STAT 345 Final Project - Real Estate

Ethan Manhart, Kayla Newman, Colin Fitzpatrick, Nick Kartschoke

## Introduction

## Code Breakdown

#In this section, we introduce a dictionary. This method of storage mirrors that of the dictionary in Python.

position <- dict()  
position[1] <- 1  
position[2] <- 1  
position[3] <- 1  
position[4] <- 1  
  
spaces <- dict()  
locations <- c(prop$Properties)  
i <- 0  
while(i < 41){  
 spaces[i] <- locations[i]  
 i <- i + 1  
}

## Various Roll Dice Functions:

#All functions mimic rolling dice in a game of monopoly. roll\_dice() is used for regular dice movement; force\_doubles() and differing\_dice() were used for testing.   
  
roll\_dice <- function()  
{  
 dice <- sample(1:6, 2, replace = TRUE)  
 return(dice)  
}  
  
force\_doubles <- function()  
{  
 dice <- sample(1:1, 2, replace = TRUE)  
 return(dice)  
}

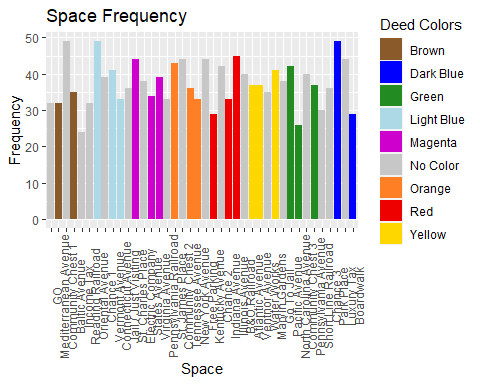
differing\_dice <- function()  
{  
 dice <- sample(1:6, 2, replace = TRUE)  
   
 while (dice[1] == dice[2])   
 {  
 dice <- sample(1:6, 2, replace = TRUE)  
 }  
   
 return(dice)  
}

## Basic Movement Function:

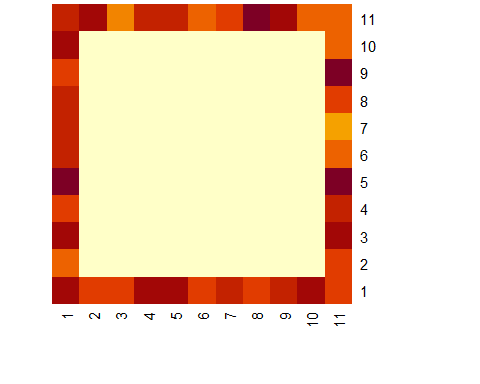
#The basic movement function takes an input of a player id and updates player position based on a roll to two six-sided dice. At this point in time, special movement, including rolling doubles, is not considered.  
  
#For clarity, the position of player i for i in {1, 2, 3, 4} is stored as position[i] <- return of basic\_movement(). (This will always be an integer between 1 and 12.)  
  
basic\_movement <- function(playerID)  
{  
 r <- sum(roll\_dice())  
 pl <- position[[playerID]] + r  
   
 if(pl >= 41)  
 {  
 pl <- pl - 40  
 }  
   
 print(paste0("Player", playerID, " rolls: ", r))  
 print(paste0("Player", playerID, " moves ", r, " spaces and lands on: ", spaces[[pl]]))  
 return(pl)  
}

## Heatmap with Simulation (Section 02):

deeds\_properties <- full\_join(frequency\_frame, deeds, by = "Properties")  
deeds\_properties$Color[is.na(deeds\_properties$Color)] <- "No Color"  
  
deeds\_properties$Properties <- factor(deeds\_properties$Properties, levels = deeds\_properties$Properties)  
  
ggplot(deeds\_properties, aes(Properties, value)) + geom\_col(aes(fill = Color)) + labs(title = "Space Frequency", x = "Space", y = "Frequency") + theme(axis.text.x = element\_text(angle = 90)) + scale\_fill\_manual(name = "Deed Colors", values = c("tan4", "blue", "forest green", "light blue", "magenta3", "grey78","chocolate1", "red2", "gold"))



#Visualization: HeatMap  
frequency\_subset <- frequency\_frame %>% select(value)  
frequency\_data <- unname(unlist(frequency\_subset))  
#frequency\_data  
  
monopoly\_board\_matrix <- matrix(data = c(frequency\_data[21], frequency\_data[22], frequency\_data[23], frequency\_data[24], frequency\_data[25], frequency\_data[26], frequency\_data[27], frequency\_data[28], frequency\_data[29], frequency\_data[30], frequency\_data[31], frequency\_data[20],0,0,0,0,0,0,0,0,0, frequency\_data[32],   
frequency\_data[19],0,0,0,0,0,0,0,0,0, frequency\_data[33],   
frequency\_data[18],0,0,0,0,0,0,0,0,0, frequency\_data[34],  
frequency\_data[17],0,0,0,0,0,0,0,0,0, frequency\_data[35],   
frequency\_data[16],0,0,0,0,0,0,0,0,0, frequency\_data[36],   
frequency\_data[15],0,0,0,0,0,0,0,0,0, frequency\_data[37],  
frequency\_data[14],0,0,0,0,0,0,0,0,0, frequency\_data[38],   
frequency\_data[13],0,0,0,0,0,0,0,0,0, frequency\_data[39],   
frequency\_data[12],0,0,0,0,0,0,0,0,0, frequency\_data[40],  
frequency\_data[11], frequency\_data[10], frequency\_data[9], frequency\_data[8], frequency\_data[7], frequency\_data[6], frequency\_data[5], frequency\_data[4], frequency\_data[3], frequency\_data[2], frequency\_data[1]), nrow = 11, ncol = 11)  
heatmap <- heatmap(x=monopoly\_board\_matrix, Colv = NA, Rowv = NA, scale = "none")



1. *Double Down* Improve your system to incorporate going to jail, including the doubles-rolling condition. Which properties are landed on most frequently now? Has this changed dramatically from the previous system (that didn’t account for jail)? Again, visualize this!

## In Jail/Exiting Jail:

#The In\_Out\_Jail() takes a player and, utilizing game rules and logical if/else statements, simulates a player entering and exiting jail. The statements are rather self-explanatory, but we would like to highlight the use of section3\_jail\_roll. This function allows us to move a player with starting location equal to Jail / Just Visiting = 11. (Implements the section3\_moving and section3\_moving\_doubles functions explained later.)  
  
section3\_In\_Out\_Jail <- function(player\_identifer, counter = 0)  
{  
 position[player\_identifer] <- 11  
   
 if(counter == 0)  
 {  
 print(paste0("Player", player\_identifer, " is now in jail."))  
 section3\_In\_Out\_Jail(player\_identifer, counter + 1)  
 }  
   
 else  
 {  
 hand <- Check\_Hand(player\_identifer)  
 card <- hand %>% select(Get.Out.of.Jail.Free)   
   
 if(is.na(card) == FALSE)  
 {  
 players[player\_identifer, ]$Get.Out.of.Jail.Free <- NA  
 print(paste0("Player", player\_identifer, " uses a 'Get Out of Jail Free Card.'"))  
 section3\_jail\_roll(player\_identifer)  
 }  
   
 else  
 {  
 funds <- hand %>% select(Bank)  
   
 if(funds >= 500)  
 {  
 players[player\_identifer, ]$Bank <- players[player\_identifer, ]$Bank - 50  
 print(paste0("Player", player\_identifer, " pays $50 to exit jail."))  
 section3\_jail\_roll(player\_identifer)  
 }  
   
 else  
 {  
 roll <- roll\_dice()  
   
 if(roll[1] == roll[2])  
 {   
 print(paste0("Player", player\_identifer, " rolls doubles to exit jail."))  
 section3\_jail\_roll(player\_identifer)  
 }  
   
 else  
 {  
 if(counter < 2)  
 {  
 print(paste0("Player", player\_identifer, " remains in jail."))  
 section3\_In\_Out\_Jail(player\_identifer, counter + 1)  
 }  
   
 else  
 {  
 funds <- hand %>% select(Bank)  
   
 if(funds >= 50)  
 {  
 funds[[1]] <- funds[[1]] - 50  
   
 print(paste0("Player", player\_identifer, " pays $50 to exit jail."))  
 section3\_jail\_roll(player\_identifer)  
 }  
   
 else  
 {  
 print(paste0("Player", player\_identifer, " is bankrupted."))  
 #remove player from player dataset?  
 }  
 }  
 }  
 }  
 }  
 }  
   
 #return(exit\_roll)  
 #invisible(players)  
}

## General Movement Functions:

#1) Moving (3) allows us to move a player from a starting position. We found that using this function alone creates an issue for movement, as the function does not “remember” the previous roll when moving twice consecutively as is the case when rolling doubles.

#2) Moving Doubles (3) is used when multiple dice rolls need to be considered. This stores player's last location and moves the player from their last known location.  
  
section3\_moving\_doubles <- function(playerID, spaces\_to\_move, player\_location)  
{  
 position[[playerID]] <- player\_location  
 position[[playerID]] <- position[[playerID]] + spaces\_to\_move  
 user\_place <- position[[playerID]]  
   
 #Restarts Count after Passing GO:  
 if(user\_place >= 41)  
 {  
 user\_place <- user\_place - 40  
 }  
   
 print(paste0("Player", playerID, " moves ", spaces\_to\_move, " spaces and lands on: ", spaces[[user\_place]]))  
 #position[playerID] <- user\_place  
 return(user\_place)  
}

## Movement from Jail:

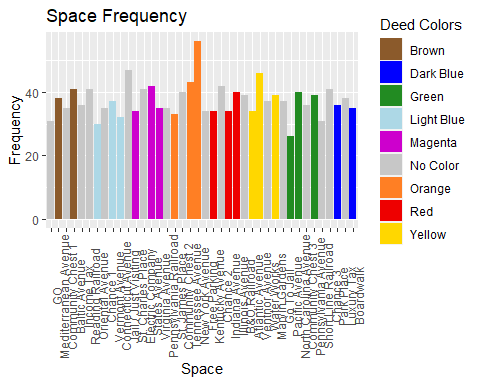
#Used for exiting jail; see section3\_roll\_to\_move player for understanding of process.  
  
section3\_jail\_roll <- function(playerID, counter = 1, player\_position = 11, type\_of\_roll = roll\_dice())  
{  
 #position[[playerID]] <- player\_position  
 p <- player\_position  
   
 roll <- type\_of\_roll  
   
 if(roll[1] == roll[2])  
 {  
 if(counter == 3)  
 {  
 totalDice <- sum(roll)  
   
 print(paste0("Player", playerID, " rolls doubles for a third time!", " The sum of their roll is: ", totalDice))  
 print(paste0("Player", playerID, " returns to Jail for rolling too many doubles."))  
 section3\_In\_Out\_Jail(playerID) #(Go\_to\_Jail implemented in following section)  
 }  
   
 else  
 {  
 totalDice <- sum(roll)  
   
 if(counter == 1)  
 {  
 print(paste0("Player", playerID, " rolls doubles!", " The sum of their roll is: ", totalDice))  
 new\_position <- section3\_moving\_doubles(playerID, totalDice, 11)  
 player\_position <- new\_position  
 }  
   
 else  
 {  
 #print(roll)  
 print(paste0("Player", playerID, " rolls doubles again!", " The sum of their roll is: ", totalDice))  
 new\_position <- section3\_moving\_doubles(playerID, totalDice, p)  
 player\_position <- new\_position  
 }  
   
 section3\_jail\_roll(playerID, counter + 1, new\_position)  
 }  
 }  
   
 else  
 {  
 totalDice <- sum(roll)  
   
 if(counter == 1)  
 {  
 print(paste0("Player", playerID, " rolls: ", totalDice))  
 }  
   
 else  
 {  
 print(paste0("Player", playerID, " rolls again and rolls: ", totalDice))  
 }  
   
 section3\_moving\_doubles(playerID, totalDice, p)  
 }  
}

## Movement Around Gameboard:

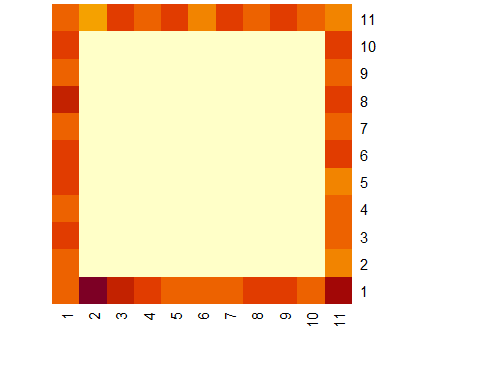
#The following function rolls the dice for movement purposes; there are a couple parts worth noting:  
 #The roll\_to\_move\_player function uses four functions: section3\_roll\_dice, section3\_moving, section3\_moving\_doubles, and section3\_In\_Out\_Jail.  
 #There are several cases to consider, but they are all designed to account for the count of the doubles and player location after each movement.  
  
section3\_roll\_to\_move\_player <- function(playerID, counter = 0, player\_position = 1, type\_of\_roll = roll\_dice())  
{  
 p <- player\_position  
   
 roll <- type\_of\_roll  
   
 if(roll[1] == roll[2])  
 {  
   
 if(counter == 2)  
 {  
 totalDice <- sum(roll)  
   
 print(paste0("Player", playerID, " rolls doubles for a third time!", " The sum of their roll is: ", totalDice))  
 print(paste0("Player", playerID, " is sent to Jail for rolling too many doubles."))  
 section3\_In\_Out\_Jail(playerID)  
 }  
   
 else  
 {  
 totalDice <- sum(roll)  
   
 if(counter == 0)  
 {  
 print(paste0("Player", playerID, " rolls doubles!", " The sum of their roll is: ", totalDice))  
 new\_position <- section3\_moving(playerID, totalDice)  
 player\_position <- new\_position  
 }  
   
 else  
 {  
   
 print(paste0("Player", playerID, " rolls doubles again!", " The sum of their roll is: ", totalDice))  
 new\_position <- section3\_moving\_doubles(playerID, totalDice, p)  
 player\_position <- new\_position  
 }  
   
 section3\_roll\_to\_move\_player(playerID, counter + 1, new\_position)  
 }  
 }  
   
 else  
 {  
 totalDice <- sum(roll)  
   
 if(counter == 0)  
 {  
 print(paste0("Player", playerID, " rolls: ", totalDice))  
 section3\_moving(playerID, totalDice)  
 }  
   
 else  
 {  
 print(paste0("Player", playerID, " rolls again and rolls: ", totalDice))  
 section3\_moving\_doubles(playerID, totalDice, p)  
 }  
 }  
}

## Heatmap with Simulation (Section 03):

deeds\_properties <- full\_join(frequency\_frame, deeds, by = "Properties")  
deeds\_properties$Color[is.na(deeds\_properties$Color)] <- "No Color"  
  
deeds\_properties$Properties <- factor(deeds\_properties$Properties, levels = deeds\_properties$Properties)  
  
ggplot(deeds\_properties, aes(Properties, value)) + geom\_col(aes(fill = Color)) + labs(title = "Space Frequency", x = "Space", y = "Frequency") + theme(axis.text.x = element\_text(angle = 90)) + scale\_fill\_manual(name = "Deed Colors", values = c("tan4", "blue", "forest green", "light blue", "magenta3", "grey78","chocolate1", "red2", "gold"))



#Visualization: HeatMap  
frequency\_subset <- frequency\_frame %>% select(value)  
frequency\_data <- unname(unlist(frequency\_subset))  
#frequency\_data  
  
monopoly\_board\_matrix <- matrix(data = c(frequency\_data[21], frequency\_data[22], frequency\_data[23], frequency\_data[24], frequency\_data[25], frequency\_data[26], frequency\_data[27], frequency\_data[28], frequency\_data[29], frequency\_data[30], frequency\_data[31], frequency\_data[20],0,0,0,0,0,0,0,0,0, frequency\_data[32],   
frequency\_data[19],0,0,0,0,0,0,0,0,0, frequency\_data[33],   
frequency\_data[18],0,0,0,0,0,0,0,0,0, frequency\_data[34],  
frequency\_data[17],0,0,0,0,0,0,0,0,0, frequency\_data[35],   
frequency\_data[16],0,0,0,0,0,0,0,0,0, frequency\_data[36],   
frequency\_data[15],0,0,0,0,0,0,0,0,0, frequency\_data[37],  
frequency\_data[14],0,0,0,0,0,0,0,0,0, frequency\_data[38],   
frequency\_data[13],0,0,0,0,0,0,0,0,0, frequency\_data[39],   
frequency\_data[12],0,0,0,0,0,0,0,0,0, frequency\_data[40],  
frequency\_data[11], frequency\_data[10], frequency\_data[9], frequency\_data[8], frequency\_data[7], frequency\_data[6], frequency\_data[5], frequency\_data[4], frequency\_data[3], frequency\_data[2], frequency\_data[1]), nrow = 11, ncol = 11)  
heatmap <- heatmap(x=monopoly\_board\_matrix, Colv = NA, Rowv = NA, scale = "none")



1. *Take a Chance* Incorporate the movement cards into your system. You can ignore the effects of non-movement cards, but not their existence! Which properties are landed on most frequently now? Has this changed dramatically from the previous system (that didn’t account for Chance)? Again, visualize this!

## Action Card Movement:

#Helper Functions for chance\_or\_community\_chest() that performs the more challenging movement actions that appear on chance and community chest cards.  
  
#a) Distance Functions:  
  
maxnegative <- function(x)  
{  
 min(x)  
}  
  
minpositive <- function(x)  
{  
 min(x[x > 0])  
}  
  
#b) Movement Based on Action Card Description:   
  
action\_card\_movement <- function(playerID, player\_location, card\_text)  
{  
 rail\_or\_util\_spaces <- c()  
   
 if(card\_text == "Nearest Railroad")  
 {  
 rail\_or\_util\_spaces <- c(6, 16, 26, 36)  
 }  
   
 if(card\_text == "Nearest Utility")  
 {  
 rail\_or\_util\_spaces <- c(13, 29)  
 }  
   
 if(card\_text == "-3")  
 {  
 minus\_3 <- player\_location - 3  
   
 if(minus\_3 <= 0)  
 {  
 minus\_3 <- minus\_3 + 40  
 }  
   
 print(paste0("Player", playerID, " moves back 3 spaces and lands on ", spaces[[minus\_3]]))  
   
 return(minus\_3)  
 }  
   
 if(card\_text == "Jail / Just Visiting")  
 {  
 location\_after\_exiting\_jail <- In\_Out\_Jail(playerID)  
 return(location\_after\_exiting\_jail)  
 }  
   
 if((card\_text != "Nearest Utility") && (card\_text != "Nearest Railroad") && (card\_text != "-3") && (card\_text != "Jail / Just Visiting"))  
 {  
 print("You shouldn't be in here!")  
 return(player\_location)  
 }  
   
 #player\_position <- position[[playerID]]  
   
 difference <- rail\_or\_util\_spaces - player\_location  
   
 if (all(difference < 0))  
 {  
 #print("All values are negatives!")  
 distance <- maxnegative(difference)  
 }  
   
 else  
 {  
 distance <- minpositive(difference)  
 }  
   
 #print(distance)  
   
 nearest <- distance + player\_location  
   
 print(paste0("Player", playerID, " moves to ", spaces[[nearest]]))  
  
 return(nearest)  
}

## Landing on Chance:

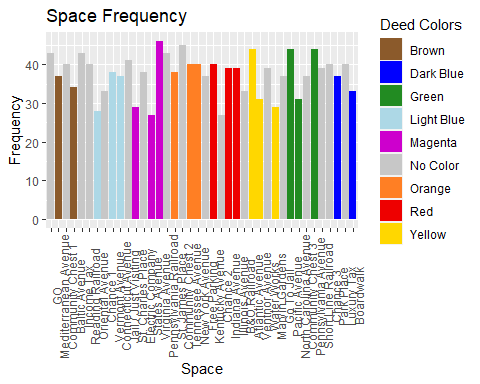
#chance\_or\_community\_chest() prints the text of a chance and community chest card and performs the action when it is movement related.  
  
chance\_or\_community\_chest <- function(playerID, player\_position)  
{  
 chance\_locations <- which(spaces %in% "Chance")  
 community\_chest\_locations <- which(spaces %in% "Community Chest")  
   
 problem\_cards <- c("Nearest Utility", "Nearest Railroad", "-3", "Jail / Just Visiting")  
   
 if(player\_position %in% chance\_locations)  
 {  
 chance\_cards <- action\_cards %>% filter(Card.Type %in% "Chance")  
 rand\_num <- sample(1:15, 1, replace = TRUE)  
 specific\_action <- chance\_cards[rand\_num,]  
 print(paste0("Player", playerID, " lands on Chance and draws a Chance card."))  
 print(paste0("Their card reads: ", specific\_action$Objective))  
 location <- player\_position  
   
 if(specific\_action$Places != "")  
 {  
 if(specific\_action$Places %in% problem\_cards)  
 {  
 #print(specific\_action$Places)  
 location <- action\_card\_movement(1, player\_position, specific\_action$Places)  
 }  
   
 else  
 {  
 #print(specific\_action$Places)  
 location <- which(spaces %in% specific\_action$Places)  
 }  
 }  
 }  
   
 if(player\_position %in% community\_chest\_locations)  
 {  
 community\_cards <- action\_cards %>% filter(Card.Type %in% "Community")  
 rand\_num <- sample(1:16, 1, replace = TRUE)  
 specific\_action <- community\_cards[rand\_num,]  
 print(paste0("Player", playerID, " lands on Community Chest and draws a Community Chest card."))  
 print(paste0("Their card reads: ", specific\_action$Objective))  
 location <- player\_position  
   
 if(specific\_action$Places != "")  
 {  
 if(specific\_action$Places %in% problem\_cards)  
 {  
 #print(specific\_action$Places)  
 location <- action\_card\_movement(1, player\_position, specific\_action$Places)  
 }  
   
 else  
 {  
 #print(specific\_action$Places)  
 location <- which(spaces %in% specific\_action$Places)  
 }  
 }  
 }  
   
 if((player\_position %in% chance\_locations) == FALSE && (player\_position %in% community\_chest\_locations) == FALSE)  
 {  
 location <- player\_position  
 }  
   
 return(location)  
}

## Functions Implemented Behind the Scenes (Updated to Include Action Cards):

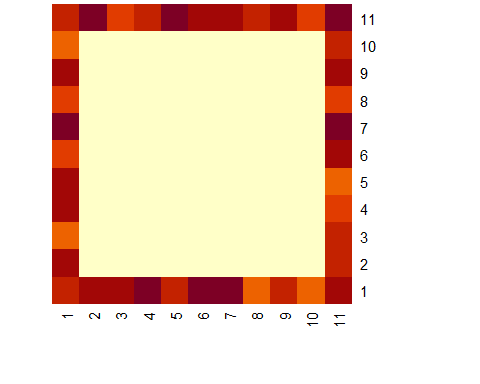
1. **In Jail/Exiting Jail:**
2. **Movement From Jail:**
3. **General Movement:**
4. **Movement Around Gameboard:**

## Heatmap with Simulation 4.1:

deeds\_properties <- full\_join(frequency\_frame, deeds, by = "Properties")  
deeds\_properties$Color[is.na(deeds\_properties$Color)] <- "No Color"  
  
deeds\_properties$Properties <- factor(deeds\_properties$Properties, levels = deeds\_properties$Properties)  
  
ggplot(deeds\_properties, aes(Properties, value)) + geom\_col(aes(fill = Color)) + labs(title = "Space Frequency", x = "Space", y = "Frequency") + theme(axis.text.x = element\_text(angle = 90)) + scale\_fill\_manual(name = "Deed Colors", values = c("tan4", "blue", "forest green", "light blue", "magenta3", "grey78","chocolate1", "red2", "gold"))



#Visualization: HeatMap  
frequency\_subset <- frequency\_frame %>% select(value)  
frequency\_data <- unname(unlist(frequency\_subset))  
#frequency\_data  
  
monopoly\_board\_matrix <- matrix(data = c(frequency\_data[21], frequency\_data[22], frequency\_data[23], frequency\_data[24], frequency\_data[25], frequency\_data[26], frequency\_data[27], frequency\_data[28], frequency\_data[29], frequency\_data[30], frequency\_data[31], frequency\_data[20],0,0,0,0,0,0,0,0,0, frequency\_data[32],   
frequency\_data[19],0,0,0,0,0,0,0,0,0, frequency\_data[33],   
frequency\_data[18],0,0,0,0,0,0,0,0,0, frequency\_data[34],  
frequency\_data[17],0,0,0,0,0,0,0,0,0, frequency\_data[35],   
frequency\_data[16],0,0,0,0,0,0,0,0,0, frequency\_data[36],   
frequency\_data[15],0,0,0,0,0,0,0,0,0, frequency\_data[37],  
frequency\_data[14],0,0,0,0,0,0,0,0,0, frequency\_data[38],   
frequency\_data[13],0,0,0,0,0,0,0,0,0, frequency\_data[39],   
frequency\_data[12],0,0,0,0,0,0,0,0,0, frequency\_data[40],  
frequency\_data[11], frequency\_data[10], frequency\_data[9], frequency\_data[8], frequency\_data[7], frequency\_data[6], frequency\_data[5], frequency\_data[4], frequency\_data[3], frequency\_data[2], frequency\_data[1]), nrow = 11, ncol = 11)  
heatmap <- heatmap(x=monopoly\_board\_matrix, Colv = NA, Rowv = NA, scale = "none")



## Cost Benefit Analysis:

#The following code takes the frequency\_frames created in the Heatmap with Simulations and creates a third column of the data that gives the percentage of the time a space is hit. We then remove all non-property spaces and evaluate the following code: (Profitability.Index = (Avg.Percent \* (Hotel - 5\*Cost.of.Houses.Hotels - List.Price))) to determine the most profitable property group.  
  
properties\_we\_want <- deeds %>% select(Properties)  
properties\_we\_want <- unname(unlist(properties\_we\_want))  
  
frequency\_frame1 <- frequency\_frame1 %>% mutate(percent.hit = value / times)  
frequency\_frame2 <- frequency\_frame2 %>% mutate(percent.hit = value / times)  
  
average\_frequency <- full\_join(frequency\_frame1, frequency\_frame2, by = "name") %>% mutate(Avg.Percent = ((percent.hit.x + percent.hit.y) / 2)) %>% select(name, Avg.Percent)  
average\_frequency\_properties <- average\_frequency %>% filter(name %in% properties\_we\_want)  
  
  
  
cost\_benefit\_df <- deeds %>% select(Color, List.Price, Hotel, Cost.of.Houses.Hotels)  
  
combined\_cost\_benefit\_df <- cbind.data.frame(average\_frequency\_properties, cost\_benefit\_df)  
combined\_cost\_benefit\_df <- combined\_cost\_benefit\_df %>% mutate(Profitability.Index = (Avg.Percent \* (Hotel - 5\*Cost.of.Houses.Hotels - List.Price))) %>% select(-c(Cost.of.Houses.Hotels, List.Price))  
  
  
most\_profitable\_property\_groups <- combined\_cost\_benefit\_df %>% select(Color, Profitability.Index)  
most\_profitable\_property\_groups <- aggregate(most\_profitable\_property\_groups$Profitability.Index, list(most\_profitable\_property\_groups$Color), mean) %>% rename(Color = Group.1, Profitability.Index = x) %>% arrange(by = desc(Profitability.Index))  
  
most\_profitable\_property\_groups

## Color Profitability.Index  
## 1 Dark Blue 8.3000000  
## 2 Orange 7.3133333  
## 3 Light Blue 5.0366667  
## 4 Magenta 3.8022222  
## 5 Yellow 3.7911111  
## 6 Red 2.2177778  
## 7 Brown 0.8533333  
## 8 Green 0.3827778

## Analysis

## Conclusion