Project Motivations

This is the report produced in 2019 for *MAS1802: Problem Solving and Computing II,* a first-year undergraduate module at Newcastle University.

Section 1:

We wish to determine which squares a player is most likely to land on during a game of Monopoly. To simplify things, we assume there is only a single player, we ignore everything to do with money, and we also ignore the 'Get out of Jail Free Cards'.

Square Number	Name	Square Number	Name
1	Go	11	Jail
2	Old Kent Road	12	Pall Mall
3	Community Chest	13	Electric Company
4	WhiteChapel Road	14	Whitehall
5	Income tax	15	Northumberland Avenue
6	King's Cross Station	16	Marylebone station
7	The Angel Islington	17	Bow Street
8	Chance	18	Community Chest
9	Euston Road	19	Marlborough Street
10	Pentonville Road	20	Vine Street
21	Free Parking	31	Go To Jail
22	Strand	32	Regent Street
23	Chance	33	Oxford Street
24	Fleet Street	34	Community Chest
25	Trafalgar Square	35	Bond Street
26	Fenchurch Street Station	36	Liverpool St Station
27	Leicester Square	37	Chance
28	Coventry St	38	Park Lane
29	Water Works	39	Super Tax
30	Piccadilly	40	Mayfair

Table 1: *Monopoly squares with associated square numbers*

Section 2:

Last year's Panini sticker album for the FIFA '18 World Cup football finals had a whopping 682 stickers to collect, including 40 awesome glitter stickers! Each packet of stickers cost 80p and contained 5 stickers.

The obvious question to ask is:

"How much does it cost, on average, to fill a Panini World Cup '18 sticker album?"

MAS1802: Computing Project

Due on Tuesday 14th May at 4:00pm

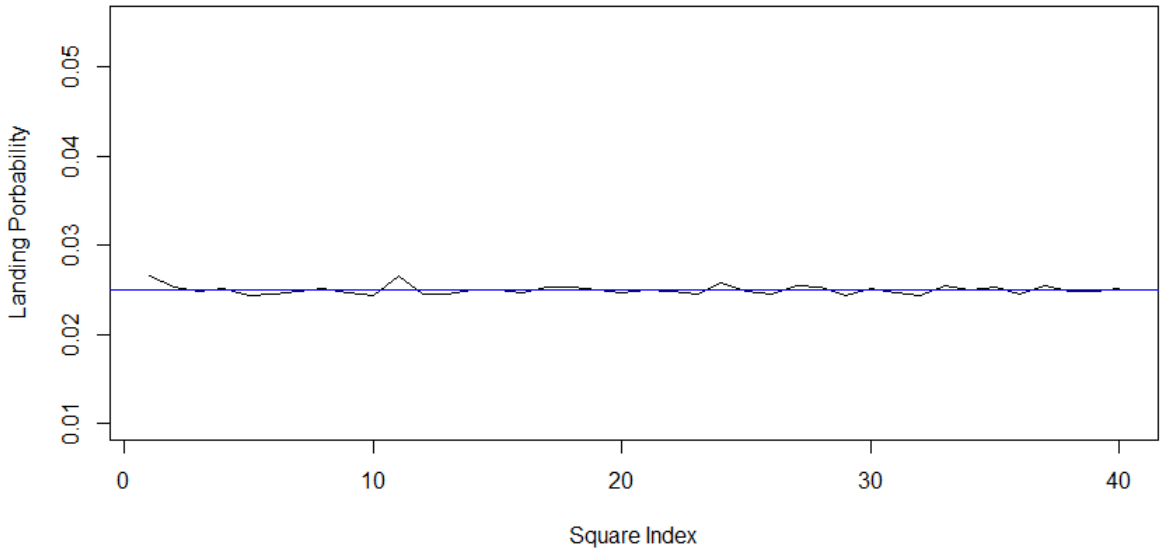
Dr Lee Fawcett

Joseph Marks

Section 1: Monopoly

Initial Simulation

Below is a plot of `SimulateMonopoly1`, showing a baseline probability of 2.5% to land on any given square.



Adding further complexity

Main Function: SimulateMonopoly

The `while` loop on line 6 freezes the roll counter if the player rolls a double as to simulate 'free rolls', and to prevent the case of the simulation ending on a double where it is possible two further doubles could result in being sent to jail.

```
1  SimulateMonopoly = function(no_of_rolls) {
2    landings = numeric(40)
3    current = 1
4    doubles = 0
5    for (i in 1:no_of_rolls) {
6      while (TRUE) { # Stops 'i' from incrementing on doubles
7        roll = RollTwoDice()
8        if (doubles == 3) {
9          current = 11 # Send to jail on rolling three doubles
10         doubles = 0 # Reset doubles counter
11       } else {
12         current = current + sum(roll)
13         if (roll[1] == roll[2]) {
14           doubles = doubles + 1 # Increment doubles counter on rolling a
                 double
15         }
16         if (current > 40) {
17           current = current - 40
18         }
19         landings[current] = landings[current] + 1
20        # Community Chests Squares
21         if (current == 3 || current == 18 || current == 34) {
```

```
            cc_move = CommunityChest(current)
            if (cc_move != current) {
              current = cc_move
              landings[current] = landings[current] + 1
            }
          }
          # Go to Jail
          if (current == 31) {
            jail = 11
            current = jail
            landings[current] = landings[current] + 1
          }
          # Chance Squares
          if (current == 8 || current == 23 || current == 37) {
            chance_move = Chance(current)
            if (chance_move != current) {
              current = chance_move
              landings[current] = landings[current] + 1
            }
          }
        }
        if (roll[1] != roll[2]) {
          break # Increment 'i' if a non-double is rolled
        }
      }
    }
    return(landings)
}
```

Helper Function: Community Chest

```
CommunityChest = function(current) {
  goto = current
  u = runif(1) # Create random number between 0 and 1
  if (u < 1 / 16) {
    goto = 1 # Advance to Go
  } else if (u < 2 / 16) {
    goto = 11 # Go to Jail
  } else if (u < 3 / 16) {
    goto = 2 # Go to Old Kent Road
  } else if (u < 4 / 16) {
    goto = Chance(current) # Take a chance card
  }
  return(goto)
}
```

Helper Function: Chance Cards

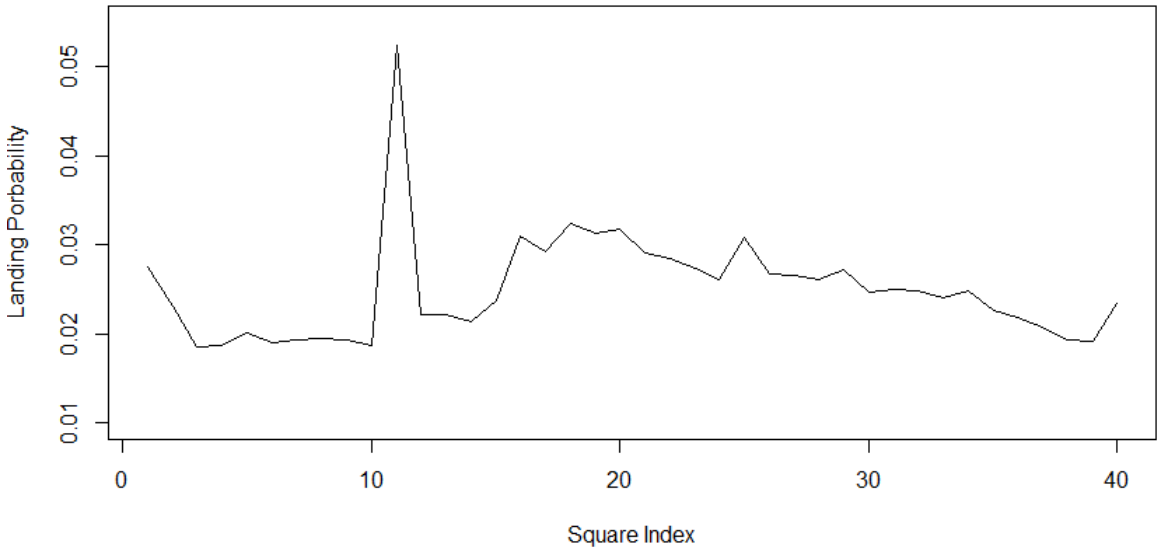
```
Chance = function(current) {
  goto = current
  u = runif(1) # Create random number between 0 and 1
  if (u < 1 / 16) {
    goto = 1 # Advance to Go
  } else if (u < 2 / 16) {
    goto = 25 # Advance to Trafalgar Square
  } else if (u < 3 / 16) {
    goto = 12 # Advance to Pall Mall
  } else if (u < 4 / 16) {
    goto = 11 # Go to Jail
  } else if (u < 5 / 16) {
    goto = 16 # Take a trip to Marylebone Station
  } else if (u < 6 / 16) {
    goto = 40 # Advance to Mayfair
  } else if (u < 7 / 16) {
    if (current >= 1 & current <= 13) {
      goto = 13 # Advance to nearest utility
    }
    if (current >= 14 & current <= 29) {
      goto = 29 # Advance to nearest utility
    }
    if (current >= 30 & current <= 40) {
      goto = 13 # Advance to nearest utility
    }
  } else if (u < 8 / 16) {
    goto = current - 3 # Go back three spaces
  }
  return(goto)
}
```

Helper Function: Rolling Dice

```
RollTwoDice = function() {
  d1 = sample(1:6, 1)
  d2 = sample(1:6, 1)
  roll = c(d1, d2)
  return(roll)
}
```

Final plot

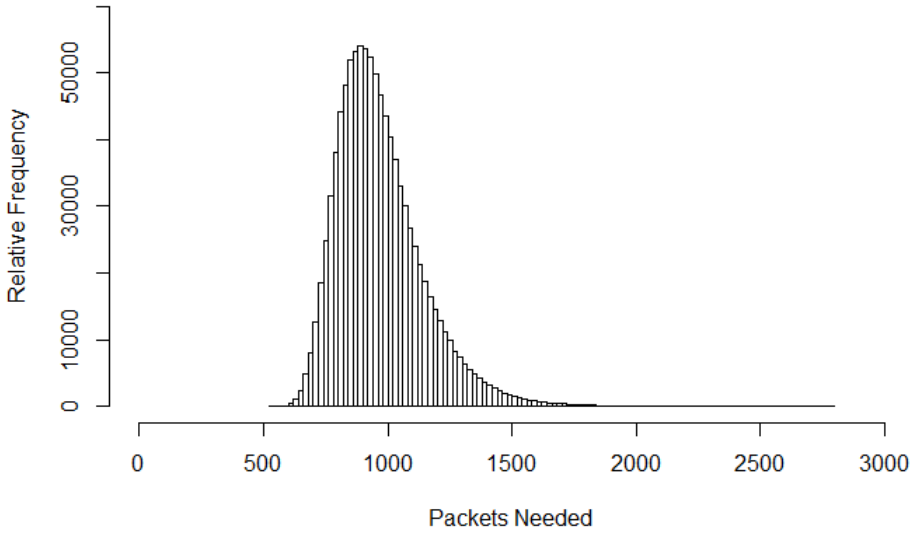
The final plot was carried out with `number_of_rolls=100,000,000`. The final results suggest landing on the Jail square had the highest likelihood, of 6.42%, whereas the lowest likelihood was to land on the Community Chest at square 3, with a probability of 2.27%. The average probability of landing on any given square was still 2.5%, meaning on average it is over twice as likely to land on jail than any other given square.



Section 2: Sticker Album

Main Function

Across 1,000,000 simulations, the average amount of packets needed for a full set was 969.417. This would suggest an average cost of £775.53. In these simulations, in the most unlucky case it took 2789 packets to complete a set, totalling £2231.20, and in the luckiest case it took only 538 packets, costing £430.40. The standard deviation for packets needed was 174.15, implying a £139.32 standard deviation for cost.




```
sim_number = 1000000
packets_bought = vector(length = sim_number)

sticker_sim = function(sim_number) {
  for (i in 1:sim_number) {
    stickers = 1:682 # Sticker sequence, a vector containing an entry for each
        sticker.
    for (j in 1:10000) { # Open up to 10,000 packs of stickers (arbitrary
       upper limit).
      packet = sample(1:682, 5, replace = TRUE) # Create a packet from a
          random pool 682 of stickers, with replacement.


      for (k in 1:5) { # Pick 5 stickers from the random pool, find the picked
          stickers' entry in 'stickers' vector and set to zero.
        stickers[packet[k]] = 0
      }

      # If all entries in 'stickers' vector are zero then all stickers have
          been collected, stop looping.
      if (sum(stickers) == 0) {
        packets_bought[i] = j
        break
      }
    }
```

```
21    }
22    return(packets_bought)
23  }
24
25  output_vector = sticker_sim(sim_number)
26  sum(output_vector) / sim_number # Outputs average number of packets needed for
        a full set
```