Probabilities in the Game of Life

Jenn Halbleib

May 6, 2018

Contents

Motivation 1

The Game of Life: The Board 2

First Things First: Transition Matrix and Time to Absorption 3

Beyond the Board: Other Game Rules 4

Outcomes from Simulating the Whole Game (with Some Assumptions) 4

Future work 10

Appendix 1: Game Materials 10

Code Appendix 1: Making the Probability Matrix 11

Code Appendix 2: Simulation 14

Sources 32

Motivation

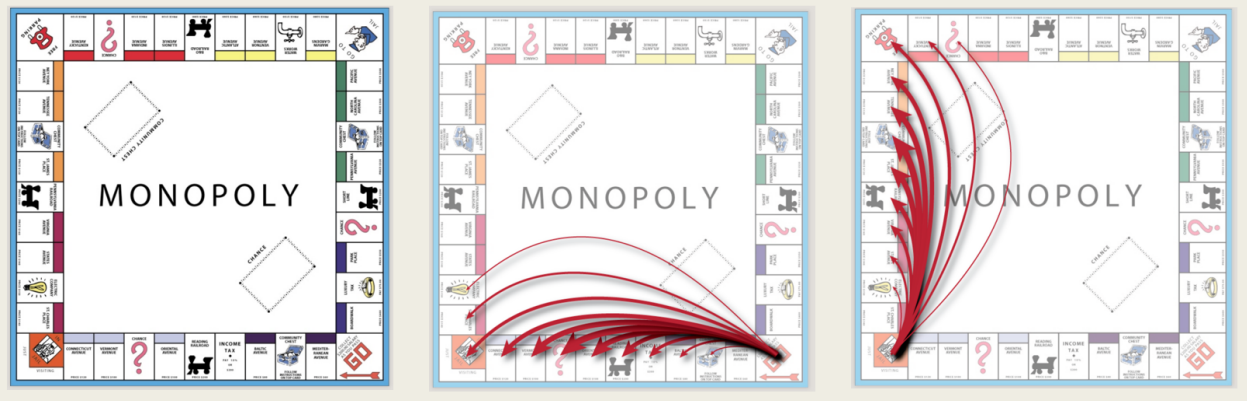


Figure 1: Visualizing Movement in Monopoly (Source: Hayes 2013)

When students are first presented with Markov Chains, the game Monopoly often serves as a motivational example. The Monopoly board (pictured above) is a continuous Markov chain with an interesting property. When a player lands on the "Go to Jail" space, forward movement in the chain is interupted when the player is sent backwards to the "Jail" space. In the long-term probabilities of the board, this means that owning the properties between "Jail" and "Go to Jail" gives the player an advantage in gameplay.

This type of analysis struck me as particularly interesting in my early study of probabilty. So, for this project, I decided to analyze a board game with the Monopoly problem in mind. After some Internet searching, I settled on The Game of Life, since I could not find a formal analysis of it. In what follows, I will construct a proability matrix for the board and create a simulation of the game with the goal of highlighting some of the interesting aspects of gameplay.

The Game of Life: The Board

Figure 2: The Game of Life: Numbered Board (Source: Hasbro Gaming, Numbers Added)

In The Game of Life, players pass through the stages of life (going to college, getting a job, etc.), earning money along the way until they reach retirement at the end of the board path. Like Monopoly, the Game of Life has several idiosyncratic gameplay features. Movement is determined by a spinner, so each turn a player may move one to ten spaces. The board has six spaces that force the player to stop, voiding any part of the roll that passes the stopping space (see spaces 11, 26, 38, 67, 77, and 104). Additionally, the board path diverges in four places, where the player must choose which path to take (see the start and spaces 38, 67, and 104). It seems reasonable to assume that path choice affects whether a player wins or loses, on average. However, before analyzing questions about optimal game strategy, I will look at a few details using the board's transition matrix.

First Things First: Transition Matrix and Time to Absorption

In contructing a transition matrix, I made two assumptions:

1. The game spinner is unbiased, so spins $\in [1, 10]$ are equally likely.
2. At each path split, the player chooses their path randomly.

To view the code for the matrix, please see Code Appendix 1. Here, I will load the matrix from a .csv file.

```
#Loading matrix
board_matrix <- read.csv("Life_Matrix.csv")
#Removing column with line numbers added by export to .csv
board_matrix <- board_matrix[,2:136]
```

Matrix Eigenvalues

Viewing the board, we expect the long-term distribution $\bar{\pi}$ to show absorption into retirement. (I.e. We expect to find the player at the end of the game as the number of turns heads to infinity.) To verify this assumption, I constructed $\bar{\pi}$ from the left eigenvalues of the transition matrix. And, as expected, as $n = number\ of\ turns \to \infty, \mathbb{P}(in\ retirement) = 1$. (Dobrow (2016))

```
r <- eigen(t(board_matrix))
V <- r$vectors
lambda <- r$values
pibar <- V[,1]/sum(V[,1])
pibar
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [106] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Game Length

Using the transition matrix, we can also ask how long we expect a game to take, on average. To do so, we make retirement an absorbing state and calculate the expected time until absorption. (Dobrow (2016))

```
#Let retirement be absorbing
Q <- data.matrix(board_matrix[1:134, 1:134])

#Find the expected time matrix F
F <- ginv(diag(134) - Q)
```

```r
#Sum Row 0 for the expected number of steps
sum(F[1,])
```

```
## [1] 22.20811
```

Therefore, we expect the average game to take about 22.2 turns per player.

Beyond the Board: Other Game Rules

The Game of Life has only one goal: retire with more money than the other players. Players earn money in the following ways.

1. The players is assigned a Salary from one of two decks: College or Career. The average salary for the College deck is higher than the average salary for the Career deck. The player has two opportunites to obtain a college career. First, they may choose the College path from the starting position. Second, they may choose to follow the Night School path from space 38. These paths have an opportunity cost, since the paths for college careers are longer. The player recieves their salary each time a "Payday" space is passed or landed on.
2. Players collect children. At the time of retirement, each child is worth 50K.
3. Players buy houses. At the time of retirement, the house is sold for a price determined by spinning. The sale price may be lower or higher than the purchase price, depending on the spin.
4. Players draw Action cards. While the cards may pay or require payment from the player, each is worth 100K upon retirement. In almost all cases, the net effect is a gain upon payment in retirement.
5. Players land on Spin-to-Win spaces. Payment is determined by spinning.
6. Players take the Risky path at space 104. On this path, the player may land on spaces that pay or require payment of 100K.
7. Players retire. Each player recieves a bonus upon reaching retirement, with bonuses decreasing by place (i.e. first to finish recieves 400K, next recieves 300K, etc.).

Outcomes from Simulating the Whole Game (with Some Assumptions)

To explore basic game strategies and average outcomes for common game events, I created a simulation of the game in R.

To simplify things, I made the following assumptions.

1. The game spinner is unbiased, so spins $\in [1, 10]$ are equally likely.
2. At each path split, the player chooses their path randomly. The one place this assumption makes things funky is if a player has chosen the College path and then chooses the Night School path. It seems unlikely that someone would choose to pursue a College salary twice in real gameplay.
3. Only one player is on the board at any given time. For this reason, the retirement bonuses are not included in the simulation.
4. All cards may be drawn at any point in the game. This means that a player could do something weird like buy the same house twice or draw the same action card again.

To review the code for this simulation, please see Code Appendix 2. Here, I will load 1,000,000 simulation records from a .csv.

```
#loading 1,000,000 simulation records
results2 <- read.csv("results.csv")
```

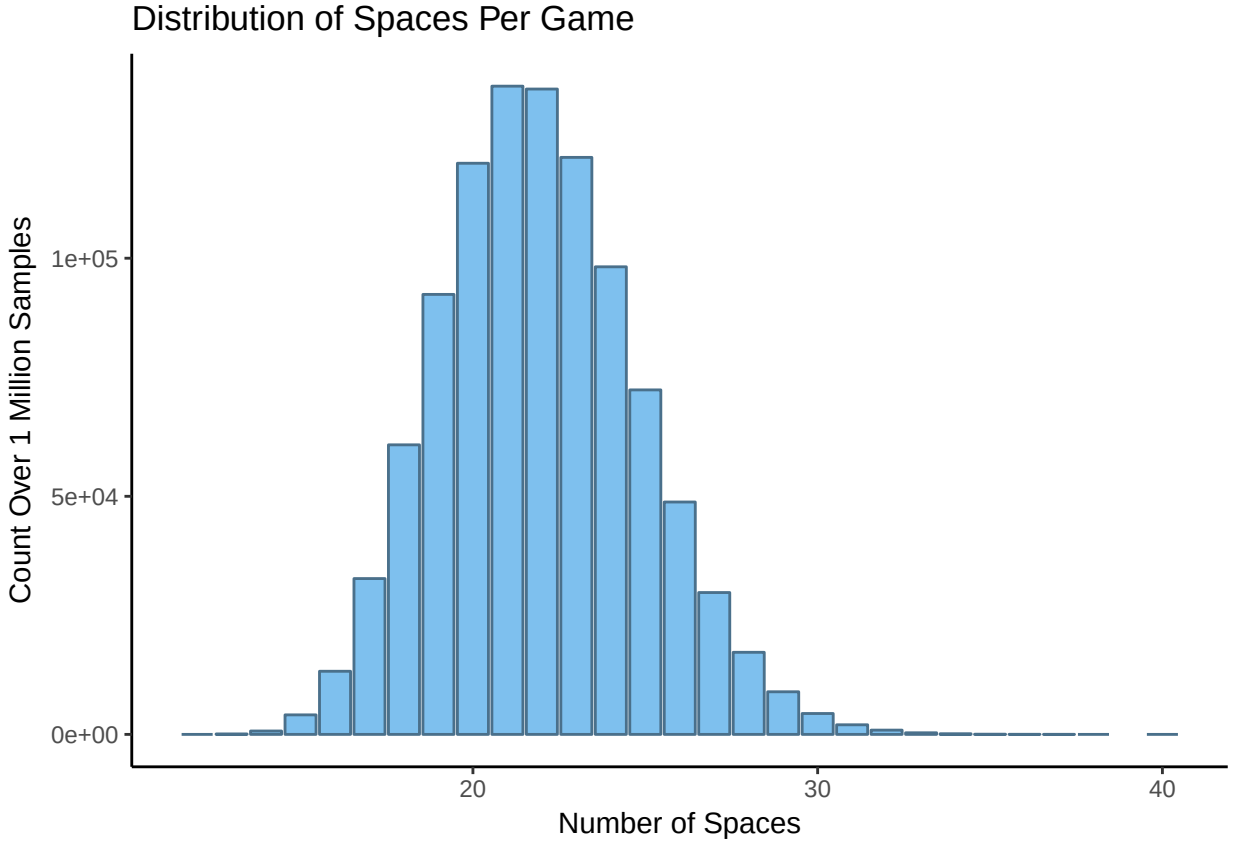
Gut Check: Number of Spaces Per Game

Since the transition matrix provided a way to calculate the theoretical expected number of spaces per game, checking the average number of spaces per game generated by the simulation offers a way to check how well the simulation is working.

```
mean_spaces <- summarise(results2, avg = mean(spaces_total))
mean_spaces
```

```
##        avg
## 1 21.93808
```

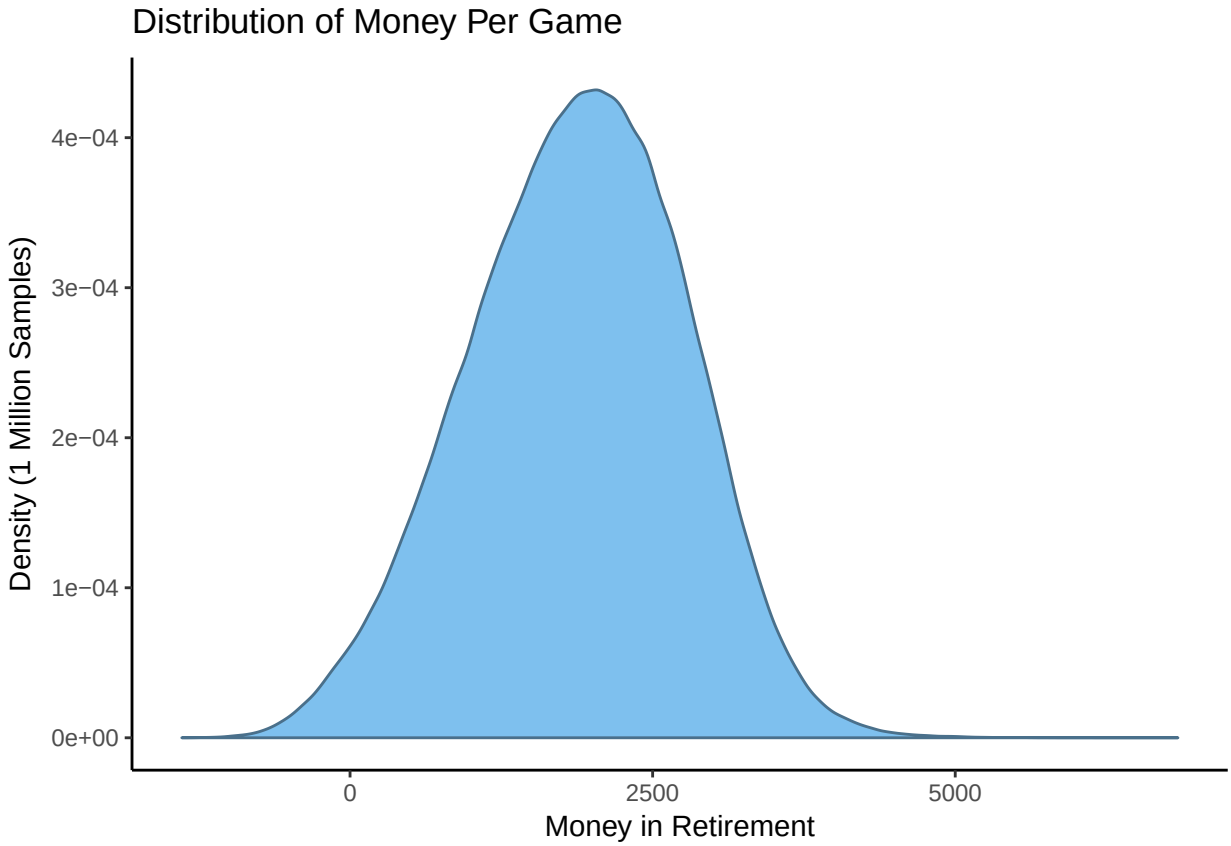
For this sample, the mean number of spaces is about 21.9, which comes reasonably close the the theoretical value of 22.2.



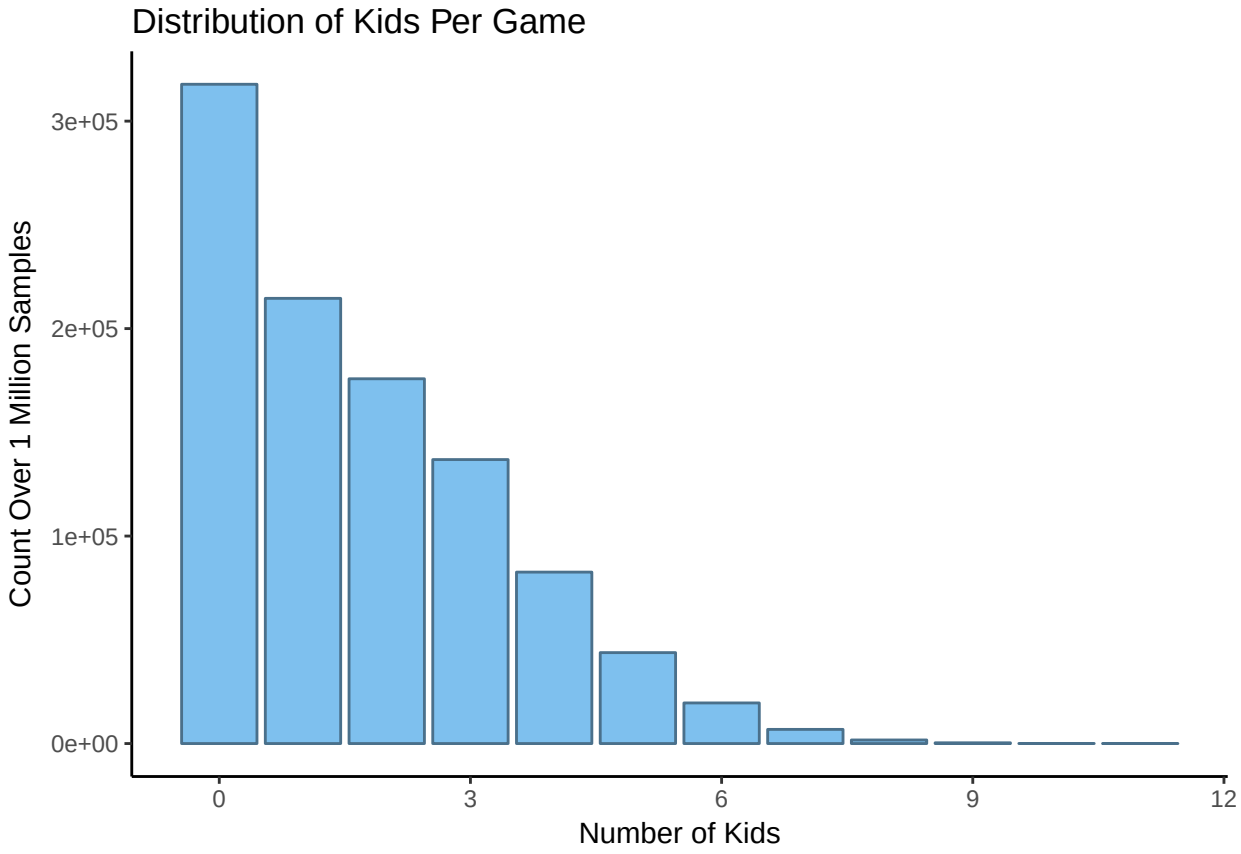
A graph of the distribution is also useful for checking the simulated mean. Here, we can see the theoretical mean of 22.2 is roughly the center of the simulated spaces per game distribution.

Money, Kids, Action Cards, and Houses

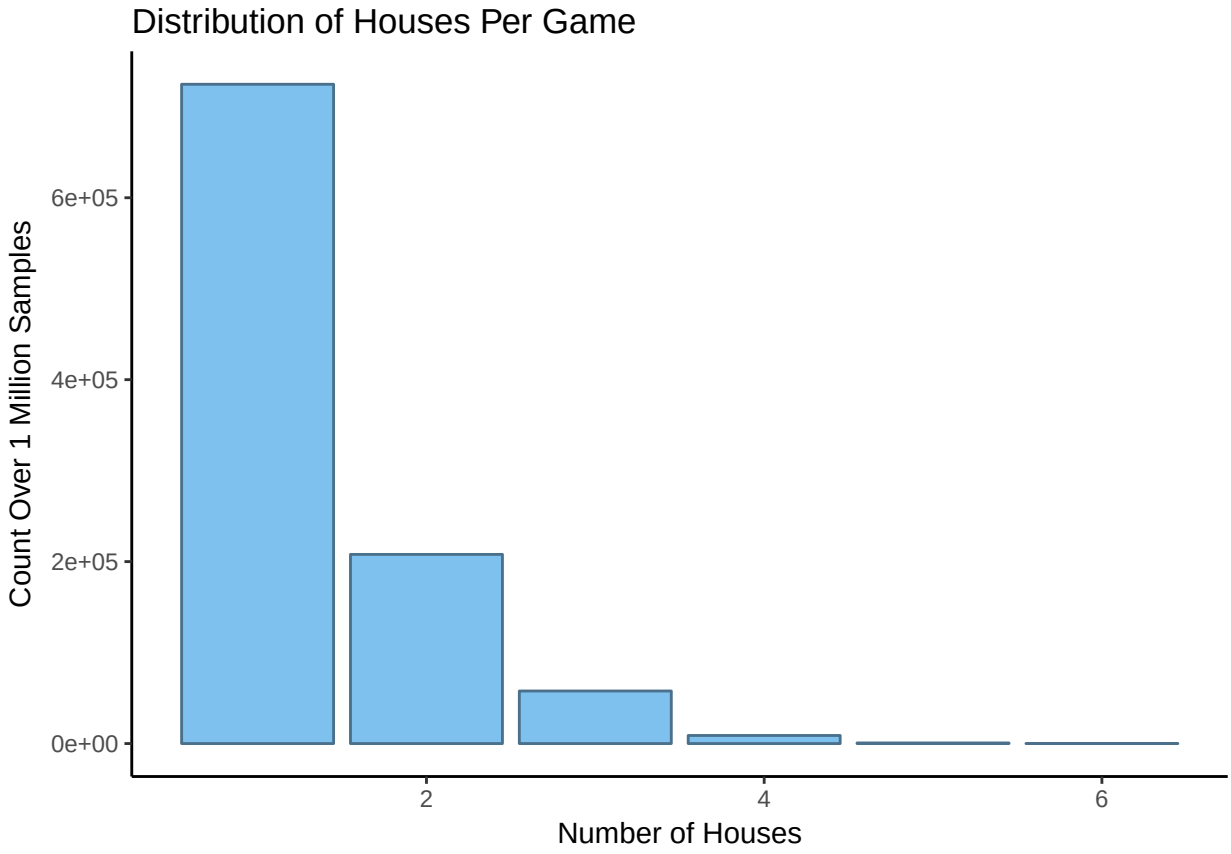
The simulation allows us to investigate game features beyond those included in the transition matrix. The following plots display the simulated distributions of money earned per game, kids collected per game, the number of houses purchased per game, and number of action cards drawn per game.



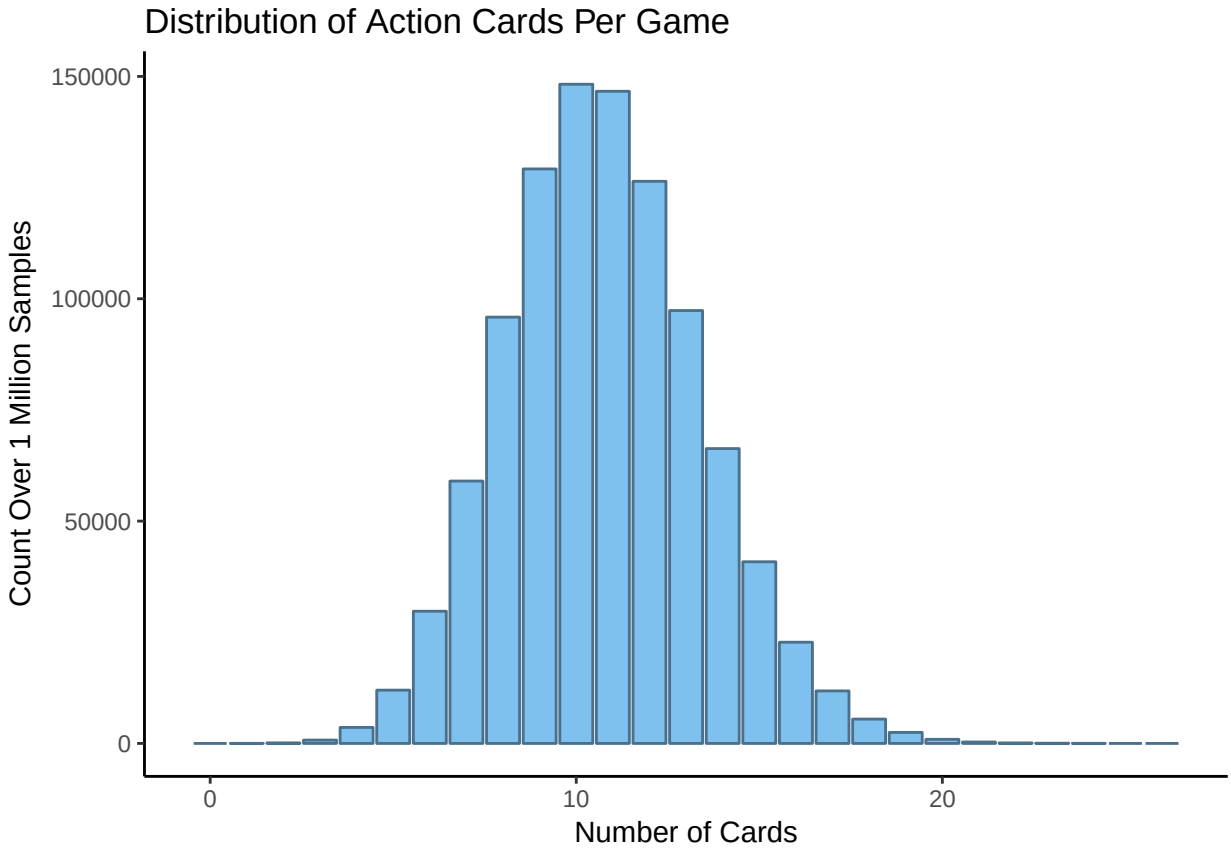
The distribution of money earned appears normal with a mean of approximately 1868.77K.



The distribution of kids collected per game is skewed right with a mean of approximately 1.71.



The distribution of houses purchased per game is skewed right with a mean of approximately 1.35.

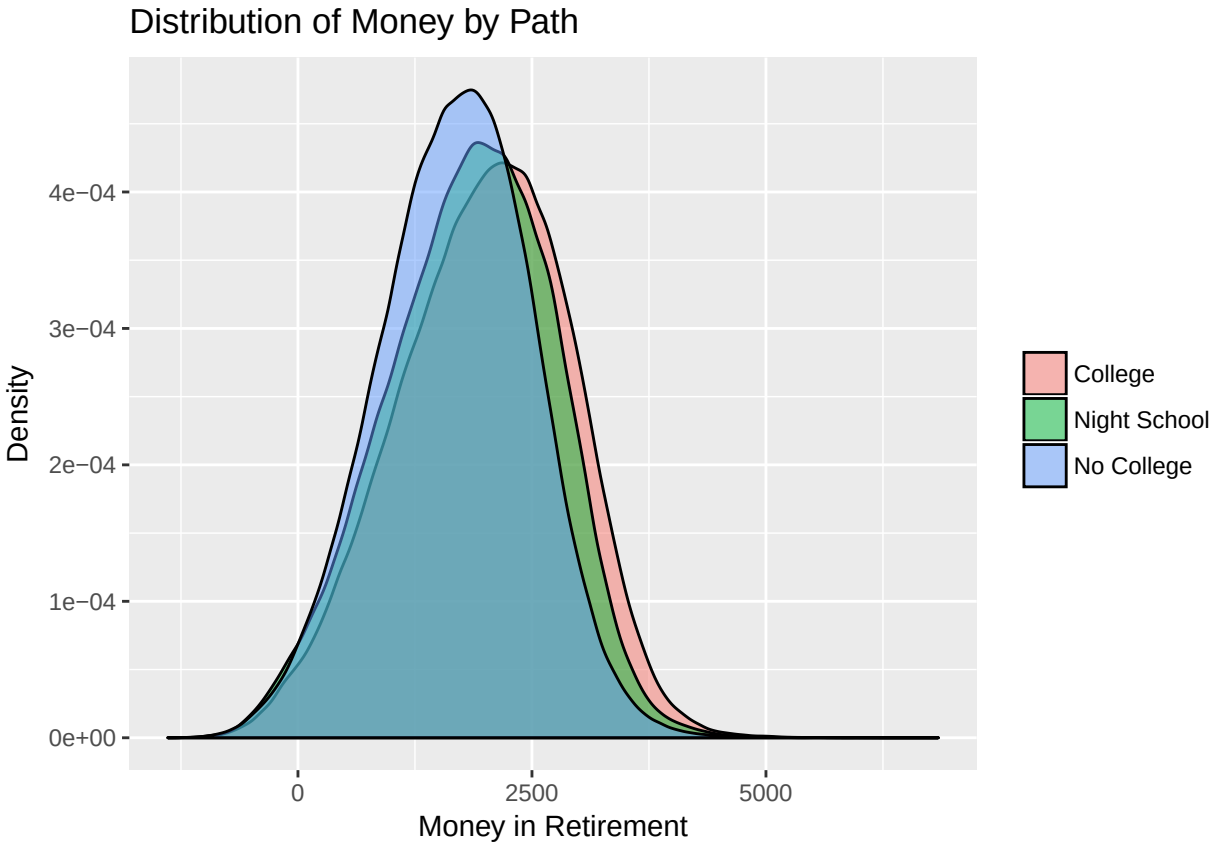


The distribution of action cards drawn per game appears normal with a mean of approximately 10.76.

Does College Pay Off?

Excluding the opportunity cost of taking a longer path to retirement (and risking a smaller retirement bonus), does having a higher college salary pay off?

```
## Warning: package 'bindrcpp' was built under R version 3.3.2
```




```
## # A tibble: 3 x 3
##   college_results mean_money money_sd
##   <chr>                <dbl>    <dbl>
## 1 College              1981.     911.
## 2 Night School         1832.     884.
## 3 No College           1681.     816.
```

By examination, it appears starting with the College Path or later taking the Night School path does result in higher earnings over the course of the game. So, taking a college path may offer the player a strategy for winning.

Future work

The simulation I created works well for visualizing basic aspects of gameplay. In the future, this project could be continued with work to decrease the reliance on assumptions that skew gameplay results. A better function for drawing cards that prevents redrawing of the same card would improve the reliability of simulation results. For this task, C++, Java, or another object oriented language with access to data structures like hash maps or linked lists would work better. Further, a function could be written to use results from this simulation to investigate the effects of retirement bonuses for finishing before others players will improve recommendations for gameplay strategies.

Appendix 1: Game Materials

INCLUDE PICTURES OF CARDS AND RULES HERE

Code Appendix 1: Making the Probability Matrix

```
board_matrix <- matrix(0, 135, 135)

#filling row 1
for (i in 1:22) {
  if (i + 1 != 12 && i + 1 != 13) {
  board_matrix[1, i + 1] <- 1/20
  }
}

#filling rows 2 to 10
for (i in 2:10) {
  board_matrix[i,12] <- (i - 1)/10
  for (j in (i + 1):11) {
    board_matrix[i,j] <- 1/10
  }
}

#filling row 11
board_matrix[11,12] <- 1

#filling row 12
for (i in 13:24) {
  if (i != 15 && i != 16) {
    board_matrix[12,i] <- 1/10
  }
}

#filling row 13
for (i in 14:25) {
  if (i != 15 && i != 16) {
    board_matrix[13,i] <- 1/10
  }
}

#filling rows 14 to 17
for (i in 14:17) {
  for (j in (i + 1):(i + 10)) {
    board_matrix[i,j] <- 1/10
  }
}

#filling rows 18 to 25
for (i in 18:25) {
  board_matrix[i,27] <- (i - 16)/10
  for (j in (i + 1):26) {
    board_matrix[i,j] <- 1/10
  }
}

#filling row 26
board_matrix[26,27] <- 1
```

```
#row 27, 28, 29
board_matrix[27, 28:37] <- 1/10
board_matrix[28, 29:38] <- 1/10
board_matrix[29, 30:39] <- 1/10

#filling rows 30 to 37
for (i in 30:37) {
  board_matrix[i, 39] <- (i - 28)/10
  for (j in (i + 1):38) {
    board_matrix[i,j] <- 1/10
  }
}

#filling row 38
board_matrix[38,39] <- 1

#filling row 39
board_matrix[39, 40:58] <- 1/20
board_matrix[39, 55] <- 2/20

#filling rows 40 to 48
for (i in 40:48) {
  for (j in (i + 1):48) {
    board_matrix[i,j] <- 1/10
  }
  for (j in 55:(i + 16)) {
    board_matrix[i,j] <- 1/10
  }
}

#filling rows 49 to 58
board_matrix[49, 50:59] <- 1/10
board_matrix[50, 51:60] <- 1/10
board_matrix[51, 52:61] <- 1/10
board_matrix[52, 53:62] <- 1/10
board_matrix[53, 54:63] <- 1/10
board_matrix[54, 55:64] <- 1/10
board_matrix[55, 56:65] <- 1/10
board_matrix[56, 57:66] <- 1/10
board_matrix[57, 58:67] <- 1/10
board_matrix[58, 59:68] <- 1/10

#filling rows 59 to 67
for (i in 59:67) {
  board_matrix[i, 68] <- (69 - i)/10
  for (j in (i + 1):67) {
    board_matrix[i,j] <- 1/10
  }
}

#filling in row 68
board_matrix[68, 69:78] <- 1/20
board_matrix[68, 79:88] <- 1/20
```

```
#filling in rows 69 to 76
for (i in 69:76) {
  board_matrix[i, 78] <- (i - 67)/10
  for (j in (i + 1):77) {
    board_matrix[i,j] <- 1/10
  }
}

#filling in row 77
board_matrix[77, 78] <- 1

#filling in row 78 to 95
board_matrix[78, 86:95] <- 1/10
board_matrix[79, 80:89] <- 1/10
board_matrix[80, 81:90] <- 1/10
board_matrix[81, 82:91] <- 1/10
board_matrix[82, 83:92] <- 1/10
board_matrix[83, 84:93] <- 1/10
board_matrix[84, 85:94] <- 1/10
board_matrix[85, 86:95] <- 1/10
board_matrix[86, 87:96] <- 1/10
board_matrix[87, 88:97] <- 1/10
board_matrix[88, 89:98] <- 1/10
board_matrix[89, 90:99] <- 1/10
board_matrix[90, 91:100] <- 1/10
board_matrix[91, 92:101] <- 1/10
board_matrix[92, 93:102] <- 1/10
board_matrix[93, 94:103] <- 1/10
board_matrix[94, 95:104] <- 1/10
board_matrix[95, 96:105] <- 1/10

#filling in rows 97 to 103
for (i in 96:103) {
  board_matrix[i,105] <- (i - 94)/10
  for (j in (i + 1):104) {
    board_matrix[i,j] <- 1/10
  }
}

#filling in row 104
board_matrix[104, 105] <- 1

#filling in rows 105 to 112
board_matrix[105, 106:112] <- 1/20
board_matrix[105, 120:122] <- 2/20
board_matrix[105, 113:119] <- 1/20

board_matrix[106, 107:112] <- 1/10
board_matrix[106, 120:123] <- 1/10

board_matrix[107, 108:112] <- 1/10
board_matrix[107, 120:124] <- 1/10
```

```
board_matrix[108, 109:112] <- 1/10
board_matrix[108, 120:125] <- 1/10

board_matrix[109, 110:112] <- 1/10
board_matrix[109, 120:126] <- 1/10

board_matrix[110, 111:112] <- 1/10
board_matrix[110, 120:127] <- 1/10

board_matrix[111, 112] <- 1/10
board_matrix[111, 120:128] <- 1/10

board_matrix[112, 120:129] <- 1/10

#filling in rows 113 to 125
for (i in 113:125) {
  for (j in (i + 1):(i + 10)) {
    board_matrix[i,j] <- 1/10
  }
}

#filling in rows 126 to 134
for (i in 126:133) {
  board_matrix[i,135] <- (i - 124)/10
  for (j in (i + 1):134) {
    board_matrix[i,j] <- 1/10
  }
}
board_matrix[134,135] <- 1
#Setting the final recurrent class, row 135
board_matrix[135,135] <- 1
```

Code Appendix 2: Simulation

Sub-Appendix: Helper Functions

```
#Helper Functions

#Increment value by 1
increment <- function(object) {
  eval.parent(substitute(object <- object + 1))
}


#Functions for the board

#Spin function
#Generates a value for the spinner
spin <- function() {
  value <- sample(x = 1:10, size = 1)
  return(value)
```

```
}

#Highest Spinner Function
#Some action cards require players to spin against each other. This function
#returns a true/false statement. True implies the player won the spinoff, false implies the player lost
highest_spinner <- function() {
  place <- sample(x = 1:4, size = 1)
  if (place == 1) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

#Red/black function
#Some action cards and house cards require the player to spin for a color (red or black)
#This function returns the color
red_black <- function(){
  color <- sample(x = c("red", "black"), size = 1)
  return(color)
}

#Spin-to-win function
#On these spaces, every player chooses a number from 1-10 without replacement
#(the current player gets to choose 2 numbers),
#and the spinner is spun until one of these numbers is selected. The player who chose that number
#recieves 200K
spin_to_win <- function(){
  while (TRUE) {
  player_numbers <- sample(1:10, size = 5, replace = FALSE)
  winning_numbers <- sample(player_numbers, size = 2, replace = FALSE)
  spin <- spin()

  for (i in 1:5) {
    if (spin == player_numbers[i]) {
      if (spin == winning_numbers[1] | spin == winning_numbers[2]) {
        return(200)
      } else {
        return(0)
      }
    }
  }
  }
}

#Career card functions
#Every career has a salary attached that gets paid to player every time a "payday" space is landed on

#Choose a college career function
college <- function(){
  salary <- sample(x = c(130, 120, 110, 100, 100, 100, 100, 80), size = 1)
  return(salary)
}
```

```r
#Choose a non-college career function
career <- function(){
  salary <- sample(x = c(80, 100, 100, 70, 60, 50, 50, 50, 50), size = 1)
  return(salary)
}

#Select a house function
house <- function(){
  return(sample(x = c(250, 250, 300, 300, 600, 600, 600, 350, 100, 100, 100, 150, 200, 200),
                size = 1))
}

#Draw a card function
#The deck of action cards contains a number of ways for a player to win or lose money
#This function takes player's current money and salary and returns an updated money and salary

#Pass in salary and money
draw_card <- function(money, salary, college_status){
  #key:
  #1 correpsonds with "highest spinner gets paid"
  #2 corresponds with "red/black spin determines result"
  #3 corresponds with "payment by number on spinner"
  #4 corresponds with "collect money from bank or another player"
  #5 corresponds with "pay money to bank"
  #6 corresponds with "bonus payday"
  #7 corresponds with "change career"

  #Determine card type
  card <- sample(x = (c(rep(1,15), 2, rep(3,9), rep(4,14), rep(5, 13), 6, rep(7,2))), size = 1)

  #Do card action by card type

  #Card type 1
  if (card == 1) {
    #return 0 if not highest spinner
    if (highest_spinner() == FALSE) {
      return(c(money, salary))
    }
    #Drawn specific card if highest spinner (14 possible cards have "highest spinner" feature)
    else {
      #key
      #0 corresponds to "bank pays 10K times spin"
      #20, 50, 70, 100 corresponds to "bank payment of 10K times this value"
      card <- sample(x = c(20, rep(50, 4), rep(70, 5), rep(100, 2), rep(0, 3)), size = 1)

      #spin for 0
      if (card == 0) {
        spin_curr <- spin()
        card <- 10*spin_curr
      }

      money <- money + card
      return(c(money, salary))
```

```
  }
}

#card type 2
else if (card == 2) {
  #subtract money if red
  if (red_black() == "red") {
    money <- money - 30
    return(c(money, salary))
  }
  #else pay for black
  money <- money + 30
  return(c(money, salary))
}

#card type 3
else if (card == 3) {
  #These cards are idiosyncratic. Some pay different amounts for 1-5 and 6-10 spins.
  #Others have more detailed divisions. As above, I've numbered the cards and created breakout
  #"if" statements for each card.

  #Key:
  #1 corresponds with "pay 10K times spin"
  #2 corresponds with "1-5 pay 20K, 6-10 pay 50K"
  #3 corresponds with "1-4 pay 10K, 5-8 pay 20K, 9-10 pay 30K"
  #4 corresponds with "1-5 pay 50K, 6-10 pay 100K"
  #5 corresponds with "1-3 pay 50K, 4-7 pay 80K, 8-10 pay 100K"

  #Determine card type
  card <- sample(c(rep(1,3), 2, 3, rep(4, 3), 5), size = 1)

  #return value by card type
  if (card == 1) {
    spin_curr <- spin()
    money <- money + 10*spin_curr
    return(c(money, salary))
  }

  else if (card == 2) {
    if (spin() <= 5) {
      money <- money + 20
      return(c(money, salary))
    } else {
      money <- money + 50
      return(c(money, salary))
    }
  }

  else if (card == 3) {
    spin <- spin()
    if (spin <= 4) {
      money <- money + 10
      return(c(money, salary))
```

```
    } else if (spin >= 5 & spin <= 8) {
      money <- money + 20
      return(c(money, salary))
    } else {
      money <- money + 30
      return(c(money, salary))
    }
  }

  else if (card == 4) {
    if (spin() <= 5) {
      money <- money + 50
      return(c(money, salary))
    } else {
      money <- money + 100
      return(c(money, salary))
    }
  }

  else {
    spin <- spin()
    if (spin <= 3) {
      money <- money + 40
      return(c(money, salary))
    } else if (spin >= 4 & spin <= 7) {
      money <- money + 80
      return(c(money, salary))
    } else {
      money <- money + 100
      return(c(money, salary))
    }
  }
}

#card type 4
else if (card == 4) {
  card <- sample(x = c(20, rep(40, 4), rep(50, 3), rep(70, 2), 80, 100, 120, 200),
                         size = 1)
  money <- card + money
  return(c(money, salary))
}

#card type 5
else if (card == 5) {
  card <- sample(x = c(rep(20, 2), rep(30, 4), rep(50, 6), 70), size = 1)
  money <- card - money
  return(c(money, salary))
}

#card type 6
#Pays zero if no salary yet
else if (card == 6) {
  money <- money + salary
```

```
    return(c(money, salary))
  }

  #card type 7
  #Does nothing if no salary yet
  else {
    if (salary == 0) {
      return(c(money, salary))
    }

    if (college_status == TRUE) {
      salary <- college()
    } else {
      salary <- career()
    }

    return(c(money, salary))
  }
}


#Functions for retirement:
#Once a player hits retirement, a number of game features determine final life earnings

#Sell house function
#House sale price in retirement determined by color of spin after card drawn
sell_house <- function(house_price){
if (house_price == 100) {
  red <- 80
  black <- 150
} else if (house_price == 150) {
  red <- 120
  black <- 200
}  else if (house_price == 200) {
  red <- 180
  black <- 300
} else if (house_price == 250) {
  red <- 200
  black <- 300
} else if (house_price == 300) {
  red <- 250
  black <- 380
}else if (house_price == 350) {
  red <- 300
  black <- 500
} else if (house_price == 600) {
  red <- 600
  black <- 750
}
  spin_color <- red_black()
  if (spin_color == "red") {
    return(red)
  } else {
```

```
    return(black)
  }
}
```

Sub-Appendix: Choosing Paths and Iterating the Board

```
#Determine paths function
#Randomly selects paths player will take at forks in game
#Index:
#paths_vect[1] = TRUE if College Path
#paths_vect[2] = TRUE if Night School Path
#paths_vect[3] = TRUE if Family Path
#paths_vect[4] = TRUE if Risky Path

paths <- function() {
  paths_vect <- rep(FALSE, 4)

  if (runif(1, min = 0, max = 1) < 0.5) {
    paths_vect[1] <- TRUE
  }

  if (runif(1, min = 0, max = 1) < 0.5) {
    paths_vect[2] <- TRUE
  }

  if (runif(1, min = 0, max = 1) < 0.5) {
    paths_vect[3] <- TRUE
  }

  if (runif(1, min = 0, max = 1) < 0.5) {
    paths_vect[4] <- TRUE
  }

  return(paths_vect)

}

#play_game function
#Iterates the board and returns a final winnings sum, number of spaces landed on total,
#and paths taken
play_game <- function(college_path, night_school_path, family_path, risky_path) {

  #Tracks current place on board
  current_space <- 0

  #Tracks total number of spaces landed on
  #Start at 1 to count start as a space
  #(Do this because the theoretical time to absorption counts the starting space 0)
  spaces_total <- 1

  #Current career's salary
  salary <- 0
```

```r
#Tracks if career card is from college salaries or career salaries deck
college_status <- FALSE

#Tracks current spin value
curr_spin <- 0

#Tracks current money (1K units)
money <- 200

#Tracks number of action cards drawn (each worth 100K at end of game)
action_card_count <- 0

#Tracks number of kids "collected" during the game (worth 50K each at end of game)
kid_count <- 0

#Tracks purchase price of all houses, updated to a vector of a new length each time
#house is purchased
house_prices <- 0

#Taking college path
if (college_path == TRUE) {
  curr_spin <- spin()
  current_space <- current_space + curr_spin
  #Takes to stop at 11
  while (current_space < 11) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)

    curr_spin <- spin()
    current_space <- current_space + curr_spin
  }

  current_space <- 11
  increment(spaces_total)
  #Assign college salary
  salary <- college()

  #Spin until path merge with career
  curr_spin <- spin()
  if (curr_spin == 1) {
    current_space <- curr_spin + current_space

    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    curr_spin <- curr_spin + 2
    current_space <- current_space + curr_spin
```

```
    }
  }

  #Taking career path

  else {
    current_space <- 13
    salary <- career()


    while (current_space < 15) {
    curr_spin <- spin()
    current_space <- curr_spin + current_space

    if (current_space == 13) {
      card <- draw_card(money, salary, college_status)
      money <- card[1]
      salary <- card[2]
      increment(action_card_count)
      increment(spaces_total)
    }

    else if (current_space == 14) {
      increment(spaces_total)
    }
  }
    money <- money + salary

  }

#Paths join for spaces 15-38

# Moves until forced stop at 26 (get married)

  while (current_space < 26) {

    #Space 20 = spin to win
    if (current_space == 20) {
      money <- spin_to_win() + money
      increment(spaces_total)
    }

    #Draw action card for all spaces except payday on 17, 22 and spin to win on 20
    else if (current_space != 17 & current_space != 22) {
      card <- draw_card(money, salary, college_status)
      money <- card[1]
      salary <- card[2]
      increment(action_card_count)
      increment(spaces_total)
    } else {
      increment(spaces_total)
    }
```

```
    curr_spin <- spin()
    current_space <- current_space + curr_spin

  }

  #Pay for 2 paydays passed before 26
  money <- salary + salary + money
  current_space <- 26
  increment(spaces_total)



# Moves until forced stop at 38 (life path or night school path)

    #spin before start loop
    curr_spin <- spin()
    current_space <- current_space + curr_spin

    while (current_space < 38) {

    #Space 34 = spin to win
    if (current_space == 34) {
      money <- spin_to_win() + money
      increment(spaces_total)
    }

    #36 = draw house card
    else if (current_space == 36) {
      house_prices <- house()
      money <- money - house_prices
      increment(spaces_total)
    }

    #Draw action card for all spaces except payday on 32, spin to win on 34, house on 36
    else if (current_space != 32) {
      card <- draw_card(money, salary, college_status)
      money <- card[1]
      salary <- card[2]
      increment(action_card_count)
      increment(spaces_total)
    } else {
      increment(spaces_total)
    }

    curr_spin <- spin()
    current_space <- current_space + curr_spin

  }

  #Pay for payday on 32
  money <- money + salary

  #Force stop on 38
```

```
current_space <- 38
increment(spaces_total)

#Fork for spaces 34-54: night school path or life path

if (night_school_path) {

  #pay 100K "for night school"
  money <- money - 100
  #get new College Career
  salary <- college()
  college_status <- TRUE

  #spin before start loop
  curr_spin <- spin()
  current_space <- current_space + curr_spin

  while (current_space < 48) {

  #Draw action card for all spaces except payday on 44
  if (current_space != 44) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    increment(spaces_total)
  }

  curr_spin <- spin()
  current_space <- current_space + curr_spin

  }

  #pay for payday passed (space 44)
  money <- money + salary
  #increment current_space to merge paths
  current_space <- current_space + 6
}

else {

  #increment current space to match up with correct path
  current_space <- 47

  #spin before start loop
  curr_spin <- spin()
  current_space <- current_space + curr_spin

  while (current_space < 53) {

  #Draw action card for all spaces except payday on 52
```

```
  if (current_space != 52) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    increment(spaces_total)
  }

  curr_spin <- spin()
  current_space <- current_space + curr_spin
  }

  #Pay for payday passed at 52
  money <- money + salary
}

#Paths merge for spaces 54 - 67
  while (current_space < 67) {

  #Space 61 = spin to win
  if (current_space == 61) {
    money <- spin_to_win() + money
    increment(spaces_total)
  }

  #Spaces 56, 64 = draw house card
  else if (current_space == 56 | current_space == 64) {

    if (length(house_prices) == 1) {
      if (house_prices == 0) {
        house_prices <- house()
        money <- money - house_prices
      } else {
        new_house <- house()
        money <- money - new_house
        house_prices <- c(house_prices, new_house)
      }
    } else {
      new_house <- house()
      money <- money - new_house
      house_prices <- c(house_prices, new_house)
    }

    increment(spaces_total)
  }

  #Increment baby count for space 59
    else if (current_space == 59) {
      kid_count <- kid_count + 2
      increment(spaces_total)
    }
```

```
  #Draw action card for all spaces except payday on 58, 65; spin to win on 61;
    #house on 56, 64; kid on 59
  else if (current_space != 58 & current_space != 65) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    increment(spaces_total)
  }

  curr_spin <- spin()
  current_space <- current_space + curr_spin
  }

#pay for paydays passed
money <- salary + salary + money

#force stop at 67
current_space <- 67
increment(spaces_total)

if (family_path) {

  #spin before entering while loop
  curr_spin <- spin()
  current_space <- current_space + curr_spin

  while (current_space < 77) {

    #Spaces 71, 74, 75 = 1 kid
    if (current_space == 71 | current_space == 74 | current_space == 75) {
      increment(kid_count)
      increment(spaces_total)
    }

    #Spaces 69, 71 = 2 kids
    else if (current_space == 69 | current_space == 71) {
      kid_count <- kid_count + 2
      increment(spaces_total)
    }

    #Space 70 = house
    else if (current_space == 70) {
      if (length(house_prices) == 1) {
      if (house_prices == 0) {
        house_prices <- house()
        money <- money - house_prices
      } else {
        new_house <- house()
        money <- money - new_house
        house_prices <- c(house_prices, new_house)
```

```
      }
    } else {
      new_house <- house()
      money <- money - new_house
      house_prices <- c(house_prices, new_house)
    }
      increment(spaces_total)
    }

    #Spaces 68, 76 = draw action card
    else if (current_space == 68 | current_space == 76) {
      card <- draw_card(money, salary, college_status)
      money <- card[1]
      salary <- card[2]
      increment(action_card_count)
      increment(spaces_total)
    } else {
        increment(spaces_total)
    }

    curr_spin <- spin()
    current_space <- curr_spin + current_space
  }

  #Pay for payday passed
  money <- money + salary

  #Force stop at 77 (incremented to join with life path)
  current_space <- 84
  increment(spaces_total)

  curr_spin <- spin()
  if (curr_spin >= 4 & curr_spin <= 6) {
    increment(kid_count)
  } else if (curr_spin == 7 | curr_spin == 8) {
    kid_count <- kid_count + 2
  } else if (curr_spin == 9 | curr_spin == 10) {
    kid_count <- kid_count + 3
  }

  #Spin for next turn to be ready to merge with life path
  curr_spin <- spin()
  current_space <- current_space + curr_spin
}

#Take life path
else {

  #Increment spaces to join path
  current_space <- 77

  #spin before starting loop
  curr_spin <- spin()
```

```
  current_space <- curr_spin + current_space

  while (current_space < 85) {

    #Draw action card for all spaces except payday at 83
    if (current_space != 83) {
      card <- draw_card(money, salary, college_status)
      money <- card[1]
      salary <- card[2]
      increment(action_card_count)
      increment(spaces_total)
    } else {
        increment(spaces_total)
    }
     curr_spin <- spin()
     current_space <- curr_spin + current_space

  }

  #Pay for payday passed
  money <- money + salary
}

#Paths merge for spaces 85-104
while (current_space < 104) {

  #Space 86, 96 = spin to win
  if (current_space == 86 | current_space == 96) {
    money <- spin_to_win() + money
    increment(spaces_total)
  }

  #Spaces 91, 100 = draw house card
  else if (current_space == 91 | current_space == 100) {

    if (length(house_prices) == 1) {
      if (house_prices == 0) {
        house_prices <- house()
        money <- money - house_prices
      } else {
        new_house <- house()
        money <- money - new_house
        house_prices <- c(house_prices, new_house)
      }
    } else {
      new_house <- house()
      money <- money - new_house
      house_prices <- c(house_prices, new_house)
    }

    increment(spaces_total)
  }
```

```
  #Increment baby count for spaces 89, 93
    else if (current_space == 89 | current_space == 93) {
      increment(kid_count)
      increment(spaces_total)
    }

  #Draw action card for all spaces except paydays on 92, 102 and actions above
  else if (current_space != 92 & current_space != 102) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    increment(spaces_total)
  }

  curr_spin <- spin()
  current_space <- current_space + curr_spin
}

#Pay for two paydays passed
money <- salary + salary + money

#Force stop on 104
current_space <- 104
increment(spaces_total)

#Paths split to Risky and safe

if (risky_path) {
  #spin before entering loop
  curr_spin <- spin()
  current_space <- current_space + curr_spin

  while (current_space < 111) {

    #recieve 100K for spaces 106, 109
    if (current_space == 106 | current_space == 109) {
      money <- money + 100
    }

    #lose 100K for spaces 108, 110
    else if (current_space == 108 | current_space == 110) {
      money <- money - 100
    }

    #draw action card for spaces 105, 107
    else if (current_space == 105 | current_space == 107) {
     card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
```

```
    increment(spaces_total)
    }
    #increment for payday space
    else {
    increment(spaces_total)
    }
  curr_spin <- spin()
  current_space <- current_space + curr_spin
  }

  #Increment bank for payday space passed
  money <- money + salary

  #Increment to merge with other path
  current_space <- current_space + 7

}

#Take Safe Path
else {

  #Increment to skip risky path
  current_space <- 111

  #roll before loop
  curr_spin <- spin()
  current_space <- current_space + curr_spin

  while (current_space < 119) {
    #Draw action cards for all spaces except payday on 118
    if (current_space != 119) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
    }
    #increment for payday space
    else {
    increment(spaces_total)
    }
  curr_spin <- spin()
  current_space <- current_space + curr_spin
  }

  #pay for payday passed
  money <- money + salary
}

#Path merges until retirement
while (current_space < 134) {
  #Spaces 120, 131 = spin to win
  if (current_space == 120 | current_space == 131) {
```

```
    win <- spin_to_win()
    money <- win + money
    increment(spaces_total)
  }

  #Space 123 = draw house card
  else if (current_space == 91 | current_space == 100) {

    if (length(house_prices) == 1) {
      if (house_prices == 0) {
        house_prices <- house()
        money <- money - house_prices
      } else {
        new_house <- house()
        money <- money - new_house
        house_prices <- c(house_prices, new_house)
      }
    } else {
      new_house <- house()
      money <- money - new_house
      house_prices <- c(house_prices, new_house)
    }

    increment(spaces_total)
  }

  #Draw action card for all spaces except paydays on 124, 129 and actions above
  else if (current_space != 124 & current_space != 129) {
    card <- draw_card(money, salary, college_status)
    money <- card[1]
    salary <- card[2]
    increment(action_card_count)
    increment(spaces_total)
  } else {
    increment(spaces_total)
  }

  curr_spin <- spin()
  current_space <- current_space + curr_spin
}

#Pay for two paydays passed
money <- salary + salary + money

#Note: In the final output, retirement is not considered a space.

#Calculate retirement earnings

#action cards pay 100K for each picked up during game
money <- money + 100*action_card_count

#calculate sale price of all houses
  #initilize variable to track money earned from houses
```

```
    house_profit <- 0

    #look up price of each house
    num_houses <- length(house_prices)

    if (num_houses > 1) {
    for (i in 1:num_houses) {
      house_profit <- house_profit + sell_house(house_prices[i])
    }} else {
      if (house_prices != 0) {
        house_profit <- sell_house(house_prices)
      }
    }

    money <- house_profit + money

  #Calculate money from kids
  money <- money + 50*kid_count

  #Returns: total money earned, finishing salary, number of spaces landed on,
  #number of action cards drawn, number of kids, profit from houses, number of houses, college_path,
  #night_school_path, family_path, risky_path
  return(c(money, salary, spaces_total, action_card_count, kid_count,
           house_profit, length(house_prices), college_path, night_school_path, family_path, risky_path

  }
```

Sub-Appendix 3: Running Simulations

```
num_sim <- 1000000
results <- matrix(0, num_sim, 11)
colnames(results) <- c("money", "salary", "spaces_total", "action_card_count", "kid_count",
           "house_profit", "number_of_houses", "college_path", "night_school_path", "family_path", "risk
for (i in 1:num_sim) {
  path <- paths()
  results[i,] <- play_game(path[1], path[2], path[3], path[4])
}

#Optional: save simulation results
#write.csv(results, "results.csv")
```

Sources

Dobrow, Robert P. 2014. *Probability: With Applications and R*. Wiley.

———. 2016. *Introduction to Stochastic Processes with R*. Wiley.

Hayes, Brian. 2013. "First Links in the Markov Chain: Probability and Poetry." *First Links in the Markov Chain*. Harvard SEAS. http://bit-player.org/wp-content/extras/markov/#/.

———. 2017. "First Links in the Markov Chain." *American Scientist*. American Scientist. http://www.americanscientist.org/article/first-links-in-the-markov-chain.