

Randomness and Simulation

- Introduction
- The random Procedure
- Simulating a Die

Introduction

Many computing applications involve the simulation of games or events, with the hope of gaining insights and identifying underlying principles. In some cases, simulations can apply definite, well-known formulae. For example, in studying the effect of a pollution source in a lake or stream, one can keep track of pollutant concentrations in various places. Then, since the flow of water and the interactions of pollutants is reasonably well understood, one can follow the flow of the pollutants over a period time, according to known equations.

In other cases, specific outcomes involve some chance. For example, when a car begins a trip and encounters a traffic light, it may be a matter of chance as to whether the light is green or not. Similar uncertainties arise when considering specific organisms or when tabulating the outcomes involving flipping a coin, tossing a die, or dealing cards. In these cases, one may know about the probability of an event occurring (a head may occur about half the time), but the result of any one event depends on chance.

In studying events that involve some chance, one approach is to model the event or game, using a random number generator as the basis for decisions. If such models are run on computers many times, the results may give some statistical information about what outcomes are likely and how often each type of outcome might be expected to occur. This approach to problem solving is called the *Monte Carlo Method*.

The random Procedure

A random number generator for a typical computer language is a procedure which produces different values each time it is called. Such procedures simulate a random selection process. Scheme provides the procedure `random` for this purpose. This procedure returns integer values which depend on its parameter. In particular, `random` returns a value between 0 and one less than its parameter, inclusive.

```
> (random 10)
1
> (random 10)
9
> (random 10)
7
> (random 10)
0
> (random 10)
5
```

```

> (random 10)
1
> (random 10)
0

```

Simulating a Die

We can use Random to write a program to simulate the rolling of a die, by generating integers from 1 to 6, to correspond to the faces on the die cube. The details of this simulation are shown in the following procedure:

```

;;; Procedure:
;;;   roll-a-die
;;; Parameters:
;;;   None
;;; Purpose:
;;;   To simulate the rolling of one six-sided die.
;;; Produces:
;;;   An integer between 1 and 6, inclusive.
;;; Preconditions:
;;;   None.
;;; Postconditions:
;;;   Returns an integer between 1 and 6, inclusive.
;;;   It should be difficult (or impossible) to predict which
;;;   number is produced.
(define roll-a-die
  (lambda ()
    (let ((tmp (random 6))) ; tmp is in the range [0 .. 5]
      (+ 1 tmp)))         ; result is in the range [1 .. 6]

```

To roll a number of dice, we could construct a list where each member is created by a separate call to the roll-a-die procedure. A parameter *n* allows us to count the number of dice left to roll.

```

;;; Procedure:
;;;   roll-dice
;;; Parameters:
;;;   n, an integer (the number of dice to roll)
;;; Purpose:
;;;   Roll n dice.
;;; Produces:
;;;   A list of integers, each between 1 and 6 (inclusive).
;;; Preconditions:
;;;   n >= 1.
;;; Postconditions:
;;;   Returns a list of length n.
;;;   Each element of the list is between 1 and 6 (inclusive).
;;;   The elements of the list are difficult (or impossible) to predict.
(define roll-dice
  (lambda (n)
    ; If there are no dice left to roll, ...
    (if (<= n 0)
        ; then give an empty list of rolls
        null
        ; Otherwise, roll once and then roll n-1 more times.
        (cons (roll-a-die) (roll-dice (- n 1))))))

```