UrnVersion2_solution

Paul M

12/9/2020

Solution to Urn version 2, counting how many non-black colors there are at the end

This is an example answer for lab task 4 of week 4. So the goal is to simulate an Urn that contains a black ball and look at how may different colors there are in it at the end.

This particular code is written in “object-oriented” style. This is something that R is not really designed for, but many of us who learned languages like C++ are used to writing object-oriented code, so I wanted to see how well R could mimic that style of coding.

As ever, we start with a few global variables…

```
set.seed(87)
ColorBreaks <- seq(0.5,10.5,1)
```

We also use a library that allows R to display a “progress bar” that shows how much long the code is likely to run.

```
#install.packages("progress")
library(progress)
```

Now, the object oriented bit! We set up a giant function to represent the things in the Urn and all the operations we wil need to perform on them.

```
Urn <- function(StartingConfig,FinalNumberOfBalls,UseMutationBall,WeightOfMutationBall){
  UrnThings <- list(
    # set up the elements of the Urn
    Balls = mat.or.vec(1,FinalNumberOfBalls),    # Note that we need to use "=" here rat
her than "<-"
    InitialNumberOfBalls = length(StartingConfig),
    UseMutationBall = UseMutationBall,
    MutationBallWeight = WeightOfMutationBall,
    StartingConfiguration = StartingConfig
  )

  # A sanity checks
  if (FinalNumberOfBalls<length(StartingConfig)){
    cat("Too many balls in starting configuration. Exit.")
  }

  # Initialize the laas in the Urn to start with
  Nballs <- length(StartingConfig)
  for (i in 1:Nballs){
    UrnThings$Balls[i] <- StartingConfig[i]
  }

  ##############################################################################
  ### Class functions   - these are the functions that will operate on the Urn  ###
  ##############################################################################

  UrnThings$DrawFromUrn<-function(NumBalls){
    r<-sample(1:NumBalls,1)
    return (UrnThings$Balls[r])
  }

  UrnThings$RunUrnModel<-function(){
    HowManyBallsDoWeNeed<-length(UrnThings$Balls)
    Nballs<-length(UrnThings$StartingConfiguration)
    # set the starting configuration
    for (i in 1:length(StartingConfig)){
      UrnThings$Balls[i]<-StartingConfig[i]
    }
    # how many colors do we have already?
    iColorCounter <- length(apply(UrnThings$Balls,1,unique)) # this is going to keep tra
ck of how many new colors we use
    while (Nballs < HowManyBallsDoWeNeed){
      # do we pick the black ball (if there is one)?
      if (UrnThings$UseMutationBall==1){   # this is not actually represented by a ball
 in the urn
        p<-runif(1,0,1)
        if (p<UrnThings$MutationBallWeight/(UrnThings$MutationBallWeight+Nballs)){
          #  we picked the black ball
          WhichToCopy<-0    # 0 codes the black ball
        }else{
          WhichToCopy<-UrnThings$DrawFromUrn(Nballs)
        }
      }else{
```

```
      WhichToCopy<-UrnThings$DrawFromUrn(Nballs)
    }
    if (WhichToCopy==0){    # 0 is used to code the black ball
      # change the color of some other ball
      IndexOfBall<-ceiling(runif(1,0,1))
      iColorCounter<-iColorCounter+1
      #colorname<-paste("NewColor",iColorCounter)
      UrnThings$Balls[IndexOfBall]<-iColorCounter
    }else{
      Nballs<-Nballs+1
      UrnThings$Balls[Nballs]<-WhichToCopy
    }
    #cat(WhichToCopy," ")
  }
  #return(U)
}

UrnThings$CountNumberOfColorOfFirstBall<-function(){
  ColorNeeded<-UrnThings$Balls[1]
  return (sum(UrnThings$Balls==ColorNeeded))
}

UrnThings$CountNumberOfColors<-function(){
  # counts the number of (non-black colors in the urn)
  NumColors<-length(table(Urn$Balls))
  return (NumColors)
}

UrnThings$FindCommonestColor<-function(){
  Commonest<-max(table(Urn$Balls))
  return (Commonest)
}

##############################################################################
################### End of class functions ###################################
##############################################################################




UrnThings<-list2env(UrnThings)     # this is needed, but I don't know why!
class(UrnThings)<-"Urn"   # the name of the class we have created

return(UrnThings)
}
```

Having defined a new Urn class, we can define the way some common functions will work on it. Here, we define how it will be printed.

```
print.Urn <- function(U){
  if (class(U)!="Urn") stop();
  cat("Balls: ")
  cat(U$Balls)
  cat("\nStarting configuration: ")
  cat(U$StartingConfiguration)
  cat(paste("\nUse Black ball?: ",U$UseMutationBall,"\nBlack ball weight: ",U$MutationBa
llWeight))
}
```

Let's test it out. First, we set up the scenario we are considering. We wil run the model until there are 50 non-black balls in the urn. We start by considering the case in which the black ball has weight 1.

```
NoReps <- 1000   # how many samples to generate
WeightOfBlackBall <- 1
NoOfBalls <- 50
ThisUrn<-Urn(c(1,1),NoOfBalls,1,1)   # The urn we will use for the simulations
ThisUrn$MutationBallWeight<-WeightOfBlackBall  # now the black ball has the required wei
ght
```

Now simulate a large number of Urns and look at the distribution of how many colors there are at the end.

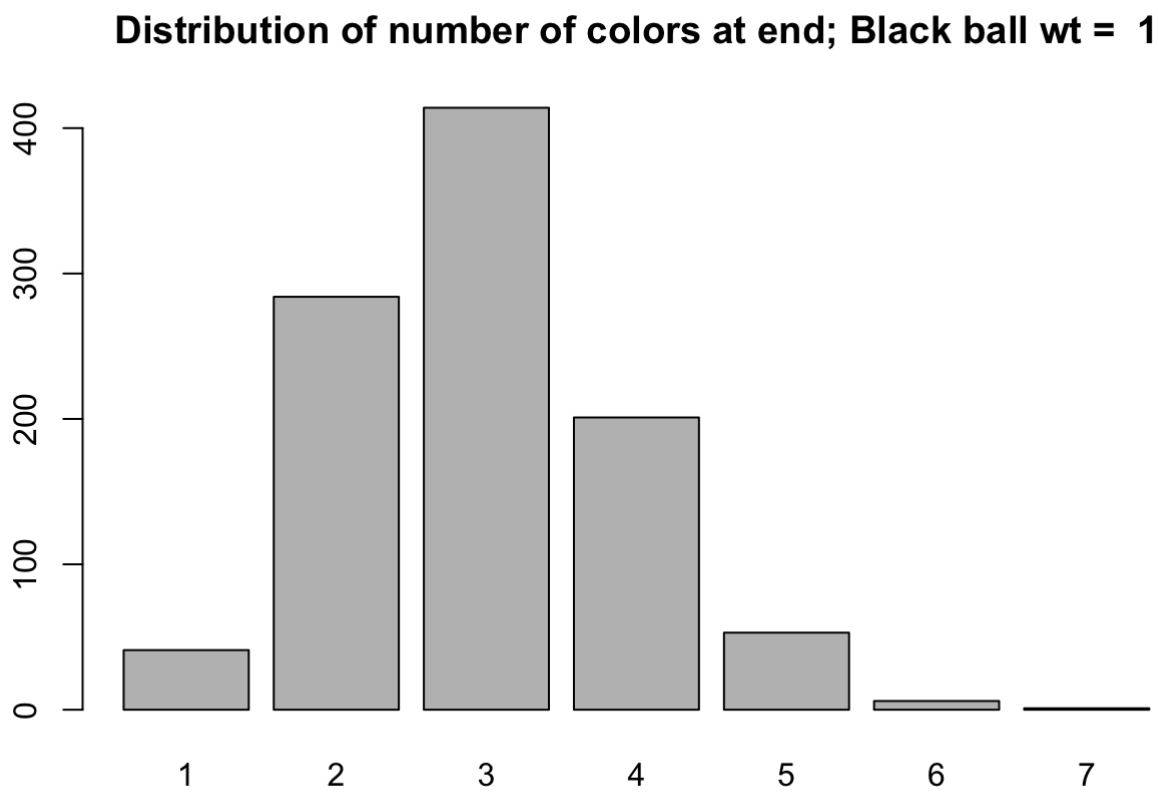
```
NumberOfColorsAtEnd <- rep(0,NoReps)
#pb <- progress_bar$new(total = NoReps)   # This sets=up the progress bar
for (k in 1:NoReps){
 # pb$tick()   # this makes the progress bar 'tick over'
  ThisUrn$RunUrnModel()   # simulate the urn using this weight
  NumberOfColorsAtEnd[k] <- length(table(ThisUrn$Balls))
}
```

We will now have NoReps accepted weights, so plot a histogram to see what they look like `# posterior distribuf(weight|HowManyColorsNeeded) looks like

```
##
## For black ball weight  1  Mean number of colors is  2.963
```

```
## Table showing distribution of colors at end
```

```
## NumberOfColorsAtEnd
##   1   2   3   4   5   6   7
##  41 284 414 201  53   6   1
```

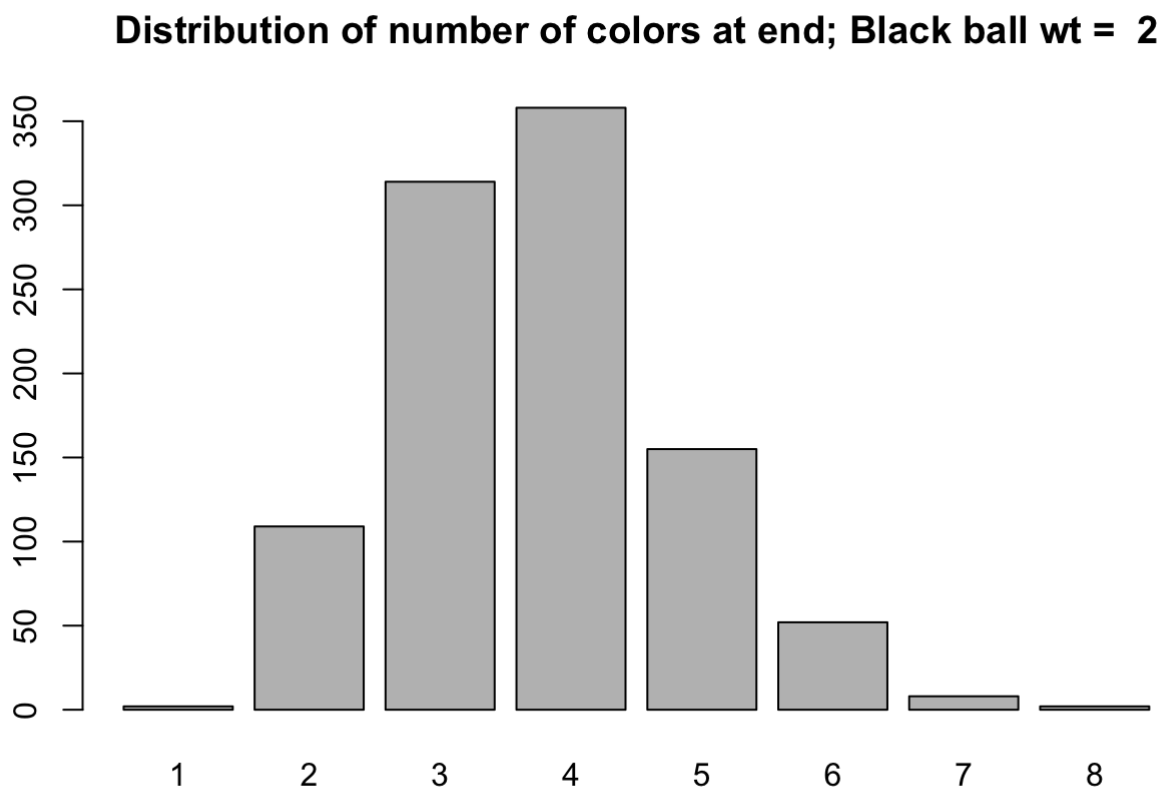


Repeat for other weights. This shows an example of how to re-use an existing chunk of code (here the chunk that plotted the results). Rmd will insert the code in the chunk called “plot” where you put the reference “<>”. (This only seems to work when you “knit” the entire file; if, instead, you “Run all” it returens an error message, for reasons I do not yet understand.)

```
##
## For black ball weight  2  Mean number of colors is  3.753
```

```
## Table showing distribution of colors at end
```

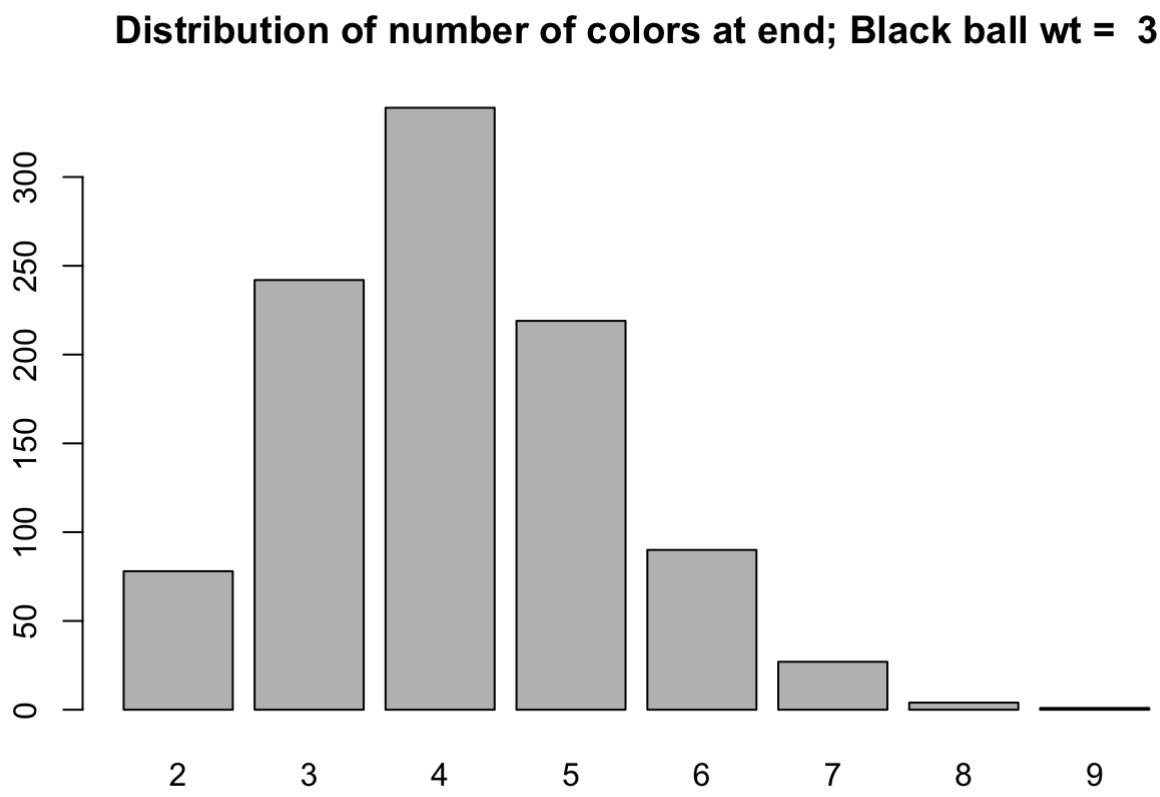
```
## NumberOfColorsAtEnd
##   1   2   3   4   5   6   7   8
##   2 109 314 358 155  52   8   2
```




```
##
## For black ball weight  3  Mean number of colors is  4.103
```

```
## Table showing distribution of colors at end
```

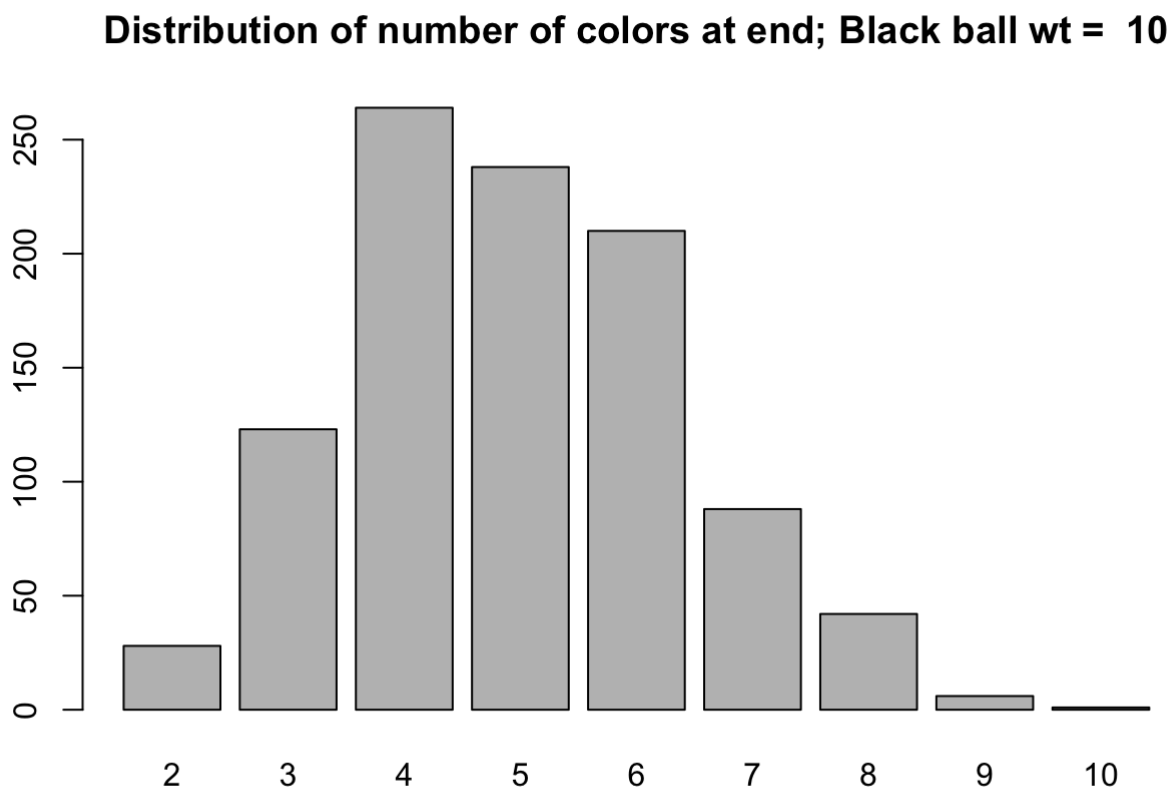
```
## NumberOfColorsAtEnd
##   2   3   4   5   6   7   8   9
##  78 242 339 219  90  27   4   1
```




```
##
## For black ball weight  10  Mean number of colors is  4.947
```

```
## Table showing distribution of colors at end
```

```
## NumberOfColorsAtEnd
##   2   3   4   5   6   7   8   9  10
##  28 123 264 238 210  88  42   6   1
```


Distribution of number of colors at end; Black ball wt = 10
0
50
100
150
200
250
2
3
4
5
6
7
8
9
10