








Guía 3: alternativa y expresiones

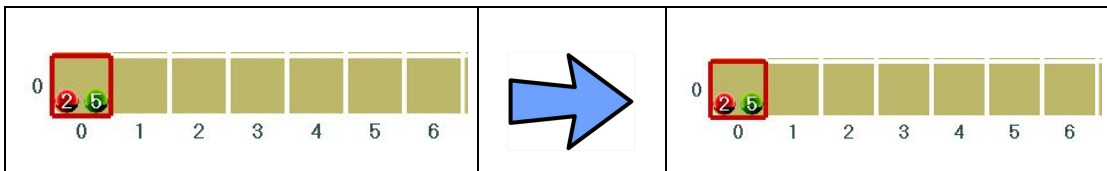
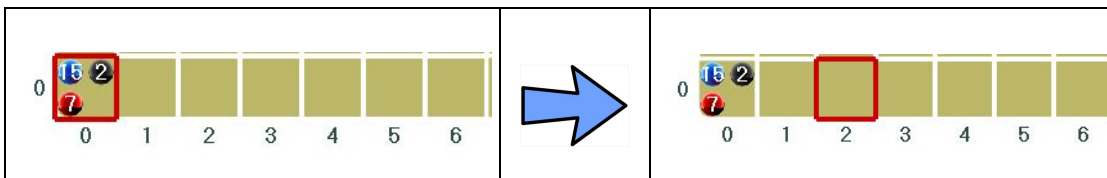
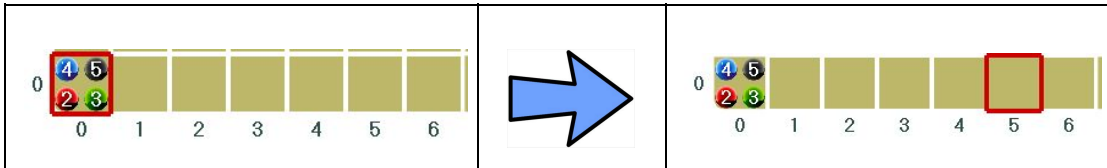
Nuestra primer expresión: nroBolitas.

1. Varios procedimientos usando nroBolitas.
 - a. Escribir el procedimiento CopiarColor(*col1*, *col2*), que agrega a la celda actual tantas bolitas de color *col2* como bolitas de color *col1* haya.
 - b. Escribir el procedimiento SacarTodas(*col*), que saca todas las bolitas de color *col* de la celda actual.
 - c. Escribir el procedimiento DuplicarBolitas(*col*), que duplica la cantidad de bolitas de color *col* en la celda actual.
 - d. Escribir el procedimiento DuplicarTodas(), que duplica la cantidad de bolitas de cada color en la celda actual.
 - e. Escribir el procedimiento PasarTodoANegro(), que “reemplaza” todas las bolitas verdes, rojas y azules en la celda actual, por bolitas negras.
 - f. Escribir el procedimiento VaciarCelda(), que saca todas las bolitas de la celda actual.

Este es un ejemplo de uso de cada uno de estos procedimientos.

Celda inicial	Al evaluar	Se obtiene
	CopiarColor(Rojo, Negro)	
	SacarTodas(Rojo)	
	DuplicarBolitas(Rojo)	
	DuplicarTodas()	
	PasarTodoANegro()	
	VaciarCelda()	

2. Escribir el procedimiento `NivelarColor(cant, col)`, que deja exactamente `cant` bolitas de color `col` en la celda actual. Suponer que hay, a lo sumo, `cant` bolitas de color `col` en esa celda, o sea que hay que poner, nunca sacar. No vale sacar todas y poner las necesarias, hay que poner las que faltan.
3. Escribir el procedimiento `MoverSegunColor(dir, col)`, que mueve el cabezal en la dirección `col` tantas celdas como bolitas de color `col` hay en la celda actual. Veamos algunos ejemplos de evaluar `MoverSegunColor(Este, Negro)`, según la situación inicial.



En el último caso, como la celda no tiene bolitas negras, o sea tiene 0 bolitas negras, entonces el cabezal no se mueve, o se mueve 0 celdas hacia el Este.

Alternativa

1. Escribir el procedimiento `MoverSeguro(dir)`, que mueve el cabezal en la dirección indicada, si hay a dónde moverse. Si no, no hace nada.
2. Escribir el procedimiento `SacarSiHay(col)`, que saca una bolita de color `col` de la celda actual, si hay bolitas de ese color. Si no, no hace nada.
3.
 - a. Escribir el procedimiento `AsegurarAlMenosUna(col)`, que pone una bolita del color indicado en la celda actual, si no hay ninguna bolita de ese color. Si ya hay bolitas de ese color, no hace nada.
 - b. Escribir un procedimiento `AsegurarUnaDeCada` que asegure al menos una bolita de cada color en la celda actual. (ejercicio 38 de la guía común 3)

4. Escribir una versión mejorada del procedimiento `NivelarColor(cant, col)`, en la que también se contemple el caso en que hay más de `cant` bolitas del color indicado en la celda actual. Acá tampoco vale sacar todas y poner las que quiero. Hay que poner lo que falta ó sacar lo que sobra.
5. Hacer los ejercicios 27, 28, 30 y 31 de la guía común 3.
6. Escribir el procedimiento `MoverPalanteOPatras(dirAdelante, colGuia)`, que mueve el cabezal en dirección `dirAdelante` si en la celda actual hay bolitas de color `colGuia`, y en la dirección opuesta si no. Vale suponer que hay celdas en ambas direcciones ... o no, usando `MoverSeguro`.
7. Escribir el procedimiento `MoverSegunColor(dir1, dir2, col)`, que mueve el cabezal una celda en dirección `dir1` si en la celda actual hay bolitas de color `col`, y en caso contrario, lo mueve en dirección `dir2`. Si no se puede mover en la dirección elegida, no hace nada.
8. Escribir el procedimiento `MoverSegunCantDeColor(col)`, que mueve el cabezal una celda en una dirección que depende de la cantidad de bolitas de color `col` que haya en la celda actual:

Cantidad de bolitas de color <code>col</code>	Dirección del movimiento
1	Este
2	Norte
3	Oeste
4 o más	Sur
ninguna	no hace nada

Si no se puede mover en la dirección elegida, no hace nada.

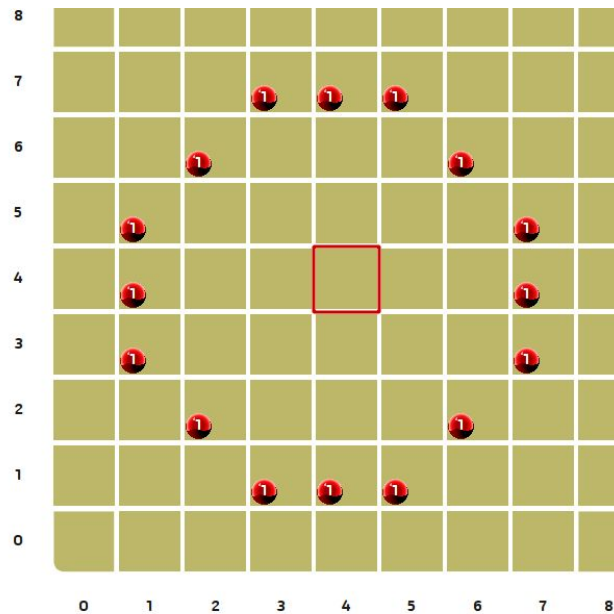
9. Escribir el procedimiento `MoverSegunColor(dir1, dir2, col)`, que mueve el cabezal una celda en dirección `dir1` si en la celda actual hay bolitas de color `col`, y en caso contrario, lo mueve en dirección `dir2`. Si no se puede mover en la dirección elegida, no hace nada.
9. Escribir el procedimiento `MoverUnaBolitaHacia(dir, col)`, que “mueve” una bolita de color `col` a la celda vecina en dirección `dir`. O sea, sacar una bolita de ese color de la celda actual, y la pone en la celda vecina. El cabezal queda en la celda vecina. Si no hay vecina en dirección `dir`, o en la celda actual no hay bolitas de color `col`, entonces no se hace nada.
10. Escribir el procedimiento `DescomprimirHacia(dir, col1, col2)`, que “descomprime” la celda actual, sacando una bolita y poniéndola en la celda vecina en dirección `dir`, o sea, “moviendo” una bolita en esa dirección. La bolita que se

mueve es del color, entre col1 y col2, del que haya más bolitas en la celda. Si no hay vecina en dirección dir, o en la celda actual no hay bolitas ni de color col1 ni de color col2, entonces no se hace nada.

Las Manecillas del Reloj

Se desea dibujar las manecillas de un reloj para poner la hora. Una manecilla azul de largo 3 para la hora, y una manecilla negra de largo 4 para el minuterero.

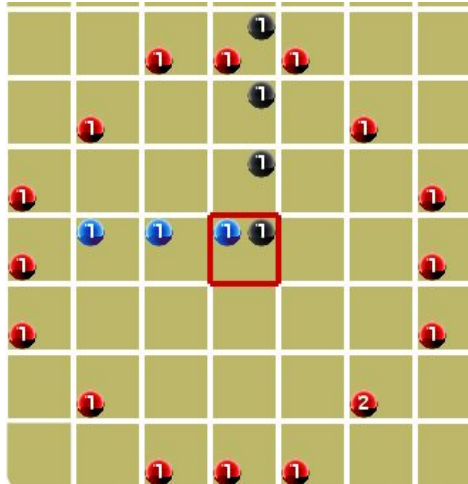
Supongamos este tablero inicial:



Debe crearse este tablero inicial, con el cabezal en esa posición. Donde se encuentra subida esta guía se incluye ManecillasReloj.gbb (el tablero), para agilizar. Además, se propone codificar las horas así:

```
procedure MarcarLas9EnPunto(){
    PonerHora(Oeste)
    PonerMinutero(Norte)
}
```

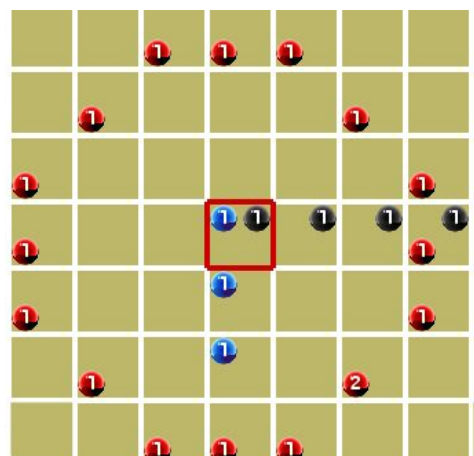
Lo que produce el siguiente resultado:



Las 9 en punto

```
procedure MarcarLas6YCuarto(){
    PonerHora(Sur)
    PonerMinutero(Este)
}
```

Lo que produce el siguiente resultado:



Las 6 y cuarto

Consigna:

1. Escribir **PonerHora**(dir) y **PonerMinutero**(dir) para que **MarcarLas9EnPunto** y **MarcarLas6YCuarto** funcionen tal como están.
2. Hacer **MarcarHora**(hora,min) tal que se pueda marcar las seis y cuarto así: **MarcarHora(6,15)**. Deben poder marcarse las 12, 3, 6 y 9, y los minutos deben poder ser 0, 15, 30, y 45.

El caminante

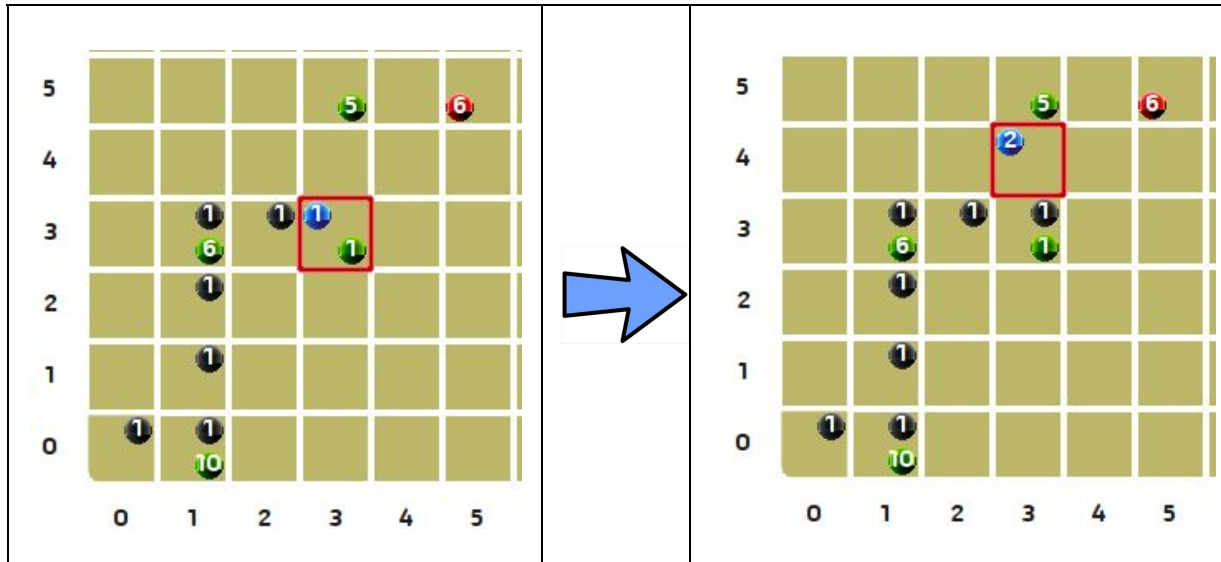
Vamos a modelar el paseo de un caminante por el tablero:

- El caminante está representado por una o dos bolitas azules. La dirección de su paseo es Este si es una bolita, y Norte si es dos.
- Las bolitas verdes son indicadores de cambio de dirección. Si el caminante llega a una celda con casillas verdes, debe cambiar de dirección.
- El caminante deja una huella de bolitas negras a su paso.
- El paseo del caminante termina en cualquier celda con bolitas rojas.

Implementar los siguientes procedimientos para hacer caminar a nuestro caminante

1. **MoverBolitas**(cant,col,dir): "mueve" (como en **MoverUnaBolitaHacia**) la cantidad indicada de bolitas de color col a la casilla vecina en dirección dir, y deja el cabezal en esa celda. Suponer que hay una vecina en esa dirección.
2. **CambiarEntreUnaYDos**(col): si en la celda actual hay una bolita de color col agrega una (para que queden dos), si hay dos saca una (para que quede una), en cualquier otro caso no hace nada.
3. **DarUnPaso**(): implementar un paso en el paseo del caminante, de acuerdo a estas reglas:
 - a. si hay bolitas rojas, el caminante ya llegó a destino, no hay nada más que hacer.
 - b. si hay bolitas verdes, cambiar la dirección (o sea, si hay una bolita azul pasar a dos y si hay dos pasar a una), y moverse en la nueva dirección.
 - c. si no, moverse en la dirección que corresponde a su número de bolitas.

En cualquier caso, hay que recordar dejar la huella, conviene que sea en la casilla que se deja. A continuación un ejemplo:



El bosque

Versión inicial

Vamos a usar el tablero para representar un bosque. Cada celda es una parcela. Cada bolita verde representa un árbol. Cada bolita roja representa una semilla. Una bolita negra representa una bomba, que al explotar, derriba 5 árboles en la celda actual, y 3 en la celda vecina al norte. Una bolita azul representa una unidad de nutrientes.

Escribir estos procedimientos:

1. Germinar(), que transforma una semilla en planta en la celda actual. La germinación consume tres unidades de nutrientes. Si en la celda no hay semilla, o no hay suficientes nutrientes, no se hace nada.
2. Alimentarse(), que hace que los árboles de la celda actual se alimenten, consumiendo un nutriente cada uno. El único cambio que hay que hacer es la eliminación de los nutrientes. Si hay menos nutrientes de lo que se necesita, se consumen todos los que hay.
3. ExplotarBomba(), que explota una bomba en la celda actual, eliminando árboles según lo que se indica al principio. Si la celda actual está en el borde norte, entonces sólo se eliminan los árboles de la celda actual. Ojo, cuando haya menos árboles de los que la bomba puede eliminar, entonces elimina los que haya. La bomba se consume en el proceso, o sea, hay que eliminarla.
4. Polinizar(): los árboles en la celda actual polinizan la celda vecina en la dirección este, generando tantas semillas en esa celda como árboles haya en la celda actual, menos 3. P.ej. si en la celda actual hay 5 árboles, se generan 2 semillas en la celda vecina al este. Si en la celda actual hay menos de 3 árboles, o no tiene vecina al este, entonces no se hace nada.

Un poco de pimienta

Implementar estos procedimientos, que corresponden a una versión un poco más compleja de la misma idea. Para estos procedimientos, tener en cuenta la expresión `opuesto(dir)`.

5. `PolinizarHacia(dir)`: misma idea que en `Polinizar()`, pero ahora la dirección es la indicada, no necesariamente hacia el este.
6. `ExplotarBombaEnTodasDirecciones()`, que explota una bomba derribando 5 árboles en la celda actual, y 3 en cada vecina. Tener en cuenta que en cada dirección puede, o no, haber una celda vecina.