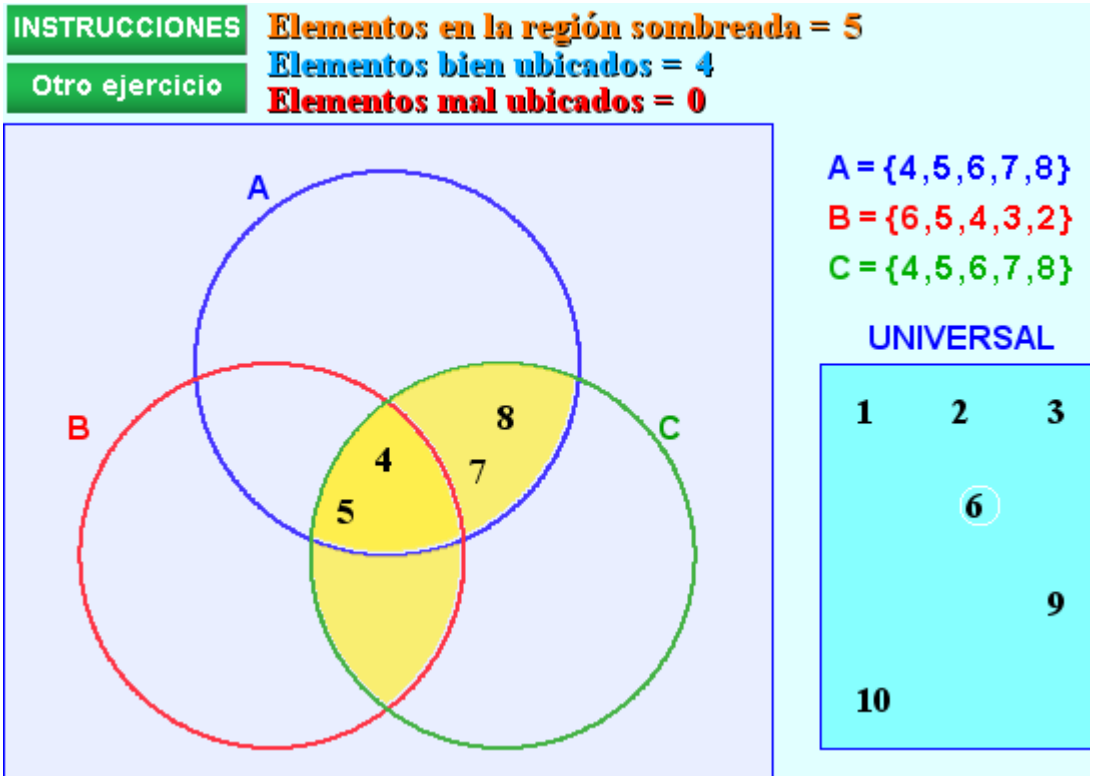
CLASE 19
ELEMENTOS MÍNIMOS DE PROGRAMACIÓN
DE REGRESO A LOS VECTORES

En la clase 12 trabajamos una actividad muy simple con el uso de los vectores de Descartes. En esta clase retomaremos esta utilidad para construir una escena de arrastre. Podrás repasar, con esta actividad, otros conceptos antes trabajados como: controles gráficos, números aleatorios, algoritmos, ecuaciones, rellenos y condicionales. Es decir, se trata de una actividad que usa al máximo las utilidades de Descartes.

Para el logro de nuestros objetivos hemos seleccionado una escena similar a la última actividad de la clase 15, diagramas de Venn. Esta selección no implica un acuerdo con este tipo de diagramas, al cual ya le hicimos una crítica en la clase 15. Se trata de aprovechar la complejidad del tipo de diagrama para explicar con más detalle los conceptos antes señalados.

Actividad 1. Diseñar una escena que visualice un diagrama de Venn para tres conjuntos con las siguientes características: deben aparecer aleatoriamente regiones sombreadas (ver nota sobre combinaciones estadísticas), se deben generar y visualizar tres conjuntos, cada uno con cinco números aleatorios del 1 al 10, debe aparecer el universal (números del 1 al 10) de tal forma que se puedan arrastrar su elementos a las regiones sombreadas, la escena debe detectar si el elemento arrastrado está bien ubicado.

La escena a diseñar deber ser similar a la mostrada en la siguiente imagen:



La ventaja de escenas como la propuesta, es la posibilidad de representar cientos de ejercicios en una sola actividad. Como habrás notado, varias de las actividades de las clases anteriores tienen esta característica. La razón es la posibilidad de un mayor número de ejercicios, evitando así el abandono por falta de material para trabajar los temas de forma continuada. Es decir, la construcción de escenas interactivas debe ser de tal forma que el tiempo del estudiante se agote en la práctica y no al contrario.

En la actividad que proponemos, se pueden generar 256 regiones sombreadas diferentes, que a su vez son multiplicadas por todas las posibles combinaciones de los tres conjuntos. La primera afirmación se puede demostrar estadísticamente así:

Número posible regiones sombreadas = 0 regiones + 1 región + 2 regiones +… + 8 regiones.

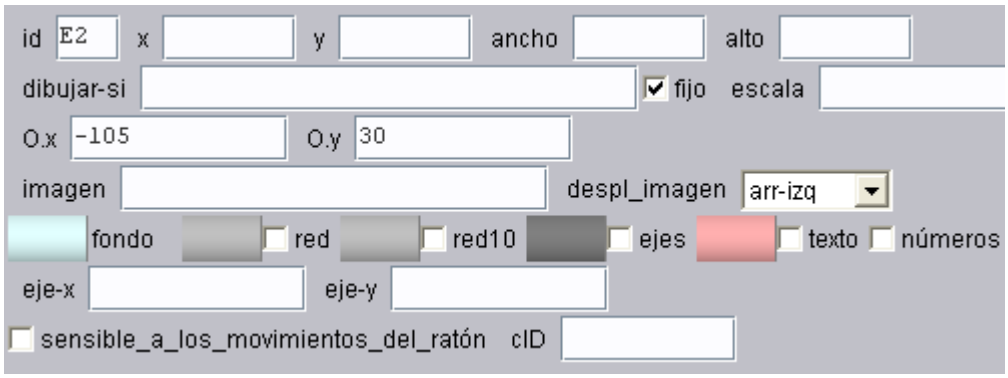
Este número de regiones posibles se obtiene así:

$$\binom{8}{0}+\binom{8}{1}+\binom{8}{2}+\binom{8}{3}+\binom{8}{4}+\binom{8}{5}+\binom{8}{6}+\binom{8}{7}+\binom{8}{8}$$

$$= 1 + 8 + 28 + 56 + 70 + 56 + 28 + 8 + 1 = 256$$

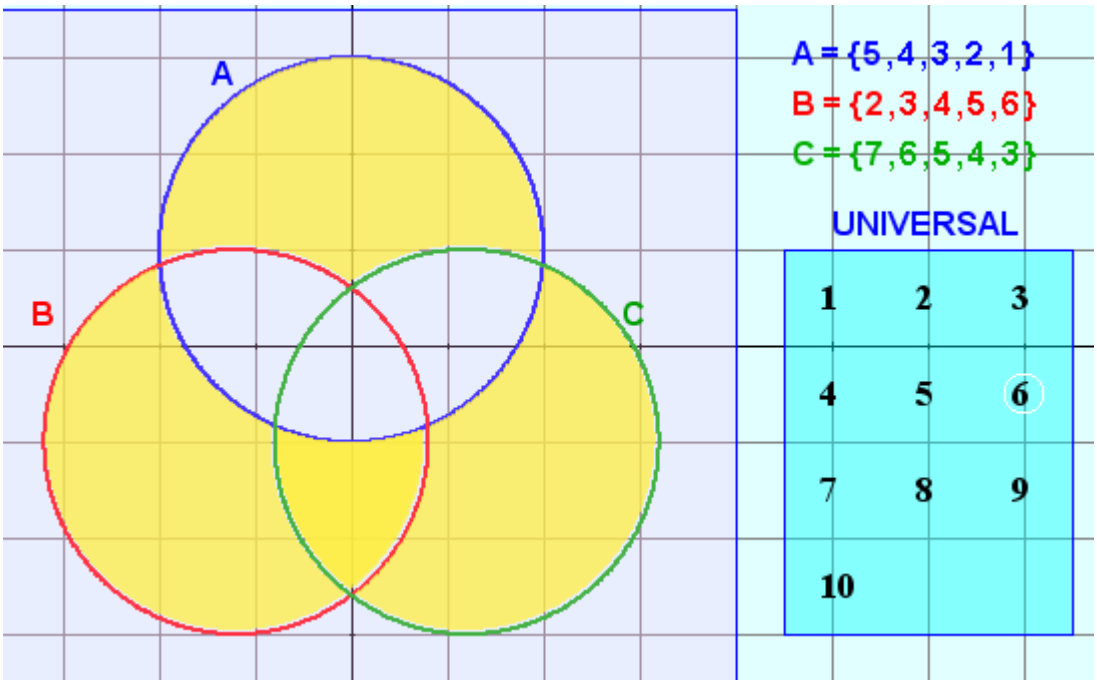
Veamos, entonces, como lograr nuestra escena.

1.1 **Espacio.** Crea un espacio 2D tal como se muestra en la imagen siguiente. El color del fondo queda a tu criterio, se sugiere colores claros.



1.2 **Controles.** Usaremos diez controles gráficos correspondientes a los elementos del Universal y dos controles numéricos tipo botón (ver la imagen anterior). En las clases anteriores hemos seguido un orden de construcción de escenas de la siguiente forma: espacio, controles, auxiliares y gráficos. Esto no significa que nuestras escenas se deban construir en ese sentido estricto. Para el caso de los diez controles de esta escena comprenderás esta última afirmación.

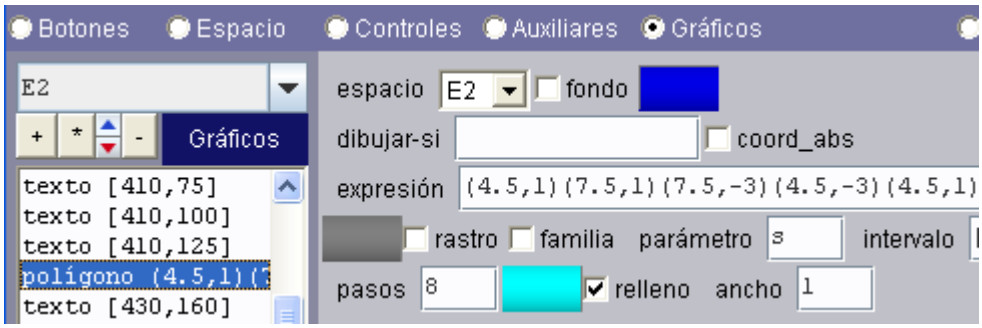
Controles gráficos. Estos controles serán ubicados en el recuadro azul que se observa en la siguiente imagen:



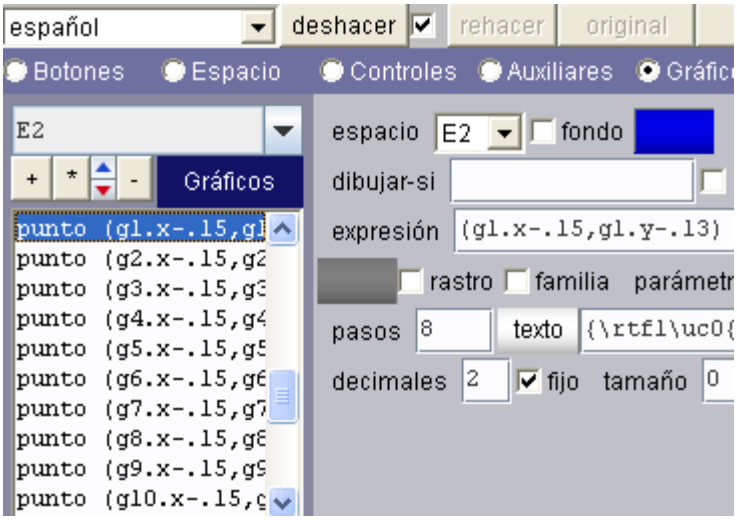
Hemos habilitado la cuadrícula para comprender la situación de dichos controles, observa que el origen coordenadas se encuentra en la intersección de los tres conjuntos. El control **1**, en este ejemplo, está ubicado en las coordenadas (5, 0.5); es decir, g1.x = 5, g1.y = 0.5. Añade, entonces, los controles gráficos de acuerdo a la tabla siguiente, asígnales un **tamaño de 12 y color transparente** (para que no se vea el control).

Control gn	gn.x	gn.y
g1	5	0.5
g2	6	0.5
g3	7	0.5
g4	5	-0.5
g5	6	-0.5
g6	7	-0.5
g7	5	-1.5
g8	6	-1.5
g9	7	-1.5
g10	5	-2.5

Para verificar esta primera parte de la escena, inserta un polígono con características dadas en la siguiente imagen. Comprueba comparando las coordenadas de los vértices del polígono con la imagen anterior. El color de relleno queda a tu elección.



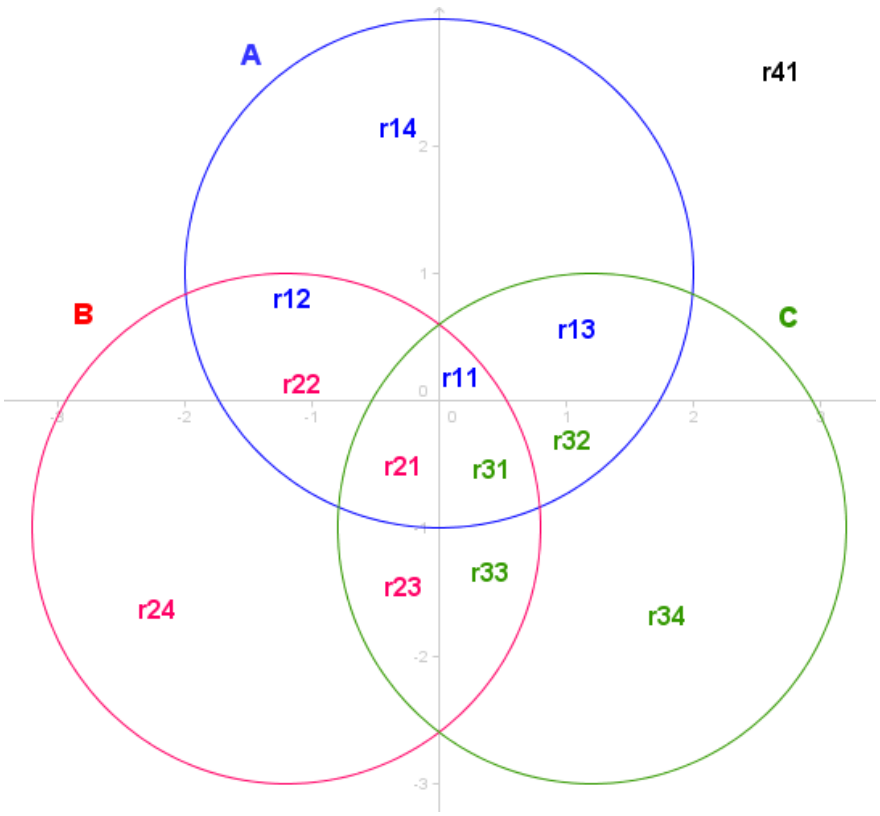
Inserta también 10 puntos (en gráficos) que permitan visualizar los 10 números que constituyen el conjunto Universal. Las coordenadas de cada punto son las correspondientes al control gráfico restándoles unos decimales de la unidad, se busca el efecto de situar mejor dichos números (practica con las coordenadas del control gráfico sin variaciones para verificar su correcto funcionamiento). Por ejemplo, para el control gráfico **g7,** la expresión que define las coordenadas del punto sería: **(g7.x-.15, g1.y-.13).** A cada punto se le asigna el texto correspondiente; es decir, para el caso anterior el texto es **7**. Observa en la imagen el resto de puntos.



Controles numéricos. El primer control numérico tipo botón estará ubicado al interior de la escena en las coordenadas (5, 32, 120, 25), su nombre es **Otro ejercicio** y la acción puede ser **inicio** o, si lo prefieres, **calcular** añadiendo en parámetro todas las constantes y variables que reinician la escena. El segundo botón estará situado en el interior con posición y tamaño definido por (5, 3, 120, 25), su nombre será **Instrucciones**, la acción es **mensaje** descrito así:

> *Tienes tres conjuntos (A, B y C) con cinco elementos cada uno. Al azar aparece una región sombreada en el diagrama de Venn. Debes trasladar (con clic sostenido) los elementos del conjunto Universal que pertenecen a esta región.*
>
> *Una vez lo hagas aparecerá el mensaje de logro. Puedes practicar con otro ejercicio haciendo clic en el botón que tiene el texto* ***Otro ejercicio****.*

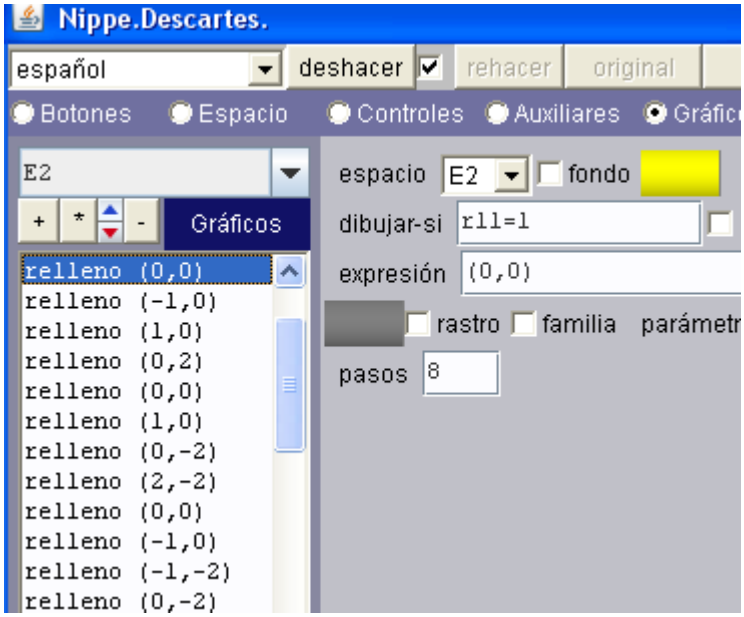
1.3 **Auxiliares.** Usaremos 16 constantes, 8 vectores y 8 algoritmos. Para este ejercicio, las constantes serán la clave, en tanto que representan las 16 subregiones del diagrama de Venn. Observa estas regiones en la siguiente imagen.



Constantes y rellenos. Inserta las siguientes 13 constantes: r11, r12, r13, r14, r21, r22, r23, r24, r31, r32, r33, r34 y r41. Cada una tendrá el siguiente valor **ent(2*rnd).** Esto significa que tomarán valores entre cero y uno, útil para activar o desactivar la región que define cada constante (ver imagen anterior).

Para añadir los rellenos, primero lo haremos con las circunferencias que se ven en la imagen anterior, así podremos verificar el funcionamiento de los rellenos. Inserta las siguientes ecuaciones:

x^2 + (y - 1)^2 = 4
(x + 1.2)^2 + (y + 1)^2 = 4
(x - 1.2)^2 + (y + 1)^2 = 4



Ahora podremos insertar igual número de rellenos que el de las constantes anteriores, que serán dibujados siempre que la condición **dibujar-si** corresponda a r11 = 1 para el primer relleno (ver imagen siguiente), r12 = 1 para el segundo, y así sucesivamente.

Añade, entonces estos rellenos y haz clic en **aplicar** para verificar el funcionamiento de la escena hasta este punto (observa las coordenadas asignadas a los rellenos y confróntalas con la imagen anterior).

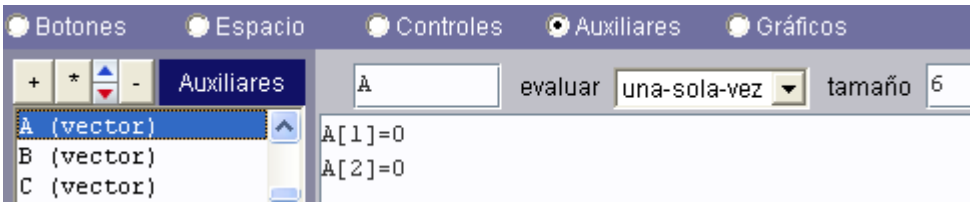
Última constantes. Para completar las 16 constantes, agrega las siguientes:

A1 = ent(rnd*10)+1, B1 = ent(rnd*10)+1 y **C1 = ent(rnd*10)+1**

Estas últimas constantes nos generarán el primer elemento de cada conjunto. En el primer algoritmo comprenderás esta afirmación.

Vectores. Definiremos (añadiremos) los siguientes vectores:

Vectores para los conjuntos. Añade los vectores A, B y C de tamaño 6 como se indica en la imagen. En realidad sólo necesitamos un tamaño de cinco, sin embargo para no usar el elemento **cero** del vector, lo dimensionamos con una unidad más.



Vectores región. Inserta los vectores R1, R2 y R3 con tamaño 11. Estos vectores nos permitirán determinar las regiones a las cuales pertenecen los elementos de los conjuntos A, B, C y el Universal. Igualmente los usaremos para determinar a qué región el usuario **arrastra** los elementos del Universal. Con el análisis de los algoritmos, comprenderás mejor lo anterior.

Vectores posición. Finalmente, agrega dos vectores **X** e **Y** con tamaño 11. Estos vectores almacenarán la posición de los controles gráficos.

Algoritmos. Una de las grandes utilidades de Descartes son sus algoritmos. A partir de ellos podemos facilitar la creación y recreación del conocimiento que deseamos compartir con nuestro alumnado. Esta actividad te permitirá acercarte más a la lógica de programación que permite la herramienta Descartes. Analiza, entonces, con mucho cuidado cada uno de los algoritmos que se presentan a continuación y agrégalos a la escena.

Algoritmo para generar los elementos de A, B y C

Inicio. En este campo incluye las siguientes asignaciones:

n=1; A[1]=A1; B[1]=B1; C[1]=C1

Observa que al primer elemento de cada conjunto se le asigna una de las tres últimas constantes añadidas anteriormente, las cuales son un número aleatorio entre 1 y 10. Recuerda que **n** es un control para indicar cuantas veces se ejecuta el algoritmo.

Hacer. En este campo escribe las siguientes proposiciones.

n=n+1

A[n]=(A1<5)?A[n-1]+1:A[n-1]-1

B[n]=(B1<5)?B[n-1]+1:B[n-1]-1

C[n]=(C1<5)?C[n-1]+1:C[n-1]-1

Mientras. En este campo escribiremos **n<5.** Esto significa que el algoritmo se ejecutará hasta que n sea igual a 5.

Analicemos paso a paso el algoritmo para que entiendas su lógica, este seguimiento suelen llamarlo los programadores una **prueba de escritorio**. Lo haremos sólo para el vector A (conjunto A), los demás vectores se comportan igual.

Supondremos dos casos, uno con A1 = 2 y otro con A1 = 7.

Caso 1. A1 = 2

En el campo inicio se asigna los siguientes valores n = 1 y A[1] = 2 (ver las asignaciones que definimos). Ahora vamos al **hacer.**

La proposición **n = n + 1** es todo un adefesio para los matemáticos que están leyendo este tutorial. Pero, antes de que sigan protestando, les aclaramos que no se trata de una igualdad sino de una asignación. La simbología correcta sería:

n ← n + 1

y su significado es que a **n** se le suma **1** y el resultado se le asigna a **n.** En este caso funciona como un **contador.**

Lenguajes de programación como el antiguo Pascal usan asignaciones como **n := n + 1.** El lenguaje C prefiere usar **n += 1** equivalente a nuestra asignación. No obstante, la mayoría de lenguajes, incluidos el C, aceptan **n = n + 1** como una asignación y no una igualdad.

Es de aclarar que no se requiere ser un experto en programación para continuar con esta actividad, sólo se trata de proposiciones o sentencias sencillas que sólo demandan un poco de atención.

Continuemos pues.

La primera acción del **mientras** es asignar a **n** el mismo valor de **n** sumado en **1.** Es decir, el resultado de la proposición es **n = 2.**

Ahora viene el condicional

A[n]=(A1<5)?A[n-1]+1:A[n-1]-1

Veamos con detalle esta proposición. A[n] toma valores iguales a alguna de las sentencias que están después del signo "**?"** y separadas por el signo ":**".** Si A1<5, se ejecuta la proposición A[n-1]+1, en caso contrario se ejecuta la sentencia A[n-1]-1.

Para este caso A1 < 5, entonces a A[n] o, mejor, a A[2] se le asigna el valor de A[n-1]+1, es decir el valor de A[1]+1. Lo cual se reduce en A[2] = 3.

Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 3
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[3] = 4

Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 4
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[4] = 5

Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 5
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[5] = 6

Como **n** no es menor que cinco, termina el **hacer mientras** y hemos obtenido el conjunto **A = {2, 3, 4, 5, 6}.**

Caso 2. A1 = 7

En el campo inicio se asigna los siguientes valores n = 1 y A[1] = 7 (ver las asignaciones que definimos). Ahora vamos al **hacer.**

La primera acción del **mientras** es asignar a **n** el mismo valor de **n** sumado en **1.** Es decir, el resultado de la proposición es **n = 2** y A[2]=(A1<5)?A[n-1]+1:A[n-1]-1, lo cual se reduce a A[2] = A[1]-1, o lo mismo que A[1] = 6.

Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 3
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[3] = 5

Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 4
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[4] = 4

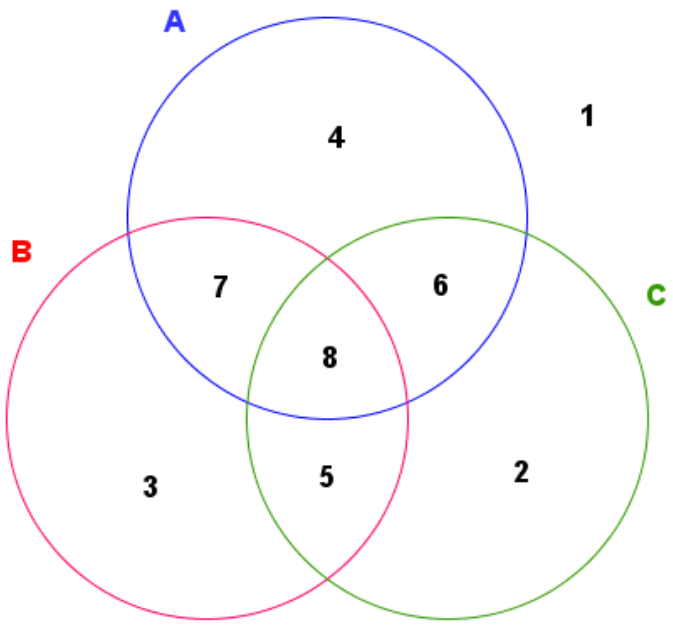
Como **n** es aún menor que cinco, se repite el **hacer** así:
n = n + 1 → n = 5
A[n]=(A1<5)?A[n-1]+1:A[n-1]-1 → A[5] = 3

Como **n** no es menor que cinco, termina el **hacer mientras** y hemos obtenido el conjunto **A = {7, 6, 5, 4, 3}**

Algoritmo identificador de regiones para los elementos de A, B y C.

Antes de iniciar el algoritmo, es importante que comprendas la asignación de las regiones que se observan en la imagen siguiente que corresponde lógicamente a las siguientes situaciones, considerando **x** como **pertenece a A (x∈A)**, **y** como **pertenece a B (y∈B)** y **z** como **pertenece a C (z∈C)**:

Región 1: (x=0)^(y=0)^(z=0)
Región 2: (x=0)^(y=0)^(z=1)
Región 3: (x=0)^(y=1)^(z=0)
Región 4: (x=1)^(y=0)^(z=0)
Región 5: (x=0)^(y=1)^(z=1)
Región 6: (x=1)^(y=0)^(z=1)
Región 7: (x=1)^(y=1)^(z=0)
Región 8: (x=1)^(y=1)^(z=1)



Ahora, vamos al algoritmo.

Crea un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

n=0;x=0;y=0;z=0

Hacer. En este campo escribe las siguientes proposiciones:

n=n+1

x=(A[1]=n)+(A[2]=n)+(A[3]=n)+(A[4]=n)+(A[5]=n)
y=(B[1]=n)+(B[2]=n)+(B[3]=n)+(B[4]=n)+(B[5]=n)
z=(C[1]=n)+(C[2]=n)+(C[3]=n)+(C[4]=n)+(C[5]=n)
R1[n]=((x=1)&(y=1)&(z=1))?8:R1[n]
R1[n]=((x=1)&(y=1)&(z=0))?7:R1[n]
R1[n]=((x=1)&(y=0)&(z=1))?6:R1[n]
R1[n]=((x=0)&(y=1)&(z=1))?5:R1[n]
R1[n]=((x=1)&(y=0)&(z=0))?4:R1[n]
R1[n]=((x=0)&(y=1)&(z=0))?3:R1[n]
R1[n]=((x=0)&(y=0)&(z=1))?2:R1[n]
R1[n]=((x=0)&(y=0)&(z=0))?1:R1[n]

Mientras. En este campo escribiremos **n<10.** Esto significa que el algoritmo se ejecutará hasta que n sea igual a 10.

Para comprender la lógica del algoritmo, haremos otra prueba de escritorio.

Supongamos que A={1, 2, 3, 4, 5}, B={3, 4, 5, 6, 7} y C={10, 9, 8, 7, 6}. Supongamos, además, que el ciclo **mientras** ya va en **n = 3**.

Entonces las proposiciones:

x=(A[1]=n)+(A[2]=n)+(A[3]=n)+(A[4]=n)+(A[5]=n)
y=(B[1]=n)+(B[2]=n)+(B[3]=n)+(B[4]=n)+(B[5]=n)
z=(C[1]=n)+(C[2]=n)+(C[3]=n)+(C[4]=n)+(C[5]=n)

son equivalentes a:

x=(A[1]=3)+(A[2]=3)+(A[3]=3)+(A[4]=3)+(A[5]=3)
y=(B[1]=n)+(B[2]=3)+(B[3]=3)+(B[4]=3)+(B[5]=3)
z=(C[1]=n)+(C[2]=3)+(C[3]=3)+(C[4]=3)+(C[5]=3)

Las igualdades entre paréntesis son lógicas; es decir, asigna **1** si es **verdadera** o **0** si es **falsa**. En ese sentido, las expresiones anteriores se reducen a:

x = 0 + 0 + 1 + 0 + 0 = 1
y = 1 + 0 + 0 + 0 + 0 = 1
z = 0 + 0 + 0 + 0 + 0 = 0

Finalmente, las expresiones:

R1[n]=((x=1)&(y=1)&(z=1))?8:R1[n]
R1[n]=((x=1)&(y=1)&(z=0))?7:R1[n]
R1[n]=((x=1)&(y=0)&(z=1))?6:R1[n]
R1[n]=((x=0)&(y=1)&(z=1))?5:R1[n]
R1[n]=((x=1)&(y=0)&(z=0))?4:R1[n]
R1[n]=((x=0)&(y=1)&(z=0))?3:R1[n]
R1[n]=((x=0)&(y=0)&(z=1))?2:R1[n]
R1[n]=((x=0)&(y=0)&(z=0))?1:R1[n]

son equivalentes a:

R1[3]=((x=1)&(y=1)&(z=1))?8:R1[3]
R1[3]=((x=1)&(y=1)&(z=0))?7:R1[3]
R1[3]=((x=1)&(y=0)&(z=1))?6:R1[3]
R1[3]=((x=0)&(y=1)&(z=1))?5:R1[3]
R1[3]=((x=1)&(y=0)&(z=0))?4:R1[3]
R1[3]=((x=0)&(y=1)&(z=0))?3:R1[3]
R1[3]=((x=0)&(y=0)&(z=1))?2:R1[3]
R1[3]=((x=0)&(y=0)&(z=0))?1:R1[3]

De acuerdo a los valores de **x**, **y** y **z** calculados, la única sentencia que asigna una región es R1[3]=((x=1)&(y=1)&(z=0))?7:R1[3]. Es decir, R1[3] = 7.

Lo anterior significa que el elemento **3** pertenece a la región **7** como era de esperarse. Puedes hacer más pruebas de escritorio para comprender mejor el algoritmo. El vector R1, entonces, almacena la región a la cual pertenece cada uno de los elementos del Universal.

Algoritmo identificador de regiones sombreadas.

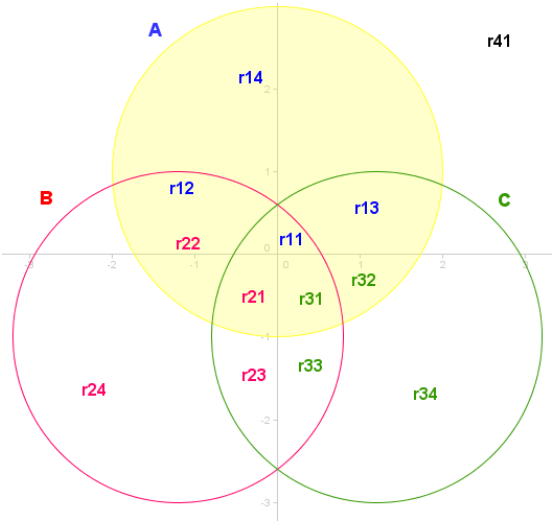
Crea un algoritmo con las siguientes características:

Inicio. En este campo incluye la siguiente asignación:

n=0

Hacer. En este campo escribe las siguientes proposiciones o sentencias:

n=n+1
R2[8]=((r11=1)|(r21=1)|(r31=1))?1:0
R2[7]=((r12=1)|(r22=1))?1:0
R2[6]=((r13=1)|(r32=1))?1:0
R2[5]=((r23=1)|(r33=1))?1:0
R2[4]=(r14=1)?1:0
R2[3]=(r24=1)?1:0
R2[2]=(r34=1)?1:0
R2[1]=(r41=1)?1:0



Mientras. En este campo escribiremos **n<1.** Esto significa que el algoritmo se ejecutará una sola vez, cuando **n** toma el valor de 1.

Si observas las sentencias te darás cuenta que el vector R2 almacena las regiones sombreadas; es decir, aquellas donde **rij** es igual a 1**.** Por ejemplo, la imagen de arriba nos permite inferir que:

R2[8] = 1 ó dicho de otra forma, la región 8 está sombreada. Esto es cierto porque la proposición lógica (r11=1)v(r21=1)v(r31=1) es verdadera (recuerda que el símbolo ”|” representa, en Descartes, el conector lógico de la disyunción).

Algoritmo que almacena las posiciones de los controles gráficos.

Crea un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

n=0

Hacer. En este campo escribe las siguientes proposiciones:

```
n=n+1
X[1]=g1.x
Y[1]=g1.y
X[2]=g2.x
Y[2]=g2.y
X[3]=g3.x
Y[3]=g3.y
X[4]=g4.x
Y[4]=g4.y
X[5]=g5.x
Y[5]=g5.y
X[6]=g6.x
Y[6]=g6.y
X[7]=g7.x
Y[7]=g7.y
X[8]=g8.x
Y[8]=g8.y
X[9]=g9.x
Y[9]=g9.y
X[10]=g10.x
Y[10]=g10.y
```

Mientras. En este campo escribiremos **n<1.** Esto significa que el algoritmo se ejecutará una sola vez, cuando **n** toma el valor de 1.

Como puedes observar, este algoritmo se explica por si solo.

Algoritmo que almacena las regiones donde están los controles gráficos.

Crea un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

```
n=0; e=0; f=0; g=0; h=0; i=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; s7=0; s8=0
```

Hacer. En este campo escribe las siguientes proposiciones:

```
n=n+1
e=X[n]^2+(Y[n]-1)^2
f=(X[n]+1.2)^2+(Y[n]+1)^2
g=(X[n]-1.2)^2+(Y[n]+1)^2
h=X[n]
i=Y[n]
s8=(e<4)+(f<4)+(g<4)
s7=(e<4)+(f<4)+(g>4)
s6=(e<4)+(f>4)+(g<4)
```

s5=(e>4)+(f<4)+(g<4)
s4=(e<4)+(f>4)+(g>4)
s3=(e>4)+(f<4)+(g>4)
s2=(e>4)+(f>4)+(g<4)
s1=(e>4)+(f>4)+(g>4)+(h<4)+(i<3.5)
R3[n]=0
R3[n]=(s8=3)?8:R3[n]
R3[n]=(s7=3)?7:R3[n]
R3[n]=(s6=3)?6:R3[n]
R3[n]=(s5=3)?5:R3[n]
R3[n]=(s4=3)?4:R3[n]
R3[n]=(s3=3)?3:R3[n]
R3[n]=(s2=3)?2:R3[n]
R3[n]=(s1=5)?1:R3[n]

Mientras. En este campo escribiremos **n<10.** Esto significa que el algoritmo se ejecutará 10 veces, permitiendo la asignación para cada elemento del vector R3.

El análisis del algoritmo es sencillo:

Las variables e, f, g son calculadas con las ecuaciones de las circunferencias que definen el diagrama de Venn, esto nos permite determinar si un control gráfico está fuera o dentro de las circunferencias. Las variables h e i nos permitan determinar si el control está por fuera del rectángulo que contiene los diagramas.

Las demás expresiones las dejo para tu análisis. Si quieres usa una prueba de escritorio con valores supuestos para un control gráfico cualquiera.

Algoritmo calculador del número de elementos en zona sombreada

Una vez hemos asignado los valores de los vectores anteriores, los algoritmos que siguen son de comparación para calcular el número de elementos dentro de la zona sombreada, el número de elementos bien ubicados por el usuario y los errados.
Agrega un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

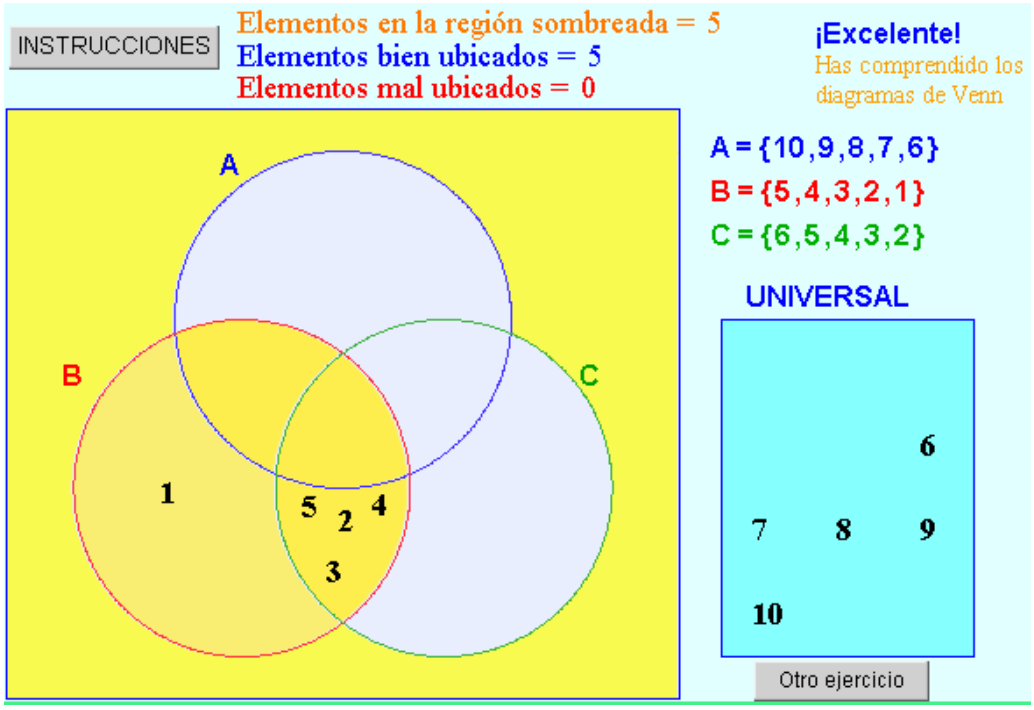
n=0;k=0;elementos=0

Hacer. En este campo escribe las siguientes proposiciones

n= n+1
k=(R2[R1[n]]=1)?1:0
elementos = elementos + k

Mientras. En este campo escribiremos **n<10.** Esto significa que el algoritmo se ejecutará 10 veces, permitiendo recorrer todos los valores del vector R2.

Para entender el algoritmo hagamos una prueba de escritorio para la siguiente escena, en la cual se han ubicado correctamente los elementos del Universal.



El algoritmo debe calcular cuántos elementos hay en la zona sombreada, que para el ejemplo debería dar como resultado **5**. Haremos nuestra prueba de escritorio a través de la siguiente tabla y usando los algoritmos anteriores.

N	R1[n] = ¿?	R2[R1[n]]=¿?	K	Elementos
1	R1[1]=3	R2[3]=1	1	1
2	R1[2]=5	R2[5]=1	1	2
3	R1[3]=5	R2[5]=1	1	3
4	R1[4]=5	R2[5]=1	1	4
5	R1[5]=5	R2[5]=1	1	5
6	R1[6]=6	R2[6]=0	0	5
7	R1[7]=4	R2[4]=0	0	5
8	R1[8]=4	R2[4]=0	0	5
9	R1[9]=4	R2[4]=0	0	5
10	R1[10]=4	R2[4]=0	0	5

Recuerda:

R1[n] almacena la región donde debe estar el elemento.

R2[n] almacena unos y ceros para regiones **n** sombreadas o no respectivamente. Para el ejercicio, se observa que el Universal está contenido en las regiones 3, 4, 5 y 6, de las cuales sólo están sombreadas las regiones 3 y 5 (ver imagen anterior).

La sentencia **k=(R2[R1[n]]=1)?1:0** asigna valores de 1 a **k** cuando R2[R1[n]]=1 y de cero en caso contrario.

La proposición **elementos = elementos + k** acumula los valores unitarios de **k** permitiendo obtener el total de elementos contenidos en la región sombreada.

Algoritmo calculador del número de aciertos.

Crea un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

n=0;k1=0;k2=0;aciertos=0

Hacer. En este campo escribe las siguientes proposiciones:

n=n+1
k2=(R1[n]=R3[n])+(R2[R1[n]]=1)
k1=(k2=2)?1:0
aciertos=aciertos+k1

Mientras. En este campo escribiremos **n<10.** Esto significa que el algoritmo se ejecutará 10 veces, permitiendo recorrer todos los valores del vector R1 y de R3.

Algoritmo calculador del número de fallos.

Agrega un algoritmo con las siguientes características:

Inicio. En este campo incluye las siguientes asignaciones:

n=0;k3=0;k4=0;k5=0;fallos=0

Hacer. En este campo escribe las siguientes proposiciones:

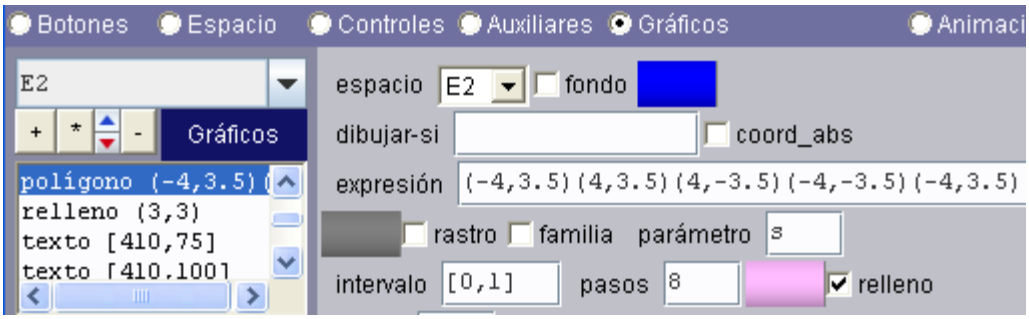
n=n+1
k3=(R1[n]=R3[n])+(R2[R1[n]]=0)
k4=(k3=2)?1:0
k5=(R1[n]!=R3[n])*(R3[n]>0)
fallos=fallos+k4+k5

Mientras. En este campo escribiremos **n<10.** Esto significa que el algoritmo se ejecutará 10 veces, permitiendo recorrer todos los valores del vector R1 y de R3.

El análisis de estos dos últimos algoritmos te lo dejamos como ejercicio.

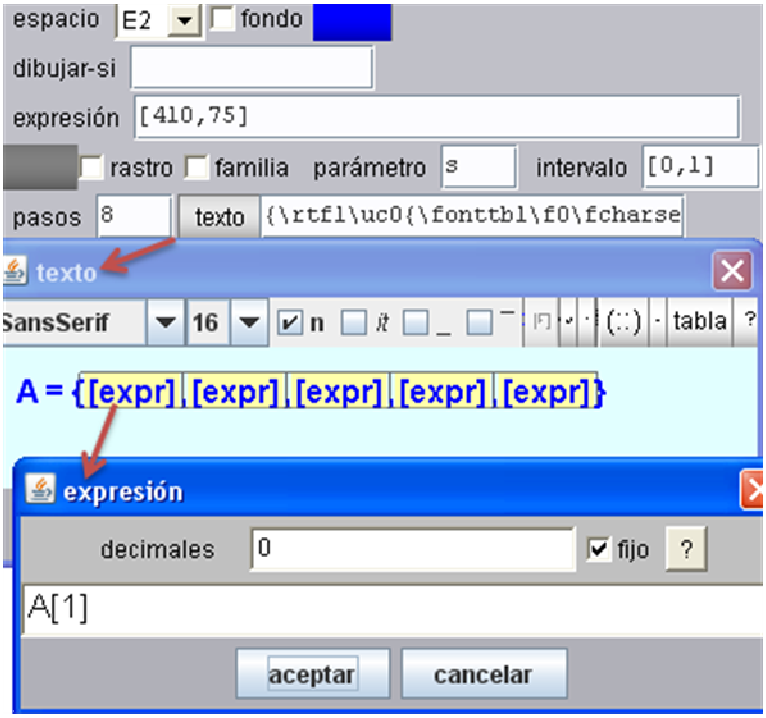
1.4 **Gráficos.** Los gráficos que nos restan por incluir son pocos. La mayoría ya los incluimos en los pasos anteriores.

Polígono. Nos faltaba añadir el rectángulo que incluye los diagramas de Venn. Hazlo de acuerdo a la siguiente imagen:



Textos. Finalmente, añade los siguientes textos:

Texto 1. En la posición [400,75] inserta un texto que muestre el conjunto A con sus elementos. La imagen siguiente te da una idea de cómo hacerlo:



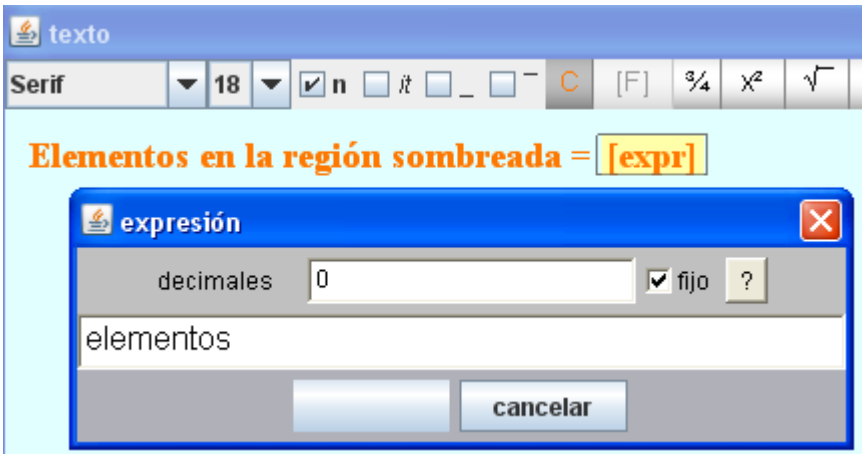
La primera expresión se muestra en el recuadro inferior.

Texto 2. En la posición [410, 100] haces lo mismo con el conjunto B.

Texto 3. En la posición [410, 125] representas el conjunto C.

Texto 4. En la posición [430, 160] añades un texto que diga UNIVERSAL.

Texto 5. En la posición [130, 3] incluyes un texto de acuerdo a la siguiente imagen.

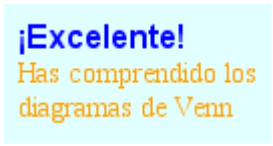


Texto 6. En la posición [130,22] lo haces con el texto "elementos bien ubicados". En este texto debes usar la variable **aciertos.**

Texto 7. En la posición [130,40] añades el texto "elementos mal ubicados", usando la variable **fallos**.

Texto 8, 9 y 10. En las posiciones [120, 85], [30,205] y [325,205] insertarás los textos **A, B** y **C** respectivamente.

Texto 11. En la posición [460, 10] agregas un texto que diga "Excelente, has comprendido los diagramas de Venn" o el texto que desees. Este texto sólo aparecerá bajo la siguiente condición **dibujar-si = (fallos=0)&(aciertos=elementos).**



La escena final es como la primera imagen de esta clase.

Bueno, eso es todo, espero que hayas comprendido esta clase. Hasta la próxima.