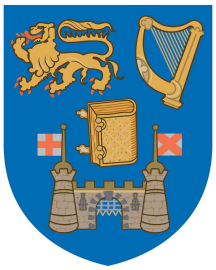
University of Dublin,
Trinity College



ST2004 Optional Assignment

Investigation into Probabilistic Processes of Monopoly

Author:
Edmond O'Flynn 12304742

Lecturer:
Brett Houlding

December 7, 2015

Contents

1 Introduction

1.1 What is Monopoly?

Monopoly is a game where players traverse a board across 40 squares consisting of properties, which players may choose to purchase, as well as other spaces that offer chances and opportunities. Whilst the players move around the board with respect to independent dice rolls, the player may choose to buy and improve properties, or to pay rent to other players. The ultimate goal in the game is to cause the other players to become bankrupt, and to be the tycoon of the board. In this investigation, the American board's layout and tiles will be referred to. I will be investigating the probabilistic outcome of landing on given tiles, proving that not all spaces are equally distributed, as well as expansions related to this exploitation.

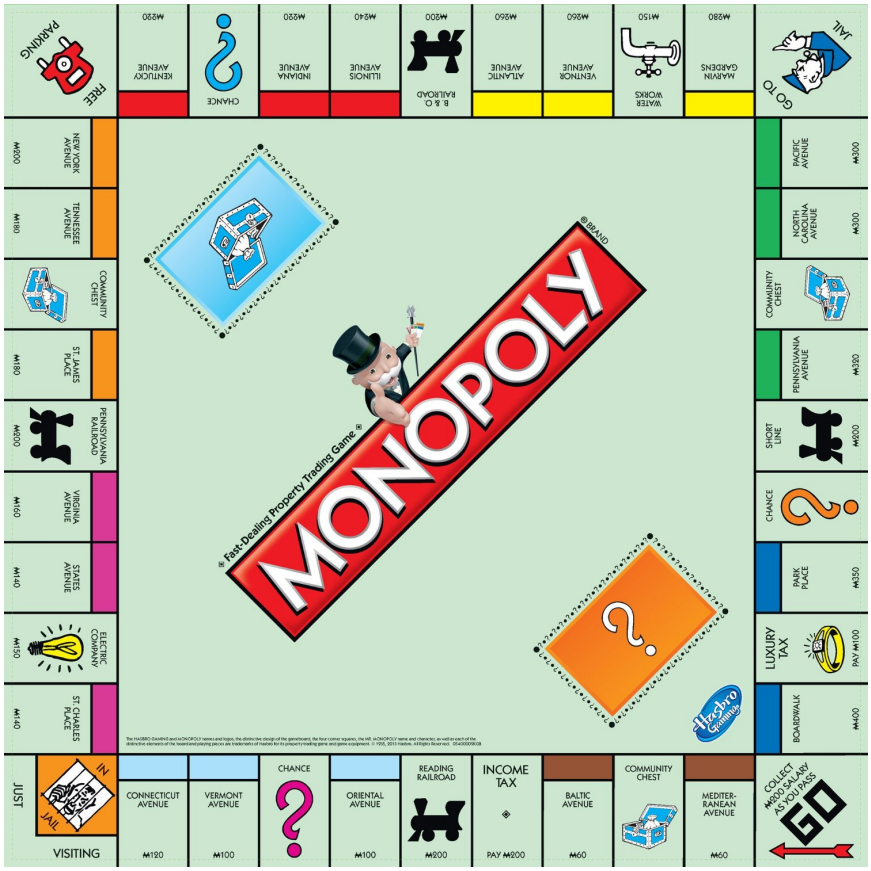


Figure 1: Monopoly board

1.2 Rules

Monopoly board traversal is the result of fundamentally rolling two independent dice and offsetting by the sum of the dice to the next end square.

1. **Players start at Go**
   The player moves with respect to the sum of the dice faces clockwise around the board. If a player rolls a double, ie {1,1}, {2,2} ... {6,6}, the player may roll again.

2. **Players Can Get Sent to Jail**
   On landing on a space, picking up a specific card, or rolling three doubles in a row, players may get sent to the jail tile.

3. **Players May Buy Properties**
   On landing on a non-special tile, a player may choose to buy the land and receive rent if other players

land on it. Players may also develop their properties if and only if they possess all colours within that tile group.

4. **Winning Conditions**
   A player is deemed the winner if the other players become bankrupt, and only one player is left standing.

2 Model & Simulation Background Theory

2.1 Dice Rolls

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Table 1: Dice Roll Outcomes

Given that a double has not been rolled on the first turn of the game, the following is the probabilistic outcome of landing on a square from starting position.

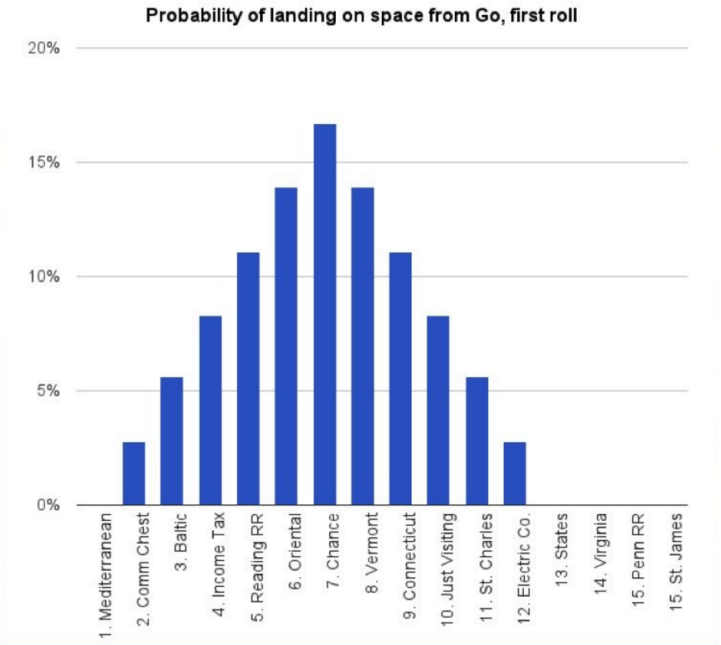


Figure 2: Distribution of Dice without Doubles Rule from Go

Given that the outcome was not a double, there is a significantly higher chance of landing on a chance square, which is comparatively higher than landing on squares Oriental Ave or Vermont Ave. Comparatively so, there is still a higher than 10% chance of landing on squares 5-9.

Doubles can be extremely powerful, thus changing the probabilistic outcomes of square landing probability, allowing for some tiles to obtain higher and lower frequencies than other tiles over the long run. The Monopoly board itself is a vector of length 40 which recursively loops on itself, containing a mixture of tiles ranging from properties to varying cards that affect the player in different ways, both positively and negatively. It is possible to traverse from starting position to the highlighted chance space in one go by rolling 6 6s over 3 turns, its chance of occurrence is exceedingly low, and the player must succumb to the

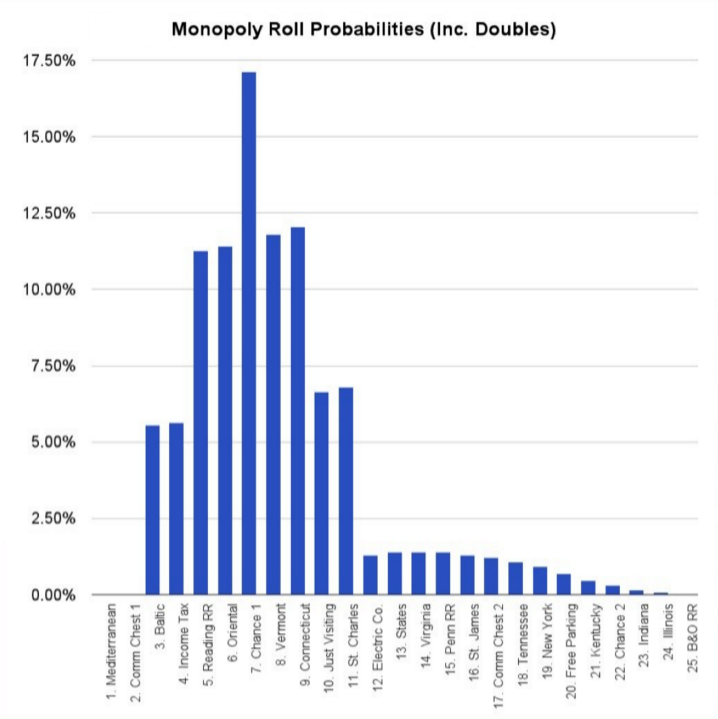


Figure 3: Distribution of Dice Including Doubles Rule from Go

No Speeding rule, by which one must go to jail. The chance of rolling three doubles has a $\left(6 \cdot \frac{1}{6} \cdot \frac{1}{6}\right)^3$, or 0.4%, chance of occurrence. Although small, in every 1,000 rolls there will tend to be 4 triple double-rolls thrown and therefore the player gets sent to jail. Rolling 3 double 6s has a chance of 0.0000214% (or $\left(\frac{1}{6}\right)^6$) of happening, which is probabilistically extremely unlikely, but still possible with fair dice, where it tends to happen twice in every 100,000 rolls.

$$\frac{1}{6} \approx 16.7\% \quad \textit{of rolling 1 double}$$
$$\frac{1}{36} \approx 2.8\% \quad \textit{of rolling 2 doubles}$$
$$\frac{1}{216} \approx 0.4\% \quad \textit{of rolling 3 doubles and ergo get sent to jail}$$

Theoretically it is possible for double-rolls to tend to infinity, albeit exponentially this becomes probabilistically unlikely due to constant decreasing odds as well as the No Speeding rule. The expected value of rolling 1 die is 3.5, therefore on rolling two dice, it's most likely that the sum of the scores will be 7 as independent events. It's also quite likely that your opponents will roll a 5, 6, 8 or 9 with respect to cumulative probabilities of dice rolling, given the figures above.

2.2 Chance and Community Chests

Monopoly contains 16 Community Chest cards, and 16 Chance cards. On landing on a community chest or chance card square, the player must abide by one of the five outcomes of varying probabilities on the cards.

2.2.1 Chance Cards

The 32 cards, 16 from Community Chest and 16 from Chance, drastically change the outcome of what happens within a game. By drawing a card from either deck, one of the following five results may happen:

1. **Go to the Stated Tile**
   There is a $\frac{8}{16}$ chance of this.

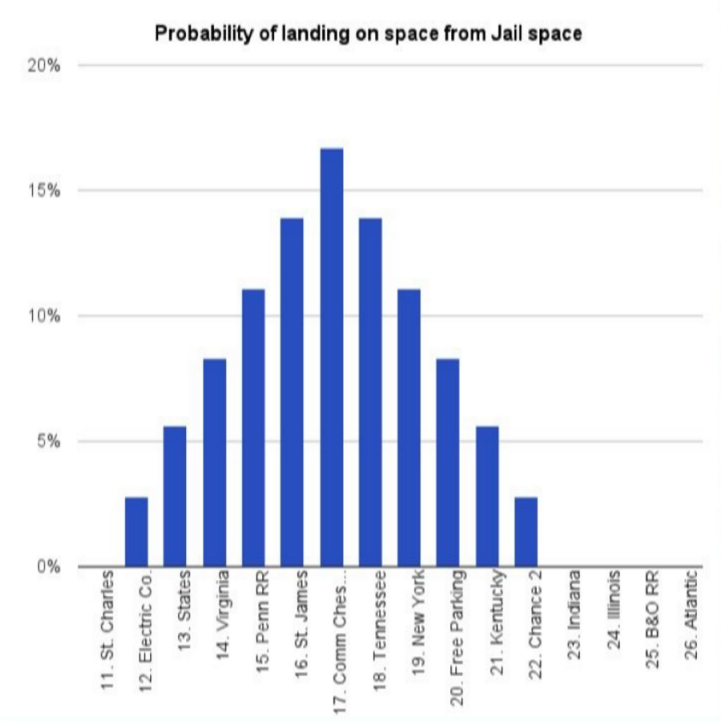


Figure 4: Distribution of Dice without Doubles Rule from Jail

2. **Collect Money**
   There is a $\frac{3}{16}$ chance of this.
3. **Pay Money**
   There is a $\frac{3}{16}$ chance of this.
4. **Go to Jail**
   There is a $\frac{1}{16}$ chance of this.
5. **Get out of Jail**
   There is a $\frac{1}{16}$ chance of this.

Therefore, a probability of $\frac{9}{16}$ cards have the ability to transport the player across the board to another location, thereby altering the odds to being in another place to having a higher probability of landing. There is a total of three possible outcomes of spaces to land on within the weighting effect of certain spaces on the board:

- Go to Jail
- Go Back Three Spaces
- Advance to Go
- Advance to Nearest Utility
- Advance to Nearest Railroad
- Advance to Boardwalk
- Advance to Illinois Avenue
- Advance to Reading Railroad
- Advance to St. Charles' Place

Within the diagram shown, there is a tendency for the game to probabilistically relocate the player more frequently to the first and second rows of the board (due to the selection of cards from Community Chest and Chance), thus creating another weight for frequency on which certain spaces land from card chances alone.

2.2.2 Community Chest Cards

The 16 Community Chest cards have similar adverse effect as Chance cards, with the difference that the 5 outcomes have different weightings:

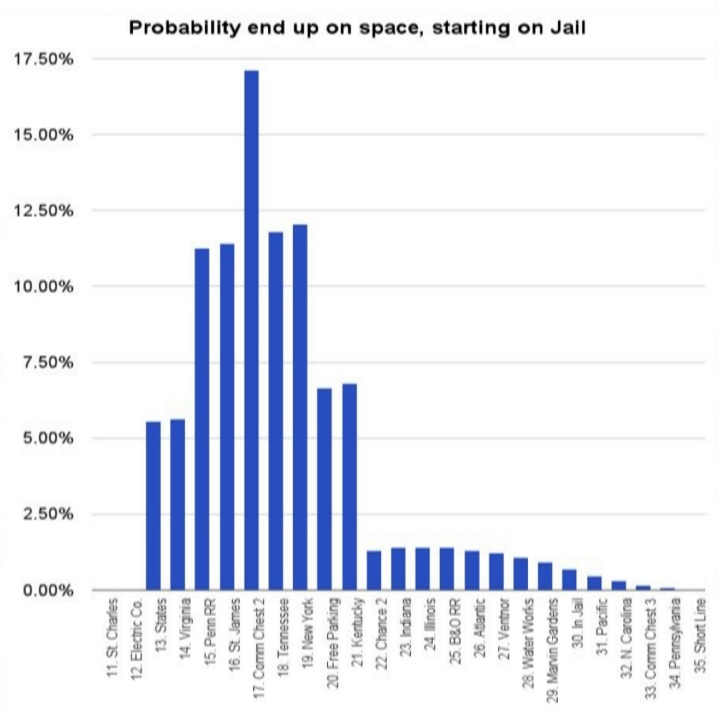


Figure 5: Distribution of Dice without Doubles Rule from Jail

1. **Collect Money**
   There is a $\frac{9}{16}$ chance of this.
2. **Pay Money**
   There is a $\frac{4}{16}$ chance of this.
3. **Advance to Stated Tile**
   There is a $\frac{1}{16}$ chance of this.
4. **Go to Jail**
   There is a $\frac{1}{16}$ chance of this.
5. **Get out of Jail**
   There is a $\frac{1}{16}$ chance of this.

Therefore, mathematically it is possible to anticipate the outcomes of certain cards under the weighting of the *Long Run Average*, where the favourability is an abstraction determined on a scale from -2 to +2:

Action	Favourability	Chance Card	Community Chest
Get out of Jail Free	+2	$\frac{1}{16}$	$\frac{1}{16}$
Collect Money	+1	$\frac{3}{16}$	$\frac{9}{16}$
Advance to Stated Space	0	$\frac{8}{16}$	$\frac{1}{16}$
Pay Money	-1	$\frac{3}{16}$	$\frac{4}{16}$
Go to Jail	-2	$\frac{1}{16}$	$\frac{1}{16}$

Table 2: Favourability Odds of Cards

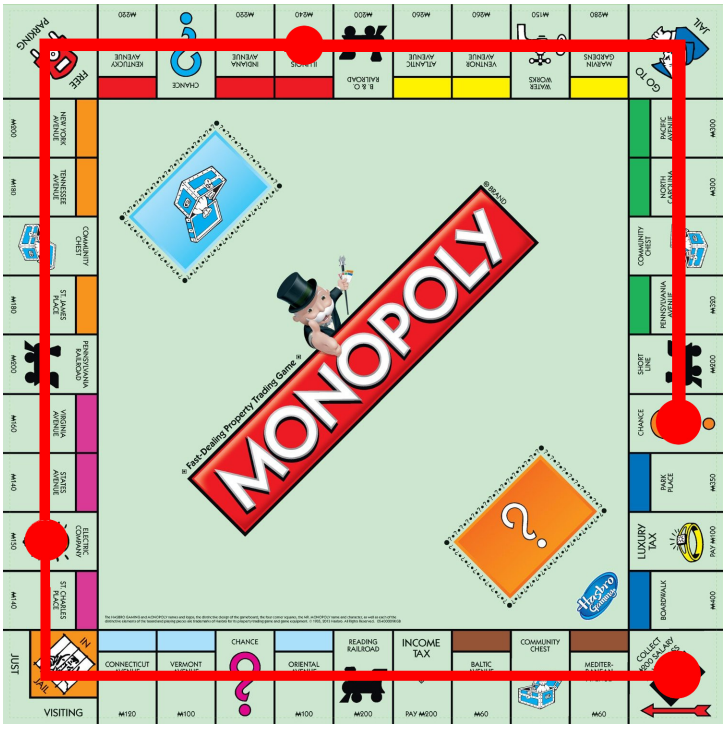


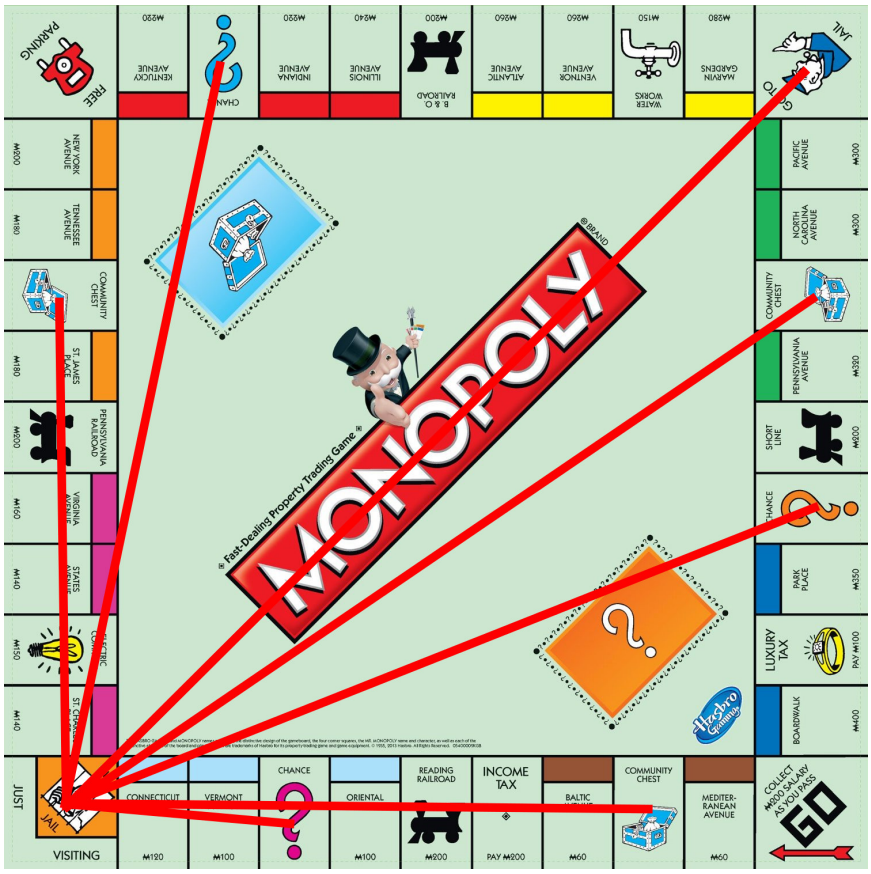
Figure 6: Traversing on 3 Double 6s

2.2.3 Exploitation

Therefore within the confines of Monopoly, it is possible to exploit these weightings in order to buy properties in these areas of higher weightings as well as their neighbouring tiles such that the probabilistic outcome of landing on a space within the vicinity is higher. As previously discussed and shown, the expected value combined with the positioning of Jail and Go allow for properties residing within these areas to have a higher chance of being landed on in comparison to other tiles on the board. Given the expected value of 3.5 per dice where 2 dice are rolled on any given turn, the expected value of the sum of two dice is given to be 7 over the long run.

2.3 Jail

2.3.1 Increasing Odds



Jail in monopoly acts as a hole on the map into which players are released from at a much higher percentage than any other tile on the map. From here, players emerge, and the expected value of dice causes a shift in dynamics in the field of play. Jail in Monopoly is by far the most probabilistic event within the set of cards of Monopoly's scope. Under regular game conditions, there is a total of 4 reasons for which one may get sent to jail:

- No Speeding Rule on Three Doubles
  - $\frac{1}{216} \approx 0.4\%$ chance of occurrence
- Go to Jail Tile
  - $\frac{1}{40} = 2.5\%$ chance of occurrence given no tile bias
- Go to Jail Chance Card
  - Chance of occurrence within the deck is $\frac{1}{16}$, or 6.25%, and $\frac{1}{16} \cdot \frac{3}{40}$, or 0.47% chance given the amount of Chance cards on the map.
- Go to Jail Community Card
  - Chance of occurrence within the deck is $\frac{1}{16}$, or 6.25%, and $\frac{1}{16} \cdot \frac{3}{40}$, or 0.47% chance given the amount of Community Chest cards on the map.

2.3.2 Paradigm

There is a a set of four different ways to attempt to break custody and escape from jail:

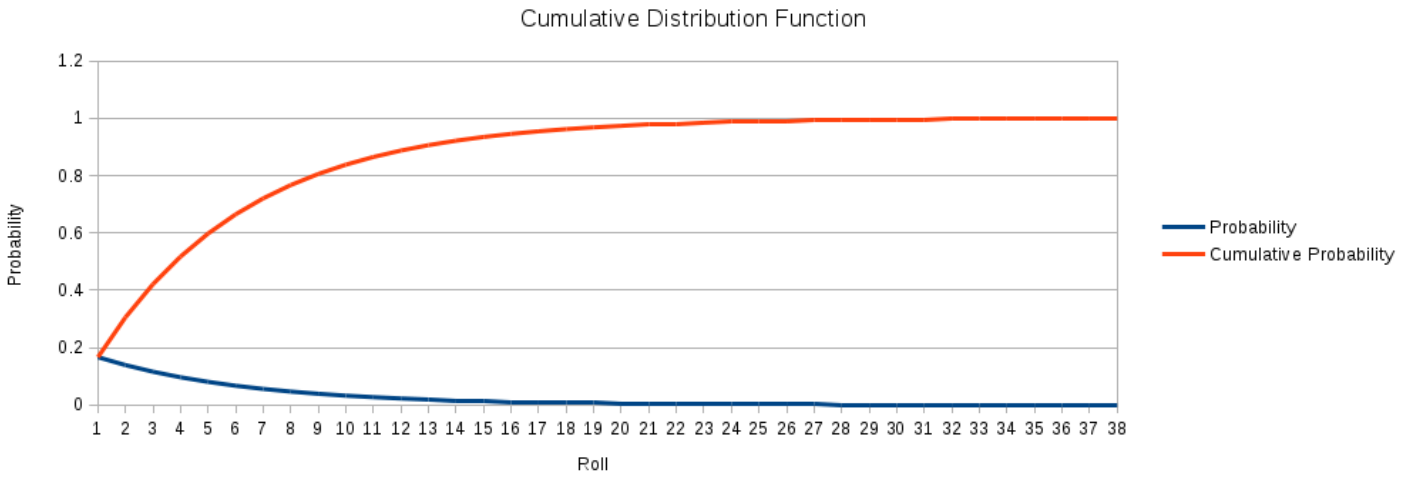
1. Get out of Jail Free Card
2. Pay $50 at the Start of a Turn
3. Roll the Dice and Attempt to Roll a Double
4. Pay $50 on Failure to Roll Doubles for the Third Time

On success of rolling a double out of jail, the player advances that amount of tiles displayed on the dice from the jail tile. Else on failure to roll a double three times, the player must pay $50 and again move the total of spaces displayed on the dice from the previous roll. The cumulative effect of rolling dice means that over the course of three rolls. The important probabilistic factor to note is that getting sent to jail is quite a high factor in the scheme of the game, therefore the game has an explicitly hidden factor of game design to route players through the jail tile as both a starting point and a sink. The probability of rolling out of jail within 3 turns is approximately 42%, given two independent and fair dice as such:

Roll	Process	Probability	Cumulative Probability
1	$\frac{1}{6}$	16.7%	16.7%
2	$\frac{5}{6} \cdot \frac{1}{6}$	13.8%	30.5%
3	$\frac{5}{6} \cdot \frac{5}{6} \cdot \frac{1}{6}$	11.8%	42.3%

Table 3: Cumulative Double Frequency

This distribution gives rise to what is known as a *Probability Distribution Function*, in which the repeated roll in search of doubles will eventually happen given a long enough series of rolls. The probability of the function tends to 1.



3 Markov Chains

3.1 What is a Markov Chain?

Monopoly is based on a 40x40 dimension probability matrix of tiles where each row and column pair represents a space on the board which is subject to a process called a *Markov Chain*. A Markov chain is a probabilistic process in which a steady state probability matrix which is seeded from long term rolls and forecasts. It involves a finite number of states within a finite encapsulation, in this case, the 40 tiles. Markov chains are memoryless, which means that the process in which the player may move freely from

tile to tile is independent of the previous tile via dice rolls. Markovian processes are random process whose future probabilities are based only on the current state, and not on the sequence of events preceding the element.

3.2 Parametrisation

In order to calculate the probabilities per tile, Monopoly must have a set of Markovian properties calculable by some general factors:

- Probability vectors
  - A vector with non-negative entries populated to sum to 1.
- Stochastic matrices
  - A square matrix whose columns are populated by probability vectors.
- Steady state vector
  - Adheres to where the sequence $x_0, x_1, ...x_n$ is a set of probability vectors whereby the stochastic matrix A is such that $x_{n+1} = Ax_n$.
  - Approximates the long term process of the behaviour whereby over subsequent iterations, the Markov Chain tends to $(Ax_{n+1})A = A^2x_{n+1}, A^3x_{n+1}, ..., A^nx_{n+1}$ where n is the period later from the initial time t where in this case, $n = 40$ given the transition matrix below.

3.2.1 Transition Matrices

Transition matrices are a stochastic process within Markovian theory where a square matrix is populated by non-negative real numbers whereby each row sums to 1. Given the element in the i^{th} row and j^{th} column, the probability of moving from state i to state j may be calculated over its long run average within one time step.

		To				
		1	2	3	...	n
From	1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	...	$a_{1,n}$
	2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	...	$a_{2,n}$
	3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	...	$a_{3,n}$
	...	...	...	...	...	...
	n	$a_{n,1}$	$a_{n,2}$	$a_{n,3}$	...	$a_{n,n}$

Table 4: Transition Matrix

The 1,600 transitional probabilities for all tiles within the Markov sequence are encoded in a 40x40 matrix of the probabilities of moving from one tile to another. Given the states, the transition matrix can be made more complex by extending to 42x42, given the intention to stay in jail for up to 3 turns inclusive.

3.2.2 Simplified Chains

In a steady population where the population remains constant, neglecting population changes by immigration, emigration, deaths and births; it is possible to work out the proportions of the population inhabiting each area A, B, and C in which there is a system of change occurring.

For example, assume areas A, B, and C where the following conditions of the system are true on an annual basis of t:

- Of the population within A, 97% will stay in A

  - 1% will more to B from A
  - 1% will move to C from A
- Of the population within B, 95% will stay in B
  - 3% will more to A from B
  - 2% will move to C from B
- Of the population within C, 96% will stay in C
  - 3% will more to A from C
  - 1% will move to B from C

		To		
		A	B	C
From	A	0.97	0.01	0.02
	B	0.03	0.95	0.02
	C	0.03	0.01	0.96

Table 5: Transitional Matrix for A, B, and C

Assuming a population of 1000 in B, given no restrictions on initial population distributions and 0 inhabitants in A and C, time can elapse and the distributive effect of the population spread should be observed, where n is the n^{th} year at which a steady state system has been achieved.

Year	A	B	C
0	0	1000	0
1	30	950	20
2	58.2	903	38.8
...	...	...	...
100	499	168.4	332.6
200	499.998	166.667	333.32
$t \to n$	$1000 \cdot \frac{1}{2}$	$1000 \cdot \frac{1}{6}$	$1000 \cdot \frac{1}{3}$

Table 6: Long Run Observation within the system

Given the system reaches its steady state conditions, the proportions no longer change, and the value at $t+1$, where t has reached steady state, are equivalent in value, therefore

- $P_A = 0.5$
- $P_B = 0.166\dot{6}$
- $P_C = 0.33\dot{3}$

Under conditions, generic equations can be formed for the system, where any two of the three states can be solved via algebraic simultaneous equations forming the steady state proportions.

- $P_A = 0.97 \cdot P_A + 0.03 \cdot P_B + 0.03 \cdot P_C$
- $P_B = 0.01 \cdot P_A + 0.95 \cdot P_B + 0.01 \cdot P_C$
- $P_C = 0.02 \cdot P_A + 0.02 \cdot P_B + 0.96 \cdot P_C$
- $P_A + P_B + P_C = 1.000$

This process can be extended to Monopoly, where instead of a 3x3 matrix, there is a 40x40 matrix of tiles that is immutable and governs steady state proportions by long run averages.

3.3 Seeded Matrices

3.3.1 Assumptions

- On picking up a chance or community chest card, the deck is entirely shuffled in a randomised order having performed the indicated action.
- Jail escape is a process of only escape with respect to rolling doubles or the effect of having picked up a Get Out of Jail Free card prior to getting sent to the tile.
- There are three selective states within Monopoly over 40 tiles:
  - 40 tiles in a regular state
  - Doubles rule with the roll again state
  - Community Chest/Chance cards state
- Therefore there are 120 states in Monopoly in any given turn.

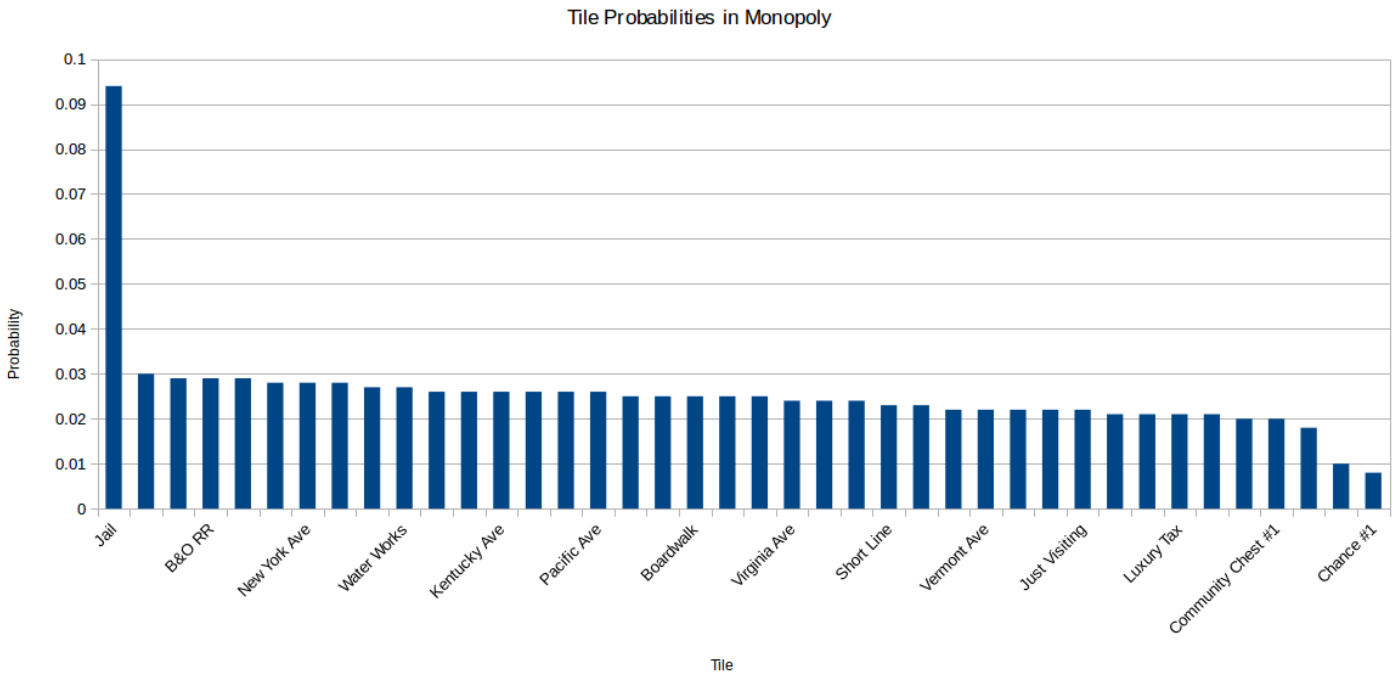
3.4 Simulation

3.4.1 Set Up

In order to quickly simulate a game of Monopoly over an exceedingly large amount of rolls, a simulation of the game was programmed in *Java* with respect to assumptions within the Markovian processes, as well as some generified aspects within the pseudo-random rolling process whereby a pseudo-random number was generated twice per turn using different seed values with respect to the current time in milliseconds on my machine.

3.4.2 Aspects

The Monopoly simulation worked on the bases of simplified aspects where 100,000 turns of play were simulated without the aspect of money or other players involved within the turns, therefore taking only cards, jail and dice values into account as per the effects possible per turn. The following is the list of tiles in order of their probability to be landed on which tends to the long run Markovian averages:



3.5 Tile Probabilities

- 01. Jail - 0.094
- 02. Illinois Ave - 0.03
- 03. Go - 0.029
- 04. B&O RR - 0.029
- 05. Free Parking - 0.029
- 06. Tennessee Ave - 0.028
- 07. New York Ave - 0.028
- 08. Reading RR - 0.028
- 09. St. James' Place - 0.027
- 10. Water Works - 0.027
- 11. Pennsylvania Ave - 0.026
- 12. Electric Company - 0.026
- 13. Kentucky Ave - 0.026
- 14. Indiana Ave - 0.026
- 15. St. Charles' Place - 0.026
- 16. Pacific Ave - 0.026
- 17. Atlantic Ave - 0.025
- 18. Ventnor Ave - 0.025
- 19. Boardwalk - 0.025
- 20. Marvin Gardens - 0.025
- 21. North Carolina Ave - 0.025
- 22. Virginia Ave - 0.024
- 23. Pennsylvania RR - 0.024
- 24. Community Chest #2 - 0.024
- 25. Short Line - 0.023
- 26. Community Chest #3 - 0.023
- 27. Income Tax - 0.022
- 28. Vermont Ave - 0.022
- 29. States Ave - 0.022
- 30. Conneticut Ave - 0.022
- 31. Just Visiting - 0.022
- 32. Oriental Ave - 0.021
- 33. Park Place - 0.021
- 34. Luxury Tax - 0.021
- 35. Baltic Ave - 0.021
- 36. Mediterranean Ave - 0.020
- 37. Community Chest #1 - 0.020
- 38. Chance #2 - 0.018
- 39. Chance #3 - 0.010
- 40. Chance #1 - 0.008

Note that Jail, as forecasted, is the most probabilistic tile through long-run simulation; followed then by Illinois Avenue, which is a total of 14 tiles away from Jail (the expected value over two turns summed: $2 \cdot 3.5 \cdot 2$).
Here is a graphical representation of the degree of simulated probabilities within long run averages of Monopoly within the most probabilistic 20 tiles:

3.5.1 Probabilistic Outcome

Given that the expected value of two dice is 7, it is optimal to purchase as per a threshold of +7 tiles with respect to turn from the jail tile in order to gain the maximum probability of other players landing on it. By maximising the probability of other players landing on tiles owned by the player himself, there is a higher cumulative frequency for receiving rent and the chance of inflicting bankruptcy on others. Note that the second most probabilistic tile is a total of 14 tiles away from jail, which over two rolls corresponds to $4 \cdot E[X]$, or $4 \cdot 3.5$. What is interesting to note is the fact that the lowest probability on the map is also an expected value of the initial dice rolls from *Go*.

3.6 Further Expansion

Given the probabilistic outcome strategy for individual tiles through Markovian logic, it is possible to extend this theory to developing property within the game. Given a probability of landing on a certain tile within a turn as the outcome of two dice, the following inequality must be satisfied in order to ensure a steady income and a lack of bankruptcy:

$$Income \geq Cost$$

This begets an expansion of what revenue really pertains to:

$$p(n) \cdot Rent \cdot E[X] \cdot n \geq Cost$$

Whereby on reordering becomes:

$$n = \left(\frac{Cost}{p(n) \cdot Rent \cdot E[X]} \right)$$

Where n is the current turn of play. Given that properties have the potential to be developed when all three tiles within the colour group are owned, this causes the potential for bankruptcy to increase on landing on a developed tile as rent now increases proportionally with respect to the tile's state of development.

4 Conclusion

Overall a traditional game of Monopoly pertains to probabilistic rules governed by Markovian processes and dice theory. Through simulation and comparison with theory, features that were predicted to occur, were seen through Markov Chain probability matrices. Simulations and theory taken from studies on Monopoly games will differ slightly in comparison to this study, as dice may contain bias by having defects, as well as the addition of money and other players to play against. Simulations for determining the probability of landing on squares were on a single player, infinite rolls basis which is not true to actual gameplay; but nonetheless, provides a good basis for theory on understanding the influence of factors within a game, as well as that of seeing how each tile is not equal, and therefore choices of property determined by Markovian processes can be interpolated to improve chances of winning.

5 Bibliography

References

[1] Jorg Bewersdorff, *Luck, Logic, and White Lies: The Mathematics of Game*, A K Peters (2005), 106-120.

[2] J. Laurie Snell, *Finite Markov Chains and their Applications*, The American Mathematical Monthly (1959), 66 (2), 99-104.

[3] R. B. Ash, *Basic Probability Theory*, (1970), Chapter 7.

[4] Maxine Brady, *The Monopoly Book*, D. McKay, (1974).

[5] W Feller, *An Introduction to Probability Theory and Its Applications, Vol. 1*, Wiley (1950).

[6] Irving R. Hentzel, *How to Win at Monopoly*, Saturday Review of the Sciences, (April 1973), 44-48.

6 Source Code

```
// Dice Rolling
public int rollDice(){
      int result;
      int roll1, roll2;
      roll1 = die1.nextInt(6)+1;
      roll2 = die2.nextInt(6)+1;
      if(roll1 == roll2){
         doublesInARow++;
         if(doublesInARow==3){
            pawn.goToJail();
            doublesInARow=0;
            return 0;
         }
         System.out.println("You rolled doubles!");
      }
      else{
         System.out.println("You did not roll doubles!");
```

```
            doublesInARow=0;
        }
        result = roll1 + roll2;
        return result;
    }

// Handle Traversal
public void takeTurn(Player pawn){
        if(doublesInARow==0){
            pawn.increaseTurns();
        }
        if(pawn.isInJail){
            takeJailTurn(pawn);
            return;
        }
        int moveSpaces = rollDice();
        if(moveSpaces==0){
            return;
        }
        pawn.move(moveSpaces);
        if(!pawn.isInJail && doublesInARow>0){
            takeTurn(pawn);
        }
    }

// Handle Jail Rolling
public void takeJailTurn(Player pawn){
        int jailRoll;
        turnsInJail++;
        if(turnsInJail>2){
            pawn.currentPosition=10;
            pawn.isInJail=false;
            takeTurn(pawn);
            return;
        }
        //use get out of jail card
        else if(pawn.hasJailCard == true && pawn.numberOfTurns < NUMBER_OF_TURNS / 2){
            pawn.currentPosition=10;
            pawn.hasJailCard=false;
            pawn.isInJail=false;
            takeTurn(pawn);
            return;
        }
        jailRoll = rollDice();
        //Check number of turns, if no doubles rolled, sit in jail, otherwise get out
        if(doublesInARow==1){
            pawn.currentPosition=11;
            pawn.move(jailRoll);
            pawn.increaseTurns();
            doublesInARow=0;
            return;
        }
        else{
            pawn.landed(10);
            pawn.increaseTurns();
```

```
            return;
        }
    }

// Handle Board Looping and Special Tiles
public void move(int n){
        currentPosition = currentPosition + n;
        if(currentPosition>40)
            currentPosition = currentPosition - 41;

        if(currentPosition == 31)
            this.goToJail();
        else if(currentPosition == 7 || currentPosition == 23 ||
                currentPosition == 37){
            chance(currentPosition);
        }
        else if(currentPosition == 2 || currentPosition == 18 ||
                currentPosition == 34){
            communityChest(currentPosition);
        }
        else{
            landed(currentPosition);
        }
    }

// Ascertain How Many Times a Tile Has Been Landed On
public void landed(int currentPosition){
        this.landedOn[currentPosition]++;
        System.out.println(this.currentPosition+" has been landed on
            "+landedOn[this.currentPosition]+" times");
    }

// Increment Turns Counter
public void increaseTurns(){
        this.numberOfTurns++;
    }

// Set the Player to Position 11 (Jail Cell)
public void goToJail(){
        this.currentPosition = 11;
        this.landed(11);
    }

// Handle Landing on a Chance Card
public void chance(int currentPosition){
    chan = new Random();
    int card = chan.nextInt(16);
    System.out.println("You pulled card number "+card+" from Chance");
    if(card==0){
        this.currentPosition=0;
        this.landed(0);
    }
    else if(card==1){
        this.currentPosition=25;
        this.landed(25);
```

```
}
else if(card==2){
   this.currentPosition=12;
   this.landed(12);
}
else if(card==3){
   //advance to nearest utility
   if(this.currentPosition==7){
      this.currentPosition=13;
      this.landed(13);
   }
   else{
      this.currentPosition=29;
      this.landed(29);
   }
}
else if(card==4){
   //advance to nearest RR
   if(this.currentPosition==7){
      this.currentPosition=5;
      this.landed(5);
   }
   else if(this.currentPosition==23){
      this.currentPosition=26;
      this.landed(26);
   }
   else{
      this.currentPosition=36;
      this.landed(36);
   }
}
else if(card==6){
   //get out of jail free card
   this.hasJailCard=true;
   landed(this.currentPosition);
}
else if(card==7){
   //go back 3 spaces
   this.currentPosition=currentPosition-3;
   if(this.currentPosition==34){
      this.communityChest(34);
   }
   else{
      landed(this.currentPosition);
   }
}
else if(card==8){
   this.goToJail();
}
else if(card==11){
   //go to reading RR
   this.currentPosition=5;
   this.landed(5);
}
else if(card==12){
```

```
      this.currentPosition=40;
      landed(40);
   }
   else{
      landed(this.currentPosition);
   }
}

// Handle Landing on a Community Chest
public void communityChest(int position){
      comChest = new Random();
      int card = comChest.nextInt(16);
      System.out.println("You pulled card number "+card+" from comChest");
      if(card==0){
         this.currentPosition=0;
         landed(0);
      }
      else if(card==4){
         this.hasJailCard=true;
         landed(this.currentPosition);
      }
      else if(card==5){
         this.goToJail();
      }
      else{
         landed(this.currentPosition);
      }
   }

//Override the Native Print Function to Print a Hash Map in Pairs of Two in Descending Order
@Override
   public String toString(){

      String positionFrequencies="";
      int i;
      int sum=0;
      float totProb=0;

      System.out.println("---------------------------------");
      HashMap<String, Float> map = new HashMap<String, Float>();

      for(i=0;i<41;i++){
         float freq = (float)this.landedOn[i]/Game.NUMBER_OF_TURNS;
         sum = sum + this.landedOn[i];
         map.put(Game.positions[i], freq);
         totProb+=freq;
      }

      Set<Entry<String, Float>> entries = map.entrySet();
      Comparator<Entry<String, Float>> valueComparator = new
          Comparator<Entry<String,Float>>() {
         @Override
         public int compare(Entry<String, Float> e1, Entry<String, Float> e2) {
               Float v1 = e2.getValue();
               Float v2 = e1.getValue();
```

```
            return v1.compareTo(v2);
        }
    };

    List<Entry<String, Float>> listOfEntries = new ArrayList<Entry<String, Float>>(entries);
    Collections.sort(listOfEntries, valueComparator);
    LinkedHashMap<String, Float> sortedByValue = new LinkedHashMap<String,
        Float>(listOfEntries.size());
    for(Entry<String, Float> entry : listOfEntries){
        sortedByValue.put(entry.getKey(), entry.getValue());
    }
    System.out.println("HashMap after sorting entries by values ");
    Set<Entry<String, Float>> entrySetSortedByValue = sortedByValue.entrySet();

    int count = 1;
    for(Entry<String, Float> mapping : entrySetSortedByValue){
        System.out.println(count + ": " + mapping.getKey() + " ==> " + mapping.getValue());
        count++;
    }
System.out.println("---------------------------------");
    System.out.println("Summation of probabilities: " + totProb);
    System.out.println("Number of spaces per turn: "+(float)(sum/this.numberOfTurns));
    System.out.println("Total number of turns: "+this.numberOfTurns+"\nTotal number of
        landed ons: "+sum);
    return positionFrequencies;
}
```