v.6.11

Resolución práctica 3

A continuación presentamos soluciones a ejercicios seleccionados de la Práctica 3.

Como siempre, en el proceso de resolución de un problema, primero debemos dedicar un tiempo al análisis del mismo para asegurarnos de entender correctamente qué se pide. Luego, pasamos a la resolución propiamente dicha.

Recomendamos usar este documento como un elemento de ayuda y referencia. Es importante que no lean la solución sin antes haber leído y entendido el problema y sin haber dedicado un tiempo a pensar cómo resolverlo.

Por otro lado, no hay una única forma correcta de escribir una solución.

1 Programas interactivos

En esta práctica comenzaremos a trabajar con programas interactivos. Para hacerlo, deberemos familiarizarnos con conceptos como *evento*, *estado*, y las expresiones `big-bang`. A su vez, para poder llevar adelante estos programas más complejos, es necesario que tengamos en cuenta todo lo aprendido en los ejercicios anteriores en cuanto a buenas prácticas de programación.

1.1 Ejercicio 4

En este ejercicio se nos pide que movamos un objeto verticalmente sobre una escena, a través de las flechas del teclado, del click del mouse, o de la barra espaciadora. El enunciado ya nos dice que el estado del sistema será un número, que representa la posición vertical sobre la que se va a dibujar un círculo. También nos indica cuál será el estado inicial.

Empezaremos definiendo constantes para representar el ancho y alto de la escena, el radio y el color del círculo, y el estado inicial del programa.

```
; alto y ancho de la escena
(define ALTO 500)
(define ANCHO 500)
; radio y color del círculo
(define RADIO 20)

; Estado del sistema: Un número que representa la posición
; vertical del objeto en la escena.

; Estado inicial
(define INICIAL (/ ALTO 2))
```

Luego, escribiremos la función asociada a la cláusula `to-draw`. Deberá dibujar un círculo centrado horizontalmente, y cuya posición sobre el eje vertical estará determinada por el estado actual. También será útil definir nuevas constantes.

```
; escena vacía:
(define FONDO (empty-scene ANCHO ALTO))
; objeto a dibujar
(define OBJETO (circle RADIO "solid" COLOR))

; interpretar : Estado -> Imagen
; dado un estado y, dibuja el objeto centrado horizontalmente en la posición
; vertical indicada por y
```

```
(define (interpretar y)
  (place-image OBJETO (/ ANCHO 2) y FONDO))

; expresión big-bang
(big-bang INICIAL
          [to-draw interpretar])
```

El enunciado nos pide que el círculo se mueva verticalmente a través de ambas flechas del teclado, una longitud DELTA. A su vez, debe volver al estado inicial si se presiona la tecla espaciadora. Por lo tanto, escribiremos una primera versión de la función asociada al manejador del teclado. Para realizarlo, será útil definir nuevas constantes.

```
; constante que determina cuántos píxeles se mueve el objeto en la escena
(define DELTA 5)

; tecladoSimple : Estado String -> Estado
; dado el estado actual y un código de tecla, calcula el estado siguiente:
; - tecla "up" : decrementa el estado DELTA unidades
; - tecla "down": incrementa el estado DELTA unidades
; - tecla " " : estado inicial
(check-expect (tecladoSimple INICIAL "up") (- INICIAL DELTA))
(check-expect (tecladoSimple INICIAL "down") (+ INICIAL DELTA))
(check-expect (tecladoSimple INICIAL "q") INICIAL)
(define (tecladoSimple y k)
  (cond [(string=? k "up") (- y DELTA)]
        [(string=? k "down") (+ y DELTA)]
        [(string=? k " ") INICIAL]
        [else y]))


(big-bang INICIAL
          [to-draw interpretar]
          [on-key tecladoSimple])
```

A esta altura del ejercicio, el círculo debería poder moverse verticalmente, incluso desapareciendo de la escena. Para evitar esto, podemos definir una función `acomodar` que, dado un estado, se encargue de que este nunca esté fuera de los límites de nuestra escena. Es decir, si la ubicación del círculo excediera hacia arriba o hacia abajo los límites de la escena, lo ubicaremos en la posición límite correspondiente, en donde el círculo es completamente visible.

```
; acomodar : Estado -> Estado
; dado un estado, asegura que este se encontrará en el intervalo [RADIO, ALTO - RADIO]
; para posiciones menores a RADIO (el objeto sobrepasa el límite superior), devuelve RADIO
; para posiciones mayores a ALTO-RADIO (el objeto sobrepasa el límite inferior), devuelve ALTO-RADIO
(check-expect (acomodar 0) RADIO)
(check-expect (acomodar INICIAL) INICIAL)
(check-expect (acomodar ALTO) (- ALTO RADIO))
(define (acomodar y)
  (cond [(< y RADIO) RADIO]
        [(> y (- ALTO RADIO)) (- ALTO RADIO)]
        [else y]))
```

Una vez hecha la función anterior, definir un nuevo manejador del teclado que asegura que el objeto no se va de la escena, es sencillo.

```
; teclado : Estado String -> Estado
; dado el estado actual y un código de tecla, calcula el estado próximo,
; cuidando que el objeto no salga de la pantalla
(check-expect (teclado INICIAL "up") (- INICIAL DELTA))
(check-expect (teclado 0 "up") RADIO)
(define (teclado y k) (acomodar (tecladoSimple y k)))
```

Para terminar el ejercicio, agregaremos la función correspondiente al manejador del mouse, que está dada en el enunciado.

```
; mouse : Estado Number Number String -> Estado
; dados el estado actual y un evento del mouse, devuelve
; el nuevo estado de acuerdo al tipo de evento:
; - "button-down": posición vertical del mouse
; cualquier otro evento: no modifica el estado
(check-expect (mouse INICIAL 100 RADIO "button-down") RADIO)
(check-expect (mouse INICIAL 200 ALTO "button-down") ALTO)
(check-expect (mouse INICIAL 300 100 "move") INICIAL)
(define (mouse y xm ym event)
  (cond [(string=? event "button-down") ym]
        [else y]))

; expresión big-bang
(big-bang INICIAL
          [to-draw pantalla]
          [on-key teclado]
          [on-mouse mouse])
```

> ¿Qué pasa si hace click con el mouse en una posición lo suficientemente cerca de los límites? ¿El círculo se ve completo? ¿Por qué? Piense cómo resolverlo de una manera similar a lo que se hizo con las funciones `tecladoSimple`, `acomodar` y `teclado`.

1.2 Ejercicio 6

En este ejercicio se nos pide crear un mini-editor de texto. Como bien aclara el ejercicio, pide familiarizarse con la función `text`, y el uso de `place/image-align`, que daremos por entendido para esta parte. Si no hicieron esas pruebas, les recomendamos que las hagan antes de hacer el ejercicio, para que sea menos confuso.

Como siempre, comenzaremos con un **diseño de datos**, pensando qué debemos representar y cómo. En este caso, al ser una función interactiva, también su estado inicial.

En nuestro caso particular, la idea es representar texto, por lo tanto, nuestro tipo de dato será un string y nuestro estado inicial será una cadena vacía. Además definiremos algunas constantes necesarias.

```
; Estado del sistema: Una string vacía que representa que todavía
; no hay nada escrito.
(define ALTO 800)
(define ANCHO 80)

; Estado inicial
(define INICIAL "")
```

Luego, crearemos la función que funcionará en la clausula `to-draw`.

El objetivo de esta función será representar la pantalla donde el usuario escribirá las palabras en nuestro editor. Oara realizarla utilizaremos la función dada por la práctica anteriormente. Lo único que cambiaremos es que en vez de escribir "Arriba a la izquierda" escribirá lo que haya en nuestro estado.

```
; Estado del sistema: Una string vacía que representa que todavía no hay nada escrito.
(define ALTO 800)
(define ANCHO 80)

; Estado inicial
(define INICIAL "")
```

```
; escena vacía:
(define FONDO (empty-scene ANCHO ALTO))

; pantalla : Estado -> Imagen
```

```
; dado un estado s, dibuja un texto que representa la cadena de caracteres
; dada por s.
(define (pantalla s) (place-image/align (text s 20 "indigo") 0 0 "left" "top" FONDO))
```

Con esto tenemos un fondo de pantalla que escribe en un texto lo que existe en la variable `s` de tipo string. Nuestro próximo paso será agregarle caracteres.

Para realizar esto, vamos a definir la función que será utilizada en la clausula `on-key`. Ésta consistirá en tomar la tecla como string y agregarla al final de la string que tenemos como estado.

Esto se puede expresar con la función `string-append`, que hace justamente eso.

```
; agregar-letra : Estado String -> Estado
; Dado una tecla, concatena esta a la cadena que ya habíamos escrito hasta ahora.

(check-expect (agregar2 "" "a") "a")
(check-expect (agregar2 "hola mund" "o") "hola mundo")
(check-expect (agregar2 "racke" "t") "racket")
(define (agregar-letra s k) (string-append s k))

; Expresión Big-Bang
(big-bang INICIAL
          [to-draw pantalla]
          [on-key agregar-letra])
```

Finalmente, nos queda agregar la posibilidad de borrar caracteres. El borrado de caracteres debe suceder cuando apretamos la tecla `backspace`. Es decir, se ejecutará en la clausula `on-key`, y por lo tanto, deberá ir a la función `agregar-letra`.

Inicialmente, la idea de borrar caracteres consiste simplemente en retirar el último caracter de la cadena. Sin embargo, tenemos un **caso borde**, ¿qué pasa si queremos eliminar un caracter pero no hay ningún caracter? Debemos considerar la posibilidad de que la cadena sea vacía y en ese caso, el borrado no deberá hacer nada. En caso contrario, esto daría un error.

Para estar totalmente convencides de que no hay problema, ese deberá ser sí o sí uno de nuestros casos de testeo.

A continuación, armaremos una nueva función `agregar-letra-con-borrado`.

```
; agregar-letra-con-borrado : Estado String -> Estado
; Dado una tecla, concatena esta a la cadena que ya habíamos escrito hasta ahora.

(check-expect (agregar-letra-con-borrado "" "a") "a")
(check-expect (agregar-letra-con-borrado "hola mund" "o") "hola mundo")
(check-expect (agregar-letra-con-borrado "racke" "t") "racket")
(check-expect (agregar-letra-con-borrado "" "\b") "")
(check-expect (agregar-letra-con-borrado "hola mundo" "\b") "hola mund")

(define (agregar-letra-con-borrado s k)
                (cond [(key=? "\b" k) (if (> (string-length s) 0)
                                          (substring s 0 (- (string-length s) 1))
                                           s)]
                      [else (agregar-letra s k)]))

; Expresión Big-Bang
(big-bang INICIAL
          [to-draw pantalla]
          [on-key agregar-letra-con-borrado])
```

1.3 Ejercicio 7

Comenzamos definiendo algunas constantes que son parámetros del programa:

```
```

```
(define AUTO  )

; Estado: número
; Intepretación: distancia en píxeles desde el borde izquierdo de la pantalla
; hasta el centro del auto (coordenada horizontal del auto)

; Alto de la escena.
(define ALTO 100)

; Ancho de la escena.
(define ANCHO 800)

; Desplazamiento del auto, en píxeles, cada tick del reloj.
(define DELTA 3)

; Fondo sobre el que dibujaremos el auto.
(define MARCO (empty-scene ANCHO ALTO))
```

Definimos el manejador correspondiente a los dibujos en pantalla, que simplemente consiste en posicionar el auto sobre el fondo en la posición horizontal indicada por el estado:

```
; pantalla: Estado -> Image
; Dibuja el auto en la posición horizontal determinada por el estado y
; centrado verticalmente
(define (pantalla x) (place-image AUTO x (/ ALTO 2) MARCO))
```

Tenemos que tener en cuenta que el manejador correspondiente a los dibujos en pantalla no comprueba que la posición del auto sea válida (es decir, que quepa dentro del marco). Para evitar esto, definimos una función que corrija el estado:

```
; acomodar : Estado -> Estado
; Devuleve un estado que se encuentra en el intervalo que garantiza
; que la imagen se dibujará completa en la escena
(define (acomodar x) (cond [(< x (/ (image-width AUTO) 2))
                            ( + 1 (/ (image-width AUTO) 2))]
                           [(> x (- ANCHO (/ (image-width AUTO) 2)))
                                    (- ANCHO (/ (image-width AUTO) 2) 1)]
                           [else x]))
```

A continuación, podemos definir el manejador de eventos de teclado. Este produce cambios en el estado según la tecla presionada y, posteriormente, acomoda cada nuevo valor del estado:

> ¿Podríamos haber definido esta función de otra forma utilizando `acomodar`? ¿Y sin utilizarla?

```
; teclado: Estado String -> Estado
; Modifica el estado de acuerdo a la tecla presionada:
; * Espacio: vuelve al inicio
; * Flecha derecha: avanza 20 píxeles
; * Flecha izquierda: retrocede 20 píxeles
; * Cualquier otra tecla: no se modifica el estado
(define (teclado x k)
  (cond [(string=? k " ") (acomodar (/ (image-width AUTO) 2))]
        [(string=? k "left") (acomodar (- x 20))]
        [(string=? k "right") (acomodar (+ x 20))]
        [else (acomodar x)]))
```

Procedemos de forma similar con el manejador de eventos del mouse:

```
; mouse: Estado Number Number String -> Estado
; Responde al mouse, asignando al estado la coordenada horizontal donde
; se produce un evento "click". Para cualquier otro evento del mouse, el
```

```
; estado no cambia.
(define (mouse x xm ym em) (cond [(string=? em "button-down") (acomodar xm)]
                                 [else (acomodar x)]))
```

El último manejador que resta implementar es el del paso del tiempo:

```
; desplazar: Estado -> Estado
; Incrementa el estado en DELTA unidades
(define (desplazar x) (acomodar (+ x DELTA)))
```

Finalmente, conectamos todos los manejadores a `big-bang`:

```
(big-bang  (/ (image-width AUTO) 2)
           [to-draw pantalla]
           [on-tick desplazar]
           [on-key teclado]
           [on-mouse mouse])
```