```scheme
;;;;;;;;;;;;;;;;;;
;;
;; A program for playing the Prisoner's Dilemma
;; Each player's strategy is represented as a procedure
;; that takes two arguments -- the player's own history
;; and the other player's history -- and returns either
;; the symbol C (for cooperate) or D (for defect).

;; The standard PD game matrix

(define *game-matrix*
  '(
    ((c c) (3 3))
    ((c d) (0 5))
    ((d c) (5 0))
    ((d d) (1 1))))

;; The top level procedure, called on two arguments
;; corresponding to the two players' strategies

(define (play-loop p1-strategy p2-strategy)
  (play-loop-helper
   p1-strategy p2-strategy
   0 '() '() (+ 90 (random 21))))


(define (play-loop-helper p1 p2 count history1 history2 limit)
  (cond ((= count limit) (print-out-results history1 history2 limit))
        (else
         (let ((result1 (p1 history1 history2))
               (result2 (p2 history2 history1)))
           (play-loop-helper
            p1 p2 (+ 1 count)
            (cons result1 history1)
            (cons result2 history2)
            limit)))))


;; Making a "full game" data structure
;;

(define (make-game player-1-choice player-2-choice)
  (list player-1-choice player-2-choice))

(define (get-point-list-for-game game)
  (cadr (assoc game *game-matrix*)))

(define (get-player-points player-no game)
  (list-ref (get-point-list-for-game game) (- player-no 1)))

;; A history is just a list of the given player's responses:
;; Each player's choices in the games played so
;; far will be represented in an individual "game history"
;; consisting of C's and D's.

(define (extend-player-history this-play history)
  (cons this-play history))
```

```
(define (empty-history? history)
  (null? history))

(define (most-recent-play history)
  (car history))

(define (rest-of-plays history)
  (cdr history))


;;;;;;;
;;
;; The next two procedures are simply for printing out
;; the net results of games

(define (print-out-results hist1 hist2 number-of-games)
  (let ((scores (get-scores hist1 hist2)))
    (display "Player 1 Score: ")
    (display (/ (car scores) number-of-games))
    (newline)
    (display "Player 2 Score: ")
    (display (/ (cadr scores) number-of-games))
    (newline)
    'done))

(define (get-scores hist1 hist2)
  (let ((games (map list hist1 hist2)))
    (define (scoreloop gms result1 result2)
      (cond ((null? gms) (list result1 result2))
            (else (scoreloop (cdr gms)
                             (+ result1
                                (get-player-points 1 (car gms)))
                             (+ result2
                                (get-player-points 2 (car gms)))))))
    (scoreloop games 0 0)))

;;;;;;;
;;
;; Three simple game-playing strategies

(define (poor-trusting-fool my-history other-guy)
  'C)

(define (all-defect my-history other-guy)
  'D)

(define (random-strategy my-history other-guy)
  (if (= (random 2) 0) 'C 'D))



;;>>> (play-loop random-strategy random-strategy)
;;Player 1 Score: 2.4
;;Player 2 Score: 2.3523809523809525
;;done
;;>>> (play-loop poor-trusting-fool  all-defect)
;;Player 1 Score: 0
;;Player 2 Score: 5
```

```
;;done
;;>>> (play-loop poor-trusting-fool random-strategy)
;;Player 1 Score: 1.5326086956521738
;;Player 2 Score: 3.9782608695652173
;;done
```