By the Numbers

Mathematical Analysis of *Snakes and Ladders*

If you're a parent, the chances are high that you have spent quite a few hours playing the board game *Snakes and Ladders* (or *Chutes and Ladders* as it is known in the USA). It's a simple game with origins that can be traced back to the 16th century. There's no skill required to play, so for parents games seem to go on forever! But just how long does an average game really last? This article will explore two mathematical methods for determining that answer.

Although performing analysis of a simple child's game like *Snakes and Ladders* might seem trivial, I hope it will demonstrate beneficial techniques you can apply to the analysis of the more complex games you are developing. For example, if your game involves the use of some form of currency, and you have not correctly modeled some aspect of your game, or if your probabilities are too loose, the resultant flood of currency could cause rampant and undesirable inflation.

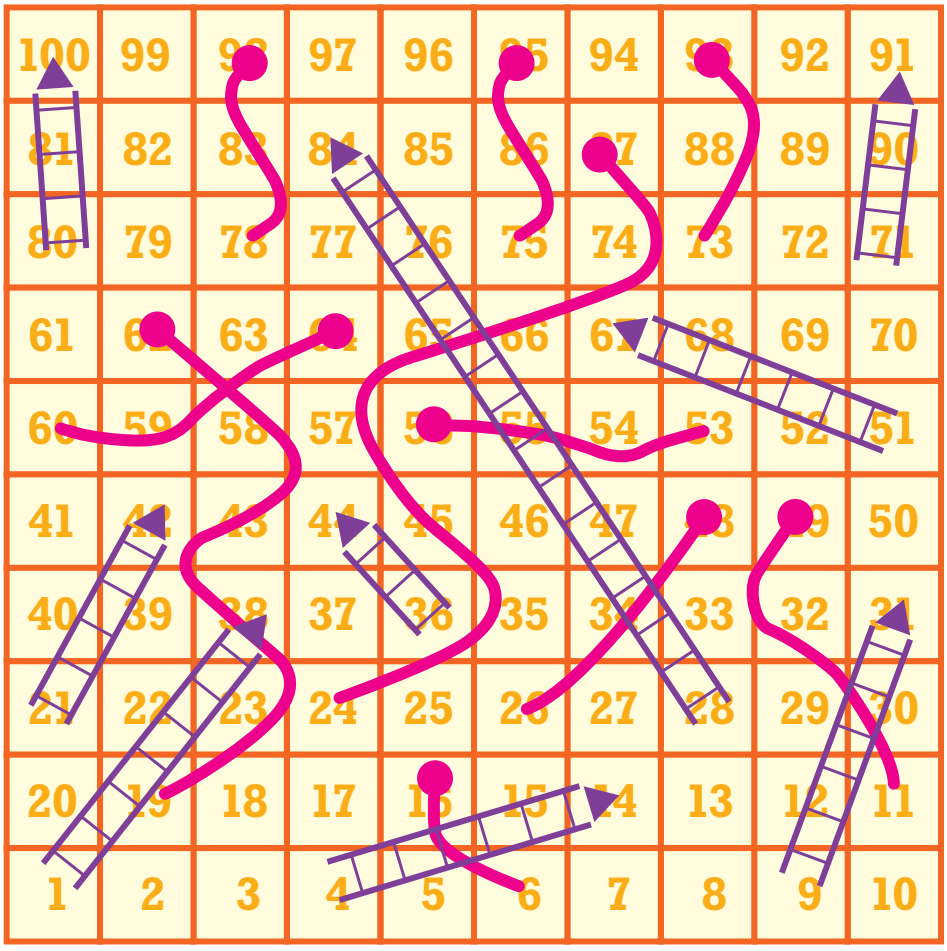


figure 1

Rules

The game of *Snakes and Ladders* is played on a board with a 10x10 grid, numbered sequentially in a zigzag pattern from 1 (the start, in the lower left corner) to 100 (the end, in the top left corner). At various locations on the board are placed snakes (or chutes) and ladders, each of which connects a pair of squares. A representation of the board is shown in *Figure 1.* (Snakes are shown in pink, with a dot representing their heads, and ladders in purple with a point showing the direction of travel.

All players start off the board and take turns rolling a single die and moving the corresponding number of squares. If the completion of a move lands you at the foot of a ladder, you instantly climb to the top of that ladder. Likewise, if a move lands you on the head of a snake, you are forced to slide down the snake to an earlier square. There is no consequence to landing on the top of a ladder or on the tail of a snake; snakes and ladders are one-way passages. Mathematically, this is called a *Directed Graph*. The first player to square 100 wins. (In the version of rules we play, and in this analysis, an exact roll is *not* required to finish).

Analysis

Because landing on a snake can send you backwards and, if you are incredibly unlucky, you can land on another snake, and another, and so on) there is no theoretical upper limit to the number of moves a game can take. Practically, however, as we will see later, the probability that a game will last more than a couple of hundred moves falls to essentially zero (99.97 percent of games finish in 200 moves or fewer).

To compute the average length of a game, we need to create a table showing the total number of moves to finish a game and what percentage of games finish in that number of moves. Since *Snakes and Ladders* requires no interaction between the players (even when they share the same square), a player's moves

by Nick Berry;
President, DataGenetics
Seattle, WA

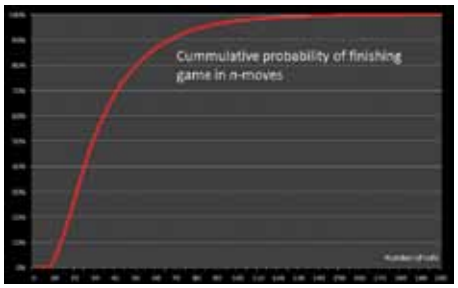


figure 2

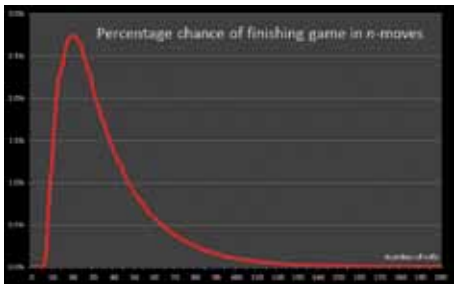


figure 3

are independent. We can use the fact to simplify the game and only consider the moves of a single player.

We're going to compare two methods of calculating the moves required to complete a game: *Monte-Carlo Simulation*, and *Markov chain Analysis*.

Monte-Carlo Simulation

A Monte-Carlo simulation (named after the casino of the same name) is simple and easy to describe: The game is programmatically modeled and, using a random number generator to simulate the dice rolls, played to completion. The results are noted, and then the game is played again, and again, and again. Over many simulations, the more likely results appear more often, and the less likely results less often and so, proportionally, the results of the simulation give an estimate of the relative probabilities of each outcome. The more samples or experiments that are run, the higher the confidence in your results.

Figure 2 shows the results of simulating a billion games of *Snakes and Ladders*. The x-axis shows the number of rolls, and the y-axis shows the percentage of games that are completed in that number of moves or fewer. Interestingly, no game can finish in less than seven rolls; this is the smallest number of rolls required to win. (There are multiple ways this can be achieved. One such solution: rolls of 4, 6, 6, 2, 6, 6, and 4.) All games that finish in seven moves require the use of the longest ladder. A game finishes in seven moves approximately twice in every thousand games played. More than three quarters of all games are completed in 46 moves or fewer.

Figure 3 is a plot of the same data, but this time showing the percentage of games that are completed in that exact number of moves, instead of the cumulative probability.

Average

Statisticians have many different mechanisms to describe "average." The most common are: *Median, Mode* and *Arithmetic Mean*.

The *mode* is defined as the most popular outcome. Looking at *Figure 3*, we can see that the peak occurs at 20 rolls. Over time, the most frequently occurring number of rolls will be 20.

The *median* makes reference to the midpoint. Looking at *Figure 2*, we can see that our distribution crosses the 50 percent line at 29. What this means is that, over time, as many games will take *more* than 29 moves as will take *fewer* than 29 moves.

The *arithmetic mean* is the total of all the rolls made, divided by the number of games. To simulate one billion games, I rolled a total of 36,203,113,317 dice, resulting in an average number of rolls per game of approximately 36.2.

Limitations

Monte-Carlo simulations are easy to code because they don't require you to understand the intricacies of the underlying math. They can be very efficient tools to test and tweak parameters in games without having to perform complex theoretical computations. However, they do have limitations.

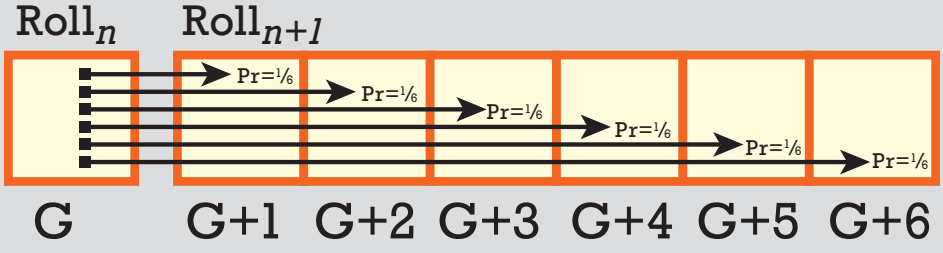
First, you are at the mercy of your random number generator: If your random number generator is biased, your results will also be biased. Second, sufficient iterations have to be made to ensure that you have accurate enough results. Finally, events that occur infrequently need to be exercised amply enough to be properly and proportionally represented in the probabilities. Depending on the complexity of your model, these last two items might make Monte-Carlo simulations too much of a burden.

Markov chain Analysis

Markov chain analysis is a way of modeling a system that allows the calculation of an exact answer.

At the heart of Markov chain analysis is the concept of a *stochastic* process. This is just a fancy set of words to say that, from a given state, there are a series of possibilities that could happen next, and these possibilities are defined by a probability distribution. (Implied in the definition is that the sum of all the probabilities is 100 percent—something *will* happen next).

figure 4



Games like *Snakes and Ladders* are ideal candidates for Markov chain analysis because, at any time, the probabilities of events that will happen in the future are unaffected by what happened in the past. If a player is on grid square 18 of the board, the probability of what will happen on the next roll is independent of how the player got to square 18.

In *Figure 4*, if a player is at grid square G when he rolls, one of six things could happen (with equal probability), and based on these probabilities the player would advance to one of the next squares. These probabilities can be represented as a sparse matrix which records the probability of moving from position

Mathematical Analysis of *Snakes and Ladders*

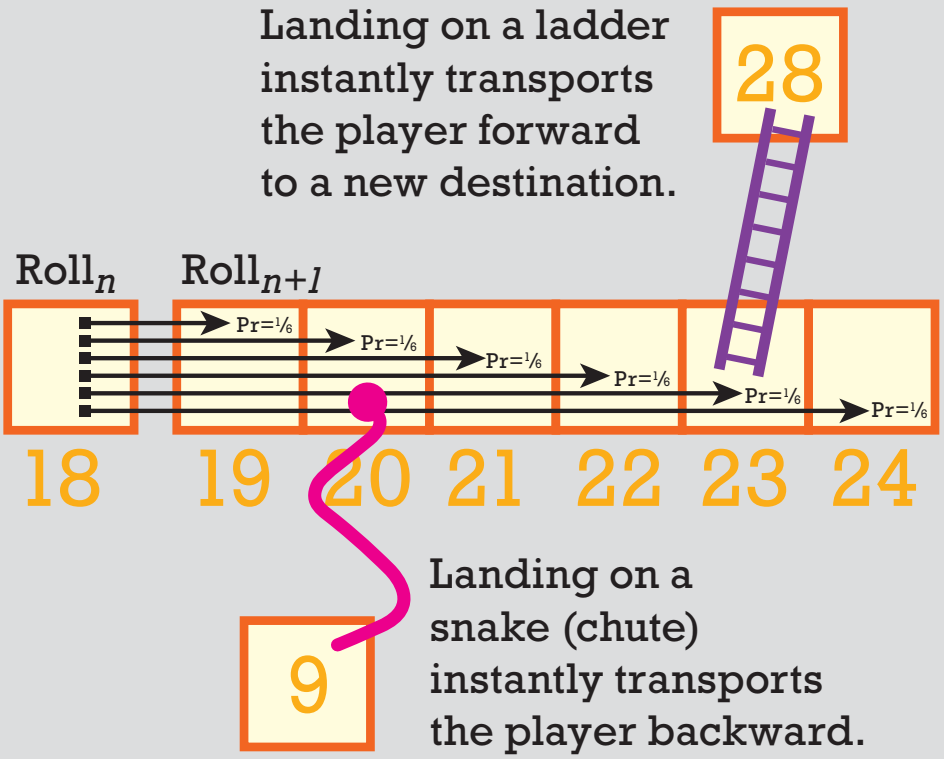


figure 6

i on the game board to position *j*. This matrix is called a *Transition Matrix*.

An entry in row-i and column-j of the transition matrix gives the probability of moving from location *i* to location *j* on the next move. By stochastic definition, the probabilities of each row add up to *1.0*. A vanilla snippet of this matrix can be seen in *Figure 5*. Here, there is a 1/6 equal chance of reaching each of the next six squares.

Things get a little more interesting when we add *Snakes and Ladders* into the mix. Now there is a chance that the next roll may land a player onto the business-end of one of these special entities and he will get "teleported" to a new location.

Consider the fictitious location depicted in *Figure 6*. There are still six possible outcomes with equal probability of 1/6, but this time, rather than being consecutive, they sometimes record the locations that would be jumped to if the player lands on a snake or a ladder. The section of the transition matrix shown in *Figure 7* itemizes the probabilities. Looking at row 18 we see there is a 1/6 chance of moving to square 19, a 1/6 chance of moving back to square 9 after landing on the snake that start on square 20, a 1/6 chances of moving to squares 21, 22 and 24, and a 1/6 chance of landing on square 28 after taking the ladder from square 23.

There are just a couple of other scenarios we need to correctly address and we'll be able to construct a full stochastic transition matrix for our game.

The first is a condition in which it's possible to land in a location by more than one means from a single roll. An example of this can be seen in *Figure 8*. If a player is on square 50, then a roll of 3 will take her to square 53, but a roll of 6 will also land her on square 53 (because landing on square 56 is the head of a snake which slides her back to 53). Thus, the probability of moving from square 50 to square 53 is 2/6 and not 1/6. *Figure 9* shows this snippet of the transition matrix.

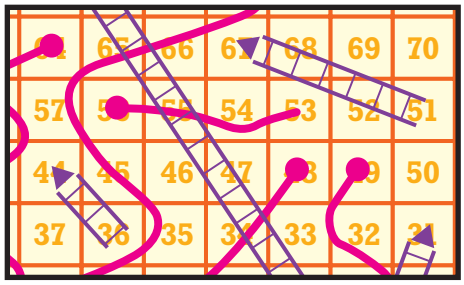


figure 8

The second condition that interests us is the boundary scenario in which the player is close to the finish. Since an exact roll is not needed, there are multiple ways to get to square 100. In the matrix snippet shown in *Figure 10* you can see the probability of moving from square 97 to square 100 is 4/6.

The Transition Matrix

After the stochastic probabilities for each square are entered, the result is a transition matrix that is (101 x 101) and is sparse in nature. It's (101 x 101) rather than (100 x 100) because, prior to their first rolls, players begin the game with their tokens off the board as if there were a square 0.

You may have already have realized that actually we don't need a (101 x 101) matrix; rather, we can instead represent the transition matrix as an (82 x 82) grid. Why? Well the simple explanation is that it is impossible for

	8	9	10	11	12	13	14	15	16	17	18	19
10	0	0	0	1/6	1/6	1/6	1/6	1/6	1/6	0	0	0
11	0	0	0	0	1/6	1/6	1/6	1/6	1/6	1/6	0	0
12	0	0	0	0	0	1/6	1/6	1/6	1/6	1/6	1/6	0

figure 5

	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
17	0	1/6	0	0	0	0	0	0	0	0	1/6	1/6	0	1/6	1/6	0	0	0	0	0	1/6	0	0
18	0	1/6	0	0	0	0	0	0	0	0	0	1/6	0	1/6	1/6	0	1/6	0	0	0	1/6	0	0
19	0	1/6	0	0	0	0	0	0	0	0	0	0	0	1/6	1/6	0	1/6	1/6	0	0	1/6	0	0

figure 7

Mathematical Analysis of *Snakes and Ladders*

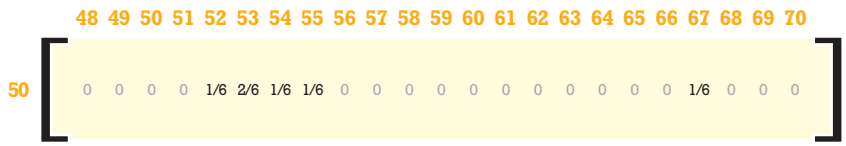


figure 9

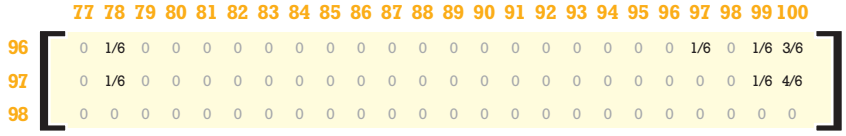


figure 10

a player to rest on the head of a snake or the bottom of a ladder. These squares don't need to be defined as separate states since landing on one of them instantly transports the player to the other end. In the (101 x 101) matrix these rows and columns are full of redundant zeros.

Interestingly, compressing these redundant rows makes a non-trivial difference to calculation speed. Matrix multiplication (which as we will see below is used for this calculation) is O(n3), so reducing the size of a square matrix from 101 to 82 doubles the speed!

The transition matrix encapsulates the probability of moving from any square to any other square. Now all we need to do is provide it with an input. A player starts the game off board, and nowhere else, so we create a column vector with 1.0 in the row0, and zeros in all the other positions. (In other words, there is a 100 percent probability that the player will start at position zero).

Next we multiply our column vector by the transition matrix, and the vector produced at the output is the probability distribution at the end of the first roll. Each row value in the output vector is the probability that the player's token will be in that square at the end of that roll. This is represented graphically in *Figure 11*. The darker squares represent the regions of higher probability, while the white squares represent regions of zero probability. There are six shaded squares, each with equal probability, representing the squares that would have been achieved with each distinct roll of the die. You can see that two of the rolls resulted in the use of ladders.

To determine the probabilities of what will happen on the next roll, we use the *output* of the first roll as the *input* for the second roll. We do this b y multiplying by the transition matrix again. The resulting output (*Figure 12*) is the superposition of the probabilities from each of the starting locations. Already the probability "cloud" is spreading out. The dark shading of some of the cells (especially on the lower row) highlight how these spaces are more likely to be occupied after two rolls because of the multiple ways to get there.

Figure 13, Figure 14, Figure 15 and Figure 16 shows three, four, five and six rolls respectively. *Figure 17* shows (very, very faintly) shading in square 100. A non-zero value is present in this square for the first time showing that the probability cloud has touched the finish, and that it is possible for the game to end in seven rolls. *Figure 18* shows the probability distribution after 20 moves.

Results

The results of the Markov chain analysis are plotted in *Figure 19*. It's very close to the curve obtained by the Monte-Carlo simulation. The fact that both curves are almost identical, despite being generated in two entirely different ways, reinforces the likelihood that we don't have logic errors in our implementation.

Figure 20 shows a close-up look at how similar the curves are. The blue line is the exact Markov chain probability, and the red line is the line generated by the random Monte-Carlo simulation. The red line is ever so slightly jagged, highlighting the fact that a random process was used in its creation.

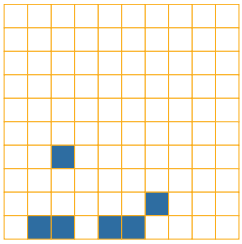


figure 11

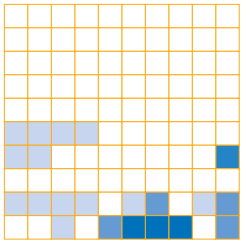


figure 12

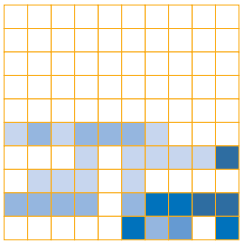


figure 13

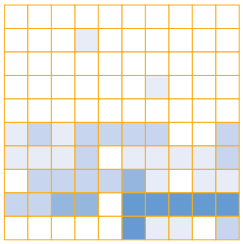


figure 14

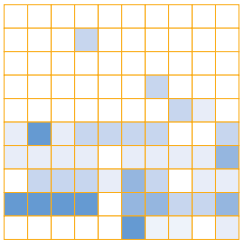


figure 15

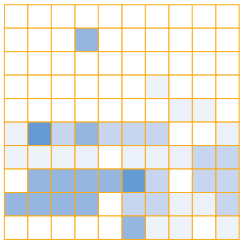


figure 16

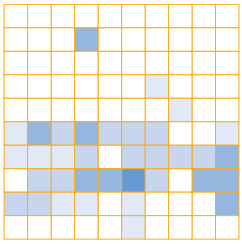


figure 17

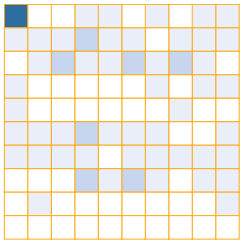


figure 18

Summary

It's possible to use quite simple math to model outcomes of seemingly complex games. If you can represent your game as some form of "memoryless" finite state machine then, through the use of Markov chain analysis, you can exactly calculate the probability distribution of future states.

If a non-exact solution is all that is needed, or you don't understand the subtleties of the probabilities, then a Monte-Carlo simulation with the appropriate number of experiments can produce perfectly acceptable approximations. ❊