```swift
//
//  Paths.swift
//  Arm
//
//  Created by Erik Nordlund on 4/10/19.
//  Copyright © 2019 Erik Nordlund. All rights reserved.
//
//  Arm Controller includes the following open-source components:
//      • swiftBluetoothSerial: https://github.com/hoiberg/SwiftBluetoothSerial
//      • peertalk-simple: https://github.com/kirankunigiri/peertalk-simple

import Foundation


enum Height {
    case up
    case down
}

struct Path2D {
    init(pen: Pen, segments: [Line2D], height: Height) {
        self.pen = pen
        self.segments = segments
        self.height = height
        self.estimatedTime = pen.dryTime
    }

    init(asLine: Line2D, pen: Pen) {
        self.pen = pen
        self.segments = [asLine]
        self.height = .down
        self.estimatedTime = pen.dryTime
    }

    func isContinuous(with: Path2D) -> Bool {
        debugPrint("checking path continutity with: ", with)
        if self.height == with.height {
            if self.pen.isEqual(to: with.pen) {

                if let lastPoint = self.segments.last?.pointB {
                    if let nextPoint = with.segments.first?.pointA {
                        if lastPoint.isEqual(to: nextPoint) {
                            debugPrint("paths are continuous")
                            return true
                        }
                    }
                }
            }
        }
        debugPrint("paths are not continuous")
        return false
```

```
    }

    var pen: Pen
    var segments: [Line2D]
    var estimatedTime: Double
    var height: Height
    var overlapsWith: [UnsafeMutablePointer<Path2D>]?// Lines that either
     intersect, or overlap (by pen width). Don't worry about this for now.
}

struct PathSet2D {
    init() {
        self.paths = []
    }
    init(withPaths: [Path2D]) {
        self.paths = withPaths
        // ------------------------------ This is where we will assess
         overlaps
    }

    mutating func append(path: Path2D) {
        self.paths.append(path)

        // ------------------------------ This is where we will assess
         overlaps
    }

    mutating func removeLastPath() {
        if self.paths.count > 0 {
            self.paths.removeLast()
        } else {
            debugPrint("ERROR: no last path to remove.")
        }
    }

    var paths: [Path2D]

    func getEstimatedTimeInSeconds() -> Double {
        var sum = 0.0
        for path in paths {
            sum = sum + path.estimatedTime
        }
        return sum
    }

    private func addGapPaths() -> PathSet2D {
        var lastPointMemory: Point2D? = nil

        var newPathSet = PathSet2D(withPaths: [Path2D]())
```

```
    // adding gap paths to path set
    for path in paths {

        // adding a gap path (height up) if the path is not continuous
         with the last point.
        if let lastPoint = lastPointMemory {
            // if paths are not continuous:
            if !lastPoint.isContinuous(with: path) {
                // create a gap path (with pen up), and add it to the path
                 set.
                if let nextPoint = path.segments.first?.pointA {

                    let gapPathLine = Line2D(pointA: lastPoint, pointB:
                     nextPoint)
                    let gapPath = Path2D(pen: path.pen, segments:
                     [gapPathLine], height: .up)

                    // update memory of last point
                    lastPointMemory = path.segments.last?.pointB

                    // insert gap path before the path
                    newPathSet.paths.append(gapPath)

                }
            } else {
                // update memory of last point
                lastPointMemory = path.segments.last?.pointB
            }
        } else {
            // update memory of last point
            lastPointMemory = path.segments.last?.pointB
        }

        // add the path to the path set.
        newPathSet.paths.append(path)
    }

    return newPathSet
}

func getCommands() -> [CoordinateCommand] {
    let tempPathSet = addGapPaths()

    var coordinateCommands = [CoordinateCommand]()

    // lift to move to first position
    coordinateCommands.append(ZCommand(z: 1, pauseWhenReached: false))
    // move to first position
    let firstPosition = tempPathSet.paths.first!.segments.first!.pointA
```

```
coordinateCommands.append(XYCommand(x: firstPosition.x, y:
 firstPosition.y, pauseWhenReached: false))

var pathMemory: Path2D?

for path in tempPathSet.paths {

    if let previousPath = pathMemory {
        if path.height != previousPath.height {
            // setup gap path

            switch path.height {
            case .down:
                coordinateCommands.append(ZCommand(z: 0,
                 pauseWhenReached: false))
            case .up:
                coordinateCommands.append(ZCommand(z: 1,
                 pauseWhenReached: false))
            }


        }
    } else {
        debugPrint("ERROR: no path memory in loop")

        switch path.height {
        case .down:
            coordinateCommands.append(ZCommand(z: 0, pauseWhenReached:
             false))
        case .up:
            coordinateCommands.append(ZCommand(z: 1, pauseWhenReached:
             false))
        }
    }



    // adding XY commands for the path
    for line in path.segments {
        // segments are continuous, so we only need to move between
         pointA values.
        coordinateCommands.append(XYCommand(x: line.pointB.x, y:
         line.pointB.y, pauseWhenReached: false))

    }

    pathMemory = path
}

// end by lifting pen
coordinateCommands.append(ZCommand(z: 1, pauseWhenReached: false))
```

```
        return coordinateCommands
    }
}
```