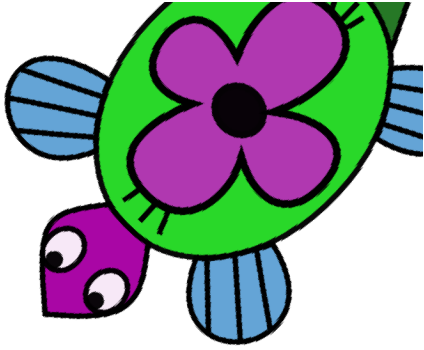


Capítulo 13

Técnicas avanzadas de dibujo

13.1. Más opciones para la tortuga

En la sección 4.4 presentamos algunas primitivas que controlan las opciones de la tortuga. Ampliemos las opciones:

Primitiva	Forma larga	Forma corta
`muestratortuga`, `mt`	`no`	Hace que la tortuga se vea en pantalla.
`ocultatortuga`, `ot`	`no`	Hace invisible a la tortuga.
`pongrosor`	`número`	Define el grosor del trazo del lápiz (en pixels). Por defecto es 1.
`ponformalapiz`, `pfl`	`0 ó 1`	Fija la forma del lápiz: `pfl 0`: cuadrada; `pfl 1`: ovalada. Por defecto la forma es cuadrada.
`grosorlapiz`, `gl`	`no`	Devuelve el grosor del lápiz.
`formalapiz`, `fl`	`no`	Devuelve la forma del lápiz.

Entre otros, `ocultatortuga` es interesante en dos casos:

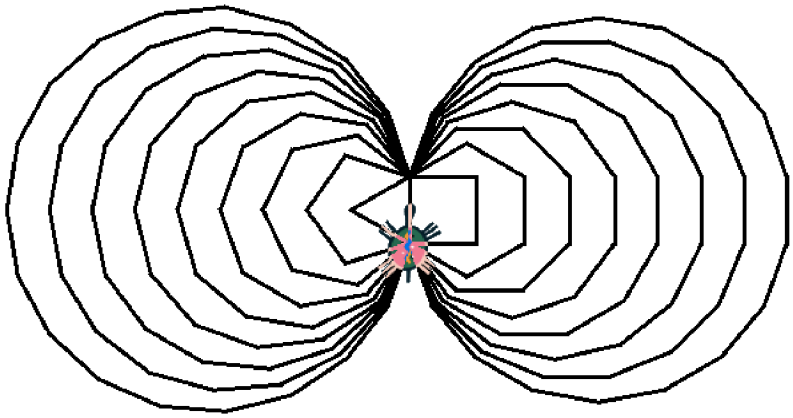
- No tapar parte del dibujo
- Dibujar más rápido

Analiza el procedimiento siguiente:

```
para poli.poligonos :cuantos
 repite :cuantos
```

```
    [ hazlocal "angulo 360/(suma cuentarepite 2)
      si (0 = resto cuentarepite 2)
        [ haz "angulo (-:angulo)]
      repite (suma cuentarepite 2)
        [ avanza 50 giraderecha :angulo ] ]
fin
```

respuesta a un problema planteado en el capítulo 11.4:



Prueba a ejecutarlo de dos formas distintas:

```
borrapantalla poli.poligonos 100

borrapantalla ocultatortuga poli.poligonos 100
```

y observa cómo la segunda vez el dibujo se consigue mucho antes. (En el capítulo 18 veremos como medir exactamente el tiempo de ejecución de un programa).

Respecto a las demás primitivas, vamos a estudiar la siguiente hipótesis:

> "*Si pongo un grosor n y avanza n pasos, dibujaré un cuadrado*"

Teclea las órdenes adecuadas y razona si la hipótesis es cierta o no.

Compara el resultado con el obtenido con la siguiente secuencia de comandos:

```
borrapantalla ponformalapiz 1 pongrosor 200 punto posicion

borrapantalla ponformalapiz 0 pongrosor 200 punto posicion
```

¿Qué observas? ¿Influye en el resultado del análisis anterior?

Prueba distintos dibujos modificando simultáneamente el grosor y la forma, observando claramente las diferencias entre un lápiz cuadrado y ovalado, cúando se aprecia mejor esa diferencia, ...

13.2. Control del color

13.2.1. Primitivas que controlan los colores

Conozcamos ahora las primitivas que controlan el color del trazo, del papel (fondo) y del texto:

Primitiva	Forma larga	Forma corta
Cambiar el `color` del `lápiz`	`poncolorlapiz n`	`poncl n`
Cambiar el `color` del `papel`	`poncolorpapel n`	`poncp n`
`Invertir` el `lápiz`	`inviertelapiz`	`ila`
Averiguar el `color` del `lápiz`	`colorlapiz`	`cl`
Averiguar el `color` del `papel`	`colorpapel`	`cp`
Mirar el `color` de la posición `[X Y]`	`encuentracolor [X Y]`	`ec [X Y]`
`Borrar` por donde pasa	`goma`	`go`
Para volver a dibujar, debe usarse `poncolorlapiz`		

¿Qué es "*invertir el lápiz*"? Como veremos en la sección siguiente y detallaremos en 13.6, cada color en xLogo está codificado usando tres valores: rojo, verde y azul, o `R V A` (RGB en inglés). Deberíamos conocer mínimamente la teoría del color asociada a la luz, que difiere de la que conocemos al tratar con pinturas:
`http://w3.cnice.mec.es/eos/MaterialesEducativos/mem2000/color/Intro/indice.htm`

Si trabajo sobre una paleta de pintor, sabemos que los colores básicos son tres, rojo, amarillo y azul; pero si son "colores de luz" los colores básicos cambian, y pasan a ser rojo, azul y verde y que si mezclo los colores obtengo:

Mezcla	Paleta	Luz
rojo + azul =	violeta	magenta
rojo + amarillo =	naranja	amarillo
azul + amarillo =	verde	cyan
azul + rojo + amarillo =	negro	blanco

Utiliza las opciones del menú para VER (literalmente) lo que ocurre al sumar colores en xLogo. Menú Herramientes → Elegir color de lápiz (o de fondo) → pestaña `RVA` y "juega" con las barras de desplazamiento para comprobar lo que acabamos de contar.

Pues bien, al "invertir el lápiz":

- si la zona del papel por la que pasa está en blanco, pinta del color activo
- si la zona del papel tiene un color traza, no la suma, sino la **resta** de colores:

  rojo − azul = verde rojo − verde = azul azul − verde = rojo

Respecto a la primitiva `encuentracolor`, devuelve la lista `[R V A]` asociada al color, y que detallamos a continuación.

13.2.2. Descripción de los colores

El color en xLogo está especificado por una lista de tres números `[r v a]` comprendidos entre 0 y 255. El número `r` es el componente rojo, `v` el verde y `a` el azul (`[r g b]` en inglés). xLogo tiene 17 colores predefinidos, que pueden ser indicados con su lista `[r v a]`, con un número o con una primitiva. Las primitivas correspondientes son:

Número	Primitiva	[R V A]	Color
0	`negro`	`[0 0 0]`	
1	`rojo`	`[255 0 0]`	
2	`verde`	`[0 255 0]`	
3	`amarillo`	`[255 255 0]`	
4	`azul`	`[0 0 255]`	
5	`magenta`	`[255 0 255]`	
6	`cyan`	`[0 255 255]`	
7	`blanco`	`[255 255 255]`	
8	`gris`	`[128 128 128]`	
9	`grisclaro`	`[192 192 192]`	
10	`rojooscuro`	`[128 0 0]`	
11	`verdeoscuro`	`[0 128 0]`	
12	`azuloscuro`	`[0 0 128]`	
13	`naranja`	`[255 200 0]`	
14	`rosa`	`[255 175 175]`	
15	`violeta`	`[128 0 255]`	
16	`marron`	`[153 102 0]`	

Ejemplo: Estas tres órdenes son la misma:

```
poncolorlapiz naranja
poncolorlapiz 13
poncolorlapiz [255 200 0]
```

13.2.3. Función avanzada de relleno

Las primitivas `rellena` y `rellenazona` permiten pintar una figura. Se pueden comparar a la función "`rellena`" disponible en la mayoría de los programas de dibujo. Esta funcionalidad se extiende hasta los márgenes del área de dibujo. Hay tres reglas a tener en cuenta para usar correctamente estas primitivas:

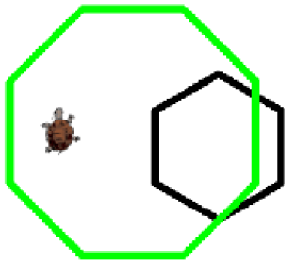
1. El lápiz debe estar bajo (`bl`).

2. La tortuga no debe estar sobre un punto del mismo color que se usará para rellenar. (Si quieres pintar rojo, la tortuga no puede estar sobre un punto rojo).

3. Observar si `cuadricula` está o no activada.

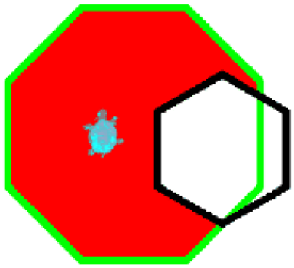
Veamos un ejemplo para explicar la diferencia entre estas dos primitivas:

Los píxeles por donde pasa la tortuga son, en este momento, blancos. La primitiva `rellena` va a colorear todos los píxeles blancos vecinos con el color elegido para el lápiz hasta llegar a una frontera de cualquier color (incluida la cuadrícula):



```
poncolorlapiz 1
rellena
```

produce:

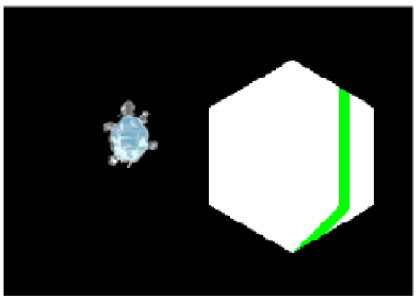


es decir, ha coloreado de rojo la región cerrada en la que se encuentra la tortuga.

Sin embargo, si hacemos:

```
poncolorlapiz 0
rellenazona
```

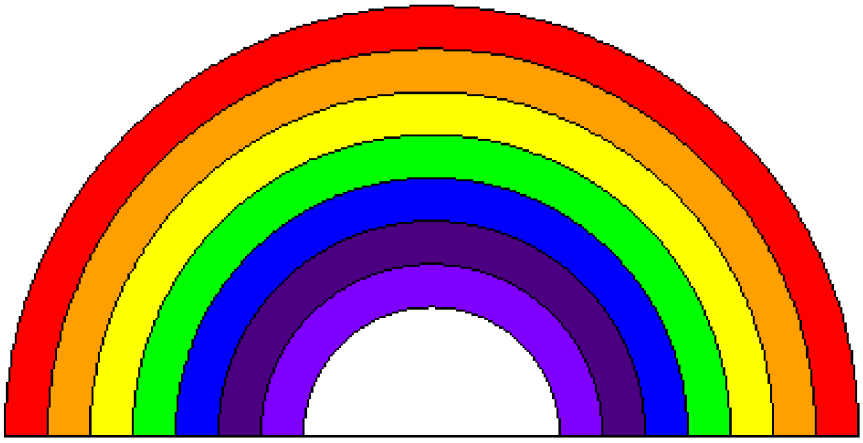
se obtiene:



es decir, rellena todos los píxeles vecinos hasta encontrar una "frontera" del color activo.

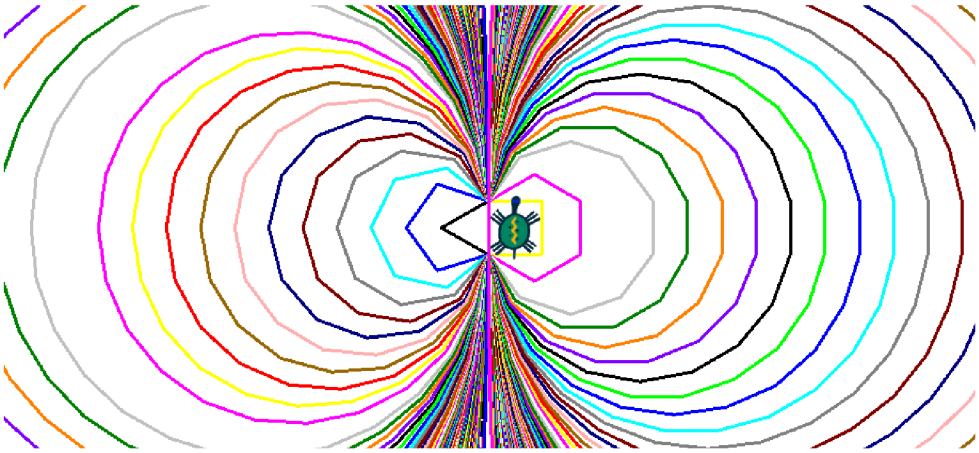
Este es un buen ejemplo para usar la primitiva `rellena`:

```
para mediocirc :c
# dibuja un semicirculo de diametro :c
  repite 180 [
    avanza :c * tan 0.5
    giraderecha 1 ]
  avanza :c * tan 0.5
  giraderecha 90 avanza :c
fin

para arcohueco :c
# Utiliza el procedimiento mediocirc para dibujar un arcoiris sin colores
  si :c < 100 [alto]
  mediocirc :c
  giraderecha 180 avanza 20 giraizquierda 90
  arcohueco :c - 40
fin

para arcoiris
 borrapantalla ocultatortuga arcohueco 400
 subelapiz giraderecha 90 retrocede 150
 giraizquierda 90 avanza 20 bajalapiz
 haz "color [ [255 0 0] [255 160 0] [255 255 0] [0 255 0] [0 0 255]
        [75 0 130] [128 0 255] ]
 repitepara [colores 1 7]
  [ poncolorlapiz elemento :colores :color rellena
    subelapiz giraderecha 90 avanza 20 giraizquierda 90 bajalapiz ]
fin
```

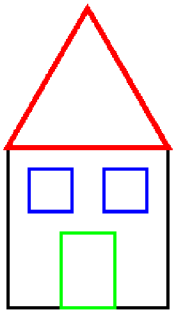


13.3. Ejercicios

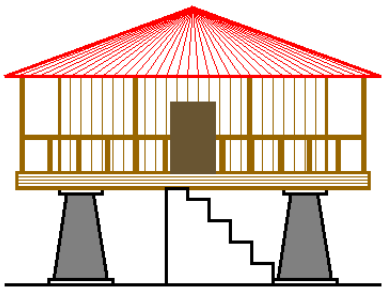
1. Modifica el procedimiento `poli.poligono` para obtener:



2. Combina adecuadamente los procedimientos `cuadrado`, `rectangulo` y `triangulo` que hemos ido diseñando a lo largo del libro para obtener la casa coloreada con la que presentamos a la tortuga "hace tiempo":



3. ¿Te atreves con este hórreo asturiano con corredor?



4. En el capítulo anterior hablamos de usar un mapa mudo en el que el alumno tuviera que ir haciendo "*clic*" en un área concreta según se le fueran mostrando nombres de la misma.

   a) ¿Cómo harías para tener en cuenta las fronteras reales?

   b) ¿Qué requisitos debería tener entonces el mapa para poder hacerlo?

13.4. Control del Área de dibujo

En esta sección podemos distinguir dos tipos de primitivas, las que controlan el tamaño del área de dibujo y las que determinan aspectos del dibujo:

13.4.1. Control del dibujo

Primitiva	Forma larga	Forma corta
`ponzoom` o `ponlupa`	`número`	Acerca o aleja el Área de dibujo. En concreto, el valor de `n` es el factor de escala respecto a la imagen original: `(n >1)` acerca el Área de dibujo; `(0 <n <1)` aleja el Área de dibujo.
`zoom` o `lupa`	`no`	Devuelve el valor del escalado anterior.
`modoventana`	`no`	La tortuga puede salir del área de dibujo (pero no dibujará nada).
`modovuelta`	`no`	Si la tortuga sale del área de dibujo, vuelve a aparecer en el lado opuesto
`modojaula`	`no`	La tortuga queda confinada al área de dibujo. Si intenta salir, aparecerá un mensaje de error avisando cuántos pasos faltan para el punto de salida.

`poncalidaddibujo`, `pcd`	`0, 1 ó 2`	Fija la calidad del dibujo: `pcd 0`: normal; `pcd 1`: alta; `pcd 2`: baja;
`calidaddibujo`, `cdib`	`no`	Devuelve la calidad del dibujo

Ajusta el tamaño de la ventana (**Herramientas** → **Preferencias** → **Opciones** → **Tamaño de la ventana**) a 500 por 300, copia el siguiente procedimiento:

```
para modos
  borrapantalla subelapiz ponx -170 bajalapiz
  repite 360 [avanza 3 giraderecha 1]
fin
```

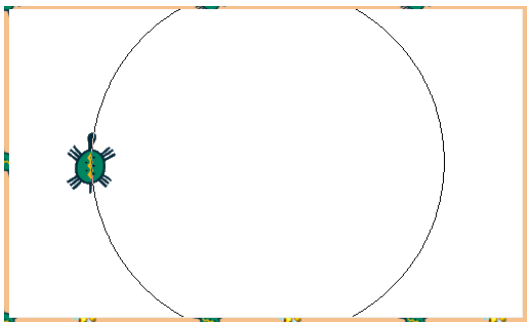
y determina qué pasará en cada uno de los tres modos: `ventana`, `vuelta` y `jaula`.

¿Has podido? Este es el resultado:
Obviamente, en `modovuelta` no se ha completado el dibujo y aparece el mensaje de error:

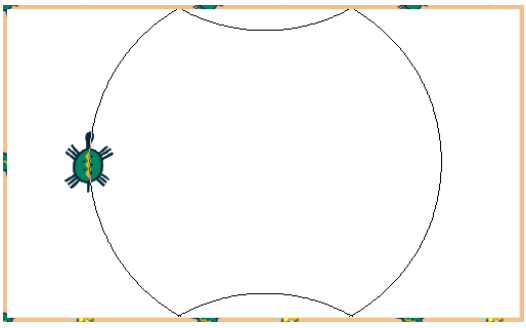
```
En modos, línea 2:
La tortuga sale de la pantalla.
Número de pasos antes de salir:0
```

mientras que para `modovuelta` y `modoventana` nos encontramos con:

Modo ventana

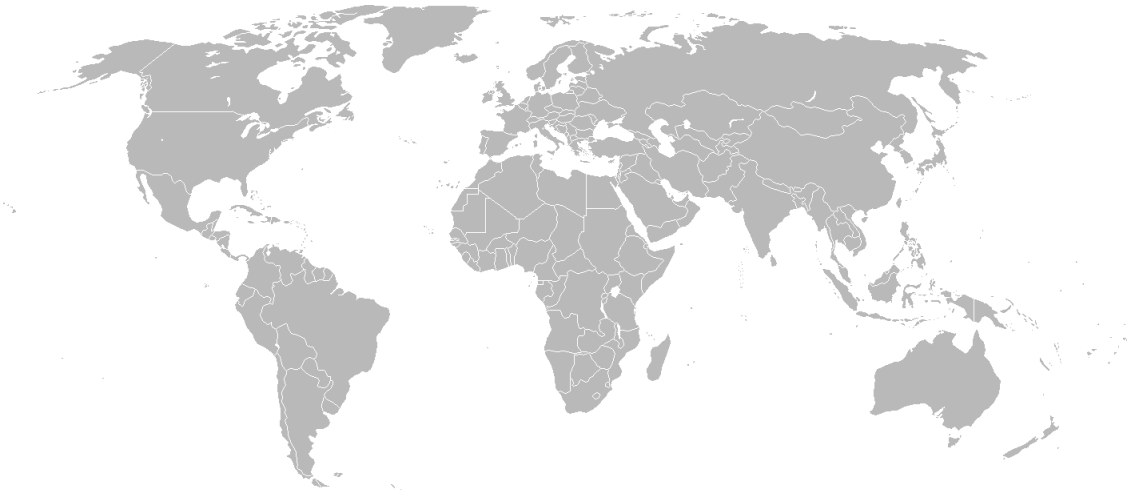


Modo vuelta



pero, cuidado, NO es que la tortuga haya rebotado en la "pared" superior, sino que vuelve a aparecer por la parte de abajo.

Prueba a cambiar el número de pasos que avanza la tortuga o ralentízala, para ver claramente cómo se comporta en el modo vuelta. ¿Se te ocurre para qué podríamos usar este modo? Una pista:



(Recuerda la primitiva `cargaimagen`)

13.4.2. Control de las dimensiones

Primitiva	Forma larga	Forma corta
`pontamañopantalla` `ptp`	`lista`	Fija el tamaño de la pantalla.
`tamañopantalla`, `tpant`	`no`	Devuelve una lista que contiene el tamaño de la pantalla
`tamañoventana`, `tv`, `esquinasventana`	`no`	Devuelve una lista con cuatro elementos, las coordenadas de la esquina superior izquierda y de la esquina inferior derecha.
`ponseparacion`, `ponsep`	`número comprendido entre 0 y 1`	Determina la proporción de pantalla ocupada por el **Área de Dibujo** y el **Histórico de Comandos**.
`separacion`	`no`	Devuelve el valor de la proporción de pantalla ocupada por el **Área de Dibujo** y el **Histórico de Comandos**.

El control de la separación también puede hacerse con el ratón, simplemente arrastrando la separación entre el Área de dibujo y el Histórico de comandos hacia arriba o hacia abajo. En `ponseparacion`, si `n` vale 1, el **Área de Dibujo** ocupará toda la pantalla. Si `n` vale 0, será el **Histórico** quien la ocupe.

13.5. Ejercicios

1. Vamos a calcular el valor de π. Para ello, usa el procedimiento que dibuja una circunferencia (o mejor dicho un Trihectahexacontágono)[1] y:

   a) Permite que el avance sea una variable (p.e. `:lado`)

   b) Haz que la tortuga se desplace a través del diámetro hasta alcanzar el punto diametralmente opuesto al de partida, que localizará con `encuentracolor`

   c) Determine la distancia hasta el punto de partida

   d) Muestre nuestra estimación de π, el resultado de dividir:

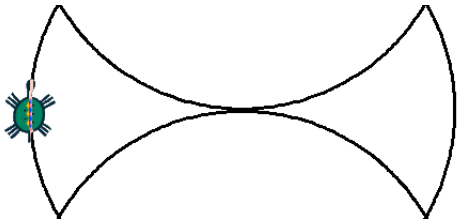
   $$\pi \simeq \frac{360 \times \texttt{:lado}}{\texttt{distancia}}$$

   Prueba para distintos valores de `:lado` y observa los resultados

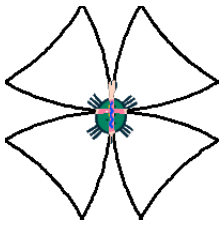
2. En el ejemplo sobre `modojaula`, `modovuelta` y `modoventana`, ¿qué ocurre si usamos `circulo` para dibujar la circunferencia en vez del procedimiento anterior?

3. Observa el dibujo que obtuvimos con `modovuelta`. ¿A qué te recuerda? ¿Podrías usarlo para dibujar, por ejemplo, un murciélago?

4. ¿Cómo harías para que la circunferencia se divida y sea tangente **a sí misma**?

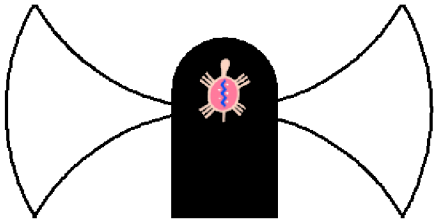


   ¿Y si hacemos que la pantalla sea cuadrada?:

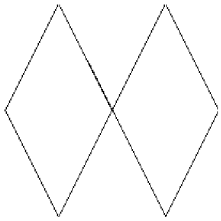


5. Aprovecha ese dibujo para que nuestra tortuga "hable" a través de dos altavoces:

[1] Trihectahexacontágono: tri = 3, hecta = 100, hexaconta = 60, gono = ángulo



6. Intenta dibujar estos dos rombos:



(El tamaño de la pantalla es `200 * 200`)

7. ¿Podrías dibujar un rectángulo (o un cuadrado) **sin que haya giros de 90°**?
(No vale sumar ángulos hasta que den 90°)

8. Dibuja los ejes cartesianos en pantalla

CUIDADO: Si pretendes usar estos dibujos como parte de otros redimensionando la pantalla a mitad de un dibujo, NO LO HAGAS. Al igual que `cuadricula`, `ejes`, ... xLogo borra la pantalla al modificar sus dimensiones. Debes guardarlo como imagen y cargarla depués.

13.6. Manejando imágenes

13.6.1. Introducción

Primero, algunas aclaraciones: Habrás visto en la sección 13.2.2 que el comando `poncolorlapiz` puede tomar como argumento tanto un número como una lista. Aquí nos centraremos en codificar valores RVA. Cada color en xLogo está codificado usando tres valores: rojo, verde y azul, de ahí RVA (RGB en inglés).

Estos tres números conforman una lista que es argumento de la primitiva `poncl`, por lo que representan respectivamente los componentes rojo, verde y azul de un color. Esta manera de codificar no es muy intuitiva, así que para tener una idea del color que obtendrás puedes usar la caja de diálogo **Herramientas → Elegir color del lápiz**.

Sin embargo, usando esta forma de codificar colores, se hace muy fácil transformar una imagen. Por ejemplo, si quieres convertir una foto color en escala de grises, puedes cambiar

cada punto (píxel) de la imagen a un valor promedio de los 3 componentes RVA. Imagina que el color de un punto de la imagen está dado por `[0 100 80]`. Calculamos el promedio: `(0 + 100 + 80)/3 = 60`, y asignamos el color `[60 60 60]` a este punto. Esta operación debe ser realizada para cada punto de la imagen.

13.6.2. Práctica: Escala de grises

Vamos a transformar una imagen color de 100 por 100 a escala de grises. Esto significa que tenemos `100 * 100 = 10000` puntos a modificar.

La imagen de ejemplo utilizada aquí está disponible en la siguiente dirección:

```
http://xlogo.tuxfamily.org/images/transfo.png
```

Utilizaremos la primitiva `cargaimagen` o `ci` que carga el archivo de imagen indicado con una palabra.

La esquina superior izquierda de dicha imagen se ubica en la posición actual de la tortuga. Los únicos formatos soportados son jpg y png. La ruta debe especificarse previamente con `pondirectorio` (capítulo 15) y debe ser absoluta, empezando en el nivel superior del árbol de directorios.

Ejemplo:

```
pondirectorio [/home/alumnos/mis\ imagenes]
cargaimagen "turtle.jpg
```

Así es como vamos a proceder: primero, nos referiremos al punto superior izquierdo como `[0 0]`. Luego, la tortuga examinará los primeros 100 puntos (pixeles) de la primera línea, seguidos por los primeros 100 de la segunda, y así sucesivamente. Cada vez tomaremos el color del punto usando `encuentracolor`, y el color será cambiado por el promedio de los tres `[r v a]` valores. Aquí está el código principal: (¡No olvides cambiar la ruta del archivo en el procedimiento!)

```
para transform
# Debes cambiar la ruta de la imagen transfo.png
# Ej: cargaimagen [/home/usuario/imagenes/transfo.png]
  borrapantalla ocultatortuga
  pondirectorio "/home/usuario/imagenes
  cargaimagen "transfo.png
  escalagris
fin

para escalagris
  repitepara [y 0 -100 -1]
```

```
    [ repitepara [x 0 100]
# asignamos el promedio de color del punto al color del lapiz
      [ poncolorlapiz pixel encuentracolor lista :x :y
# convertimos el punto escala de grises
        punto lista :x :y ] ]
fin

para pixel :lista1
# devuelve el promedio de los 3 numeros [r v a]
  haz "r primero :lista1
  haz "lista1 menosprimero :lista1
  haz "v primero :lista1
  haz "lista1 menosprimero :lista1
  haz "a primero :lista1
  haz "color redondea (:r+:v+:a)/3
  devuelve frase :color frase :color :color
fin
```

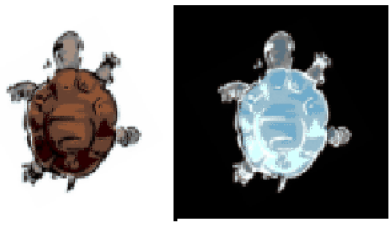
ANTES DESPUES

13.6.3. Negativo

Para cambiar una imagen a su negativo, se puede usar el mismo proceso de la escala de grises, excepto que en lugar de hacer el promedio de los números `[r v a]`, los reemplazamos por su complemento, o sea la diferencia a 255.
Ejemplo: Si un punto (píxel) tiene un color `[2 100 200]`, lo reemplazamos con `[253 155 55]`. Podríamos usar el mismo código que en el ejemplo anterior, cambiando únicamente el procedimiento `pixel`, pero veamos un procedimiento recursivo:

```
para transform2
# Debes cambiar la ruta de la imagen transfo.png
# Ej: c:\Mis Documentos\Mis imagenes\transfo.png
  borrapantalla
  ocultatortuga
  pondirectorio "c:\\Mis\ Documentos\\Mis\ imagenes
  cargaimagen "transfo.png
  negativo 0 0
```

```
fin

para negativo :x :y
  si :y = -100
   [ alto ]
   [ si :x = 100
     [ haz "x 0 haz "y :y-1]
     [ poncolorlapiz pixel2 encuentracolor lista :x :y
       punto lista :x :y ] ]
  negativo :x+1 :y
fin

para pixel2 :lista1
# devuelve el promedio de los 3 numeros [r v a]
  haz "r primero :lista1
  haz "lista1 menosprimero :lista1
  haz "v primero :lista1
  haz "lista1 menosprimero :lista1
  haz "a primero :lista1
  devuelve frase (255 - :r) frase (255 - :v) (255 - :a)
fin
```



13.7. Ejercicios

1. Carga la imagen de un laberinto (o mejor aún, que la tortua dibuje uno) y puedas guiar a la tortuga por el camino de salida. Utiliza la primitiva `encuentracolor` para evitar que "atraviese" las paredes del mismo.

   Intenta que la tortuga dibuje un rastro (recuerda, que debes dibujar el rastro después de encontrar el color del papel) de un color si el camino es el correcto y de otro cuando debes retroceder.

2. Busca o dibuja una casa para nuestra tortuga (no muy grande). Busca también imágenes de casas, parques, bancos, edificios gubernamentales y comerciales, ..., y:

a) Redimensiónalos hasta un tamaño de unos 50 píxeles.

b) Carga esas imágenes formando una "ciudad"

c) Diseña "recorridos", con "tareas" que debe realizar la tortuga, y controla que esas tareas se realizan sin desplazarse por fuera de las "calles".

d) Puedes dibujar puntos de distintos colores en las entradas de cada lugar que se debe visitar para controlar que los sitios a los que se desplaza son los correctos.

3. Vamos a dibujar un "Diagrama de Venn" de colores. Plantea un procedimiento que:

   a) Dibuje dos círculos de radios variables y los rellene con distintos colores

   b) Analice si se cortan (dos circunferencias se cortan si la distancia entre sus centros es menor que la suma de los radios) y, en caso afirmativo:

      1) encuentre el color de cada círculo
      2) sitúe a la tortuga en la intersección de ambas circunferencias (por ejemplo, el punto medio entre sus centros)
      3) rellene esa intersección con el color resultante de sumar o promediar (tú eliges) ambos

   ¿Sabrías ampliarlo a tres circunferencias?