DISEÑO DE UNA ACTIVIDAD DE SECUENCIAS TEMPORALES

PROYECTO DESCARTES

Juan Guillermo Rivera Berrío

2016

Introducción

Este instructivo se elaborado como una contribución a los diseñadores de escenas Descartes que están apenas incursionando en el desarrollo de actividades interactivas, igualmente puede servir a desarrolladores cartesianos, especialmente en algunas novedades que presentan las últimas versiones del editor Descartes, tales como el centrado de textos en cajones, diseño JavaScript de botones y cambio de estilos de fuente.

Acompaña a este instructivo una carpeta que contiene:

- Archivo index.html. Es el archivo que ejecuta la actividad.
- Archivo indexb.html. Es el archivo a intervenir con el editor Descartes.
- Carpeta imágenes. Contiene 12 imágenes a usar en el diseño de la actividad.
- Carpeta images. Contiene la imagen de fondo.
- Carpeta lib. Contiene el intérprete Descartes.
- Carpeta js. Allí se encuentra un archivo que permite que la actividad sea adaptable a cualquier tamaño de ventana del navegador.
- Carpeta fonts. Contiene una fuente especial que usaremos al final del instructivo.

El archivo indexb.html sólo presenta la configuración del espacio de trabajo, el objetivo es construir la actividad siguiendo las instrucciones que se dan a lo largo de este documento. El diseñador es libre de cambiar algunos elementos de diseño, como imágenes, los textos, máxima calificación y número de ejercicios. Comprendido el instructivo, se pueden emprender otras actividades similares o esta misma con un mayor número de contenedores.

DISEÑANDO UNA ACTIVIDAD DE SECUENCIAS TEMPORALES

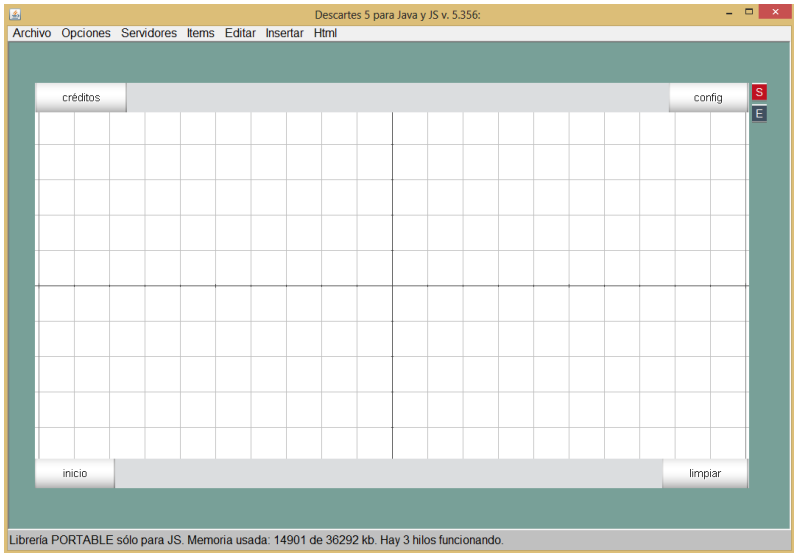
En este apartado presentaremos, paso a paso, cómo se diseña una actividad lúdica de secuencias temporales con el editor Descartes. Nuestro propósito es explicar algunos elementos mínimos de programación usados en este diseño, para lo cual suponemos que el usuario está familiarizado con el editor Descartes. No entregamos el resultado final, pues nuestro objetivo es que el diseñador lo logre.

Hemos seleccionado el tema de secuencias temporales que, para nuestra actividad, contendrá tres imágenes por ejercicio.

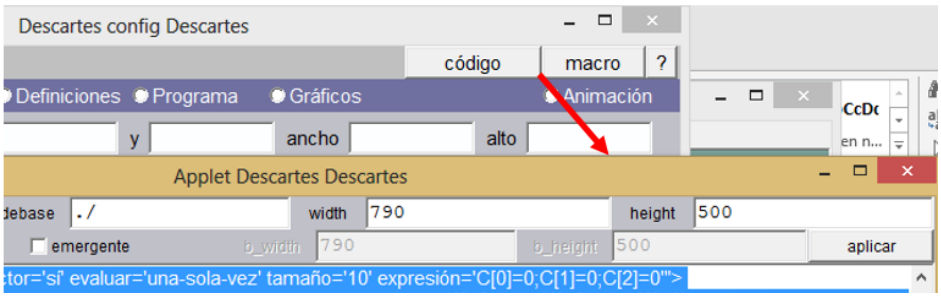
Dimensiones de la plantilla

Para esta actividad hemos optado por dimensiones de 790x500 pixeles para el espacio de trabajo. Lo primero, entonces, es cambiar las dimensiones, para lo cual seguimos los siguientes pasos:

- Abrimos el editor de Descartes:



- Abrimos la ventana de configuración. En esta ventana seleccionamos el botón **código**. Cambiamos el ancho y el alto de la plantilla como lo indica la imagen. Hacemos clic en aplicar y, finalmente, en la ventana de configuración, hacemos clic en aceptar.



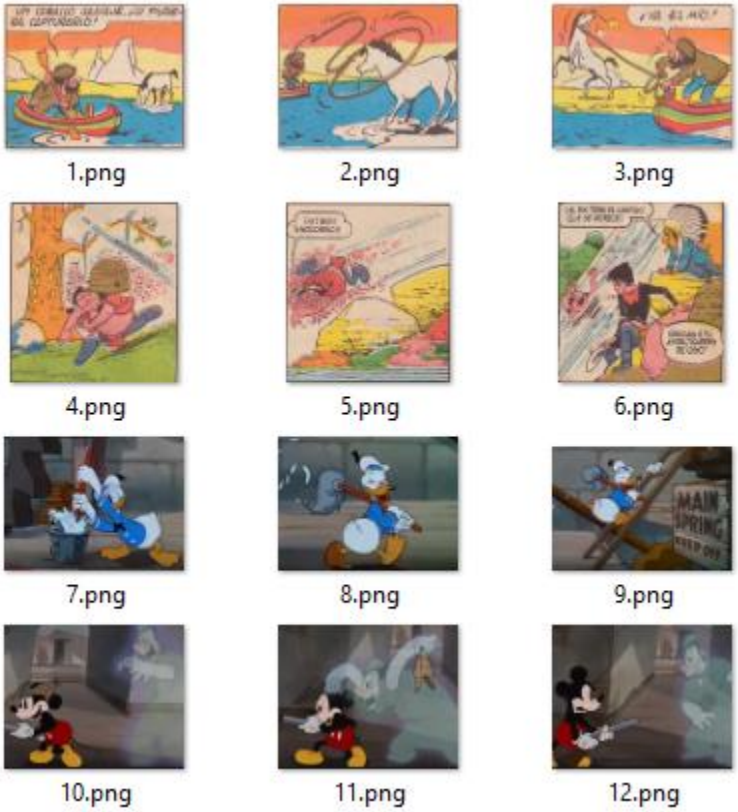
Este paso ya lo hemos realizado, así que sólo tienes que abrir el archivo indexb.html con el editor Descartes o, si ya realizaste el cambio de tamaño de la escena, guarda la actividad con el nombre indexb.html.

Imágenes

Son dos carpetas de imágenes que vamos a usar. La primera, llamada **images**, contiene una imagen correspondiente al fondo que tendrá nuestro espacio de trabajo. Obviamente, se puede elegir otra imagen de fondo o sólo usar un color de fondo.

fondo.jpg

La segunda carpeta, llamada **imágenes**, contiene las imágenes que usaremos como secuencias temporales. Para esta actividad, hemos seleccionado cuatro tríos de imágenes que hemos obtenido de la página http://www.dedominiopublico.org/, la cual ofrece revistas y películas que, por su antigüedad, son de dominio público. El criterio de selección, el nombre y el tamaño de las imágenes, las explicamos más adelante. Las imágenes de esta carpeta, también, se pueden cambiar a elección del diseñador (estas carpetas se suministran con este instructivo).



1.png 2.png 3.png

4.png 5.png 6.png

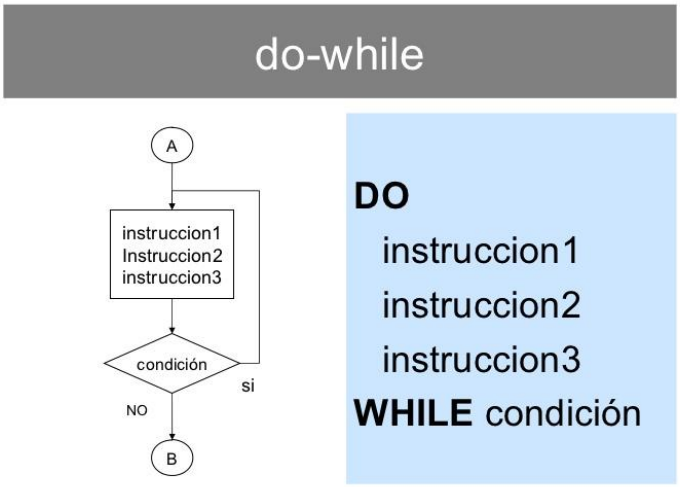
7.png 8.png 9.png

10.png 11.png 12.png

Los dos primeros tríos de imágenes son capturados de la revista de cómics TBO y, los dos últimos tríos, de dos películas de Disney de 1937 (limpiadores de relojes y fantasmas solitarios).

Diseño de los cajones – Uso de familias

Una de las estructuras más utilizadas en programación es el ciclo o bucle tipo *do-while* (hacer – mientras), cuyo propósito es ejecutar un bloque de código y repetir la ejecución mientras se cumpla cierta condición expresada en la cláusula *while*, tal como se observa en la siguiente imagen:



En código JavaScript, un ejemplo de este tipo de ciclo sería:

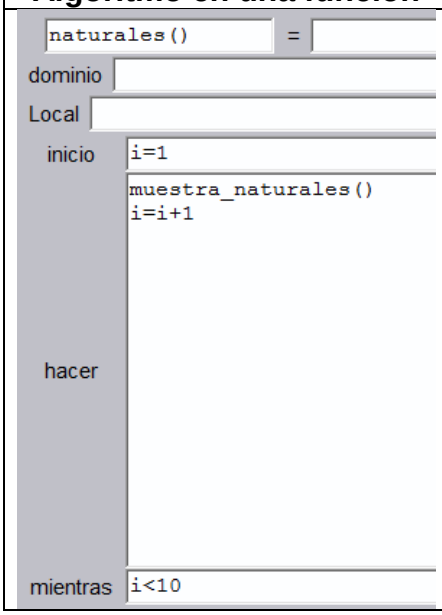
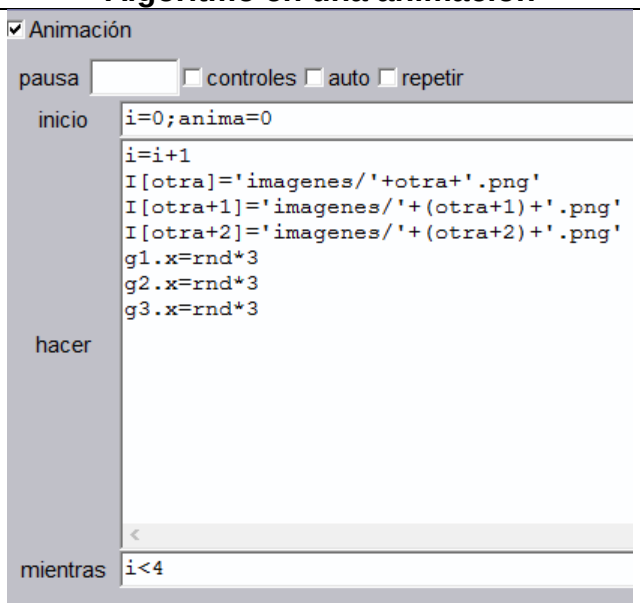
```
i = 1;
do {
        document.write(i);
        i = i + 1;
while (i<10);}
```

Cuyo propósito es escribir los números naturales del 1 al 10. Una variante del código anterior, que nos permitirá acercarnos al funcionamiento de las familias Descartes, sería el siguiente:

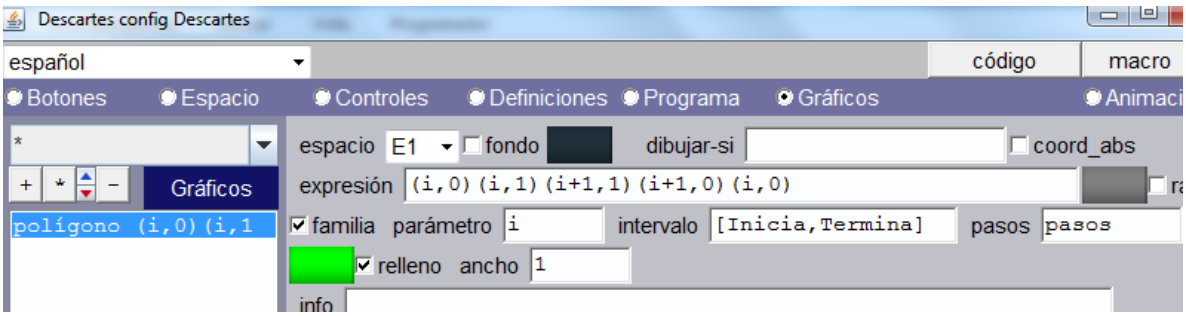
```
i = Inicia;
do {
        function Dibujar_Poligono(i);
        i = i + Incremento;
while (i<Termina);}
```

En el que se presenta una llamada a una función que dibujará un polígono, es decir, se dibujarán tantos polígonos como la condición del while sea verdadera.

En el editor Descartes hay varias formas de crear algoritmos con la estructura *do-while*. En la siguiente tabla observamos dos ejemplos, el primero corresponde a una función (opción **definiciones** del menú de configuración) que muestra los números naturales del 1 al 9 (verifica por qué hasta nueve). El segundo algoritmo corresponde a una animación, que explicaremos al final de este instructivo.

Algoritmo en una función	Algoritmo en una animación
	

Otra de las opciones interesantes de Descartes es el uso de familias para el diseño de gráficos. El funcionamiento de esta opción es similar al de un ciclo o bucle tipo *do-while*. En Descartes, el segundo ejemplo *do-while* en código JavaScript sería de la siguiente forma:

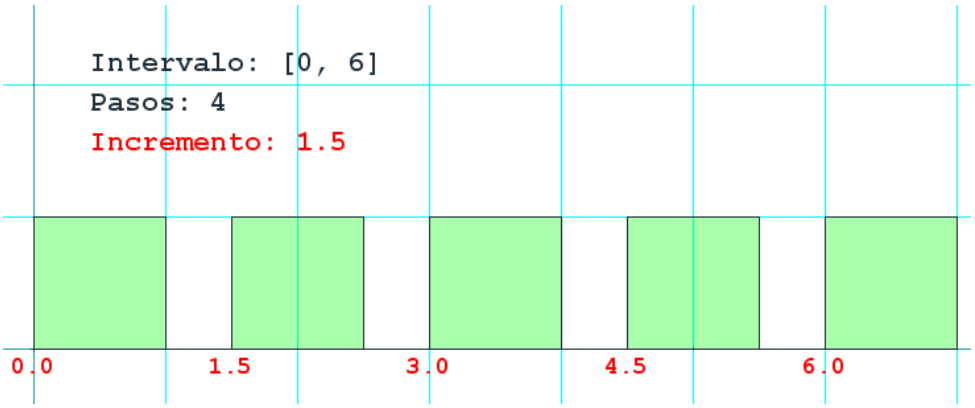


En la opción gráficos del editor de configuración hemos creado un polígono cuyas coordenadas varían de acuerdo a la variable **i** (obsérvese que está activada la opción **familia**). Esta variable toma valores según el intervalo, es decir, desde un valor inicial que hemos llamado **Inicia**, hasta un valor final llamado **Termina**, el incremento de la variable depende del número de pasos definidos, que se calcularía así:

Incremento = (Termina – Inicia)/pasos

Veamos unos ejemplos para diferentes valores del intervalo y de la variable pasos:

Primer ejemplo

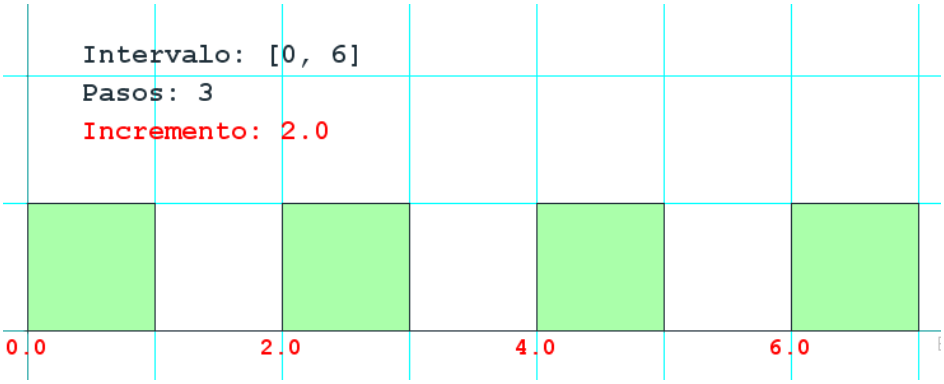


El incremento en este ejemplo sería 6/4 = 1.5. El funcionamiento del ciclo do-while sería así:

Para i=0 se dibuja el polígono (0, 0)(0, 1)(1, 1)(1, 0)(0, 0). Luego i se incrementa en 1.5

Para i=1.5 se dibuja el polígono (1.5, 0)(1.5, 1)(2.5, 1)(2.5, 0)(1.5, 0). Luego i se incrementa en 1.5… y así **mientras** i<6. Una característica del ciclo do-while que aplica para las familias Descartes, es que cuando i=6 las instrucciones se ejecutan, es decir, el polígono con i = 6, también se dibuja.

Segundo ejemplo

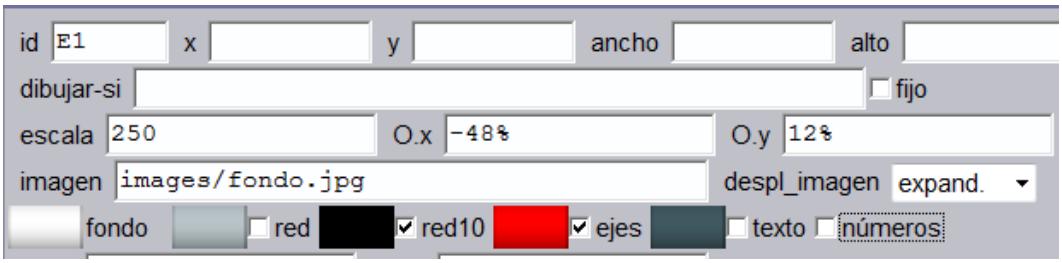


El análisis anterior queda de tu cuenta. Una conclusión del funcionamiento de las familias de Descartes está en la variable pasos, pues el número de polígonos dibujados es igual a **pasos + 1**, con pasos igual a un número entero positivo.

Ahora, vamos a nuestra actividad. Nuestra tarea es dibujar tres polígonos que se constituirán en los cajones en los cuales irán en orden secuencial las imágenes. Alguien podría decir que bastaría con dibujar tres rectángulos y olvidar lo de las familias, conclusión que es cierta, pero el uso de familias facilita el diseño de plantillas similares con 4 o 6 cajones, tal como están publicadas en el proyecto

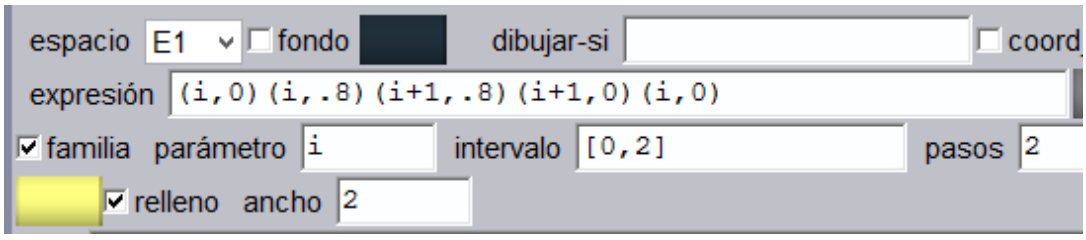
Plantillas de Descartes, además de otros diseños en los cuales las familias son fundamentales (los puzles, por ejemplo).

Nuestro espacio de diseño, como vimos antes, es de 790x500 pixeles que, desde un análisis aritmético simple, nos permite inferir que un ancho de 250 para nuestros cajones sería el adecuado. Este ancho determina la escala que usaremos en nuestro espacio de trabajo, tal como se aprecia en la siguiente imagen (opción espacio en la ventana de configuración):

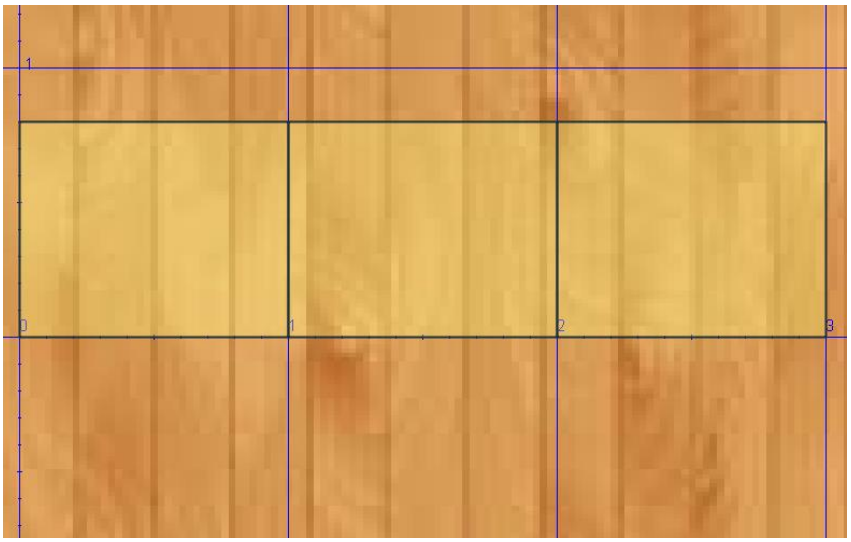


La posición del origen de coordenadas la hemos desplazado, de tal forma que podamos dibujar nuestros polígonos en el primer cuadrante así: desplazamiento del "**eje y"** en un 48% a la izquierda, esto significa que el **eje y** queda casi en el borde izquierdo del espacio de trabajo (inicialmente estaba en el centro del espacio, por ello lo de 48%); desplazamiento del eje x en un 12% hacia arriba, este valor se puede ajustar de tal forma que quede el espacio suficiente para los títulos y las imágenes, es decir, a medida que avances en el diseño de la actividad puedes regresar a este paso para cambiar la posición del origen de coordenadas.

Nuestras imágenes provienen de tiras cómicas que, generalmente, tienen un valor mayor en el ancho. Partiendo de esta premisa, hemos escogido como tamaño de imágenes de 250x200 pixeles. Así las cosas, la familia de polígonos a construir es la siguiente:



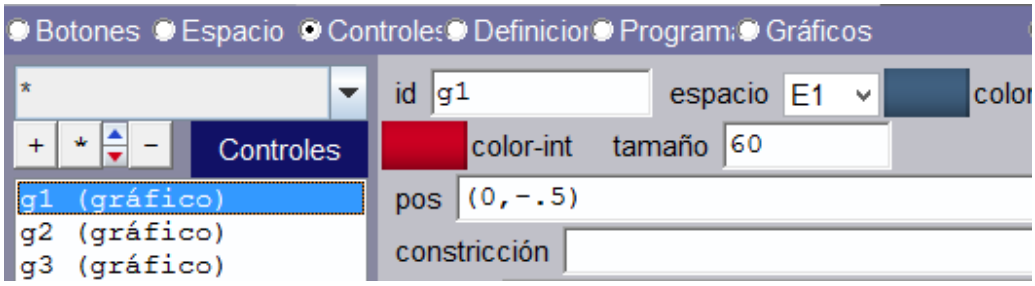
Con un fondo de color amarillo y transparencia **ae**, el resultado es el siguiente:



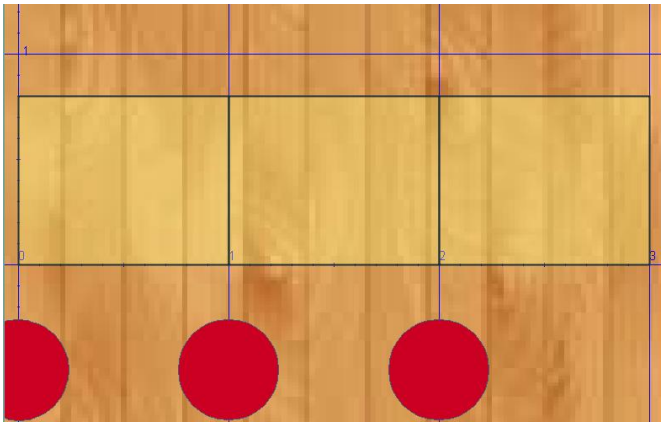
El fondo del espacio es la imagen **fondo.jpg** con desplazamiento expandido, hemos dejado, por ahora, los ejes en color azul. Observa que los tres polígonos inician en i=0, i=1 y i=2.

Controles gráficos e imágenes

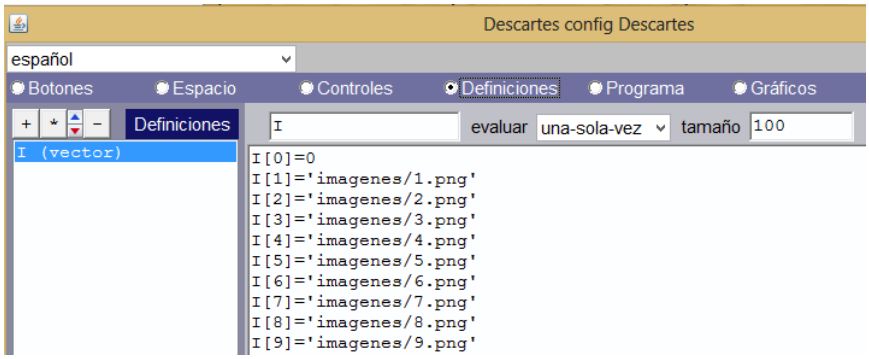
Como nuestro propósito es arrastrar imágenes a los cajones, necesitamos tres controles gráficos a los cuales asociaremos las imágenes. Estos controles tendrían la siguiente configuración:



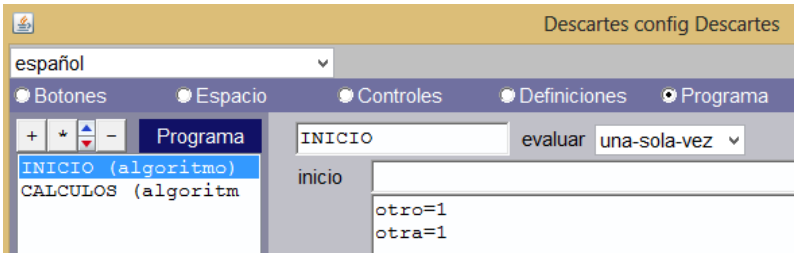
Inicialmente ubicados en (0,-0.5), (1,-0.5) y (2,-0.5) para g1, g2 y g3 respectivamente, de un tamaño de 60, tal como se aprecia en la imagen:



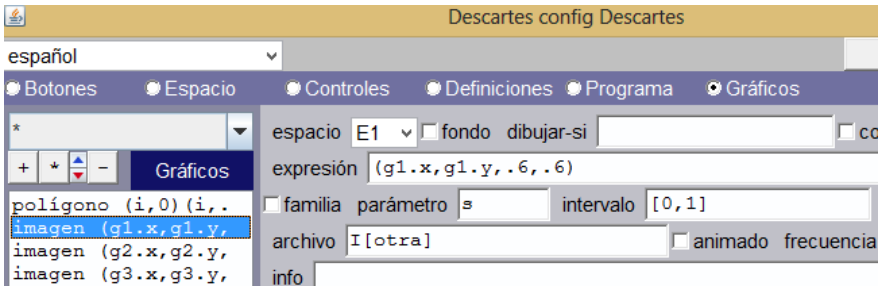
En la opción **definiciones** creamos un vector **I** que contendrá las imágenes que vamos a usar. En principio, hemos dejado un tamaño de cien (100), suficiente para una treintena de ejercicios y, obviamente, para los cuatro de nuestra actividad. Es importante definir los elementos del vector, así: I[1]='imágenes/1.png', I[2]='imágenes/2.png' y así sucesivamente hasta la última imagen a usar en la actividad.



Nuestras imágenes se irán mostrando dependiendo del ejercicio que se esté ejecutando, por ello, hemos definido dos variables en el **algoritmo inicio**, iniciadas con valor uno (1). La variable **otro** determina el ejercicio en ejecución, inicialmente uno para el primero. La variable **otra** determinará el número de la primera imagen correspondiente a ese ejercicio, **uno** para el primer ejercicio. Las variaciones correspondientes a estas variables las veremos más adelante.

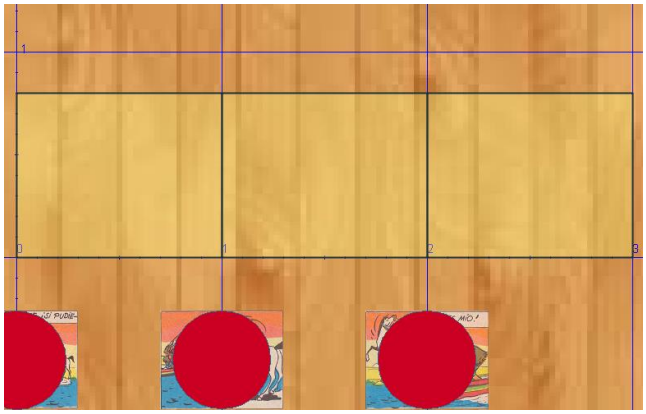


Nuestro siguiente paso es agregar las imágenes en el opción gráficos. Para ello, agregamos una primera imagen cuya configuración es la siguiente:

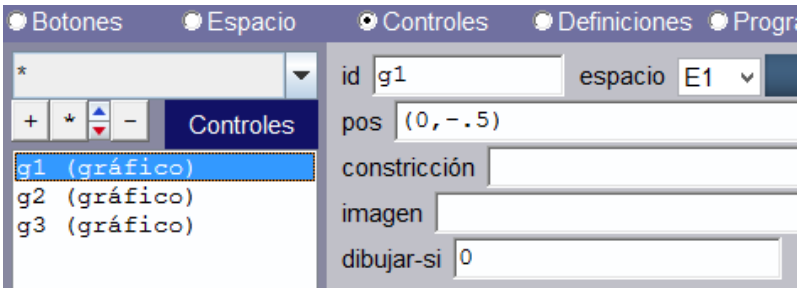


La expresión (g1.x, g1.y, 0.6, 0.6) significa lo siguiente: los dos primeros términos indican la posición de la imagen, es decir, se encontrará donde esté el control gráfico g1, igual para las otras dos imágenes asociadas a los controles g2 y g3; los dos últimos valores indican el tamaño de la imagen, es decir, un ancho y un alto

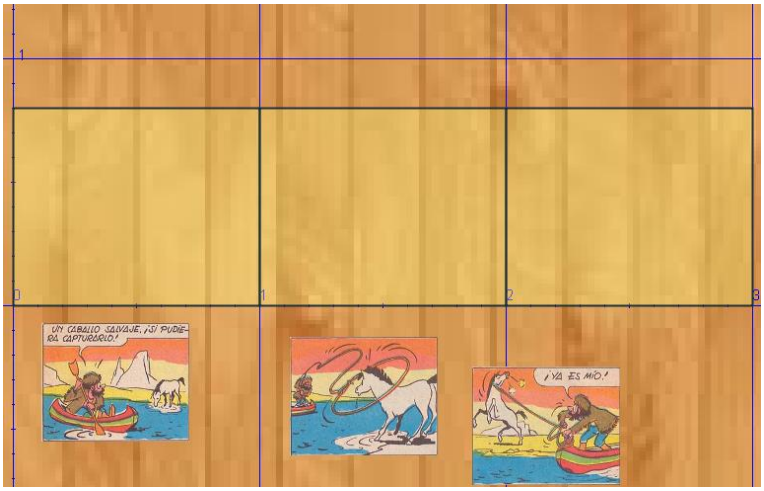
del 60% del tamaño original. Ahora, el archivo que contiene la primera imagen se carga a través el elemento I[otra], donde la variable otra tiene un valor de uno (1), el archivo para la segunda imagen es I[otra+1] , y para la tercera I[otra+2]. Así las cosas, obtendríamos lo siguiente:



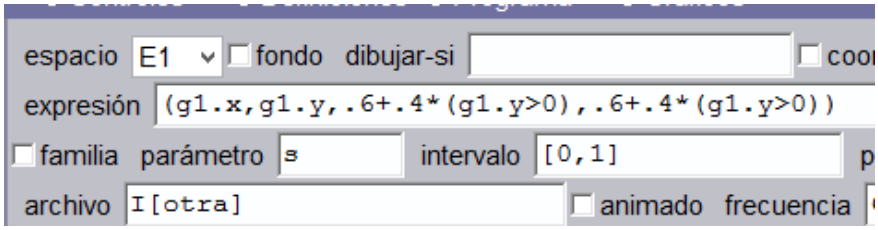
El tamaño de 60 para el control gráfico, ahora es comprensible. Obviamente, no nos interesa que el control se visione en nuestro espacio de trabajo, por ello, dejaremos un valor cero (0) en la casilla **dibujar-si**, que tendrá como efecto la desaparición visual de los controles.



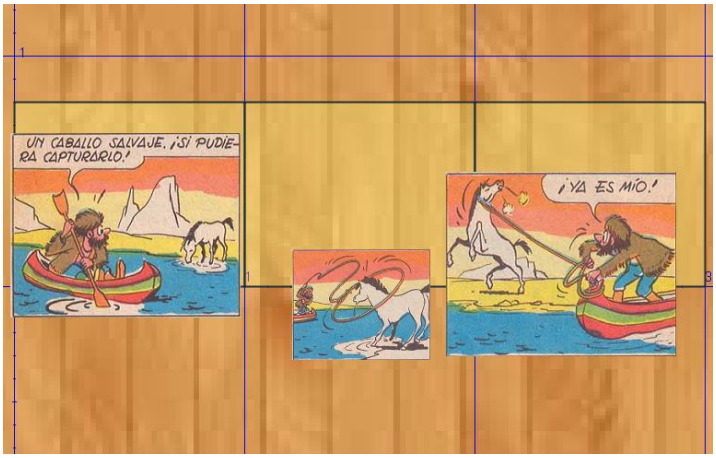
Resumiendo, nuestra actividad tendría la siguiente presentación:



Una última acción sobre las imágenes, que haremos en este apartado, es el cambio del tamaño cuando las imágenes están cerca de los cajones o contenedores. En la imagen anterior se observa que los tamaños de las imágenes están reducidas, lo cual se hizo intencionalmente para que cupieran en el espacio inferior de la escena. Intervendremos los dos últimos valores de la expresión de cada imagen, sumándoles 0.4*(g1.y>0), es decir, cuando arrastremos la imagen hacia arriba, una vez que sobrepasemos el eje x, el tamaño se aumentará en 0.4, obteniendo su tamaño original. Para las otras imágenes es similar, cambiando el control gráfico.

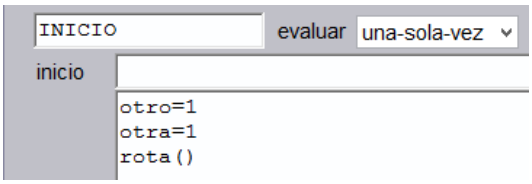


El efecto sobre la actividad se puede observar en la siguiente imagen:

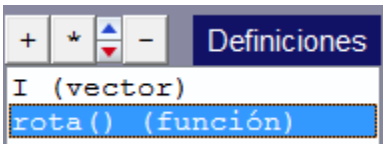


Rotaciones de las imágenes – algoritmos y funciones

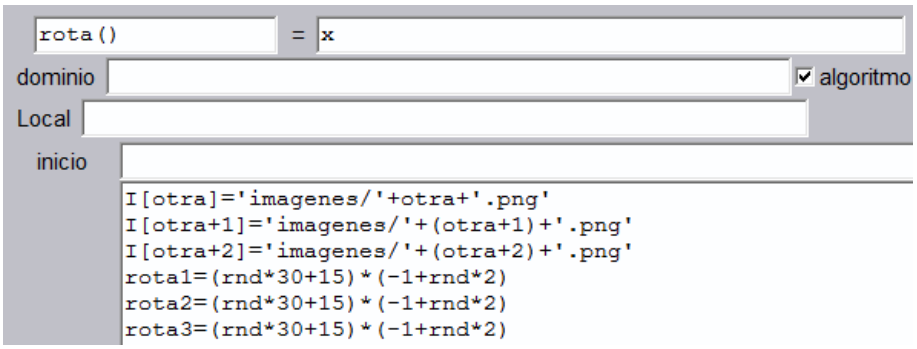
Un efecto adicional es generar rotaciones en las imágenes que desparecerán una vez las arrastremos hacia los contenedores. Para ello, hemos incluido una función que denominamos **rota()**. Esta función la invocamos **una sola vez** por ejercicio en el algoritmo de inicio.



En la opción **Definiciones** creamos la función rota():

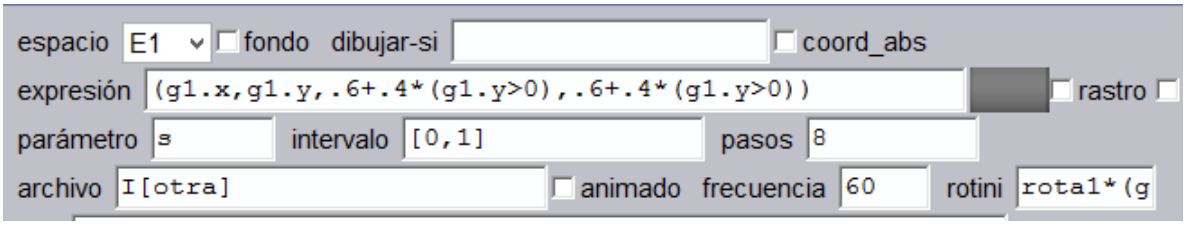


Esta función será un algoritmo con las siguientes instrucciones:

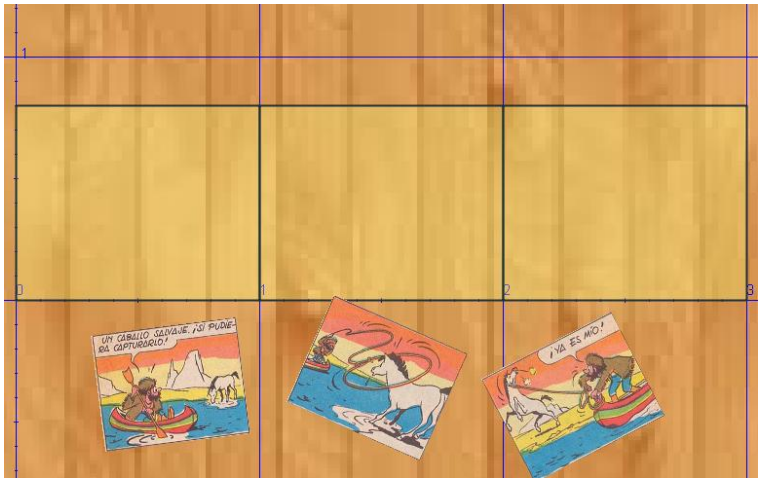


Las primeras tres instrucciones asignan a los elementos I[otra], I[otra+1] y I[otra+2], las imágenes correspondientes a cada ejercicio que, para la primera secuencia de imágenes sería I[1]='imágenes/1.png', I[2]='imágenes/2.png' y I[3]='imágenes/3.png'. A continuación, se generan tres valores asociados a las variables rota1, rota2 y rota3, valores aleatorios entre 0 y 45 (obtenidos por rnd*30+15), cada unjo de estos valores se multiplican por la expresión, también aleatoria, (-1+rnd*2), la cual asigna un valor positivo o negativo a las rotaciones.

En nuestras imágenes incluiremos en la casilla **rotini** la expresión rota1*(g1.y<0), que significa asignar las rotaciones antes creadas, mientras las imágenes estén por debajo del eje x.



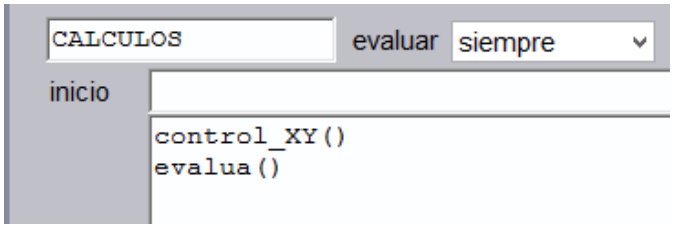
Así, entonces, nuestra actividad tendrá esta apariencia:



Control de coordenadas y evaluación – más funciones

Nuestros siguientes pasos están centrados en evitar que las imágenes se salgan del espacio de trabajo y se ajusten adecuadamente a los cajones. Por otra parte, debemos verificar que la ubicación de cada imagen esté en la posición correcta.

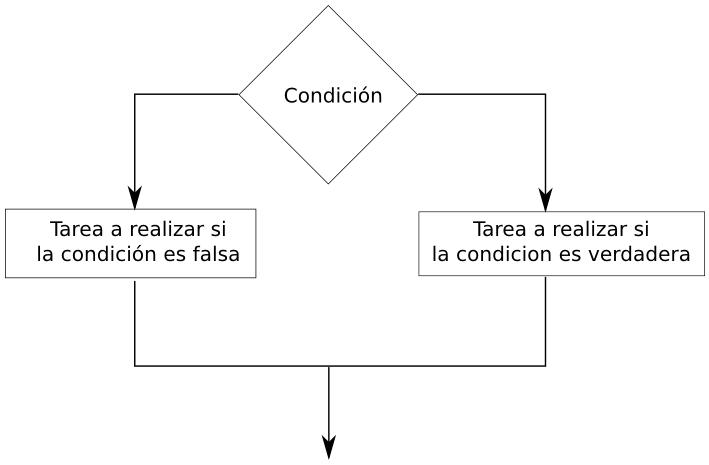
En el algoritmo **CALCULOS** incluiremos el llamado a dos funciones. La primera la hemos denominado control_xy(), la segunda la llamamos evalua(). Teniendo como referencia la imagen de arriba, explicaremos la primera función.



Función control_xy(). Debemos evitar que el control gráfico **g1** asociado a la primera imagen se desborde del espacio de trabajo. Las instrucciones que permiten evitar este desborde son:

```
g1.y=(g1.y>0.4)?0.4:g1.y
g1.y=(g1.y<-0.5)?-0.5:g1.y
g1.x=(g1.x<0.5)?0.5:g1.x
g1.x=(g1.x>2.8)?2.8:g1.x
```

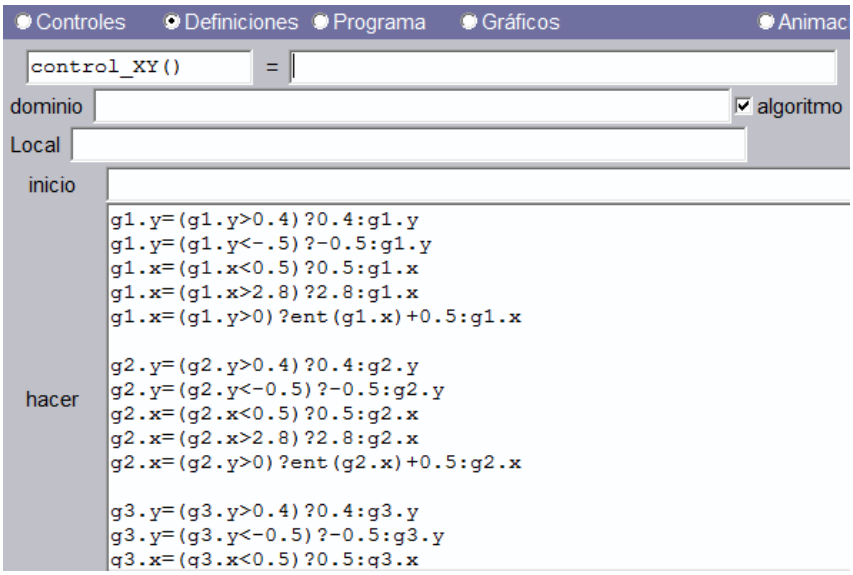
Si observamos la escena, el tope superior está en la mitad de los cajones, es decir, en 0.4, para evitar un desplazamiento más arriba de este valor usamos la estructura condicional **g1.y = (g1.y>0.4)?0.4:g1.y**, la cual podríamos explicar desde su forma en diagrama de flujo:



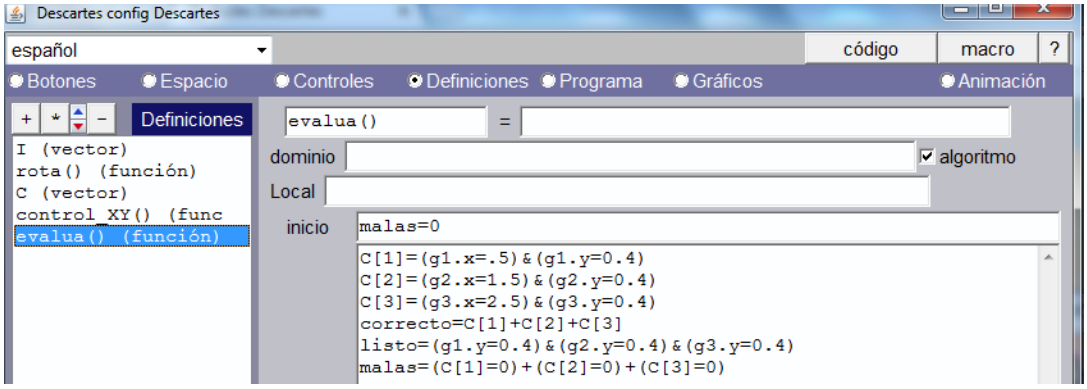
La condición es (**g1.y>0.4)**, si es verdadera se ejecuta la instrucción que está después del signo **?**, es decir, g1.y=0.4 (la ordenada del control gráfico será igual a 0.4, siempre que ésta supere ese valor); si es falsa, se ejecuta la instrucción que está después del signo **:**, es decir, **g1.y = g1.y** (no cambia la ordenada del control gráfico). Igual análisis puedes hacer para las otras instrucciones, las cuales evitan

que la ordenada del control gráfico sea menor a -0.5 y, además, que su abscisa no sea inferior a 0.5 ni superior a 2.8.

Por otra parte, debemos garantizar que la imagen quede incrustada en alguno de los tres cajones, para lo cual recurrimos al siguiente condicional: **g1.x = (g1.y>0)?ent(g1.x)+0.5:g1.x.** El cual fija la abscisa al centro de los cajones (0.5, 1.5 o 2.5). Estas mismas instrucciones las replicamos a los otros controles gráficos.



Función evalua(). Para esta función hemos creado, inicialmente, un vector C, que permitirá almacenar las posiciones correctas (1) o no (0) de las tres imágenes. La secuencia estará bien colocada si la primera imagen está en la posición g1.x = 0.5 y g1.y = 0.4, la segunda imagen en g2.x = 1.5 y g2.y = 0.4 y la tercera en g3.x =2.5 y g3.y = 0.4. Si lo anterior ocurre, C[1], C[2] y C[3] serán iguales a uno (1).



El algoritmo de la función incluye, además, las siguientes variables:

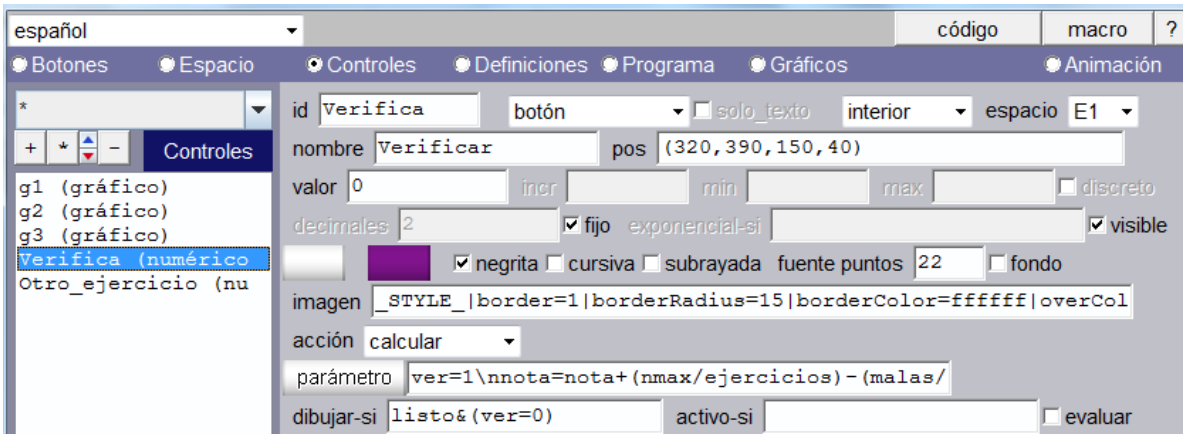
- **correcto**. Es la suma de los tres elementos del vector C, para una secuencia correcta su valor debe ser tres (3).
- **listo**. Es igual a uno (1) cuando todas las piezas están en los cajones

- **malas**. Es la suma de las piezas mal colocadas, es decir, la suma de los elementos del vector C que valen cero (0).

Botones verificar y otro ejercicio

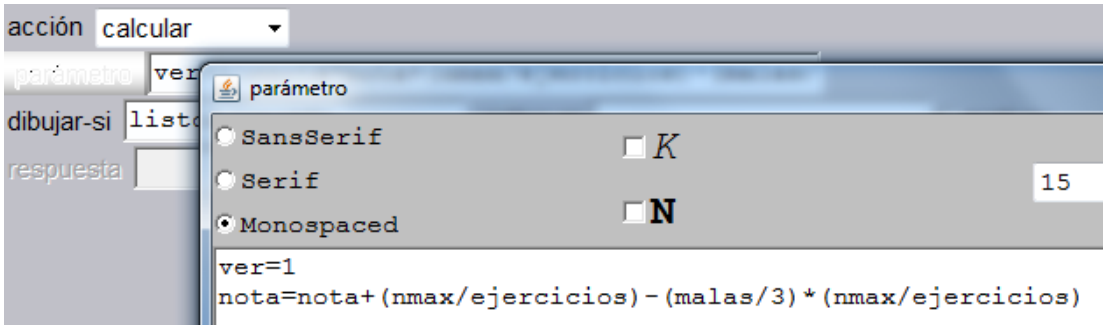
Siempre que estemos diseñando una actividad con Descartes debemos adoptar el papel de usuario, pues ello nos permitirá incluir algunas acciones que hacen la actividad más atractiva o divertida para los demás usuarios. Por ejemplo, una vez se ubiquen las piezas en los cajones, podríamos optar por presentar los mensajes de acierto o error, lo que nos obligaría a inactivar los controles gráficos, impidiendo alguna corrección. Para evitar esta situación, es preferible usar un botón de verificación, en el que el usuario hará clic una vez esté seguro de su respuesta.

Diseñamos, entonces, este botón así:



El texto del botón será **Verificar** en negrilla y tamaño 22 con los colores mostrados en la imagen anterior, lo hemos centrado horizontalmente en 320 (este valor se obtiene restando el ancho de 790 y dividendo por dos), su ancho es 150 y el alto de 40.

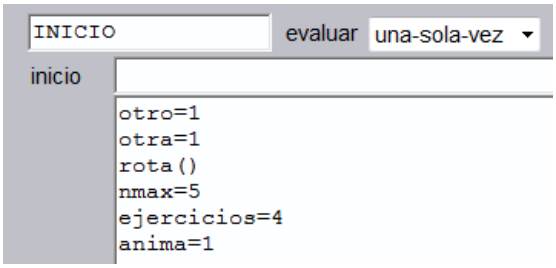
La acción es **calcular** las siguientes instrucciones:



- ver = 1
- nota=nota+(nmax/ejercicios)-(malas/3)*(nmax/ejercicios)

La variable **ver**, cuando vale uno (1), nos servirá para mostrar el segundo botón y los textos.

La variable **nota** almacena la calificación de la actividad. Las demás variables de la expresión las hemos definido en el algoritmo **INICIO**:



Aquí es importante observar que a medida que avanzamos en el diseño de una actividad, tendremos que intervenir pasos anteriores, como es el caso de este algoritmo, en otras palabras, la programación de una actividad no es posible hacerla en forma lineal, siempre tendremos que retornar a los pasos anteriores.

Continuando con las instrucciones, la variable **nmax** tiene un valor de cinco (5), que corresponde a la nota más alta de calificación, valor que puede ser cambiado según la escala de calificación utilizada en el entorno del diseñador (20 en Venezuela, por ejemplo). La variable **ejercicios** tiene un valor de cuatro, que es la cantidad de ejercicios de esta actividad, igualmente la puedes cambiar.

Ahora, con estos valores entenderemos la expresión que calcula la **nota**. Por cada ejercicio, sumará a la nota (nmax/ejercicios) o (5/4) o 1.25, que para los cuatro ejercicios sería un total de cinco (5). Por cada pieza **mal** colocada, la **nota** se castigará en (malas/3)*(nmax/ejercicios), es decir, si tenemos dos piezas mal ubicadas, la nota se reduce en (2/3)*1.25.

El botón se muestra cuando la variable **listo** es igual a uno (ver función evalua()) y **ver** es cero.

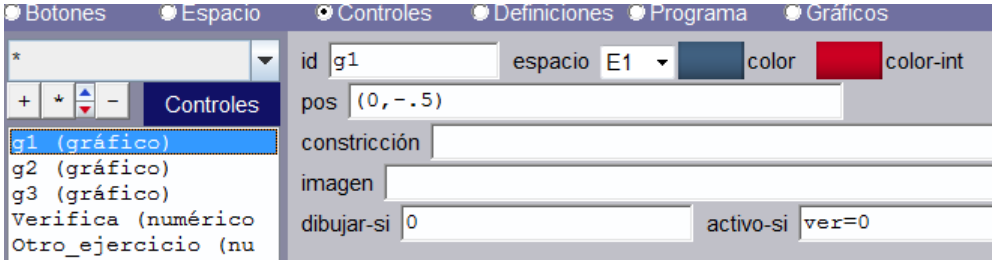
Finalmente, hemos incluido un diseño especial al botón en la casilla **imagen**. Si está casilla la dejamos en blanco, el diseño del botón es el que trae por defecto Descartes, también es posible usar una imagen o, para nuestro caso, el siguiente código JavaScript:

STYLE|border=1|borderRadius=15|borderColor=ffffff|overColor=e0a12b|downColor=ff0000|font=Monospace|shadowTextColor=000000|shadowBoxColor=808080

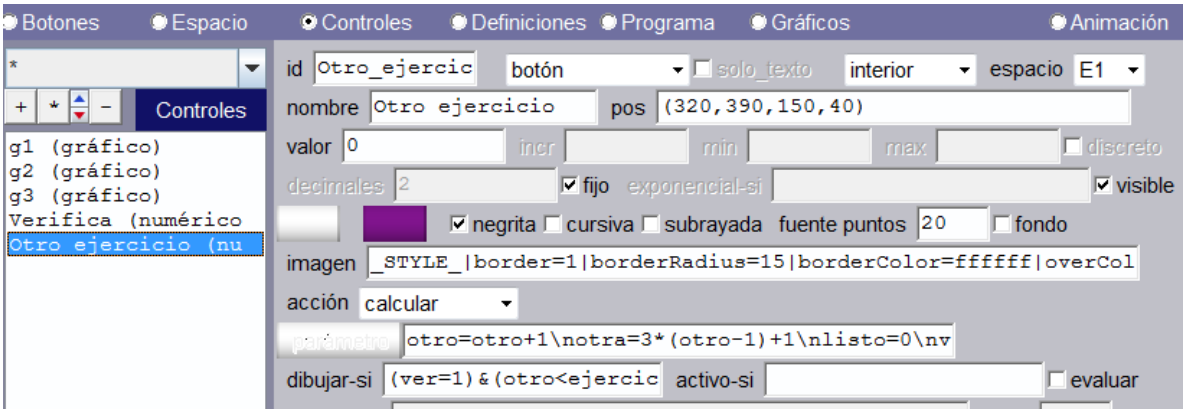
Puedes probar diferentes valores para los colores, el radio de los bordes del botón y tipo de fuente.

En la siguiente imagen, se observa cómo va nuestra actividad:

Como lo advertimos antes, una vez el usuario haga clic en este botón, los controles gráficos deben inactivarse, para ello, incluimos la expresión **ver = 0** en las casilla **activo-si**. Observa que cuando hacemos clic en el botón **Verificar** a la variable **ver** se le asigna el valor de uno (1), que tiene como primer efecto la inactivación de los controles gráficos,



Botón "Otro ejercicio". Su diseño es similar al anterior botón, por lo que nos detendremos sólo en las instrucciones de cálculo y en la casilla **dibujar-si**. En los parámetros iniciales el único cambio es el tamaño de fuente, que para este botón es de 20.



El botón debe aparecer una vez se haga clic en el botón **Verificar** y, obviamente, exista aún ejercicios por resolver, por ello, en la casilla **dibujar-si** debemos escribir la siguiente condición:

(ver=1)&(otro<ejercicios).

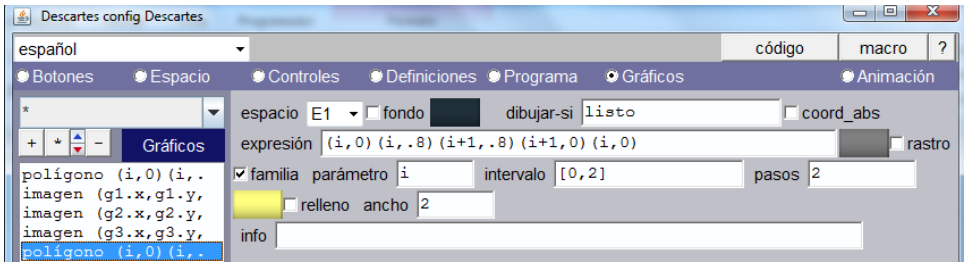
Cuando se hace clic en este botón, las instrucciones que se ejecutan son las siguientes:

- **otro=otro+1**. Aumenta en uno la variable **otro**, que responde a número de ejercicio en curso.
- **otra=3*(otro-1)+1**. Habíamos explicado que esta variable está asociada al número de la imagen a mostrar. Por ejemplo, si el ejercicio que se está desarrollando es el cuarto (otro = 4), la variable **otra** tomará el valor de 3*(4 – 1) + 1 = 10, la imagen asociada sería 10.png.
- **ver=0**. Al retornar a cero esta variable, se oculta el botón, pues no cumpliría la condición (ver=1)&(otro<ejercicios)
- **g1.y=-0.5**. Retorna la primera imagen a la parte inferior de la escena.
- **g2.y=-0.5**. Retorna la segunda imagen a la parte inferior de la escena.
- **g3.y=-0.5**. Retorna la tercera imagen a la parte inferior de la escena.
- **anima=1**. La variable anima la usamos para ejecutar una animación, que explicaremos al final de este instructivo.

Más polígonos

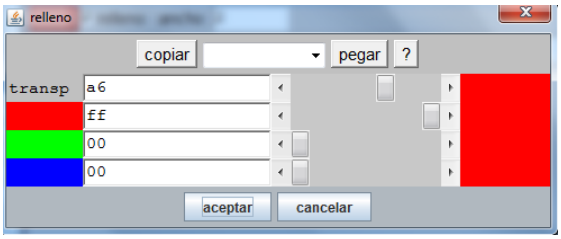
Dibujaremos dos polígonos adicionales, que deben estar después de las imágenes. Un polígono igual al primero pero sin relleno. Su utilidad es evitar que se pierdan los bordes de los cajones una vez estén colocadas las imágenes, como se muestra en la siguiente imagen.

El polígono se debe dibujar cuando las imágenes estén colocadas, es decir, cuando la variable **listo** es igual a uno (1).

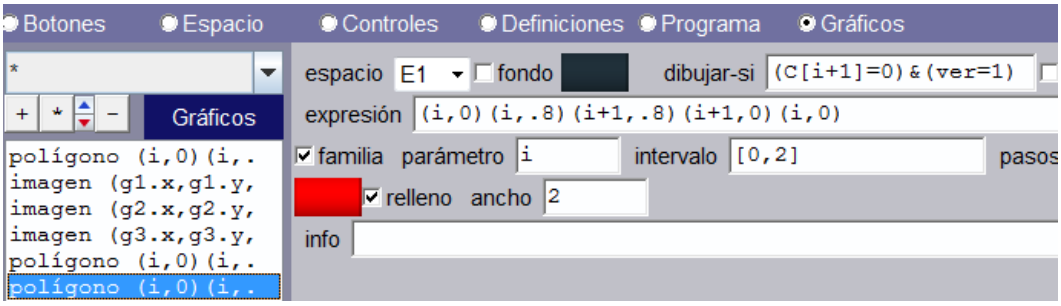


Nótese que en la casilla **dibujar-si** hemos puesto la variable listo sola, esta expresión, para Descartes, es equivalente a listo = 1. Observemos su efecto:

El otro polígono tiene como función dibujar cajones con relleno colorado para las imágenes mal puestas. Para ello, copiamos el polígono original, en el cual hacemos los siguientes cambios: el color de relleno es rojo con transparencia.



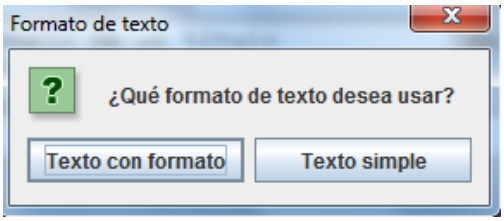
La condición para mostrar los cajones es (C[i+1]=0)&(ver=1), es decir, se muestra el cajón colorado cuando se hace clic en el botón verificar y el elemento del vector C es cero (pieza mal puesta).



Un ejemplo del efecto de este polígono se muestra en la siguiente imagen:

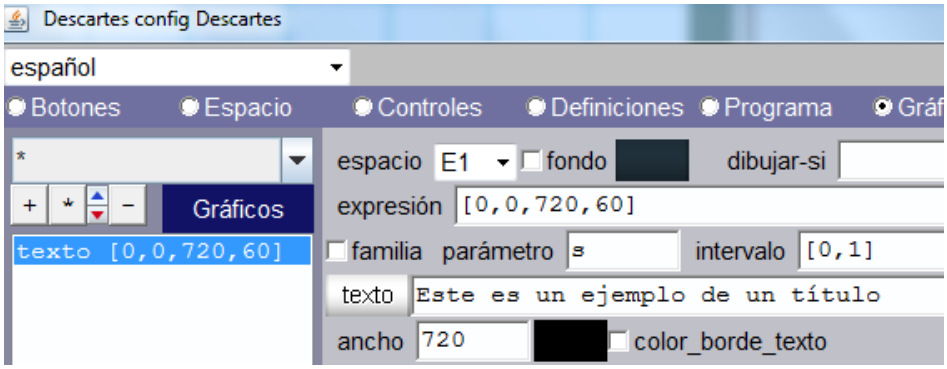
Textos

Ya estamos llegando al final de nuestra actividad. Los textos que usaremos tendrán algunas características especiales que debes considerar para los efectos finales de la actividad. En primer lugar, usaremos textos en formato simple, el cual basta con escribirlo en la casilla de texto o, si se prefiere, hacer clic en el botón **texto** y seleccionar **Texto simple.**



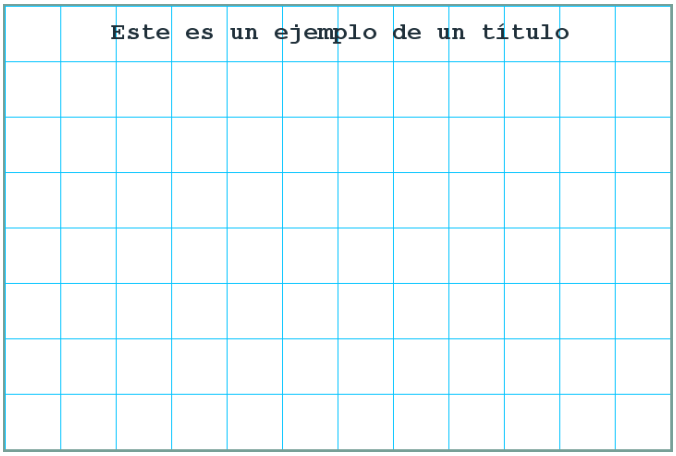
El tipo de fuente será Monospaced. Obviamente, puedes escoger Serif o SansSerif, pero se perderá el efecto que explicaremos más adelante.

Una última característica, en la cual nos detendremos para una explicación más amplia, es el centrado de textos en cajones. Tomamos como ejemplo de la explicación un espacio de trabajo de 720x480 pixeles y una escala de 60. El título escogido es “Este es un ejemplo de título”, el cual configuramos como lo muestra la siguiente imagen:

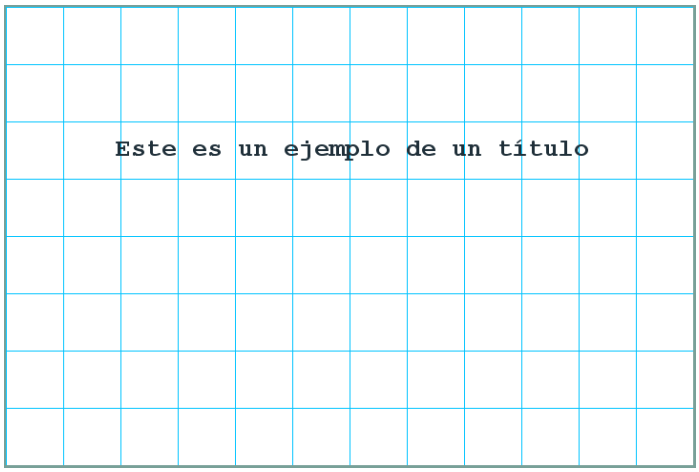


En la casilla texto hemos escrito el título en formato simple. La expresión [0, 0, 720, 60] tiene el siguiente significado: el texto se escribirá centrado en el cajón de

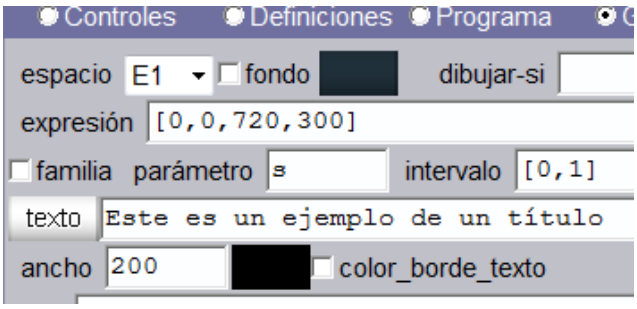
ancho 720 y alto 60, este cajón tiene su esquina superior izquierda en las coordenadas (0, 0). El resultado sería el siguiente:



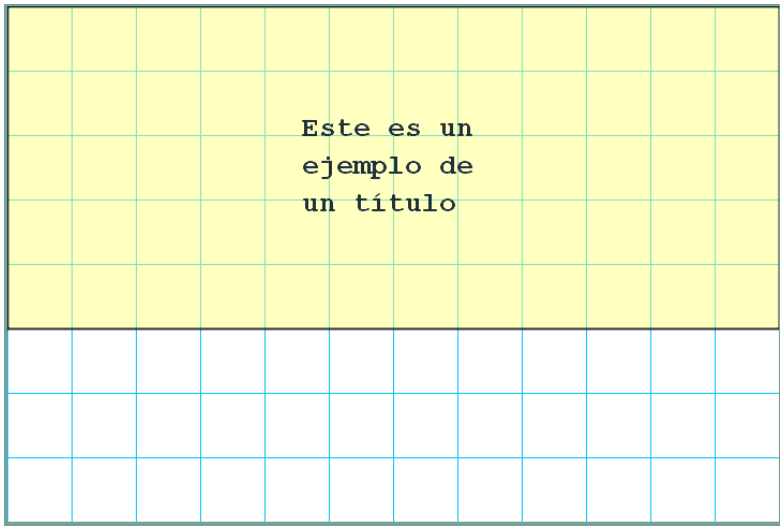
Recuerda que la escala es 60, por lo tanto cada cuadrícula es de tamaño 60x60. Si usamos esta expresión [0, 120, 720, 60], el texto estaría centrado en la tercera línea. Sí usamos [0, 0, 720, 300], el efecto sería el mismo, pues centraría el texto en un cajón de ancho 720 y alto 300.



Habrás notado que hay un valor de 720 en la casilla **ancho**, este valor no está asociado al ancho del cajón, corresponde al ancho del texto. Observa qué ocurre si reducimos este ancho a 200 y usamos la expresión anterior:

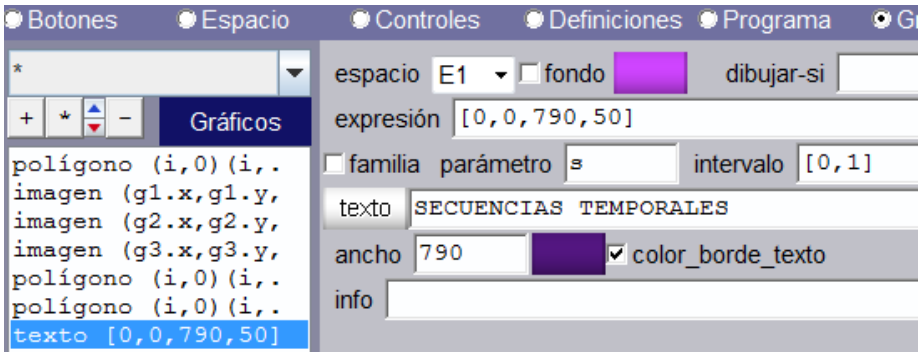


El texto de ancho 200, centrado en el cajón de 720x300 quedaría así (hemos resaltado en amarillo el cajón):

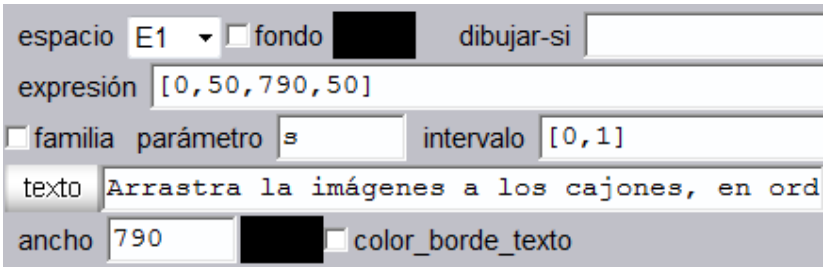


Luego de esta explicación, retornamos a nuestra actividad cuyas dimensiones son de 790x500 y escala 250.

Título de la actividad. Tal como aparece en la siguiente imagen. El alto del cajón lo hemos dejado en 50 para que el título quede cerca del borde superior. Incluimos, también, un borde para el texto.



Instructivo. Es el texto que instruye sobre lo que hay que hacer en la actividad.



Estos dos primeros textos los podemos observar en la siguiente imagen:

Cambio de fuente. Seguramente ya habrás notado que el texto correspondiente al botón de otro ejercicio, no se ajusta al ancho del botón, algo que se soluciona cambiando el ancho o el tipo de fuente. Una de las carpetas que acompañan esta actividad es una que hemos llamado **fonts**, la cual contiene varios tipos de fuente que usaremos para nuestros textos, para ello, debes incluir en el archivo indexb.html (usa un editor de texto sin formato), el siguiente vínculo:

<link rel="stylesheet" href="fonts/font1.css" type="text/css">

Lo puedes hacer luego de la etiqueta <title>, como se indica en la siguiente imagen:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<head>
<title>TITULO</title>
    <link rel="stylesheet" href="fonts/font1.css" type="text/css">
     <script type='text/javascript' src='lib/descartes-min.js'></script>
</head>
<body BGCOLOR="e0e4e8">

<ajs name="Descartes" width=790 height=500 code="Descartes">
    <param name="tamaño" value="790x500">
    <param name="image_loader" value="">
```

Nuestra actividad, para este tipo de fuente, tendrá la siguiente presentación:

En la carpeta fonts encontrarás 16 tipos de fuente, puedes practicar cambiando la fuente en el vínculo anterior. Algunos ejemplos, son:

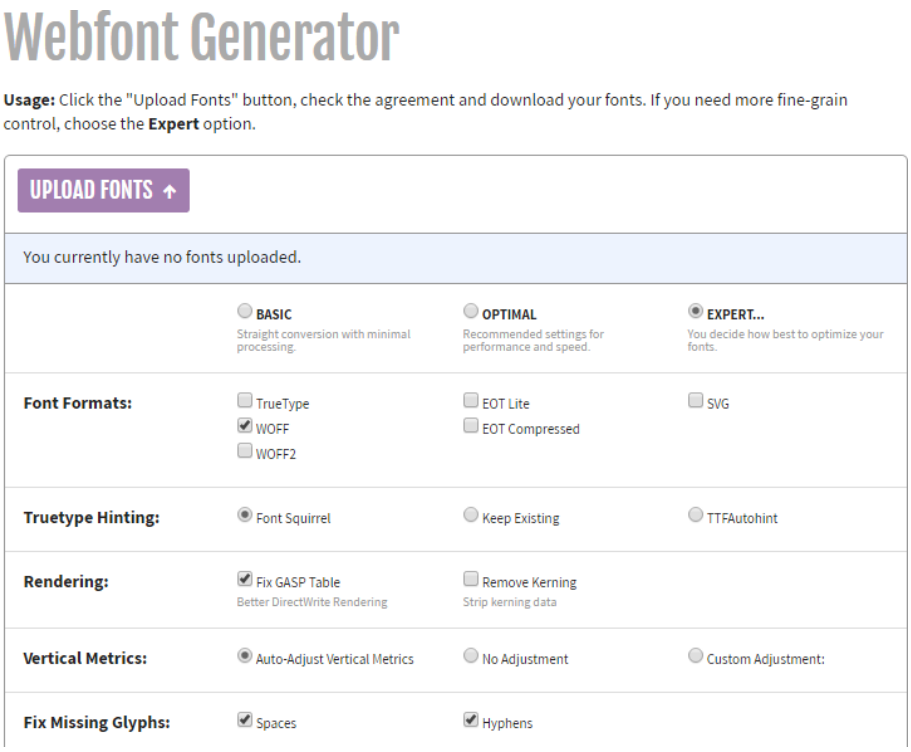
FUENTE 1 **Este texto está en tamaño 40** **Este texto está en tamaño 30, sin bordes y centrado** Este texto está en tamaño 30, centrado y con bordes	*FUENTE 3* *Este texto está en tamaño 40* *Este texto está en tamaño 30, sin bordes y centrado* *Este texto está en tamaño 30, centrado y con bordes*
FUENTE 10 ESTE TEXTO ESTÁ EN TAMAÑO 28 ESTE TEXTO ESTÁ EN TAMAÑO 30, SIN BORDES Y CENTRADO ESTE TEXTO ESTÁ EN TAMAÑO 30, CENTRADO Y CON BORDES	FUENTE 11 Este texto está en tamaño 28 Este texto está en tamaño 30, sin bordes y centrado Este texto está en tamaño 30, centrado y con bordes
FUENTE 13 Este texto está en tamaño 28 Este texto está en tamaño 30, sin bordes y centrado Este texto está en tamaño 30, centrado y con bordes	*FUENTE 14* *Este texto está en tamaño 28* *Este texto está en tamaño 30, sin bordes y centrado* *Este texto está en tamaño 30, centrado y con bordes*

Si deseas generar otros tipos de fuente, te presentamos un procedimiento sencillo para obtenerlas:

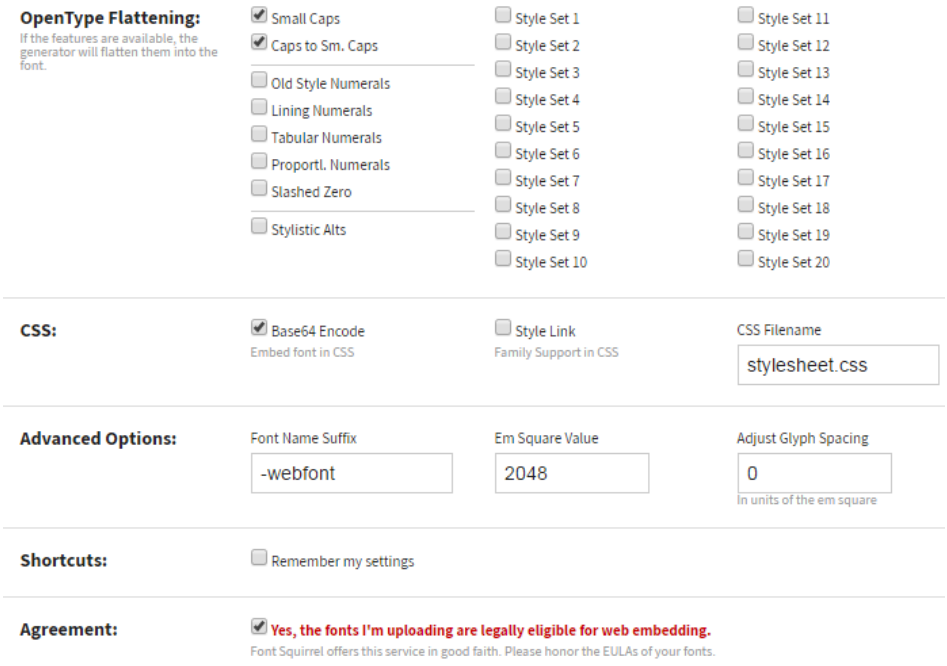
Encontrando fuentes. En primer lugar, accede a una página que te permita descargar las fuentes en tipografía tipo *True Type Font* (ttf) u *Open Type Font* (otf), una de ellas es https://www.fontsquirrel.com/fonts/list/popular

Una vez descargada la fuente debemos generar un código **base64**, el cual se puede lograr desde la página anterior desde la opción **Generator**: https://www.fontsquirrel.com/tools/webfont-generator

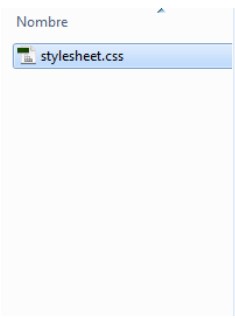
Los pasos son simples, subes la fuente (Upload fontes), seleccionas EXPERT, formato WOFF.



Es importante activar la opción **Base64**



Una vez descargada la fuente (*Download your kit*), abrimos el archivo **stylesheet.css** y copiamos todo el código que hay después de **src**. Este código lo pegamos en font1 o font2 o… en una nueva fuente y… eso es todo

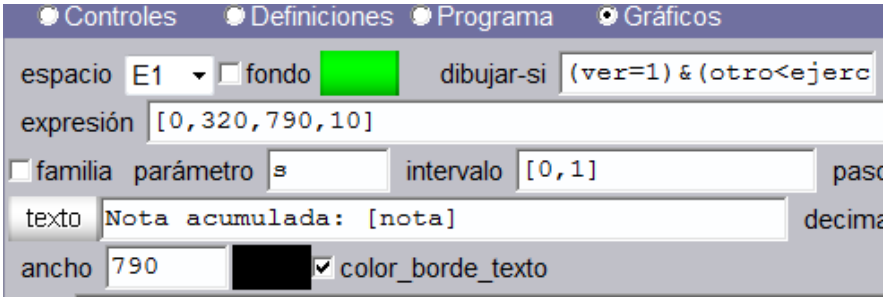


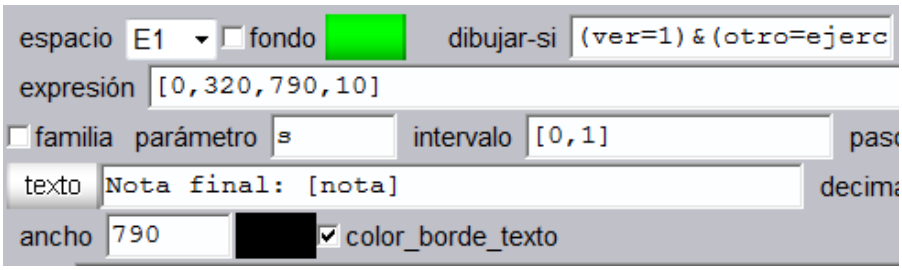

```
/* Generated by Font Squirrel
(https://www.fontsquirrel.com) on July 24, 2016 */



@font-face {
    font-family: 'ceviche_oneregular';
    src: url(data:application/font-woff;charset=utf-
8;base64,d09GRgABAAAAAIE8ABMAAAAA3xgAAQAAAAAAAAAAAAAAAAAA
AAAAAAAAAABGRlRNAAABqAAAABwAAAAcay3LeEdERUYAAAHEAAAAHgAAA
CABFQAER1BPUwAAAeQAAA41AAAZdJhYdClHU1VCAAAQHAAAACwAAAAwuP
```

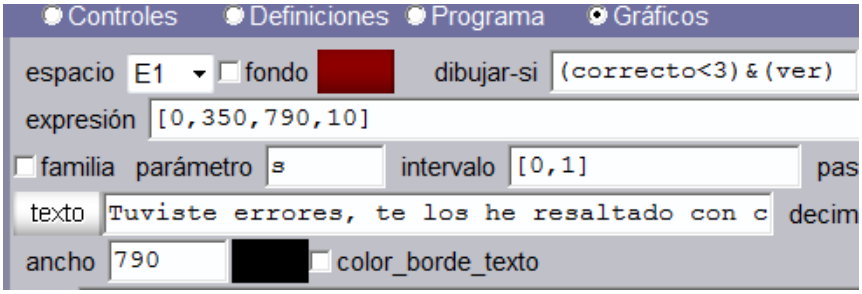
Calificaciones. En dos textos presentaremos la nota acumulada y la nota final, así:





La diferencia entre uno y otro es la condición, para el primero se presentará una nota acumulada mientras aún falten ejercicios por realizar, el segundo texto presenta la nota final al no haber más ejercicios por realizar.

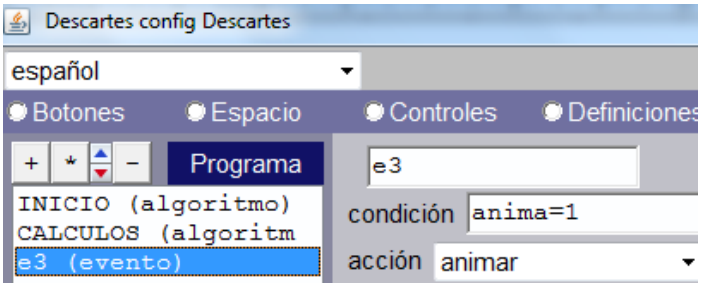
Mensaje de error. Texto que se muestra cuando hay piezas mal colocadas.



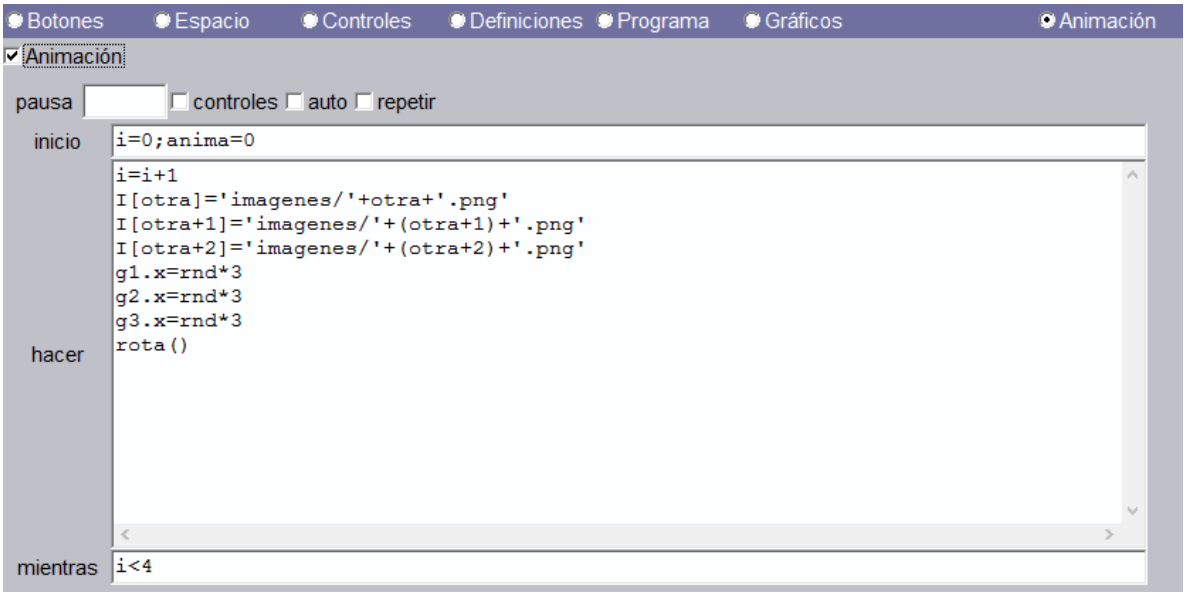
Estos son algunos de los textos

Animación

Finalmente, incluimos una animación, la cual se invoca a través de un evento, cuya condición es que la variable **anima** sea uno (1). Esto siempre ocurrirá cuando hacemos clic en el botón **Otro ejercicio.**



La animación es la que se muestra en la siguiente imagen:



Se trata de un algoritmo tipo do-while que se ejecuta cuatro veces. En cada ejecución se generan posiciones horizontales y aleatorias de los controles gráficos, se invoca, también, la función rota(). Estas ejecuciones tiene como efecto una animación en las imágenes cada vez que iniciamos un nuevo ejercicio.

Los elementos del vector I incorporados en la animación sólo tiene como función el forzamiento al visionado de las imágenes, en tanto que Descartes presenta algunos problemas de presentación de imágenes en el navegador. Observa, además, que se asigna cero a la variable **anima**, de tal forma que se pueda ejecutar la animación cuando el valor vuelva a ser uno (1).

¡ESO ES TODO!