Sacar con miedo

Vamos a definir un procedimiento que saque una bolita azul "con miedo": no tiene que producirse un *BOOM*, aún cuando no haya ninguna bolita en la celda actual.

Con lo que sabés hasta ahora, probablemente tu primera idea sea hacer algo como esto:

```
procedure SacarAzulConMiedo() {
  Sacar(Azul)
}
```

> ¡Probalo! Copiá el código anterior en el editor y apretá *Enviar*.

```
1 procedure SacarAzulConMiedo() {
2   Sacar(Azul)
3 }
```

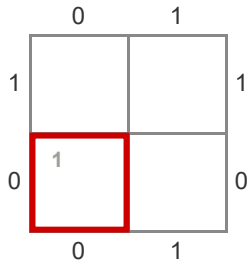
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

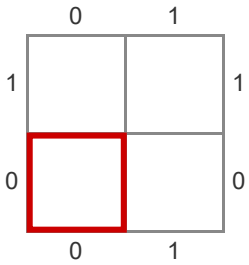
Resultados de las pruebas:

- Cuando hay una bolita azul, la saca

Tablero inicial

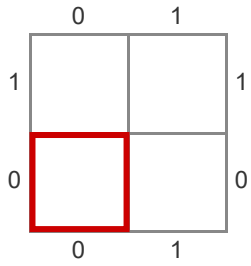


Tablero final



✅ Cuando no hay ninguna bolita azul, hace BOOM

Tablero inicial



Tablero final

BOOM

```
[2:3]: No se puede sacar una bolita de color Azul: no hay bolitas de ese color.
```

¿Te diste cuenta qué pasó?

Funcionó para el primer tablero porque tenía una bolita azul, pero hizo *BOOM* para el segundo porque estaba vacío, claro.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Sacar con miedo, segundo intento

Ahora probá esta segunda versión que agrega una **alternativa condicional**. No te preocupes por la sintaxis, ya te lo vamos a explicar.

```
procedure SacarAzulConMiedo() {
  if (hayBolitas(Azul)) {
    Sacar(Azul)
  }
}
```

> Copiá el código anterior en el editor y apretá *Enviar*.

```
1 procedure SacarAzulConMiedo() {
2   if (hayBolitas(Azul)) {
3     Sacar(Azul)
4   }
5 }
```

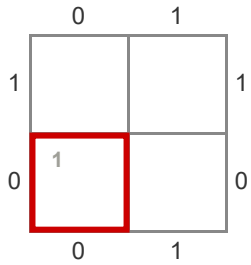
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

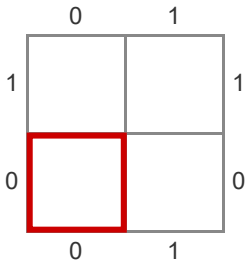
Resultados de las pruebas:

- Cuando hay una bolita azul, la saca

Tablero inicial

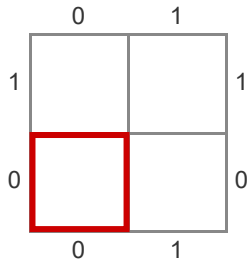


Tablero final

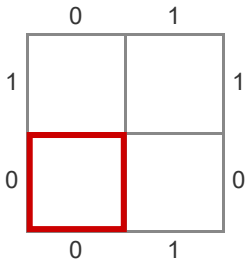


✅ Cuando no hay ninguna bolita azul, no hace nada

Tablero inicial



Tablero final



¡Bien!

Hiciste tu primer procedimiento que **decide** antes de ejecutar. Seguí con nosotros para entender de qué se trata esto...

Eliminando la bolita roja

Analicemos el procedimiento del ejercicio anterior:

3. Eliminando la bolita roja

```
procedure SacarAzulConMiedo() {
  if (hayBolitas(Azul)) {
    Sacar(Azul)
  }
}
```

Como notarás, introdujimos una nueva estructura de control: el **if**, que en castellano significa *si*; entendiendo al *si* como **condicional** ("*si* tuviera hambre me comería una empanada") y no como afirmación ("*sí*, yo rompí el teléfono").

Entonces, lo que le estamos diciendo a la computadora es *"si hay bolitas azules, sacá una bolita azul"*, que dicho así suena un poco tonto ¡y lo es!. Ya te dijimos que la computadora sólo sabe cumplir órdenes.

> ¡Ahora te toca a vos! Modificá el procedimiento que te dimos para que saque una bolita roja, sólo *si* hay alguna.

¡Dame una pista!

```
procedure SacarRojoConMiedo() {
  if (hayBolitas(Rojo)) {
    Sacar(Rojo)
  }
}
```

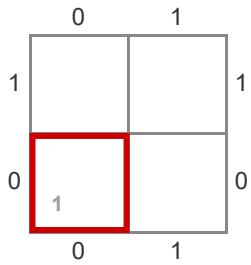
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

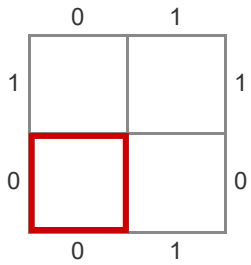
Resultados de las pruebas:

Cuando hay una bolita roja, la saca

Tablero inicial

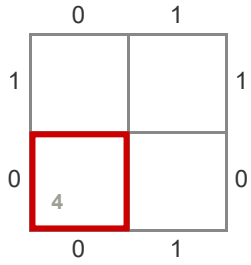


Tablero final

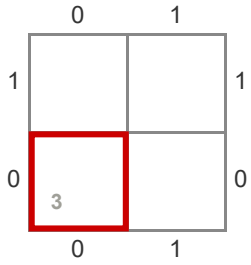


Cuando hay más de una bolita roja, saca una

Tablero inicial

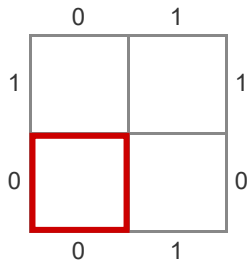


Tablero final

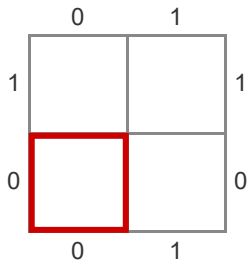


Cuando no hay ninguna bolita roja, no hace nada

Tablero inicial



Tablero final



Un ejemplo medio rebuscado

Vamos a ponerle nombre a las partes del `if`.

En primer lugar, tenemos la **condición**. Por ahora siempre fue `hayBolitas(color)` pero podría ser cualquier otra cosa, ya veremos más ejemplos. Lo importante acá es que eso es lo que **decide** si la **acción** se va a ejecutar o no.

¿Y qué es la **acción**? Básicamente, cualquier cosa que queramos hacer sobre el tablero. Al igual que en el `repeat`, podemos hacer cuantas cosas se nos ocurran, no necesariamente tiene que ser una sola.

Resumiendo: La **acción** que está dentro de la estructura del `if` podrá realizarse solo cuando la **condición** sea *verdadera*.

> Para ejercitar esto ultimo, te vamos a pedir que definas un procedimiento `CompletarCelda()` que, *si* ya hay alguna bolita negra, complete la celda poniendo una roja, una azul y una verde.

¡Dame una pista!

```
procedure CompletarCelda() {
  if (hayBolitas(Negro)) {
    Poner(Rojo)
    Poner(Verde)
    Poner(Azul)
  }
}

```

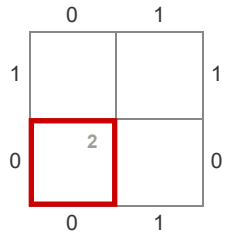
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

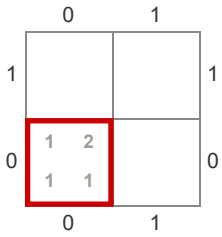
Resultados de las pruebas:

- Cuando hay alguna bolita negra, completa la celda

Tablero inicial

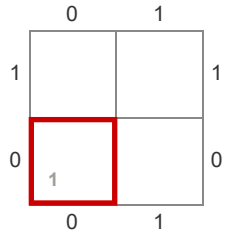


Tablero final

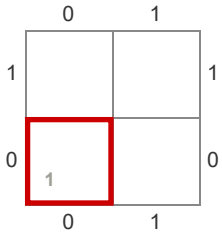


Cuando no hay ninguna bolita negra, no hace nada

Tablero inicial



Tablero final



¿Y sólo sirve para ver si hay bolitas?

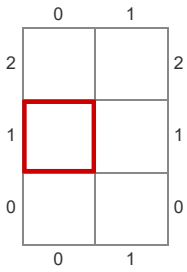
Claro que no, ¡por suerte!

La **condición** puede ser cualquier expresión *booleana*. En criollo: cualquier cosa que represente una "pregunta" que se pueda responder con **sí** o **no**. En Gobstones el *sí* se representa con el valor **True** (*Verdadero* en castellano) y el *no* con el valor **False** (*Falso* en castellano).

En los ejercicios anteriores te mostramos una de las expresiones que trae Gobstones, `hayBolitas(color)`, que recibe un `color` y retorna *True* o *False*.

Otra que trae *True* o *False* (y que vas a tener que usar ahora) es `puedeMover(direccion)` que nos sirve para saber si el cabezal puede moverse en una cierta dirección.

Por ejemplo, si tenemos este tablero:



- `puedeMover(Norte)` será `True`.
- `puedeMover(Sur)` será `True`.
- `puedeMover(Este)` será `True`.
- Pero `puedeMover(Oeste)` será `False`

> Creá un programa que se mueva al `Este` sólo *si es* posible. Recordá utilizar `puedeMover(direccion)`.

¡Dame una pista!

```
program {
 if (puedeMover(Este)) {
    Mover(Este)
  }
}
```

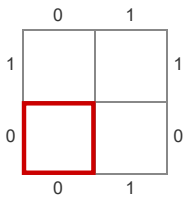
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

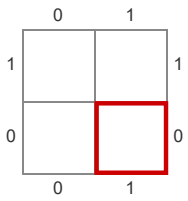
Resultados de las pruebas:

Si hay celdas al Este, se mueve

Tablero inicial

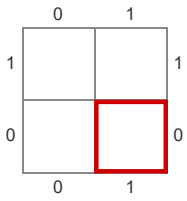


Tablero final

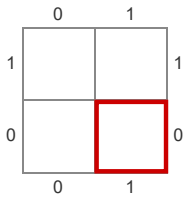


Si no hay celdas al Este, no hace nada

Tablero inicial



Tablero final



¿Y si hubieramos querido movernos hacia el Norte en caso de que **no** hubiera celdas al Este?

Un poquito de matemática

Otra cosa que se puede hacer adentro de un **if** es comparar números, como seguramente alguna vez hiciste en matemática.

Por suerte, esto se escribe en Gobstones igual que en la matemática tradicional, con un `<` para el menor y un `>` para el mayor. Ejemplo: `nroBolitas(Verde) > 5` nos indica si hay más de 5 bolitas verdes.

> Sabiendo esto, intentá crear un programa que ponga 1 bolita **negra** *sólo si* hay menos de 5 bolitas **negras**.

```
program {
 if (nroBolitas(Negro)<5) {
    Poner(Negro)
  }
}
```

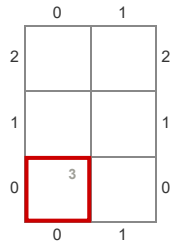
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

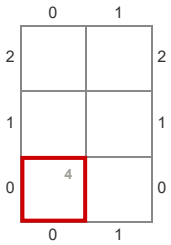
Resultados de las pruebas:

Si hay menos de 5 bolitas negras, agrega una

Tablero inicial

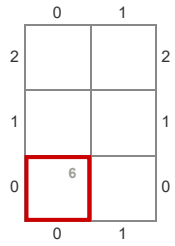


Tablero final

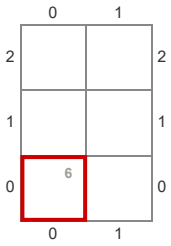


Si hay más de 5 bolitas negras, no hace nada

Tablero inicial



Tablero final



Cómo decirle que no...

En todos los problemas que hicimos hasta ahora, siempre preguntamos si una cierta condición se cumplía: ¿hay alguna bolita roja? ¿me puedo mover al Este? ¿hay más de 3 bolitas azules?

Algo que también se puede hacer es **negar** una condición, algo que en castellano puede sonar medio raro pero que en programación se hace un montón. Los ejemplos anteriores quedarían: ¿**no** hay alguna bolita roja? ¿**no** me puedo mover al Este? ¿**no** hay más de 3 bolitas azules?

¿Y cómo se hace en Gobstones? Fácil, se agrega la palabra clave `not` antes de la expresión que ya teníamos.

Original		Negada
`hayBolitas(Rojo)`	→	`not hayBolitas(Rojo)`
`puedeMover(Este)`	→	`not puedeMover(Este)`
`nroBolitas(Azul) > 3`	→	`not nroBolitas(Azul) > 3`

> Definí un procedimiento `AsegurarUnaBolitaVerde()` que se asegure que en la celda actual hay al menos una bolita verde. Esto es: si ya hay bolitas verdes no hay que hacer nada, pero si **no** hay tendría que poner una.

```
procedure AsegurarUnaBolitaVerde() {
  if (not hayBolitas(Verde)) {
    Poner(Verde)
  }
}
```

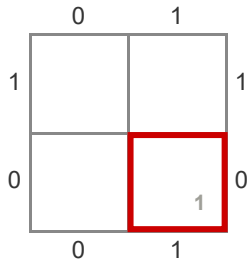
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

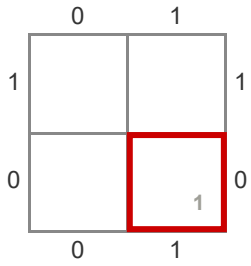
Resultados de las pruebas:

Si hay bolitas verdes, no hace nada

Tablero inicial

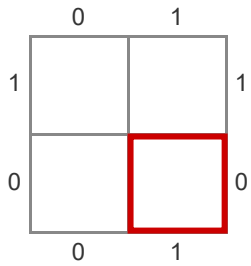


Tablero final

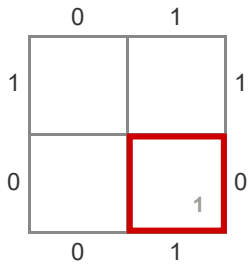


Si no hay bolitas verdes, agrega una

Tablero inicial



Tablero final



A lo que acabás de hacer, en lógica se lo llama **negación** y al anteponer el `not` decimos que se está **negando** una expresión. Cualquier expresión booleana (o sea, que devuelve *True* o *False*) se puede negar.

Dos caminos distintos

En lo cotidiano, se presentan muchas situaciones donde debemos elegir entre dos acciones diferentes, dependiendo de si se cumple una cierta condición o no.

- Si la remera está limpia me la pongo, *si no* la lavo.
- Si tengo aceite para freir las milanesas lo uso, *si no* le pongo un poco de manteca.
- Si me puedo mover al Este lo hago, *si no* me muevo al Norte.

Para estos casos, en Gobstones tenemos una nueva palabra clave que nos ayuda a cumplir nuestra tarea: el **else**. En castellano significa *si no* y hace justamente lo que necesitamos: ejecuta una serie de acciones *si no se cumple* la condición que pusimos en el `if`.

Supongamos que queremos definir un procedimiento que se mueva al Oeste y, en caso de que no pueda, lo haga hacia el Norte. Haciendo uso del `else`, podemos definirlo de la siguiente manera:

```
procedure MoverComoSea() {
    if (puedeMover(Oeste)) {
        Mover(Oeste)
    } else {
        Mover(Norte)
    }
}
```

> Escribí ese código en el editor y fijate cómo resuelve el problema.

```
1 procedure MoverComoSea() {
2     if (puedeMover(Oeste)) {
3         Mover(Oeste)
4     } else {
5         Mover(Norte)
6     }
7 }
```

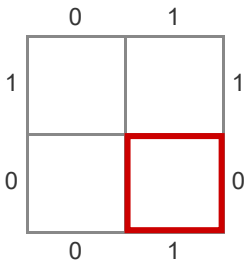
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

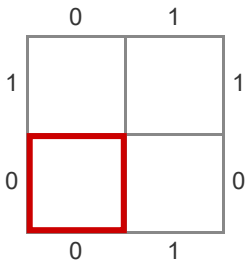
Resultados de las pruebas:

Si hay celdas al Oeste, se mueve

Tablero inicial

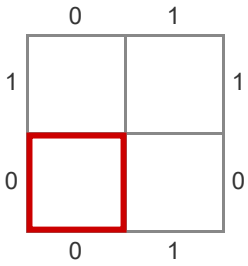


Tablero final

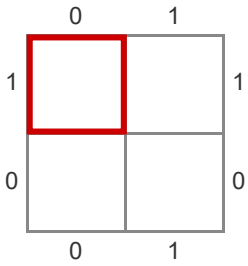


Si no hay celdas al Oeste, se mueve al Norte

Tablero inicial



Tablero final



¡Espectacular!

Ya conocés la herramienta que usan todas las aplicaciones que conociste en tu vida para decidir qué hacer, el viejo y querido **if / if...else**.

Un tablero de luces

Como ejemplo final, imaginemos que nuestro tablero está lleno de luces que están prendidas o apagadas. Vamos a decir que las celdas con una bolita verde están prendidas y las celdas con una bolita negra están apagadas.

> Definí un procedimiento `PrenderOApagarLuz()` que se encargue de prender las luces que estén apagadas o apagar las luces encendidas, según corresponda. Tené en cuenta que en cada celda solo puede haber bolitas de color verde o negro.

¡Dame una pista!

```
procedure PrenderOApagarLuz() {
  if (hayBolitas(Verde)) {
    Poner(Negro)
    Sacar(Verde)
  } else {
    Sacar(Negro)
    Poner(Verde)
  }
}
```

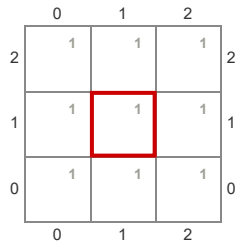
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

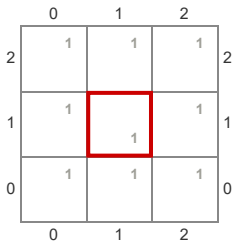
Resultados de las pruebas:

Si la celda está apagada, la prende

Tablero inicial

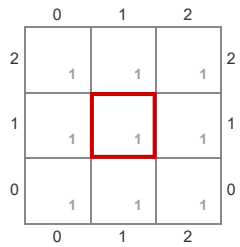


Tablero final



Si la celda está prendida, la apaga

Tablero inicial



Tablero final