

Y esto, ¿con qué se come?

Tomate unos pocos minutos y tratá de entender qué hace este procedimiento:

```
procedure MoverSegunBolitas() {
  if (nroBolitas(Azul) + nroBolitas(Negro) + nroBolitas(Rojo) + nroBolitas(Verde) > 10
) {
    Mover(Este)
  } else {
    Mover(Norte)
  }
}
```

Cuando lo logres interpretar (o te canses), presioná Enviar y mirá el resultado.

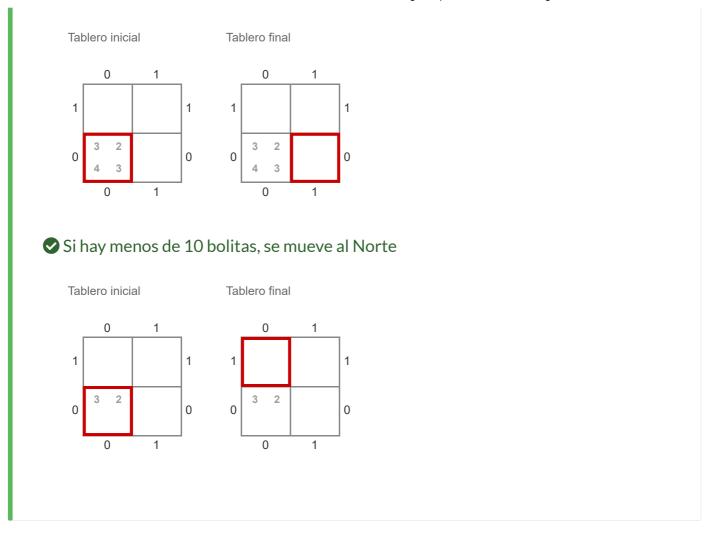
```
procedure MoverSegunBolitas() {
    if (nroBolitas(Azul) + nroBolitas(Negro) + nroBolitas(Rojo) +
    nroBolitas(Verde) > 10) {
        Mover(Este)
    } else {
        Mover(Norte)
    }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

Si hay más de 10 bolitas, se mueve al Este



Costó entender qué hace sin ejecutarlo, ¿no?

Eso es señal de que nos está faltando dividir en subtareas...

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas





La importancia de nombrar las cosas

Como vimos, el problema de lo anterior era la falta de **división en subtareas**: la **expresión** que cuenta la cantidad de bolitas que hay en la celda es demasiado **compleja**, y cuesta entender a simple vista que hace eso.

Entonces, lo que nos está faltando es algún mecanismo para poder darle un nombre a esa expresión compleja; algo análogo a los procedimientos pero que sirva para encapsular expresiones.

La buena noticia es que Gobstones nos permite hacer esto, y la herramienta para ello es definir una **función**, que se escribe así:

```
function nroBolitasTotal() {
  return (nroBolitas(Azul) + nroBolitas(Negro) + nroBolitas(Rojo) + nroBolitas(Verde))
}
```

Pegá el código anterior en el editor y observá el resultado.

```
function nroBolitasTotal() {
   return (nroBolitas(Azul) + nroBolitas(Negro) + nroBolitas(Rojo) +
   nroBolitas(Verde))
}
```

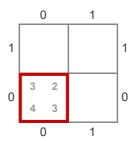


¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

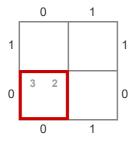
nroBolitasTotal() -> 12

Tablero inicial



nroBolitasTotal() -> 5

Tablero inicial



Algunas aclaraciones sobre las funciones:

- son un caso particular de las expresiones, y por lo tanto siguen las mismas reglas que ellas: se escriben con la primera letra minúscula y siempre denotan algún valor (en este caso, un número);
- en la última línea de su definición siempre va un return, seguido de una expresión entre paréntesis: el valor que la función va a retornar.

© 2015-2022 Ikumi SRL





MoverSegunBolitas, versión 2

Ahora que ya logramos mover la cuenta de las bolitas auna subtarea, podemos mejorar el procedimiento que habíamos hecho antes.

3. MoverSegunBolitas, versión 2

Modificá la primera versión de MoverSegunBolitas para que use la función nroBolitasTotal() en vez de la expresión larga.

```
✓ Solución 
✓ Biblioteca
```

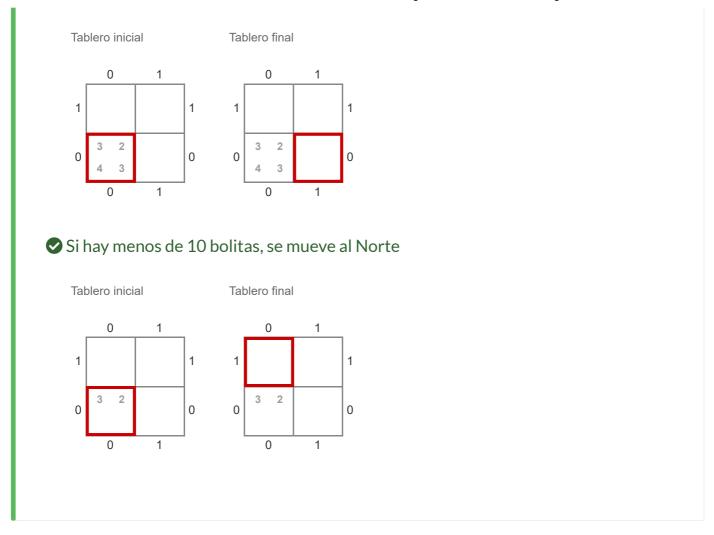
```
procedure MoverSegunBolitas() {
   if (nroBolitasTotal() > 10) {
      Mover(Este)
   } else {
      Mover(Norte)
   }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

Si hay más de 10 bolitas, se mueve al Este



Las **funciones** son una herramienta importantísima, que nos ayuda a escribir programas de mayor calidad.

Sólo mirando el código de esta nueva versión del procedimiento podemos entender de qué va nuestro problema, lo que **reduce la distancia** entre el problema real y la **estrategia** que elegimos para resolverlo.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas







todasExcepto

Te toca ahora definir tu primera función: todasExcepto(color). Lo que tiene que hacer es sencillo, contar cuántas bolitas hay en la celda actual **sin tener en cuenta** las del color recibido por parámetro.

Por ejemplo, todasExcepto(Verde) debería contar todas las bolitas azules, negras y rojas que hay en la celda actual (o dicho de otra forma: todas las bolitas que hay **menos** las verdes).

Definí la función todasExcepto para que retorne la cantidad de bolitas que **no** sean del color que se le pasa por parámetro.

🗘 ¡Dame una pista!

```
function todasExcepto(color) {
   return (nroBolitasTotal()-nroBolitas(color))
}
```

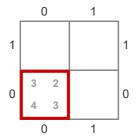


¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

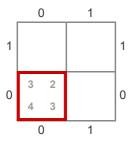
⊘ todasExcepto() -> 8

Tablero inicial



✓ todasExcepto() -> 10

Tablero inicial



Las funciones, como cualquier otra expresión, se pueden usar para definir nuevas funciones.

Y volvemos así a lo más lindo de la programación: la posibilidad de **construir nuestras propias herramientas** parándonos sobre cosas que hicimos antes, logrando que lo que hacemos sea cada vez más poderoso.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL



Una función de otro tipo

Como ya sabés, las expresiones no sólo sirven para operar con números. Vamos a definir ahora una función que retorne un valor **booleano** (True / False).

Lo que queremos averiguar es si el color Rojo es dominante dentro de una celda. Veamos algunos ejemplos.

En este casillero:

$$\begin{array}{c|c}
0\\
4 & 3\\
2 & 1
\end{array}$$

rojoEsDominante() **retorna** False (hay 2 bolitas rojas contra 8 de otros colores). Pero en este otro:

$$\begin{array}{c|c}
0\\
4 & 3\\
9 & 1
\end{array}$$

rojoEsDominante() retorna True (hay 9 bolitas rojas contra 8 de otros colores)

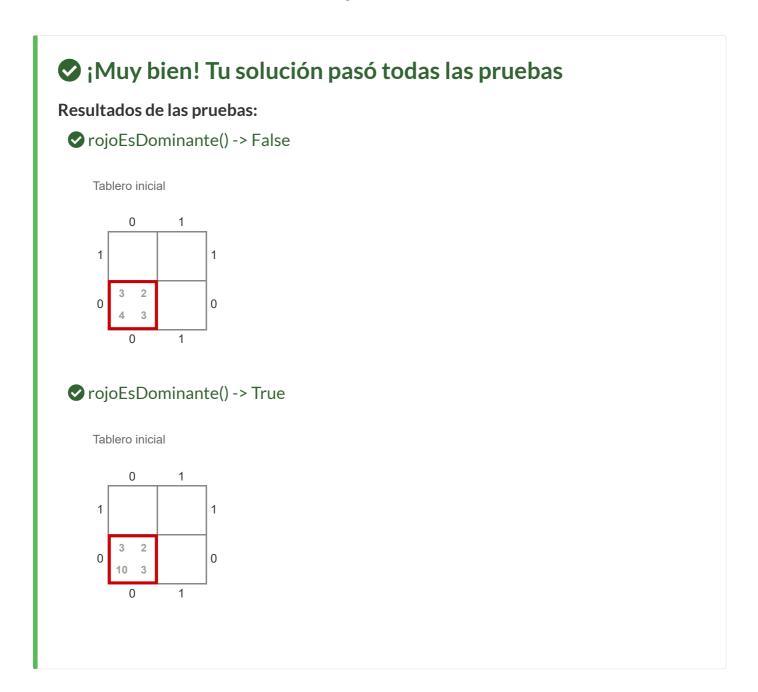
Definí la función rojoEsDominante() que nos diga si la cantidad de bolitas rojas **es mayor** que la suma de las bolitas de los otros colores. En la *Biblioteca* está todasExcepto(color) lista para ser invocada.

\mathbb{Q} ¡Dame una pista!



10/10/22, 15:09

5



Las funciones pueden retornar distintos **tipos**: un color, una dirección, un número o un booleano.

Básicamente, lo que diferencia a un tipo de otro son las **operaciones que se pueden hacer con sus elementos**: tiene sentido sumar números, pero no colores ni direcciones; tiene sentido usar Poner con un color, pero no con un booleano. Muchas veces, pensar en el tipo de una función es un primer indicador útil de si lo que estamos haciendo está bien.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL





En libertad

Queremos definir la función esLibreCostados(), que determine si el cabezal tiene libertad para moverse hacia los costados (es decir, Este y Oeste).

Antes que nada, pensemos, ¿de qué **tipo** tiene que ser el valor que retorna nuestra función? Será...

- ... ¿un color? No.
- ... ¿un **número**? Tampoco.
- ... ¿una dirección? Podría, pero no. Fijate que lo que pide es "saber si puede moverse" y no hacia dónde.
- ... ¿un **booleano**? ¡Sí! Cómo nos dimos cuenta: lo que está pidiendo tiene pinta de **pregunta** que se responde con sí o no, y eso es exactamente lo que podemos representar con un valor booleano: **Verdadero** o **Falso**.

Pero, ups, hay un problema más; hay que hacer DOS preguntas: ¿se **puede mover** al Este? **Y** ¿se **puede mover** al Oeste?.

Bueno, existe el operador & que sirve justamente para eso: toma dos expresiones booleanas y devuelve True solo si **ambas** son verdaderas. Si sabés algo de lógica, esto es lo que comunmente se denomina **conjunción** y se lo suele representar con el símbolo A.

Por ejemplo, si quisieramos saber si un casillero tiene más de 5 bolitas y el Rojo es el color dominante podríamos escribir:

```
function esUnCasilleroCargadoConRojoDominante() {
  return (nroBolitasTotal() > 5 && rojoEsDominante())
}
```

Definí la función esLibreCostados() que indique si el cabezal puede moverse tanto al Este como al Oeste.

♀¡Dame una pista!

```
function esLibreCostados() {
  return (puedeMover(Este) && puedeMover(Oeste))
}
```

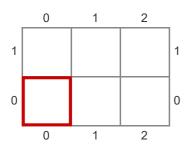
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

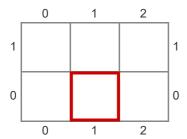
esLibreCostados() -> False

Tablero inicial



esLibreCostados() -> True

Tablero inicial



Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL





Cualquier bolita nos deja bien

Definí la función hayAlgunaBolita() que responda a la pregunta ¿hay alguna bolita en la celda actual?

7. Cualquier bolita nos deja bien

Otra vez una pregunta, por lo tanto hay que retornar un **booleano**. Además, podemos ver que acá también hay que hacer **más de una pregunta**, en particular cuatro: una por cada una de los colores.

A diferencia del ejercicio anterior, lo que queremos saber es si **alguna** de ellas es verdadera, por lo tanto hay que usar otro operador: la **disyunción**, que se escribe || y retorna verdadero si al menos **alguna de las dos** preguntas es verdadera.

De nuevo, si sabés algo de lógica, esta operación suele representarse con el símbolo ∨.

?¡Dame una pista!

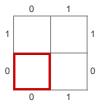
```
function hayAlgunaBolita() {
  return (hayBolitas(Rojo) || hayBolitas(Verde) || hayBolitas(Negro) || hayBolitas(Azul))
}
```

Enviar

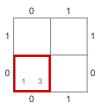
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

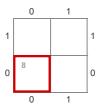
Tablero inicial



Tablero inicial

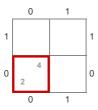


Tablero inicial



♦ hayAlgunaBolita() -> True

Tablero inicial



Tanto & como || pueden usarse varias veces sin la necesidad de usar paréntesis, siempre y cuando tengan expresiones booleanas a ambos lados.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL





Siempre al borde...

Te recordamos los operadores lógicos que vimos hasta ahora:

8. Siempre al borde...

- Negación: "da vuelta" una expresión booleana ejemplo: not hayBolitas(Rojo).
- Conjunción: determina si se cumplen ambas condiciones ejemplo: puedeMover(Norte) && puedeMover(Sur).
- Disyunción: determina si se cumple alguna de las condiciones ejemplo: esInteligente() || tieneBuenaOnda().

Con la ayuda de esa tablita, definí la función estoyEnUnBorde() que determine si el cabezal está parado en algún borde.

🗘 ¡Dame una pista!

```
function estoyEnUnBorde() {
    return (not puedeMover(Norte) || not puedeMover(Este) || not puedeMover(Sur) || not
    puedeMover(Oeste))
}
```

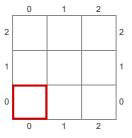
Enviar

⊘ ¡Muy bien! Tu solución pasó todas las pruebas

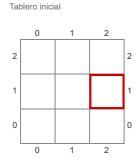
Resultados de las pruebas:

estoyEnUnBorde() -> True

Tablero inicial

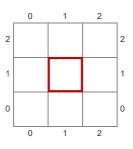


estoyEnUnBorde() -> True



estoyEnUnBorde() -> False

Tablero inicial



Como en la aritmética, en la lógica también existe el concepto de **precedencia** y ciertas operaciones se resuelven antes que otras: primero la negación (not), después la conjunción (&&) y por último la disyunción (||).

Por esta razón, la expresión not puedeMover(Norte) || not puedeMover(Este) || not puedeMover(Sur) || not puedeMover(Oeste) se puede escribir sin tener que poner paréntesis en el medio.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

Información importante Términos y Condiciones

Reglas del Espacio de Consultas





Las compañeras ideales

Vamos a ver ahora funciones que hacen cosas antes de retornar un resultado. Para ejemplificar esto, vamos a querer que definas una función que nos diga si hay una bolita de un color específico, pero en la celda de al lado.

Definí la función hayBolitasAl(direccion, color) que informe si hay alguna bolita del color especificado en la celda vecina hacia la dirección dada.

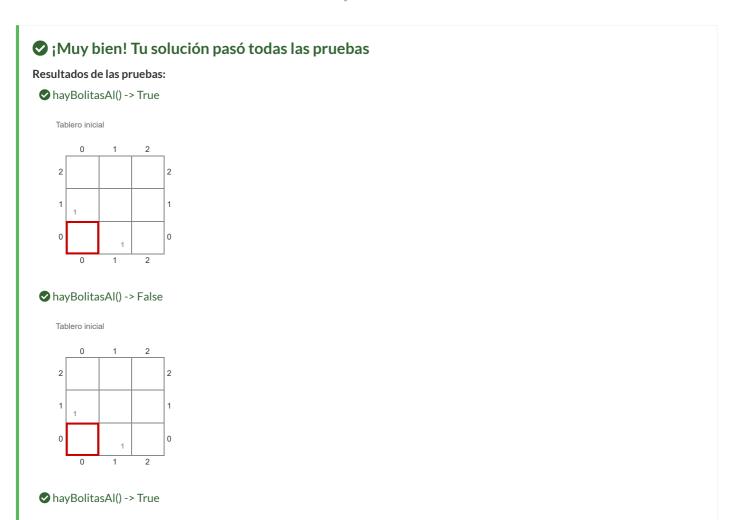
Ojo: como ya dijimos, la última línea siempre tiene que tener un return.

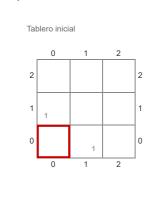
O¡Dame una pista!

```
function hayBolitasAl(direccion, color) {
   Mover(direccion)
   return (hayBolitas(color))
}

5
6
7
8
```

Enviar





¿Viste qué pasó? El cabezal "no se movió" y sin embargo la función devolvió el resultado correcto.

Esto pasa porque en Gobstones las funciones son **puras**, no tienen **efecto real** sobre el tablero. En ese sentido decimos que son las compañeras ideales: después de cumplir su tarea **dejan todo como lo encontraron**.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL





Lo ideal también se puede romper

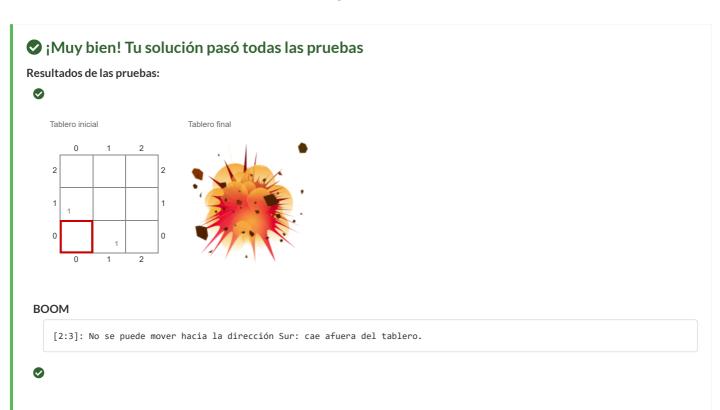
Como en la definición de hayBolitasAl se usa Mover, es obvio que hay casos en los cuales podría romperse: basta con posicionar el cabezal en el origen y preguntar si hayBolitas de algún color al Oeste.

Pero, ¿no era que las funciones eran puras y no tenían efecto real? ¿Qué pasa si una función hace BOOM?

Hagamos la prueba: vamos a probar la función hayBolitasAl del ejercicio anterior con casos donde no pueda moverse el cabezal. Presioná Enviar y mirá el resultado.

```
function hayBolitasAl(direccion, color) {
    Mover(direccion)
    return (hayBolitas(color))
}
```

Enviar





¡BOOM!

Las funciones también pueden **producir BOOM** y por lo tanto tenés que tener el mismo cuidado que al programar un procedimiento: que el cabezal no salga del tablero, no intentar sacar bolitas de un color que no hay, etc.

Pensándolo así, podemos decir que las funciones **deshacen sus efectos** una vez que terminan, pero para poder devolver un resultado necesitan que sus acciones puedan ejecutarse.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas





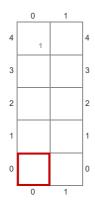
¿Hay bolitas lejos?

Ejercitemos un poco más esto de las funciones con procesamiento.

Te toca programar una nueva versión de hayBolitasAl que mire si hay bolitas a cierta distancia de la celda actual. A esta función la vamos a llamar hayBolitasLejosAl y recibirá tres parámetros: una dirección hacia donde deberá moverse, un color por el cual preguntar y una distancia que será la cantidad de veces que habrá que moverse.

Por ejemplo: hayBolitasLejosAl(Norte, Verde, 4) indica si hay alguna bolita Verde cuatro celdas al Norte de la posición actual.

Para este tablero devolvería True:



Y para este tablero devolvería False:



Definíla función hayBolitasLejosAl(direccion, color, distancia).

○ ¡Dame una pista!

✓ Solución
✓ Biblioteca

```
function hayBolitasLejosAl (direccion, color, distancia) {

MoverN(distancia, direccion)

return (hayBolitas(color))

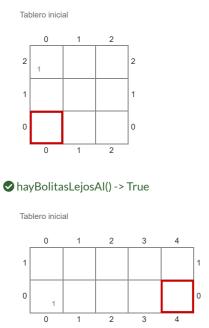
5

6
```

Enviar

Resultados de las pruebas:

♦ hayBolitasLejosAl() -> True

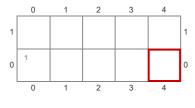


♦ hayBolitasLejosAl() -> False

• Palse

•





Se puede realizar cualquier tipo de acción antes de retornar un valor, y nada de lo que hagamos tendrá efecto real sobre el tablero.

Interesante esto de las funciones, ¿no?

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas





Estoy rodeado de viejas bolitas

Valiéndote de hayBolitasAl, definí la función estoyRodeadoDe(color) que indica si el cabezal está rodeado de bolitas de ese color. Decimos que el cabezal "está rodeado" si hay bolitas de ese color en las cuatro direcciones: Norte, Este, Sur y Oeste.

♀¡Dame una pista!

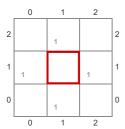
Enviar

Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

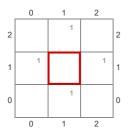
estoyRodeadoDe() -> True

Tablero inicial



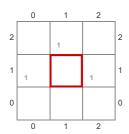
estoyRodeadoDe() -> True

Tablero inicial



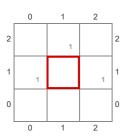
estoyRodeadoDe() -> False





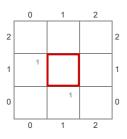
estoyRodeadoDe() -> False

Tablero inicial



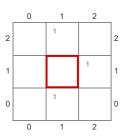
estoyRodeadoDe() -> False

Tablero inicial



stoyRodeadoDe() -> False

Tablero inicial



Por si todavía no nos creías: a pesar de que el cabezal se movió cuatro veces por cada prueba, al finalizar la función vemos que siempre quedó en la posición inicial.

Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

Reglas del Espacio de Consultas





Sin límites

Para cerrar, vamos a definir la función hayLimite(), que determina si hay algún tipo de límite a la hora de mover el cabezal.

El límite puede ser por alguno de dos factores: porque **estoy en un borde** y entonces no me puedo mover en alguna dirección, o porque **estoy rodeado de bolitas rojas** que me cortan el paso. Si ocurre **alguna** de esas dos condiciones, quiere decir que hay un límite.

Usando estoyEnUnBorde y estoyRodeadoDe, definí hayLimite.

```
✓ Solución 
✓ Biblioteca
```

```
function hayLimite() {
  return (estoyRodeadoDe(Rojo) || estoyEnUnBorde())
}

function hayLimite() {
  return (estoyRodeadoDe(Rojo) || estoyEnUnBorde())
}
```

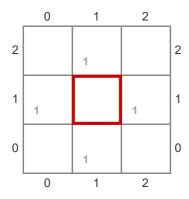
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

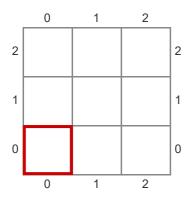
Resultados de las pruebas:

♦ hayLimite() -> True

Tablero inicial

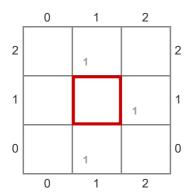


Tablero inicial



♦ hayLimite() -> False

Tablero inicial



Esta guía fue desarrollada por Federico Aloi bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

© 2015-2022 Ikumi SRL

