code-s7.rkt

```
#lang racket
(require games/cards racket/gui racket/class racket/unit)

;; Layout width and height:
(define WIDTH 5)
(define HEIGHT 4)
(define MAX-MATCHES (/ (* WIDTH HEIGHT) 2))

;; Randomize
(random-seed (modulo (current-milliseconds) 10000))

;; Set up the table
(define t (make-table "Memory" (+ 2 WIDTH) (+ 1 HEIGHT)))
(send t show #t)
(send t set-double-click-action #f)

;; Get table width & height
(define w (send t table-width))
(define h (send t table-height))

;; Set up the cards
(define deck
  (let ([cards (map (lambda (name value)
                      (let ([bm (make-object
                                 bitmap%
                                 (build-path
                                  (collection-path "games" "memory" "images")
                                  (format "~a.png" name)))])
                        (make-card bm #f 0 value)))
                    '("club" "heart" "spade" "diamond"
                      "happy" "unhappy" "fish" "two-fish" "jack" "star")
                    '(1 2 3 4 5 6 7 8 9 10))])
    (append cards (map (lambda (c) (send c copy)) cards))))
(for-each (lambda (card)
            (send card user-can-move #f)
            (send card user-can-flip #t))
          deck)

;; Card width & height
(define cw (send (car deck) card-width))
(define ch (send (car deck) card-height))

(define dx (/ cw (+ 2 WIDTH)))
(define dy (/ ch (+ 1 HEIGHT)))

(define match-x (- w cw dx))
(define match-y dy)

;; Put the cards on the table
(send t add-cards deck match-x match-y)

;; Setup
(define (setup)
  (set! deck (shuffle-list deck 7))
  (send t stack-cards deck)
  (send t move-cards deck 0 0
        (lambda (pos)
          (let ([i (modulo pos WIDTH)]
                [j (quotient pos WIDTH)])
            (values (+ dx (* i (+ cw dx)))
                    (+ dy (* j (+ ch dy))))))))
```

```
62
63  ;; Number of matches found so far:
64  (define matches 0)
65
66  ;; First card flipped, or #f if non flipped, yet
67  (define card-1 #f)
68
69  (define (flip-and-match c)
70    (cond [(eq? c card-1)
71           ;; Cancel first card
72           (send t flip-card c)
73           (set! card-1 #f)]
74          [(not (send c face-down?))
75           ;; Can't click a matched card, unless the game is over,
76           ;; in which case we reset the game
77           (when (= matches MAX-MATCHES)
78             (send t flip-cards deck)
79             (set! matches 0)
80             (setup))]
81          [else
82           ;; Flip over a card...
83           (send t flip-card c)
84           (send t card-to-front c)
85           (cond [(not card-1)
86                  ;; That was the first card
87                  (set! card-1 c)]
88                 [(and (equal? (send card-1 get-value) (send c get-value))
89                       (equal? (send card-1 get-suit) (send c get-suit)))
90                  ;; Match
91                  (send t move-cards (list card-1 c) match-x match-y)
92                  (set! card-1 #f)
93                  (set! matches (add1 matches))]
94                 [else
95                  ;; Not a match
96                  (send t pause 0.5)
97                  (send t flip-cards (list card-1 c))
98                  (set! card-1 #f)])]))
99
100 (send t set-single-click-action flip-and-match)
101
102 ;; Start the game:
103 (setup)
```