CHANGE 2

computer(s) and coins

Robert Blatt

CHANGE 2
computer(s) and coins

Remarks regarding a performance

The work may be performed by itself or with other occurrences, for oneself or any that are interested. Each performer has a computer and a set of coins, where each coin in a set is of a different currency and/or monetary value. Coins are arranged from left to right on a surface by ascending pitch, determined by the sound each coin makes when spun on its edge. The following process, unsynchronized with multiple performers, occurs as follows: All coins are first spun on their edge, from lowest to highest pitch, one after another, at a rate which does not allow a coin to come to rest before its succeeding coin has started spinning. After all coins have stopped spinning, heads or tails landing values for each coin are entered in a computer. The computer then generates instructions, either ending the performance or specifying a set of the coins to be spun. In the latter case, the specified coins are spun in the same manner as above, and the landing results are again entered in the computer, likewise generating further spin instructions. This process continues until instructed to stop, or if necessary, upon exhaustion or conditions of circumstance.

Remarks regarding the computer program

There are a finite number of possible combinations of heads or tails values for a set of coins. For example, with two coins there are four possible heads or tails values: coin 1 = heads and coin 2 = heads; coin 1 = heads and coin 2 = tails; coin 1 = tails and coin 2 = heads; coin 1 = tails and coin 2 = tails. Likewise, there are equally as many possible combinations of coins that can be spun. Again, with two coins there are four possible instructions: spin coins 1 and 2; spin coin 1; spin coin 2; spin no coins. Using a pseudo-random approximately uniform choice distribution function, the computer program assigns each possible combination of heads or tails values for a performer's set of coins to a different spin instruction, where all possible spin instructions for the set of coins are used. The program takes the current landing heads or tails values from the set of coins as an input and outputs its assigned spin instruction. In the case where all coins are instructed not to be spun, the performance is over. Note, the number of coins chosen for a performance will inherently determine the probable duration of time until an instruction to end the performance is given, where less coins will likely lead to a shorter duration and more coins will likely lead to a longer duration.

Robert Blatt, 2013 (revised/notated 2016)

```
;;;Change 2
;;;Robert Blatt, 2013/16

;;;language: Lisp
;;;dialect: Common Lisp
;;;implementation: SBCL
;;;version: 1.1.6.0-3c5581a


;;;step 1: set "number-coins" parameter value to the number of coins used

(defparameter number-coins 6)

;;;step 2: save and load this file

;;;step 3: generate coin spinning instructions by calling the function "change2" with its argument as a list of heads or tails values
;;;heads = 0 and tails = 1
;;;heads or tails values are entered in ascending coin order determined by ascending pitch when coins are spun on their side

;;;example: if 10 coins are used and the heads or tails values are
;;;coin 1 = heads | coin 2 = tails | coin 3 =  heads | coin 4 = heads | coin 5 = tails | coin 6 = heads | coin 7 = tails | coin 8 = tails | coin 9 =  heads | coin 10 = heads
;;;the function call is (change2 '(0 1 0 0 1 0 1 1 0 0))
;;;the function will return a string instructing which coins are to be spun, such as "Spin coins 2, 5, 6, 7 and 9.", or to stop

(defun combinations (lists)
  (if (car lists)
      (mapcan (lambda (inner-val)
                (mapcar (lambda (outer-val)
                          (cons outer-val inner-val))
                        (car lists)))
              (combinations (cdr lists)))
      (list nil)))

(defun nshuffle-array (array)
  (loop for i from (length array) downto 2
     do (rotatef (aref array (random i))
                 (aref array (1- i)))
     finally (return array)))
```

```
(defun nshuffle-list (list)
  (let ((array (nshuffle-array (coerce list 'vector))))
    (declare (dynamic-extent array))
    (map-into list 'identity array)))

(defparameter paired-list-of-combinations-and-spin-instructions
  (let* ((combinations-per-number-coins (combinations (loop for i from 0 to (1- number-coins) collect '(0 1))))
            (randomized-combinations-per-number-coins (copy-list combinations-per-number-coins))
            (randomized-combinations-per-number-coins (nshuffle-list randomized-combinations-per-number-coins)))
    (loop for i from 0 to (1- (length combinations-per-number-coins))
      collect (list (nth i combinations-per-number-coins) (nth i randomized-combinations-per-number-coins)))))

(defun change2 (current-landing-results)
  (let* ((spin-instructions nil)
            (output nil)
            (template "Spin coin~{~#[~; ~a~;s ~a and ~a~:;s ~@{~a~#[~;, and ~:;, ~]~}~]~}."))
    (loop for i from 0 to (1- (length paired-list-of-combinations-and-spin-instructions))
      do (cond
              ((equalp (first (nth i paired-list-of-combinations-and-spin-instructions)) current-landing-results)
               (setf spin-instructions (second (nth i paired-list-of-combinations-and-spin-instructions)))
               (setf output (loop for i from 0 to (1- number-coins)
                                  for coin = (nth i spin-instructions)
                                  when (= coin 1)
                                  collect (1+ i)))
               (if (equalp output nil)
                     (setf output "Stop.")
                     (setf output (format nil template output))))))
    (if (equalp spin-instructions nil)
          (setf output "ERROR: Check that heads or tails values are formatted correctly."))
    output))
```