R Notebook

```
## Warning: package 'tidyverse' was built under R version 3.3.2
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
## Warning: package 'ggplot2' was built under R version 3.3.2
## Warning: package 'purrr' was built under R version 3.3.2
## Warning: package 'dplyr' was built under R version 3.3.2
## Conflicts with tidy packages ----------------------------------------------
## filter(): dplyr, stats
## lag():    dplyr, stats
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

Transition Matrix

The board game Life has 134 spaces culminating in a absorbing state ("retirement"). Some interesting things happen along the board:

1. Players have four opportunities to choose between two path options. Each path has consequences and opportunities related to game play.
   (a) College Path or Career path: College path is longer but comes with higher income opportunities.
   (b) Life Path or Night School Path: Night school path is longer but comes with opportunity to draw new, potentially higher paying career card.
   (c) Family Path or Career Path: Family path is longer but comes with opportunity for kids and a house.
   (d) Safe Path or Risky Path: Paths are equal length. Risky path has alternating spaces for gaining or losing money. Chance for big payoff or big loss.
2. Six spaces with that force the player to halt. So, leading up to these spaces, the probability approaches one that the player will land on that space in the next turn as the player approaches the space.

Making 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
```

Matrix Eigenvalues

As expected, the long term distribution is absorbing. We expect to find the player at the end of the game as the number of turns heads to infinity.

```
r <- eigen(t(board_matrix))
V <- r$vectors
lambda <- r$values
lambda
```

```
##   [1] 1.0 0.1 0.1 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
##  [18] 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
##  [35] 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
##  [52] 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
##  [69] 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
##  [86] 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
## [103] 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
## [120] 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
```

```
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 1
```

Travel times

How long, in turns, does it take a player on average to get to the end of the board? We can answer this question by making state 134 absorbing.

```
#Let state b be absorbing
Q <- board_matrix[1:134, 1:134]

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

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

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

Therefore, we expect the average game to take 22 steps.

Average money earned per iteration

Assumptions: cards drawn are put back

The following assumptions make this game easier to code in R. If I had chosen to create the game in something more flexible like Java or C++, these issues could have been avoided with a set of linked lists or a hashmap, that sort of thing. However, for the sake of simplicity and since our course was taught in R, I decided to go with R when beginning the simulation. In hindsight, I should have known better.

1. Cards drawn are put back and the deck is shuffled. Note: This is not true to gameplay, since cards would probably go to the bottom of the pile and career/house cards stay with the player until the end of the game or a career switch. The biggest effect of this on gameplay is that every profession is open every game. In truth, as players draw career cards, the last player to draw from a stack will have fewer possible professions than the first player. Another effect is that, even though players technically hold onto action cards after drawing them, all action cards are in play during the full game.
2. Simulation run with only one player on the board. I.E. Some elements of player interaction have been left out. Specifically, in some instances, one player pays another player money through action cards. Since only one player is considered in the simulation, there is never an instance where money is subtracted from the player's total through payment to another player.
3. Eliminated retirement order payouts. Players get cash payouts based on the order they arrive in retirement. Since the simulation only has one player on the board at a time, this game feature was not considered in cash payouts.
4. If a player draws an action card depending on a salary before drawing a career card (spaces 1-10), the simulation assumes they will draw a new action card.

For report: include an example of each type of card

```
#Helper Functions

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


#Functions for the board
```

```r
#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
```

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

#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)
  }
}
```

During game need to record: each pathway chosen salary career or college card boolean purchase price of house number of action cards drawn (worth 100K each at end of game)

Assumption: A player chooses to the Night School path randomly, without considering current career salary. (I think in real play, this asssumption would be false. If a player chooses the college path at start, I doubt they will choose the Night School path later.)

```
#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
```

```r
#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

#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

  }
```

Simulate the game a bunch of times, commented out because results of simulation are saved

```
#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", "r
#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")
```

Next steps:

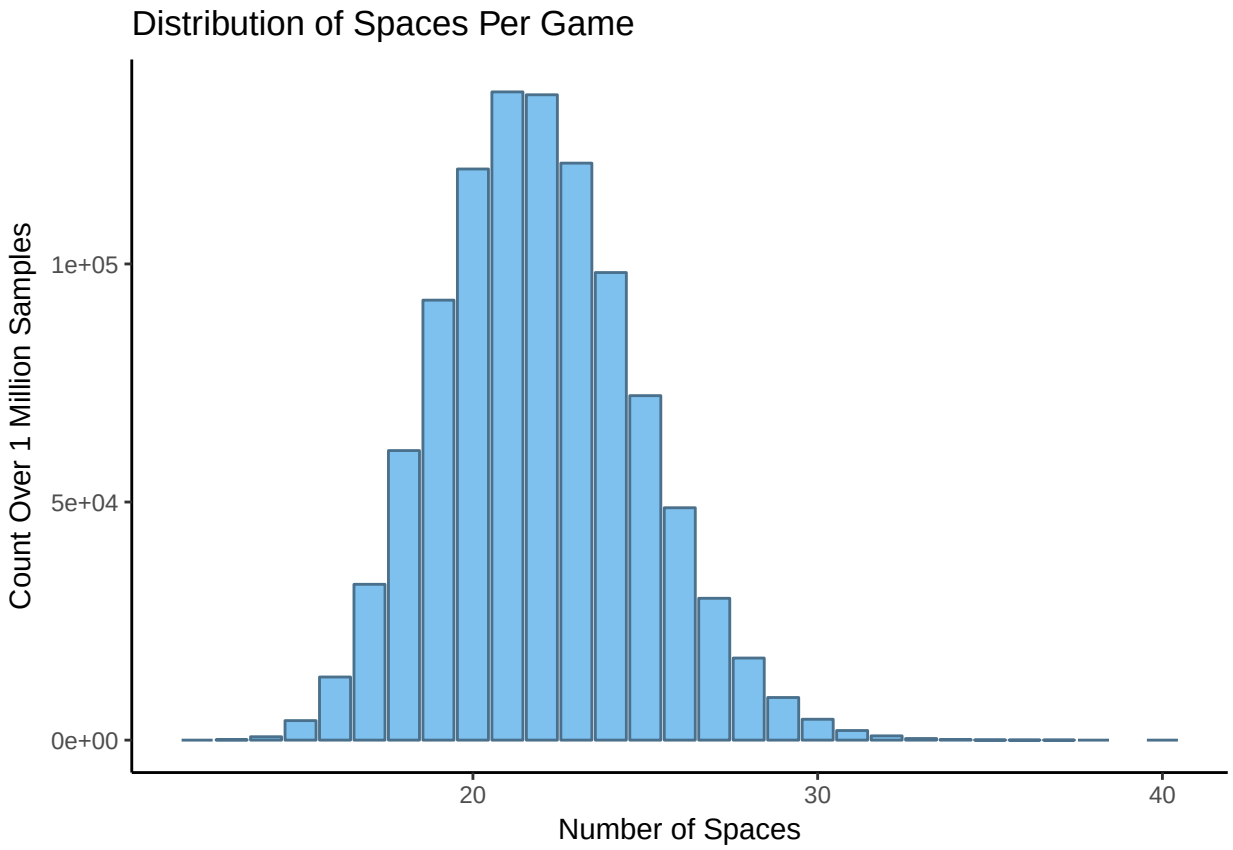
Gut check -> does the expected number of spaces make sense with the theoretical expectation?

```
results2 <- read.csv("results.csv")

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

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

```
spaces_plot <- ggplot(results2, aes(spaces_total)) + geom_bar(color = "skyblue4", fill = "skyblue2") + t
spaces_plot
```



Yes, looks good.

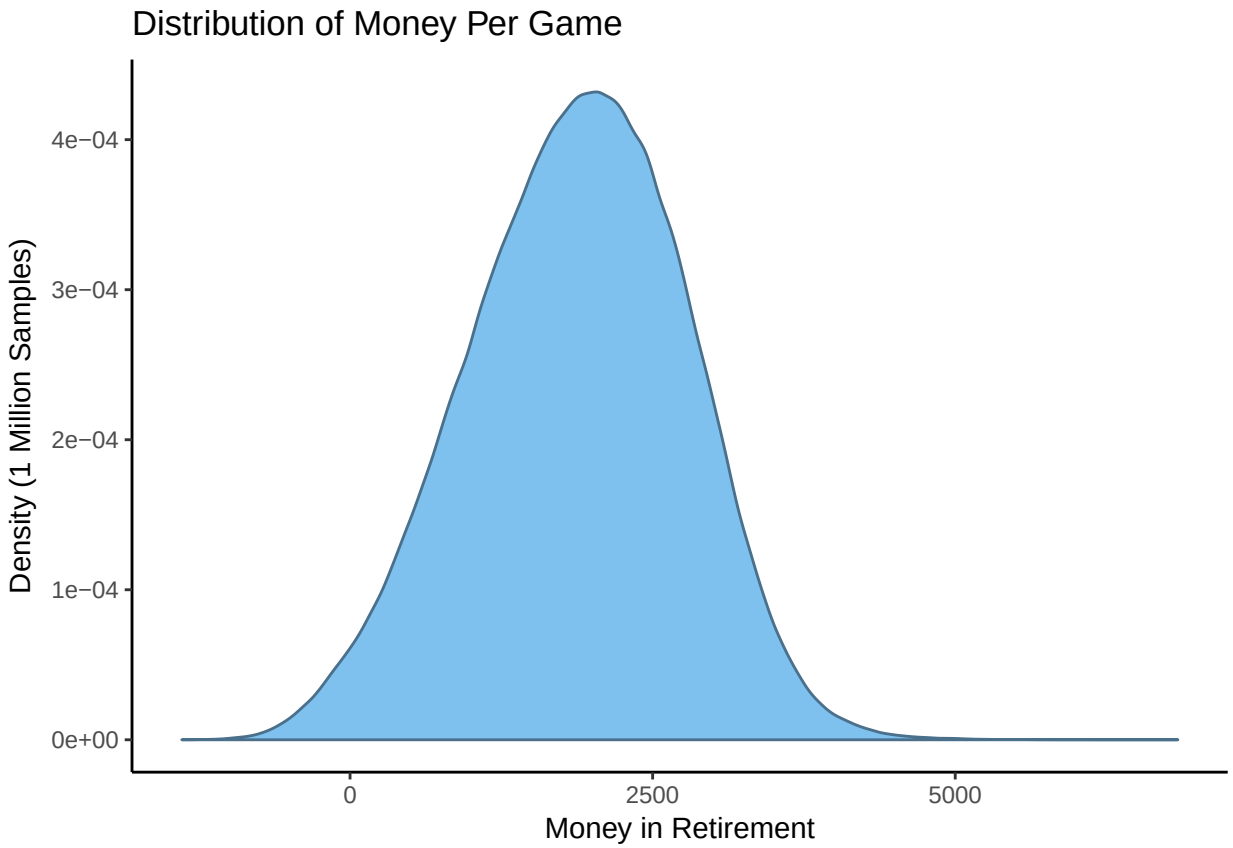
Cool stuff -> means and distributions of money, kids, houses, action cards, etc.

Means and Distributions of some summary statistics

```
#Average money in retirement
mean_money <- summarise(results2, avg = mean(money))
mean_money
```

```
##        avg
## 1 1868.773
```

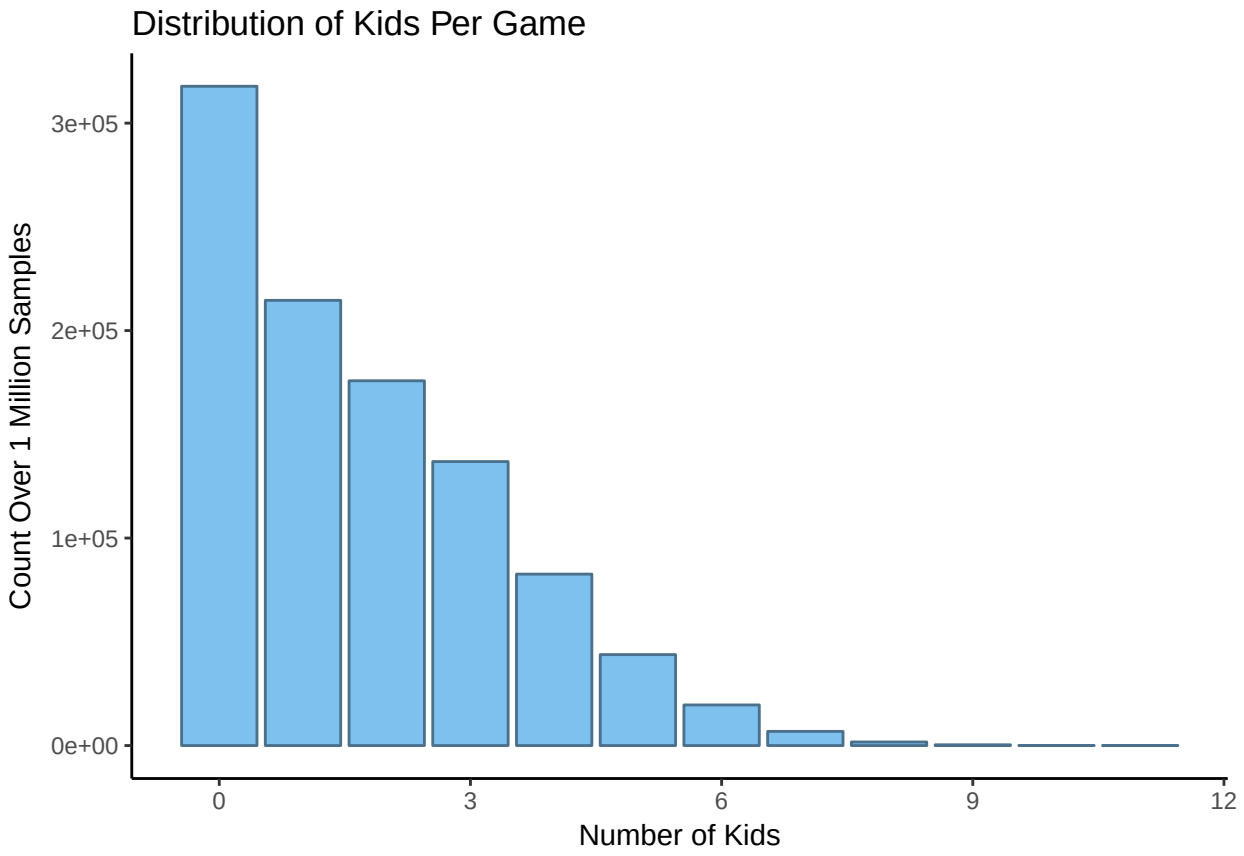
```
money_plot <- ggplot(results2, aes(money)) + geom_density(color = "skyblue4", fill = "skyblue2") + theme
money_plot
```




```
#Average number of kids
mean_kids <- summarise(results2, avg = mean(kid_count))
mean_kids
```

```
##        avg
## 1 1.709196
```

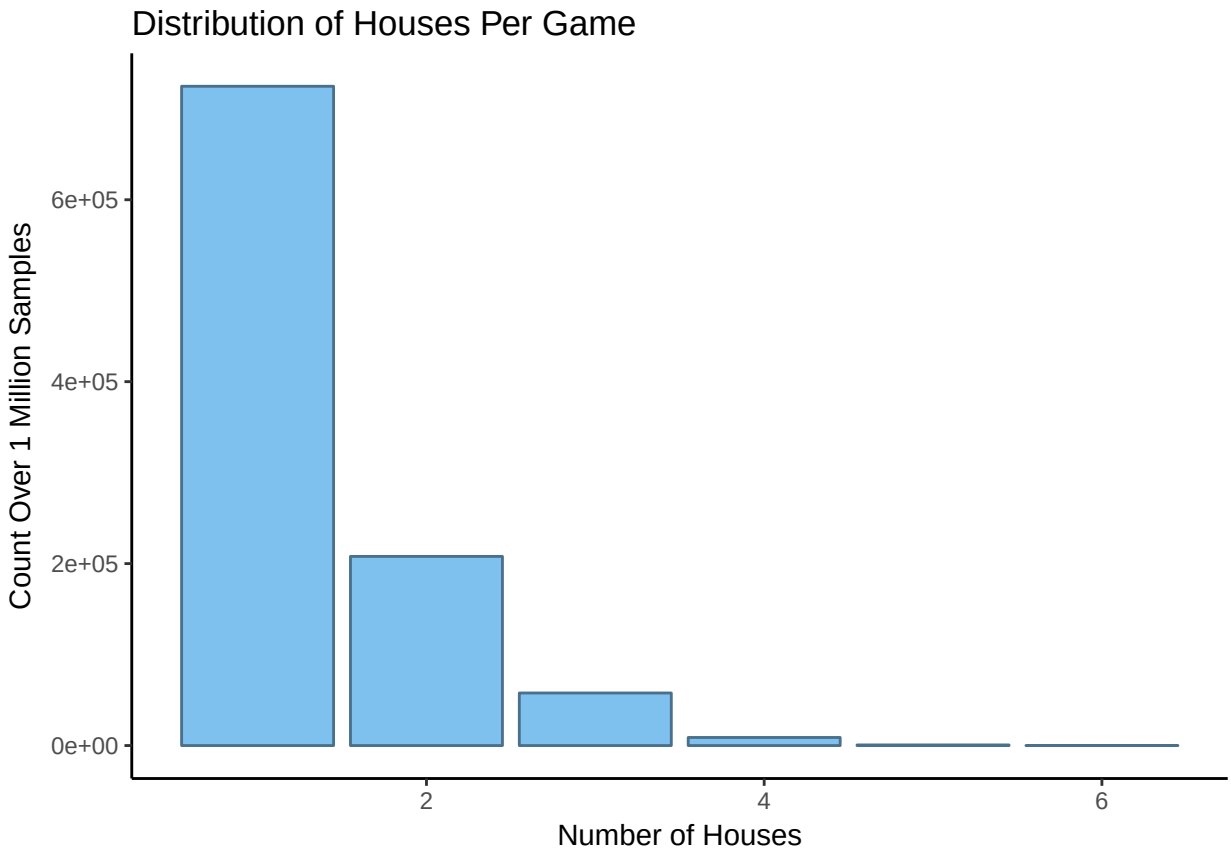
```
kids_plot <- ggplot(results2, aes(kid_count)) + geom_bar(color = "skyblue4", fill = "skyblue2") + theme_
kids_plot
```




```
#Average number of houses
mean_houses <- summarise(results2, avg = mean(number_of_houses))
mean_houses
```

```
##        avg
## 1 1.352885
```

```
houses_plot <- ggplot(results2, aes(number_of_houses)) + geom_bar(color = "skyblue4", fill = "skyblue2"
houses_plot
```



Some correlations

Q: Does a college job correlate with more money?

```
#Start with college
mean_college <- results2 %>% filter(college_path == 1) %>% summarise(avg = mean(money))
```

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

```
mean_college
```

```
##       avg
## 1 1981.13
```

```
#Never college (i.e.no college and no night school)
mean_life <- results2 %>% filter(college_path == 0 & night_school_path == 0) %>% summarise(avg = mean(m
mean_life
```

```
##        avg
## 1 1681.014
```

```
#No college then night school
mean_night_school <- results2 %>% filter(college_path == 0 & night_school_path == 1) %>% summarise(avg =
mean_night_school
```

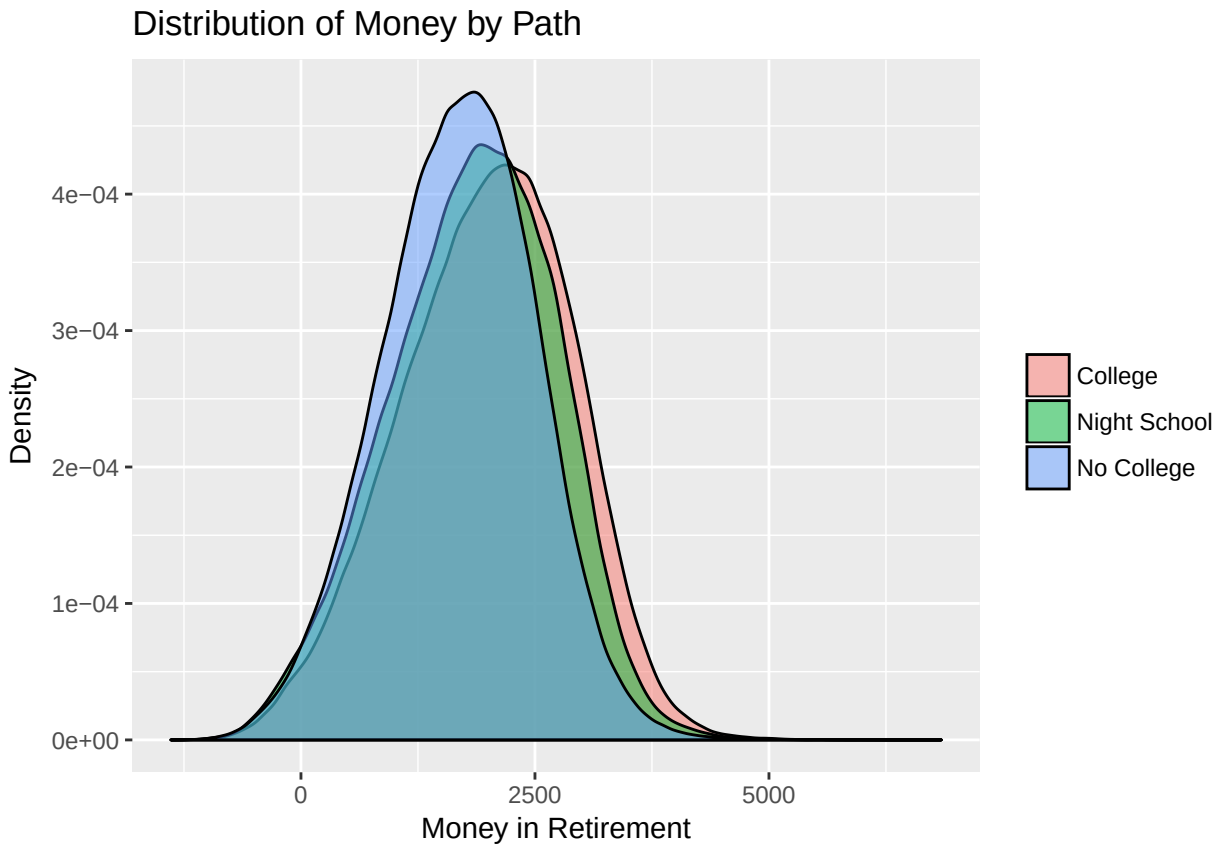
```
##        avg
## 1 1832.232
```

```r
#Subsetting data for graph
exploring_college <- results2 %>% mutate(college_results = ifelse(college_path == 1, 1, ifelse(college_

exploring_college <- exploring_college %>% mutate(college_results = ifelse(college_results == 1, "Colleg

college_plot <- ggplot(exploring_college, aes(x = money, fill = as.factor(college_results))) + geom_dens

college_plot
```



Swing by cpu lab and print Draw an outline for the final paper and get writing (shouldn't take long, now that you've written 1,000 lines of code lol)