Appendix D
A Simplified Simulation

1. R script to run simulation

```
Our intuition is that the results from our postponing model (specifically,
# the intended delay during the first encounter) can be simplified by considering
# just three factors: the immediate outcome of delaying, the opportunity costs
# of delaying, and the opportunity benefits of delaying. In the actual model, these
# three factors are highly non-linear; they depend on the exact environment, reserves
# and age of an agent, as well as any previous behavior during the same resource encounter.
# Moreover, an agent has to decide not just whether it delays, but also, how long
# it delays. Here we simplify all that, and just consider broad patterns in a
# decision where an agent can either delay, not delay, or be indifferent.
#   Specifically, the immediate outcome of delaying depends on both the
# expected change in the resource quality as we delay and the change in outcomes
# if there is an interruption. The opportunity costs is the expected quality of
# all positive resources that an agent could find in the meantime if it does not delay.
# The opportunity benefits is the expected outcome of all negative resources it might
# encounter, if it does not delay. Here we estimate all three using very simple methods,
# and combine these expected outcomes in a linear manner (each expected value will
# be weighted and then added). If the resulting score is sufficiently negative, an
# agent does not delay. If the resulting score is sufficiently positive, it does delay.
# If the score is close to 0, it is indifferent.
#   If our intuition is correct - and these three factors explain much of the variation
# in delay behavior - then there must be at least one linear combination of these
# three factors that results in qualitatively similar patterns as our model. Here
# we test that intuition. And spoilers: we don't even have to tweak the linear weights
# by much to get the same pattern.

# First, let's define some free parameters that we can use to scale
# the simulation of our intuition
parameters = list(

  # How important is the change in resource quality when an agent delays?
  scaleGrowth = 1.5,

  # How important is the expected change in outcome when there is an interruption?
  scaleInterruption = 1,

  # How important are the opportunity costs?
  scaleOpporunityCosts = 1,

  # How important are the opportunity benefits?
  scaleOpportunityBenefits = 1,

  # How strong do extrinsic events influence opportunity costs and benefits?
  scaleExtrinsicFuture = 1,

  # How far away should the resulting score maximally be away from 0 before
  # an agent is no longer indifferent?
  indifferenceThreshold = 2

)

# A function that generates an integer-only normal distribution. This function
# is the exact same as the one used in our main model.
normalDistributionIntegerOnly = function(domain, mean, sd){
  isInteger = domain %% 1 == 0
  df = data.frame(domain = domain, isInteger = isInteger)
  df$p = 0
  df$index = 1:nrow(df)
  dfIntegerOnly = subset(df, isInteger)

  if (sd == 0){
    dfIntegerOnly$p[which(abs(mean-dfIntegerOnly$domain)==min(abs(mean-dfIntegerOnly$domain)))]=1

    for (r in 1:nrow(dfIntegerOnly)){
      df$p[dfIntegerOnly$index[r]] = dfIntegerOnly$p[r]
    }
```

```
    return (df$p/sum(df$p))
  }

  options(scipen=999)
  unnormalizedProbabilities = exp((-(dfIntegerOnly$domain-mean)^2)/(2*sd^2))
  dfIntegerOnly$p = unnormalizedProbabilities/sum(unnormalizedProbabilities)

  for (r in 1:nrow(dfIntegerOnly)){
    df$p[dfIntegerOnly$index[r]] = dfIntegerOnly$p[r]
  }

  return ( df$p)

}



# Simulate all possible trials in a single environment
# Note: the environment is a list containing 5 named elements:
#   1. rmean (mean resource quality)
#   2. rsd (resource quality variation)
#   3. emean (mean extrinsic value)
#   4. erange (variation in extrinsic events - not used in this simulation)
#   5. int (the interruption rate)
runEnv = function(environment, trialData, envData){
  resValues = -15:15
  pRes = normalDistributionIntegerOnly(-15:15, environment$rmean, environment$rsd)
  currentTrialData = data.frame(rmean = rep(environment$rmean, 31),
                                rsd= rep(environment$rsd, 31),
                                emean = rep(environment$emean, 31),
                                erange = rep(environment$erange, 31),
                                int  = rep(environment$int, 31),
                                resValue = vector(length = 31, mode = 'numeric'),

                                resourceGrowth = vector(length = 31, mode = 'numeric'),
                                interruptionChange = vector(length = 31, mode = 'numeric'),
                                immediateConsequences = vector(length = 31, mode = 'numeric'),

                                opportunityCost = vector(length = 31, mode = 'numeric'),
                                opportunityBenefits = vector(length = 31, mode = 'numeric'),
                                netOpportunityCost  = vector(length = 31, mode = 'numeric'),

                                delayWeight = vector(length = 31, mode = 'numeric'),
                                delayYesNo = vector(length = 31, mode = 'numeric'))
  for (r in 1:length(resValues)){
    resValue = resValues[r]

    # Compute the delay weight: the tendency to delay or not
    # This tendency contains three dimensions:

    # 1. The immediate consequence of delaying during this resource encounter
    # This depends on growth of the resource, and the interruption rate,
    resourceGrowth = resValue/5
    interruptionChange = resValue *  environment$int

    immediateConsequences =
      (parameters$scaleGrowth * resourceGrowth) -
      (parameters$scaleInterruption * interruptionChange)

    # 2. The opportunity cost from future positive resources (i.e.,
    # by delaying I am missing out on other positive resources that I could have had)
    # This opportunity cost is the expected value of all future positive resources
    opportunityCost =
      normalDistributionIntegerOnly(-15:15, environment$rmean,environment$rsd) %*%
      c( rep(0,16), 1:15 ) *
      parameters$scaleOpporunityCosts

    # 2a. Positive opportunity costs are especially important when extrinsic events are negative
    # (as I have to get extra income from resources to offset the future losses), and are
    # less important when extrinsic events are positive (because I have to rely less on future
    # positive resource to survive)
    opportunityCost = opportunityCost * (1-environment$emean * parameters$scaleExtrinsicFuture)


    # 3. The opportunity benefits from avoiding future negative resources (i.e., the
```

```
    # benefits I have from not encountering other negative resources when I delay)
    opportunityBenefits =
      normalDistributionIntegerOnly(-15:15, environment$rmean, environment$rsd) %*%
      c(-15:-1, rep(0,16)) * parameters$scaleOpportunityBenefits * -1

    # 3a. Opportunity benefits are especially important when the mean resource is
    # negative and the extrinsic events
    # are positive (I 'just' have to avoid harm - the extrinsic events will take care of me)
    opportunityBenefits =
      opportunityBenefits * (1+ ((environment$rmean<0 & environment$emean>0) *
                             abs(environment$rmean)/4 * parameters$scaleExtrinsicFuture) )

    # 3b. Opportunity benefits are less important when the mean resource is
    # positive and the extrinsic events are positive (no harm ever comes to me)
    opportunityBenefits =
      opportunityBenefits * (1- ((environment$rmean>0 & environment$emean>0) *
                             abs(environment$rmean)/4 * parameters$scaleExtrinsicFuture) )


    # Combine the three factors together to get a delay preference
    netOpportunityCosts = opportunityCost - opportunityBenefits
    delayWeight = immediateConsequences - netOpportunityCosts

    # Determine behavior
    delayYesNo = 0 - (delayWeight < -1*parameters$indifferenceThreshold) + (delayWeight > parameters$indifferenceThreshold)

    # Store
    currentTrialData$resValue[r] = resValue

    currentTrialData$resourceGrowth[r] = resourceGrowth
    currentTrialData$interruptionChange[r] = interruptionChange
    currentTrialData$immediateConsequences[r] = immediateConsequences

    currentTrialData$opportunityCost[r] = opportunityCost
    currentTrialData$opportunityBenefits[r] = opportunityBenefits
    currentTrialData$netOpportunityCost [r] = netOpportunityCosts

    currentTrialData$delayWeight[r] = delayWeight
    currentTrialData$delayYesNo[r] = delayYesNo

  }
  trialData = rbind(trialData, currentTrialData)

  # Now the we have the behavior for each possible resource in an environment,
  # we can compute the expected behavior in each environment, weighing the
  # outcome of each encounter by the likelihood of that resource quality
  envData[nrow(envData)+1,] = c(environment$rmean,
                          environment$rsd,
                          environment$emean,
                          environment$erange,
                          environment$int,
                          currentTrialData$immediateConsequences %*% normalDistributionIntegerOnly(-15:15, rmean, rsd),
                          currentTrialData$opportunityCost %*% normalDistributionIntegerOnly(-15:15, rmean, rsd),
                          currentTrialData$opportunityBenefits %*% normalDistributionIntegerOnly(-15:15, rmean, rsd),

                          currentTrialData$delayWeight %*% normalDistributionIntegerOnly(-15:15, rmean, rsd),
                          currentTrialData$delayYesNo %*% normalDistributionIntegerOnly(-15:15, rmean, rsd))
  return (list(trialData, envData))

}


# Create two empty data sets, containing the environment and trial level (=
# single encounter) data
trialData = data.frame(rmean = vector(),
                 rsd = vector(),
                 emean  = vector(),
                 erange = vector(),
                 int  = vector(),
                 resValue = vector(),

                 resourceGrowth = vector(),
                 interruptionChange  = vector(),
                 immediateConsequences = vector(),
```

```
                    opportunityCost = vector(),
                    opportunityBenefits = vector(),
                    netOpportunityCost  = vector(),

                    delayWeight = vector(),
                    delayYesNo = vector())
envData = data.frame(rmean = vector(),
                   rsd= vector(),
                   emean = vector(),
                   esd = vector(),
                   int  = vector(),
                   immediateConsequences = vector(),
                   opportunityCost = vector(),
                   opportunityBenefits = vector(),

                   delayWeight = vector(),
                   delayYesNo = vector())

# Populate these data sets by calling the function above for all possible
# environments
for (rmean in seq(-4,4, 1))
  for (rsd in seq(0,8,2))
    for (emean in -1:1)
      for (erange in 1)
        for (int in c(0, 0.2, 0.5)){

          # Create the environment
          environment = list(
            rmean = rmean,
            rsd = rsd,
            emean = emean,
            erange = erange,
            int = int
          )

          # Call the function
          results = runEnv(environment, trialData, envData)

          # store the results.
          trialData = results[[1]]
          envData = results[[2]]
        }


# Interruption rates are between 0 and 1. This domain is also the home of
# floating point inaccuracy. To avoid some weird behavior (and a lot of
# headaches), multiply the interruption rates by 100 and round the result.
trialData$intPer = round(trialData$int * 100)
envData$intPer = round(envData$int * 100)


# Categorize the delay willingness to have the same aesthetics as the
# original data. We'll use 7 discrete levels.
discreteLevels = 7
envData$noZeroDelayYesNo = envData$delayYesNo + abs(min(envData$delayYesNo))
unitSize = max(envData$noZeroDelayYesNo)/(discreteLevels-1)
envData$categoricalDelayYesNo = round(envData$noZeroDelayYesNo/unitSize) * unitSize
```

2. Results From Simulation