Casey Freimund
Felipe Pincheira-Berthelon
Eagan Norman
5/13/19

Where's the Money in Monopoly?

Monopoly was first introduced to the world in 1953. Player's roll two die and move their respective token around a square board, purchasing properties in hopes that they build a monopoly, and their opponents' bank accounts hit 0. Since then, there have been 1,144 versions of the game released. The widespread problem players face with monopoly is trying to determine which properties to capture and where their opponents land on most frequently affects that decision. By doing so, they can reduce their opponents cash balance to 0 and come out victorious. One thing that affects this outcome is the variety of game rules that determine how the game is played, and how often people play with these different rules. Our project below shows how the game of Monopoly was simulated, with our simulation specific rules, and how we kept track of our information. But, before we begin talking about how the game was simulated, we will describe the rules we will be basing our simulations off.

The rules we will be discussing are the one's described on the Monopoly Wiki, (https://monopoly.fandom.com/wiki/Main_Page) and are as follows:

Movement:

- Two, 6-sided dice are rolled. The amount shown on the face of the dice is how far a player moves.
- If the player rolls two of the same number on each die ("doubles"), then the player can roll again.
- If the player rolls "doubles" a third time, then they go to jail.

Jail:

- While in jail, a player can get out by rolling "doubles", using the "Get Out of Jail Free" card, or by paying a $50 fee, either to the bank, or "Free Parking" space.
- If a player does not roll "doubles" by their third turn in jail, they must pay the $50 fee.

Chance and Community Chest:

- If a player lands on a "Chance" or "Community Chest" space, they draw a card from the corresponding deck and follow the instructions on the card.
- If a player draws the "Get Out of Jail Free" card from either deck, the card is held by the player until used. When the card is used, it is returned to the bottom of the deck.

Free Parking:

- The rules involving "Free Parking" vary with the players. Some people choose to play a version where all fees and taxes are to be placed in "Free Parking." If a player lands on "Free Parking," they gain all the money from that space. Another version of this game treats "Free Parking" as an empty space, where no action is taken place.

Income and Luxury Tax:

Casey Freimund
Felipe Pincheira-Berthelon
Eagan Norman
5/13/19

- If a player lands on “Income Tax” or “Luxury Tax” the player pays the bank, or the “Free Parking” space.

Go:

- Unless specified otherwise by a card, every time a player passes go, they collect $200.

With these rules in mind, we can begin to look at how to simulate the game using R.

The first step in simulating Monopoly was constructing an easily navigable data frame for the game board to ensure our code would be efficient. This first meant figuring out how rows and columns could be decided. We tried doing it with each column corresponding to a certain space, and each row. After trying to program both ways, we concluded that the row method would be easier, due to faster indexing when it came to movement.

The next step was deciding how to make the most general data frame for the board, while including all the information needed for gameplay functions. By looking at our game from the simplest perspective, which dealt primarily with movement and tracking which spaces were landed on, we only had to construct columns for the names of the spaces, the categories for the spaces (which we referenced as the estates), and the number of lands each space had. For gameplay incorporating property purchasing, this would also include columns for Booleans on whether the property was purchasable and purchased, columns for how many houses and hotels, and a column for the price and the rent for certain spaces. These weren’t incorporated in our simulation for reasons that will be addressed later.

When initializing the board, we created a vector with the specific names and estates, and an initial lands value of zero, which was repeated 40 times when the squares data frame was initialized. This can all be seen in the code below.

```
{r}
spaces=c("Go","Mediterranean","Community Chest","Baltic","Income Tax","Reading
RR","Oriental","Chance","Vermont","Connecticut","Jail","St. Charles","Electric Company","States","Virginia","Pennsylvania
RR","St. James","Community Chest","Tennessee","New York","Free Parking","Kentucky","Chance","Indiana","Illinois","B&O
RR","Atlantic","Ventnor","Water Works","Marvin Gardens","Go To Jail","Pacific","North Carolina","Community
Chest","Pennsylvania","Short Line","Chance","Park","Luxury Tax","Boardwalk")

estates=c("Go","Brown","Community Chest","Brown","Income Tax","RR","Light Blue","Chance","Light Blue","Light
Blue","Jail","Pink","Utility","Pink","Pink","RR","Orange","Community Chest","Orange","Orange","Free
Parking","Red","Chance","Red","Red","RR","Yellow","Yellow","Utility","Yellow","Go To Jail","Green","Green","Community
Chest","Green","RR","Chance","Blue","Luxury Tax","Blue")

lands=0

#Creates a data frame of the spaces/estates and lands to keep track of how many times a space gets landed on
squares=data.frame(spaces,estates,lands)
```

The next step was to create a data frame for each player. When initially starting this project, these data frames were simple, but as we continued programming, we kept having to come back and add additional categories. Initially, just the names and current spots were tracked, but throughout the project, we added columns to keep track of the number of rolls in a turn, whether a player was in jail and for how long, whether they had a get out of jail free card, and money, for each player. When accounting for these complexities, we also created Booleans to modify how the game was played. These were made so we could have one general gameplay function, rather than copying and pasting it for each

added part of the game. Additionally, in the first chunk of code, we included a couple of vectors to track which rolls were most common and which jailing methods were most common. These can all be seen in the code below.

```
currentSpot=1
money=1500
inJail=0
jailRounds=0
com.jailFree = FALSE
chance.jailFree=FALSE
rolls=0
freeParking=0

#Creates a copy of the squares data frame that will keep a running total of how many times a space gets landed on through
'x' amount of games played.
spaceMaster=squares

shoe=data.frame(currentSpot,inJail,jailRounds, com.jailFree, chance.jailFree, money,rolls)
boat=shoe
car=shoe
dog=shoe
shoe$name <- "shoe"
boat$name <- "boat"
car$name <- "car"
dog$name <- "dog"

incorporateJail <- FALSE
incorporateCards <- FALSE
incorporateMoney <- FALSE
incorporateFreeParking <- FALSE
playToWin=FALSE
rollCount=rep(0,12)
jailMethod=rep(0,3)
```

After we created our player and space counter data frames, we had to create a system that allowed all 4 players to take turns “moving” around our game board. Due to our version of Monopoly having two die, we created a simple function called roll that was used to simulate rolling two die. When roll was called, it randomly generated two numbers from 1 to 6, and returned those two values in the form of a list. Next, we created our movement function. Our movement function takes in a player profile and a value called dice, which we set equal to our roll function. After the player rolls the dice, the move function changes their current spot, depending on what they rolled. If a player rolls and their new position is greater than 40, the number of spaces on a Monopoly board, then our function sets their current spot back to 1, the “Go Space,” and moves the player forward based on how many spaces they have left to move, from their roll. In between each player’s turn, we made a call to a space counter function. This function takes in the most recent player profile, after they have moved, and our space counter data frame. Next, it takes the player’s new location and finds the equivalent spot in our space counter data frame. It finally adds a plus one to our “lands” category, for the given spaces, and keeps a running tally of how many times a space was landed on, by returning our space counter data frame.

At this point in our game, player movement isn’t being affected by cards or space actions, such as “Go to Jail,” and our players are playing without money. If we were to simulate a game in our current state, our players would be taking turns moving around the board, with no end game in sight. To solve this issue for now, we put our roll function, followed by a player profile and our space counter function, for each player, inside a repeat function where the repeat would break once each player has played 10,000 turns. After simulating a few games and looking at how many times a space was landed on, in descending order, we noticed something odd. For example, in one simulation, Ventnor was the fifth most landed on spot. In a second simulation, Ventnor was now the thirteenth most landed on spot. The

drop in rank from fifth to thirteenth, in amount of times landed on, doesn't seem too significant when we're talking about 1 out of 40 spaces, but it does become a problem when the majority of our 40 spaces are varying in rank by three to twenty-six positions. To reduce this error and produce more consistent results, we first set a seed to produce the same results every time. Then, we put our current repeat function inside another repeat function. What happens now is our players play 10,000 rounds, like before, and once the game is over, we save the space totals in a master space counting data frame. Our game board and player profiles then get reset and the process starts over. Now, we can simulate 10 games of 10,000 rounds and keep a running total for the space counts in our master space counting data frame.

Due to the players simply moving around the board, with nothing but their roll affecting their position, our variation in most frequently encountered properties, is very small. Our two most landed on spaces were Chance and Community Chest, which were both landed on 7.5% of the time, because there were 3 of each space on the board. Meanwhile, every other property on the board was landed on about 2.5% of the time. If we were to divide each proportion for amount of times community chest and chance were landed by 3, we would get about 2.5%. Therefore, each space had a 1/40 chance, 2.5%, to be landed on, throughout the 10 games. This proportion was expected, due to nothing affecting player movement. As for the colors, the railroads were landed on about 10% of the time, due to that colored space having 4 locations, all of which are spread equally throughout the board. All colors with 3 spaces on the board were landed on about 7.5% of the time. Finally, we have 6 unique spaces that only appear on the board once. These spaces include, "Go To Jail," "Income Tax," "Luxury Tax," "Free Parking," "Go," and "Jail," all of which were landed on 2.5% of the time.

Our next step was to simulate how the distribution in lands per space would change with the rules jail and rolling doubles incorporated. In Monopoly, players can get sent to jail in one of four ways. Players can either: land on the "Go To Jail" space, roll three consecutive doubles in the same turn, or draw the "Go To Jail" card from the community chest or chance decks. Once a player is in jail, they have two ways to get out of jail before facing a penalty. Their first option is to roll a double on their first, second, or third round in jail. Their second option is to use one of the two "Get Out of Jail Free" cards, each found in both card decks. If a player is unable to get out of jail in those two ways, they can only get out by paying a $50 fine on their last round (third round) in jail. Since we aren't incorporating cards and money into the game yet, players can only get out of jail by rolling a double before their third consecutive turn in jail.

At this point, our players still don't have an end goal, therefore we again simulated 10 games where each player gets 10,000 turns. Due to the players movement around the board being affected by things other than a single roll of 2 die, we were expecting our results to have more variation than our last simulation. Our results showed that in the property set of spaces, Jail was the most landed on space, with it being landed on about 10.97% of the time. One thing to consider is this proportion doesn't directly correlate to players only spending time in jail. If a player lands on the jail space, a count is recorded in our space counting data frame, but they are technically just visiting and are not subject to the rules of getting out of jail. Meanwhile, if a player lands on "Go To Jail" or rolls 3 consecutive doubles, they get sent to jail, which also adds a count to that space. If a player fails to get out of jail before their

third turn, each consecutive turn also adds a count to the jail space. All these factors considered, it seems reasonable that the Jail space is the most landed on space, given our current set of rules. Another feature worth noting in this simulation is the spaces between Jail and the closest Chance space (12 spaces forward, i.e. the most amount of spaces a player can move off a roll) have a higher percent chance to be landed on than other spots on the board. The percent chance a player lands on these spaces ranges from 2.4%-2.6% per space, which is about 0.4% higher than most spaces not in the Jail to Chance range. This can also be shown in the space colors, where 3 of our top 5 most landed on colors are orange (7.5%), yellow (7.4%), and red (7.2%), all of which are spaces a player runs into immediately, or shortly after coming out of jail. The rest of our colors are landed on less than 6.8% of the time with most colors only being landed on 2%-4% of the time. At our current state, it seems as if the orange, red, and yellow properties are the most frequent properties landed on.

The next addition to our simulation is adding the effects of the Chance and Community Chest cards. To do this, we first had to create the Chance and Community Chest Cards as well as a way to simulate drawing a card. First, we created a dataframe representative of both decks with one column containing the name of the card and the other the position of the card in the deck (1 being the card on the top). In order to simulate shuffling the cards, we utilized the "sample" function when assigning the position of the card when initializing the dataframe.

```
Cards <- c("Card 1", "Card 2", "Card 3")
Deck <- data.frame(Cards,position=sample(1:3))
Deck <- Deck%>% arrange(position)
```

In order to simulate drawing cards from either the Chance or Community Chest deck, we created a function which takes the card in position 1 of a deck, removes this card from the dataframe, and then insert the same card at the bottom of the dataframe. Afterward, we corrected the position values so that the card that was in position 2 would then be in position 1 and did so for the other cards in the deck. The card that was drawn was then given the last position in the deck. In other words, the function takes the top card, puts it at the bottom of the deck, and leaves the deck ready to be drawn from again.

The Chance and Community Chest decks both include a "Get Out of Jail Free" card which, if drawn, is removed from the deck and is not put back in the deck until the player in possession of the card has used it. In order to incorporate these details, we updated our function to discriminate between the Chance and Community Chest decks and to check if the card being drawn was the "Get Out of Jail Free" card. Additionally, the function checked how many cards were in the deck in the cases when the "Get Out of Jail Free" card was removed, our function would still have functionality. Lastly, we saved the card that was being drawn and returned it along with the new deck so we could then apply the effect of the card to the player. The pseudo code below shows what some of the checks could look like.

Casey Freimund
Felipe Pincheira-Berthelon
Eagan Norman
5/13/19

```
Draw_Card <- function(Deck){

  #Check position and how many cards are in deck
  if(Deck == "Chance" & length(Cards) == 3){
    card=comDeck[1,1]

    if(Deck$Cards[1] == "JailFree"){
    Temporary_Deck <- Deck[-1,]

    Deck <- Temporary_Deck
    Deck$position <- Deck$position + c(rep(-1, 2))
    return(list(comDeck,card))}


  if(!Deck$Cards[1]== "JailFree"){
  card=comDeck[1,1]

  Temporary_Deck <- Deck[-1,]
  Deck <- Temporary_Deck %>% rbind(Deck[1,])

  Deck$position <- Deck$position + c(rep(-1, 2), 2)
  return(list(Deck,card))}
```

At this point, we had a way to draw a card from either deck and the name of the card that was drawn. Next, we needed to program the effects that the cards in the Chance and Community Chest decks would have on the player. To do this, we created two functions, one for the Chance cards and one for the Community Chest cards. We made the function take in the player's profile, the name of the card that was drawn, and both of the decks. From here, we altered the player's current profile appropriately with the corresponding card. For example, if the player drew the "Get Out of Jail Free" card, we updated the Boolean in their profile that kept track of the card to "TRUE" or if a player moved to a new location on the board, we updated their position in their profile appropriately. Additionally, we also programmed the cards that contained changes to player money (i.e. earn $50, pay another player $50, etc.) however, we did not incorporate money in our simulation until later in our simulations. The pseudo code below shows how our Chance function was structured.

```
Chance <- function(player, Card_Name, Chance_Deck,Com_Deck){
  if(Card_Name == "JailFree"){
    player$chance.jailFree <- TRUE

  }else if(Card_Name == "Move to This Space"){
    player$currentSpot <- New_Location
    player=spaceAction(player,freeParking,chanceDeck,comDeck)[[1]]

  }else if(Card_Name == "Get $50"){
      player$money <- player$money + 50
```

In order to apply these actions to a player during a simulation, we created a function that would consider alterations to the player profile and incorporated it in our movement function described earlier above. For example, if we were going to run a simulation and wanted to incorporate the cards and other actions like "Go To Jail" we would reference this new function inside of the movement function when

the player landed on a space that corresponds with an action. So, if a player landed on the space "Go To Jail" this function would update the game and move the player to jail. Likewise, if a player lands on a spot that corresponded to Chance or Community Chest, the function to draw a card would be called for the corresponding deck, then the corresponding Chance or Community Chest function be called using the name of card that was drawn and then the action would be applied to the player afterward. This function that applies actions does this for the rest of the appropriate spaces on the game board as well (i.e. Luxury Tax, Income Tax, Free Parking).

Finally, we can simulate a game incorporating movement, jail, and now movement cards. Similarly to our previous simulation, each player had 10,000 turns and 10 games were simulated. The results of the simulation including actions to the player still have Jail as the most frequent spot, being landed on about 26.7% of the time. This is much different than before incorporating the cards (only 10.97% without cards). While the proportion may have increased for the jail spot, decreasing the proportions for the other spaces, the order of the most landed on spaces is still very similar to our game simulated without movement actions. For example, Chance, Community Chest, Free Parking, Tennessee, and St. James are still all in the top six most frequent spaces along with Jail with the only difference being Free Parking and Tennessee switched positions. Furthermore, when looking at the space colors, we do not see much of a difference either when adding the cards to the simulation. Jail, The Railroads, and Orange spaces are still in the top 3 most frequently landed spaces, Chance and Community Chest are now the 6^{th} and 8^{th} most frequently landed spaces (compared to 8^{th} and 7^{th} respectively), and Free Parking, Go To Jail, Luxury Tax, Go, and Income Tax are still the least frequently visited spaces. So even with adding movement cards into the game, the three most landed on color properties are still Orange, Red, and Yellow.

The last thing we tried to incorporate in the game was money, which is arguably one of the most important parts of the game. In the traditional game of Monopoly, each player starts with $1,500, and the game goes until one of the players has won. Players then use their money to make purchases, make rent payments on owned properties, and pay fees incurred by certain spots on the board. This added the variable "Game Length" into our simulation, which removed our 10,000-round ending clause. However, this also created a problem we hadn't anticipated.

With a general game of Monopoly, the game gets increasingly difficult as players begin to purchase properties. Therefore, the number of squares players can land on without incurring a fee starts to decrease significantly. However, given that our simulation didn't account for property purchasing, players were only losing money when chance and community chest cards had fees, or when players landed on income and luxury tax. This meant that players were losing money much slower than a real game. Additionally, when players would earn money going past the "Go" space, they would gain $200, so for the game to end, each player would have to lose more than $200, on average, each round. Incorporating "Free Parking" also caused a similar issue, but on a larger scale. In order to combat these problems, we took away the $200 reward for making it around the board, and we awarded half the pot when players landed on "Free Parking." The simulations then had the ability to finish in a practical amount of time.

Casey Freimund
Felipe Pincheira-Berthelon
Eagan Norman
5/13/19

The other solution to this problem would be to go the extra step to incorporate property purchasing in the simulation. The first step to this incorporation would be to add the columns for whether a space is purchasable, had been purchased, has any houses or hotels, what the price and rent are, and other factors. Next would be to incorporate a purchase function within the "spaceAction" function to account for purchasable spaces. This would check whether a space is purchasable, and if so, if it had been purchased yet. If the space had been purchased previously, the function would charge the visiting player the appropriate rent. Since each property has specific rent given certain conditions, (I.e. houses, whether the estate is monopolized, how many of that estate is owned, etc.), a function to change the rent values needs to be written for each specific property. This is easier for the railroads and utilities, given they have functions for calculating their rent. However, all other properties have their own specific conditions, given whether the estate has been monopolized, and if it has houses and/or hotels. This doesn't make the code horribly inefficient, but it results in adding more steps to the code than the railroad and utility estates.

While there are a lot of steps and factors in landing on a space that has already been purchased, landing on a space that hasn't been purchased is much more complex. In interactive gameplay, it's as simple as giving the player the choice of whether to purchase or not. However, in a simulation, matching this conscious decision isn't as simple. The simplest way to program it would be to have a player purchase every property they land on, that they can afford. Purchasing would technically be incorporated at this point, but any findings made wouldn't be applicable in a game of Monopoly played against someone with any foresight. A line of code could be added to this though, to make a choice at random to buy the property. Simplistically, this would just be a coin flip, but it could be elaborated on much further. Ideally, a weight system would be incorporated, making properties more likely to be purchased if they were a hotspot, if the player could monopolize that estate, or if they could prevent another player from monopolizing it. These, and other dynamic weight factors could be added to make the game more realistic. To take this concept one step further, a data frame of success factors could be created, and referenced by the weighing function. This data frame could then be updated every time the code was run and passed in before the next time it would be run. This would most closely replicate a player gaining experience from games they have played before and making decisions on which properties are most lucrative.

But, due to time constraints, we were only able to simulate a game of Monopoly that didn't incorporate property purchases. From our simulations, we found that as we started adding more rules, players more frequently landed on the spaces following the jail space. This is a result of the various ways players can end up in jail or from players rolling 2 die, where the average dice roll is between 6 and 8, and even a combination of the two. Based on our findings, we also found that railroad spaces were landed on the most frequently out of any space, due to there being four railroad spaces, evenly spread throughout the board. Our conclusion for Monopoly would be for players to ideally capture and/or monopolize the following spaces, in order from most to least landed; the Rail Roads, the Orange properties, the Red properties, and then Yellow properties. By capturing or monopolizing as many of these spaces as possible, a player will quickly reduce an opponent's cash amount, and become the winner of Monopoly.