(BlackUSCF, forKey:"BlackUSCF" + Tag)

    // store the annotator to preferences //
    AppPreference.set(Annotator, forKey:"Annotator" + Tag)

    // store the eco to preferences //
    AppPreference.set(ECO, forKey:"ECO" + Tag)

    // save the position image to preferences //
    AppPreference.set(NSKeyedArchiver.archivedData(withRootObject:PositionImage), forKey:
                      "PositionImage" + Tag)
}

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

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

        // save the user preference for the image key //
        AppPreference.set(ImageData, 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 AppPreference.bool(forKey:Key + "Exists")
    {
        // return the user saved preference for this key //
        return AppPreference.string(forKey:Key)
    }
    // the user saved preference does not yet exist //
    else
    {
        // set that the user preference now exists for this key //
        AppPreference.set(true, forKey:Key + "Exists")

        // set the default value for this key //
        AppPreference.set(Default, 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.premultiplied\first.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(Context, size:CGSize(width:Width, height:Height), auxiliaryInfo:nil)
                       Scale, height:Height * Scale), auxiliaryInfo:nil)
    }

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

        // draw the image into the context //
        Context.draw(NSImage.cgImage(forProposedRect:nil, context:nil, hints:nil), in:CGRect(x:0, y:0,
                     width:Context.width, height:Context.height), byTiling:false)

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

    // create a square cgimage //
    func CreateSquare(_ Size:CGFloat, Fraction:CGFloat, Scale:CGFloat, ColorHSBColor:->) -> 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(ColorHSBColor:ColorHSBColor())

        // fill the rectangle with the fill color //
        Context.fillRect(rect:CGRect(x:-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, ColorHSBColor:->) -> 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(ColorHSBColor:ColorHSBColor()

        // fill in the ellipse with the given fill color //
        Context.fillEllipse(in:CGRect(x:-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 nsimage //
    func CGImageToNSImage(_ Image:CGImage, Size:CGFloat) -> NSImage
    {
        // return an nsimage from a cgimage from the given image and size //
        return NSImage(cgImage:Image, size:CGSize(width:Size, height:Size))
    }

    // convert a context to an nsimage //
    func ContextToNSImage(_ Context:CGContext, Size:CGFloat) -> NSImage
    {
        // return an nsimage from a cgimage //
        return NSImage(cgImage:Context.makeImage()!, size:CGSize(width:Size, height:Size))
    }
}

// the marker pens class //
class MarkerPens
{
    // the draw colors for the markers //
    var DrawColors:[NSColor] = [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(_ Color:Int) -> NSCursor
    {
        // return the diagram cursor for the given marker color //
        return NSCursor.init(image:NSImage(named:MarkerNames[Color]), hotSpot:CGPoint(x:2, y:30))
    }

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