v.7.6

Práctica 3

Programación I

1 Programas Interactivos

Desde el punto de vista del código, un programa es un conjunto de definiciones de constantes, funciones y expresiones que calculan valores. Sin embargo, desde el punto de vista de la ejecución del programa; hay dos grandes categorías que pueden ser identificadas. Los programas llamados **programas por lotes**, que una vez lanzado el proceso no necesitan ningún tipo de interacción con el usuario; y los **programas interactivos**, que una vez iniciados esperan la intervención del usuario (o de eventos producidos por las computadoras) para llevar a cabo sus funciones.

> Un *compresor de archivos* es un ejemplo de **programa por lotes**. El programa que maneja un *cajero automático* es un ejemplo de **programa interactivo**.

Las funciones que hemos visto y construido hasta el momento son programas por lotes; por ejemplo, el programa que calcula el monto total de comprar cuadernos y lápices en la librería *El lápiz curioso* (Práctica 1, 1ra parte). En este caso, como en los otros ejercicios y ejemplos, una vez dados los datos de entrada e iniciado el programa simplemente esperamos a que éste termine para obtener el resultado.

Ahora veremos programas interactivos, donde se espera que durante la ejecución el usuario (o alguna computadora) intervenga proporcionándole información según se presenten eventos que lo soliciten.

1.1 Eventos y manejadores de eventos

Los programas reciben información de su entorno a través de **eventos**. La palabra evento tiene un significado un poco diferente en nuestro idioma, aunque la utilizaremos en programación se usa para describir la ocurrencia de alguna situación contemplada en la programación que requiere información para tomar una determinada acción.

Un evento ocurre cuando se presiona una tecla del teclado, cuando se mueve el mouse o se presiona uno de sus botones, cuando otro programa envía un mensaje, cuando el reloj interno de la computadora marca el paso del tiempo (ticks), etc.

La idea fundamental de un programa interactivo es que ante un determinado *evento* el programa llevará a cabo cierta acción; dicha acción queda establecida a partir de la definición de una función llamada *manejador de eventos*. En el caso de un evento del teclado, el manejador de eventos será una función que, dado un argumento (la tecla presionada), devuelve la acción a efectuarse. Supongamos que queremos escribir un programa que dibuja un círculo de un determinado color. Éste será azul cuando se presione la tecla "a", rojo cuando se presione la tecla "r" y verde cuando se presione la tecla "v". Presionar una tecla produce un evento, y el programa deberá responder al mismo ejecutando cierta acción. En este caso, la acción será cambiar de color el círculo dibujado.

1.2 Estado

Durante la ejecución de un programa, ante ciertos *eventos*, los correspondientes *manejadores de eventos* llevan a cabo acciones que **cambian determinadas propiedades o valores** dentro del

programa. Dichas propiedades o valores serán lo que llamaremos el **estado** del programa.

Durante la ejecución de un programa interactivo, el programa estará en un estado particular, el cual cambiará (por medio del manejador de eventos) ante la aparición de un evento.

> **¿Qué cambia?** Esto es lo que deberemos preguntarnos cuando necesitemos determinar cómo será el **ESTADO** de un programa.

En el ejemplo del círculo, aquello que cambia es el color del mismo. El estado del programa deberá ser entonces un color. Aplicando nuestra receta de diseño, es momento de representar esta información. Decidimos hacerlo mediante un string, que podrá tomar los siguientes valores: `"blue"`, `"red"` y `"green"`:

```
; Estado es un string, que representa el color del círculo a dibujar en pantalla.
; Posibles valores:
; - "blue"
; - "red"
; - "green"
```

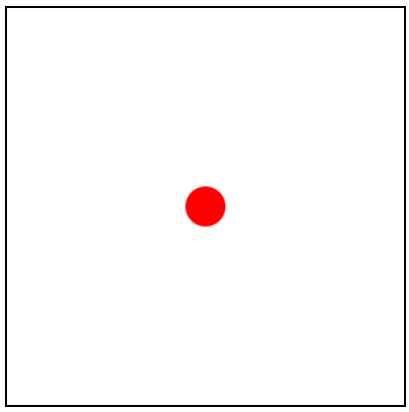
Observemos que también debemos ser capaces de interpretar nuestros estados para mostrar el resultado de la interacción al usuario. En los casos que veremos en este curso, siempre interpretaremos el estado como una imagen a mostrar (algo muy habitual en los programas interactivos), por lo que también necesitamos definir una función que "muestre" el estado actual al usuario. En nuestro ejemplo del circulo, la siguiente función puede servir para esta tarea:

> Queda como ejercicio diseñar correctamente esta función. Por ejemplo, definir las constantes adecuadas.

```
; interpretar: Estado -> Image
; dado un estado, devuelve la imagen a mostrar por el programa
; (omitimos los ejemplos para faciltar la legibilidad)
(define
   (interpretar s)
   (place-image (circle 10 "solid" s) 100 100 (empty-scene 200 200)))
```

Por ejemplo:

```
> (interpretar "red")
```



1.3 Expresiones `big-bang`

> Para usar esta biblioteca, añada el paquete de enseñanza `2htdp/universe` en el menú `Lenguaje`

El mecanismo para establecer la relación *evento - manejador de evento* e indicarle a *DrRacket* cómo tratar un evento, es a través de la función `big-bang` (provisto por la biblioteca

`2htdp/universe`).

Para usar esta función, debemos primero haber definido cómo se representa e interpreta el estado del programa, y elegir un valor inicial para el mismo. Asimismo, debemos definir los distintos manejadores de eventos y asociarlos con los correspondiente tipos de eventos. La estructura de una expresión `big-bang` es la siguiente.

```
(big-bang <estado inicial>
          [to-draw <controlador de pantalla>]
          [on-key <manejador de teclado>]
          [on-mouse <manejador de mouse>]
          [on-tick <manejador de reloj>]
          [stop-when <predicado de fin de ejecución>]
          etc...
          )
```

Tenemos entonces que `big-bang` recibe dos argumentos: el estado inicial y una lista de parejas. De estas parejas, sólo una es obligatoria (`to-draw ...`). Esta primera pareja nos indica cómo interpretar el estado actual (en nuestro caso, como una imagen, y la función `interpretar` de más arriba nos sirve).

> Para una primera lectura, sugerimos leer la siguiente lista ignorando el paso 4. Ya habrá tiempo de entender este paso más adelante.

Cuando se evalúa, el comportamiento de esta expresión es como sigue:

1. La función asociada a `to-draw` es invocada con el estado inicial como argumento, y su resultado se muestra por pantalla.
2. El programa queda a la espera de un evento.
3. Cuando un evento ocurre, el manejador asociado a dicho evento (si existe) es invocado, y devuelve el nuevo estado.
4. En caso de estar presente, se aplica el predicado (la condición) asociado a la clásula `stop-when` al nuevo estado. Si devuelve `#true`, el programa termina, si no,
5. la función asociada a `to-draw` es nuevamente invocada con el nuevo estado, devolviendo la nueva imagen o escena.
6. El programa queda a la espera de un nuevo evento (volvemos al paso 2).

En nuestro ejemplo, la expresión `big-bang` que debemos evaluar será como sigue.

```
(big-bang "blue"                    ; estado inicial del sistema

          [to-draw interpretar]     ; dibuja en la pantalla
                                    ; el círculo en el estado actual

          [on-key manejarTeclado]   ; cuando se presiona una tecla,
                                    ; la función manejarTeclado
                                    ; se invoca para manejar el evento)
```

Donde:

- el estado inicial es `"blue"`;
- `interpretar` recibe el estado y dibuja una escena con el círculo en ese estado (inicialmente azul). Ver arriba para su definición.

- `manejarTeclado` recibe el estado actual y la tecla presionada, y devuelve el nuevo estado.

```
; manejarTeclado: Estado String -> Estado
(define (manejarTeclado s k) (cond [(key=? k "a") "blue"]
                                   [(key=? k "r") "red"]
                                   [(key=? k "v") "green"]
                                   [else s]))
```

  Siempre que asociemos una función al evento `on-key`, esta tendrá dos argumentos: el estado actual y un string que representa la tecla que generó el evento. Y siempre devolverá un nuevo estado.

Ejercicio 1. Escriba el programa que cambia el color del círculo utilizando las definiciones de funciones y la expresión `big-bang` presentadas más arriba. Para esto, agregue los paquetes `2htdp/image` y `2htdp/universe`, y escriba las funciones y expresiones en el orden adecuado (recuerde que *DrRacket* sólo aceptará utilizar funciones que fueron definidas previamente).

1. Ejecute el programa y vea cómo funciona.
2. Modifique el programa de modo tal que
   - cuando se presiona la tecla espacio, el círculo vuelve al estado inicial;
   - cuando se presione la tecla "n", el círculo tome el color negro.

1.4 Sobre los otros manejadores de eventos

Antes de pasar a los ejercicios, nos puede quedar la pregunta de cómo definir los manejadores para otro tipo de eventos. Por ejemplo, para el mouse, para el reloj del sistema, etc. Lo primero que debemos recordar, es que **todos los manejadores de eventos** son funciones que **toman como argumento el estado actual del sistema y una descripción del evento ocurrido**, y **devuelven siempre el nuevo estado**. Es decir, cualquier manejador de eventos tiene la siguiente forma:

```
; manejarEvento: Estado ... -> Estado
```

En los puntos suspensivos en ocasiones se incluye una descripción del evento ocurrido, que es diferente de acuerdo al dispositivo de entrada que estemos utilizando. Por ejemplo, vimos que para el teclado es un string que representa la tecla presionada. Para el caso del reloj, no hay ninguna descripción, sólo sabemos que evento ocurrió (es decir, el reloj indicó que el tiempo pasó) y por lo tanto su tipo es el siguiente:

```
; manejarReloj: Estado -> Estado
```

En el caso del mouse, tenemos un ejemplo en las secciones siguientes. Pero podemos pensar que necesitamos saber qué tipo de evento ocurrió (se movió el mouse, se hizo click con algún botón, estamos moviendo con un botón presionado, etc), así como también las coordenadas donde este evento ocurre (dónde está el puntero del mouse). Por lo tanto es razonable pensar que esos puntos suspensivos se completan con dos números y algo que nos informe qué ocurre (`racket` elige un string para esto último):.

```
; manejarMouse: Estado Number Number String -> Estado
```

1.5 Sobre la condición de terminación

Si observamos la descripción del comportamiento de un programa interactivo, podemos ver que estos programas parecen estar diseñados para no terminar nunca. Siempre están a la espera de un nuevo evento, actúan en consecuencia cuando este ocurre, y luego vuelven a esperar el

siguiente evento. Esto es muy común en programas de este tipo, y en general suelen ser finalizados desde fuera de la aplicación (por ejemplo, a través de una "X" en la ventana).

Sin embargo, también es posible que el propio programa decida dejar de manejar eventos y termine su ejecución. Para este fin, tenemos la cláusula `stop-when` en las expresiones `big-bang`. Esta cláusula recibe un predicado sobre estados. Es decir, una función cuyo tipo es: `Estado -> Bool`.

Supongamos que tenemos una expresión donde la cláusula `stop-when` está presente:

```
(big-bang ...
          [stop-when terminar?]
          ...)
```

En tal caso, podemos volver a mirar la descripción de las secciones anteriores, ahora sin ignorar el paso 4. Observamos entonces que, luego de que un evento ocurra y el estado actual sea actualizado a través del manejador de eventos correspondiente al nuevo estado `b`, la expresión `(terminar? b)` es evaluada. Si el resultado de evaluar esta expresión resulta ser `#true`, entonces el programa termina sin intepretar el nuevo estado `b` a través del controlador de pantalla. Es decir, no se ejecuta la función asociada a la cláusula `to-draw`. En caso que el resultado sea `#false`, entonces el estado `b` es intepretado en la pantalla y se pasa a esperar el siguiente evento.

---

2 Dibujando círculos

Ejercicio 2. Escriba un programa que dibuje un círculo color azul de radio 100 el cual vaya disminuyendo su tamaño con el paso del tiempo (es decir, con cada tick del reloj).

Observe que en este caso lo que cambia es el tamaño del círculo; esto será entonces el estado del programa.

Observe también, que dicho estado cambiará con el paso del tiempo; es decir, con los ticks del reloj, con lo cual se debe definir un manejador de eventos para la sentencia `on-tick` dentro de `big-bang`.

Para escribir el programa, complete las siguientes expresiones. Recuerde que el programa deberá ejecutarse habiendo elegido como lenguaje **Estudiante principiante** y habiendo incluido los paquetes `2htdp/image` y `2htdp/universe`.

1. Escriba y complete el siguiente controlador de pantalla.

   ```
   ; pantalla : Number -> Image
   ; transforma el estado del sistema en una imagen a mostrar a través
   ; de la cláusula to-draw
   (define
      (pantalla n)
      (place-image (circle  ... COMPLETAR ...) 150 150 (empty-scene 300 300)))
   ```

2. Con cada tick del reloj el círculo debe decrementar su tamaño; esto es, modificar su estado.

   Escriba y complete el siguiente manejador de reloj de modo tal que devuelva el predecesor de un número positivo y si el número es 0 devuelva 100.

   ```
   ; decrementar : Number -> Number
   ; Devuelve el predecesor de un número positivo.
   ; Si el número es 0, devuelve 100.
     (define (decrementar n) ...COMPLETAR...)
   ```

3. Escriba la siguiente sentencia `big-bang` que vinculará los eventos con sus correspondientes manejadores y complete el estado inicial.

   ```
   (big-bang ...COMPLETAR...                ; estado inicial del sistema
   ```

```
          [to-draw pantalla]
          [on-tick decrementar])
```

4. Ejecute el programa y vea qué sucede.

5. Escriba la siguiente función `incrementar` y utilícela como el manejardor de eventos de reloj.

```
; incrementar : Number -> Number
; suma uno a su argumento
 (define (incrementar n) (+ n 1))
```

   Ejecute el programa y vea qué sucede.

6. Invente tres preguntas del estilo "*¿Qué pasa si ...?*" e implemente las respuestas en *DrRacket* .

   Por ejemplo: "*¿Qué pasa si hago que `decrementar` reste uno siempre, incluso si su argumento es 0?*"

Ejercicio 3. Asumiendo que ya realizó las actividades propuestas en el **Ejercicio 2**, se pide que realice las siguientes modificaciones:

> Realice cada modificación de manera secuencial. Es decir, sobre el resultado de la primera, realice la segunda, y así sucesivamente.

- Modifique la función `incrementar` para que incremente en 1 el estado actual sólo si el círculo resultante cabe completamente en la pantalla. En caso contrario, que devuelva 0 como nuevo estado.
- Modifique ahora la función `pantalla` de forma tal que cambie el color del círculo de acuerdo a la siguiente especificación:
  - Si n está entre 0 y 50, que sea de color amarillo,
  - Si n está entre 51 y 100, que sea de color rojo,
  - Si n es mayor a 100, que sea verde.
- Agregue un manejador de eventos para controlar el teclado, de acuerdo a las siguientes reglas:
  - Si la tecla presionada es un dígito `d`, entonces el nuevo estado debe ser `10*d`,
  - Para cualquier otra tecla, deja el estado sin cambiar.
- Agregue una condición de terminación para que el programa finalice si el radio del círculo es mayor a 110 o menor a 10.

Ejercicio 4. En este ejercicio moveremos un objeto verticalmente sobre una escena. El estado del sistema será un número, que representa la posición vertical sobre la que se va a dibujar un círculo. Defina constantes para representar el ancho y alto de la escena, así como también el radio inicial del círculo. Genere una expresión `big-bang` tal que:

- El estado inicial del sistema es `ALTO/2` (el centro de la imagen).
- La función que responde a la cláusula `to-draw` dibuja un círculo centrado horizontalmente, y cuya posición sobre el eje vertical está determinada por el estado actual.
- Al presionar la "flecha para arriba", el objeto debe subir `DELTA` unidades, donde `DELTA` es una constante que también debe definir. Análogamente, el objeto debe "bajar" `DELTA` unidades si se presiona la tecla "flecha para abajo".
- En el item anterior puede suceder que el objeto desaparezca de la escena. Si no lo tuvo en cuenta, modifique sus funciones ahora para que esto no ocurra. Es decir, que el objeto se mueva sólo si no va a desaparecer de los bordes de la escena. Caso contrario, el estado no debe modificarse.

- Si se presiona la barra espaciadora, el sistema debe volver al estado inicial.
- Agregue la siguiente pareja a su expresión big bang:

  ```
  [on-mouse mouse-handler]
  ```

  Donde la función `mouse-handler se define como sigue:`

  ```
  (define (mouse-handler n x y event) (cond [(string=? event "button-down") y]
                                            [else n]))
  ```

  Recuerde que, para manejar eventos del teclado, la función asociada recibía el estado actual y la tecla presionada, y devolvía el nuevo estado.

  Para el caso del mouse, también devolvemos el nuevo estado, pero recuerde que el manejador recibe los siguientes argumentos:

  - El estado actual `n`
  - La coordenada `x` donde se produjo el evento
  - La coordenada `y` donde se produjo el evento
  - Qué clase de evento se produjo. En nuestro caso respondemos al mouse cuando se aprieta un botón. Hay otros eventos que iremos aprendiendo en nuevos ejercicios.

  Observe qué sucede al ejecutar el programa y hacer click con el mouse en diferentes puntos de la escena.

3 Secuencia de colores

Ejercicio 5. Este programa comienza con el fondo de color blanco y un círculo en el centro de la escena. A medida que pasen los ticks del reloj irá cambiando el color del círculo central. Pasará por los siguientes colores: amarillo, rojo, verde y azul, en el orden presentado.

El estado del sistema guardará el color del círculo central de la escena.

Defina constantes para representar el ancho y el alto de la escena, así como el color de fondo. Defina cualquier otra constante que considere necesaria (por ejemplo, el tamaño del círculo a dibujar).

La expresión big-bang deberá comportarse de manera tal que:

1. El estado inicial del sistema sea: color amarillo.
2. La función que responde a la cláusula to-draw debe dibujar el círculo del color indicado en el estado en el centro de la escena.
3. Cada tick del reloj modifica el color guardado en el estado. Dicho estado debe respetar la siguiente secuencia: amarillo, rojo, verde, azul y vuelve a comenzar la secuencia pasando al amarillo.

4 Un mini-editor de texto, primera parte.

Ejercicio 6.

Para este ejercicio, es necesario que se familiarice con la función `text`. Consulte la documentación y escriba algunos ejemplos. Además, en lugar de utilizar la función `place-image`, utilizaremos `place-image/align`, que funciona un poco diferente. Revise la documentación de esta función y pruebe la siguiente expresión:

```
(place-image/align (text "Arriba a la izquierda" 20 "indigo")
                   0 0 "left" "top" (empty-scene 800 60))
```

Con entender esta forma particular de usar la función alcanza.

Se pide crear una expresión `big-bang` cuyo estado sea una cadena de caracteres, y que funcione de la siguiente forma:

- Inicialmente, esta cadena es `""`
- La función asociada a la cláusuda `to-draw` debe imprimir la cadena en pantalla como una imagen, con fuente de 20 píxeles de alto, alineada a la izquierda y de color índigo, en una escena vacía de 800 por 60. **Sugerencia:** mire arriba.
- Cada vez que se presiona una letra o número, este caracter se agrega al estado actual.
- Por último, agregaremos a nuestro mini-editor la posibilidad de borrar caracteres con la tecla `backspace`. En caso de presionarla, el manejador del evento debe remover del estado el último caracter agregado. Es decir, el efecto obtenido es el mismo que en cualquier editor de texto. **Ayuda:** la tecla `backspace` se codifica mediante la cadena `"\b"`.

> Observación: Si bien no es difícil ignorar las teclas que no representan letras o números, por ahora no tendremos en cuenta qué sucede al presionarlas. Si las presiona, aprecerá su código como una palabra.

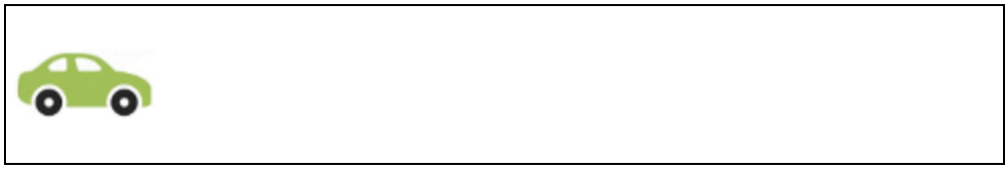
Por ahora es suficiente con estas características para el editor. En prácticas siguientes iremos mejorándolo, hasta crear algo que pueda llamarse, sin avergonzarnos, editor de texto.

5 Vamos de paseo

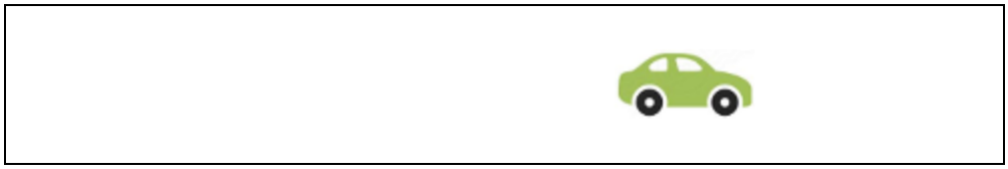
Ejercicio 7. Agregue la siguiente línea al área de definiciones:

```
(define AUTO  )
```

Se pide escribir un programa en el que el auto avance de izquierda a derecha sobre una escena vacía, a razón de 3 píxeles con cada tick del reloj. La imagen inicial debe ser similar a la siguiente:



Luego de 100 ticks,



Defina todas las constantes que considere necesarias. Modifique el programa para que:

- al presionar la barra espaciadora, el auto vuelva a empezar su recorrido,
- El auto no avance en caso de haber llegado al final del recorrido
- al hacer click sobre la imagen, el auto se desplace a la coordenada horizontal donde se produjo el evento

- al presionar la flecha derecha, el auto avance 20 píxeles
- al presionar la flecha izquierda, el auto retroceda 20 píxeles

6 Imágenes como estado

Ejercicio 8. En este ejercicio agregaremos estrellas a un cielo vacío. El estado del sistema será una imagen, nuestro cielo, sobre la cual incorporaremos las estrellas que vayamos dibujando. Defina constantes para representar el color, el ancho y el alto de la escena, así como también el color y el tamaño de las estrellas. Genere una expresión `big-bang` tal que:

- El estado inicial del sistema sea `FONDO` (el cielo vacío).
- La función que responde a la cláusula `to-draw` dibuje el estado actual.
- Al clickear sobre el cielo, se dibuje una estrella donde se realizó el click, respetando el tamaño y el color definidos previamente.
- Puede suceder que una estrella no entre completamente en la escena. Modifique sus funciones para que ésto no ocurra, es decir que una estrella se grafique sólo si va a entrar completamente en el cielo. Caso contrario, la imagen no debe modificarse.
- Si se presiona `backspace`, todas las estrellas deben desaparecer, es decir, el cielo debe volver al estado inicial.
- Por último, modifique sus funciones para que el tamaño de las estrellas varíe de acuerdo a la posición donde se produjo el click, cuanto más a la izquierda sea, más pequeñas deben ser.

7 Más dibujos

Ejercicio 9. En este ejercicio dibujaremos un objeto sobre una escena. Los posibles objetos a dibujar son un círculo o un triángulo, y estos pueden ser de color azul o verde. Para representar el estado utilizaremos un string, con la convención de que la primera letra representa la figura a dibujar (`"t"` o `"c"`), y la segunda letra el color de la figura (`"a"` o `"v"`).

Por lo tanto, si queremos dibujar un círculo azul, el estado que lo representa será `"ca"`. Deténgase y piense ¿Cuántos estados hay? ¿Qué string representa a cada uno de ellos?

Defina constantes para representar el ancho y el alto de la escena, así como el color de fondo. Defina cualquier otra constante que considere necesaria. Por ejemplo, la dimensión del objeto a dibujar.

La expresión `big-bang` deberá comportarse de la siguiente forma:

1. El estado inicial representa un triángulo verde.
2. La función que responde a la cláusula to-draw debe dibujar la figura indicada en el estado en una posición aleatoria. No se preocupe si la figura no cabe íntegramente en la escena. Sin embargo, la coordenada del centro del objeto sí debe estar dentro de la escena.
3. Si se presiona la tecla `"t"`, la figura a dibujar debe pasar a ser un triángulo, manteniendo el color actual. Al presionar la `"c"`, la figura pasa a ser un círculo, también conservando su color. No se preocupe si el objeto se mueve al presionar una tecla, aunque no cambie la forma.
4. Cada 1 segundo la figura debe cambiar de color, alternando entre azul y verde.

Recuerde aplicar la receta para el diseño que hemos visto a todas las funciones que defina. En aquellas funciones que devuelven imágenes puede omitir los casos de test si así lo desea.

8 Huellas

Ejercicio 10. Busque en internet dos imágenes: una huella de un pie derecho y otra de un pie izquierdo. Defina estas imágenes como dos constantes PIEDER Y PIEIZQ, respectivamente.

Se pide escribir un programa que simule a una persona caminando en línea recta de izquierda a derecha o de derecha a izquierda sobre una imagen vacía, a razón de un número impar de píxeles con cada tick del reloj. Tener en cuenta la orientación de las huellas según el sentido en que se avanza. Si llega a un extremo de la escena se considera que la persona gira y se dirige hacia el otro extremo. El estado del programa interactivo guardará la posicion central del pie que se encuentre apoyado en un momento dado. Se arranca con un pie apoyado mirando hacia la derecha. El pie que se dibuja sera el izquierdo si el estado es par o el derecho si es impar. Los pies deben aparecer enteros, no se permite una huella cortada.

Defina todas las constantes que considere necesarias. Modifique el programa para que:

- al presionar la barra espaciadora, vuelva la huella a la posición inicial,
- al hacer click sobre la imagen, se dibuje una huella en la coordenada horizontal donde se produjo el evento (dependiendo de la paridad se debera dibujar uno u otro pie)