```
#
# Implements procedureWithGrowthPattern() which allows a general
# thresholding procedure to be run with a growth pattern where
# when one location finishes, it is used to seed any of its
# immediate neighbours that have a 'wave number' one greater
# than itself.
# Note that this is different from some growth patterns which allow
# for the starting guess to be the average of several neighbours,
# thus requiring all neighbours to complete before that location
# can commence (eg Turpin et al's implementation of HFA growth pattern).
#
# The procedure is controlled by a growth pattern matrix, and
# requires a matrix of guesses for the first subset of locations,
# and functions to start, step, test-for-stop, and get-final-threshold
# for the procedure. See comments and Zest242.r for an example.
# The grid must have a 'chain' of immediate neighbours to locations
# not numbered 1 (the initial points) as they can only be opened for
# testing if an immediate 8-neighbour is complete.
#
# This version also has allowance for catch trials and an adaptive response
# window.
#
# WARNING - the function setResponseWindow assume that the responseWindow
#           is in position 8 of the 3rd line of the makeStim function
#           This is truly disgusting! It needs to be fixed in the future.
#
# Author: Andrew Turpin & Luke Chong
# Date: Sun 17 May 2015 09:26:05 AEST
#

####################################################################
# Use startFun, stepFun and stopFun to run all locations with number
# 1 in the growthPattern. When one finishes it opens up any locations
# numbered 2 in the immediate 8-neighbourhood. When one of those finishes
# a 3 is opened if abutting, and so on.
# After each location finishes, use finalFun to get threshold and num pres.
# Within each subset of possible locations,
# locations are presented in random order.
#
# INPUTS:
#   gp  - growth pattern matrix (min(gp) == 1). See 'chain' comment above.
#   gn  - growth next. lookup table of next locations to open up in the growth pattern
#   statrs - matrix same size as gp giving start values for gp==1
#   startFun - function(guess, rw, cl) that creates a state for
#              location (rw, cl) in gp using guess as start guess
#   stepFun  - function(state) returns a new state after one presentation
#   stopFun  - function(state) returns TRUE if location finished
#   finalFun - function(state) returns c(final threshold, number presenations)
#   FPTrials - perform an FP trial every time the presentation counter hits this value
#   FNTrials - perform an FN trial every time the presentation counter hits this value
#   FPLevel  - level of stimuli for false positive in cd/m^2
#   FNPause  - duration of pause (in ms) after a FN presentation, to allow for recovery after bright stimulus.
#   FNDelta  - the degree of brightness relative to current threshold at that location (in dB)
#               e.g. if FNDelta = 10, FN stimulus of 10 dB brighter than current estimated threshold is presented.
#   FNLocationThreshold - locations with a final threshold of at least this value (cd/m^2)
#                         are candidates for a FN presentation.
#   FPSize  - size of FP stimulus (diameter in mm)
#   FNSize  - size of FN stimulus (diameter in mm)
#   initialRespWin - starting response window in ms
#   respWinBuffer  - time added to mean of respWin to determine final response Window
#   catchTrialLoadFreq - Frequency of catch trials in the first minute. Usually more frequent in order to front load.
#   catchTrialFreq - Frequency of catch trials for the remainder of the test
#   catchTrialMax - Maximum number of FP and FN catch trials for the test (e.g. if catchTrialMAx = 5, will result in 5 FP and 5 FN catch trials)
#
# RETURNS: list of two matrices, each with same dimensions as gp
#           t is final threshold at each location
#           n is number of presentations at each location
####################################################################
#source("testStatusOutput.r")
if((!'audio' %in% installed.packages()))
  install.packages("audio")
library("audio")

procedureSuprathreshold <- function(
  startTime,
  locations, #previously 'gp'
  # gn,
  # starts,
  startFun,
  stepFun,
  stopFun,
  finalFun,
  gridPat,
  catchTrialLoadFreq=6,
  catchTrialFreq=20,
  catchTrialMax=14,
  FPLevel=dbTocd(60, 4000/pi),
  FNDelta=10,
  FNPause=300,
  FNLocationThreshold=20,
  FPSize=1.72,
  FNSize=1.72,
  initialRespWin=1200,
  respWinBuffer=250,
  moveProj = T,
  minInterStimInt = 0) {

    # ####################################################################
    # # Return any locations that are in the immediate 8 neighbours
    # # of [rw,cl] and have a wave number == gp[rw,cl]+1.
    # #
    # # INPUTS
    # #   gp   - growth Pattern matrix
    # #   rw   - row of location
    # #   cl   - column of location
    # #
    # # RETURNS: matrix of locations where column 1 = row index, column 2 = column index
    # ########################################################################
    # openUP <- function(gp, gn, rw, cl, states) {
    #   wave <- gp[rw,cl]
    #   locations <- NULL
    #   x <- states[[rw,cl]]$x
    #   y <- states[[rw,cl]]$y
    #
    #   nextLocs <- gn[[wave]][[paste(x,y,sep=" ")]]
    #
    #   if (!is.null(unlist(nextLocs))) {
    #
    #     locations <- matrix(nextLocs,length(unlist(nextLocs))/2,2) #make sure locations are in matrix form (becomes numeric if there is only one row)
    #
    #     for (row in 1:nrow(locations)) { # convert back to row and column
    #       locations[row,1] <- 91 + locations[row,1]
    #       locations[row,2] <- 55 - locations[row,2]
    #     }
    #     locations <- t(apply(locations,1,rev)) # flip so that row index is in first column and col index in second column
    #     locations <- locations[apply(locations,1,function (x) {is.null(states[[x[1],x[2]]])}),] # remove locations that have already been opened up
    #     locations <- matrix(locations,length(unlist(locations))/2,2)
    #     if (nrow(locations) == 0) {locations <- NULL}
    #   }
    #   return(locations)
    # }

    ####################################################################
    # Present FP or FN trial. Return TRUE(seen) or FALSE (not seen)
    ##################################################################
  # NEED TO REDO FOR NEG CATCH TRIALS
  presentCatch <- function(posOrNeg, responseWindow, currentThresholds, states,index) {
        if (posOrNeg == "POS") {
            s <- list(x=9, y=9, level=FPLevel, size=FPSize, duration=200,
                  responseWindow=responseWindow)
        } else {

            k <- which(currentThresholds > FNLocationThreshold,arr.ind=TRUE) ## find rw and cl of eligible locations
            l <- k[sample(nrow(k),size=1),] # choose a location at random: l[rw,cl]

            s <- list(x=states[[l[1],l[2]]]$x, y=states[[l[1],l[2]]]$y,
                      level=dbTocd(round(currentThresholds[l[1],l[2]] - FNDelta),4000/pi),
                      size=FNSize, duration=200,
                      responseWindow=responseWindow)
        }
        class(s) <- "opiStaticStimulus"

        if (moveProj) {
          showStim <- opiPresent(stim=s,states[[index[[1]][1],index[[1]][2]]]$makeStim(0,0))
        } else {
          showStim <- opiPresent(stim=s)
        }

        return (c(showStim,list(stimulus=s$level)))
    }

    ##################################################################
    # If there is only one location remaining, present a random stimulus
    # so that that particular location is not tested in a row
    #
    ##################################################################
  # NEED TO REDO
  presentDummy <- function (grid,responseWindow,dummyState,states) {
      locIndex <- which(!is.na(grid),arr.ind=TRUE)
      loc <- locIndex[round(runif(1,1,nrow(locIndex))),] # choose a location at random
      stimulus <- round(runif(1,0,30)) # choose a stimulus intensity at random
      s <- list(x=dummyState(30,loc[1],loc[2])$x,y=dummyState(30,loc[1],loc[2])$y,
                level=dbTocd(stimulus, 4000/pi),size=FNSize,duration=200,responseWindow=responseWindow)

      class(s) <- "opiStaticStimulus"

      if (moveProj) {
        showStim <- c(opiPresent(stim=s,states[[index[[1]][1],index[[1]][2]]]$makeStim(0,0)),list(x=s$x,y=s$y,stimulus=stimulus))
      } else {
        showStim <- c(opiPresent(stim=s),list(x=s$x,y=s$y,stimulus=stimulus))
      }
        return (showStim)
    }

    ##################################################################
    # Function to alter the response window of a state
    # Note does not allow responseWindow to be less than respWinBuffer
    # Returns the state after alteration
    ##################################################################
    setResponseWindow <- function(state,respWinBuffer,responseWindow) {

        m <- state$makeStim
        body(m)[[2]][[3]][[8]] <- respWinBuffer + responseWindow
        state$makeStim <- m

        return(state)
    }

    ##################################################################
    # Function to remove previous presentation response information
    # if the observer made a known false response
    #
    ##################################################################
    #NEED TO FIX
    applyUndos <- function () {
      #print(locsPresented)
      myEnv <- parent.env(environment())
      delLocs <- NULL
      while (gUndos > 0 && nrow(locsPresented) >= gUndos) {
        rr <- locsPresented[nrow(locsPresented),1]
        cc <- locsPresented[nrow(locsPresented),2]

        #delLocs - table of locations to be deleted.
        #Purpose: to identify locations that need to be deleted twice.
        delLocs <- rbind(delLocs,c(rr,cc))

        z <- unlist(lapply(locs, function(x) all(x == c(rr,cc))))

        if (any(z)) {
          print('removing an unterminated location')
        } else {
          print('removing a terminated location')
          myEnv$locs <- c(locs,list(c(rr,cc)))
          myEnv$finished_counter <- finished_counter - 1
        }

        # check for locations with repeated deletions.
        # If there has been a repeat, need to look up second last stim value rather than last.
        lookupIndex <- sum(apply(delLocs,1,function (x) ((x[1] == rr) && x[2] == cc)))
        prevStimVal <- tail(states[[rr,cc]]$stimuli,lookupIndex)[1]
        prevStimVal <- which(prevStimVal == states[[rr,cc]]$domain)

        if (tail(states[[rr,cc]]$responses,lookupIndex)[1] == FALSE) {
          states[[rr,cc]]$pdf <- states[[rr,cc]]$pdf / (1-states[[rr,cc]]$likelihood[prevStimVal,])
          myEnv$states[[rr,cc]]$pdf <- states[[rr,cc]]$pdf / sum(states[[rr,cc]]$pdf)
        } else {
          states[[rr,cc]]$pdf <- states[[rr,cc]]$pdf / states[[rr,cc]]$likelihood[prevStimVal,]
          myEnv$states[[rr,cc]]$pdf <- states[[rr,cc]]$pdf / sum(states[[rr,cc]]$pdf)
        }
        myEnv$locsPresented <- locsPresented[-nrow(locsPresented),]
        gUndos <<- gUndos - 1

        if (details$gridType != "practice") {
          cat(file=paste(details$dx,"/",details$gridType,"
",details$stimSizeRoman,"/",details$name,"_",details$dx,"_",details$grid,"_",details$stimSizeRoman,"_",details$eye,"Eye_",details$date,"_",details$startTime,"_stimResponses.txt",sep=""),

            append=TRUE,paste("Presentation at location x =",states[[rr,cc]]$x,"y =",states[[rr,cc]]$y,"was deleted\n",sep=" "))
        }
      }

    }

    ##########################################################################################################
    # Function for the adaptive interstimulus interval
    #
    # INPUTS
    #   responseTime - vector of total response times throughout test
    #   minISI - the minimum inter-stimulus interval
    #   interStimMultiplier - multiplier of the mean response time, which determines the max interstim interval
    #
    ############################################################################################################
    interStimInt <- function (responseTime = respTime,minISI,interStimMultiplier = 1) {
      if (!is.null(respTime)) {
        Sys.sleep(runif(1, min=minISI, max= max(minISI,mean(responseTime) * interStimMultiplier))/1000)  # pause before presenting next stimulus
      } else {
        Sys.sleep(200/1000)  #If there have been no response times recorded yet, make interstim interval 200 ms
      }
    }


    #################################################################
    # set up answers and loop vars
    #################################################################

    currentThresholds <- starts
    finishedThresholds <- matrix(NA, nrow(gp), ncol(gp))
    currentNumPres    <- matrix(NA, nrow(gp), ncol(gp))

    states <- array(list(), dim=c(nrow(gp), ncol(gp)))

        # Set up locations whose wave == 1
    locs <- which(gp == 1, arr.ind=TRUE)
    locs <- split(locs, 1:nrow(locs))
    for (i in 1:length(locs)) {
        rc <- locs[[i]]
        states[[rc[1], rc[2]]] <- startFun(starts[rc[1],rc[2]], rc[1], rc[2])
    }

    respWin <- rep(initialRespWin,5)  ## set up adaptive response window

    respTime <- NULL      # vector of response times
    fp_counter <- NULL    # vector of responses for FP
    fn_counter <- NULL    # vector of responses for FN
    finished_counter <- 0 # number of terminated locations
    counter <- 1          # number of presentations
    locsPresented <- NULL # vector of locations presented in order
    gUndos <<- 0
    index <- locs[runif(1,1,length(locs))] # choose random location to test first

    #################################################################
    # loop while still some unterminated locations
    #################################################################
    # what is gRunning?
    while (length(locs) > 0 && gRunning) {
        start_time <- Sys.time()
        applyUndos()

        if ((counter <= 60 && length(fp_counter) < catchTrialMax && (counter %% catchTrialLoadFreq == 0) && ((counter/catchTrialLoadFreq) %% 2 != 0)) ||
            (counter > 60 && length(fp_counter) < catchTrialMax && (counter %% catchTrialFreq == 0) && ((counter/catchTrialFreq) %% 2 != 0))) {

            result <- presentCatch("POS", mean(respWin) + respWinBuffer, currentThresholds, states,index) #adaptive response window
            #result <- presentCatch("POS", initialRespWin, currentThresholds, states,index) # fixed response window

            if (result$seen) {
              for (i in 1:2) {
                wait((play(sin(1:8000/20)))) ## play 2 beeps if FP error is made
              }
            }

            fp_counter <- c(fp_counter, min(1,result$seen))

            testStatus(result$seen,currentNumPres,currentThresholds,finishedThresholds,finished_counter,gp,fp_counter,fn_counter,stateInfo=states[[rw,cl]],respTime,testGrid = gridPat)
            if (details$gridType != "practice") {
            cat(file=paste(details$dx,"/",details$gridType,"
",details$stimSizeRoman,"/",details$name,"_",details$dx,"_",details$grid,"_",details$stimSizeRoman,"_",details$eye,"Eye_",details$date,"_",details$startTime,"_stimResponses.txt",sep=""),

                append=TRUE,sprintf("Location: %5s Stim: %2g dB Seen: %5s Resp Time: %5.2f Trial Time: %.0f\n", "FPCatch",cdTodb(FPLevel,4000/pi), result$seen, result$time,
difftime(Sys.time(),start_time,units = "secs") * 1000))
            }
            counter <- counter + 1
            start_time <- Sys.time()  #reset start_time counter for trial time
        }

        if ((counter <= 60 && length(fn_counter) < catchTrialMax && ((counter %% catchTrialLoadFreq == 0) && ((counter/catchTrialLoadFreq) %% 2 == 0)) && any(currentThresholds >
FNLocationThreshold,na.rm=TRUE)) ||
            (counter > 60 && length(fn_counter) < catchTrialMax && ((counter %% catchTrialFreq == 0) && ((counter/catchTrialFreq) %% 2 == 0)) && any(currentThresholds >
FNLocationThreshold,na.rm=TRUE))){

          result <- presentCatch("NEG", mean(respWin) + respWinBuffer, currentThresholds, states,index)
          #result <- presentCatch("NEG", initialRespWin, currentThresholds, states,index) #fixed response window

          Sys.sleep(FNPause/1000)
            fn_counter <- c(fn_counter, result$seen == FALSE)
            testStatus(result$seen,currentNumPres,currentThresholds,finishedThresholds,finished_counter,gp,fp_counter,fn_counter,stateInfo=states[[rw,cl]],respTime, testGrid = gridPat)
            if (details$gridType != "practice") {
              cat(file=paste(details$dx,"/",details$gridType,"
",details$stimSizeRoman,"/",details$name,"_",details$dx,"_",details$grid,"_",details$stimSizeRoman,"_",details$eye,"Eye_",details$date,"_",details$startTime,"_stimResponses.txt",sep=""),

                append=TRUE,sprintf("Location: %5s Stim: %2g dB Seen: %5s Resp Time: %5.2f\n Trial Time: %.0f\n", "FNCatch",cdTodb(result$stimulus,4000/pi), result$seen,
result$time,difftime(Sys.time(),start_time,units = "secs") * 1000))
            }

          counter <- counter + 1
          start_time <- Sys.time() #reset start_time counter for trial time
        }

        counter <- counter + 1

        # weight stimulus choice by growth pattern wave
        getWave <- lapply(locs, function (x) {gp[x[1],x[2]]})
        weight <- sapply(getWave, function (x) {1/(x^2)})

        if (length(locs) > 1) {
          index[2] <- locs[sample(1:length(locs),1,prob=weight)]
          while (all(index[[1]] == index[[2]])) {
            index[2] <- locs[sample(1:length(locs),1,prob=weight)]
          }
        } else {
          index[[1]] <- locs[[1]]
          index[[2]] <- locs[[1]]
        }

        rw <- index[[1]][1]
        cl <- index[[1]][2]
        rw2 <- index[[2]][1]
        cl2 <- index[[2]][2]

        locsPresented <- rbind(locsPresented,c(rw,cl))

        states[[rw,cl]] <- setResponseWindow(states[[rw,cl]],respWinBuffer,mean(respWin))  #Updates response window

        if (length(locs) > 1 && moveProj == TRUE) {
          states[[rw,cl]] <- stepFun(states[[rw,cl]],nextStimState=states[[rw2,cl2]])
        } else {
          states[[rw,cl]] <- stepFun(states[[rw,cl]])
        }

        if (all(details$gridType != c("Peripheral","P-Peripheral","P-Edge"))) {interStimInt(respTime,minInterStimInt)}
        currentThresholds[rw,cl] <- sum(states[[rw,cl]]$pdf*states[[rw,cl]]$domain) # update currentThresholds
        currentNumPres[rw,cl] <- states[[rw,cl]]$numPresentations

        testStatus(tail(states[[rw,cl]]$responses,1),currentNumPres,currentThresholds,finishedThresholds,finished_counter,gp,fp_counter,fn_counter,stateInfo=states[[rw,cl]],respTime,
testGrid = gridPat)
        if (details$gridType != "practice") {
          cat(file=paste(details$dx,"/",details$gridType,"
",details$stimSizeRoman,"/",details$name,"_",details$dx,"_",details$grid,"_",details$stimSizeRoman,"_",details$eye,"Eye_",details$date,"_",details$startTime,"_stimResponses.txt",sep=""),

          append=TRUE, sprintf("Location: x=%3g, y=%3g Stim: %2g dB Seen: %5s Resp Time: %5.2f Trial Time: %.0f\n", states[[rw,cl]]$x, states[[rw,cl]]$y,
            tail(states[[rw,cl]]$stimuli,1), tail(states[[rw,cl]]$responses,1), tail(states[[rw,cl]]$responseTimes,1),difftime(Sys.time(),start_time,units = "secs") * 1000))
        }

        if (length(locs) == 1) {
          dummy_start_time <- Sys.time()
          result <- presentDummy (gridPat,mean(respWin) + respWinBuffer,startFun,states)
          #result <- presentDummy (gridPat,initialRespWin,startFun,states) #fixed response window

          if (all(details$gridType != c("Peripheral","P-Peripheral","P-Edge"))) {interStimInt(respTime,minInterStimInt)}

          if (details$gridType != "practice") {
            testStatus(result$seen,currentNumPres,currentThresholds,finishedThresholds,finished_counter,gp,fp_counter,fn_counter,stateInfo=list(x=result$x,y=result$y),respTime, testGrid
= gridPat)
            cat(file=paste(details$dx,"/",details$gridType,"
",details$stimSizeRoman,"/",details$name,"_",details$dx,"_",details$grid,"_",details$stimSizeRoman,"_",details$eye,"Eye_",details$date,"_",details$startTime,"_stimResponses.txt",sep=""),

              append=TRUE,sprintf("Location: x=%3g, y=%3g Stim: %2g dB Seen: %5s Resp Time: %5.2f Trial Time: %.0f %5s\n",result$x,result$y,result$stimulus, result$seen,
result$time,difftime(Sys.time(),dummy_start_time,units = "secs") * 1000,"(Dummy Trial)"))
          }
        }

        if (tail(states[[rw,cl]]$responses,1)) {
          respWin <- c(tail(states[[rw,cl]]$responseTimes,1),respWin[-5])
          respTime <- c(tail(states[[rw,cl]]$responseTimes,1),respTime)
        }

        if (stopFun(states[[rw,cl]])) {
                # fill in finishedThresholds and remove from locs
            finishedThresholds[rw,cl] <- finalFun(states[[rw,cl]])[1]
            finished_counter <- finished_counter + 1

            locs <- locs[-which(sapply(locs,function(x) {all(x == index[[1]])}))]

                # look around for neighbours that can be opened
            newLocs <- openUP(gp, gn, rw, cl,states)

            if (!is.null(newLocs)) {
                for (i in 1:nrow(newLocs)) {
                    rc <- newLocs[i,]
                    states[[rc[1], rc[2]]] <- startFun(finishedThresholds[rw,cl], rc[1], rc[2])
                    locs <- c(locs, list(rc))
                }
            }
        }
        index[[1]] <- index[[2]]  #move next stimulus to current stimulus before next presentation sequence
    }
    if (gRunning) {
      currentThresholds[currentThresholds < states[[rw,cl]]$domain[6]] <- -1 #Set censored thresholds to -1
      finishedThresholds[finishedThresholds < states[[rw,cl]]$domain[6]] <- -1 #Set censored thresholds to -1
    }
    ### NEED TO INCORPORATE PARENT FILE FOR THIS

testStatus(result$seen,currentNumPres,currentThresholds,finishedThresholds,finished_counter,gp,fp_counter,fn_counter,stateInfo=states[[rw,cl]],respTime,plotStimResponse=FALSE,testGrid =
gridPat)
    return(list(t=currentThresholds, n=currentNumPres,fpc=fp_counter,fnc=fn_counter,rt=respTime))
}
```