R Simulations for Kids

W. John Braun*
Department of Statistical and Actuarial Sciences
University of Western Ontario

June 30, 2011

Abstract

How does one introduce young students to ideas of randomness and uncertainty? Does the statistical program R have a role to play in elementary schools? This paper describes the code for an example used by the author in a session with approximately 20 fifth- and sixth-grade students.

Key Words: random numbers, simulation, Markov chain, fire, Monopoly

1 Introduction

There has been limited attention to probability and statistics in the public schools at the elementary level. In Canada, elementary statistics and probability concepts are introduced as part of the math curriculum. However, it is not clear that this material is engaging the students and attracting them into further study of probability and statistics.

More attention has been paid to probability and statistics at the high school level. In Ontario, for example, the course Mathematics of Data Management (Ontario Ministry of Education, 2000) covers material which is often found in introductory statistics university courses at a somewhat reduced difficulty level. However, the course is designed as a prerequisite for social science and business students and is not explicitly directed towards math, science and engineering students. Thus, there appears to be a discontinuity in probability and statistics in the school system and at the university.

Is it possible to attract young students to the field of probability and statistics? Monte Carlo simulation on the computer may be one way to approach this. The R program (R Core Development Team, 2011) provides a relatively easy way to produce executable code for such simulations. This short article provides an illustration.

2 Simulating a Popular Children's Game

A relatively simple-to-program example which is particularly captivating for students of this age involves the simulation of plays of a Monopoly game. Most children are familiar with the game, and when the board is displayed, they recognize it immediately.

The code below assumes simplified rules for the Monopoly game; the effects of the cards which sometimes give movement instructions are ignored. An ambitious coder would be able to add this feature in fairly easily.

The graphics described here are easily rendered using functions from the `grid` package (e.g. Murrell, 2005). A quick introduction to the kinds of functions needed to produce the figures can be found in the paper by Zhou and Braun (2010).

We start by assigning the property colours to a character vector, in the order that the properties are laid out on the board.

```
propertycolors <- c("grey", "purple", "grey", "purple", "grey",
  "grey", "lightblue", "grey", "lightblue", "lightblue", "black", "magenta",
  "grey", "magenta", "magenta", "grey", "orange", "grey", "orange",
  "orange", "grey", "red", "grey", "red", "red", "grey", "yellow",
  "yellow", "grey", "yellow", "black", "green", "green",
  "grey", "green", "grey", "grey", "darkblue", "grey", "darkblue")
```

The board is constructed using the following `MonopolyBoard()` function, which makes repeated calls to the `grid.rect()` function in order to produce the rectangular property locations. The ith location is filled with the ith colour from the above vector.

```
MonopolyBoard <- function() {
    # This function is used to create the image of the
    # Monopoly Board, identifying the color-coded property
    # locations.
    require(grid)
    heights <- widths <- c(1.5, rep(1,9), 1.5)/12
    ycenter <- cumsum(c(1.5, rep(1,9), 1.5)/12)-widths/2
    xcenter <- rep(.75/12, 11)
    ypropcenter <- xpropcenter <- numeric(40)

    # The first 11 properties (From "GO" to "JAIL")
    for (i in 1:11) {
        xpropcenter[i] <- xcenter[i]
        ypropcenter[i] <- ycenter[i]
        vp <- viewport(x=xpropcenter[i], y=ypropcenter[i],
             height=heights[i], width=1.5/12)
        pushViewport(vp)
        grid.rect(gp=gpar(fill=propertycolors[i]))
        upViewport()
    }

    # The next 9 properties
    for (i in 2:10) {
        xpropcenter[i+10] <- ycenter[i]
        ypropcenter[i+10] <- 1-xcenter[i]
        vp <- viewport(x=xpropcenter[i+10], y=ypropcenter[i+10],
             height=1.5/12, width=widths[i])
        pushViewport(vp)
        grid.rect(gp=gpar(fill=propertycolors[10+i]))
        upViewport()
    }

    # (From Free Parking to "Go to Jail!")
    for (i in 1:11) {
        xpropcenter[i+20] <- 1-xcenter[i]
        ypropcenter[i+20] <- rev(ycenter)[i]
        vp <- viewport(x=xpropcenter[i+20], y=ypropcenter[i+20],
             height=heights[i], width=1.5/12)
        pushViewport(vp)
        grid.rect(gp=gpar(fill=(propertycolors[20+i])))
        upViewport()
    }

    # The last 9 properties
    for (i in 2:10) {
        xpropcenter[i+30] <- rev(ycenter)[i]
        ypropcenter[i+30] <- xcenter[i]
        vp <- viewport(x=xpropcenter[i+30], y=ypropcenter[i+30],
```

```
            height=1.5/12, width=widths[i])
        pushViewport(vp)
        grid.rect(gp=gpar(fill=propertycolors[i+30]))
        upViewport()
    }
    list(xpropcenter, ypropcenter)  # locations of the property centers
}
```

In order to simulate the play of the game, the function `MonopolyPlays()` is used.

```
MonopolyPlays <- function(Nplays = 0) {
    BoardOutput <- MonopolyBoard()  # Build the Board First
    xpropcenter <- BoardOutput[[1]]   # Locate the Property Centers
    ypropcenter <- BoardOutput[[2]]
    if (Nplays > 0) {  # The following code governs moves of a single player.
            player <- circleGrob(r=.025, gp=gpar(fill="white"), name="player")
        playerland <- editGrob(player, vp=viewport(x=xpropcenter[1],
            y=ypropcenter[1]), name="playerland") # The player starts at Go
        dicevalue <- textGrob(x=.5, y=.5, "start", name="dicevalue")
        grid.draw(playerland)  # Draw the Player's First Position (at Go)
        grid.draw(dicevalue)    # Indicate that the game is at Start
        currentthrow <- sample(1:6, size=2, replace=TRUE)
        lands <- sum(currentthrow) + 1 # Position after First Roll of the Dice
                grid.edit("dicevalue", label=sum(currentthrow))  # Change the Value of Dice
        grid.edit("playerland", vp=viewport(x=xpropcenter[lands],
                y=ypropcenter[lands]))  # Move the Player to New Location
        if (Nplays > 1) {
            for (i in 2:Nplays){
                currentthrow <- sample(1:6, size=2, replace=TRUE)
                lands <- (lands + sum(currentthrow) - 1)%%40 + 1
                        # Add the current dice roll to the current location on the board
                if (lands == 31) lands <- 11  # Go to Jail!
                             grid.edit("dicevalue", label=sum(currentthrow))
                        # Roll the Dice
                grid.edit("playerland", vp=viewport(x=xpropcenter[lands],
                    y=ypropcenter[lands]))
                        # Move the Player
            }
        }
    }
}
```

The output from `MonopolyBoard()` is the coordinate locations of the property centers on the square board (assumed to be a unit square). The horizontal coordinates are listed first, followed by the corresponding vertical coordinates, as listed below. A side effect of the function is to draw the board in colour. This can be seen in Figure 1.

```
MonopolyBoard()
```

```
## Loading required package:  grid
```

```
## [[1]]
##  [1] 0.0625000 0.0625000 0.0625000 0.0625000 0.0625000 0.0625000 0.0625000
##  [8] 0.0625000 0.0625000 0.0625000 0.0625000 0.1666667 0.2500000 0.3333333
## [15] 0.4166667 0.5000000 0.5833333 0.6666667 0.7500000 0.8333333 0.9375000
## [22] 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000
## [29] 0.9375000 0.9375000 0.9375000 0.8333333 0.7500000 0.6666667 0.5833333
## [36] 0.5000000 0.4166667 0.3333333 0.2500000 0.1666667
##
```

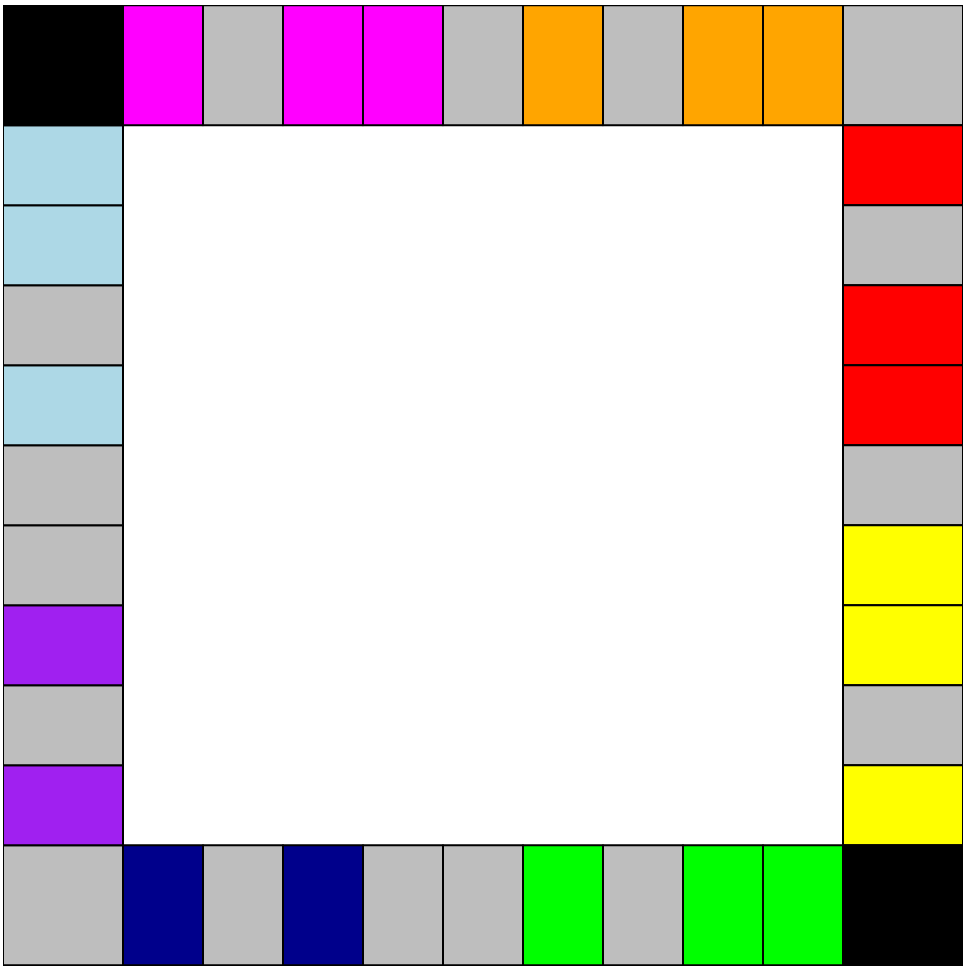
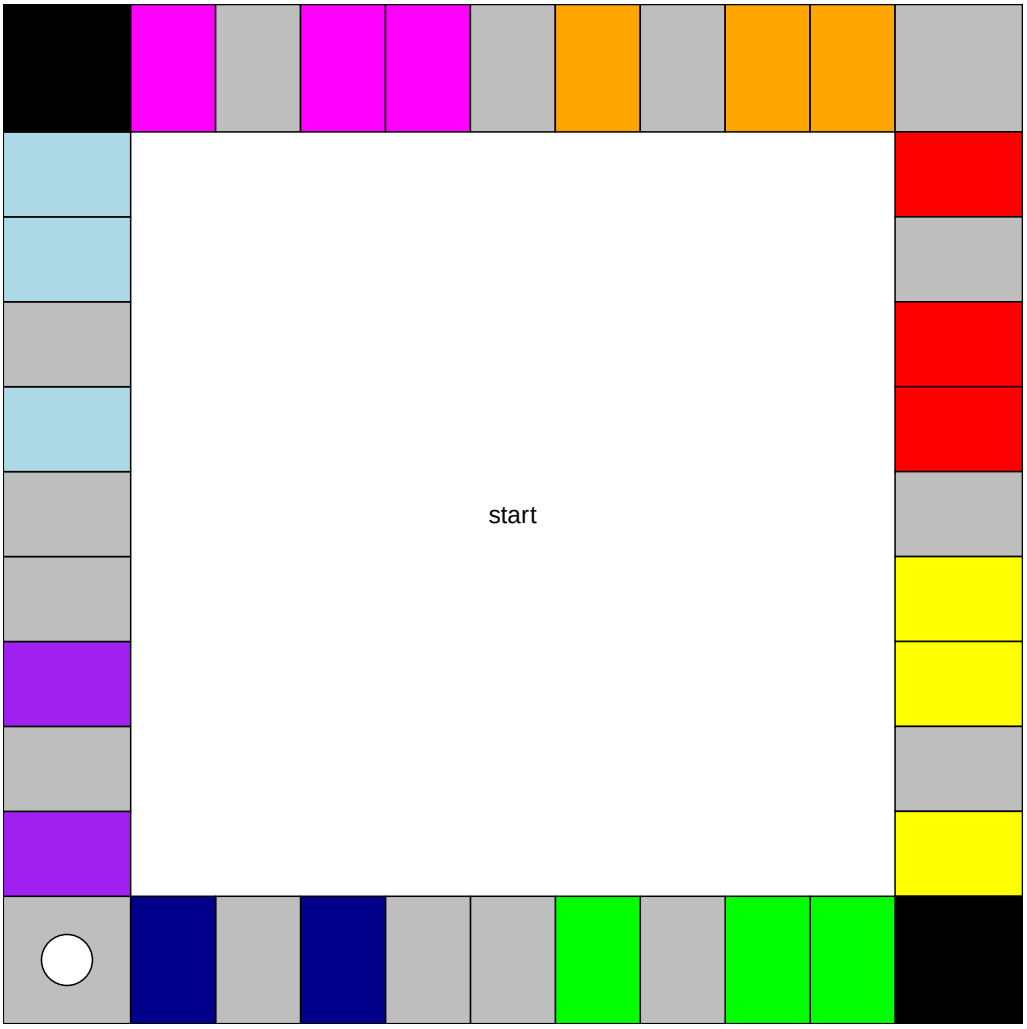


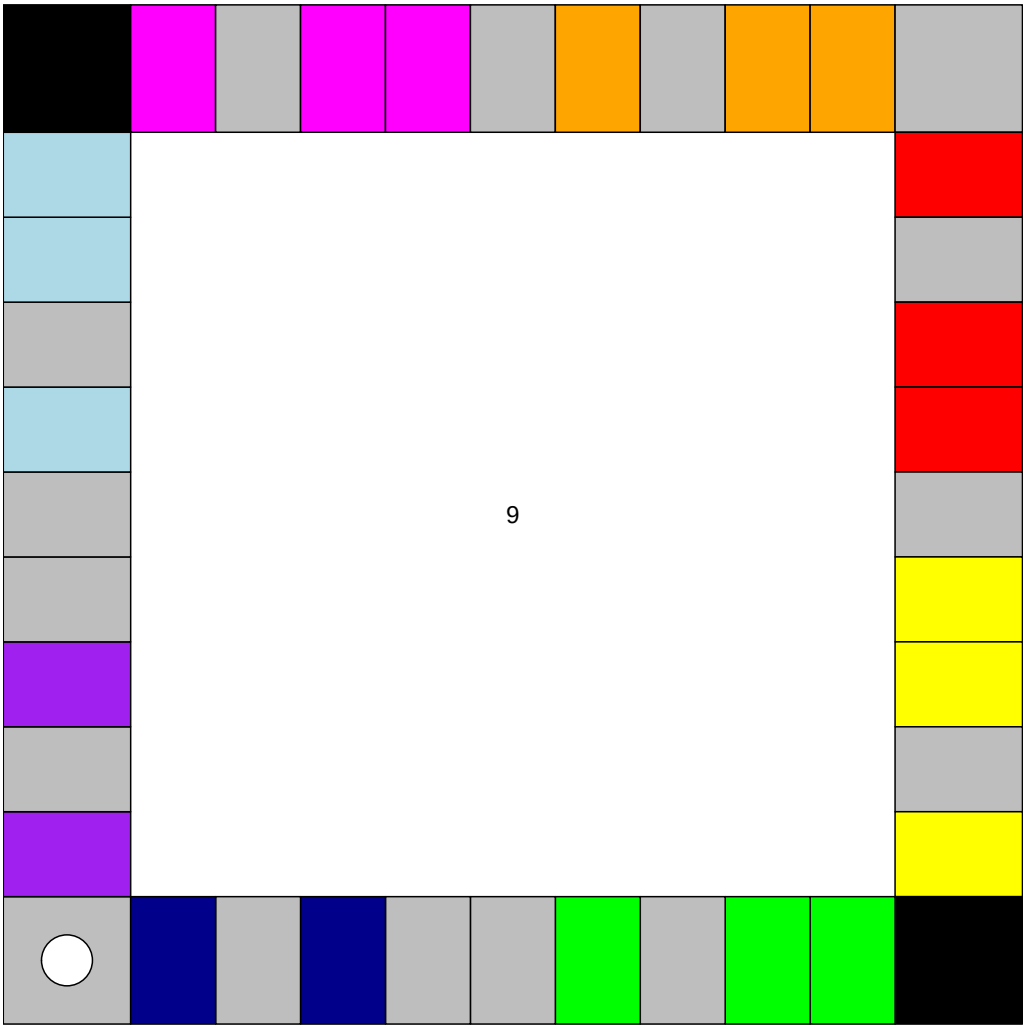
Figure 1: The graph above shows the various color-coded properties of the Monopoly board. The color coding has been chosen to match the colored properties used in the actual game. The grey bars represent non-colored areas of the board such as railroad, utilities, and so on. The black bar represents Jail.

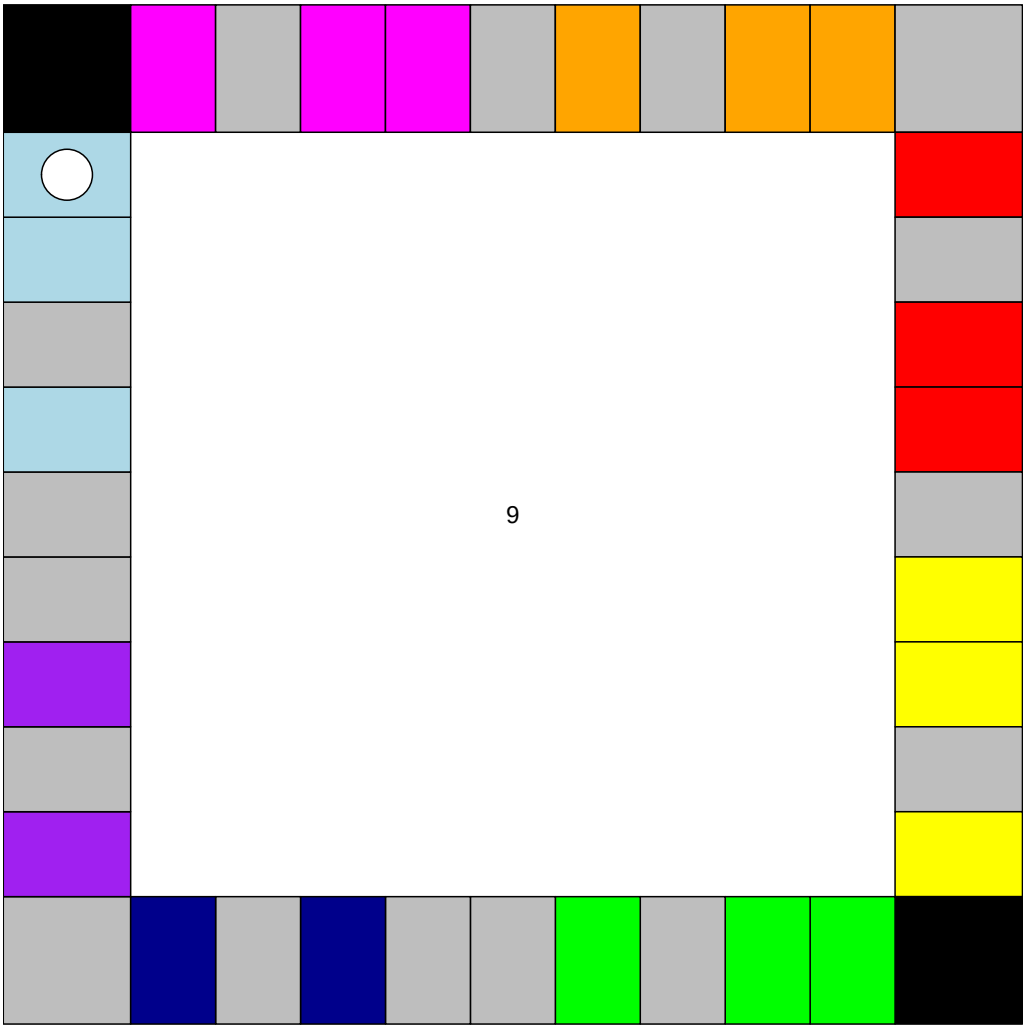
```
## [[2]]
##  [1] 0.0625000 0.1666667 0.2500000 0.3333333 0.4166667 0.5000000 0.5833333
##  [8] 0.6666667 0.7500000 0.8333333 0.9375000 0.9375000 0.9375000 0.9375000
## [15] 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000 0.9375000
## [22] 0.8333333 0.7500000 0.6666667 0.5833333 0.5000000 0.4166667 0.3333333
## [29] 0.2500000 0.1666667 0.0625000 0.0625000 0.0625000 0.0625000 0.0625000
## [36] 0.0625000 0.0625000 0.0625000 0.0625000 0.0625000
```

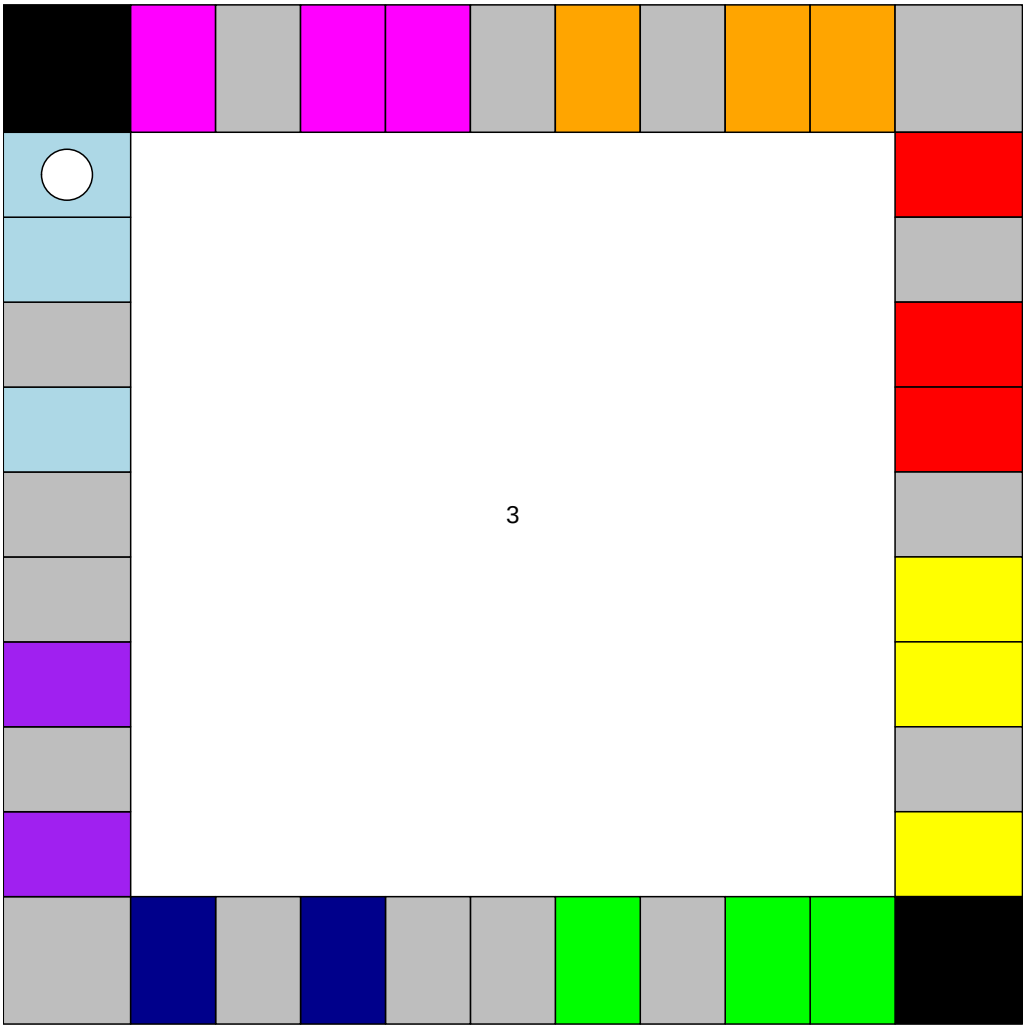
The results of the first two dice throws are obtained through executing the following command. The sequence of moves is illustrated on the succeeding pages of this document.

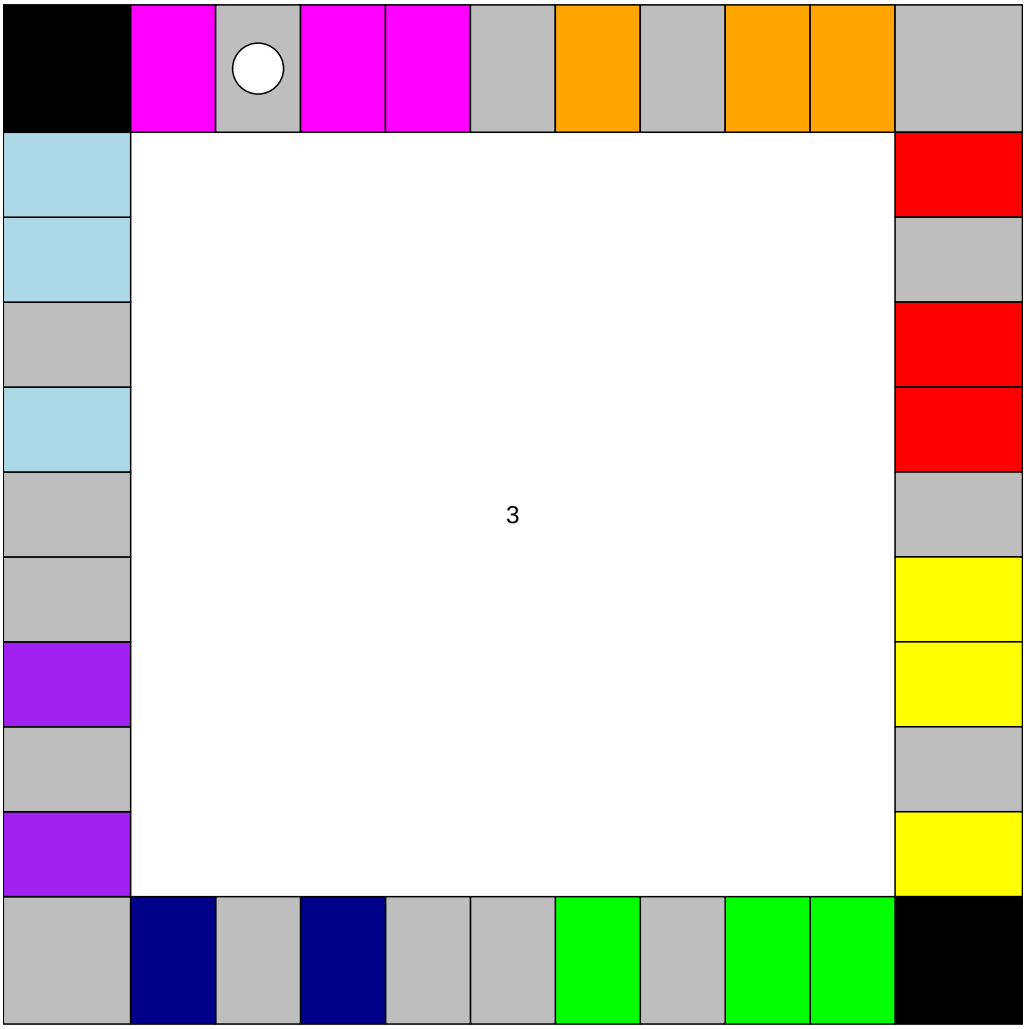
```
MonopolyPlays(2)
```




9


9


3



The white circle represents the player's current location on the board. The value at the center represents the sum of the values on the dice.

Having demonstrated a few moves on the Monopoly board, we are now ready to conduct a major simulation to determine which properties are most frequently landed on. The basic code for the simulation is as follows:

```
Nplays <- 1000000
lands <- numeric(Nplays)
lands[1] <- 1                # The player starts at Go.
for (i in 2:Nplays){
    currentthrow <- sample(1:6, size=2, replace=TRUE)
    lands[i] <- (lands[i-1] + sum(currentthrow) - 1)%%40 + 1
    if (lands[i]==31) lands[i] <- 11  # "Go To Jail" is in position 31
}
frequencies<- table(lands)
visitedproperties <- as.numeric(names(frequencies))
```

Of course, the output must be visualized in a reasonable way. The following code produces a histogram whose bars are color-coded according to the property colors. The result is pictured in Figure 2.

```
hist(lands, breaks=seq(.5, 40.5, 1), col=propertycolors,
xlab="Location", main="",  ylab="Number of Visits", axes=FALSE)
box(); axis(2)
title("Which Property is Landed on Most Often?")
```

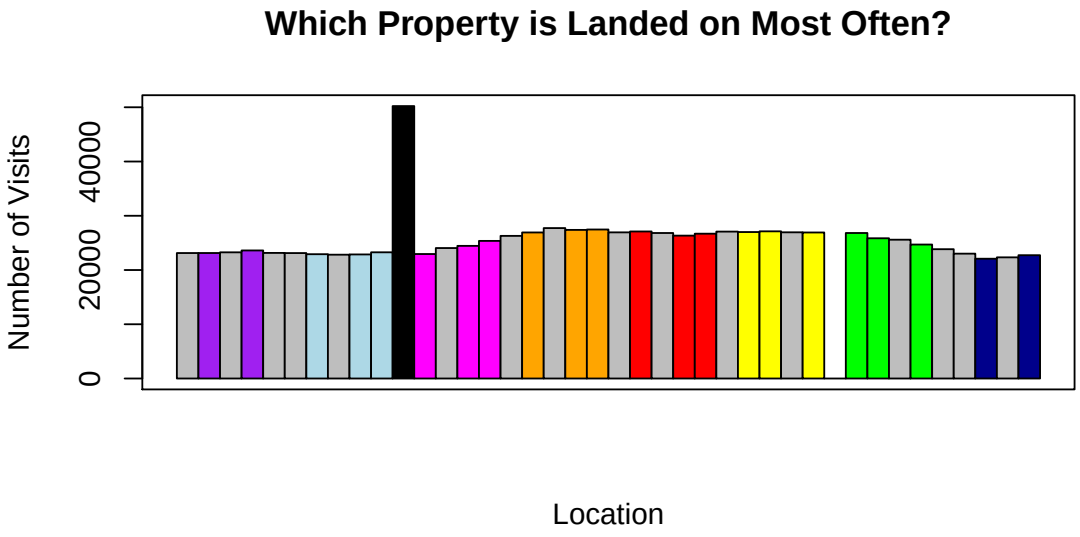


Figure 2: Histogram showing frequency distribution of landings on the various properties.

On a 2 GHz laptop computer, this code takes less than 2 seconds to execute, resulting in the sequence of properties landed on by a single individual in 1000000 moves. Ten- and eleven-year-olds will not be interested in the code, but high school students should be able to understand it. Furthermore, it is a simple 40-state Markov chain, so it is likely of interest even in certain undergraduate probability and statistics courses.

At this point, the students will naturally want to discuss the results. Several questions can be addressed. Is it surprising that the dark blue properties are not landed on as frequently as the reds and oranges? Why is Jail so popular? Notice the increase in frequency as one proceeds from Jail through the light purple properties towards the orange properties; how is this increase related to the frequencies of the values of pairs of dice?

It is also interesting to consider the amount of money that could be spent at each of the color-coded properties. Some code that could be used to estimate expected values of the amount paid at each type of color-coded property is provided below. It is assumed that hotels are used at each location.

```
locations <- c("Go", "purple1", "card", "purple2", "incometax",
"RR1", "blue1", "card", "blue2", "blue3", "jail", "pink1",
"util1", "pink2", "pink3", "RR2", "orange1", "card", "orange2",
"orange3", "FP", "red1", "card", "red2", "red3", "RR3", "yellow1",
"yellow2", "util2", "yellow3", "gotojail", "green1", "green2",
"card", "green3", "RR4", "card", "darkblue1", "luxurytax", "darkblue2")

hotelvalue <- c(0, 250, 0, 450, -200, 200, 550, 0, 550, 600, 0, 750, 0,
750, 900, 200, 950, 0, 950, 1000, 0, 1050, 0, 1050, 1100,  200, 1150,
1150, 0, 1200, 0, 1275, 1275, 0, 1400, 200, 0, 1500, -75, 2000)

payments <- hotelvalue[visitedproperties]*frequencies
names(payments) <- locations[visitedproperties]
payments

##        Go   purple1      card   purple2 incometax       RR1     blue1
##         0   5782750         0  10621800  -4630200   4624600  12602700
##      card     blue2     blue3      jail     pink1     util1     pink2
##         0  12577950  13958400         0  17210250         0  18337500
```

```
##     pink3       RR2   orange1      card   orange2   orange3        FP
##  22827600   5258400  25576850         0  26023350  27466000         0
##      red1      card      red2      red3       RR3   yellow1   yellow2
##  28459200         0  27652800  29362300   5416000  31046550  31200650
##     util2   yellow3    green1    green2      card    green3       RR4
##         0  32300400  34199325  32958750         0  34573000   4767200
##      card darkblue1 luxurytax darkblue2
##         0  33118500  -1674975  45454000
```

A better way to display this information is with a bar chart as shown in Figure 3. The code is below.

```
barplot(payments, col=propertycolors[-31])  # Remove Go to Jail from the Colour List
```

This final plot seems to show that kids may understand this game better than adults. Boardwalk is best – provided you can put a hotel on it!

3 Concluding Remarks

The activity described in this paper was demonstrated in a short session with about 20 ten- and eleven-year-old students. Throughout the session, the students appeared to be attentive, and at numerous points, they posed good questions and were quick to respond when they were asked questions.

The use of simulation to illustrate processes which are of immediate interest to the children appears to be a good way of facilitating learning about ideas of randomness and uncertainty.

References

[1] Murrell, P. (2005) *R Graphics*. Chapman and Hall/CRC, Boca Raton, Florida.

[2] Ontario Ministry of Education (2000) *The Ontario Curriculum Grades 11 and 12* 48–54. `www.edu.gov.on.ca/eng/curriculum/secondary/math1112curr.pdf`

[3] R Development Core Team (2011). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`

[4] Zhou, L. and Braun, W.J. (2010) Fun with the Grid Package, *Journal of Statistics Education* **18**.

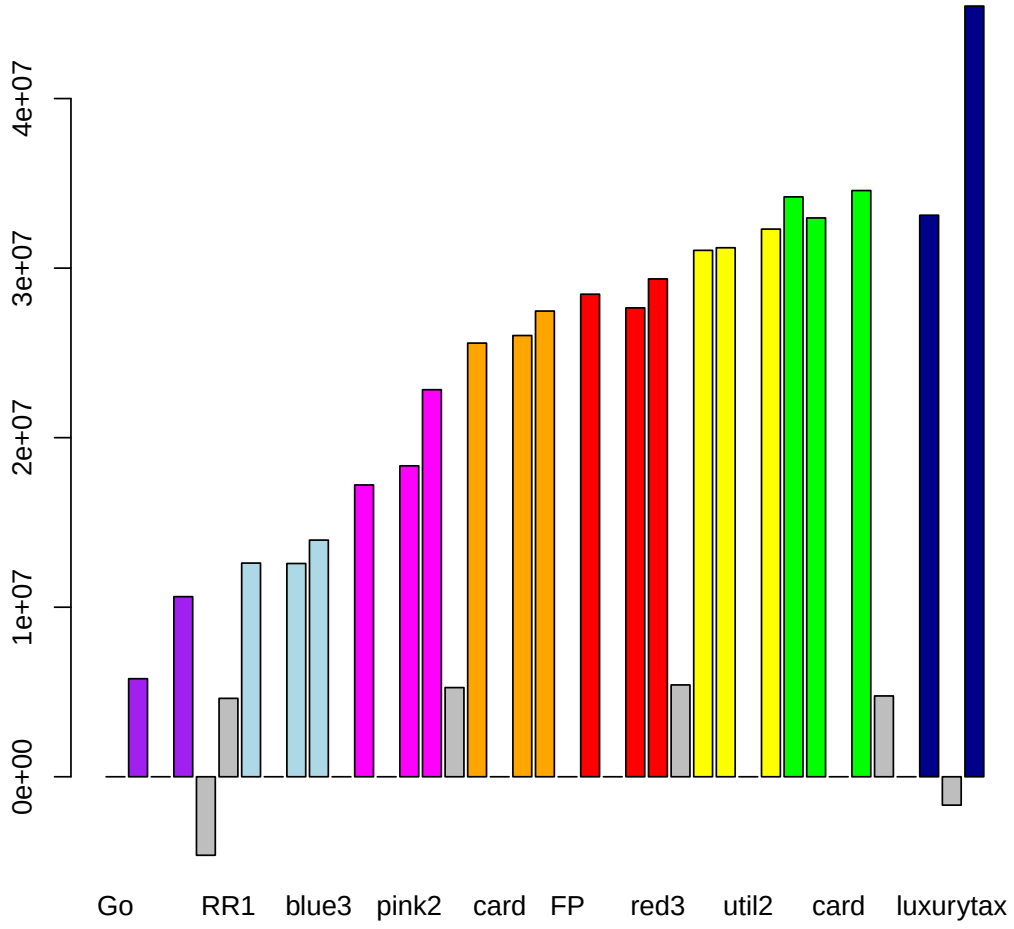


Figure 3: Bar chart of payouts for various properties assuming hotels everywhere possible. Negative values correspond to the Income Tax and Luxury Tax locations.