Snakes and Ladders Markov chain analysis

Daan Noordenbos

2023-07-07

Introduction

Snakes and Ladders is a classic children's board game played by two or more players on a 10 by 10 grid with a die. The game is stochastic as play is completely governed by chance, making it suitable for analysis using Markov chains. Each player has a token placed on the starting square at the beginning of the game. On their turn, players roll the dice and move their token forward along the numbered squares based on the number rolled. If a player's token lands at the base of a ladder, it moves up to the top of the ladder. If the token lands on the head of a snake, it moves down to the tail of the snake. If a player rolls a six, they get another turn and roll the die again. The game ends when a player reaches the last square exactly. If a player overshoots the last square, they move the remaining squares backwards.

Implementation and simulation

To implement the game we need to be able to deal with the snakes and the ladders, we achieve this by creating a mapping, $m : \text{squares} \times \text{rolls} \to \text{squares}$, that takes as input the dice and the number rolled and outputs the destination square. This mapping is easily made using a lookup array.

```r
board <- c(1:105)
# Snakes
board[99] <- 78
board[95] <- 75
board[93] <- 73
board[87] <- 24
board[64] <- 60
board[62] <- 19
board[54] <- 34
board[17] <- 7
# Ladders
board[4]  <- 14
board[9]  <- 31
board[20] <- 38
board[28] <- 84
board[40] <- 59
board[51] <- 67
board[63] <- 81
board[71] <- 91
# Overshooting
board[101] <- board[99]
board[102] <- board[98]
```

```
board[103] <- board[97]
board[104] <- board[96]
board[105] <- board[95]
```

The specific placement of the snakes and ladders depends on the version of the game and can be modified. With m we can simulate the turns and simulate a game aswell.

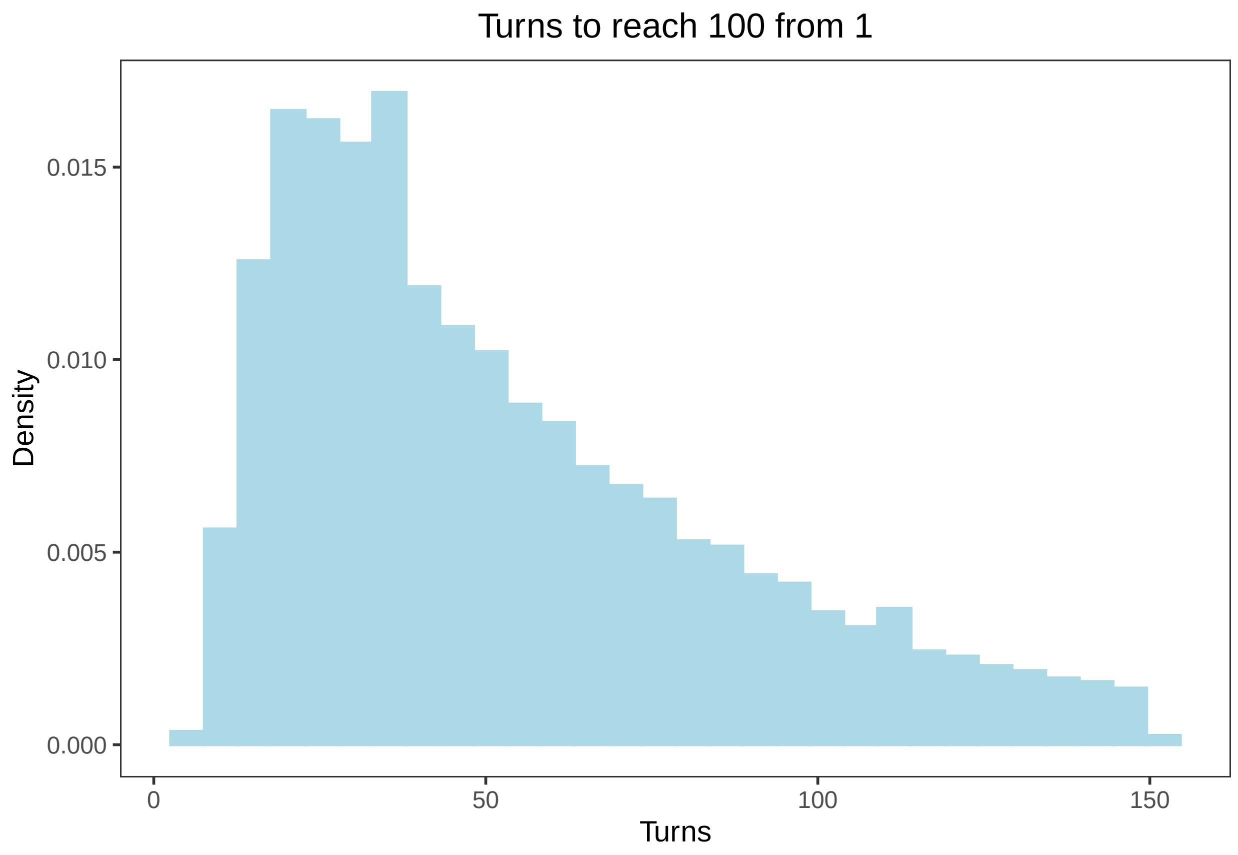
```
nextSquare <- function(square)
{
  roll <- floor(runif(1, 1, 7))
  newSquare <- board[square + roll]
  while (roll == 6 && newSquare != 100)
  {
    roll <- floor(runif(1, 1, 7))
    newSquare <- board[newSquare + roll]
  }
  return(newSquare)
}

sampleGame <- function(start = 1)
{
  square <- board[start]
  game <- c(1)
  while (square != 100) {
    square <- nextSquare(square)
    game <- c(game, square)
  }
  return(game)
}
```

A sample game will look as follows:

```
sampleGame()
```

```
##  [1]   1   3   8  11  13   7  11  12  14  18  22  26  30  34  38  49  50  55  81
## [20]  83  84  85  86  88  73  77  24  29  32  35  37  59  61  60  68  72  74  76
## [39]  81  83  24  27  29  30  31  33  38  59  61  88  90  91  73  78  79  84  85
## [58]  90  92  97  78  81  82  24  29  45  49  50  67  72  77  82  86  89  90 100
```

Using only simulated games we can already do some preliminary analysis, for example using 50 thousand simulated games we find the following distribution,



Markov chain analysis

However, we do not need to settle for an approximation, we can determine these probabilities exactly. To do so we need to make the following observation, if we are at square a then the probability of reaching square b on our turn is given by:

$$\begin{aligned}\mathbb{P}(a \to b) &= \frac{1}{6}\sum_{r=1}^{5} \mathbb{1}\{m(a+r) = b\} + \frac{1}{6}\mathbb{P}(m(a+6) \to b) \\ &= \frac{1}{6}\sum_{r=1}^{5} \mathbb{1}\{m(a+r) = b\} + \frac{1}{36}\sum_{r=1}^{5} \mathbb{1}\{m(m(a+6)+r) = b\} + \frac{1}{36}\mathbb{P}(m(m(a+6)+6) \to b).\end{aligned}$$

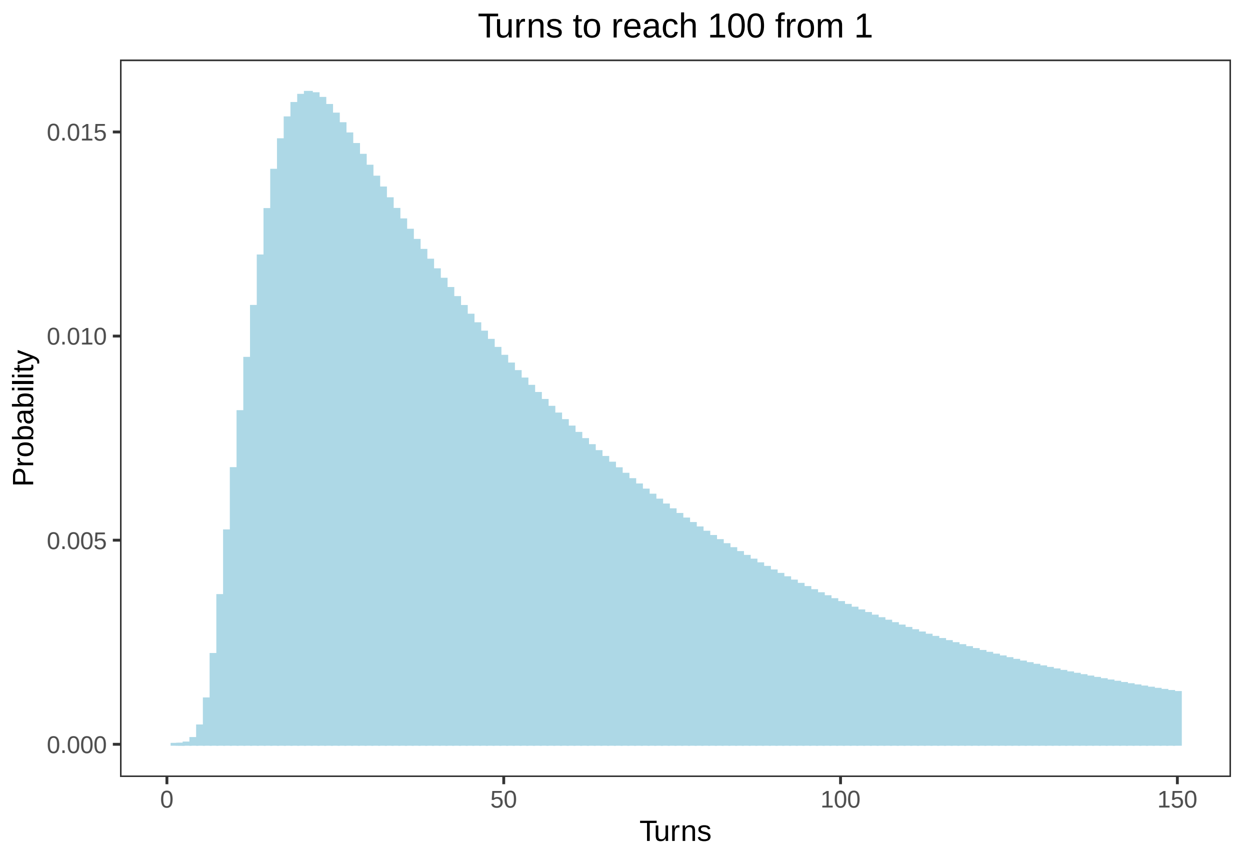
From the recursive structure it becomes apparent that we always have an unknown probability on the right side of the equation, but that the contribution of this term shrinks exponentially. We will cut the recursion of after 30 steps, because $\frac{1}{6}^{30}$ is smaller than machine epsilon, so it is as exact as it can be. The final function to calculate $\mathbb{P}(a \to b)$ is therefore

```
probabilityAToB <- function(a, b, depth = 30)
{
  if (a == 100 || depth == 0)
  {
    return(as.numeric(b == 100))
  }
  return(sum(as.numeric(board[a + c(1:5)] == b)) / 6 +
```

```r
         probabilityAToB(board[a + 6], b, depth - 1) / 6)
}
```

With the transition probabilities we create a transition matrix and determine the probability of reaching 100 from any square in at most k turns.

```r
transitionMatrix <- matrix(apply(expand.grid(1:100, 1:100), 1,
                                  function(x) probabilityAToB(x[1], x[2])),
                            nrow = 100, ncol = 100)

# cdfs[k, square] is the probability of reaching 100 from 'square'
# in at most 'k' turns
cdfs <- matrix(0, ncol = 100, nrow = 1000)
multipleTransition <- transitionMatrix

for (i in 1:1000)
{
  cdfs[i, ] <- multipleTransition[, 100]
  multipleTransition <- multipleTransition %*% transitionMatrix
}

pdfs <- matrix(0, ncol = 100, nrow = 1001)
for (s in 1:100)
{
  pdfs[, s] <- c(cdfs[, s], 1) - c(0, cdfs[, s])
}
```



Win probabilities

Lastly, what remains is the probability of winning the game given the current game state. This can be calculated easily. Let 1 through n represent the players, x_i be the square the token of player i is on, and T_i be a random variable for the amount of turns until i reaches 100. Taking the order in which players move in account we find that,

$$\begin{aligned}
\mathbb{P}(i \text{ wins}) &= \mathbb{P}(T_1 > T_i, \ldots, T_{i-1} > T_i, T_{i+1} \geq T_i, \ldots, T_n \geq T_i) \\
&= \sum_{k=1}^{\infty} \mathbb{P}(T_i = k) \prod_{j=1}^{i-1} \mathbb{P}(T_j > k) \prod_{j=i+1}^{n} \mathbb{P}(T_j \geq k) \\
&= \sum_{k=1}^{\infty} \mathbb{P}(T_i = k) \prod_{j=1}^{i-1} (1 - \mathbb{P}(T_j \leq k)) \prod_{j=i+1}^{n} (1 - \mathbb{P}(T_j \leq k) + \mathbb{P}(T_j = k)).
\end{aligned}$$

Note that we can calculate $\mathbb{P}(T_i \leq k)$ as it is given by `cdfs[k, x_i]`. Therefore, the code which computes the winning probabilities for each player is as follows:

```
winProbabilities <- function(tokens)
{
  n <- length(tokens)
  res <- rep(0, n)
  for (i in 1:n)
  {
    probability <- 0
    for (k in 1:1000)
    {
      late <- 1
      for (j in setdiff(c(1:n), i))
      {
        late <- late * (1 - cdfs[k, tokens[j]] +
                        ifelse(j > i, pdfs[k, tokens[j]], 0))
      }
      probability <- probability + pdfs[k, tokens[i]] * late
    }
    res[i] <- probability
  }
  return(res)
}
```

The validity of this is easily verified by simulation.

```
winProbabilitiesEstimate <- function(positions)
{
  n <- 10000
  freq <- rep(0, length(positions))

  for (i in 1:n){
    ind <- which.min(sapply(positions, sampleGameLength))
    freq[ind] <- freq[ind] + 1
  }
  return(freq / n)
}
```

```r
tokens <- c(14, 60, 31, 26, 48, 48)
winProbabilities(tokens)
```

```
## [1] 0.1328226 0.2318284 0.1391085 0.1554380 0.1718596 0.1689429
```

```r
winProbabilitiesEstimate(tokens)
```

```
## [1] 0.1360 0.2315 0.1353 0.1544 0.1720 0.1708
```